PACE project

ANR 12IS02001

Deliverable D2013-123

December 2013

Introduction. We give below a presentation of the results obtained during the first year of PACE (january-december 2013). This corresponds to the actual deliverable. Before that, we present some facts related to the life of the project.

Nota: The project started officially on Jan. 1st, 2013. As a consequence, some of the works published in 2013 do not acknowledge explicitly the PACE project. We mention nevertheless those that correspond to contributions within the scope of PACE, marking them with ★.

A description of the goals and organisation of the ANR PACE project (*"beyond plain Processes: Analysis techniques, Coinduction and Expressiveness"*) can be found here. The website of the PACE project can be found here.

Life of the project

0.1 Personnel

Funded by PACE.

- **Thomas Given-Wilson** is a new postdoc hired to work on PACE. He did his PhD with Barry Jay and Daniele Gorla on the expressiveness of a concurrent functional programming language. In PACE he will study the expressiveness of process calculi with respect to their capabilities to provide support for controlling information flow and protecting privacy.
- **Mingzhang Huang** has been recently enrolled as a PhD candidate under the support of this project.

Contributing to PACE.

- **Salim Perchy** is a new PhD student hired in Nov, 2013 and funded by (7161- Labex DIGICOSME, 2013). He is co-supervised by Stefan Haar and Frank Valencia. He will be working on a calculus for social networks; one of the subjects of PACE.
- **Alberto Cappai**, PhD Student Starting date: 1/1/2013 Topics: applicative bisimulation and computational indistinguishability Reference: Davide Sangiorgi and Ugo Dal Lago
- D. Hirschkoff and D. Pous are members of the ANR Project "PiCoq". This project funds the post-doctoral contract of **Daniela Petrisan**, who joined the Lyon group on November the first, 2013. We expect the expertise of D. Petrisan on coalgebraic models of behavioural equivalences and process calculi to be fruitful for the PACE project, throuhg interactions with F. Bonchi and D. Pous.

0.2 Visits, meetings

PACE kick-off meeting. A meeting of the researchers involved in PACE took place on april 22 and 23, 2013, in Bologna. See the web page presenting the scientific programme of the meeting. In additional to scientific discussions, some time has been devoted to the organisational matters related to the organisation of the project.

Yuxi Fu was granted a research fund (200,000 RMB) to support the joint collaboration between partners of this project.

Visits between PACE partners

- Daniel Hirschkoff and Jean-Marie Madiot (ENS Lyon) visited BASICS (Shanghai) in november 2013 to work together with Fu Yuxi and Xu Xian on the behavioural equivalence (characterisation, proof techniques, axiomatisation) of a π-calculus with preorders.
- Catuscia Palamidessi (INRIA Saclay) visited BASICS (Shanghai) in november 2013 to discuss questions related to the PACE project (and, in particular, differential privacy).

Other visits and exchanges

- Frank Valencia (INRIA Saclay) visited PUJ, Colombia in July 2013 for two weeks. The purpose of the visit was to present F. Valencia's work and future direction on calculi for social networks. This visit was financed by PACE.
- Frank Valencia (INRIA Saclay) will visit PUJ, Colombia in January 2014 for ten days. The purpose of this visit is to continue the collaboration on the development of calculi for social networks. This visit will be financed by PACE.
- Ugo Dal Lago (INRIA Sophia) visited Andy Pitts (in Cambridge) and Andy Gordon (at MSR Cambridge) for a week (July 15th to July 23rd, 2013). They discussed about probabilistic applicative bisimimulation, biorthogonality in a probabilistic setting, probabilistic programming and machine learning.
- Filippo Bonchi (ENS Lyon) visited Braga (Universidade do Minho, Portugal) in September 2013, to discuss coalgebraic presentations of behavioural equivalences.
- Jorge PEREZ from New University of Lisbon visited COMETE (INRIA Saclay) in Nov, 2013 for three days. The purpose of the visit was to work on the development of a calculus for social networks. His visit was partially funded by PACE.
- Jurriaan Rot from Leiden University visited F. Bonchi and D. Pous (Lyon), between dec. the 9th and dec. the 13th, 2013, to work on coalgebraic presentations of bisimulations and up-to techniques.

Invited lectures Damien Pous presented some results on up-to techniques (tasks T1.1 and T3.1) at the PSATTT workshop at LIX, École Polytechnique (november 2013) and at the Calco 13 conference (september 2013).

1 Task 1: Advanced Coinductive Techniques

The following papers present contributions to the topics of Task 1:

- Ugo Dal Lago and Paolo Parisen Toldin. An higher-order characterization of probabilistic polynomial time. Information and Computation, 2013. to appear.

  (T1.3)

- Ugo Dal Lago, Davide Sangiorgi, and Michele Alberti. On Coinductive Equivalences for Higher-Order Probabilistic Functional Programs. In POPL. ACM Press, 2014.

  (T1.3)

- Ugo Dal Lago and Margherita Zorzi. Wave-style token machines and quantum lambda calculi. 2013.

  (T1.4)

- D. Hirschkoff, J.-M. Madiot and D. Sangiorgi, Name-Passing Calculi: From Fusions to Preorders and Types. LICS 2013: 378-387

  (T1 & T2.1)

- ⋆ Chatzikokolakis, Konstantinos; Palamidessi, Catuscia; Braun, Christelle, "Compositional Methods for Information-Hiding", Mathematical Structures in Computer Science, Cambridge University Press, To appear

  (T1.3)

- ⋆ C. Olarte, C. Rueda, F. Valencia. Models and emerging trends of concurrent constraint programming. Constraints Vol 18 (4), Pages 535-578.

  (T1)

- Damien Pous. Coalgebraic Up-to Techniques. Invited talk at CALCO 2013. LNCS 8089

  (T1.1)

- ⋆ Filippo Bonchi, Fabio Zanasi, "Saturated Semantics for Coalgebraic Logic Programming" In Proc. of CALCO, LNCS 8089, Springer 2013.

  (T1)

- Xian Xu. On Context Bisimulation for Parameterized Higher-order Processes. Proceedings of the 6th Interaction and Concurrency Experience (ICE 2013), 2013. Electronic Proceedings in Theoretical Computer Science 131: 37-51, 2013.

- Xian Xu, Qiang Yin and Huan Long. On the Expressiveness of Parameterization in Process-passing. (post-proceedings of WS-FM2013 in LNCS are in preparation)

We now discuss the announced deliverables for Task 1 (Year 1) in the submission document for PACE:

- D1.1.1 A metatheory of bisimulation enhancements for pure Higher-Order pi-calculus with only process passing

  The works by Pous (CALCO'13) and Xu (ICE'13, WS-FM2013) represent contributions to this subject. Moreover, current work on this specific delivrable is under way, and we expect it to be published during Year 2 or Year 3.

- D1.1.2 Accounts of metric bisimulations using coalgebras

  This topic has turned out to be more difficult than expected, and we have no contribution on this subject yet. We shall continue studying metric bisimulations (mostly in Lyon).

- D1.1.3 A probabilistic bisimulation for untyped lambda-calculus along the lines of applicative bisimulation

  By contrast to the previous one, this topic has been quite fruitful. The paper by Dal Lago et al. at POPL'14 is clearly a contribution on the subject. More broadly, we plan to deepen the study of probabilistic computation, beyond the investigation of behavioural equivalence (this is the case, in particular, for the two other papers coauthored by Dal Lago mentioned above).

- D1.1.4 Symbolic characterisations of strong bisimulation on a quantum CCS

  The paper *Symbolic bisimulation for quantum processes*, by Yuan Feng, Yuxin Deng, and Mingsheng Ying (to appear in ACM Transactions on Computational Logic), which is mentioned in Task 3 below, is also a contribution to this deliverable. Work on quantum computation is also under way in the Bologna site.

2 Task 2: Expressiveness

The following papers present contributions to the topics of Task 2:

- ⋆ Gazeau, Ivan; Miller, Dale; Palamidessi, Catuscia, "Preserving differential privacy under finite-precision semantics", Proceedings of the 11th International Workshop on Quantitative Aspects of Programming Languages and Systems. Luca Bortolussi and Herbert Wiklicky (eds.) LIPIcs, Open Publishing Association, 1-18 (2013)

  (T2.3 & T3)

- ⋆ Elsalamouny, Ehab; Chatzikokolakis, Konstantinos; Palamidessi, Catuscia, "A differentially private mechanism of optimal utility for a region of priors", Proc. of the ETAPS Conference on Principles of Security and Trust. David Basin and John Mitchel (eds.), LNCS 7796. Springer, 41-62 (2013)

  (T2.3)

- Chatzikokolakis, Konstantinos; Andrés, Miguel, E.; Bordenabe, Nicolás, E.; Palamidessi, Catuscia, "Broadening the Scope of Differential Privacy Using Metrics", Proc. of the Privacy Enhancing Technology Symposium. Emiliano De Cristofaro, Matthew Wright (eds.), Lecture Notes in Computer Science 7981, Springer, 82-102 (2013)

  (T2.3)

- Andrés, Miguel, E.; Bordenabe, Nicolás, E.; Chatzikokolakis, Konstantinos; Palamidessi, Catuscia, "Geo-Indistinguishability: Differential Privacy for Location-Based Systems", Proc. of the ACM Conference on Computer Communication and Security. ACM Press 901-914 (2013)

  (T2.3)

- Yuxi Fu. Checking Equality and Regularity for Normed BPA with Silent Moves. F.V. Fomin et al. (Eds.): ICALP 2013, Part II, LNCS 7966, 244-255, 2013.

- Yuxi Fu. The Value-Passing Calculus. Theories of Programming and Formal Methods, LNCS 8051, 166-195, 2013.

- Yuxi Fu. Nondeterministic Structure of Computation. Mathematical Structures in Computer Science, accepted for publication.

- Xiaojuan Cai, Mizuhito Ogawa. Well-Structured Pushdown Systems. In Proceedings of the 24th International Conference on Concurrency Theory (CONCUR'13), LNCS 8052, 121-136, 2013.

- Guoqiang Li, Xiaojuan Cai, Mizuhito Ogawa, Shoji Yuen. Nested Timed Automata. In Proceedings of the 11th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'13), LNCS 8053, 168-182, 2013.

- Mizuhito Ogawa and Xiaojuan Cai. On Reachability of Dense Timed Pushdown Automata. JAIST technical report No. IS-RR-2013-005. 2013

Recent publications worth mentioning, outside of PACE The following papers, published in 2013, present work that has been done before the beginning of the PACE project. We mention them here because they correspond to PACE-relevant topics that we want to investigate further in the future.

- *Maurizio Gabbrielli, Jacopo Mauro, and Maria Chiara Meo. The expressive power of CHR with priorities. Inf. Comput., 228:62–82, 2013.*

  (T2)

- *Ivan Lanese and Gianluigi Zavattaro. Decidability results for dynamic installation of compensation handlers. In Proc. of COORDINATION, volume 7890 of Lecture Notes in Computer Science, pages 136–150. Springer, 2013.*
  (T2)
- *Roberto Di Cosmo, Jacopo Mauro, Stefano Zacchiroli, Gianluigi Zavattaro: Component Reconfiguration in the Presence of Conflicts. ICALP (2) 2013: 187-198*
  (T2)

We now discuss the announced deliverables for Task 2 (Year 1) in the submission document for PACE:

- D2.1.1 Extend Fu's expressiveness framework to probabilistic processes and higher-order processes. There should be several coincidence results relating the model independent equivalence to the concrete equivalence.

  We have not addressed probabilistic and higher-order computation within Fu's framework yet. The two papers by Fu (TPFM'13 and MSCS) present however advances in the study of this framework.

- D2.1.2 An implementation of Erlang into a more basic first-order process model.

  This deliverable has not been treated yet. We hope to work on this translation in the next years.

- D2.1.3 A model-independent equivalence based on subbisimilarity for reasoning about inter-language equivalences

  This deliverable has not been treated yet. We hope to work on this translation in the next years.

- D2.1.4 Proof that hiding-free fragments of the spatial-epistemic language are not Turing complete.

  The work on spatial-epistemic logics has progressed during the first year of PACE. We expect the aforementioned results to be published in the future.

As appears above, some of the first year deliverables for Task 2 have not been treated yet. Task 2 is about expressiveness. We would like to mention that expressiveness has been treated from a new perspective in several works by PACE members in Year 1, by focusing on ***branching bisimilarity***. Adopting the point of view of branching bisimilarity (BB) makes it possible to revisit existing literature on weak bisimilarity, yielding sometimes unexpected results, due to the elegance of BB. In some cases, BB may be preferable to weak bisimilarity to check properties of systems. The results in Fu's paper at ICALP'13 go in this direction, and further studies, by members of the Shanghai group, are underway. We expect to obtain in the next years expressiveness results about BB, as well as decidability results (and, possibly, decision methods in some cases): the former belong to Task 2, while the latter belong to Task 3.

We can also remark that the group in Lyon has also been working on branching bisimilarity, and on enhancements of it (Task 1), during the first year of PACE.

Although we did not predict the importance of branching bisimilarity in PACE when submitting the project, we feel that this is a promising research subject, and that studying it fits well within the scientific objectives of PACE.

3 Task 3: Analysis techniques

The following papers present contributions to the topics of Task 3:

- Filippo Bonchi, Georgiana Caltais, Damien Pous, Alexandra Silva. Brzozowski's and Up-To Algorithms for Must Testing. Invited talk. In Proc. APLAS 2013. LNCS 8089.

  (T3.1)

- ⋆ Damien Pous. Kleene Algebra with Tests and Coq Tools for while Programs. In Proc. ITP 2013, LNCS 7998.

  (T3.1)

- Alexandra Silva, Filippo Bonchi, Marcello Bonsangue, Jan Rutten, "Generalizing determinization from automata to coalgebras" In Logical Methods in Computer Science 9(1) (2013)

  (T3.1)

- L. Pino, F. Bonchi, F. Valencia. Efficient Computation of Program Equivalence for Confluent Concurrent Constraint Programming. In Proceedings of 15th ACM International Symposium on Symposium on Principles and Practice of Declarative Programming - PPDP 2013. Pages 263-274.

  (T3)

- Deng, Rob van Glabbeek, Matthew Hennessy, and Carroll Morgan. Real-Reward Testing for Probabilistic Processes. Theoretical Computer Science. Elsevier, 2013. To appear.

- Yuan Feng, Yuxin Deng, and Mingsheng Ying. Symbolic bisimulation for quantum processes. ACM Transactions on Computational Logic. 2013. To appear.

Recent publications worth mentioning, outside of PACE The following paper:

> *F. Bonchi and D. Pous. "Checking NFA equivalence with bisimulations up to congruence." in Proc. POPL'13, ACM.*

presents results that have been obtained before the beginning of the PACE project, but that clearly belong to the goals of PACE (T3.1). We expect that this work will lead to several offspins in the course of the PACE project.
We now discuss the announced deliverables for Task 3 (Year 1) in the submission document for PACE:

- D3.1.1 Finite automata algorithms based on up-to techniques

  The main contribution regarding this deliverable is the algorithm by Bonchi and Pous (to be published in CACM Research Highlights). Further generalisations of usages of bisimulation enhancements in reasoning about automata, have also been proposed (by the same authors and colleagues).

- D3.1.2 Basic algorithms for probabilistic bisimulation for untyped lambda-calculus

  Forms of probabilistic computation are studied by PACE members. We do not have algorithms for probabilistic bisimulation, but the papers mentioned above, as well as some papers presented in Task 1, testify of the progress made in this direction. We hope to come up with algorithms for probabilistic bisimulation for untyped λ-calculus, and for other models that we study in PACE, in the next years.

Tasks and subtasks in the PACE project

Task 1: Advanced Coinductive Techniques

Task leader: Davide Sangiorgi / Deputy task leader: Xu Xian

- T1.1: Up-to techniques
- T1.2: From equivalences to metrics
- T1.3: Probabilistic and quantum higher-order languages
- T1.4: Quantum processes

Task 2: Expressiveness

Task leader: Fu Yuxi / Deputy task leader: Catuscia Palamidessi

- T2.1: Absolute theory
- T2.2: Expressiveness in social networks
- T2.3: Applications to privacy, confidentiality and anonymity

Task 3: Analysis techniques

Task leader: Damien Pous / Deputy task leader: Deng Yuxin

- T3.1: Algorithms relying on up-to techniques
- T3.2: Up-to techniques in algorithms for metrics
- T3.3: Algorithms for quantum bisimulations
- T3.4: Minimization algorithms for symbolic bisimulation

On Coinductive Equivalences for Higher-Order Probabilistic Functional Programs

Ugo Dal Lago Davide Sangiorgi Michele Alberti

Abstract

We study bisimulation and context equivalence in a probabilistic λ-calculus. The contributions of this paper are threefold. Firstly we show a technique for proving congruence of *probabilistic applicative bisimilarity*. While the technique follows Howe's method, some of the technicalities are quite different, relying on non-trivial "disentangling" properties for sets of real numbers. Secondly we show that, while bisimilarity is in general strictly finer than context equivalence, coincidence between the two relations is attained on pure λ-terms. The resulting equality is that induced by *Levy-Longo trees*, generally accepted as the finest extensional equivalence on pure λ-terms under a lazy regime. Finally, we derive a coinductive characterisation of context equivalence on the whole probabilistic language, via an extension in which terms akin to distributions may appear in redex position. Another motivation for the extension is that its operational semantics allows us to experiment with a different congruence technique, namely that of *logical bisimilarity*.

1 Introduction

Probabilistic models are more and more pervasive. Not only are they a formidable tool when dealing with uncertainty and incomplete information, but they sometimes are a *necessity* rather than an option, like in computational cryptography (where, e.g., secure public key encryption schemes need to be probabilistic [17]). A nice way to deal computationally with probabilistic models is to allow probabilistic choice as a primitive when designing algorithms, this way switching from usual, deterministic computation to a new paradigm, called probabilistic computation. Examples of application areas in which probabilistic computation has proved to be useful include natural language processing [31], robotics [48], computer vision [8], and machine learning [36].

This new form of computation, of course, needs to be available to programmers to be accessible. And indeed, various probabilistic programming languages have been introduced in the last years, spanning from abstract ones [24, 40, 35] to more concrete ones [37, 18], being inspired by various programming paradigms like imperative, functional or even object oriented. A quite common scheme consists in endowing any deterministic language with one or more primitives for probabilistic choice, like binary probabilistic choice or primitives for distributions.

One class of languages that copes well with probabilistic computation are functional languages. Indeed, viewing algorithms as functions allows a smooth integration of distributions into the playground, itself nicely reflected at the level of types through monads [20, 40]. As a matter of fact, many existing probabilistic programming languages [37, 18] are designed around the λ-calculus or one of its incarnations, like Scheme. All these allows to write higher-order functions (i.e., programs can take functions as inputs and produce them as outputs).

The focus of this paper are operational techniques for understanding and reasoning about program equality in higher-order probabilistic languages. Checking computer programs for equivalence is a crucial, but challenging, problem. Equivalence between two programs generally means that the programs should behave "in the same manner" under any context [32]. Specifically, two λ-terms are *context equivalent* if they have the same convergence behavior (i.e., they do or do not terminate) in any possible context. Finding effective methods for context equivalence proofs is particularly challenging in higher-order languages.

Bisimulation has emerged as a very powerful operational method for proving equivalence of programs in various kinds of languages, due to the associated coinductive proof method. To be useful, the behavioral relation resulting from bisimulation — *bisimilarity* — should be a *congruence*, and should also be sound with respect to context equivalence. Bisimulation has been transplanted onto higher-order languages by Abramsky [1]. This version of bisimulation, called *applicative bisimulation* has received considerable attention [19, 38, 42, 27, 39, 28]. In short, two functions M and N are applicative bisimilar when their applications MP and NP are applicative bisimilar for any argument P.

Often, checking a given notion of bisimulation to be a congruence in higher-order languages is nontrivial. In the case of applicative bisimilarity, congruence proofs usually rely on Howe's method [22]. Other forms of bisimulation have been proposed, such as environmental bisimulation and logical bisimulation [44, 45, 25], with the goal of relieving the burden of the proof of congruence, and of accommodating language extensions.

In this work, we consider the pure λ-calculus extended with a probabilistic choice operator. Context equivalence of two terms means that they have the same *probability of convergence* in all contexts. The objective of the paper is to understand context equivalence and bisimulation in this paradigmatic probabilistic higher-order language, called $\Lambda_{\oplus}$.

The paper contains three main technical contributions. The first is a proof of congruence for probabilistic applicative bisimilarity along the lines of Howe's method. This technique consists in defining, for every relation on terms $\mathcal{R}$, its Howe's lifting $\mathcal{R}^H$. The construction, essentially by definition, ensures that the relation obtained by lifting bisimilarity is a congruence; the latter is then proved to be itself a bisimulation, therefore coinciding with applicative bisimilarity. Definitionally, probabilistic applicative bisimulation is obtained by setting up a labelled Markov chain on top of λ-terms, then adapting to it the coinductive scheme introduced by Larsen and Skou in a first-order setting [26]. In the proof of congruence, the construction $(\cdot)^H$ closely reflects analogous constructions for nondeterministic extensions of the λ-calculus. The novelties are in the technical details for proving that the resulting relation is a bisimulation: in particular our proof of the so-called Key Lemma — an essential ingredient in Howe's method — relies on non-trivial "disentangling" properties for sets of real numbers, these properties themselves proved by modeling the problem as a flow network and then apply the Max-flow Min-cut Theorem. The congruence of applicative bisimilarity yields soundness with respect to context equivalence as an easy corollary. Completeness, however, fails: applicative bisimilarity is proved to be finer. A subtle aspect is also the late vs. early formulation of bisimilarity; with a choice operator the two versions are semantically different; our construction crucially relies on the late style.

In our second main technical contribution we show that the presence of higher-order functions and probabilistic choice in contexts gives context equivalence and applicative bisimilarity maximal discriminating power on pure λ-terms. We do so by proving that, on pure λ-terms, both context equivalence and applicative bisimilarity coincide with the *Levy-Longo tree equality*, which equates terms with the same Levy-Longo tree (briefly LLT). The LLT equality is generally accepted as the finest extensional equivalence on pure λ-terms under a *lazy* regime. The result is in sharp contrast with what happens under a nondeterministic interpretation of choice (or in the absence of choice), where context equivalence is coarser than LLT equality.

Our third main contribution is a coinductive characterisation of probabilistic context equivalence on the whole language $\Lambda_{\oplus}$ (as opposed to the subset of pure λ-terms). We obtain this result by setting a bisimulation game on an extension of $\Lambda_{\oplus}$ in which weighted formal sums — terms akin to distributions — may appear in redex position. Thinking of distributions as sets of terms, the construction reminds us of the reduction of nondeterministic to deterministic automata. The technical details are however quite different, because we are in a higher-order language and therefore — once more — we are faced with the congruence problem for bisimulation, and because formal sums may contain an *infinite* number of terms. For the proof of congruence of bisimulation in this extended language, we have experimented the technique of logical bisimulation. In this method (and in the related method of environmental bisimulation), the clauses of applicative bisimulation are modified so to allow the standard congruence argument for bisimulations in first-order languages, where the bisimulation method itself is exploited to establish that the closure of

the bisimilarity under contexts is again a bisimulation. Logical bisimilarities have two key elements. First, bisimilar functions may be tested with *bisimilar* (rather than identical) arguments (more precisely, the arguments should be in the context closure of the bisimulation; the use of contexts is necessary for soundness). Secondly, the transition system should be small-step, deterministic (or at least confluent), and the bisimulation game should also be played on internal moves. In our probabilistic setting, the ordinary logical bisimulation game has to be modified substantially. Formal sums represent possible evolutions of running terms, hence they should appear in redex position only (allowing them anywhere would complicate matters considerably), also making the resulting bisimulation proof technique more cumbersome). The obligation of redex position for certain terms is in contrast with the basic schema of logical bisimulation, in which related terms can be used as arguments to bisimilar functions and can therefore end up in arbitrary positions. We solve this problem by moving to *coupled logical bisimulations*, where a bisimulation is formed by a pair of relations, one on $\Lambda_{\oplus}$-terms, the other on terms extended with formal sums. The bisimulation game is played on both relations, but only the first relation is used to assemble input arguments for functions.

Another delicate point is the meaning of internal transitions for formal sums. In logical bisimilarity the transition system should be small-step; and formal sums should evolve into values in a finite number of steps, even if the number of terms composing the formal sum is infinite. We satisfy these requirements by defining the transition system for extended terms on top of that of $\Lambda_{\oplus}$-terms. The proof of congruence of coupled logical bisimilarity also exploits an "up-to distribution" bisimulation proof technique.

In the paper we adopt call-by-name evaluation. The results on applicative bisimilarity can be transported onto call-by-value; in contrast, transporting the other results is less clear, and we leave it for future work. See Section 8 for more details. An extended version of this paper with more details is available [9].

1.1 Further Related Work

Research on (higher-order) probabilistic functional languages have, so far, mainly focused on either new programming constructs, or denotational semantics, or applications. The underlying operational theory, which in the ordinary λ-calculus is known to be very rich, has remained so far largely unexplored. In this section, we give some pointers to the relevant literature on probabilistic λ-calculi, without any hope of being exhaustive.

Various probabilistic λ-calculi have been proposed, starting from the pioneering work by Saheb-Djahromi [41], followed by more advanced studies by Jones and Plotkin [24]. Both these works are mainly focused on the problem of giving a denotational semantics to higher-order probabilistic computation, rather than on studying it from an operational point view. More recently, there has been a revamp on this line of work, with the introduction of adequate (and sometimes also fully-abstract) denotational models for probabilistic variations of PCF [11, 16]. There is also another thread of research in which various languages derived from the λ-calculus are given types in monadic style, allowing this way to nicely model concrete problems like Bayesian inference and probability models arising in robotics [40, 35, 20]; these works however, do not attack the problem of giving an operationally based theory of program equivalence.

Nondeterministic extensions of the λ-calculus have been analysed in typed calculi [3, 47, 27] as well as in untyped calculi [23, 7, 33, 13]. The emphasis in all these works is mainly domain-theoretic. Apart from [33], all cited authors closely follow the testing theory [12], in its modalities *may* or *must*, separately or together. Ong's approach [33] inherits both testing and bisimulation elements.

Our definition of applicative bisimulation follows Larsen and Skou's scheme [26] for fully-probabilistic systems. Many other forms of probabilistic bisimulation have been introduced in the literature, but their greater complexity is usually due to the presence of *both* nondeterministic and probabilistic behaviors, or to continuous probability distributions. See surveys such as [5, 34, 21].

Contextual characterisations of LLT equality include [6], in a λ-calculus with multiplicities in which deadlock is observable, and [15], in a λ-calculus with choice, parallel composition, and both call-by-name and call-by-value applications. The characterisation in [43] in a λ-calculus with

$$\frac{}{M \Downarrow \emptyset}\ \mathsf{bt} \qquad \frac{}{V \Downarrow \{V\}}\ \mathsf{bv} \qquad \frac{M \Downarrow \mathscr{D} \quad \{P\{^N/_x\} \Downarrow \mathscr{E}_{P,N}\}_{\lambda x.P \in \mathsf{S}(\mathscr{D})}}{MN \Downarrow \sum_{\lambda x.P \in \mathsf{S}(\mathscr{D})} \mathscr{D}(\lambda x.P) \cdot \mathscr{E}_{P,N}}\ \mathsf{ba} \qquad \frac{M \Downarrow \mathscr{D} \quad N \Downarrow \mathscr{E}}{M \oplus N \Downarrow \frac{1}{2} \cdot \mathscr{D} + \frac{1}{2} \cdot \mathscr{E}}\ \mathsf{bs}$$

Figure 1: Big-step call-by-name approximation semantics for $\Lambda_\oplus$.

non-deterministic operators, in contrast, is not contextual, as derived from a bisimulation that includes a clause on internal moves so to observe branching structures in behaviours. See [14] for a survey on observational characterisations of λ-calculus trees.

2 Preliminaries

2.1 A Pure, Untyped, Probabilistic Lambda Calculus

Let $\mathsf{X} = \{x, y, \ldots\}$ be a denumerable set of variables. The set $\Lambda_\oplus$ of *term expressions*, or *terms* is defined as follows:

$$M, N, L \ ::= \ x \ \Big| \ \lambda x.M \ \Big| \ MN \ \Big| \ M \oplus N,$$

where $x \in \mathsf{X}$. The only non-standard operator in $\Lambda_\oplus$ is probabilistic choice: $M \oplus N$ is a term which is meant to behave as either M or N, each with probability $\frac{1}{2}$. A more general construct $M \oplus_p N$ where p is any (computable) real number from $[0, 1]$, is derivable, given the universality of the λ-calculus (see, e.g., [10]). The set of free variables of a term M is indicated as $\mathsf{FV}(M)$ and is defined as usual. Given a finite set of variables $\overline{x} \subseteq \mathsf{X}$, $\Lambda_\oplus(\overline{x})$ denotes the set of terms whose free variables are among the ones in $\overline{x}$. A term M is *closed* if $\mathsf{FV}(M) = \emptyset$ or, equivalently, if $M \in \Lambda_\oplus(\emptyset)$. The (capture-avoiding) substitution of N for the free occurrences of x in M is denoted $M\{^N/_x\}$. We sometimes use the identity term $I \stackrel{\text{def}}{=} \lambda x.x$, the projector $K \stackrel{\text{def}}{=} \lambda x.\lambda y.x$, and the purely divergent term $\Omega \stackrel{\text{def}}{=} (\lambda x.xx)(\lambda x.xx)$.

Terms are now given a call-by-name semantics following [10]. A term is a *value* if it is a closed λ-abstraction. We call $\mathsf{V}\Lambda_\oplus$ the set of all values. Values are ranged over by metavariables like V, W, X. Closed terms evaluates not to a single value, but to a *(partial) value distribution*, that is, a function $\mathscr{D} : \mathsf{V}\Lambda_\oplus \to \mathbb{R}_{[0,1]}$ such that $\sum_{V \in \mathsf{V}\Lambda_\oplus} \mathscr{D}(V) \leq 1$. The set of all value distributions is $\mathcal{P}_\mathsf{v}$. Distributions do not necessarily sum to 1, so to model the possibility of (probabilistic) divergence. Given a value distribution $\mathscr{D}$, its *support* $\mathsf{S}(\mathscr{D})$ is the subset of $\mathsf{V}\Lambda_\oplus$ whose elements are values to which $\mathscr{D}$ attributes positive probability. Value distributions ordered pointwise form both a lower semilattice and an ω**CPO**: limits of ω-chains always exist. Given a value distribution $\mathscr{D}$, its *sum* $\sum \mathscr{D}$ is $\sum_{V \in \mathsf{V}\Lambda_\oplus} \mathscr{D}(V)$.

The call-by-name semantics of a closed term M is a value distribution $[\![M]\!]$ defined in one of the ways explained in [10]. We recall this now, though only briefly for lack of space. The first step consists in defining a formal system deriving finite *lower approximations* to the semantics of M. Big-step approximation semantics, as an example, derives judgments in the form $M \Downarrow \mathscr{D}$, where M is a term and $\mathscr{D}$ is a value distribution of finite support (see Figure 1). Small-step approximation semantics can be defined similarly, and derives judgments in the form $M \Rightarrow \mathscr{D}$. Noticeably, big-step and small-step can simulate each other, i.e. if $M \Downarrow \mathscr{D}$, then $M \Rightarrow \mathscr{E}$ where $\mathscr{E} \geq \mathscr{D}$, and *vice versa* [10]. In the second step, $[\![M]\!]$, called the *semantics* of M, is set as the least upper bound of distributions obtained in either of the two ways:

$$[\![M]\!] \stackrel{\text{def}}{=} \sup_{M \Downarrow \mathscr{D}} \mathscr{D} = \sup_{M \Rightarrow \mathscr{D}} \mathscr{D}.$$

Notice that the above is well-defined because for every M, the set of all distributions $\mathscr{D}$ such that $M \Downarrow \mathscr{D}$ is directed, and thus its least upper bound is a value distribution because of ω-completeness.

```
expone f n = (f n) (+) (expone f n+1)
exptwo f = (\x -> f x) (+) (exptwo (\x -> f (x+1)))
expthree k f n = foldp k n f (expthree (expone id k) f)

foldp 0 n f g = g n
foldp m n f g = (f n) (+) (foldp (m-1) (n+1) f g)
```

Figure 2: Three Higher-Order Functions

Example 2.1 *Consider the term* $M \stackrel{\text{def}}{=} I \oplus (K \oplus \Omega)$. *We have* $M \Downarrow \mathscr{D}$, *where* $\mathscr{D}(I) = \frac{1}{2}$ *and* $\mathscr{D}(V)$ *is* 0 *elsewhere, as well as* $M \Downarrow \emptyset$, *where* $\emptyset$ *is the empty distribution. The distribution* $[\![M]\!]$ *assigns* $\frac{1}{2}$ *to* I *and* $\frac{1}{4}$ *to* K.

The semantics of terms satisfies some useful equations, such as:

Lemma 2.2 $[\![(\lambda x.M)N]\!] = [\![M\{^N/_x\}]\!]$.

Lemma 2.3 $[\![M \oplus N]\!] = \frac{1}{2}[\![M]\!] + \frac{1}{2}[\![N]\!]$.

Proof. See [10] for detailed proofs. □

We are interested in context equivalence in this probabilistic setting. Typically, in a qualitative scenario as the (non)deterministic one, terms are considered context equivalent if they both converge or diverge. Here, we need to take into account quantitative information.

Definition 2.4 (Context Preorder and Equivalence) *The expression* $M\Downarrow_p$ *stands for* $\sum[\![M]\!] = p$, *i.e., the term* M *converges with probability* p. *The* context preorder $\leq_\oplus$ *stipulates* $M \leq_\oplus N$ *if* $C[M]\Downarrow_p$ *implies* $C[N]\Downarrow_q$ *with* $p \leq q$, *for every closing context* C. *The equivalence induced by* $\leq_\oplus$ *is* probabilistic context equivalence, *denoted as* $\simeq_\oplus$.

Remark 2.5 (Types, Open Terms) *The results in this paper are stated for an untyped language. Adapting them to a simply-typed language is straightforward; we use integers, booleans and recursion in examples. Moreover, while the results are often stated for closed terms only, they can be generalized to open terms in the expected manner. In the paper, context equivalences and preorders are defined on open terms; (bi)similarities are defined on closed terms and it is then intended that they are extended to open terms by requiring the usual closure under substitutions.*

Example 2.6 *We give some basic examples of higher-order probabilistic programs, which we will analyse using the coinductive techniques we introduce later in this paper. Consider the functions* `expone`, `exptwo`, *and* `expthree` *from Figure 2. They are written in a* Haskell*-like language extended with probabilistic choice, but can also be seen as terms in a (typed) probabilistic* λ*-calculus with integers and recursion akin to* $\Lambda_\oplus$. *Term* `expone` *takes a function* `f` *and a natural number* `n` *in input, then it proceeds by tossing a fair coin (captured here by the binary infix operator* `(+)`*) and, depending on the outcome of the toss, either calls* `f` *on* `n`, *or recursively calls itself on* `f` *and* `n+1`. *When fed with, e.g., the identity and the natural number* `1`, *the program* `expone` *evaluates to the geometric distribution assigning probability* $\frac{1}{2^n}$ *to any positive natural number* n. *A similar effect can be obtained by* `exptwo`, *which only takes* `f` *in input, then "modifying" it along the evaluation. The function* `expthree` *is more complicated, at least apparently. To understand its behavior, one should first look at the auxiliary function* `foldp`. *If* `m` *and* `n` *are two natural numbers and* `f` *and* `g` *are two functions,* `foldp m n f g` *call-by-name reduces to the following expression:*

`(f n) (+) ((f n+1) (+) ... ((f n+m-1) (+) (g n+m))).`

The term `expthree` *works by forwarding its three arguments to* `foldp`. *The fourth argument is a recursive call to* `expthree` *where, however,* `k` *is replaced by any number greater or equal to it, chosen according to a geometric distribution. The functions above can all be expressed in* $\Lambda_\oplus$, *using fixed-point combinators. As we will see soon,* `expone`, `exptwo`, *and* `expthree k` *are context equivalent whenever* `k` *is a natural number.*

2.2 Probabilistic Bisimulation

In this section we recall the definition and a few basic notions of bisimulation for labelled Markov chains, following Larsen and Skou [26]. In Section 3 we will then adapt this form of bisimilarity to the probabilistic λ-calculus $\Lambda_{\oplus}$ by combining it with Abramsky's applicative bisimilarity.

Definition 2.7 *A* labelled Markov chain *is a triple* $(\mathcal{S}, \mathcal{L}, \mathcal{P})$ *such that:*

- $\mathcal{S}$ *is a countable set of* states*;*
- $\mathcal{L}$ *is set of* labels*;*
- $\mathcal{P}$ *is a* transition probability matrix*, i.e. a function*

$$\mathcal{P} : \mathcal{S} \times \mathcal{L} \times \mathcal{S} \to \mathbb{R}_{[0,1]}$$

such that the following normalization condition holds:

$$\forall \ell \in \mathcal{L}. \forall s \in \mathcal{S}. \mathcal{P}(s, \ell, \mathcal{S}) \leq 1$$

where, as usual $\mathcal{P}(s, \ell, X)$ *stands for* $\sum_{t \in X} \mathcal{P}(s, \ell, t)$ *whenever* $X \subseteq \mathcal{S}$*.*

If $\mathcal{R}$ is an equivalence relation on $\mathcal{S}$, $\mathcal{S}/\mathcal{R}$ denotes the *quotient* of $\mathcal{S}$ modulo $\mathcal{R}$, i.e., the set of all equivalence classes of $\mathcal{S}$ modulo $\mathcal{R}$. Given any binary relation $\mathcal{R}$, its reflexive and transitive closure is denoted as $\mathcal{R}^*$.

Definition 2.8 *Given a labelled Markov chain* $(\mathcal{S}, \mathcal{L}, \mathcal{P})$*, a* probabilistic bisimulation *is an equivalence relation* $\mathcal{R}$ *on* $\mathcal{S}$ *such that* $(s, t) \in \mathcal{R}$ *implies that for every* $\ell \in \mathcal{L}$ *and for every* $\mathtt{E} \in \mathcal{S}/\mathcal{R}$*,* $\mathcal{P}(s, \ell, \mathtt{E}) = \mathcal{P}(t, \ell, \mathtt{E})$*.*

Note that a probabilistic bisimulation has to be, by definition, an *equivalence relation*. This means that, in principle, we are not allowed to define probabilistic bisimilarity simply as the union of all probabilistic bisimulations. As a matter of fact, given $\mathcal{R}, \mathcal{T}$ two equivalence relations, $\mathcal{R} \cup \mathcal{T}$ is not necessarily an equivalence relation. The following is a standard way to overcome the problem:

Lemma 2.9 *If* $\{\mathcal{R}_i\}_{i \in I}$*, is a collection of probabilistic bisimulations, then also their reflexive and transitive closure* $(\bigcup_{i \in I} \mathcal{R}_i)^*$ *is a probabilistic bisimulation.*

Proof. Let us fix $\mathcal{T} \stackrel{\text{def}}{=} (\bigcup_{i \in I} \mathcal{R}_i)^*$. The fact that $\mathcal{T}$ is an equivalence relation can be proved as follows:

- Reflexivity is easy: $\mathcal{T}$ is reflexive by definition.
- Symmetry is a consequence of symmetry of each of the relations in $\{\mathcal{R}_i\}_{i \in I}$: if $s \; \mathcal{T} \; t$, then there are $n \geq 0$ states $v_0, \ldots, v_n$ such that $v_0 = s$, $v_n = t$ and for every $1 \leq i \leq n$ there is j such that $v_{i-1} \; \mathcal{R}_j \; v_i$. By the symmetry of each of the $\mathcal{R}_j$, we easily get that $v_i \; \mathcal{R}_j \; v_{i-1}$. As a consequence, $t \; \mathcal{T} \; s$.
- Transitivity is itself very easy: $\mathcal{T}$ is transitive by definition.

Now, please notice that for any $i \in I$, $\mathcal{R}_i \subseteq \bigcup_{j \in I} \mathcal{R}_j \subseteq \mathcal{T}$. This means that any equivalence class with respect to $\mathcal{T}$ is the union of equivalence classes with respect to $\mathcal{R}_i$. Suppose that $s \; \mathcal{T} \; t$. Then there are $n \geq 0$ states $v_0, \ldots, v_n$ such that $v_0 = s$, $v_n = t$ and for every $1 \leq i \leq n$ there is j such that $v_{i-1} \; \mathcal{R}_j \; v_i$. Now, if $\ell \in \mathcal{L}$ and $\mathtt{E} \in \mathcal{S}/\mathcal{T}$, we obtain

$$\mathcal{P}(s, \ell, \mathtt{E}) = \mathcal{P}(v_0, \ell, \mathtt{E}) = \ldots = \mathcal{P}(v_n, \ell, \mathtt{E}) = \mathcal{P}(t, \ell, \mathtt{E}).$$

This concludes the proof. □

Lemma 2.9 allows us to define the largest probabilistic bisimulation, called *probabilistic bisimilarity*. It is $\sim \stackrel{\text{def}}{=} \bigcup \{\mathcal{R} \mid \mathcal{R} \text{ is a probabilistic bisimulation}\}$. Indeed, by Lemma 2.9, $(\sim)^*$ is a probabilistic bisimulation too; we now claim that $\sim = (\sim)^*$. The inclusion $\sim \subseteq (\sim)^*$ is obvious. The other way around, $\sim \supseteq (\sim)^*$, follows by $(\sim)^*$ being a probabilistic bisimulation and hence included in the union of them all, that is $\sim$.

In the notion of a probabilistic simulation, preorders play the role of equivalence relations: given a labelled Markov chain $(\mathcal{S}, \mathcal{L}, \mathcal{P})$, a *probabilistic simulation* is a preorder relation $\mathcal{R}$ on $\mathcal{S}$ such that $(s,t) \in \mathcal{R}$ implies that for every $\ell \in \mathcal{L}$ and for every $X \subseteq \mathcal{S}$, $\mathcal{P}(s,\ell,X) \leq \mathcal{P}(t,\ell,\mathcal{R}(X))$, where as usual $\mathcal{R}(X)$ stands for the $\mathcal{R}$-*closure* of X, namely the set $\{y \in \mathcal{S} \mid \exists\, x \in X.\ x\ \mathcal{R}\ y\}$. Lemma 2.9 can be adapted to probabilistic simulations:

Proposition 2.10 *If $\{\mathcal{R}_i\}_{i\in I}$, is a collection of probabilistic simulations, then also their reflexive and transitive closure $(\bigcup_{i\in I} \mathcal{R}_i)^*$ is a probabilistic simulation.*

Proof. The fact that $\mathcal{R} \stackrel{\text{def}}{=} (\bigcup_{i\in I} \mathcal{R}_i)^*$ is a preorder follows by construction. Then, for being a probabilistic simulation $\mathcal{R}$ must satisfy the following property: $(s,t) \in \mathcal{R}$ implies that for every $\ell \in \mathcal{L}$ and for every $X \subseteq \mathcal{S}$, $\mathcal{P}(s,\ell,X) \leq \mathcal{P}(t,\ell,\mathcal{R}(X))$. Let $(s,t) \in \mathcal{R}$. There are $n \geq 0$ states $v_1, \ldots, v_n$ and for every $2 \leq i \leq n$ there is j_i such that

$$s = v_1\ \mathcal{R}_{j_2}\ v_2 \ldots v_{n-1} \mathcal{R}_{j_n} v_n = t.$$

As a consequence, for every $\ell \in \mathcal{L}$ and for every $X \subseteq \mathcal{S}$, it holds that

$$\mathcal{P}(v_1,\ell,X) \leq \mathcal{P}(v_2,\ell,\mathcal{R}_{j_2}(X)) \leq \mathcal{P}(v_3,\ell,\mathcal{R}_{j_3}(\mathcal{R}_{j_2}(X))) \leq \cdots \leq \mathcal{P}(v_n,\ell,\mathcal{R}_{j_n}(\ldots(\mathcal{R}_{j_2}(\mathcal{R}_{j_1}(X)))))$$

Since, by definition,

$$\mathcal{R}_{j_n}(\ldots(\mathcal{R}_{j_2}(\mathcal{R}_{j_1}(X)))) \subseteq \mathcal{R}(X),$$

it follows that $\mathcal{P}(s,\ell,X) \leq \mathcal{P}(t,\ell,\mathcal{R}(X))$. This concludes the proof. □

As a consequence, we define *similarity* simply as $\lesssim \stackrel{\text{def}}{=} \bigcup\{\mathcal{R} \mid \mathcal{R} \text{ is a probabilistic simulation}\}$.

Any symmetric probabilistic simulation is a probabilistic bisimulation.

Lemma 2.11 *If $\mathcal{R}$ is a symmetric probabilistic simulation, then $\mathcal{R}$ is a probabilistic bisimulation.*

Proof. If $\mathcal{R}$ is a symmetric probabilistic simulation, by definition, it is also a preorder: that is, it is a reflexive and transitive relation. Therefore, $\mathcal{R}$ is an equivalence relation. But for being a probabilistic bisimulation $\mathcal{R}$ must also satisfy the property that $s\mathcal{R}t$ implies, for every $\ell \in \mathcal{L}$ and for every $\mathtt{E} \in \mathcal{S}/\mathcal{R}$, $\mathcal{P}(s,\ell,\mathtt{E}) = \mathcal{P}(t,\ell,\mathtt{E})$. From the fact that $\mathcal{R}$ is a simulation, it follows that if $s\mathcal{R}t$, for every $\ell \in \mathcal{L}$ and for every $\mathtt{E} \in \mathcal{S}/\mathcal{R}$, $\mathcal{P}(s,\ell,\mathtt{E}) \leq \mathcal{P}(t,\ell,\mathcal{R}(\mathtt{E}))$. Since $\mathtt{E} \in \mathcal{S}/\mathcal{R}$ is an $\mathcal{R}$-equivalence class, it holds $\mathcal{R}(\mathtt{E}) = \mathtt{E}$. Then, from the latter follows $\mathcal{P}(s,\ell,\mathtt{E}) \leq \mathcal{P}(t,\ell,\mathtt{E})$. We get the other way around by symmetric property of $\mathcal{R}$, which implies that, for every label ℓ and for every $\mathtt{E} \in \mathcal{S}/\mathcal{R}$, $\mathcal{P}(t,\ell,\mathtt{E}) \leq \mathcal{P}(s,\ell,\mathtt{E})$. Hence, $\mathcal{P}(s,\ell,\mathtt{E}) = \mathcal{P}(t,\ell,\mathtt{E})$ which completes the proof. □

Moreover, every probabilistic bisimulation, and its inverse, is a probabilistic simulation.

Lemma 2.12 *If $\mathcal{R}$ is a probabilistic bisimulation, then $\mathcal{R}$ and $\mathcal{R}^{op}$ are probabilistic simulation.*

Proof. Let us prove $\mathcal{R}$ probabilistic simulation first. Consider the set $\{X_i\}_{i\in I}$ of equivalence subclasses module $\mathcal{R}$ contained in X. Formally, $X = \biguplus_{i\in I} X_i$ such that, for all $i \in I$, $X_i \subseteq \mathtt{E}_i$ with $\mathtt{E}_i$ equivalence class modulo $\mathcal{R}$. Please observe that, as a consequence, $\mathcal{R}(X) = \biguplus_{i\in I} \mathtt{E}_i$. Thus, the result easily follows, for every $\ell \in \mathcal{L}$ and every $X \subseteq \mathcal{S}$,

$$\begin{aligned}
\mathcal{P}(s,\ell,X) &= \sum_{i\in I} \mathcal{P}(s,\ell,X_i) \\
&\leq \sum_{i\in I} \mathcal{P}(s,\ell,\mathtt{E}_i) \\
&= \sum_{i\in I} \mathcal{P}(t,\ell,\mathtt{E}_i) = \mathcal{P}(t,\ell,\mathcal{R}(X)).
\end{aligned}$$

Finally, $\mathcal{R}^{op}$ is also a probabilistic simulation as a consequence of symmetric property of $\mathcal{R}$ and the fact, just proved, that $\mathcal{R}$ is a probabilistic simulation. □

Contrary to the nondeterministic case, however, simulation equivalence coincides with bisimulation:

Proposition 2.13 *$\sim$ coincides with $\lesssim \cap \lesssim^{op}$.*

Proof. The fact that $\sim$ is a subset of $\lesssim \cap \lesssim^{op}$ is a straightforward consequence of symmetry property of $\sim$ and the fact that, by Lemma 2.12, every probabilistic bisimulation is also a probabilistic simulation. Let us now prove that $\lesssim \cap \lesssim^{op}$ is a subset of $\sim$, i.e., the former of being a probabilistic bisimulation. Of course, $\lesssim \cap \lesssim^{op}$ is an equivalence relation because $\lesssim$ is a preorder. Now, consider any equivalence class $\mathtt{E}$ modulo $\lesssim \cap \lesssim^{op}$. Define the following two sets of states $X \stackrel{\text{def}}{=} \lesssim(\mathtt{E})$ and $Y \stackrel{\text{def}}{=} X - \mathtt{E}$. Observe that Y and $\mathtt{E}$ are disjoint set of states whose union is precisely X. Moreover, notice that both X and Y are closed with respect to $\lesssim$:

- On the one hand, if $s \in \lesssim(X)$, then $s \in \lesssim(\lesssim(\mathtt{E})) = \lesssim(\mathtt{E}) = X$;
- On the other hand, if $s \in \lesssim(Y)$, then there is $t \in X$ which is not in $\mathtt{E}$ such that $t \lesssim s$. But then s is itself in X (see the previous point), but cannot be $\mathtt{E}$, because otherwise we would have $s \lesssim t$, meaning that s and t are in the same equivalence class modulo $\lesssim \cap \lesssim^{op}$, and thus $t \in \mathtt{E}$, a contradiction.

As a consequence, given any $(s,t) \in \lesssim \cap \lesssim^{op}$ and any $\ell \in \mathcal{L}$,

$$\mathcal{P}(s,\ell,X) \leq \mathcal{P}(t,\ell,\lesssim(X)) = \mathcal{P}(t,\ell,X),$$
$$\mathcal{P}(t,\ell,X) \leq \mathcal{P}(s,\ell,\lesssim(X)) = \mathcal{P}(s,\ell,X).$$

It follows $\mathcal{P}(s,\ell,X) = \mathcal{P}(t,\ell,X)$ and, similarly, $\mathcal{P}(s,\ell,Y) = \mathcal{P}(t,\ell,Y)$. But then,

$$\begin{aligned}\mathcal{P}(s,\ell,\mathtt{E}) &= \mathcal{P}(s,\ell,X) - \mathcal{P}(s,\ell,Y)\\ &= \mathcal{P}(t,\ell,X) - \mathcal{P}(t,\ell,Y) = \mathcal{P}(t,\ell,\mathtt{E})\end{aligned}$$

which is the thesis. □

For technical reasons that will become apparent soon, it is convenient to consider Markov chains in which the state space is partitioned into disjoint sets, in such a way that comparing states coming from different components is not possible. Remember that the disjoint union $\biguplus_{i\in I} X_i$ of a family of sets $\{X_i\}_{i\in I}$ is defined as $\{(a,i) \mid i \in I \wedge a \in X_i\}$. If the set of states $\mathcal{S}$ of a labelled Markov chain is a disjoint union $\biguplus_{i\in I} X_i$, one wants that (bi)simulation relations only compare elements coming from the same X_i, i.e. $(a,i)\mathcal{R}(b,j)$ implies $i = j$. In this case, we say that the underlying labelled Markov chain is *multisorted*.

3 Probabilistic Applicative Bisimulation and Howe's technique

In this section, notions of similarity and bisimilarity for $\Lambda_\oplus$ are introduced, in the spirit of Abramsky's work on applicative bisimulation [1]. Definitionally, this consists in seeing $\Lambda_\oplus$'s operational semantics as a labelled Markov chain, then giving the Larsen and Skou's notion of (bi)simulation for it. States will be terms, while labels will be of two kinds: one can either *evaluate* a term, obtaining (a distribution of) values, or *apply* a term to a value.

The resulting bisimulation (probabilistic applicative bisimulation) will be shown to be a congruence, thus included in probabilistic context equivalence. This will be done by a non-trivial generalization of Howe's technique [22], which is a well-known methodology to get congruence results in presence of higher-order functions, but which has not been applied to probabilistic calculi so far.

Formalizing probabilistic applicative bisimulation requires some care. As usual, two values $\lambda x.M$ and $\lambda x.N$ are defined to be bisimilar if for every L, $M\{^L/_x\}$ and $N\{^L/_x\}$ are themselves bisimilar. But how if we rather want to compare two arbitrary closed terms M and N? The simplest solution consists in following Larsen and Skou and stipulate that every equivalence class

of $\mathsf{V}\Lambda_\oplus$ modulo bisimulation is attributed the same measure by both $[\![M]\!]$ and $[\![N]\!]$. Values are thus treated in two different ways (they are both terms and values), and this is the reason why each of them corresponds to *two* states in the underlying Markov chain.

Definition 3.1 *$\Lambda_\oplus$ can be seen as a multisorted labelled Markov chain $(\Lambda_\oplus(\emptyset) \uplus \mathsf{V}\Lambda_\oplus, \Lambda_\oplus(\emptyset) \uplus \{\tau\}, \mathcal{P}_\oplus)$ that we denote with $\Lambda_\oplus$. Labels are either closed terms, which model parameter passing, or τ, that models evaluation. Please observe that the states of the labelled Markov chain we have just defined are elements of the disjoint union $\Lambda_\oplus(\emptyset) \uplus \mathsf{V}\Lambda_\oplus$. Two distinct states correspond to the same value V, and to avoid ambiguities, we call the second one (i.e. the one coming from $\mathsf{V}\Lambda_\oplus$) a* distinguished value. *When we want to insist on the fact that a value $\lambda x.M$ is distinguished, we indicate it with $\nu x.M$. We define the transition probability matrix $\mathcal{P}_\oplus$ as follows:*

- *For every term M and for every distinguished value $\nu x.N$,*

$$\mathcal{P}_\oplus(M, \tau, \nu x.N) \stackrel{\text{def}}{=} [\![M]\!](\nu x.N);$$

- *For every term M and for every distinguished value $\nu x.N$,*

$$\mathcal{P}_\oplus(\nu x.N, M, N\{^M/_x\}) \stackrel{\text{def}}{=} 1;$$

- *In all other cases, $\mathcal{P}_\oplus$ returns* 0.

Terms seen as states only interact with the environment by performing τ, while distinguished values only take other closed terms as parameters.

Simulation and bisimulation relations can be defined for $\Lambda_\oplus$ as for any labelled Markov chain. Even if, strictly speaking, these are binary relations on $\Lambda_\oplus(\emptyset) \uplus \mathsf{V}\Lambda_\oplus$, we often see them just as their restrictions to $\Lambda_\oplus(\emptyset)$. Formally, a *probabilistic applicative bisimulation* (a PAB) is simply a probabilistic bisimulation on $\Lambda_\oplus$. This way one can define *probabilistic applicative bisimilarity*, which is denoted $\sim$. Similarly for *probabilistic applicative simulation* (PAS) and *probabilistic applicative similarity*, denoted $\lesssim$.

Remark 3.2 (Early vs. Late) *Technically, the distinction between terms and values in Definition 3.1 means that our bisimulation is in* late *style. In bisimulations for value-passing concurrent languages, "late" indicates the explicit manipulation of functions in the clause for input actions: functions are chosen first, and only later, the input value received is taken into account [46]. Late-style is used in contraposition to* early *style, where the order of quantifiers is exchanged, so that the choice of functions may depend on the specific input value received. In our setting, adopting an early style would mean having transitions such as $\lambda x.M \xrightarrow{N} M\{^N/_x\}$, and then setting up a probabilistic bisimulation on top of the resulting transition system. We leave for future work a study of the comparison between the two styles. In this paper, we stick to the late style because easier to deal with, especially under Howe's technique. Previous works on applicative bisimulation for nondeterministic functions also focus on the late approach [33, 38].*

Remark 3.3 *Defining applicative bisimulation in terms of multisorted labelled Markov chains has the advantage of recasting the definition in a familiar framework; most importantly, this formulation will be useful when dealing with Howe's method. To spell out the explicit operational details of the definition, a probabilistic applicative bisimulation can be seen as an equivalence relation $\mathcal{R} \subseteq \Lambda_\oplus(\emptyset) \times \Lambda_\oplus(\emptyset)$ such that whenever $M\ \mathcal{R}\ N$:*

1. *$[\![M]\!](\mathtt{E} \cap \mathsf{V}\Lambda_\oplus) = [\![N]\!](\mathtt{E} \cap \mathsf{V}\Lambda_\oplus)$, for any equivalence class $\mathtt{E}$ of $\mathcal{R}$ (that is, the probability of reaching a value in $\mathtt{E}$ is the same for the two terms);*
2. *if M and N are values, say $\lambda x.P$ and $\lambda x.Q$, then $P\{^L/_x\}\ \mathcal{R}\ Q\{^L/_x\}$, for all $L \in \Lambda_\oplus(\emptyset)$.*

The special treatment of values, in Clause 2., motivates the use of multisorted *labelled Markov chains in Definition 3.1.*

As usual, one way to show that any two terms are bisimilar is to prove that one relation containing the pair in question is a PAB. Terms with the same semantics are indistinguishable:

Lemma 3.4 *The binary relation* $\mathcal{R} = \{(M,N) \in \Lambda_\oplus(\emptyset) \times \Lambda_\oplus(\emptyset) \text{ s.t. } [\![M]\!] = [\![N]\!]\} \uplus \{(V,V) \in \mathsf{V}\Lambda_\oplus \times \mathsf{V}\Lambda_\oplus\}$ *is a* PAB.

Proof. The fact $\mathcal{R}$ is an equivalence easily follows from reflexivity, symmetry and transitivity of set-theoretic equality. $\mathcal{R}$ must satisfy the following property for closed terms: if $M\mathcal{R}N$, then for every $\mathtt{E} \in \mathsf{V}\Lambda_\oplus/\mathcal{R}$, $\mathcal{P}_\oplus(M,\tau,\mathtt{E}) = \mathcal{P}_\oplus(N,\tau,\mathtt{E})$. Notice that if $[\![M]\!] = [\![N]\!]$, then clearly $\mathcal{P}_\oplus(M,\tau,V) = \mathcal{P}_\oplus(N,\tau,V)$, for every $V \in \mathsf{V}\Lambda_\oplus$. With the same hypothesis,

$$\begin{aligned}\mathcal{P}_\oplus(M,\tau,\mathtt{E}) &= \sum_{V\in\mathtt{E}} \mathcal{P}_\oplus(M,\tau,V)\\ &= \sum_{V\in\mathtt{E}} \mathcal{P}_\oplus(N,\tau,V) = \mathcal{P}_\oplus(N,\tau,\mathtt{E}).\end{aligned}$$

Moreover, $\mathcal{R}$ must satisfy the following property for cloned values: if $\nu x.M\mathcal{R}\nu x.N$, then for every close term L and for every $\mathtt{E} \in \Lambda_\oplus(\emptyset)/\mathcal{R}$, $\mathcal{P}_\oplus(\nu x.M,L,\mathtt{E}) = \mathcal{P}_\oplus(\nu x.N,L,\mathtt{E})$. Now, the hypothesis $[\![\nu x.M]\!] = [\![\nu x.N]\!]$ implies $M = N$. Then clearly $\mathcal{P}_\oplus(\nu x.M,L,P) = \mathcal{P}_\oplus(\nu x.N,L,P)$ for every $P \in \Lambda_\oplus(\emptyset)$. With the same hypothesis,

$$\begin{aligned}\mathcal{P}_\oplus(\nu x.M,L,\mathtt{E}) &= \sum_{P\in\mathtt{E}} \mathcal{P}_\oplus(\nu x.M,L,P)\\ &= \sum_{P\in\mathtt{E}} \mathcal{P}_\oplus(\nu x.N,L,P) = \mathcal{P}_\oplus(\nu x.N,L,\mathtt{E}).\end{aligned}$$

This concludes the proof. □

Please notice that the previous result yield a nice consequence: for every $M, N \in \Lambda_\oplus(\emptyset)$, $(\lambda x.M)N \sim M\{^N/_x\}$. Indeed, Lemma 2.3 tells us that the latter terms have the same semantics.

Conversely, knowing that two terms M and N are (bi)similar means knowing quite a lot about their convergence probability:

Lemma 3.5 (Adequacy of Bisimulation) *If* $M \sim N$, *then* $\sum[\![M]\!] = \sum[\![N]\!]$. *Moreover, if* $M \lesssim N$, *then* $\sum[\![M]\!] \leq \sum[\![N]\!]$.

Proof.

$$\begin{aligned}\sum[\![M]\!] &= \sum_{\mathtt{E}\in\mathsf{V}\Lambda_\oplus/\sim} \mathcal{P}_\oplus(M,\tau,\mathtt{E})\\ &= \sum_{\mathtt{E}\in\mathsf{V}\Lambda_\oplus/\sim} \mathcal{P}_\oplus(N,\tau,\mathtt{E}) = \sum[\![N]\!].\end{aligned}$$

And,

$$\begin{aligned}\sum[\![M]\!] &= \mathcal{P}_\oplus(M,\tau,\mathsf{V}\Lambda_\oplus)\\ &\leq \mathcal{P}_\oplus(N,\tau,\lesssim(\mathsf{V}\Lambda_\oplus))\\ &= \mathcal{P}_\oplus(N,\tau,\mathsf{V}\Lambda_\oplus) = \sum[\![N]\!].\end{aligned}$$

This concludes the proof. □

Example 3.6 *Bisimilar terms do not necessarily have the same semantics. After all, this is one reason for using bisimulation, and its proof method, as basis to prove fine-grained equalities among functions. Let us consider the following terms:*

$$M \stackrel{\text{def}}{=} ((\lambda x.(x \oplus x)) \oplus (\lambda x.x)) \oplus \Omega;$$
$$N \stackrel{\text{def}}{=} \Omega \oplus (\lambda x.Ix);$$

Their semantics differ, as for every value V, we have:

$$\begin{aligned}[\![M]\!](V) &= \begin{cases} \frac{1}{4} & \text{if } V \text{ is } \lambda x.(x \oplus x) \text{ or } \lambda x.x; \\ 0 & \text{otherwise}; \end{cases} \\ [\![N]\!](V) &= \begin{cases} \frac{1}{2} & \text{if } V \text{ is } \lambda x.Ix; \\ 0 & \text{otherwise}. \end{cases}\end{aligned}$$

Nonetheless, we can prove $M \sim N$. Indeed, $\nu x.(x \oplus x) \sim \nu x.x \sim \nu x.Ix$ because, for every $L \in \Lambda_\oplus(\emptyset)$, the three terms L, $L \oplus L$ and IL all have the same semantics, i.e., $[\![L]\!]$. Now, consider any equivalence class $\mathtt{E}$ of distinguished values modulo $\sim$. If $\mathtt{E}$ includes the three distinguished values above, then

$$\mathcal{P}_\oplus(M, \tau, \mathtt{E}) = \sum_{V \in \mathtt{E}} [\![M]\!](V) = \frac{1}{2} = \sum_{V \in \mathtt{E}} [\![N]\!](V) = \mathcal{P}_\oplus(N, \tau, \mathtt{E}).$$

Otherwise, $\mathcal{P}_\oplus(M, \tau, \mathtt{E}) = 0 = \mathcal{P}_\oplus(N, \tau, \mathtt{E})$.

Let us prove the following technical result that, moreover, stipulate that bisimilar distinguished values are bisimilar values.

Lemma 3.7 *$\lambda x.M \sim \lambda x.N$ iff $\nu x.M \sim \nu x.N$ iff $M\{^L/_x\} \sim N\{^L/_x\}$, for all $L \in \Lambda_\oplus(\emptyset)$.*

Proof. The first double implication is obvious. For that matter, distinguished values are value terms. Let us now detail the second double implication. ($\Rightarrow$) The fact that $\sim$ is a PAB implies, by its definition, that for every $L \in \Lambda_\oplus(\emptyset)$ and every $\mathtt{E} \in \Lambda_\oplus(\emptyset)/\!\sim$, $\mathcal{P}_\oplus(\nu x.M, L, \mathtt{E}) = \mathcal{P}_\oplus(\nu x.N, L, \mathtt{E})$. Suppose then, by contradiction, that $M\{^L/_x\} \not\sim N\{^L/_x\}$, for some $L \in \Lambda_\oplus(\emptyset)$. The latter means that, there exists $\mathtt{F} \in \Lambda_\oplus(\emptyset)/\!\sim$ such that $M\{^L/_x\} \in \mathtt{F}$ and $N\{^L/_x\} \notin \mathtt{F}$. According to its definition, for all $P \in \Lambda_\oplus(\emptyset)$, $\mathcal{P}_\oplus(\nu x.M, L, P) = 1$ iff $P \equiv M\{^L/_x\}$, and $\mathcal{P}_\oplus(\nu x.M, L, P) = 0$ otherwise. Then, since $M\{^L/_x\} \in \mathtt{F}$, we derive $\mathcal{P}_\oplus(\nu x.M, L, \mathtt{F}) = \sum_{P \in \mathtt{F}} \mathcal{P}_\oplus(\lambda x.M, L, P) \geq \mathcal{P}_\oplus(\nu x.M, L, M\{^L/_x\}) = 1$, which implies $\sum_{P \in \mathtt{F}} \mathcal{P}_\oplus(\nu x.M, L, P) = \mathcal{P}_\oplus(\nu x.M, L, \mathtt{F}) = 1$. Although $\nu x.N$ is a distinguished value and the starting reasoning we have just made above still holds, $\mathcal{P}_\oplus(\nu x.N, L, \mathtt{F}) = \sum_{P \in \mathtt{F}} \mathcal{P}_\oplus(\nu x.N, L, P) = 0$. We get the latter because there is no $P \in \mathtt{F}$ of the form $N\{^L/_x\}$ due to the hypothesis that $N\{^L/_x\} \notin \mathtt{F}$.

From the hypothesis on the equivalence class $\mathtt{F}$, i.e. $\mathcal{P}_\oplus(\nu x.M, L, \mathtt{F}) = \mathcal{P}_\oplus(\nu x.N, L, \mathtt{F})$, we derive the absurd:

$$1 = \mathcal{P}_\oplus(\nu x.M, L, \mathtt{F}) = \mathcal{P}_\oplus(\nu x.N, L, \mathtt{F}) = 0.$$

($\Leftarrow$) We need to prove that, for every $L \in \Lambda_\oplus(\emptyset)$ and every $\mathtt{E} \in \Lambda_\oplus(\emptyset)/\!\sim$, $\mathcal{P}_\oplus(\nu x.M, L, \mathtt{E}) = \mathcal{P}_\oplus(\nu x.N, L, \mathtt{E})$ supposing that $M\{^L/_x\} \sim N\{^L/_x\}$ holds. First of all, let us rewrite $\mathcal{P}_\oplus(\nu x.M, L, \mathtt{E})$ and $\mathcal{P}_\oplus(\nu x.N, L, \mathtt{E})$ as $\sum_{P \in \mathtt{E}} \mathcal{P}_\oplus(\nu x.M, L, P)$ and $\sum_{P \in \mathtt{E}} \mathcal{P}_\oplus(\nu x.N, L, P)$ respectively. Then, from the hypothesis and the same reasoning we have made for ($\Rightarrow$), for every $\mathtt{E} \in \Lambda_\oplus(\emptyset)/\!\sim$:

$$\sum_{P \in \mathtt{E}} \mathcal{P}_\oplus(\nu x.M, L, P) = \begin{cases} 1 & \text{if } M\{^L/_x\} \in \mathtt{E}; \\ 0 & \text{otherwise} \end{cases} = \begin{cases} 1 & \text{if } N\{^L/_x\} \in \mathtt{E}; \\ 0 & \text{otherwise} \end{cases} = \sum_{P \in \mathtt{E}} \mathcal{P}_\oplus(\nu x.N, L, P)$$

which proves the thesis. □

The same result holds for $\lesssim$.

3.1 Probabilistic Applicative Bisimulation is a Congruence

In this section, we prove that probabilistic applicative bisimulation is indeed a congruence, and that its non-symmetric sibling is a precongruence. The overall structure of the proof is similar to the one by Howe [22]. The main idea consists in defining a way to turn an arbitrary relation $\mathcal{R}$ on (possibly open) terms to another one, $\mathcal{R}^H$, in such a way that, if $\mathcal{R}$ satisfies a few simple

conditions, then $\mathcal{R}^H$ is a (pre)congruence including $\mathcal{R}$. The key step, then, is to prove that $\mathcal{R}^H$ is indeed a (bi)simulation. In view of Proposition 2.13, considering similarity suffices here.

It is here convenient to work with generalizations of relations called $\Lambda_\oplus$*-relations*, i.e. sets of triples in the form $(\overline{x}, M, N)$, where $M, N \in \Lambda_\oplus(\overline{x})$. Thus if a relation has the pair (M, N) with $M, N \in \Lambda_\oplus(\overline{x})$, then the corresponding $\Lambda_\oplus$-relation will include $(\overline{x}, M, N)$. (Recall that applicative (bi)similarity is extended to open terms by considering all closing substitutions.) Given any $\Lambda_\oplus$-relation $\mathcal{R}$, we write $\overline{x} \vdash M\ \mathcal{R}\ N$ if $(\overline{x}, M, N) \in \mathcal{R}$. A $\Lambda_\oplus$-relation $\mathcal{R}$ is said to be *compatible* iff the four conditions below hold:
(Com1) $\forall \overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X})$, $x \in \overline{x}$: $\overline{x} \vdash x\ \mathcal{R}\ x$,
(Com2) $\forall \overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X}), \forall x \in \mathsf{X} - \overline{x}, \forall M, N \in \Lambda_\oplus(\overline{x} \cup \{x\})$: $\overline{x} \cup \{x\} \vdash M\ \mathcal{R}\ N \Rightarrow \overline{x} \vdash \lambda x.M\ \mathcal{R}\ \lambda x.N$,
(Com3) $\forall \overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X}), \forall M, N, L, P \in \Lambda_\oplus(\overline{x})$: $\overline{x} \vdash M\ \mathcal{R}\ N \wedge \overline{x} \vdash L\ \mathcal{R}\ P \Rightarrow \overline{x} \vdash ML\ \mathcal{R}\ NP$,
(Com4) $\forall \overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X}), \forall M, N, L, P \in \Lambda_\oplus(\overline{x})$: $\overline{x} \vdash M\ \mathcal{R}\ N \wedge \overline{x} \vdash L\ \mathcal{R}\ P \Rightarrow \overline{x} \vdash M \oplus L\ \mathcal{R}\ N \oplus P$.
We will often use the following technical results to establish (Com3) and (Com4) under particular hypothesis.

Lemma 3.8 *Let us consider the properties*
(Com3L) $\forall \overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X}), \forall M, N, L \in \Lambda_\oplus(\overline{x})$: $\overline{x} \vdash M\ \mathcal{R}\ N \Rightarrow \overline{x} \vdash ML\ \mathcal{R}\ NL$,
(Com3R) $\forall \overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X}), \forall M, N, L \in \Lambda_\oplus(\overline{x})$: $\overline{x} \vdash M\ \mathcal{R}\ N \Rightarrow \overline{x} \vdash LM\ \mathcal{R}\ LN$.
If $\mathcal{R}$ is transitive, then (Com3L) *and* (Com3R) *together imply* (Com3).

Proof. Proving (Com3) means to show that the hypothesis $\overline{x} \vdash M\ \mathcal{R}\ N$ and $\overline{x} \vdash L\ \mathcal{R}\ P$ imply $\overline{x} \vdash ML\ \mathcal{R}\ NP$. Using (Com3L) on the first one, with L as steady term, it follows $\overline{x} \vdash ML\ \mathcal{R}\ NL$. Similarly, using (Com3R) on the second one, with N as steady term, it follows $\overline{x} \vdash NL\ \mathcal{R}\ NP$. Then, we conclude by transitivity property of $\mathcal{R}$. □

Lemma 3.9 *Let us consider the properties*
(Com4L) $\forall \overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X}), \forall M, N, L \in \Lambda_\oplus(\overline{x})$: $\overline{x} \vdash M\ \mathcal{R}\ N \Rightarrow \overline{x} \vdash M \oplus L\ \mathcal{R}\ N \oplus L$,
(Com4R) $\forall \overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X}), \forall M, N, L \in \Lambda_\oplus(\overline{x})$: $\overline{x} \vdash M\ \mathcal{R}\ N \Rightarrow \overline{x} \vdash L \oplus M\ \mathcal{R}\ L \oplus N$.
If $\mathcal{R}$ is transitive, then (Com4L) *and* (Com4R) *together imply* (Com4).

Proof. Proving (Com4) means to show that the hypothesis $\overline{x} \vdash M\ \mathcal{R}\ N$ and $\overline{x} \vdash L\ \mathcal{R}\ P$ imply $\overline{x} \vdash M \oplus L\ \mathcal{R}\ N \oplus P$. Using (Com4L) on the first one, with L as steady term, it follows $\overline{x} \vdash M \oplus L\ \mathcal{R}\ N \oplus L$. Similarly, using (Com4R) on the second one, with N as steady term, it follows $\overline{x} \vdash N \oplus L\ \mathcal{R}\ N \oplus P$. Then, we conclude by transitivity property of $\mathcal{R}$. □

The notions of an equivalence relation and of a preorder can be straightforwardly generalized to $\Lambda_\oplus$-relations, and any compatible $\Lambda_\oplus$-relation that is an equivalence relation (respectively, a preorder) is said to be a *congruence* (respectively, a *precongruence*).

If bisimilarity is a congruence, then $C[M]$ is bisimilar to $C[N]$ whenever $M \sim N$ and C is a context. In other words, terms can be replaced by equivalent ones in any context. This is a crucial sanity-check any notion of equivalence is expected to pass.

It is well-known that proving bisimulation to be a congruence may be nontrivial when the underlying language contains higher-order functions. This is also the case here. Proving (Com1), (Com2) and (Com4) just by inspecting the operational semantics of the involved terms is indeed possible, but the method fails for (Com3), when the involved contexts contain applications. In particular, proving (Com3) requires probabilistic applicative bisimilarity of being stable with respect to substitution of bisimilar terms, hence not necessarily the same. In general, a $\Lambda_\oplus$-relation $\mathcal{R}$ is called (term) *substitutive* if for all $\overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X})$, $x \in \mathsf{X} - \overline{x}$, $M, N \in \Lambda_\oplus(\overline{x} \cup \{x\})$ and $L, P \in \Lambda_\oplus(\overline{x})$

$$\overline{x} \cup \{x\} \vdash M\ \mathcal{R}\ N \wedge \overline{x} \vdash L\ \mathcal{R}\ P \Rightarrow \overline{x} \vdash M\{^L/_x\}\ \mathcal{R}\ N\{^P/_x\}. \tag{1}$$

Note that if $\mathcal{R}$ is also reflexive, then this implies

$$\overline{x} \cup \{x\} \vdash M\ \mathcal{R}\ N \wedge L \in \Lambda_\oplus(\overline{x}) \Rightarrow \overline{x} \vdash M\{^L/_x\}\ \mathcal{R}\ N\{^L/_x\}. \tag{2}$$

We say that $\mathcal{R}$ is *closed under term-substitution* if it satisfies (2). Because of the way the open extension of $\sim$ and $\lesssim$ are defined, they are closed under term-substitution.

$$\frac{\overline{x} \vdash x \; \mathcal{R} \; M}{\overline{x} \vdash x \; \mathcal{R}^H \; M} \; (\mathsf{How1}) \qquad \frac{\overline{x} \cup \{x\} \vdash M \; \mathcal{R}^H \; L \qquad \overline{x} \vdash \lambda x.L \; \mathcal{R} \; N \qquad x \notin \overline{x}}{\overline{x} \vdash \lambda x.M \; \mathcal{R}^H \; N} \; (\mathsf{How2})$$

$$\frac{\overline{x} \vdash M \; \mathcal{R}^H \; P \qquad \overline{x} \vdash N \; \mathcal{R}^H \; Q \qquad \overline{x} \vdash PQ \; \mathcal{R} \; L}{\overline{x} \vdash MN \; \mathcal{R}^H \; L} \; (\mathsf{How3})$$

$$\frac{\overline{x} \vdash M \; \mathcal{R}^H \; P \qquad \overline{x} \vdash N \; \mathcal{R}^H \; Q \qquad \overline{x} \vdash P \oplus Q \; \mathcal{R} \; L}{\overline{x} \vdash M \oplus N \; \mathcal{R}^H \; L} \; (\mathsf{How4})$$

Figure 3: Howe's Lifting for $\Lambda_\oplus$.

Unfortunately, directly prove $\lesssim$ to enjoy such *substitutivity* property is hard. We will thus proceed indirectly by defining, starting from $\lesssim$, a new relation $\lesssim^H$, called the *Howe's lifting* of $\lesssim$, that has such property by construction and that can be proved equal to $\lesssim$.

Actually, the Howe's lifting of any $\Lambda_\oplus$-relation $\mathcal{R}$ is the relation $\mathcal{R}^H$ defined by the rules in Figure 3. The reader familiar with Howe's method should have a sense of *déjà vu* here: indeed, this is *precisely* the same definition one finds in the realm of *nondeterministic* λ-calculi. The language of terms, after all, is the same. This facilitates the first part of the proof. Indeed, one already knows that if $\mathcal{R}$ is a preorder, then $\mathcal{R}^H$ is compatible and includes $\mathcal{R}$, since all these properties are already known (see, e.g. [38]) and only depend on the shape of terms and not on their operational semantics.

Lemma 3.10 *If $\mathcal{R}$ is reflexive, then $\mathcal{R}^H$ is compatible.*

Proof. We need to prove that (Com1), (Com2), (Com3), and (Com4) hold for $\mathcal{R}^H$:

- Proving (Com1) means to show:

$$\forall \overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X}), x \in \overline{x} \Rightarrow \overline{x} \vdash x \; \mathcal{R}^H \; x.$$

  Since $\mathcal{R}$ is reflexive, $\forall \overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X})$, $x \in \overline{x} \Rightarrow \overline{x} \vdash x \; \mathcal{R} \; x$. Thus, by (How1), we conclude $\overline{x} \vdash x \; \mathcal{R}^H \; x$. Formally,

$$\frac{\overline{\overline{x} \vdash x \; \mathcal{R} \; x}}{\overline{x} \vdash x \; \mathcal{R}^H \; x} \; (\mathsf{How1})$$

- Proving (Com2) means to show: $\forall \overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X})$, $\forall x \in \mathsf{X} - \overline{x}$, $\forall M, N \in \Lambda_\oplus(\overline{x} \cup \{x\})$,

$$\overline{x} \cup \{x\} \vdash M \; \mathcal{R}^H \; N \Rightarrow \overline{x} \vdash \lambda x.M \; \mathcal{R}^H \; \lambda x.N.$$

  Since $\mathcal{R}$ is reflexive, we get $\overline{x} \vdash \lambda x.N \; \mathcal{R} \; \lambda x.N$. Moreover, we have $\overline{x} \cup \{x\} \vdash M \; \mathcal{R}^H \; N$ by hypothesis. Thus, by (How2), we conclude $\overline{x} \vdash \lambda x.M \; \mathcal{R}^H \; \lambda x.N$ holds. Formally,

$$\frac{\overline{x} \cup \{x\} \vdash M \; \mathcal{R}^H \; N \qquad \overline{\overline{x} \vdash \lambda x.N \; \mathcal{R} \; \lambda x.N} \qquad x \notin \overline{x}}{\overline{x} \vdash \lambda x.M \; \mathcal{R}^H \; \lambda x.N} \; (\mathsf{How2})$$

- Proving (Com3) means to show: $\forall \overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X})$, $\forall M, N, L, P \in \Lambda_\oplus(\overline{x})$,

$$\overline{x} \vdash M \; \mathcal{R}^H \; N \wedge \overline{x} \vdash L \; \mathcal{R}^H \; P \Rightarrow \overline{x} \vdash ML \; \mathcal{R}^H \; NP.$$

  Since $\mathcal{R}$ is reflexive, we get $\overline{x} \vdash NP \; \mathcal{R} \; NP$. Moreover, we have $\overline{x} \vdash M \; \mathcal{R}^H \; N$ and $\overline{x} \vdash L \; \mathcal{R}^H \; P$ by hypothesis. Thus, by (How3), we conclude $\overline{x} \vdash ML \; \mathcal{R}^H \; NP$ holds. Formally,

$$\frac{\overline{x} \vdash M \; \mathcal{R}^H \; N \qquad \overline{x} \vdash L \; \mathcal{R}^H \; P \qquad \overline{\overline{x} \vdash NP \; \mathcal{R} \; NP}}{\overline{x} \vdash ML \; \mathcal{R}^H \; NP} \; (\mathsf{How3})$$

- Proving (Com4) means to show: $\forall \overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X})$, $\forall M, N, L, P \in \Lambda_{\oplus}(\overline{x})$,

$$\overline{x} \vdash M \; \mathcal{R}^H \; N \wedge \overline{x} \vdash L \; \mathcal{R}^H \; P \Rightarrow \overline{x} \vdash M \oplus L \; \mathcal{R}^H \; N \oplus P.$$

  Since $\mathcal{R}$ is reflexive, we get $\overline{x} \vdash N \oplus P \; \mathcal{R} \; N \oplus P$. Moreover, we have $\overline{x} \vdash M \; \mathcal{R}^H \; N$ and $\overline{x} \vdash L \; \mathcal{R}^H \; P$ by hypothesis. Thus, by (How4), we conclude $\overline{x} \vdash M \oplus L \; \mathcal{R}^H \; N \oplus P$ holds. Formally,

$$\dfrac{\overline{x} \vdash M \; \mathcal{R}^H \; N \qquad \overline{x} \vdash L \; \mathcal{R}^H \; P \qquad \overline{\overline{x} \vdash N \oplus P \; \mathcal{R} \; N \oplus P}}{\overline{x} \vdash M \oplus L \; \mathcal{R}^H \; N \oplus P} \; (\mathsf{How4})$$

This concludes the proof. □

Lemma 3.11 *If $\mathcal{R}$ is transitive, then $\overline{x} \vdash M \; \mathcal{R}^H \; N$ and $\overline{x} \vdash N \; \mathcal{R} \; L$ imply $\overline{x} \vdash M \; \mathcal{R}^H \; L$.*

Proof. We prove the statement by inspection on the last rule used in the derivation of $\overline{x} \vdash M \; \mathcal{R}^H \; N$, thus on the structure of M.

- If M is a variable, say $x \in \overline{x}$, then $\overline{x} \vdash x \; \mathcal{R}^H \; N$ holds by hypothesis. The last rule used has to be (How1). Thus, we get $\overline{x} \vdash x \; \mathcal{R} \; N$ as additional hypothesis. By transitivity of $\mathcal{R}$, from $\overline{x} \vdash x \; \mathcal{R} \; N$ and $\overline{x} \vdash N \; \mathcal{R} \; L$ we deduce $\overline{x} \vdash x \; \mathcal{R} \; L$. We conclude by (How1) on the latter, obtaining $\overline{x} \vdash x \; \mathcal{R}^H \; L$, i.e. $\overline{x} \vdash M \; \mathcal{R}^H \; L$. Formally,

$$\dfrac{\dfrac{\overline{x} \vdash x \; \mathcal{R} \; N \qquad \overline{x} \vdash N \; \mathcal{R} \; L}{\overline{x} \vdash x \; \mathcal{R} \; L} \; (\mathsf{How1})}{\overline{x} \vdash x \; \mathcal{R}^H \; L}$$

- If M is a λ-abstraction, say $\lambda x.Q$, then $\overline{x} \vdash \lambda x.Q \; \mathcal{R}^H \; N$ holds by hypothesis. The last rule used has to be (How2). Thus, we get $\overline{x} \cup \{x\} \vdash Q \; \mathcal{R}^H \; P$ and $\overline{x} \vdash \lambda x.P \; \mathcal{R} \; N$ as additional hypothesis. By transitivity of $\mathcal{R}$, from $\overline{x} \vdash \lambda x.P \; \mathcal{R} \; N$ and $\overline{x} \vdash N \; \mathcal{R} \; L$ we deduce $\overline{x} \vdash \lambda x.P \; \mathcal{R} \; L$. We conclude by (How2) on $\overline{x} \cup \{x\} \vdash Q \; \mathcal{R}^H \; P$ and the latter, obtaining $\overline{x} \vdash \lambda x.Q \; \mathcal{R}^H \; L$, i.e. $\overline{x} \vdash M \; \mathcal{R}^H \; L$. Formally,

$$\dfrac{\overline{x} \cup \{x\} \vdash Q \; \mathcal{R}^H \; P \qquad \dfrac{\overline{x} \vdash \lambda x.P \; \mathcal{R} \; N \qquad \overline{x} \vdash N \; \mathcal{R} \; L}{\overline{x} \vdash \lambda x.P \; \mathcal{R} \; L}}{\overline{x} \vdash \lambda x.Q \; \mathcal{R}^H \; L} \; (\mathsf{How2})$$

- If M is an application, say RS, then $\overline{x} \vdash RS \; \mathcal{R}^H \; N$ holds by hypothesis. The last rule used has to be (How3). Thus, we get $\overline{x} \vdash R \; \mathcal{R}^H \; P$, $\overline{x} \vdash S \; \mathcal{R}^H \; Q$ and $\overline{x} \vdash PQ \; \mathcal{R} \; N$ as additional hypothesis. By transitivity of $\mathcal{R}$, from $\overline{x} \vdash PQ \; \mathcal{R} \; N$ and $\overline{x} \vdash N \; \mathcal{R} \; L$ we deduce $\overline{x} \vdash PQ \; \mathcal{R} \; L$. We conclude by (How3) on $\overline{x} \vdash R \; \mathcal{R}^H \; P$, $\overline{x} \vdash S \; \mathcal{R}^H \; Q$ and the latter, obtaining $\overline{x} \vdash RS \; \mathcal{R}^H \; L$, i.e. $\overline{x} \vdash M \; \mathcal{R}^H \; L$. Formally,

$$\dfrac{\overline{x} \vdash R \; \mathcal{R}^H \; P \qquad \overline{x} \vdash S \; \mathcal{R}^H \; Q \qquad \dfrac{\overline{x} \vdash PQ \; \mathcal{R} \; N \qquad \overline{x} \vdash N \; \mathcal{R} \; L}{\overline{x} \vdash PQ \; \mathcal{R} \; L}}{\overline{x} \vdash RS \; \mathcal{R}^H \; L} \; (\mathsf{How3})$$

- If M is a probabilistic sum, say $R \oplus S$, then $\overline{x} \vdash R \oplus S \; \mathcal{R}^H \; N$ holds by hypothesis. The last rule used has to be (How4). Thus, we get $\overline{x} \vdash R \; \mathcal{R}^H \; P$, $\overline{x} \vdash S \; \mathcal{R}^H \; Q$ and $\overline{x} \vdash P \oplus Q \; \mathcal{R} \; N$ as additional hypothesis. By transitivity of $\mathcal{R}$, from $\overline{x} \vdash P \oplus Q \; \mathcal{R} \; N$ and $\overline{x} \vdash N \; \mathcal{R} \; L$ we deduce $\overline{x} \vdash P \oplus Q \; \mathcal{R} \; L$. We conclude by (How4) on $\overline{x} \vdash R \; \mathcal{R}^H \; P$, $\overline{x} \vdash S \; \mathcal{R}^H \; Q$ and the latter, obtaining $\overline{x} \vdash R \oplus S \; \mathcal{R}^H \; L$, i.e. $\overline{x} \vdash M \; \mathcal{R}^H \; L$. Formally,

$$\dfrac{\overline{x} \vdash R \; \mathcal{R}^H \; P \qquad \overline{x} \vdash S \; \mathcal{R}^H \; Q \qquad \dfrac{\overline{x} \vdash P \oplus Q \; \mathcal{R} \; N \qquad \overline{x} \vdash N \; \mathcal{R} \; L}{\overline{x} \vdash P \oplus Q \; \mathcal{R} \; L}}{\overline{x} \vdash R \oplus S \; \mathcal{R}^H \; L} \; (\mathsf{How4})$$

This concludes the proof. □

Lemma 3.12 *If $\mathcal{R}$ is reflexive, then $\overline{x} \vdash M\ \mathcal{R}\ N$ implies $\overline{x} \vdash M\ \mathcal{R}^H\ N$.*

Proof. We will prove it by inspection on the structure of M.

- If M is a variable, say $x \in \overline{x}$, then $\overline{x} \vdash x\ \mathcal{R}\ N$ holds by hypothesis. We conclude by (How1) on the latter, obtaining $\overline{x} \vdash x\ \mathcal{R}^H\ N$, i.e. $\overline{x} \vdash M\ \mathcal{R}^H\ N$. Formally,

$$\frac{\overline{x} \vdash x\ \mathcal{R}\ N}{\overline{x} \vdash x\ \mathcal{R}^H\ N}\ (\mathsf{How1})$$

- If M is a λ-abstraction, say $\lambda x.Q$, then $\overline{x} \vdash \lambda x.Q\ \mathcal{R}\ N$ holds by hypothesis. Moreover, since $\mathcal{R}$ reflexive implies $\mathcal{R}^H$ compatible, $\mathcal{R}^H$ is reflexive too. Then, from $\overline{x} \cup \{x\} \vdash Q\ \mathcal{R}^H\ Q$ and $\overline{x} \vdash \lambda x.Q\ \mathcal{R}\ N$ we conclude, by (How2), $\overline{x} \vdash \lambda x.Q\ \mathcal{R}^H\ N$, i.e. $\overline{x} \vdash M\ \mathcal{R}^H\ N$. Formally,

$$\frac{\overline{x} \cup \{x\} \vdash Q\ \mathcal{R}^H\ Q \qquad \overline{x} \vdash \lambda x.Q\ \mathcal{R}\ N \qquad x \notin \overline{x}}{\overline{x} \vdash \lambda x.Q\ \mathcal{R}^H\ N}\ (\mathsf{How2})$$

- If M is an application, say LP, then $\overline{x} \vdash LP\ \mathcal{R}\ N$ holds by hypothesis. By reflexivity of $\mathcal{R}$, hence that of $\mathcal{R}^H$ too, we get $\overline{x} \vdash L\ \mathcal{R}^H\ L$ and $\overline{x} \vdash P\ \mathcal{R}^H\ P$. Then, from the latter and $\overline{x} \vdash LP\ \mathcal{R}\ N$ we conclude, by (How3), $\overline{x} \vdash LP\ \mathcal{R}^H\ N$, i.e. $\overline{x} \vdash M\ \mathcal{R}^H\ N$. Formally,

$$\frac{\overline{x} \vdash L\ \mathcal{R}^H\ L \qquad \overline{x} \vdash P\ \mathcal{R}^H\ P \qquad \overline{x} \vdash LP\ \mathcal{R}\ N}{\overline{x} \vdash LP\ \mathcal{R}^H\ N}\ (\mathsf{How3})$$

- If M is a probabilistic sum, say $L \oplus P$, then $\overline{x} \vdash L \oplus P\ \mathcal{R}\ N$ holds by hypothesis. By reflexivity of $\mathcal{R}$, hence that of $\mathcal{R}^H$ too, we get $\overline{x} \vdash L\ \mathcal{R}^H\ L$ and $\overline{x} \vdash P\ \mathcal{R}^H\ P$. Then, from the latter and $\overline{x} \vdash L \oplus P\ \mathcal{R}\ N$ we conclude, by (How4), $\overline{x} \vdash L \oplus P\ \mathcal{R}^H\ N$, i.e. $\overline{x} \vdash M\ \mathcal{R}^H\ N$. Formally,

$$\frac{\overline{x} \vdash L\ \mathcal{R}^H\ L \qquad \overline{x} \vdash P\ \mathcal{R}^H\ P \qquad \overline{x} \vdash L \oplus P\ \mathcal{R}\ N}{\overline{x} \vdash L \oplus P\ \mathcal{R}^H\ N}\ (\mathsf{How4})$$

This concludes the proof. □

Moreover, if $\mathcal{R}$ is a preorder and closed under term-substitution, then its lifted relation $\mathcal{R}^H$ is substitutive. Then, reflexivity of $\mathcal{R}$ implies compatibility of $\mathcal{R}^H$ by Lemma 3.10. It follows $\mathcal{R}^H$ reflexive too, hence closed under term-substitution.

Lemma 3.13 *If $\mathcal{R}$ is reflexive, transitive and closed under term-substitution, then $\mathcal{R}^H$ is (term) substitutive and hence also closed under term-substitution.*

Proof. We show that, for all $\overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X})$, $x \in \mathsf{X} - \overline{x}$, M, $N \in \Lambda_{\oplus}(\overline{x} \cup \{x\})$ and L, $P \in \Lambda_{\oplus}(\overline{x})$,

$$\overline{x} \cup \{x\} \vdash M\ \mathcal{R}^H\ N \wedge \overline{x} \vdash L\ \mathcal{R}^H\ P \Rightarrow \overline{x} \vdash M\{^L/_x\}\ \mathcal{R}^H\ N\{^P/_x\}.$$

We prove the latter by induction on the derivation of $\overline{x} \cup \{x\} \vdash M\ \mathcal{R}^H\ N$, thus on the structure of M.

- If M is a variable, then either $M = x$ or $M \in \overline{x}$. In the latter case, suppose $M = y$. Then, by hypothesis, $\overline{x} \cup \{x\} \vdash y\ \mathcal{R}^H\ N$ holds and the only way to deduce it is by rule (How1) from $\overline{x} \cup \{x\} \vdash y\ \mathcal{R}\ N$. Hence, by the fact $\mathcal{R}$ is closed under term-substitution and $P \in \Lambda_{\oplus}(\overline{x})$, we obtain $\overline{x} \vdash y\{^P/_x\}\ \mathcal{R}\ N\{^P/_x\}$ which is equivalent to $\overline{x} \vdash y\ \mathcal{R}\ N\{^P/_x\}$. Finally, by Lemma 3.12, we conclude $\overline{x} \vdash y\ \mathcal{R}^H\ N\{^P/_x\}$ which is equivalent to $\overline{x} \vdash y\{^L/_x\}\ \mathcal{R}^H\ N\{^P/_x\}$, i.e. $\overline{x} \vdash M\{^L/_x\}\ \mathcal{R}^H\ N\{^P/_x\}$ holds. Otherwise, $M = x$ and $\overline{x} \cup \{x\} \vdash x\ \mathcal{R}^H\ N$ holds. The only way to deduce the latter is by the rule (How1) from $\overline{x} \cup \{x\} \vdash x\ \mathcal{R}\ N$. Hence, by the fact $\mathcal{R}$ is closed under term-substitution and $P \in \Lambda_{\oplus}(\overline{x})$, we obtain $\overline{x} \vdash x\{^P/_x\}\ \mathcal{R}\ N\{^P/_x\}$ which is equivalent to $\overline{x} \vdash P\ \mathcal{R}\ N\{^P/_x\}$. By Lemma 3.11, we deduce the following:

$$\frac{\overline{x} \vdash L\ \mathcal{R}^H\ P \qquad \overline{x} \vdash P\ \mathcal{R}\ N\{^P/_x\}}{\overline{x} \vdash L\ \mathcal{R}^H\ N\{^P/_x\}}$$

  which is equivalent to $\overline{x} \vdash x\{^L/_x\}\ \mathcal{R}^H\ N\{^P/_x\}$. Thus, $\overline{x} \vdash M\{^L/_x\}\ \mathcal{R}^H\ N\{^P/_x\}$ holds.

- If M is a λ-abstraction, say $\lambda y.Q$, then $\overline{x} \cup \{x\} \vdash \lambda y.Q \; \mathcal{R}^H \; N$ holds by hypothesis. The only way to deduce the latter is by rule (How2) as follows:

$$\frac{\overline{x} \cup \{x, y\} \vdash Q \; \mathcal{R}^H \; R \qquad \overline{x} \cup \{x\} \vdash \lambda y.R \; \mathcal{R} \; N \qquad x, y \notin \overline{x}}{\overline{x} \cup \{x\} \vdash \lambda y.Q \; \mathcal{R}^H \; N} \; (\mathsf{How2})$$

  Let us denote $\overline{y} = \overline{x} \cup \{y\}$. Then, by induction hypothesis on $\overline{y} \cup \{x\} \vdash Q \; \mathcal{R}^H \; R$, we get $\overline{y} \vdash Q\{^L/_x\} \; \mathcal{R}^H \; R\{^P/_x\}$. Moreover, by the fact $\mathcal{R}$ is closed under term-substitution and $P \in \Lambda_\oplus(\overline{x})$, we obtain that $\overline{x} \vdash (\lambda y.R)\{^P/_x\} \; \mathcal{R} \; N\{^P/_x\}$ holds, i.e. $\overline{x} \vdash \lambda y.R\{^P/_x\} \; \mathcal{R} \; N\{^P/_x\}$. By (How2), we deduce the following:

$$\frac{\overline{x} \cup \{y\} \vdash Q\{^L/_x\} \; \mathcal{R}^H \; R\{^P/_x\} \qquad \overline{x} \vdash \lambda y.R\{^P/_x\} \; \mathcal{R} \; N\{^P/_x\} \qquad y \notin \overline{x}}{\overline{x} \vdash \lambda y.Q\{^L/_x\} \; \mathcal{R}^H \; N\{^P/_x\}} \; (\mathsf{How2})$$

  which is equivalent to $\overline{x} \vdash (\lambda y.Q)\{^L/_x\} \; \mathcal{R}^H \; N\{^P/_x\}$. Thus, $\overline{x} \vdash M\{^L/_x\} \; \mathcal{R}^H \; N\{^P/_x\}$ holds.
- If M is an application, say QR, then $\overline{x} \cup \{x\} \vdash QR \; \mathcal{R}^H \; N$ holds by hypothesis. The only way to deduce the latter is by rule (How3) as follows:

$$\frac{\overline{x} \cup \{x\} \vdash Q \; \mathcal{R}^H \; Q' \qquad \overline{x} \cup \{x\} \vdash R \; \mathcal{R}^H \; R' \qquad \overline{x} \cup \{x\} \vdash Q'R' \; \mathcal{R} \; N}{\overline{x} \cup \{x\} \vdash QR \; \mathcal{R}^H \; N} \; (\mathsf{How3})$$

  By induction hypothesis on $\overline{x} \cup \{x\} \vdash Q \; \mathcal{R}^H \; Q'$ and $\overline{x} \cup \{x\} \vdash R \; \mathcal{R}^H \; R'$, we get $\overline{x} \vdash Q\{^L/_x\} \; \mathcal{R}^H \; Q'\{^P/_x\}$ and $\overline{x} \vdash R\{^L/_x\} \; \mathcal{R}^H \; R'\{^P/_x\}$. Moreover, by the fact $\mathcal{R}$ is closed under term-substitution and $P \in \Lambda_\oplus(\overline{x})$, we obtain that $\overline{x} \vdash (Q'R')\{^P/_x\} \; \mathcal{R} \; N\{^P/_x\}$ holds, i.e. $\overline{x} \vdash Q'\{^P/_x\}R'\{^P/_x\} \; \mathcal{R} \; N\{^P/_x\}$. By (How3), we deduce the following:

$$\frac{\overline{x} \vdash Q\{^L/_x\} \; \mathcal{R}^H \; Q'\{^P/_x\} \qquad \overline{x} \vdash R\{^L/_x\} \; \mathcal{R}^H \; R'\{^P/_x\} \qquad \overline{x} \vdash Q'\{^P/_x\}R'\{^P/_x\} \; \mathcal{R} \; N\{^P/_x\}}{\overline{x} \vdash Q\{^L/_x\}R\{^L/_x\} \; \mathcal{R}^H \; N\{^P/_x\}} \; (\mathsf{How3})$$

  which is equivalent to $\overline{x} \vdash (QR)\{^L/_x\} \; \mathcal{R}^H \; N\{^P/_x\}$. Thus, $\overline{x} \vdash M\{^L/_x\} \; \mathcal{R}^H \; N\{^P/_x\}$ holds.
- If M is a probabilistic sum, say $Q \oplus R$, then $\overline{x} \cup \{x\} \vdash Q \oplus R \; \mathcal{R}^H \; N$ holds by hypothesis. The only way to deduce the latter is by rule (How4) as follows:

$$\frac{\overline{x} \cup \{x\} \vdash Q \; \mathcal{R}^H \; Q' \qquad \overline{x} \cup \{x\} \vdash R \; \mathcal{R}^H \; R' \qquad \overline{x} \cup \{x\} \vdash Q' \oplus R' \; \mathcal{R} \; N}{\overline{x} \cup \{x\} \vdash Q \oplus R \; \mathcal{R}^H \; N} \; (\mathsf{How4})$$

  By induction hypothesis on $\overline{x} \cup \{x\} \vdash Q \; \mathcal{R}^H \; Q'$ and $\overline{x} \cup \{x\} \vdash R \; \mathcal{R}^H \; R'$, we get $\overline{x} \vdash Q\{^L/_x\} \; \mathcal{R}^H \; Q'\{^P/_x\}$ and $\overline{x} \vdash R\{^L/_x\} \; \mathcal{R}^H \; R'\{^P/_x\}$. Moreover, by the fact $\mathcal{R}$ is closed under term-substitution and $P \in \Lambda_\oplus(\overline{x})$, we obtain that $\overline{x} \vdash (Q' \oplus R')\{^P/_x\} \; \mathcal{R} \; N\{^P/_x\}$, i.e. $\overline{x} \vdash Q'\{^P/_x\} \oplus R'\{^P/_x\} \; \mathcal{R} \; N\{^P/_x\}$. By (How4), we conclude the following:

$$\frac{\overline{x} \vdash Q\{^L/_x\} \; \mathcal{R}^H \; Q'\{^P/_x\} \qquad \overline{x} \vdash R\{^L/_x\} \; \mathcal{R}^H \; R'\{^P/_x\} \qquad \overline{x} \vdash Q'\{^P/_x\} \oplus R'\{^P/_x\} \; \mathcal{R} \; N\{^P/_x\}}{\overline{x} \vdash Q\{^L/_x\} \oplus R\{^L/_x\} \; \mathcal{R}^H \; N\{^P/_x\}} \; (\mathsf{How4})$$

  which is equivalent to $\overline{x} \vdash (Q \oplus R)\{^L/_x\} \; \mathcal{R}^H \; N\{^P/_x\}$. Thus, $\overline{x} \vdash M\{^L/_x\} \; \mathcal{R}^H \; N\{^P/_x\}$ holds.

This concludes the proof. □

Something is missing, however, before we can conclude that $\lesssim^H$ is a precongruence, namely transitivity. We also follow Howe here building the transitive closure of a $\Lambda_\oplus$-relation $\mathcal{R}$ as the relation $\mathcal{R}^+$ defined by the rules in Figure 4. Then, it is easy to prove $\mathcal{R}^+$ of being compatible and closed under term-substitution if $\mathcal{R}$ is.

Lemma 3.14 *If $\mathcal{R}$ is compatible, then so is $\mathcal{R}^+$.*

Proof. We need to prove that (Com1), (Com2), (Com3), and (Com4) hold for $\mathcal{R}^+$:

$$\frac{\overline{x} \vdash M \mathcal{R} N}{\overline{x} \vdash M \mathcal{R}^+ N} \ (\mathsf{TC1})$$

$$\frac{\overline{x} \vdash M \mathcal{R}^+ N \qquad \overline{x} \vdash N \mathcal{R}^+ L}{\overline{x} \vdash M \mathcal{R}^+ L} \ (\mathsf{TC2})$$

Figure 4: Transitive Closure for $\Lambda_\oplus$.

- Proving (Com1) means to show:

$$\forall \overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X}), x \in \overline{x} \Rightarrow \overline{x} \vdash x \ \mathcal{R} \ x.$$

  Since $\mathcal{R}$ is compatible, therefore reflexive, $\overline{x} \vdash x \ \mathcal{R} \ x$ holds. Hence $\overline{x} \vdash x \ \mathcal{R}^+ \ x$ follows by (TC1).
- Proving (Com2) means to show: $\forall \overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X})$, $\forall x \in \mathsf{X} - \overline{x}$, $\forall M, N \in \Lambda_\oplus(\overline{x} \cup \{x\})$,

$$\overline{x} \cup \{x\} \vdash M \ \mathcal{R}^+ \ N \Rightarrow \overline{x} \vdash \lambda x.M \ \mathcal{R}^+ \ \lambda x.N.$$

  We prove it by induction on the derivation of $\overline{x} \cup \{x\} \vdash M \ \mathcal{R}^+ \ N$, looking at the last rule used. The base case has (TC1) as last rule: thus, $\overline{x} \cup \{x\} \vdash M \ \mathcal{R} \ N$ holds. Then, since $\mathcal{R}$ is compatible, it follows $\overline{x} \vdash \lambda x.M \ \mathcal{R} \ \lambda x.N$. We conclude applying (TC1) on the latter, obtaining $\overline{x} \vdash \lambda x.M \ \mathcal{R}^+ \ \lambda x.N$. Otherwise, if (TC2) is the last rule used, we get that, for some $L \in \Lambda_\oplus(\overline{x} \cup \{x\})$, $\overline{x} \cup \{x\} \vdash M \ \mathcal{R}^+ \ L$ and $\overline{x} \cup \{x\} \vdash L \ \mathcal{R}^+ \ N$ hold. Then, by induction hypothesis on both of them, we have $\overline{x} \vdash \lambda x.M \ \mathcal{R}^+ \ \lambda x.L$ and $\overline{x} \vdash \lambda x.L \ \mathcal{R}^+ \ \lambda x.N$. We conclude applying (TC2) on the latter two, obtaining $\overline{x} \vdash \lambda x.M \ \mathcal{R}^+ \ \lambda x.N$.
- Proving (Com3) means to show: $\forall \overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X})$, $\forall M, N, L, P \in \Lambda_\oplus(\overline{x})$,

$$\overline{x} \vdash M \ \mathcal{R}^+ \ N \ \wedge \ \overline{x} \vdash L \ \mathcal{R}^+ \ P \Rightarrow \overline{x} \vdash ML \ \mathcal{R}^+ \ NP.$$

  Firstly, we prove the following two characterizations:

$$\forall M, N, L, P \in \Lambda_\oplus(\overline{x}). \ \overline{x} \vdash M \ \mathcal{R}^+ \ N \ \wedge \ \overline{x} \vdash L \ \mathcal{R} \ P \Rightarrow \overline{x} \vdash ML \ \mathcal{R}^+ \ NP, \tag{3}$$

$$\forall M, N, L, P \in \Lambda_\oplus(\overline{x}). \ \overline{x} \vdash M \ \mathcal{R} \ N \ \wedge \ \overline{x} \vdash L \ \mathcal{R}^+ \ P \Rightarrow \overline{x} \vdash ML \ \mathcal{R}^+ \ NP. \tag{4}$$

  In particular, we only prove (3) in details, since (4) is similarly provable. We prove (3) by induction on the derivation $\overline{x} \vdash M \ \mathcal{R}^+ \ N$, looking at the last rule used. The base case has (TC1) as last rule: we get that $\overline{x} \vdash M \ \mathcal{R} \ N$ holds. Then, using $\mathcal{R}$ compatibility property and $\overline{x} \vdash L \ \mathcal{R} \ P$, it follows $\overline{x} \vdash ML \ \mathcal{R} \ NP$. We conclude applying (TC1) on the latter, obtaining $\overline{x} \vdash ML \ \mathcal{R}^+ \ NP$. Otherwise, if (TC2) is the last rule used, we get that, for some $Q \in \Lambda_\oplus$, $\overline{x} \vdash M \ \mathcal{R}^+ \ Q$ and $\overline{x} \vdash Q \ \mathcal{R}^+ \ N$ hold. Then, by induction hypothesis on $\overline{x} \vdash M \ \mathcal{R}^+ \ Q$ along with $\overline{x} \vdash L \ \mathcal{R} \ P$, we have $\overline{x} \vdash ML \ \mathcal{R}^+ \ QP$. Then, since $\mathcal{R}$ is compatible and so reflexive too, $\overline{x} \vdash P \ \mathcal{R} \ P$ holds. By induction hypothesis on $\overline{x} \vdash Q \ \mathcal{R}^+ \ N$ along with the latter, we get $\overline{x} \vdash QP \ \mathcal{R}^+ \ NP$. We conclude applying (TC2) on $\overline{x} \vdash ML \ \mathcal{R}^+ \ QP$ and $\overline{x} \vdash QP \ \mathcal{R}^+ \ NP$, obtaining $\overline{x} \vdash ML \ \mathcal{R}^+ \ NP$.
  Let us focus on the original (Com3) statement. We prove it by induction on the two derivations $\overline{x} \vdash M \ \mathcal{R}^+ \ N$ and $\overline{x} \vdash L \ \mathcal{R}^+ \ P$, which we name here as π and ρ respectively. Looking at the last rules used, there are four possible cases as four are the combinations that permit to conclude with π and ρ:
  1. (TC1) for both π and ρ;
  2. (TC1) for π and (TC2) for ρ;
  3. (TC2) for π and (TC1) for ρ;
  4. (TC2) for both π and ρ.

  Observe now that the first three cases are addressed by (3) and (4). Hence, it remains to prove the last case, where both derivations are concluded applying (TC2) rule. According to (TC2) rule definition, we get two additional hypothesis from each derivation. In particular, for π, we get that, for some $Q \in \Lambda_\oplus(\overline{x})$, $\overline{x} \vdash M \ \mathcal{R}^+ \ Q$ and $\overline{x} \vdash Q \ \mathcal{R}^+ \ N$ hold. Similarly, for ρ, we get

that, for some $R \in \Lambda_\oplus(\overline{x})$, $\overline{x} \vdash L\ \mathcal{R}^+\ R$ and $\overline{x} \vdash R\ \mathcal{R}^+\ P$ hold. Then, by a double induction hypothesis, firstly on $\overline{x} \vdash M\ \mathcal{R}^+\ Q$, $\overline{x} \vdash L\ \mathcal{R}^+\ R$ and secondly on $\overline{x} \vdash Q\ \mathcal{R}^+\ N$, $\overline{x} \vdash R\ \mathcal{R}^+\ P$, we get $\overline{x} \vdash ML\ \mathcal{R}^+\ QR$ and $\overline{x} \vdash QR\ \mathcal{R}^+\ NP$ respectively. We conclude applying (TC2) on these latter, obtaining $\overline{x} \vdash ML\ \mathcal{R}^+\ NP$.

- Proving (Com4) means to show: $\forall \overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X})$, $\forall M, N, L, P \in \Lambda_\oplus(\overline{x})$,

$$\overline{x} \vdash M\ \mathcal{R}^+\ N\ \wedge\ \overline{x} \vdash L\ \mathcal{R}^+\ P \Rightarrow \overline{x} \vdash M \oplus L\ \mathcal{R}^+\ N \oplus P.$$

  We do not detail the proof since it boils down to that of (Com3), where partial sums play the role of applications.

This concludes the proof. □

Lemma 3.15 *If $\mathcal{R}$ is closed under term-substitution, then so is $\mathcal{R}^+$.*

Proof. We need to prove $\mathcal{R}^+$ of being closed under term-substitution: for all $\overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X})$, $x \in \mathsf{X} - \overline{x}$, $M, N \in \Lambda_\oplus(\overline{x} \cup \{x\})$ and $L, P \in \Lambda_\oplus(\overline{x})$,

$$\overline{x} \cup \{x\} \vdash M\ \mathcal{R}^+\ N \wedge L \in \Lambda_\oplus(\overline{x}) \Rightarrow \overline{x} \vdash M\{{}^L/_x\}\ \mathcal{R}^+\ N\{{}^L/_x\}.$$

We prove the latter by induction on the derivation of $\overline{x} \cup \{x\} \vdash M\ \mathcal{R}^+\ N$, looking at the last rule used. The base case has (TC1) as last rule: we get that $\overline{x} \cup \{x\} \vdash M\ \mathcal{R}\ N$ holds. Then, since $\mathcal{R}$ is closed under term-substitution, it follows $\overline{x} \vdash M\{{}^L/_x\}\ \mathcal{R}\ N\{{}^L/_x\}$. We conclude applying (TC1) on the latter, obtaining $\overline{x} \vdash M\{{}^L/_x\}\ \mathcal{R}^+\ N\{{}^L/_x\}$. Otherwise, if (TC2) is the last rule used, we get that, for some $P \in \Lambda_\oplus(\overline{x} \cup \{x\})$, $\overline{x} \cup \{x\} \vdash M\ \mathcal{R}^+\ P$ and $\overline{x} \cup \{x\} \vdash P\ \mathcal{R}^+\ N$ hold. Then, by induction hypothesis on both of them, we have $\overline{x} \vdash M\{{}^L/_x\}\ \mathcal{R}^+\ P\{{}^L/_x\}$ and $\overline{x} \vdash P\{{}^L/_x\}\ \mathcal{R}^+\ N\{{}^L/_x\}$. We conclude applying (TC2) on the latter two, obtaining $\overline{x} \vdash M\{{}^L/_x\}\ \mathcal{R}^+\ N\{{}^L/_x\}$. □

It is important to note that the transitive closure of an already Howe's lifted relation is a preorder if the starting relation is.

Lemma 3.16 *If a $\Lambda_\oplus$-relation $\mathcal{R}$ is a preorder relation, then so is $(\mathcal{R}^H)^+$.*

Proof. We need to show $(\mathcal{R}^H)^+$ of being reflexive and transitive. Of course, being a transitive closure, $(\mathcal{R}^H)^+$ is a a transitive relation. Moreover, since $\mathcal{R}$ is reflexive, by Lemma 3.10, $\mathcal{R}^H$ is reflexive too because compatible. Then, by Lemma 3.14, so is $(\mathcal{R}^H)^+$. □

This is just the first half of the story: we also need to prove that $(\lesssim^H)^+$ is a simulation. As we already know it is a preorder, the following lemma gives us the missing bit:

Lemma 3.17 (Key Lemma) *If $M \lesssim^H N$, then for every $X \subseteq \Lambda_\oplus(x)$ it holds that $[\![M]\!](\lambda x.X) \leq [\![N]\!](\lambda x.(\lesssim^H(X)))$.*

The proof of this lemma is delicate and is discussed in the next section. From the lemma, using a standard argument we derive the needed substitutivity results, and ultimately the most important result of this section.

Theorem 3.18 *$\lesssim$ is a precongruence relation for $\Lambda_\oplus$-terms.*

Proof. We prove the result by observing that $(\lesssim^H)^+$ is a precongruence and by showing that $\lesssim = (\lesssim^H)^+$. First of all, Lemma 3.10 and Lemma 3.14 ensure that $(\lesssim^H)^+$ is compatible and Lemma 3.16 tells us that $(\lesssim^H)^+$ is a preorder. As a consequence, $(\lesssim^H)^+$ is a precongruence. Consider now the inclusion $\lesssim \subseteq (\lesssim^H)^+$. By Lemma 3.12 and by definition of transitive closure operator $(\cdot)^+$, it follows that $\lesssim \subseteq (\lesssim^H) \subseteq (\lesssim^H)^+$. We show the converse by proving that $(\lesssim^H)^+$ is included in a relation $\mathcal{R}$ that is a call-by-name probabilistic applicative simulation, therefore contained in the largest one. In particular, since $(\lesssim^H)^+$ is closed under term-substitution (Lemma 3.13 and Lemma 3.15), it suffices to show the latter only on the closed version of terms and cloned values. $\mathcal{R}$ acts like $(\lesssim^H)^+$ on terms, while given two cloned values $\nu x.M$ and $\nu x.N$, $(\nu x.M)\mathcal{R}(\nu x.N)$ iff $M(\lesssim^H)^+N$. Since we already know that $(\lesssim^H)^+$ is a preorder (and thus $\mathcal{R}$ is itself a preorder), all that remain to be checked are the following two points:

- If $M(\lesssim^H)^+N$, then for every $X \subseteq \Lambda_\oplus(x)$ it holds that

$$\mathcal{P}_\oplus(M, \tau, \nu x.X) \leq \mathcal{P}_\oplus(N, \tau, \mathcal{R}(\nu x.X)). \tag{5}$$

  Let us proceed by induction on the structure of the proof of $M(\lesssim^H)^+N$:
  - The base case has (TC1) as last rule: we get that $\emptyset \vdash M \lesssim^H N$ holds. Then, in particular by Lemma 3.17,

$$\begin{aligned}\mathcal{P}_\oplus(M, \tau, \nu x.X) &= [\![M]\!](\lambda x.X)\\ &\leq [\![N]\!](\lambda x.\lesssim^H(X))\\ &\leq [\![N]\!](\lambda x.(\lesssim^H)^+(X))\\ &\leq [\![N]\!](\mathcal{R}(\nu x.X)) = \mathcal{P}_\oplus(N, \tau, \mathcal{R}(\nu x.X)).\end{aligned}$$

  - If (TC2) is the last rule used, we obtain that, for some $P \in \Lambda_\oplus(\emptyset)$, $\emptyset \vdash M\ (\lesssim^H)^+\ P$ and $\emptyset \vdash P\ (\lesssim^H)^+\ N$ hold. Then, by induction hypothesis, we get

$$\begin{aligned}\mathcal{P}_\oplus(M, \tau, X) &\leq \mathcal{P}_\oplus(P, \tau, \mathcal{R}(X)),\\ \mathcal{P}_\oplus(P, \tau, \mathcal{R}(X)) &\leq \mathcal{P}_\oplus(N, \tau, \mathcal{R}(\mathcal{R}(X))).\end{aligned}$$

    But of course $\mathcal{R}(\mathcal{R}(X)) \subseteq \mathcal{R}(X)$, and as a consequence:

$$\mathcal{P}_\oplus(M, \tau, X) \leq \mathcal{P}_\oplus(N, \tau, \mathcal{R}(X))$$

    and (5) is satisfied.
- If $M(\lesssim^H)^+N$, then for every $L \in \Lambda_\oplus(\emptyset)$ and for every $X \subseteq \Lambda_\oplus(\emptyset)$ it holds that

$$\mathcal{P}_\oplus(\nu x.M, L, X) \leq \mathcal{P}_\oplus(\nu x.N, L, \mathcal{R}(X)).$$

  But if $M(\lesssim^H)^+N$, then $M\{^L/_x\}(\lesssim^H)^+N\{^L/_x\}$. This is means that whenever $M\{^L/_x\} \in X$, $N\{^L/_x\} \in \lesssim^H(X) \subseteq (\lesssim^H)^+(X)$ and ultimately

$$\begin{aligned}\mathcal{P}_\oplus(\nu x.M, L, X) &= 1\\ &= \mathcal{P}_\oplus(\nu x.N, L, (\lesssim^H)^+(X))\\ &= \mathcal{P}_\oplus(\nu x.N, L, \mathcal{R}(X)).\end{aligned}$$

  If $M\{^L/_x\} \notin X$, on the other hand,

$$\mathcal{P}_\oplus(\nu x.M, L, X) = 0 \leq \mathcal{P}_\oplus(\nu x.N, L, \mathcal{R}(X)).$$

This concludes the proof. □

Corollary 3.19 *$\sim$ is a congruence relation for $\Lambda_\oplus$-terms.*

Proof. $\sim$ is an equivalence relation by definition, in particular a symmetric relation. Since $\sim = \lesssim \cap \lesssim^{op}$ by Proposition 2.13, $\sim$ is also compatible as a consequence of Theorem 3.18. □

3.2 Proof of the Key Lemma

As we have already said, Lemma 3.17 is indeed a crucial step towards showing that probabilistic applicative simulation is a precongruence. Proving the Key Lemma 3.17 turns out to be much more difficult than for deterministic or nondeterministic cases. In particular, the case when M is an application relies on another technical lemma we are now going to give, which itself can be proved by tools from linear programming.

The combinatorial problem we will face while proving the Key Lemma can actually be decontextualized and understood independently. Suppose we have $n = 3$ non-disjoint sets X_1, X_2, X_3

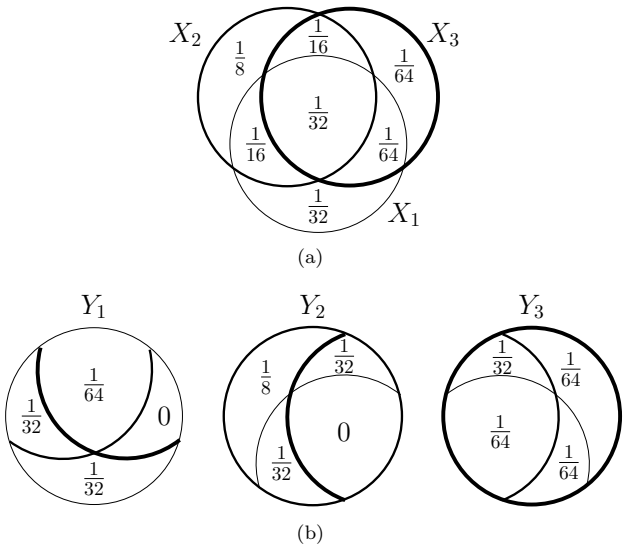


Figure 5: Disentangling Sets

whose elements are labelled with real numbers. As an example, we could be in a situation like the one in Figure 5(a) (where for the sake of simplicity only the labels are indicated). We fix three real numbers $p_1 \stackrel{\text{def}}{=} \frac{5}{64}$, $p_2 \stackrel{\text{def}}{=} \frac{3}{16}$, $p_3 \stackrel{\text{def}}{=} \frac{5}{64}$. It is routine to check that for every $I \subseteq \{1,2,3\}$ it holds that

$$\sum_{i \in I} p_i \leq ||\bigcup_{i \in I} X_i||,$$

where $||X||$ is the sum of the labels of the elements of X. Let us observe that it is of course possible to turn the three sets X_1, X_2, X_3 into three disjoint sets Y_1, Y_2 and Y_3 where each Y_i contains (copies of) the elements of X_i whose labels, however, are obtained by splitting the ones of the original elements. Examples of those sets are in Figure 5(b): if you superpose the three sets, you obtain the Venn diagram we started from. Quite remarkably, however, the examples from Figure 5 have an additional property, namely that for every $i \in \{1,2,3\}$ it holds that $p_i \leq ||Y_i||$. We now show that finding sets satisfying the properties above is always possible, even when n is arbitrary.

Suppose $p_1, \ldots, p_n \in \mathbb{R}_{[0,1]}$, and suppose that for each $I \subseteq \{1, \ldots, n\}$ a real number $r_I \in \mathbb{R}_{[0,1]}$ is defined such that for every such I it holds that $\sum_{i \in I} p_i \leq \sum_{J \cap I \neq \emptyset} r_J \leq 1$. Then $(\{p_i\}_{1 \leq i \leq n}, \{r_I\}_{I \subseteq \{1,\ldots,n\}})$ is said to be a *probability assignment* for $\{1, \ldots, n\}$. Is it always possible to "disentangle" probability assignments? The answer is positive.

The following is a formulation of Max-Flow-Min-Cut Theorem:

Theorem 3.20 (Max-Flow-Min-Cut) *For any flow network, the value of the maximum flow is equal to the capacity of the minimum cut.*

Lemma 3.21 (Disentangling Probability Assignments) *Let* $\mathbf{P} \stackrel{\text{def}}{=} (\{p_i\}_{1 \leq i \leq n}, \{r_I\}_{I \subseteq \{1,\ldots,n\}})$ *be a probability assignment. Then for every nonempty* $I \subseteq \{1, \ldots, n\}$ *and for every* $k \in I$ *there is* $s_{k,I} \in \mathbb{R}_{[0,1]}$ *such that the following conditions all hold:*

1. *for every* I*, it holds that* $\sum_{k \in I} s_{k,I} \leq 1$*;*
2. *for every* $k \in \{1, \ldots, n\}$*, it holds that* $p_k \leq \sum_{k \in I} s_{k,I} \cdot r_I$*.*

Proof. For every probability assignment $\mathbf{P}$, let us define the *flow network of* $\mathbf{P}$ as the digraph $\mathcal{N}_{\mathbf{P}} \stackrel{\text{def}}{=} (\mathcal{V}_{\mathbf{P}}, \mathcal{E}_{\mathbf{P}})$ where:

- $\mathcal{V}_{\mathbf{P}} \stackrel{\text{def}}{=} (\mathcal{P}(\{1, \ldots, n\}) - \emptyset) \cup \{s, t\}$, where s, t are a distinguished source and target, respectively;

- $\mathcal{E}_{\mathbf{P}}$ is composed by three kinds of edges:
  - $(s, \{i\})$ for every $i \in \{1, \ldots, n\}$, with an assigned capacity of p_i;
  - $(I, I \cup \{i\})$, for every nonempty $I \subseteq \{1, \ldots, n\}$ and $i \notin I$, with an assigned capacity of 1;
  - (I, t), for every nonempty $I \subseteq \{1, \ldots, n\}$, with an assigned capacity of r_I.

We prove the following two lemmas on $\mathcal{N}_{\mathbf{P}}$ which together entail the result.

- **Lemma 3.22** *If $\mathcal{N}_{\mathbf{P}}$ admits a flow summing to $\sum_{i \in \{1,\ldots,n\}} p_i$, then the $s_{k,I}$ exist for which conditions 1. and 2. hold.*

  Proof. Let us fix $p \stackrel{\text{def}}{=} \sum_{i \in \{1,\ldots,n\}} p_i$. The idea then is to start with a flow of value p in input to the source s, which by hypothesis is admitted by $\mathcal{N}_{\mathbf{P}}$ and the maximum one can get, and split it into portions going to singleton vertices $\{i\}$, for every $i \in I$, each of value p_i. Afterwards, for every other vertex $I \subseteq \{1, \ldots, n\}$, values of flows on the incoming edges are summed up and then distributed to the outgoing adges as one wishes, thanks to conservation property of the flow. Formally, a flow $f : \mathcal{E}_p \to \mathbb{R}_{[0,1]}$ is turned into a function $\overline{f} : \mathcal{E}_p \to (\mathbb{R}_{[0,1]})^n$ defined as follows:
  - For every $i \in \{1, \ldots, n\}$, $\overline{f}_{(s,\{i\})} \stackrel{\text{def}}{=} (0, \ldots, f_{(s,\{i\})}, \ldots, 0)$, where the only possibly nonnull component is exactly the i-th;
  - For every nonempty $I \subseteq \{1, \ldots, n\}$, as soon as $\overline{f}$ has been defined on all ingoing edges of I, we can define it on all its outgoing ones, by just splitting each component as we want. This is possible, of course, because f is a flow and, as such, ingoing and outgoing values are the same. More formally, let us fix $\overline{f}_{(*,I)} \stackrel{\text{def}}{=} \sum_{K \subseteq \{1,\ldots,n\}} \overline{f}_{(K,I)}$ and indicate with $\overline{f}_{(*,I),k}$ its k-th component. Then, for every $i \notin I$, we set $\overline{f}_{(I,I\cup\{i\})} \stackrel{\text{def}}{=} (q_{1,i} \cdot \overline{f}_{(*,I),1}, \ldots, q_{n,i} \cdot \overline{f}_{(*,I),n})$ where, for every $j \in \{1, \ldots, n\}$, $q_{j,i} \in \mathbb{R}_{[0,1]}$ are such that $\sum_{i \notin I} q_{j,i} \cdot \overline{f}_{(*,I),j} = \overline{f}_{(*,I),j}$ and $\sum_{j=1}^{n} q_{j,i} \cdot \overline{f}_{(*,I),j} = f_{(I,I\cup\{i\})}$. Of course, a similar definition can be given to $\overline{f}_{(I,t)}$, for every nonempty $I \subseteq \{1, \ldots, n\}$.

  Notice that, the way we have just defined $\overline{f}$ guarantees that the sum of all components of $\overline{f}_e$ is always equal to f_e, for every $e \in \mathcal{E}_{\mathbf{P}}$. Now, for every nonempty $I \subseteq \{1, \ldots, n\}$, fix $s_{k,I}$ to be the ratio q_k of $\overline{f}_{(I,t)}$; i.e., the k-th component of $\overline{f}_{(I,t)}$ (or 0 if the first is itself 0). On the one hand, for every nonempty $I \subseteq \{1, \ldots, n\}$, $\sum_{k \in I} s_{k,I}$ is obviously less or equal to 1, hence condition 1. holds. On the other, each component of $\overline{f}$ is itself a flow, since it satisfies the capacity and conservation constraints. Moreover, $\mathcal{N}_{\mathbf{P}}$ is structured in such a way that the k-th component of $\overline{f}_{(I,t)}$ is 0 whenever $k \notin I$. As a consequence, since $\overline{f}$ satisfies the capacity constraint, for every $k \in \{1, \ldots, n\}$,

  $$p_k \leq \sum_{k \in I} s_{k,I} \cdot \overline{f}_{(I,t)} \leq \sum_{k \in I} s_{k,I} \cdot r_I$$

  and so condition 2. holds too. □

- **Lemma 3.23** *$\mathcal{N}_{\mathbf{P}}$ admits a flow summing to $\sum_{i \in I} p_i$.*

  Proof. We prove the result by means of Theorem 3.20. In particular, we just prove that the capacity of any cut must be at least $p \stackrel{\text{def}}{=} \sum_{i \in \{1,\ldots,n\}} p_i$. A cut (S, A) is said to be *degenerate* if there are $I \subseteq \{1, \ldots, n\}$ and $i \in \{1, \ldots, n\}$ such that $I \in S$ and $I \cup \{i\} \in A$. It is easy to verify that every degenerate cut has capacity greater or equal to 1, thus greater or equal to p. As a consequence, we can just concentrate on non-degenerate cuts and prove that all of them have capacity at least p. Given two cuts $C \stackrel{\text{def}}{=} (S, A)$ and $D \stackrel{\text{def}}{=} (T, B)$, we say that $C \leq D$ iff $S \leq T$. Then, given $I \subseteq \{1, \ldots, n\}$, we call I-cut any cut (S, A) such that $\bigcup_{\{i\} \in S} \{i\} = I$. The *canonical* I-cut is the unique I-cut $C_I \stackrel{\text{def}}{=} (S, A)$ such that $S = \{s\} \cup \{J \subseteq \{1, \ldots, n\} \mid J \cap I \neq \emptyset\}$. Please observe that, by definition, C_I is non-degenerate and that the capacity $c(C_I)$ of C_I is at least p, because the forward edges in C_I (those connecting elements of S to those of A) are those going from s to the singletons not in S, plus the edges going from any $J \in S$ to t. The sum of the capacities of such edges are greater or equal to p by hypothesis. We now need to prove the following two lemmas.

  - **Lemma 3.24** *For every non-degenerate I-cuts C, D such that $C > D$, there is a non-degenerate I-cut E such that $C \geq E > D$ and $c(E) \geq c(D)$.*

Proof. Let $C \stackrel{\text{def}}{=} (S, A)$ and $D \stackrel{\text{def}}{=} (T, B)$. Moreover, let J be any element of $S \backslash T$. Then, consider $E \stackrel{\text{def}}{=} (T \cup \{K \subseteq \{1, \ldots, n\} \mid J \subseteq K\}, B \backslash \{K \subseteq \{1, \ldots, n\} \mid J \subseteq K\})$ and verify that E is the cut we are looking for. Indeed, E is non-degenerate because it is obtained from D, which is non-degenerate by hypothesis, by adding to it J and all its supersets. Of course, $E > D$. Moreover, $C \geq E$ holds since $J \in S$ and C is non-degenerate, which implies C contains all supersets of J as well. It is also easy to check that $c(E) \geq c(D)$. In fact, in the process of constructing E from D we do not lose any forward edges coming from s, since J cannot be a singleton with C and D both I-cuts, or any other edge coming from some element of T, since D is non-degenerate. □

- **Lemma 3.25** *For every non-degenerate I-cuts C, D such that $C \geq D$, $c(C) \geq c(D)$.*

  Proof. Let $C \stackrel{\text{def}}{=} (S, A)$ and $D \stackrel{\text{def}}{=} (T, B)$. We prove the result by induction on the $n \stackrel{\text{def}}{=} |S| - |T|$. If $n = 0$, then $C = D$ and the thesis follows. If $n > 0$, then $C > D$ and, by Lemma 3.24, there is a non-degenerate I-cut E such that $C \geq E > D$ and $c(E) \geq c(D)$. By induction hypothesis on C and E, it follows that $c(C) \geq c(E)$. Thus, $c(C) \geq c(D)$. □

The two lemmas above permit to conclude. Indeed, for every non-degenerate cut D, there is of course a I such that D is a I-cut (possibly with I as the empty set). Then, let us consider the canonical C_I. On the one hand, $c(C_I) \geq p$. On the other, since C_I is non-degenerate, $c(D) \geq c(C_I)$ by Lemma 3.25. Hence, $c(D) \geq p$. □

This concludes the main proof. □

In the coming proof of Lemma 3.17 we will widely, and often implicitly, use the following technical Lemmas. We denote with $\nu x.\lesssim(X)$ the set of distinguished values $\{\nu x.M \mid \exists N \in X.\, N \lesssim M\}$.

Lemma 3.26 *For every $X \subseteq \Lambda_{\oplus}(x)$, $\lesssim(\nu x.X) = \nu x.\lesssim(X)$.*

Proof.

$$\begin{aligned} \nu x.M \in \lesssim(\nu x.X) &\Leftrightarrow \exists N \in X.\, \nu x.N \lesssim \nu x.M \\ &\Leftrightarrow \exists N \in X.\, N \lesssim M \\ &\Leftrightarrow \nu x.M \in \nu x.\lesssim(X). \end{aligned}$$

This concludes the proof. □

Lemma 3.27 *If $M \lesssim N$, then for every $X \in \Lambda_{\oplus}(x)$, $[\![M]\!](\lambda x.X) \leq [\![N]\!](\lambda x.\lesssim(X))$.*

Proof. If $M \lesssim N$, then by definition $[\![M]\!](\nu x.X) \leq [\![N]\!](\lesssim(\nu x.X))$. Therefore, by Lemma 3.26, $[\![N]\!](\lesssim(\nu x.X)) \leq [\![N]\!](\nu x.\lesssim(X))$. □

Remark 3.28 *Throughout the following proof we will implicitly use a routine result stating that $M \lesssim N$ implies $[\![M]\!](\lambda x.X) \leq [\![N]\!](\lambda x.\lesssim(X))$, for every $X \subseteq \Lambda_{\oplus}(x)$. The property needed by the latter is precisely the reason why we have formulated $\Lambda_{\oplus}$ as a multisorted labelled Markov chain: $\lesssim(\nu x.X)$ consists of distinguished values only, and is nothing but $\nu x.\lesssim(X)$.*

Proof. [of Lemma 3.17] This is equivalent to proving that if $M \lesssim^H N$, then for every $X \subseteq \Lambda_{\oplus}(x)$ the following implication holds: if $M \Downarrow \mathscr{D}$, then $\mathscr{D}(\lambda x.X) \leq [\![N]\!](\lambda x.(\lesssim^H(X)))$. This is an induction on the structure of the proof of $M \Downarrow \mathscr{D}$.

- If $\mathscr{D} = \emptyset$, then of course $\mathscr{D}(\lambda x.X) = 0 \leq [\![N]\!](\lambda x.Y)$ for every $X, Y \subseteq \Lambda_{\oplus}(x)$.
- If M is a value $\lambda x.L$ and $\mathscr{D}(\lambda x.L) = 1$, then the proof of $M \lesssim^H N$ necessarily ends as follows:

$$\frac{\{x\} \vdash L \lesssim^H P \qquad \emptyset \vdash \lambda x.P \;\lesssim\; N}{\emptyset \vdash \lambda x.L \lesssim^H N}$$

  Let X be any subset of $\Lambda_{\oplus}(x)$. Now, if $L \notin X$, then $\mathscr{D}(\lambda x.X) = 0$ and the inequality trivially holds. If, on the contrary, $L \in X$, then $P \in \lesssim^H(X)$. Consider $\lesssim(P)$, the set of terms that

are in relation with P via $\lesssim$. We have that for every $Q \in \lesssim(P)$, both $\{x\} \vdash L \lesssim^H P$ and $\{x\} \vdash P \lesssim Q$ hold, and as a consequence $\{x\} \vdash L \lesssim^H Q$ does (this is a consequence of a property of $(\cdot)^H$, see [9]). In other words, $\lesssim(P) \subseteq \lesssim^H(X)$. But then, by Lemma 3.27,

$$[\![N]\!](\lambda x.\lesssim^H(X)) \geq [\![N]\!](\lambda x.\lesssim(P)) \geq [\![\lambda x.P]\!](\lambda x.P) = 1.$$

- If M is an application LP, then $M \Downarrow \mathscr{D}$ is obtained as follows:

$$\frac{L \Downarrow \mathscr{F} \qquad \{Q\{^P/_x\} \Downarrow \mathscr{H}_{Q,P}\}_{Q,P}}{LP \Downarrow \sum_Q \mathscr{F}(\lambda x.Q) \cdot \mathscr{H}_{Q,P}}$$

Moreover, the proof of $\emptyset \vdash M \lesssim^H N$ must end as follows:

$$\frac{\emptyset \vdash L \lesssim^H R \qquad \emptyset \vdash P \lesssim^H S \qquad \emptyset \vdash RS \lesssim N}{\emptyset \vdash LP \lesssim^H N}$$

Now, since $L \Downarrow \mathscr{F}$ and $\emptyset \vdash L \lesssim^H R$, by induction hypothesis we get that for every $Y \subseteq \Lambda_\oplus(x)$ it holds that $\mathscr{F}(\lambda x.Y) \leq [\![R]\!](\lambda x.\lesssim^H(Y))$. Let us now take a look at the distribution

$$\mathscr{D} = \sum_Q \mathscr{F}(\lambda x.Q) \cdot \mathscr{H}_{Q,P}.$$

Since $\mathscr{F}$ is a *finite* distribution, the sum above is actually the sum of finitely many summands. Let the support $\mathsf{S}(\mathscr{F})$ of $\mathscr{F}$ be $\{\lambda x.Q_1, \ldots, \lambda x.Q_n\}$. It is now time to put the above into a form that is amenable to treatment by Lemma 3.21. Let us consider the n sets $\lesssim^H(Q_1), \ldots, \lesssim^H(Q_n)$; to each term U in them we can associate the probability $[\![R]\!](\lambda x.U)$. We are then in the scope of Lemma 3.21, since by induction hypothesis we know that for every $Y \subseteq \Lambda_\oplus(x)$,

$$\mathscr{F}(\lambda x.X) \leq [\![R]\!](\lambda x.\lesssim^H(X)).$$

We can then conclude that for every

$$U \in \lesssim^H(\{Q_1, \ldots, Q_n\}) = \bigcup_{1 \leq i \leq n} \lesssim^H(Q_i)$$

there are n real numbers $r_1^{U,R}, \ldots, r_n^{U,R}$ such that:

$$[\![R]\!](\lambda x.U) \geq \sum_{1 \leq i \leq n} r_i^{U,R} \qquad \forall\, U \in \bigcup_{1 \leq i \leq n} \lesssim^H(Q_i);$$
$$\mathscr{F}(\lambda x.Q_i) \leq \sum_{U \in \lesssim^H(Q_i)} r_i^{U,R} \qquad \forall\, 1 \leq i \leq n.$$

So, we can conclude that

$$\begin{aligned}\mathscr{D} &\leq \sum_{1 \leq i \leq n} \left(\sum_{U \in \lesssim^H(Q_i)} r_i^{U,R} \right) \cdot \mathscr{H}_{Q_i,P} \\ &= \sum_{1 \leq i \leq n} \sum_{U \in \lesssim^H(Q_i)} r_i^{U,R} \cdot \mathscr{H}_{Q_i,P}.\end{aligned}$$

Now, whenever $Q_i \lesssim^H U$ and $P \lesssim^H S$, we know that, by Lemma 3.13, $Q_i\{^P/_x\} \lesssim^H U\{^S/_x\}$. We can then apply the inductive hypothesis to the n derivations of $Q_i\{^P/_x\} \Downarrow \mathscr{H}_{Q_i,P}$, obtaining

that, for every $X \subseteq \Lambda_\oplus(x)$,

$$\begin{aligned}
\mathscr{D}(\lambda x.X) &\leq \sum_{1\leq i\leq n}\sum_{U\in\lesssim^H(Q_i)} r_i^{U,R}\cdot[\![U\{S/x\}]\!](\lambda x.\lesssim^H(X))\\
&\leq \sum_{1\leq i\leq n}\sum_{U\in\lesssim^H(\{Q_1,\ldots,Q_n\})} r_i^{U,R}\cdot[\![U\{S/x\}]\!](\lambda x.\lesssim^H(X))\\
&= \sum_{U\in\lesssim^H(\{Q_1,\ldots,Q_n\})}\sum_{1\leq i\leq n} r_i^{U,R}\cdot[\![U\{S/x\}]\!](\lambda x.\lesssim^H(X))\\
&= \sum_{U\in\lesssim^H(\{Q_1,\ldots,Q_n\})}\left(\sum_{1\leq i\leq n} r_i^{U,R}\right)\cdot[\![U\{S/x\}]\!](\lambda x.\lesssim^H(X))\\
&\leq \sum_{U\in\lesssim^H(\{Q_1,\ldots,Q_n\})}[\![R]\!](\lambda x.U)\cdot[\![U\{S/x\}]\!](\lambda x.\lesssim^H(X))\\
&\leq \sum_{U\in\Lambda_\oplus(x)}[\![R]\!](\lambda x.U)\cdot[\![U\{S/x\}]\!](\lambda x.\lesssim^H(X))\\
&= [\![RS]\!](\lambda x.\lesssim^H(X)) \leq [\![N]\!](\lambda x.\lesssim((\lesssim^H)(X)))\\
&\leq [\![N]\!](\lambda x.\lesssim^H(X)),
\end{aligned}$$

which is the thesis.

- If M is a probabilistic sum $L\oplus P$, then $M\Downarrow\mathscr{D}$ is obtained as follows:

$$\frac{L\Downarrow\mathscr{F}\qquad P\Downarrow\mathscr{G}}{L\oplus P\Downarrow\frac{1}{2}\cdot\mathscr{F}+\frac{1}{2}\cdot\mathscr{G}}$$

Moreover, the proof of $\emptyset\vdash M\lesssim^H N$ must end as follows:

$$\frac{\emptyset\vdash L\lesssim^H R\qquad\emptyset\vdash P\lesssim^H S\qquad\emptyset\vdash R\oplus S\ \lesssim\ N}{\emptyset\vdash L\oplus P\lesssim^H N}$$

Now:

- Since $L\Downarrow\mathscr{F}$ and $\emptyset\vdash L\lesssim^H R$, by induction hypothesis we get that for every $Y\subseteq\Lambda_\oplus(x)$ it holds that $\mathscr{F}(\lambda x.Y)\leq[\![R]\!](\lambda x.\lesssim^H(Y))$;
- Similarly, since $P\Downarrow\mathscr{G}$ and $\emptyset\vdash P\lesssim^H S$, by induction hypothesis we get that for every $Y\subseteq\Lambda_\oplus(x)$ it holds that $\mathscr{G}(\lambda x.Y)\leq[\![S]\!](\lambda x.\lesssim^H(Y))$.

Let us now take a look at the distribution

$$\mathscr{D}=\frac{1}{2}\cdot\mathscr{F}+\frac{1}{2}\cdot\mathscr{G}.$$

The idea then is to prove that, for every $X\subseteq\Lambda_\oplus(x)$, it holds $\mathscr{D}(\lambda x.X)\leq[\![R\oplus S]\!](\lambda x.\lesssim^H(X))$. In fact, since $[\![R\oplus S]\!](\lambda x.\lesssim^H(X))\leq[\![N]\!](\lambda x.\lesssim^H(X))$, the latter would imply the thesis $\mathscr{D}(\lambda x.X)\leq[\![N]\!](\lambda x.\lesssim^H(X))$. But by induction hypothesis and Lemma 2.3:

$$\begin{aligned}
\mathscr{D}(\lambda x.X) &= \frac{1}{2}\cdot\mathscr{F}(\lambda x.X)+\frac{1}{2}\cdot\mathscr{G}(\lambda x.X)\\
&\leq \frac{1}{2}\cdot[\![R]\!](\lambda x.\lesssim^H(X))+\frac{1}{2}\cdot[\![S]\!](\lambda x.\lesssim^H(X))\\
&= [\![R\oplus S]\!](\lambda x.\lesssim^H(X)).
\end{aligned}$$

This concludes the proof. □

3.3 Context Equivalence

We now formally introduce probabilistic context equivalence and prove it to be coarser than probabilistic applicative bisimilarity.

Definition 3.29 *A $\Lambda_\oplus$-term context is a syntax tree with a unique "hole" $[\cdot]$, generated as follows:*

$$C, D \in \mathsf{C}\Lambda_\oplus ::= [\cdot] \mid \lambda x.C \mid CM \mid MC \mid C \oplus M \mid M \oplus C.$$

We denote with $C[N]$ the $\Lambda_\oplus$-term that results from filling the hole with a $\Lambda_\oplus$-term N:

$$\begin{aligned}
[\cdot][N] &\stackrel{\text{def}}{=} N;\\
(\lambda x.C)[N] &\stackrel{\text{def}}{=} \lambda x.C[N];\\
(CM)[N] &\stackrel{\text{def}}{=} C[N]M;\\
(MC)[N] &\stackrel{\text{def}}{=} MC[N];\\
(C \oplus M)[N] &\stackrel{\text{def}}{=} C[N] \oplus M;\\
(M \oplus C)[N] &\stackrel{\text{def}}{=} M \oplus C[N].
\end{aligned}$$

We also write $C[D]$ for the context resulting from replacing the occurrence of $[\cdot]$ in the syntax tree C by the tree D.

We continue to keep track of free variables by sets $\overline{x}$ of variables and we inductively define subsets $\mathsf{C}\Lambda_\oplus(\overline{x}\,;\,\overline{y})$ of contexts by the following rules:

$$\frac{}{[\cdot] \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\,\overline{x})}\ (\mathsf{Ctx1})$$

$$\frac{C \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\,\overline{y} \cup \{x\}) \qquad x \notin \overline{y}}{\lambda x.C \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\,\overline{y})}\ (\mathsf{Ctx2})$$

$$\frac{C \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\,\overline{y}) \qquad M \in \Lambda_\oplus(\overline{y})}{CM \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\,\overline{y})}\ (\mathsf{Ctx3})$$

$$\frac{M \in \Lambda_\oplus(\overline{y}) \qquad C \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\,\overline{y})}{MC \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\,\overline{y})}\ (\mathsf{Ctx4})$$

$$\frac{C \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\,\overline{y}) \qquad M \in \Lambda_\oplus(\overline{y})}{C \oplus M \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\,\overline{y})}\ (\mathsf{Ctx5})$$

$$\frac{M \in \Lambda_\oplus(\overline{y}) \qquad C \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\,\overline{y})}{M \oplus C \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\,\overline{y})}\ (\mathsf{Ctx6})$$

We use double indexing over $\overline{x}$ and $\overline{y}$ to indicate the sets of free variables before and after the filling of the hole by a term. The two following properties explain this idea.

Lemma 3.30 *If $M \in \Lambda_\oplus(\overline{x})$ and $C \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\,\overline{y})$, then $C[M] \in \Lambda_\oplus(\overline{y})$.*

Proof. By induction on the derivation of $C \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\,\overline{y})$ from the rules (Ctx1)-(Ctx6). □

Lemma 3.31 *If $C \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\,\overline{y})$ and $D \in \mathsf{C}\Lambda_\oplus(\overline{y}\,;\,\overline{y})$, then $D[C] \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\,\overline{y})$.*

Proof. By induction on the derivation of $D \in \mathsf{C}\Lambda_\oplus(\overline{y}\,;\,\overline{y})$ from the rules (Ctx1)-(Ctx6). □

Let us recall here the definition of context preorder and equivalence.

Definition 3.32 *The* probabilistic context preorder *with respect to call-by-name evaluation is the $\Lambda_\oplus$-relation given by $\overline{x} \vdash M \leq_\oplus N$ iff $\forall\, C \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\,\emptyset)$, $C[M]{\Downarrow_p}$ implies $C[N]{\Downarrow_q}$ with $p \leq q$. The $\Lambda_\oplus$-relation of* probabilistic context equivalence*, denoted $\overline{x} \vdash M \simeq_\oplus N$, holds iff $\overline{x} \vdash M \leq_\oplus N$ and $\overline{x} \vdash N \leq_\oplus M$ do.*

Lemma 3.33 *The context preorder $\leq_\oplus$ is a precongruence relation.*

Proof. Proving $\leq_\oplus$ being a precongruence relation means to prove it transitive and compatible. We start by proving $\leq_\oplus$ being transitive, that is, for every $\overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X})$ and for every M, N, $L \in \Lambda_\oplus(\overline{x})$, $\overline{x} \vdash M \leq_\oplus N$ and $\overline{x} \vdash N \leq_\oplus L$ imply $\overline{x} \vdash M \leq_\oplus L$. By Definition 3.32, the latter boils down to prove that, the following hypotheses

- For every C, $C[M]\Downarrow_p$ implies $C[N]\Downarrow_q$, with $p \leq q$;
- For every C, $C[N]\Downarrow_p$ implies $C[L]\Downarrow_q$, with $p \leq q$,
- $D[M]\Downarrow_r$

imply $D[L]\Downarrow_s$, with $r \leq s$. We can easily apply the first hypothesis when C is just D, then the second hypothesis (again with C equal to D), and get the thesis. We prove $\leq_\oplus$ of being a compatible relation starting from (Com2) property because (Com1) is trivially valid. In particular, we must show that, for every $\overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X})$, for every $x \in \mathsf{X} - \{\overline{x}\}$ and for every M, $N \in \Lambda_\oplus(\overline{x} \cup \{x\})$, if $\overline{x} \cup \{x\} \vdash M \leq_\oplus N$ then $\overline{x} \vdash \lambda x.M \leq_\oplus \lambda x.N$. By Definition 3.32, the latter boils down to prove that, the following hypotheses

- For every C, $C[M]\Downarrow_p$ implies $C[N]\Downarrow_q$, with $p \leq q$,
- $D[\lambda x.M]\Downarrow_r$

imply $D[\lambda x.N]\Downarrow_s$, with $r \leq s$. Since $D \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\emptyset)$, let us consider the context $\lambda x.[\cdot] \in \mathsf{C}\Lambda_\oplus(\overline{x} \cup \{x\}\,;\overline{x})$. Then, by Lemma 3.31, the context E of the form $D[\lambda x.[\cdot]]$ is in $\mathsf{C}\Lambda_\oplus(\overline{x} \cup \{x\}\,;\emptyset)$. Please note that, by Definition 3.29, $D[\lambda x.M] = E[M]$ and, therefore, the second hypothesis can be rewritten as $E[M]\Downarrow_r$. Thus, it follows that $E[N]\Downarrow_s$, with $r \leq s$. Moreover, observe that $E[N]$ is nothing else than $D[\lambda x.N]$. Since we have just proved $\leq_\oplus$ of being transitive, we prove (Com3) property by showing that (Com3L) and (Com3R) hold. In fact, recall that by Lemma 3.8, the latter two, together, imply the former. In particular, to prove (Com3L) we must show that, for every $\overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X})$ and for every M, N, $L \in \Lambda_\oplus(\overline{x})$, if $\overline{x} \vdash M \leq_\oplus N$ then $\overline{x} \vdash ML \leq_\oplus NL$. By Definition 3.32, the latter boils down to prove that, the following hypothesis

- For every C, $C[M]\Downarrow_p$ implies $C[N]\Downarrow_q$, with $p \leq q$,
- $D[ML]\Downarrow_r$

imply $D[NL]\Downarrow_s$, with $r \leq s$. Since $D \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\emptyset)$, let us consider the context $[\cdot]L \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\overline{x})$. Then, by Lemma 3.31, the context E of the form $D[[\cdot]L]$ is in $\mathsf{C}\Lambda_\oplus(\overline{x}\,;\emptyset)$. Please note that, by Definition 3.29, $D[ML] = E[M]$ and, therefore, the second hypothesis can be rewritten as $E[M]\Downarrow_r$. Thus, it follows that $E[N]\Downarrow_s$, with $r \leq s$. Moreover, observe that $E[N]$ is nothing else than $D[\lambda x.N]$. We do not detail the proof for (Com3R) that follows the reasoning made for (Com3L), but considering E as the context $D[L[\cdot]]$. Proving (Com4) follows the same pattern resulted for (Com3). In fact, by Lemma 3.9, (Com4L) and (Com4R) together imply (Com4). We do not detail the proofs since they proceed the reasoning made for (Com3L), considering the appropriate context each time. This concludes the proof. □

Corollary 3.34 *The context equivalence $\simeq_\oplus$ is a congruence relation.*

Proof. Straightforward consequence of the definition $\simeq_\oplus = \leq_\oplus \cap \leq_\oplus{}^{op}$. □

Lemma 3.35 *Let $\mathcal{R}$ be a compatible $\Lambda_\oplus$-relation. If $\overline{x} \vdash M\ \mathcal{R}\ N$ and $C \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\overline{y})$, then $\overline{y} \vdash C[M]\ \mathcal{R}\ C[N]$.*

Proof. By induction on the derivation of $C \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\overline{y})$:

- If C is due to (Ctx1) then $C = [\cdot]$. Thus, $C[M] = M$, $C[N] = N$ and the result trivially holds.
- If (Ctx2) is the last rule used, then $C = \lambda x.D$, with $D \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\overline{y} \cup \{x\})$. By induction hypothesis, it holds that $\overline{y} \cup \{x\} \vdash D[M]\ \mathcal{R}\ D[N]$. Since $\mathcal{R}$ is a compatible relation, it follows $\overline{y} \vdash \lambda x.D[M]\ \mathcal{R}\ \lambda x.D[N]$, that is $\overline{y} \vdash C[M]\ \mathcal{R}\ C[N]$.
- If (Ctx3) is the last rule used, then $C = DL$, with $D \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\overline{y})$ and $L \in \Lambda_\oplus(\overline{y})$. By induction hypothesis, it holds that $\overline{y} \vdash D[M]\ \mathcal{R}\ D[N]$. Since $\mathcal{R}$ is a compatible relation, it follows $\overline{y} \vdash D[M]L\ \mathcal{R}\ D[N]L$, which by definition means $\overline{y} \vdash (DL)[M]\ \mathcal{R}\ (DL)[N]$. Hence, the result $\overline{y} \vdash C[M]\ \mathcal{R}\ C[N]$ holds. The case of rule (Ctx4) holds by a similar reasoning.

- If (Ctx5) is the last rule used, then $C = D \oplus L$, with $D \in \mathsf{C}\Lambda_{\oplus}(\overline{x}\,;\overline{y})$ and $L \in \Lambda_{\oplus}(\overline{y})$. By induction hypothesis, it holds that $\overline{y} \vdash D[M]\ \mathcal{R}\ D[N]$. Since $\mathcal{R}$ is a compatible relation, it follows $\overline{y} \vdash D[M] \oplus L\ \mathcal{R}\ D[N] \oplus L$, which by definition means $\overline{y} \vdash (D \oplus L)[M]\ \mathcal{R}\ (D \oplus L)[N]$. Hence, the result $\overline{y} \vdash C[M]\ \mathcal{R}\ C[N]$ holds. The case of rule (Ctx6) holds by a similar reasoning.

This concludes the proof. □

Lemma 3.36 *If $\overline{x} \vdash M \sim N$ and $C \in \mathsf{C}\Lambda_{\oplus}(\overline{x}\,;\overline{y})$, then $\overline{y} \vdash C[M] \sim C[N]$.*

Proof. Since $\sim = \lesssim \cap \lesssim^{op}$ by Proposition 2.13, $\overline{x} \vdash M \sim N$ implies $\overline{x} \vdash M \lesssim N$ and $\overline{x} \vdash N \lesssim M$. Since, by Theorem 3.18, $\lesssim$ is a precongruence hence a compatible relation, $\overline{y} \vdash C[M] \lesssim C[N]$ and $\overline{y} \vdash C[N] \lesssim C[M]$ follow by Lemma 3.35, i.e. $\overline{y} \vdash C[M] \sim C[N]$. □

Theorem 3.37 *For all $\overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X})$ and every $M, N \in \Lambda_{\oplus}(\overline{x})$, $\overline{x} \vdash M \sim N$ implies $\overline{x} \vdash M \simeq_{\oplus} N$.*

Proof. If $\overline{x} \vdash M \sim N$, then for every $C \in \mathsf{C}\Lambda_{\oplus}(\overline{x}\,;\emptyset)$, $\emptyset \vdash C[M] \sim C[N]$ follows by Lemma 3.36. By Lemma 3.5, the latter implies $\sum[\![C[M]]\!] = p = \sum[\![C[N]]\!]$. This means in particular that $C[M]{\Downarrow_p}$ iff $C[N]{\Downarrow_p}$, which is equivalent to $\overline{x} \vdash M \simeq_{\oplus} N$ by definition. □

The converse inclusion fails. A counterexample is described in the following.

Example 3.38 *For $M \stackrel{\text{def}}{=} \lambda x.L \oplus P$ and $N \stackrel{\text{def}}{=} (\lambda x.L) \oplus (\lambda x.P)$ (where L is $\lambda y.\Omega$ and P is $\lambda y.\lambda z.\Omega$), we have $M \not\lesssim N$, hence $M \not\sim N$, but $M \simeq_{\oplus} N$.*

We prove that the above two terms are context equivalent by means of *CIU-equivalence*. This is a relation that can be shown to coincide with context equivalence by a Context Lemma, itself proved by the Howe's technique. See Section 4 and Section 5 for supplementary details on the above counterexample.

4 Context Free Context Equivalence

We present here a way of treating the problem of too concrete representations of contexts: right now, we cannot basically work up-to α-equivalence classes of contexts. Let us dispense with them entirely, and work instead with a coinductive characterization of the context preorder, and equivalence, phrased in terms of $\Lambda_{\oplus}$-relations.

Definition 4.1 *A $\Lambda_{\oplus}$-relation $\mathcal{R}$ is said to be adequate if, for every $M, N \in \Lambda_{\oplus}(\emptyset)$, $\emptyset \vdash M\ \mathcal{R}\ N$ implies $M{\Downarrow_p}$ and $N{\Downarrow_q}$, with $p \leq q$.*

Let us indicate with $\mathbb{CA}$ the collection of all compatible and adequate $\Lambda_{\oplus}$-relations and let

$$\leq_{\oplus}^{\mathsf{ca}} \stackrel{\text{def}}{=} \bigcup \mathbb{CA}. \tag{6}$$

It turns out that the context preorder $\leq_{\oplus}$ is the largest $\Lambda_{\oplus}$-relation that is both compatible and adequate, that is $\leq_{\oplus} = \leq_{\oplus}^{\mathsf{ca}}$. Let us proceed towards a proof for the latter.

Lemma 4.2 *For every $\mathcal{R}, \mathcal{T} \in \mathbb{CA}$, $\mathcal{R} \circ \mathcal{T} \in \mathbb{CA}$.*

Proof. We need to show that $\mathcal{R} \circ \mathcal{T} = \{(M,N) \mid \exists L \in \Lambda_{\oplus}(\overline{x}).\, \overline{x} \vdash M\ \mathcal{R}\ L \wedge \overline{x} \vdash L\ \mathcal{T}\ N\}$ is a compatible and adequate $\Lambda_{\oplus}$-relation. Obviously, $\mathcal{R} \circ \mathcal{T}$ is adequate: for every $(M,N) \in \mathcal{R} \circ \mathcal{T}$, there exists a term L such that $M{\Downarrow_p} \Rightarrow L{\Downarrow_q} \Rightarrow N{\Downarrow_r}$, with $p \leq q \leq r$. Then, $M{\Downarrow_p} \Rightarrow N{\Downarrow_r}$, with $p \leq r$. Note that the identity relation $\mathsf{ID} \stackrel{\text{def}}{=} \{(M,M) \mid M \in \Lambda_{\oplus}(\overline{x})\}$ is in $\mathcal{R} \circ \mathcal{T}$. Then, $\mathcal{R} \circ \mathcal{T}$ is reflexive and, in particular, satisfies compatibility property (Com1). Proving (Com2) means to show that, if $\overline{x} \cup \{x\} \vdash M\ (\mathcal{R} \circ \mathcal{T})\ N$, then $\overline{x} \vdash \lambda x.M\ (\mathcal{R} \circ \mathcal{T})\ \lambda x.N$. From the hypothesis, it follows that there exists a term L such that $\overline{x} \cup \{x\} \vdash M\ \mathcal{R}\ L$ and $\overline{x} \cup \{x\} \vdash L\ \mathcal{T}\ N$. Since both $\mathcal{R}$ and $\mathcal{T}$ are in $\mathbb{CA}$, hence compatible, it holds $\overline{x} \vdash \lambda x.M\ \mathcal{R}\ \lambda x.L$ and $\overline{x} \vdash \lambda x.L\ \mathcal{T}\ \lambda x.N$. The latter

together imply $\overline{x} \vdash \lambda x.M\ (\mathcal{R} \circ \mathcal{T})\ \lambda x.N$. Proving (Com3) means to show that, if $\overline{x} \vdash M\ (\mathcal{R} \circ \mathcal{T})\ N$ and $\overline{x} \vdash Q\ (\mathcal{R} \circ \mathcal{T})\ R$, then $\overline{x} \vdash MQ\ (\mathcal{R} \circ \mathcal{T})\ NR$. From the hypothesis, it follows that there exist two terms L, P such that, on the one hand, $\overline{x} \vdash M\ \mathcal{R}\ L$ and $\overline{x} \vdash L\ \mathcal{T}\ N$, and on the other hand, $\overline{x} \vdash Q\ \mathcal{R}\ P$ and $\overline{x} \vdash P\ \mathcal{T}\ R$. Since both $\mathcal{R}$ and $\mathcal{T}$ are in $\mathbb{CA}$, hence compatible, it holds:

$$\overline{x} \vdash M\ \mathcal{R}\ L \ \wedge\ \overline{x} \vdash Q\ \mathcal{R}\ P \Rightarrow \overline{x} \vdash MQ\ \mathcal{R}\ LP;$$

$$\overline{x} \vdash L\ \mathcal{T}\ N \ \wedge\ \overline{x} \vdash P\ \mathcal{T}\ R \Rightarrow \overline{x} \vdash LP\ \mathcal{T}\ NR.$$

The two together imply $\overline{x} \vdash MQ\ (\mathcal{R} \circ \mathcal{T})\ NR$.

Proceeding in the same fashion, one can easily prove property (Com4). □

Lemma 4.3 *$\Lambda_\oplus$-relation $\leq^{\mathsf{ca}}_\oplus$ is adequate.*

Proof. It suffices to note that the property of being *adequate* is closed under taking unions of relations. Indeed, if $\mathcal{R}$, $\mathcal{T}$ are adequate relations, then it is easy to see that the union $\mathcal{R} \cup \mathcal{T}$ is: for every couple $(M, N) \in \mathcal{R} \cup \mathcal{T}$, either $\overline{x} \vdash M\ \mathcal{R}\ N$ or $\overline{x} \vdash M\ \mathcal{T}\ N$. Either way, $M{\Downarrow_p} \Rightarrow N{\Downarrow_q}$, with $p \leq q$, implying $\mathcal{R} \cup \mathcal{T}$ of being adequate. □

Lemma 4.4 *$\Lambda_\oplus$-relation $\leq^{\mathsf{ca}}_\oplus$ is a precongruence.*

Proof. We need to show that $\leq^{\mathsf{ca}}_\oplus$ is a transitive and compatible relation. By Lemma 4.2, $\leq^{\mathsf{ca}}_\oplus \circ \leq^{\mathsf{ca}}_\oplus \subseteq \leq^{\mathsf{ca}}_\oplus$ which implies $\leq^{\mathsf{ca}}_\oplus$ of being transitive. Let us now prove that $\leq^{\mathsf{ca}}_\oplus$ is also compatible. Note that the identity relation $\mathsf{ID} = \{(M, M) \mid M \in \Lambda_\oplus(\overline{x})\}$ is in $\mathbb{CA}$, which implies reflexivity of $\leq^{\mathsf{ca}}_\oplus$ and hence, in particular, it satisfies property (Com1). It is clear that property (Com2) is closed under taking unions of relations, so that $\leq^{\mathsf{ca}}_\oplus$ satisfies (Com2) too. The same is not true for properties (Com3) and (Com4). By Lemma 3.8 (respectively, Lemma 3.9), for (Com3) (resp., (Com4)) it suffices to show that $\leq^{\mathsf{ca}}_\oplus$ satisfies (Com3L) and (Com3R) (resp., (Com4L) and (Com4R)). This is obvious: contrary to (Com3) (resp., (Com4)), these properties clearly are closed under taking unions of relations.

This concludes the proof. □

Corollary 4.5 *$\leq^{\mathsf{ca}}_\oplus$ is the largest compatible and adequate $\Lambda_\oplus$-relation.*

Proof. Straightforward consequence of Lemma 4.3 and Lemma 4.4. □

Lemma 4.6 *$\Lambda_\oplus$-relations $\leq_\oplus$ and $\leq^{\mathsf{ca}}_\oplus$ coincide.*

Proof. By Definition 3.32, it is immediate that $\leq_\oplus$ is adequate. Moreover, by Lemma 3.33, $\leq_\oplus$ is a precongruence. Therefore $\leq_\oplus \in \mathbb{CA}$ implying $\leq_\oplus \subseteq \leq^{\mathsf{ca}}_\oplus$. Let us prove the converse. Since, by Lemma 4.4, $\leq^{\mathsf{ca}}_\oplus$ is a precongruence hence a compatible relation, it holds that, for every M, $N \in \Lambda_\oplus(\overline{x})$ and for every $C \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\,\overline{y})$, $\overline{x} \vdash M \leq^{\mathsf{ca}}_\oplus N$ implies $\overline{y} \vdash C[M] \leq^{\mathsf{ca}}_\oplus C[N]$. Therefore, for every M, $N \in \Lambda_\oplus(\overline{x})$ and for every $C \in \mathsf{C}\Lambda_\oplus(\overline{x}\,;\,\emptyset)$,

$$\overline{x} \vdash M \leq^{\mathsf{ca}}_\oplus N \Rightarrow \emptyset \vdash C[M] \leq^{\mathsf{ca}}_\oplus C[N]$$

which implies, by the fact that $\leq^{\mathsf{ca}}_\oplus$ is adequate,

$$C[M]{\Downarrow_p} \Rightarrow C[N]{\Downarrow_q},\ \text{with } p \leq q$$

that is, by Definition 3.32,

$$\overline{x} \vdash M \leq_\oplus N.$$

This concludes the proof. □

5 CIU-Equivalence

CIU-equivalence is a simpler characterization of that kind of program equivalence we are interested in, i.e., context equivalence. In fact, we will prove that the two notions coincide. While context equivalence envisages a quantification over all contexts, CIU-equivalence relaxes such constraint to a restricted class of contexts without affecting the associated notion of program equivalence. Such a class of contexts is that of *evaluation* contexts. In particular, we use a different representation of evaluation contexts, seeing them as a stack of evaluation frames.

Definition 5.1 *The set of* frame stacks *is given by the following set of rules:*

$$\mathbf{S}, \mathbf{T} ::= \mathtt{nil} \;\Big|\; [\cdot]M :: \mathbf{S}.$$

The set of free variables of a frame stack $\mathbf{S}$ can be easily defined as the union of the variables occurring free in the terms embedded into it. Given a set of variables $\overline{x}$, define $\mathcal{FS}(\overline{x})$ as the set of frame stacks whose free variables are all from $\overline{x}$. Given a frame stack $\mathbf{S} \in \mathcal{FS}(\overline{x})$ and a term $M \in \Lambda_{\oplus}(\overline{x})$, we define the term $E_{\mathbf{S}}(M) \in \Lambda_{\oplus}(\overline{x})$ as follows:

$$E_{\mathtt{nil}}(M) \stackrel{\text{def}}{=} M;$$
$$E_{[\cdot]M::\mathbf{S}}(N) \stackrel{\text{def}}{=} E_{\mathbf{S}}(NM).$$

We now define a binary relation $\rightsquigarrow_{\mathsf{n}}$ between pairs of the form $(\mathbf{S}, M)$ and *sequences* of pairs in the same form:

$$(\mathbf{S}, MN) \rightsquigarrow_{\mathsf{n}} ([\cdot]N :: \mathbf{S}, M);$$
$$(\mathbf{S}, M \oplus N) \rightsquigarrow_{\mathsf{n}} (\mathbf{S}, M), (\mathbf{S}, N);$$
$$([\cdot]M :: \mathbf{S}, \lambda x.N) \rightsquigarrow_{\mathsf{n}} (\mathbf{S}, N\{^{M}/_{x}\}).$$

Finally, we define a formal system whose judgments are in the form $(\mathbf{S}, M) \downarrow^{p}_{\mathsf{n}}$ and whose rules are as follows:

$$\frac{}{(\mathbf{S}, M) \downarrow^{0}_{\mathsf{n}}}\ (\mathsf{empty})$$

$$\frac{}{(\mathtt{nil}, V) \downarrow^{1}_{\mathsf{n}}}\ (\mathsf{value})$$

$$\frac{(\mathbf{S}, M) \rightsquigarrow_{\mathsf{n}} (\mathbf{T}_1, N_1), \ldots, (\mathbf{T}_n, N_n) \quad (\mathbf{T}_i, N_i) \downarrow^{p_i}_{\mathsf{n}}}{(\mathbf{S}, M) \downarrow^{\frac{1}{n}\sum_{i=1}^{n} p_i}_{\mathsf{n}}}\ (\mathsf{term})$$

The expression $\mathbb{C}(\mathbf{S}, M)$ stands for the real number $\sup_{p \in \mathbb{R}} (\mathbf{S}, M) \downarrow^{p}_{\mathsf{n}}$.

Lemma 5.2 *For all closed frame stacks $\mathbf{S} \in \mathcal{FS}(\emptyset)$ and closed $\Lambda_{\oplus}$-terms $M \in \Lambda_{\oplus}(\emptyset)$, $\mathbb{C}(\mathbf{S}, M) = p$ iff $E_{\mathbf{S}}(M) \Downarrow_p$. In particular, $M \Downarrow_p$ holds iff $\mathbb{C}(\mathtt{nil}, M) = p$.*

Proof. First of all, we recall here that the work of Dal Lago and Zorzi [10] provides various call-by-name inductive semantics, either big-steps or small-steps, which are all equivalent. Then, the result can be deduced from the following properties:

1. For all $\mathbf{S} \in \mathcal{FS}(\emptyset)$, if $(\mathbf{S}, M) \downarrow^{p}_{\mathsf{n}}$ then $\exists \mathscr{D}.\ E_{\mathbf{S}}(M) \Rightarrow_{\mathsf{IN}} \mathscr{D}$ with $\sum \mathscr{D} = p$.

   Proof. By induction on the derivation of $(\mathbf{S}, M) \downarrow^{p}_{\mathsf{n}}$, looking at the last rule used.

   - (empty) rule used: $(\mathbf{S}, M) \downarrow^{0}_{\mathsf{n}}$. Then, consider the empty distribution $\mathscr{D} \stackrel{\text{def}}{=} \emptyset$ and observe that $E_{\mathbf{S}}(M) \Rightarrow_{\mathsf{IN}} \mathscr{D}$ by se_{n} rule.
   - (value) rule used: $(\mathbf{S}, M) \downarrow^{1}_{\mathsf{n}}$ implies $\mathbf{S} = \mathtt{nil}$ and M of being a value, say V. Then, consider the distribution $\mathscr{D} \stackrel{\text{def}}{=} \{V^1\}$ and observe that $E_{\mathtt{nil}}(V) = V \Rightarrow_{\mathsf{IN}} \mathscr{D}$ by sv_{n} rule. Of course, $\sum \mathscr{D} = 1 = p$.

- (term) rule used: $(\mathbf{S}, M) \downarrow_{\mathsf{n}}^{\frac{1}{n}\sum_{i=1}^{n} p_i}$ obtained from $(\mathbf{S}, M) \rightsquigarrow_{\mathsf{n}} (\mathbf{T}_1, N_1), \ldots, (\mathbf{T}_n, N_n)$ and, for every $i \in \{1, \ldots, n\}$, $(\mathbf{T}_i, N_i) \downarrow_{\mathsf{n}}^{p_i}$. Then, by induction hypothesis, there exist $\mathscr{E}_1, \ldots, \mathscr{E}_n$ such that $E_{\mathbf{T}_i}(N_i) \Rightarrow_{\mathsf{IN}} \mathscr{E}_i$ with $\sum \mathscr{E}_i = p_i$.
  Let us now proceed by cases according to the structure of M.
  - If $M = \lambda x.L$, then $\mathbf{S} = [\cdot]P :: \mathbf{T}$ implying $n = 1$, $\mathbf{T}_1 = \mathbf{T}$ and $N_1 = L\{^P/_x\}$. Then, consider the distribution $\mathscr{D} \stackrel{\text{def}}{=} \mathscr{E}_1$ and observe that $E_{\mathbf{S}}(M) = E_{[\cdot]P::\mathbf{T}}(\lambda x.L) = E_{\mathbf{T}}((\lambda x.L)P) \mapsto_{\mathsf{n}} E_{\mathbf{T}}(L\{^P/_x\}) = E_{\mathbf{T}_1}(N_1)$. Hence, $E_{\mathbf{S}}(M) \Rightarrow_{\mathsf{IN}} \mathscr{D}$ by sm_{n} rule. Moreover, $\sum \mathscr{D} = \sum \mathscr{E}_1 = p_1 = \frac{1}{n}\sum_{i=1}^{n} p_i = p$.
  - If $M = L \oplus P$, then $n = 2$, $\mathbf{T}_1 = \mathbf{T}_2 = \mathbf{S}$, $N_1 = L$ and $N_2 = P$. Then, consider the distribution $\mathscr{D} \stackrel{\text{def}}{=} \sum_{i=1}^{2} \frac{1}{2}\mathscr{E}_i$ and observe that $E_{\mathbf{S}}(M) = E_{\mathbf{S}}(L \oplus P) \mapsto_{\mathsf{n}} E_{\mathbf{S}}(L), E_{\mathbf{S}}(P) = E_{\mathbf{T}_1}(N_1), E_{\mathbf{T}_2}(N_2)$. Hence, $E_{\mathbf{S}}(M) \Rightarrow_{\mathsf{IN}} \mathscr{D}$ by sm_{n} rule. Moreover, $\sum \mathscr{D} = \sum \sum_{i=1}^{2} \frac{1}{2}\mathscr{E}_i = \frac{1}{2}\sum_{i=1}^{2} \sum \mathscr{E}_i = \frac{1}{2}\sum_{i=1}^{2} p_i = p$.
  - If $M = LP$, then $n = 1$, $\mathbf{T}_1 = [\cdot]P :: \mathbf{S}$ and $N_1 = L$. Then, consider the distribution $\mathscr{D} \stackrel{\text{def}}{=} \mathscr{E}_1$ and observe that $E_{[\cdot]P::\mathbf{S}}(L) \Rightarrow_{\mathsf{IN}} \mathscr{E}_1$ implies $E_{\mathbf{S}}(M) \Rightarrow_{\mathsf{IN}} \mathscr{D}$. Moreover, $\sum \mathscr{D} = \sum \mathscr{E}_1 = p_1 = \frac{1}{n}\sum_{i=1}^{n} p_i = p$.

  This concludes the proof. □

2. For all $\mathscr{D}$, if $M \Rightarrow_{\mathsf{IN}} \mathscr{D}$ then $\exists \mathbf{S}, N.\ E_{\mathbf{S}}(N) = M$ and $(\mathbf{S}, N) \downarrow_{\mathsf{n}}^{p}$ with $\sum \mathscr{D} = p$.

   Proof. By induction on the derivation of $M \Rightarrow_{\mathsf{IN}} \mathscr{D}$, looking at the last rule used. (We refer here to the inductive schema of inference rules gave in [10] for small-step call-by-name semantics of $\Lambda_{\oplus}$.)
   - se_{n} rule used: $M \Rightarrow_{\mathsf{IN}} \emptyset$. Then, for every $\mathbf{S}$ and every N such that $E_{\mathbf{S}}(N) = M$, $(\mathbf{S}, N) \downarrow_{\mathsf{n}}^{0}$ by (empty) rule. Of course, $\sum \mathscr{D} = 0 = p$.
   - sv_{n} rule used: M is a value, say V, and $\mathscr{D} = \{V^1\}$ with $V \Rightarrow_{\mathsf{IN}} \{V^1\}$. Then, consider $\mathbf{S} \stackrel{\text{def}}{=} \mathtt{nil}$ and $N \stackrel{\text{def}}{=} V$: by definition $E_{\mathbf{S}}(N) = E_{\mathtt{nil}}(V) = V = M$. By (value) rule, $(\mathtt{nil}, V) \downarrow_{\mathsf{n}}^{1}$ hence $\sum \mathscr{D} = 1 = p$.
   - sv_{n} rule used: $M \Rightarrow_{\mathsf{IN}} \sum_{i=1}^{n} \frac{1}{n}\mathscr{E}_i$ from $M \mapsto_{\mathsf{n}} Q_1, \ldots, Q_n$ with, for every $i \in \{1, \ldots, n\}$, $Q_i \Rightarrow_{\mathsf{IN}} \mathscr{E}_i$. By induction hypothesis, for every $i \in \{1, \ldots, n\}$, there exist $\mathbf{T}_i$ and N_i such that $E_{\mathbf{T}_i}(N_i) = Q_i$ and $(\mathbf{T}_i, N_i) \downarrow_{\mathsf{n}}^{p_i}$ with $\sum \mathscr{E}_i = p_i$.
     Let us proceed by cases according to the structure of M.
     - If $M = (\lambda x.L)P$, then $n = 1$ and $Q_1 = L\{^P/_x\}$. Hence, consider $\mathbf{S} \stackrel{\text{def}}{=} [\cdot]P :: \mathtt{nil}$ and $N \stackrel{\text{def}}{=} \lambda x.L$: by definition, $E_{\mathbf{S}}(N) = E_{[\cdot]P::\mathtt{nil}}(\lambda x.L) = E_{\mathtt{nil}}((\lambda x.L)P) = (\lambda x.L)P = M$. By (term) rule, $(\mathbf{S}, N) = ([\cdot]P :: \mathtt{nil}, \lambda x.L) \rightsquigarrow_{\mathsf{n}} (\mathtt{nil}, L\{^P/_x\})$ with, by induction hypothesis result, $(\mathtt{nil}, L\{^P/_x\}) \downarrow_{\mathsf{n}}^{p_1}$. The latter implies $(\mathbf{S}, N) \downarrow_{\mathsf{n}}^{p_1}$. Moreover, $\sum \mathscr{D} = \sum \sum_{i=1}^{n} \frac{1}{n}\mathscr{E}_i = \sum \mathscr{E}_1 = p_1 = p$.
     - If $M = L \oplus P$, then $n = 2$, $Q_1 = L$ and $Q_2 = P$. Hence, consider $\mathbf{S} \stackrel{\text{def}}{=} \mathtt{nil}$ and $N \stackrel{\text{def}}{=} L \oplus P$: by definition, $E_{\mathbf{S}}(N) = E_{\mathtt{nil}}(L \oplus P) = L \oplus P = M$. By (term) rule, $(\mathbf{S}, N) = (\mathtt{nil}, L \oplus P) \rightsquigarrow_{\mathsf{n}} (\mathtt{nil}, L), (\mathtt{nil}, P)$ with, by induction hypothesis result, $(\mathtt{nil}, L) \downarrow_{\mathsf{n}}^{p_1}$ and $(\mathtt{nil}, P) \downarrow_{\mathsf{n}}^{p_2}$. The latter implies $(\mathbf{S}, N) \downarrow_{\mathsf{n}}^{\frac{1}{2}\sum_{i=1}^{2} p_i}$. Moreover, $\sum \mathscr{D} = \sum \sum_{i=1}^{n} \frac{1}{n}\mathscr{E}_i = \sum \sum_{i=1}^{n} \frac{1}{2}\mathscr{E}_i = \frac{1}{2}\sum_{i=1}^{2} \sum \mathscr{E}_i = \frac{1}{2}\sum_{i=1}^{2} p_i = p$.
     - If $M = LP$ and $L \mapsto_{\mathsf{n}} R_1, \ldots, R_n$, then $Q_i = R_i P$ for every $i \in \{1, \ldots, n\}$. Hence, consider $\mathbf{S} \stackrel{\text{def}}{=} [\cdot]P :: \mathtt{nil}$ and $N \stackrel{\text{def}}{=} L$: by definition, $E_{\mathbf{S}}(N) = E_{[\cdot]P::\mathtt{nil}}(L) = E_{\mathtt{nil}}(LP) = LP = M$. By (term) rule, $(\mathbf{S}, N) = ([\cdot]P :: \mathtt{nil}, L) \rightsquigarrow_{\mathsf{n}} ([\cdot]P :: \mathtt{nil}, R_1), \ldots, ([\cdot]P :: \mathtt{nil}, R_n)$ with, by induction hypothesis result, $([\cdot]P :: \mathtt{nil}, R_i) \downarrow_{\mathsf{n}}^{p_i}$ for every $i \in \{1, \ldots, n\}$. The latter implies $(\mathbf{S}, N) \downarrow_{\mathsf{n}}^{\frac{1}{n}\sum_{i=1}^{n} p_i}$. Moreover, $\sum \mathscr{D} = \sum \sum_{i=1}^{n} \frac{1}{n}\mathscr{E}_i = \frac{1}{n}\sum_{i=1}^{n} \sum \mathscr{E}_i = \frac{1}{n}\sum_{i=1}^{n} p_i = p$.

     This concludes the proof. □

Generally speaking, the two properties above prove the following double implication:

$$(\mathbf{S}, M) \downarrow^p_{\mathsf{n}} \iff E_{\mathbf{S}}(M) \Downarrow_{\mathsf{IN}} \mathscr{D} \text{ with } \sum \mathscr{D} = p. \tag{7}$$

Then,

$$\begin{aligned} p = \mathbb{C}(\mathbf{S}, M) = \sup_{q \in \mathbb{R}}(\mathbf{S}, M) \downarrow^q_{\mathsf{n}} &= \sup_{E_{\mathbf{S}}(M) \Downarrow_{\mathsf{IN}} \mathscr{D}} \sum \mathscr{D} \\ &= \sum \sup_{E_{\mathbf{S}}(M) \Downarrow_{\mathsf{IN}} \mathscr{D}} \mathscr{D} = \sum [\![E_{\mathbf{S}}(M)]\!] = E_{\mathbf{S}}(M) \Downarrow_p, \end{aligned}$$

which concludes the proof. □

Given $M, N \in \Lambda_{\oplus}(\emptyset)$, we define $M \preceq^{\mathsf{CIU}} N$ iff for every $\mathbf{S}$, $\mathbb{C}(\mathbf{S}, M) \leq \mathbb{C}(\mathbf{S}, N)$. This relation can be extended to a relation on open terms in the usual way. Moreover, we stipulate $M \cong^{\mathsf{CIU}} N$ iff both $M \preceq^{\mathsf{CIU}} N$ and $N \preceq^{\mathsf{CIU}} M$.

Since $\preceq^{\mathsf{CIU}}$ is a preorder, proving it to be a precongruence boils down to show the following implication:

$$M (\preceq^{\mathsf{CIU}})^H N \Rightarrow M \preceq^{\mathsf{CIU}} N.$$

Indeed, the converse implication is a consequence of Lemma 3.12 and the obvious reflexivity of $\preceq^{\mathsf{CIU}}$ relation. To do that, we extend Howe's construction to frame stacks in a natural way:

$$\frac{}{\mathtt{nil} \mathcal{R}^H \mathtt{nil}} \ (\mathsf{Howstk1})$$

$$\frac{\emptyset \vdash M \ \mathcal{R}^H \ N \qquad \mathbf{S} \mathcal{R}^H \mathbf{T}}{([\cdot] M :: \mathbf{S}) \mathcal{R}^H ([\cdot] N :: \mathbf{T})} \ (\mathsf{Howstk2})$$

Lemma 5.3 *For every $\overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X})$, it holds $\overline{x} \vdash (\lambda x.M)N \ \cong^{\mathsf{CIU}} \ M\{^N/_x\}$.*

Proof. We need to show that both $\overline{x} \vdash (\lambda x.M)N \ \preceq^{\mathsf{CIU}} \ M\{^N/_x\}$ and $\overline{x} \vdash M\{^N/_x\} \ \preceq^{\mathsf{CIU}} (\lambda x.M)N$ hold. Since $\preceq^{\mathsf{CIU}}$ is defined on open terms by taking closing term-substitutions, it suffices to show the result for close $\Lambda_{\oplus}$-terms only: $(\lambda x.M)N \preceq^{\mathsf{CIU}} M\{^N/_x\}$ and $M\{^N/_x\} \preceq^{\mathsf{CIU}} (\lambda x.M)N$.

Let us start with $(\lambda x.M)N \ \preceq^{\mathsf{CIU}} \ M\{^N/_x\}$ and prove that, for every close frame stack $\mathbf{S}$, $\mathbb{C}(\mathbf{S}, (\lambda x.M)N) \leq \mathbb{C}(\mathbf{S}, M\{^N/_x\})$. The latter is an obvious consequence of the fact that $(\mathbf{S}, (\lambda x.M)N)$ reduces to $(\mathbf{S}, M\{^N/_x\})$. Let us look into the details distinguishing two cases:

- If $\mathbf{S} = \mathtt{nil}$, then $(\mathbf{S}, (\lambda x.M)N) \rightsquigarrow_{\mathsf{n}} ([\cdot]N :: \mathbf{S}, \lambda x.M) \rightsquigarrow_{\mathsf{n}} (\mathbf{S}, M\{^N/_x\})$ which implies that $\mathbb{C}(\mathbf{S}, (\lambda x.M)N) = \sup_{p \in \mathbb{R}}(\mathbf{S}, (\lambda x.M)N) \downarrow^p_{\mathsf{n}} = \sup_{p \in \mathbb{R}}(\mathbf{S}, M\{^N/_x\}) \downarrow^p_{\mathsf{n}} = \mathbb{C}(\mathbf{S}, M\{^N/_x\})$.
- If $\mathbf{S} = [\cdot]L :: \mathbf{T}$, then we can proceed similarly.

Similarly, to prove the converse, $M\{^N/_x\} \preceq^{\mathsf{CIU}} (\lambda x.M)N$, let us fix p as $(\mathbf{S}, M\{^N/_x\}) \downarrow^p_{\mathsf{n}}$ and distinguish two cases:

- If $\mathbf{S} = \mathtt{nil}$ and $p = 0$, then $(\mathbf{S}, (\lambda x.M)N) \downarrow^0_{\mathsf{n}}$ holds too by (empty) rule. Otherwise,

$$\frac{(\mathbf{S}, M) \rightsquigarrow_{\mathsf{n}} ([\cdot]N :: \mathbf{S}, \lambda x.M) \qquad \dfrac{([\cdot]N :: \mathbf{S}, \lambda x.M) \rightsquigarrow_{\mathsf{n}} (\mathbf{S}, M\{^N/_x\}) \quad (\mathbf{S}, M\{^N/_x\}) \downarrow^p_{\mathsf{n}}}{([\cdot]N :: \mathbf{S}, \lambda x.M) \downarrow^p_{\mathsf{n}}} \ (\mathsf{term})}{(\mathbf{S}, (\lambda x.M)N) \downarrow^p_{\mathsf{n}}} \ (\mathsf{term})$$

  which implies $\mathbb{C}(\mathbf{S}, M\{^N/_x\}) = \sup_{p \in \mathbb{R}}(\mathbf{S}, M\{^N/_x\}) \downarrow^p_{\mathsf{n}} = \sup_{p \in \mathbb{R}}(\mathbf{S}, (\lambda x.M)N) \downarrow^p_{\mathsf{n}} = \mathbb{C}(\mathbf{S}, (\lambda x.M)N)$.
- If $\mathbf{S} = [\cdot]L :: \mathbf{T}$, then we can proceed similarly.

This concludes the proof. □

Lemma 5.4 *For every $\mathbf{S}, \mathbf{T} \in \mathcal{FS}(\emptyset)$ and $M, N \in \Lambda_{\oplus}(\emptyset)$, if $\mathbf{S} (\preceq^{\mathsf{CIU}})^H \mathbf{T}$ and $M (\preceq^{\mathsf{CIU}})^H N$ and $(\mathbf{S}, M) \downarrow^p_{\mathsf{n}}$, then $\mathbb{C}(\mathbf{T}, N) \geq p$.*

Proof. We go by induction on the structure of the proof of $(\mathbf{S}, M) \downarrow^p_{\mathsf{n}}$, looking at the last rule used.

- If $(\mathbf{S}, M)\downarrow^0_{\mathsf{n}}$, then trivially $\mathbb{C}(\mathbf{T}, N) \geq 0$.
- If $\mathbf{S} = \mathtt{nil}$, $M = \lambda x.L$ and $p = 1$, then $\mathbf{T} = \mathtt{nil}$ since $\mathbf{S}(\preceq^{\mathsf{CIU}})^H\mathbf{T}$. From $M(\preceq^{\mathsf{CIU}})^H N$, it follows that there is P with $x \vdash L\ (\preceq^{\mathsf{CIU}})^H\ P$ and $\emptyset \vdash \lambda x.P\ \preceq^{\mathsf{CIU}}\ N$. But the latter implies that $\mathbb{C}(\mathtt{nil}, N) \geq 1$, which is the thesis.
- Otherwise, (term) rule is used and suppose we are in the following situation

$$\frac{(\mathbf{S}, M) \rightsquigarrow_{\mathsf{n}} (\mathbf{U}_1, L_1), \ldots, (\mathbf{U}_n, L_n) \quad (\mathbf{U}_i, L_i)\downarrow^{p_i}_{\mathsf{n}}}{(\mathbf{S}, M)\downarrow^{\frac{1}{n}\sum_{i=1}^n p_i}_{\mathsf{n}}}\ (\mathsf{term})$$

Let us distinguish the following cases as in definition of $\rightsquigarrow_{\mathsf{n}}$:

- If $M = PQ$, then $n = 1$, $\mathbf{U}_1 = [\cdot]Q :: \mathbf{S}$ and $L_1 = P$. From $M(\preceq^{\mathsf{CIU}})^H N$ it follows that there are R, S with $\emptyset \vdash P\ (\preceq^{\mathsf{CIU}})^H\ R$, $\emptyset \vdash Q\ (\preceq^{\mathsf{CIU}})^H\ S$ and $\emptyset \vdash RS\ \preceq^{\mathsf{CIU}}\ N$. But then we can form the following:

$$\frac{\emptyset \vdash Q\ (\preceq^{\mathsf{CIU}})^H\ S \quad \emptyset \vdash \mathbf{S}\ (\preceq^{\mathsf{CIU}})^H\ \mathbf{T}}{\emptyset \vdash \mathbf{U}_1\ (\preceq^{\mathsf{CIU}})^H\ [\cdot]S :: \mathbf{T}}\ (\mathsf{Howstk2})$$

  and, by the induction hypothesis, conclude that $\mathbb{C}([\cdot]S :: \mathbf{T}, R) \geq p$. Now observe that

$$(\mathbf{T}, RS) \rightsquigarrow_{\mathsf{n}} ([\cdot]S :: \mathbf{T}, R),$$

  and, as a consequence, $\mathbb{C}(\mathbf{T}, RS) \geq p$, from which the thesis easily follows given that $\emptyset \vdash RS\ \preceq^{\mathsf{CIU}}\ N$.
- If $M = P \oplus Q$, then $n = 2$, $\mathbf{U}_1 = \mathbf{U}_2 = \mathbf{S}$ and $L_1 = P$, $L_2 = Q$. From $\mathbf{S}(\preceq^{\mathsf{CIU}})^H\mathbf{T}$, we get that $\mathbf{U}_1(\preceq^{\mathsf{CIU}})^H\mathbf{T}$ and $\mathbf{U}_2(\preceq^{\mathsf{CIU}})^H\mathbf{T}$. From $M(\preceq^{\mathsf{CIU}})^H N$ it follows that there are R, S with $\emptyset \vdash P\ (\preceq^{\mathsf{CIU}})^H\ R$, $\emptyset \vdash Q\ (\preceq^{\mathsf{CIU}})^H\ S$ and $\emptyset \vdash R \oplus S\ \preceq^{\mathsf{CIU}}\ N$. Then, by a double induction hypothesis, it follows $\mathbb{C}(\mathbf{T}, R) \geq p$ and $\mathbb{C}(\mathbf{T}, S) \geq p$. The latter together imply $\mathbb{C}(\mathbf{T}, R \oplus S) \geq p$, from which the thesis easily follows given that $\emptyset \vdash R \oplus S\ \preceq^{\mathsf{CIU}}\ N$.
- If $M = \lambda x.P$, then $\mathbf{S} = [\cdot]Q :: \mathbf{U}$ because the only case left. Hence $n = 1$, $\mathbf{U}_1 = \mathbf{U}$ and $L_1 = P\{^Q/_x\}$. From $\mathbf{S}(\preceq^{\mathsf{CIU}})^H\mathbf{T}$, we get that $\mathbf{T} = [\cdot]R :: \mathbf{V}$ where $\emptyset \vdash Q\ (\preceq^{\mathsf{CIU}})^H\ R$ and $\mathbf{U}(\preceq^{\mathsf{CIU}})^H\mathbf{V}$. From $M(\preceq^{\mathsf{CIU}})^H N$, it follows that for some S, it holds that $x \vdash P\ (\preceq^{\mathsf{CIU}})^H\ S$ and $\emptyset \vdash \lambda x.S\ \preceq^{\mathsf{CIU}}\ N$. Now:

$$(\mathbf{T}, \lambda x.S) = ([\cdot]R :: \mathbf{V}, \lambda x.S) \rightsquigarrow_{\mathsf{n}} (\mathbf{V}, S\{^R/_x\}). \tag{8}$$

  From $x \vdash P\ (\preceq^{\mathsf{CIU}})^H\ S$ and $\emptyset \vdash Q\ (\preceq^{\mathsf{CIU}})^H\ R$, by substitutivity of $\preceq^{\mathsf{CIU}}$, follow that $\emptyset \vdash P\{^Q/_x\}\ (\preceq^{\mathsf{CIU}})^H\ S\{^R/_x\}$ holds. By induction hypothesis, it follows that $\mathbb{C}(\mathbf{V}, S\{^R/_x\}) \geq p$. Then, from (8) and $\emptyset \vdash \lambda x.S\ \preceq^{\mathsf{CIU}}\ N$, the thesis easily follows:

$$\mathbb{C}(\mathbf{T}, N) \geq \mathbb{C}(\mathbf{T}, \lambda x.S) = \mathbb{C}(\mathbf{V}, S\{^R/_x\}) \geq p.$$

This concludes the proof. □

Theorem 5.5 *For all $\overline{x} \in \mathcal{P}_{\mathsf{FIN}}(\mathsf{X})$ and for all $M, N \in \Lambda_{\oplus}(\overline{x})$, $\overline{x} \vdash M\ \preceq^{\mathsf{CIU}}\ N$ iff $\overline{x} \vdash M \leq_{\oplus} N$.*

Proof. ($\Rightarrow$) Since $\preceq^{\mathsf{CIU}}$ is defined on open terms by taking closing term-substitutions, by Lemma 3.13 both it and $(\preceq^{\mathsf{CIU}})^H$ are closed under term-substitution. Then, it suffices to show the result for closed $\Lambda_{\oplus}$-terms: for all $M, N \in \Lambda_{\oplus}(\emptyset)$, if $\emptyset \vdash M\ \preceq^{\mathsf{CIU}}\ N$, then $\emptyset \vdash M \leq_{\oplus} N$. Since $\preceq^{\mathsf{CIU}}$ is reflexive, by Lemma 3.10 follows that $(\preceq^{\mathsf{CIU}})^H$ is compatible, hence reflexive too. Taking $\mathbf{T} = \mathbf{S}$ in Lemma 5.4, we conclude that $\emptyset \vdash M\ (\preceq^{\mathsf{CIU}})^H\ N$ implies $\emptyset \vdash M\ \preceq^{\mathsf{CIU}}\ N$. As we have remarked before the lemma, the latter entails that $(\preceq^{\mathsf{CIU}})^H = \preceq^{\mathsf{CIU}}$ which implies $\preceq^{\mathsf{CIU}}$ of being compatible. Moreover, from Lemma 5.2 immediately follows that $\preceq^{\mathsf{CIU}}$ is also adequate. Thus, $\preceq^{\mathsf{CIU}}$ is contained

in the largest compatible adequate $\Lambda_\oplus$-relation, $\leq^{\mathsf{ca}}_\oplus$. From Lemma 4.6 follows that $\preceq^{\mathsf{CIU}}$ is actually contained in $\leq_\oplus$. In particular, the latter means $\emptyset \vdash M \preceq^{\mathsf{CIU}} N$ implies $\emptyset \vdash M \leq_\oplus N$.

($\Leftarrow$) First of all, please observe that, since context preorder is compatible, if $\emptyset \vdash M \leq_\oplus N$ then, for all $\mathbf{S} \in \mathcal{FS}(\emptyset)$, $\emptyset \vdash E_{\mathbf{S}}(M) \leq_\oplus E_{\mathbf{S}}(N)$ by Lemma 3.35. Then, by adequacy property of $\leq_\oplus$ and Lemma 5.2, the latter implies $\emptyset \vdash M \preceq^{\mathsf{CIU}} N$. Ultimately, it holds that $\emptyset \vdash M \leq_\oplus N$ implies $\emptyset \vdash M \preceq^{\mathsf{CIU}} N$. Let us take into account the general case of open terms. If $\overline{x} \vdash M \leq_\oplus N$, then by compatibility property of $\leq_\oplus$ it follows $\emptyset \vdash \lambda\overline{x}.M \leq_\oplus \lambda\overline{x}.N$ and hence $\emptyset \vdash \lambda\overline{x}.M \preceq^{\mathsf{CIU}} \lambda\overline{x}.N$. Then, from the fact that $\preceq^{\mathsf{CIU}}$ is compatible (as established in ($\Rightarrow$) part of this proof) and Lemma 5.3, for every suitable $\overline{L} \subseteq \Lambda_\oplus(\emptyset)$, it holds $\emptyset \vdash M\{\overline{L}/\overline{x}\} \preceq^{\mathsf{CIU}} N\{\overline{L}/\overline{x}\}$, i.e. $\overline{x} \vdash M \preceq^{\mathsf{CIU}} N$. □

Corollary 5.6 *$\cong^{\mathsf{CIU}}$ coincides with $\simeq_\oplus$.*

Proof. Straightforward consequence of Theorem 5.5. □

Proposition 5.7 *$\leq_\oplus$ and $\lesssim$ do not coincide.*

Proof. We will prove that $M \preceq^{\mathsf{CIU}} N$ but $M \not\lesssim N$, where

$$M \stackrel{\text{def}}{=} \lambda x.\lambda y.x \oplus y;$$
$$N \stackrel{\text{def}}{=} (\lambda x.\lambda y.x) \oplus (\lambda x.\lambda y.y).$$

$M \not\lesssim N$ can be easily verified, so let us concentrate on $M \preceq^{\mathsf{CIU}} N$, and prove that for every $\mathbf{S}$, $\mathbb{C}(\mathbf{S}, M) \leq \mathbb{C}(\mathbf{S}, N)$. Let us distinguish three cases:

- If $\mathbf{S} = \mathtt{nil}$, then $(\mathbf{S}, M)$ cannot be further reduced and $(\mathbf{S}, N) \rightsquigarrow_{\mathsf{n}} (\mathbf{S}, \lambda x.\lambda y.x), (\mathbf{S}, \lambda x.\lambda y.y)$, where the last two pairs cannot be reduced. As a consequence, $\mathbb{C}(\mathbf{S}, M) = 0 = \mathbb{C}(\mathbf{S}, N)$.
- If $\mathbf{S} = [\cdot]L :: \mathbf{T}$, then we can proceed similarly.
- If $\mathbf{S} = [\cdot]L :: [\cdot]P :: \mathbf{T}$, then observe that

$$\begin{aligned}(\mathbf{S}, M) &\rightsquigarrow_{\mathsf{n}} ([\cdot]P :: \mathbf{T}, \lambda y.L \oplus y) \rightsquigarrow_{\mathsf{n}} (\mathbf{T}, L \oplus P)\\ &\rightsquigarrow_{\mathsf{n}} (\mathbf{T}, L), (\mathbf{T}, P);\\ (\mathbf{S}, N) &\rightsquigarrow_{\mathsf{n}} (\mathbf{S}, \lambda x.\lambda y.x), (\mathbf{S}, \lambda x.\lambda y.y);\\ (\mathbf{S}, \lambda x.\lambda y.x) &\rightsquigarrow_{\mathsf{n}} ([\cdot]P :: \mathbf{T}, \lambda y.L) \rightsquigarrow_{\mathsf{n}} (\mathbf{T}, L);\\ (\mathbf{S}, \lambda x.\lambda y.y) &\rightsquigarrow_{\mathsf{n}} ([\cdot]P :: \mathbf{T}, \lambda y.y) \rightsquigarrow_{\mathsf{n}} (\mathbf{T}, P).\end{aligned}$$

As a consequence,

$$\mathbb{C}(\mathbf{S}, M) = \frac{1}{2}\mathbb{C}(\mathbf{T}, L) + \frac{1}{2}\mathbb{C}(\mathbf{T}, P) = \mathbb{C}(\mathbf{S}, N).$$

This concludes the proof. □

Example 5.8 *We consider again the programs from Example 2.6. Terms* `expone` *and* `exptwo` *only differ because the former performs all probabilistic choices on natural numbers obtained by applying a function to its argument, while in the latter choices are done at the functional level, and the argument to those functions is provided only at a later stage. As a consequence, the two terms are not applicative bisimilar, and the reason is akin to that for the inequality of the terms in Example 3.38. In contrast, the bisimilarity between* `expone` *and* `expthree k`*, where* `k` *is any natural number, intuitively holds because both* `expone` *and* `expthree k` *evaluate to a single term when fed with a function, while they start evolving in a genuinely probabilistic way only after the second argument is provided. At that point, the two functions evolve in very different ways, but their semantics (in the sense of Section 2) is the same (cf., Lemma 3.4). As a bisimulation one can use the equivalence generated by the relation*

$$\Big(\bigcup_{\mathtt{k}}\{(\mathtt{expone}, \mathtt{expthree\ k})\}\Big) \cup \{(M, N) \mid [\![M]\!] = [\![N]\!]\}$$
$$\cup \Big(\bigcup_{L}\{(\lambda\mathtt{n}.\mathtt{B}\{^{L}/_{\mathtt{f}}\}, \lambda\mathtt{n}.\mathtt{C}\{^{L}/_{\mathtt{f}}\})\}\Big)$$

using `B` *and* `C` *for the body of* `expone` *and* `expthree` *respectively.*

6 The Discriminating Power of Probabilistic Contexts

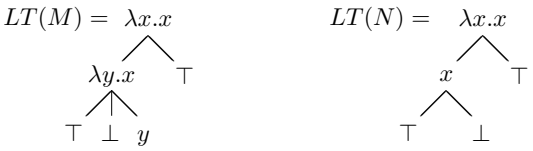
We show here that applicative bisimilarity and context equivalence collapse if the tested terms are pure, *deterministic*, λ-terms. In other words, if the probabilistic choices are brought into the terms only through the inputs supplied to the tested functions, applicative bisimilarity and context equivalence yield exactly the same discriminating power. To show this, we prove that, on pure λ-terms, both relations coincide with the *Levy-Longo tree equality*, which equates terms with the same Levy-Longo tree (briefly LLT) [14].

LLT's are the lazy variant of Böhm Trees (briefly BT), the most popular tree structure in the λ-calculus. BT's only correctly express the computational content of λ-terms in a *strong* regime, while they fail to do so in the lazy one. For instance, the term $\lambda x.\Omega$ and Ω, as both *unsolvable* [4], have identical BT's, but in a lazy regime we would always distinguish between them; hence they have different LLT's. LLT's were introduced by Longo [30], developing an original idea by Levy [29]. The *Levy-Longo tree* of M, $LT(M)$, is coinductively constructed as follows: $LT(M) \stackrel{\text{def}}{=} \lambda x_1 \dots x_n.\bot$ if M is an unsolvable of order n; $LT(M) \stackrel{\text{def}}{=} \top$ if M is an unsolvable of order ∞; finally if M has principal head normal form $\lambda x_1 \dots x_n.yM_1 \dots M_m$, then $LT(M)$ is a tree with root $\lambda x_1 \dots x_n.y$ and with $LT(M_1), \dots, LT(M_m)$ as subtrees. Being defined coinductively, LLT's can of course be infinite. We write $M =_{\text{LL}} N$ iff $LT(M) = LT(N)$.

Example 6.1 *Let Ξ be an unsolvable of order ∞ such as $\Xi \stackrel{\text{def}}{=} (\lambda x.\lambda y.(xx))(\lambda x.\lambda y.(xx))$, and consider the terms*

$$M \stackrel{\text{def}}{=} \lambda x.(x(\lambda y.(x\Xi\Omega y))\Xi); \qquad N \stackrel{\text{def}}{=} \lambda x.(x(x\Xi\Omega)\Xi).$$

These terms have been used to prove non-full-abstraction results in a canonical model for the lazy λ-calculus by Abramsky and Ong [2]. For this, they show that in the model the convergence test is definable (this operator, when it receives an argument, would return the identity function if the supplied argument is convergent, and would diverge otherwise). The convergence test, ∇, can distinguish between the two terms, as $M\nabla$ reduces to an abstraction, whereas $N\nabla$ diverges. However, no pure λ-term can make the same distinction. The two terms also have different LL trees:



Although in $\Lambda_\oplus$, as in Λ, the convergence test operator is not definable, M and N can be separated using probabilities by running them in a context C that would feed $\Omega \oplus \lambda z.\lambda u.z$ as argument; then $C[M]\Downarrow_{\frac{1}{2}}$ whereas $C[N]\Downarrow_{\frac{1}{4}}$.

Example 6.2 *Abramsky's canonical model is itself coarser than LLT equality. For instance, the terms $M \stackrel{\text{def}}{=} \lambda x.xx$ and $N \stackrel{\text{def}}{=} \lambda x.(x\lambda y.(xy))$, have different LLT's but are equal in Abramsky's model (and hence equal for context equivalence in Λ). They are separated by context equivalence in $\Lambda_\oplus$, for instance using the context $C \stackrel{\text{def}}{=} [\cdot](I \oplus \Omega)$, since $C[M]\Downarrow_{\frac{1}{4}}$ whereas $C[N]\Downarrow_{\frac{1}{2}}$.*

We already know that on full $\Lambda_\oplus$, applicative bisimilarity ($\sim$) implies context equivalence ($\simeq_\oplus$). Hence, to prove that *on pure λ-terms* the two equivalences collapse to LLT equality ($=_{\text{LL}}$), it suffices to prove that, for those pure terms, $\simeq_\oplus$ implies $=_{\text{LL}}$, and that $=_{\text{LL}}$ implies $\sim$.

The first implication is obtained by a variation on the Böhm-out technique, a powerful methodology for separation results in the λ-calculus, often employed in proofs about local structure

characterisation theorems of λ-models. For this we exploit an inductive characterisation of LLT equality via stratification approximants (Definition 6.5). The key Lemma 6.7 shows that any difference on the trees of two λ-terms within level n can be observed by a suitable context of the probabilistic λ-calculus.

We write $\uplus M$ as an abbreviation for the term $\Omega \oplus M$. We denote by Q_n, $n > 0$, the term $\lambda x_1 . \ldots \lambda x_n . x_n x_1 x_2 \cdots x_{n-1}$. This is usually called the *Böhm permutator* of degree n. Böhm permutators play a key role in the Böhm-out technique. A variant of them, the $\uplus$-permutators, play a pivotal role in Lemma 6.7 below. A term $M \in \Lambda_\oplus$ is a *$\uplus$-permutator of degree n* if either $M = Q_n$ or there exists $0 \leq r < n$ such that

$$M = \lambda x_1 . \ldots \lambda x_r . \uplus \lambda x_{r+1} \cdots \lambda x_n . x_n x_1 \cdots x_{n-1} \, .$$

Finally, a function f from the positive integers to λ-terms is a *$\uplus$-permutator function* if, for all n, $f(n)$ is a $\uplus$-permutator of degree n. Before giving the main technical lemma, it is useful some auxiliary concepts. The definitions below rely on two notions of reduction: $M \longrightarrow_p N$ means that M call-by-name reduces to N in one step with probability p. (As a matter of fact, p can be either 1 or $\frac{1}{2}$.) Then $\Longrightarrow$ is obtained by composing $\longrightarrow$ zero or more times (and multiplying the corresponding real numbers). If $p = 1$ (because, e.g., we are dealing with pure λ-terms) $\Longrightarrow_p$ can be abbreviated just as $\Longrightarrow$. With a slight abuse of notation, we also denote with $\Longrightarrow$ the multi-step lazy reduction relation of pure, *open* terms. The specialised form of probabilistic choice $\uplus M$ can be thought of as a new syntactic construct. Thus $\Lambda^{\uplus}$ is the set of pure λ-terms extended with the $\uplus$ operator. As $\uplus$ is a derived operator, its operational rules are the expected ones:

$$\frac{}{\uplus M \longrightarrow_{\frac{1}{2}} \Omega} \; \uplus L \qquad \frac{}{\uplus M \longrightarrow_{\frac{1}{2}} M} \; \uplus R$$

The restriction on $\Longrightarrow$ in which $\uplus R$, but not $\uplus L$, can be applied, is called $\Rrightarrow$. In the following, we need the following lemma:

Lemma 6.3 *Let M, N, L, P be closed $\Lambda^{\uplus}$ terms. Suppose $\sum[\![M]\!] = \sum[\![N]\!]$, that $M \Rrightarrow_p L$ and $N \Rrightarrow_p P$. Then also $\sum[\![L]\!] = \sum[\![P]\!]$.*

Proof. Of course, $p = \frac{1}{2^n}$ for some integer $n \in \mathbb{N}$. Then $\sum[\![L]\!] = 2^n \sum[\![M]\!] = 2^n \sum[\![N]\!] = \sum[\![P]\!]$. □

The proof of the key Lemma 6.7 below makes essential use of a characterization of $=_{\mathrm{LL}}$ by a bisimulation-like form of relation:

Definition 6.4 (Open Bisimulation) *A relation $\mathcal{R}$ on pure λ-terms is an* open bisimulation *if $M \, \mathcal{R} \, N$ implies:*

1. *if $M \Longrightarrow \lambda x.L$, then $N \Longrightarrow \lambda x.P$ and $L \, \mathcal{R} \, P$;*
2. *if $M \Longrightarrow x L_1 \cdots L_m$, then $P_1, \ldots, P_m$ exist such that $N \Longrightarrow x P_1 \cdots P_m$ and $L_i \, \mathcal{R} \, P_i$ for every $1 \leq i \leq m$;*

and conversely on reductions from N. Open bisimilarity*, written $\sim^{\mathrm{O}}$, is the union of all open bisimulations.*

Open bisimulation has the advantage of very easily providing a notion of approximation:

Definition 6.5 (Approximants of $\sim^{\mathrm{O}}$) *We set:*

- $\sim^{\mathrm{O}}_0 \stackrel{\mathrm{def}}{=} \Lambda \times \Lambda$;
- $M \sim^{\mathrm{O}}_{n+1} N$ *when*
  1. *if $M \Longrightarrow \lambda x.L$, then P exists such that $N \Longrightarrow \lambda x.P$ and $L \sim^{\mathrm{O}}_n P$;*
  2. *if $M \Longrightarrow x L_1 \cdots L_m$, then $P_1, \ldots, P_m$ exist such that $N \Longrightarrow x P_1 \cdots P_m$ and $L_i \sim^{\mathrm{O}}_n P_i$, for each $1 \leq i \leq m$;*

  and conversely on the reductions from N.

Please observe that:

Lemma 6.6 *On pure λ-terms, the relations $=_{\mathrm{LL}}$, $\sim^{\mathrm{O}}$ and $(\bigcap_{n\in\mathbb{N}} \sim^{\mathrm{O}}_n)$ all coincide.*

We are now ready to state and prove the key technical lemma:

Lemma 6.7 *Suppose $M \not\sim^{\mathrm{O}}_n N$ for some n, and let $\{x_1, \ldots, x_r\}$ be the free variables in M, N. Then there are integers $m_{x_1}, \ldots, m_{x_r}$ and k, and permutator functions $f_{x_1}, \ldots, f_{x_r}$ such that, for all $m > k$, there are closed terms $\overline{R_m}$ such that the following holds: if $M\{f_{x_1}(m+m_{x_1})/x_1\}\ldots\{f_{x_r}(m+m_{x_r})/x_r\}\overline{R_m}\Downarrow_r$ and $N\{f_{x_1}(m+m_{x_1})/x_1\}\ldots\{f_{x_r}(m+m_{x_r})/x_r\}\overline{R_m}\Downarrow_s$, then $r \neq s$.*

Proof. The proof proceeds by induction on the least n such that $M \not\sim^{\mathrm{O}}_n N$. For any term M, $M^{\overline{f}}$ will stand for $M\{f_{x_1}(m+m_{x_1})/x_1\}\ldots\{f_{x_r}(m+m_{x_r})/x_r\}$ where $x_1 \ldots x_r$ are the free variables in M. We also write Ω^m for a sequence of m occurrences of Ω: so, e.g., $M\Omega^3$ is $M\Omega\Omega\Omega$. Finally, for any term M, we write $M \Uparrow$ to denote the fact that M does not converge.

- **Basic case.** $M \not\sim^{\mathrm{O}}_1 N$. There are a few cases to consider (their symmetric ones are analogous).
  - The case where only one of the two terms diverges is easy.
  - $M \Longrightarrow xM_1 \cdots M_t$ and $N \Longrightarrow xN_1 \cdots N_s$ with $t < s$. Take $m_x = s$ and $f_x(n) = Q_n$ (the Böhm permutator of degree n). The values of the other integers (k, m_y for $y \neq x$) and of the other permutation functions are irrelevant. Set $\overline{R_m} \stackrel{\mathrm{def}}{=} \Omega^m$. We have
  $$M^{\overline{f}}\Omega^m \Longrightarrow Q_{m+s}M_1^{\overline{f}} \ldots M_t^{\overline{f}}\Omega^m \Downarrow_1$$
  since $t + m < s + m$. We also have
  $$N^{\overline{f}}\Omega^m \Longrightarrow Q_{m+s}N_1^{\overline{f}} \ldots N_s^{\overline{f}}\Omega^m \Uparrow$$
  since $m > 0$ and therefore an Ω term will be end up at the head of the term.
  - $M \Longrightarrow xM_1 \cdots M_t$ and $N \Longrightarrow yN_1 \cdots N_s$ with $x \neq y$. Assume $t \leq s$ without loss of generality. Take $m_x = s + 1$, $m_y = s$, and $f_x(n) = f_y(n) = Q_n$. The values of the other integers and permutation functions are irrelevant. Set $\overline{R_m} \stackrel{\mathrm{def}}{=} \Omega^m$. We have
  $$M^{\overline{f}}\Omega^m \Longrightarrow Q_{m+s+1}M_1^{\overline{f}} \ldots M_t^{\overline{f}}\Omega^m \Downarrow_1$$
  since $m + s + 1 > t + m$. We also have
  $$N^{\overline{f}}\Omega^m \Longrightarrow Q_{m+s}N_1^{\overline{f}} \ldots N_s^{\overline{f}}\Omega^m \Uparrow$$
  since $m > 0$ and therefore an Ω term will be end up at the head of the term.
  - $M \Longrightarrow \lambda x.M'$ and $N \Longrightarrow y\overline{N}$, for some y and $\overline{N}$. The values of the integers and permutator functions are irrelevant. Set $\overline{R_m} \stackrel{\mathrm{def}}{=} \emptyset$ (the empty sequence), and $f_y(n) \stackrel{\mathrm{def}}{=} \uplus Q_n$. We have $M^{\overline{f}} \Longrightarrow \lambda x.{M'}^{\overline{f}} \Downarrow_1$, whereas
  $$N^{\overline{f}} \Longrightarrow \uplus Q_{m+m_y}\overline{N^{\overline{f}}} \Downarrow_{<1}$$
- **Inductive case:** $M \not\sim^{\mathrm{O}}_{n+1} N$. There are two cases to look at.
  - $M \Longrightarrow xM_1 \cdots M_s$, $N \Longrightarrow xN_1 \cdots N_s$ and for some i, $M_i \not\sim^{\mathrm{O}}_n N_i$. By induction, (for all variables y) there are integers m_y, k and permutator functions f_y, such that for all $m > k$ there are $\overline{S_m}$ and we have $\sum[\![M_i^{\overline{f}}\overline{S_m}]\!] \neq \sum[\![N_i^{\overline{f}}\overline{S_m}]\!]$. Redefine k if necessary so to make sure that $k > s$. Set $\overline{R_m} \stackrel{\mathrm{def}}{=} \Omega^{m+m_x-s-1}(\lambda x_1 \ldots x_{m+m_x}.x_i)\overline{S_m}$. We have:
  $$M^{\overline{f}}\overline{R_m} \Longrightarrow f_x(m+m_x)M_1^{\overline{f}} \ldots M_s^{\overline{f}}\Omega^{m+m_x-s-1}(\lambda x_1 \ldots x_{m+m_x}.x_i)\overline{S_m} \Longrightarrow_p M_i^{\overline{f}}\overline{S_m}$$
  whereas
  $$N^{\overline{f}}\overline{R_m} \Longrightarrow f_x(m+m_x)N_1^{\overline{f}} \ldots N_s^{\overline{f}}\Omega^{m+m_x-s-1}(\lambda x_1 \ldots x_{m+m_x}.x_i)\overline{S_m} \Longrightarrow_p N_i^{\overline{f}}\overline{S_m}$$
  where p is $\frac{1}{2}$ or 1 depending on whether f_x contains $\uplus$ or not. In any case, in both derivations, rule $\uplus L$ has not been used. By Lemma 6.3 and the inductive assumption $\sum[\![M_i^{\overline{f}}\overline{S_m}]\!] \neq \sum[\![N_i^{\overline{f}}\overline{S_m}]\!]$ we derive that $\sum[\![M^{\overline{f}}\overline{R_m}]\!] \neq \sum[\![N^{\overline{f}}\overline{R_m}]\!]$ too.

- $M \Longrightarrow \lambda x.M'$, $N \Longrightarrow \lambda x.N'$ and $M' \not\sim_n^{\mathrm{O}} N'$. By induction, (for all variables y) there are integers m_y, k and permutator functions f_y, such that for all $m > k$ there are $\overline{S_m}$ and we have $\sum[\![M'^{\overline{f}}\overline{S_m}]\!] \neq \sum[\![N'^{\overline{f}}\overline{S_m}]\!]$. Set $\overline{R_m} \stackrel{\mathrm{def}}{=} f_x(m+m_x)\overline{S_m}$. Below for a term L, $L^{\overline{f-x}}$ is defined as $L^{\overline{f}}$ except that variable x is left uninstantiated. We have:

$$M^{\overline{f}}\overline{R_m} \Longrightarrow (\lambda x.M'^{\overline{f-x}})f_x(m+m_x)\overline{S_m} \longrightarrow (M'^{\overline{f-x}}\{f_x(m+m_x)/x\})\overline{S_m} = M'^{\overline{f}}\overline{S_m}$$

whereas

$$N^{\overline{f}}\overline{R_m} \Longrightarrow (\lambda x.N'^{\overline{f-x}})f_x(m+m_x)\overline{S_m} \longrightarrow (N'^{\overline{f-x}}\{f_x(m+m_x)/x\})\overline{S_m} = N'^{\overline{f}}\overline{S_m}$$

Again, by Lemma 6.3 and the inductive hypothesis, we derive $\sum[\![M^{\overline{f}}\overline{R_m}]\!] \neq \sum[\![N^{\overline{f}}\overline{R_m}]\!]$. This concludes the proof. □

The fact the Böhm-out technique actually works implies that the discriminating power of probabilistic contexts is at least as strong as the one of LLT's.

Corollary 6.8 *For $M, N \in \Lambda$, $M \simeq_{\oplus} N$ implies $M =_{\mathrm{LL}} N$.*

To show that LLT equality is included in probabilistic applicative bisimilarity, we proceed as follows. First we define a refinement of the latter, essentially one in which we *observe* all probabilistic choices. As a consequence, the underlying bisimulation game may ignore probabilities. The obtained notion of equivalence is strictly finer than probabilistic applicative bisimilarity. The advantage of the refinement is that *both* the inclusion of LLT equality in the refinement, and the inclusion of the latter in probabilistic applicative bisimilarity turn out to be relatively easy to prove. A *direct* proof of the inclusion of LLT equality in probabilistic applicative bisimilarity would have been harder, as it would have required extending the notion of a Levy-Longo tree to $\Lambda_{\oplus}$, then reasoning on substitution closures of such trees.

Definition 6.9 *A relation $\mathcal{R} \subseteq \Lambda_{\oplus}(\emptyset) \times \Lambda_{\oplus}(\emptyset)$ is a* strict applicative bisimulation *whenever $M \mathcal{R} N$ implies*
1. *if $M \longrightarrow_1 P$, then $N \Longrightarrow_1 Q$ and $P \mathcal{R} Q$;*
2. *if $M \longrightarrow_{\frac{1}{2}} P$, then $N \Longrightarrow_{\frac{1}{2}} Q$ and $P \mathcal{R} Q$;*
3. *if $M = \lambda x.P$, then $N \Longrightarrow_1 \lambda x.Q$ and $P\{^L/_x\} \mathcal{R} Q\{^L/_x\}$ for all $L \in \Lambda_{\oplus}(\emptyset)$;*
4. *the converse of 1., 2. and 3..*

Strict applicative bisimilarity *is the union of all strict applicative bisimulations.*

If two terms have the same LLT, then passing them the same argument $M \in \Lambda_{\oplus}$ produces *exactly* the same choice structure: intuitively, whenever the first term finds (a copy of) M in head position, also the second will find M.

Lemma 6.10 *If $M =_{\mathrm{LL}} N$ then $M \mathcal{R} N$, for some strict applicative bisimulation $\mathcal{R}$.*

Terms which are strict applicative bisimilar cannot be distinguished by applicative bisimilarity proper, since the requirements induced by the latter are less strict than the ones the former imposes:

Lemma 6.11 *Strict applicative bisimilarity is included in applicative bisimilarity.*

Since we now know that for *pure, deterministic* λ-terms, $=_{\mathrm{LL}}$ is included in $\sim$ (by Lemma 6.10 and Lemma 6.11), that $\sim$ is included in $\simeq_{\oplus}$ (by Theorem 3.37) and that the latter is included in $=_{\mathrm{LL}}$ (Corollary 6.8), we can conclude:

Corollary 6.12 *The relations $=_{\mathrm{LL}}$, $\sim$, and $\simeq_{\oplus}$ coincide in Λ.*

7 Coupled Logical Bisimulation

In this section we derive a coinductive characterisation of probabilistic context equivalence on the whole language $\Lambda_\oplus$ (as opposed to the subset of sum-free λ-terms as in Section 6). For this, we need to manipulate formal weighted sums. Thus we work with an extension of $\Lambda_\oplus$ in which such weighted sums may appear in redex position. An advantage of having formal sums is that the transition system on the extended language can be small-step and deterministic — any closed term that is not a value will have exactly one possible internal transition.

This will make it possible to pursue the *logical bisimulation* method, in which the congruence of bisimilarity is proved using a standard induction argument over all contexts. The refinement of the method handling probabilities, called *coupled* logical bisimulation, uses *pairs* of relations, as we need to distinguish between ordinary terms and terms possibly containing formal sums. Technically, in the proof of congruence we first prove a correspondence between the transition system on extended terms and the original one for $\Lambda_\oplus$; we then derive a few up-to techniques for coupled logical bisimulations that are needed in the following proofs; finally, we show that coupled logical bisimulations are preserved by the closure of the first relation with any context, and the closure of the second relation with any *evaluation* context.

We preferred to follow logical bisimulations rather then environmental bisimulations because the former admit a simpler definition (in the latter, each pair of terms is enriched with an environment, that is, an extra set of pairs of terms). Moreover it is unclear what environments should be when one also considers formal sums. We leave this for future work.

Formal sums are a tool for representing the behaviour of running $\Lambda_\oplus$ terms. Thus, on terms with formal sums, only the results for closed terms interest us. However, the characterization of contextual equivalence in $\Lambda_\oplus$ as coupled logical bisimulation also holds on open terms.

7.1 Notation and Terminology

We write $\Lambda_\oplus^{\mathrm{FS}}$ for the extension of $\Lambda_\oplus$ in which formal sums may appear in redex position. Terms of $\Lambda_\oplus^{\mathrm{FS}}$ are defined as follows (M, N being $\Lambda_\oplus$-terms):

$$E, F ::= EM \;\Big|\; \Sigma_{i\in I}\langle M_i, p_i\rangle \;\Big|\; M \oplus N \;\Big|\; \lambda x.M.$$

In a formal sum $\Sigma_{i\in I}\langle M_i, p_i\rangle$, I is a countable (possibly empty) set of indices such that $\sum_{i\in I} p_i \leq 1$. We use $+$ for binary formal sums. Formal sums are ranged over by metavariables like H, K. When each M_i is a value (i.e., an abstraction) then $\Sigma_{i\in I}\langle M_i, p_i\rangle$ is a (*formally summed*) *value*; such values are ranged over by Z, Y, X. If $H = \Sigma_{i\in I}\langle M_i, p_i\rangle$ and $K = \Sigma_{j\in J}\langle M_j, p_j\rangle$ where I and J are disjoint, then $H \oplus K$ abbreviates $\Sigma_{r\in I\cup J}\langle M_r, \frac{p_r}{2}\rangle$. Similarly, if for every $j \in J$ H_j is $\Sigma_{i\in I}\langle M_{i,j}, p_{i,j}\rangle$, then $\Sigma_j\langle H_j, p_j\rangle$ stands for $\Sigma_{(i,j)}\langle M_{i,j}, p_{i,j}\cdot p_j\rangle$. For $H = \Sigma_i\langle M_i, p_i\rangle$ we write $\mathbf{\Sigma}(H)$ for the real number $\sum_i p_i$. If $Z = \Sigma_i\langle \lambda x.M_i, p_i\rangle$, then $Z \bullet N$ stands for $\Sigma_i\langle M_i\{^N/_x\}, p_i\rangle$. The set of closed terms is $\Lambda_\oplus^{\mathrm{FS}}(\emptyset)$.

Any partial value distribution $\mathscr{D}$ (in the sense of Section 2) can be seen as the formal sum $\Sigma_{V\in\mathsf{V}\Lambda_\oplus}\langle V, \mathscr{D}(V)\rangle$. Similarly, any formal sum $H = \Sigma_{i\in I}\langle M_i, p_i\rangle$ can be mapped to the distribution $\sum_{i\in I} p_i \cdot [\![M_i]\!]$, that we indicate with $[\![H]\!]$.

Reduction between $\Lambda_\oplus^{\mathrm{FS}}$ terms, written $E \rightsquigarrow F$, is defined by the rules in Figure 6; these rules are given on top of the operational semantics for $\Lambda_\oplus$ as defined in Section 2, which is invoked in the premise of rule spc (if there is a i with M_i not a value). The reduction relation $\rightsquigarrow$ is deterministic and strongly normalizing. We use $\twoheadrightarrow$ for its reflexive and transitive closure. Lemma 7.1 shows the agreement between the new reduction relation and the original one.

Lemma 7.1 *For all $M \in \Lambda_\oplus(\emptyset)$ there is a value Z such that $M \twoheadrightarrow Z$ and $[\![M]\!] = [\![Z]\!]$.*

Proof. One first show that for all E there is n such that $E \rightsquigarrow^n Z$. Then one reasons with a double induction: an induction on n, and a transition induction, exploiting the determinism of $\rightsquigarrow$. □

$$\frac{}{M \oplus N \rightsquigarrow \langle M, \frac{1}{2}\rangle + \langle N, \frac{1}{2}\rangle}\ \mathsf{ss}$$

$$\frac{}{\lambda x.M \rightsquigarrow \langle \lambda x.M, 1\rangle}\ \mathsf{sl} \qquad \frac{[\![M_i]\!] = \mathscr{D}_i}{\Sigma_i\langle M_i, p_i\rangle \rightsquigarrow \Sigma_i\langle \mathscr{D}_i, p_i\rangle}\ \mathsf{spc}$$

$$\frac{}{ZM \rightsquigarrow Z \bullet M}\ \mathsf{sp} \qquad \frac{E \rightsquigarrow F}{EM \rightsquigarrow FM}\ \mathsf{sa}$$

Figure 6: Reduction Rules for $\Lambda^{\mathrm{FS}}_{\oplus}$

7.2 Context Equivalence and Bisimulation

In $\Lambda^{\mathrm{FS}}_{\oplus}$ certain terms (i.e., formal sums) may only appear in redex position; ordinary terms (i.e., terms in $\Lambda_{\oplus}$), by contrast, may appear in arbitrary position. When extending context equivalence to $\Lambda^{\mathrm{FS}}_{\oplus}$ we therefore have to distinguish these two cases. Moreover, as our main objective is the characterisation of context equivalence in $\Lambda_{\oplus}$, we set a somewhat constrained context equivalence in $\Lambda^{\mathrm{FS}}_{\oplus}$ in which contexts may not contain formal sums (thus the $\Lambda^{\mathrm{FS}}_{\oplus}$ contexts are the same as the $\Lambda_{\oplus}$ contexts). We call these *simple* $\Lambda^{\mathrm{FS}}_{\oplus}$ *contexts*, whereas we call *general* $\Lambda^{\mathrm{FS}}_{\oplus}$ *context* an unconstrained context, i.e., a $\Lambda^{\mathrm{FS}}_{\oplus}$ term in which the hole $[\cdot]$ may appear in any places where a term from $\Lambda_{\oplus}$ was expected — including within a formal sum. (Later we will see that allowing general contexts does not affect the resulting context equivalence.) Terms possibly containing formal sums are tested in evaluation contexts, i.e., contexts of the form $[\cdot]\overline{M}$. We write $E{\downarrow\downarrow_p}$ if $E \twoheadrightarrow Z$ and $\mathbf{\Sigma}(Z) = p$ (recall that Z is unique, for a given E).

Definition 7.2 (Context Equivalence in $\Lambda^{\mathrm{FS}}_{\oplus}$) *Two $\Lambda_{\oplus}$-terms M and N are* context equivalent *in $\Lambda^{\mathrm{FS}}_{\oplus}$, written $M \simeq^{\mathrm{FS}}_{\oplus} N$, if for all (closing) simple $\Lambda^{\mathrm{FS}}_{\oplus}$ contexts C, we have $C[M]{\downarrow\downarrow_p}$ iff $C[N]{\downarrow\downarrow_p}$. Two $\Lambda^{\mathrm{FS}}_{\oplus}$-terms E and F are* evaluation-context equivalent*, written $E \approx^{\mathrm{FS}}_{\oplus} F$, if for all (closing) $\Lambda^{\mathrm{FS}}_{\oplus}$ evaluation contexts C, we have $C[E]{\downarrow\downarrow_p}$ iff $C[F]{\downarrow\downarrow_p}$.*

In virtue of Lemma 7.1, context equivalence in $\Lambda_{\oplus}$ coincides with context equivalence in $\Lambda^{\mathrm{FS}}_{\oplus}$.

We now introduce a bisimulation that yields a coinductive characterisation of context equivalence (and also of evaluation-context equivalence). A *coupled relation* is a pair $(\mathcal{V}, \mathcal{E})$ where: $\mathcal{V} \subseteq \Lambda_{\oplus}(\emptyset) \times \Lambda_{\oplus}(\emptyset)$, $\mathcal{E} \subseteq \Lambda^{\mathrm{FS}}_{\oplus}(\emptyset) \times \Lambda^{\mathrm{FS}}_{\oplus}(\emptyset)$, and $\mathcal{V} \subseteq \mathcal{E}$. Intuitively, we place in $\mathcal{V}$ the pairs of terms that should be preserved by all contexts, and in $\mathcal{E}$ those that should be preserved by evaluation contexts. For a coupled relation $\mathcal{R} = (\mathcal{V}, \mathcal{E})$ we write $\mathcal{R}_1$ for $\mathcal{V}$ and $\mathcal{R}_2$ for $\mathcal{E}$. The union of coupled relations is defined componentwise: e.g., if $\mathcal{R}$ and SS are coupled relations, then the coupled relation $\mathcal{R} \cup SS$ has $(\mathcal{R} \cup SS)_1 \stackrel{\mathrm{def}}{=} \mathcal{R}_1 \cup SS_1$ and $(\mathcal{R} \cup SS)_2 \stackrel{\mathrm{def}}{=} \mathcal{R}_2 \cup SS_2$. If $\mathcal{V}$ is a relation on $\Lambda_{\oplus}$, then $\mathcal{V}^{\mathrm{C}}$ is the context closure of $\mathcal{V}$ in $\Lambda_{\oplus}$, i.e., the set of all (closed) terms of the form $(C[\overline{M}], C[\overline{N}])$ where C is a multi-hole $\Lambda_{\oplus}$ context and $\overline{M}\ \mathcal{V}\ \overline{N}$.

Definition 7.3 *A coupled relation $\mathcal{R}$ is a* coupled logical bisimulation *if whenever $E\ \mathcal{R}_2\ F$ we have:*

1. *if $E \rightsquigarrow D$, then $F \twoheadrightarrow G$, where $D\ \mathcal{R}_2\ G$;*
2. *if E is a formally summed value, then $F \twoheadrightarrow Y$ with $\mathbf{\Sigma}(E) = \mathbf{\Sigma}(Y)$, and for all $M\ \mathcal{R}_1^{\mathrm{C}}\ N$ we have $(E \bullet M)\ \mathcal{R}_2\ (Y \bullet N)$;*
3. *the converse of 1. and 2..*

Coupled logical bisimilarity, $\approx$*, is the union of all coupled logical bisimulations (hence $\approx_1$ is the union of the first component of all coupled logical bisimulations, and similarly for $\approx_2$).*

In a coupled bisimulation $(\mathcal{R}_1, \mathcal{R}_2)$, the bisimulation game is only played on the pairs in $\mathcal{R}_2$. However, the first relation $\mathcal{R}_1$ is relevant, as inputs for tested functions are built using $\mathcal{R}_1$ (Clause 2. of Definition 7.3). Actually, also the pairs in $\mathcal{R}_1$ are tested, because in any coupled relations it must be $\mathcal{R}_1 \subseteq \mathcal{R}_2$. The values produced by the bisimulation game for coupled bisimulation on $\mathcal{R}_2$ are formal sums (not plain λ-terms), and this is why we do not require them to be in $\mathcal{R}_1$: formal

sums should only appear in redex position, but terms in $\mathcal{R}_1$ can be used as arguments to bisimilar functions and can therefore end up in arbitrary positions.

We will see below another aspect of the relevance of $\mathcal{R}_1$: the proof technique of logical bisimulation only allows us to prove substitutivity of the bisimilarity in arbitrary contexts *for the pairs of terms* in $\mathcal{R}_1$. For pairs in $\mathcal{R}_2$ but not in $\mathcal{R}_1$ the proof technique only allows us to derive preservation in evaluation contexts.

In the proof of congruence of coupled logical bisimilarity we will push "as many terms as possible" into the first relation, i.e., the first relation will be as large as possible. However, in proofs of bisimilarity for concrete terms, the first relation may be very small, possibly a singleton or even empty. Then the bisimulation clauses become similar to those of applicative bisimulation (as inputs of tested function are "almost" identical). Summing up, in coupled logical bisimulation the use of two relations gives us more flexibility than in ordinary logical bisimulation: depending on the needs, we can tune the size of the first relation. It is possible that some of the above aspects of coupled logical bisimilarity be specific to call-by-name, and that the call-by-value version would require non-trivial modifications.

Remark 7.4 *In a coupled logical bisimulation, the first relation is used to construct the inputs for the tested functions (the formally summed values produced in the bisimulation game for the second relation). Therefore, such first relation may be thought of as a "global" environment— global because it is the same for each pair of terms on which the bisimulation game is played. As a consequence, coupled logical bisimulation remains quite different from environmental bisimulation [45], where the "environment" for constructing inputs is local to each pair of tested terms. Coupled logical bisimulation follows ordinary logical bisimulation [44], in which there is only one global environment; in ordinary logical bisimulation, however, the global environment coincides with the set of tested terms. The similarity with logical bisimulation is also revealed by non-monotonicity of the associated functional (in contrast, the functional associated to environmental bisimulation is monotone); see Remark 7.17.*

As an example of use of coupled logical bisimulation, we revisit the counterexample 3.38 to the completeness of applicative bisimilarity with respect to contextual equivalence.

Example 7.5 *We consider the terms of Example 3.38 and show that they are in $\approx_1$, hence also in $\simeq_\oplus$ (contextual equivalence of $\Lambda_\oplus$), by Corollary 7.12 and $\simeq_\oplus^{\mathrm{FS}} = \simeq_\oplus$. Recall that the terms are $M \stackrel{\text{def}}{=} \lambda x.(L \oplus P)$ and $N \stackrel{\text{def}}{=} (\lambda x.L) \oplus (\lambda x.P)$ for $L \stackrel{\text{def}}{=} \lambda z.\Omega$ and $P \stackrel{\text{def}}{=} \lambda y.\lambda z.\Omega$. We set $\mathcal{R}_1$ to contain only (M, N) (this is the pair that interests us), and R_2 to contain the pairs $(M, N), (\langle M, 1\rangle, \langle \lambda x.L, \frac{1}{2}\rangle + \langle \lambda x.P, \frac{1}{2}\rangle), (\langle L + P, 1\rangle, \langle L, \frac{1}{2}\rangle + \langle P, \frac{1}{2}\rangle)$, and a set of pairs with identical components, namely $(\langle L, \frac{1}{2}\rangle + \langle P, \frac{1}{2}\rangle, \langle L, \frac{1}{2}\rangle + \langle P, \frac{1}{2}\rangle)$, $(\langle \Omega, \frac{1}{2}\rangle + \langle \lambda u.\Omega, \frac{1}{2}\rangle, \langle \Omega, \frac{1}{2}\rangle + \langle \lambda u.\Omega, \frac{1}{2}\rangle)$, $(\langle \lambda u.\Omega, \frac{1}{2}\rangle, \langle \lambda u.\Omega, \frac{1}{2}\rangle)$, $(\langle \Omega, \frac{1}{2}\rangle, \langle \Omega, \frac{1}{2}\rangle)$, $(\emptyset, \emptyset)$, where $\emptyset$ is the empty formal sum. Thus $(\mathcal{R}_1, \mathcal{R}_2)$ is a coupled logical bisimulation.*

The main challenge towards the goal of relating coupled logical bisimilarity and context equivalence is the substitutivity of bisimulation. We establish the latter exploiting some *up-to techniques* for bisimulation. We only give the definitions of the techniques, omitting the statements about their soundness. The first up-to technique allows us to drop the bisimulation game on silent actions:

Definition 7.6 (Big-Step Bisimulation) *A coupled relation $\mathcal{R}$ is a* big-step coupled logical bisimulation *if whenever $E\ \mathcal{R}_2\ F$, the following holds: if $E \twoheadrightarrow Z$ then $F \twoheadrightarrow Y$ with $\mathbf{\Sigma}(Z) = \mathbf{\Sigma}(Y)$, and for all $M\ \mathcal{R}_1^{\mathrm{C}}\ N$ we have $(Z \bullet M)\ \mathcal{R}_2\ (Y \bullet N)$.*

Lemma 7.7 *If $\mathcal{R}$ is a big-step coupled logical bisimulation, then $\mathcal{R} \subseteq SS$ for some coupled logical bisimulation SS.*

In the reduction $\rightsquigarrow$, computation is performed at the level of formal sums; and this is reflected, in coupled bisimulation, by the application of values to formal sums only. The following up-to technique allows computation, and application of input values, also with ordinary terms. In the

definition, we extract a formal sum from a term E in $\Lambda^{\mathrm{FS}}_{\oplus}$ using the function $\mathcal{D}(\cdot)$ inductively as follows:

$$\mathcal{D}(EM) \stackrel{\text{def}}{=} \Sigma_i \langle M_i M, p_i \rangle \text{ whenever } \mathcal{D}(E) = \Sigma_i \langle M_i, p_i \rangle;$$
$$\mathcal{D}(M) \stackrel{\text{def}}{=} \langle M, 1 \rangle; \qquad \mathcal{D}(H) \stackrel{\text{def}}{=} H.$$

Definition 7.8 *A coupled relation $\mathcal{R}$ is a* bisimulation up-to formal sums *if, whenever $E\ \mathcal{R}_2\ F$, then either (one of the bisimulation clauses of Definition 7.3 applies), or $(E, F \in \Lambda_{\oplus}$ and one of the following clauses applies):*

1. *$E \rightsquigarrow D$ with $\mathcal{D}(D) = \langle M, \frac{1}{2} \rangle + \langle N, \frac{1}{2} \rangle$, and $F \rightsquigarrow G$ with $\mathcal{D}(G) = \langle L, \frac{1}{2} \rangle + \langle P, \frac{1}{2} \rangle$, $M\ \mathcal{R}_2\ L$, and $N\ \mathcal{R}_2\ P$;*
2. *$E = \lambda x.M$ and $F = \lambda x.N$, and for all $P\ \mathcal{R}_1^{\mathrm{C}}\ Q$ we have $M\{^P/_x\}\ \mathcal{R}_2\ N\{^Q/_x\}$;*
3. *$E = (\lambda x.M)P\overline{M}$ and $F = (\lambda x.N)Q\overline{N}$, and $M\{^P/_x\}\overline{M}\ \mathcal{R}_2\ N\{^Q/_x\}\overline{N}$.*

According to Definition 7.8, in the bisimulation game for a coupled relation, given a pair $(E, F) \in \mathcal{R}_2$, we can either choose to follow the bisimulation game in the original Definition 7.3; or, if E and F do not contain formal sums, we can try one of the new clauses above. The advantage of the first new clause is that it allows us to make a split on the derivatives of the original terms. The advantage of the other two new clauses is that they allow us to directly handle the given λ-terms, without using the operational rules of Figure 6 and therefore without introducing formal sums. To understand the first clause, suppose $E \stackrel{\text{def}}{=} (M \oplus N)L$ and $F \stackrel{\text{def}}{=} P \oplus Q$. We have $E \rightsquigarrow (\langle M, \frac{1}{2} \rangle + \langle N, \frac{1}{2} \rangle)L \stackrel{\text{def}}{=} G$ with $\mathcal{D}(G) = \langle ML, \frac{1}{2} \rangle + \langle NL, \frac{1}{2} \rangle$, and $F \rightsquigarrow \langle P, \frac{1}{2} \rangle + \langle Q, \frac{1}{2} \rangle \stackrel{\text{def}}{=} H$, with $\mathcal{D}(H) = H$, and it is sufficient now to ensure $(ML)\ \mathcal{R}_2\ P$, and $(NL)\ \mathcal{R}_2\ Q$.

Lemma 7.9 *If $\mathcal{R}$ is a bisimulation up-to formal sums, then $\mathcal{R} \subseteq SS$ for some coupled logical bisimulation SS.*

Proof. We show that the coupled relation SS, with $SS_1 = \mathcal{R}_1$ and

$$SS_2 \stackrel{\text{def}}{=} \mathcal{R}_2 \cup \{(\Sigma_i \langle H_i, p_i \rangle, \Sigma_i \langle K_i, p_i \rangle) \text{ s.t. for each } i, \quad \begin{array}{l} \text{either } H_i\ \mathcal{R}_2\ K_i \\ \text{or } H_i = \langle M_i, 1 \rangle, K_i = \langle N_i, 1 \rangle \text{ and } M_i\ \mathcal{R}_2\ N_i \ \}, \end{array}$$

is a big-step bisimulation and then apply Lemma 7.7. The key point for this is to show that whenever $M\ \mathcal{R}_2\ N$, if $M \twoheadrightarrow Z$ and $N \twoheadrightarrow Y$, then ZSS_2Y.

For this, roughly, we reason on the tree whose nodes are the pairs of terms produced by the up-to bisimulation game for $\mathcal{R}_2$ and with root a pair (M, N) in $\mathcal{R}_2$ (and with the proviso that a node (E, F), if not a pair of values, and not a pair of $\Lambda_{\oplus}$-terms, has one only child, namely (Z, Y) for Z, Y s.t. $E \twoheadrightarrow Z$ and $F \twoheadrightarrow Y$).

Certain paths in the tree may be divergent; those that reach a leaf give the formal sums that M and N produce. Thus, if $M \Longrightarrow Z$ and $N \Longrightarrow Y$, then we can write $Z = \Sigma_i \langle Z_i, p_i \rangle$ and $Y = \Sigma_i \langle Y_i, p_i \rangle$, for Z_i, Y_i, p_i s.t. $\{(Z_i, Y_i, p_i)\}$ represent exactly the multiset of the leaves in the tree together with the probability of the path reaching each leaf. □

Using the above proof technique, we can prove the necessary substitutivity property for bisimulation. The use of up-to techniques, and the way bisimulation is defined (in particular the presence of a clause for τ-steps and the possibility of using the pairs in the bisimulation itself to construct inputs for functions), make it possible to use a standard argument by induction over contexts.

Lemma 7.10 *If $\mathcal{R}$ is a bisimulation then the context closure SS with*

$$SS_1 \stackrel{\text{def}}{=} \mathcal{R}_1^{\mathrm{C}};$$
$$SS_2 \stackrel{\text{def}}{=} \mathcal{R}_2 \cup \mathcal{R}_1^{\mathrm{C}} \cup \{(E\overline{M}, F\overline{N}) \text{ s.t. } E\ \mathcal{R}_2\ F \text{ and } M_i\ \mathcal{R}_1^{\mathrm{C}}\ N_i\};$$

is a bisimulation up-to formal sums.

Corollary 7.11 *1. $M \approx_1 N$ implies $C[M] \approx_1 C[N]$, for all C*

2. $E \approx_2 F$ implies $C[E] \approx_2 C[F]$, for all evaluation contexts C.

Using Lemma 7.10 we can prove the inclusion in context equivalence.

Corollary 7.12 *If $M \approx_1 N$ then $M \simeq_{\oplus}^{\mathrm{FS}} N$. Moreover, if $E \approx_2 F$ then $E \cong_{\oplus}^{\mathrm{FS}} F$.*

The converse of Corollary 7.12 is proved exploiting a few simple properties of $\cong_{\oplus}^{\mathrm{FS}}$ (e.g., its transitivity, the inclusion $\rightsquigarrow \subseteq \cong_{\oplus}^{\mathrm{FS}}$).

Lemma 7.13 *$E \rightsquigarrow E'$ implies $E \cong_{\oplus}^{\mathrm{FS}} E'$.*

Proof. If $E \rightsquigarrow E'$ then $E \approx_2 E'$ hence $E \cong_{\oplus}^{\mathrm{FS}} E'$. □

Lemma 7.14 *$Z \cong_{\oplus}^{\mathrm{FS}} Y$ implies $Z \bullet M \cong_{\oplus}^{\mathrm{FS}} Y \bullet M$ for all M.*

Proof. Follows from definition of $\cong_{\oplus}^{\mathrm{FS}}$, transitivity of $\cong_{\oplus}^{\mathrm{FS}}$, and Lemma 7.13. □

Lemma 7.15 *If $M_i \simeq_{\oplus}^{\mathrm{FS}} N_i$ for each i, then $\Sigma_i \langle M_i, p_i \rangle \cong_{\oplus}^{\mathrm{FS}} \Sigma_i \langle N_i, p_i \rangle$*

Proof. Suppose $\Sigma_i \langle M_i, p_i \rangle \overline{M} \twoheadrightarrow Z$ and $\Sigma_i \langle N_i, p_i \rangle \overline{M} \twoheadrightarrow Y$. We have to show $\mathbf{\Sigma}(Z) = \mathbf{\Sigma}(Y)$. We have $Z = \Sigma_i \langle Z_i, p_i \rangle$ for Z_i with $M_i \twoheadrightarrow Z_i$. Similarly $Y = \Sigma_i \langle Y_i, p_i \rangle$ for Y_i with $N_i \twoheadrightarrow N_i$. Then the result follows from $\mathbf{\Sigma}(Z_i) = \mathbf{\Sigma}(Y_i)$. □

Theorem 7.16 *We have $\simeq_{\oplus}^{\mathrm{FS}} \subseteq \approx_1$, and $\cong_{\oplus}^{\mathrm{FS}} \subseteq \approx_2$.*

Proof. We take the coupled relation $\mathcal{R}$ with

$$\begin{array}{rcl} \mathcal{R}_1 & \stackrel{\mathrm{def}}{=} & \{(M,N) \text{ s.t. } M \simeq_{\oplus}^{\mathrm{FS}} N\} \\ \mathcal{R}_2 & \stackrel{\mathrm{def}}{=} & \{(E,F) \text{ s.t. } E \cong_{\oplus}^{\mathrm{FS}} F\}\} \end{array}$$

and show that $\mathcal{R}$ is a bisimulation.

For clause (1), one uses Lemma 7.13 and transitivity of $\cong_{\oplus}^{\mathrm{FS}}$. For clause (2), consider a term Z with $Z \cong_{\oplus}^{\mathrm{FS}} F$. By definition of $\cong_{\oplus}^{\mathrm{FS}}$, $F \twoheadrightarrow Y$ with $\mathbf{\Sigma}(Z) = \mathbf{\Sigma}(Y)$. Take now arguments $M \simeq_{\oplus}^{\mathrm{FS}} N$ (which is sufficient, since ${\simeq_{\oplus}^{\mathrm{FS}}}^{\mathrm{C}} \subseteq \simeq_{\oplus}^{\mathrm{FS}}$). By Lemma 7.14, $Z \bullet M \cong_{\oplus}^{\mathrm{FS}} Y \bullet N$. By Lemma 7.15, $W \bullet M \cong_{\oplus}^{\mathrm{FS}} Y \bullet N$. Hence also $Z \bullet M \cong_{\oplus}^{\mathrm{FS}} Y \bullet N$, and we have $Z \bullet M \; \mathcal{R}_2 \; Y \bullet N$. □

It also holds that coupled logical bisimilarity is preserved by the formal sum construct; i.e., $M_i \approx_1 N_i$ for each $i \in I$ implies $\Sigma_{i \in I} \langle M_i, p_i \rangle \approx_2 \Sigma_{i \in I} \langle N_i, p_i \rangle$. As a consequence, context equivalence defined on general $\Lambda_{\oplus}^{\mathrm{FS}}$ contexts is the same as that set on simple contexts (Definition 7.2).

Remark 7.17 *The functional induced by coupled logical bisimulation is* not *monotone. For instance, if $\mathcal{V} \subseteq \mathcal{W}$, then a pair of terms may satisfy the bisimulation clauses on $(\mathcal{V}, \mathcal{E})$, for some $\mathcal{E}$, but not on $(\mathcal{W}, \mathcal{E})$, because the input for functions may be taken from the larger relation $\mathcal{W}$. (Recall that coupled relations are* pairs *of relations. Hence operations on coupled relations, such as union and inclusion, are defined component-wise.) However, Corollary 7.12 and Theorem 7.16 tell us that there is indeed a largest bisimulation, namely the pair $(\simeq_{\oplus}^{\mathrm{FS}}, \cong_{\oplus}^{\mathrm{FS}})$.*

With logical (as well as environmental) bisimulations, up-to techniques are particularly important to relieve the burden of proving concrete equalities. A powerful up-to technique in higher-order languages is *up-to contexts*. We present a form of up-to contexts combined with the big-step version of logical bisimilarity. Below, for a relation $\mathcal{R}$ on $\Lambda_{\oplus}$, we write $\mathcal{R}^{\mathrm{CFS}}$ for the closure of the relation under general (closing) $\Lambda_{\oplus}^{\mathrm{FS}}$ contexts.

Definition 7.18 *A coupled relation $\mathcal{R}$ is a* big-step coupled logical bisimulation up-to contexts *if whenever $E \; \mathcal{R}_2 \; F$, the following holds: if $E \twoheadrightarrow Z$ then $F \twoheadrightarrow Y$ with $\mathbf{\Sigma}(Z) = \mathbf{\Sigma}(Y)$, and for all $M \; \mathcal{R}_1^{\mathrm{C}} \; N$, we have $(Z \bullet M) \; \mathcal{R}_1^{\mathrm{CFS}} \; (Y \bullet N)$.*

For the soundness proof, we first derive the soundness of a small-step up-to context technique, whose proof, in turn, is similar to that of Lemma 7.10 (the up-to-formal-sums technique of Definition 7.8 already allows some context manipulation; we need this technique for the proof of the up-to-contexts technique).

Example 7.19 *We have seen that the terms* `expone` *and* `exptwo` *of Example 2.6 are not applicative bisimilar. We can show that they are context equivalent, by proving that they are coupled bisimilar. We sketch a proof of this, in which we employ the up-to technique from Definition 7.18. We use the coupled relation* $\mathcal{R}$ *in which* $\mathcal{R}_1 \stackrel{\text{def}}{=} \{(\texttt{expone}, \texttt{exptwo})\}$*, and* $\mathcal{R}_2 \stackrel{\text{def}}{=} \mathcal{R}_1 \cup \{(A_M, B_N) \mid M \ \mathcal{R}_1^{\text{C}} \ N\}$ *where* $A_M \stackrel{\text{def}}{=} \lambda n.((Mn) \oplus (\texttt{expone}\, M\, (n+1)))$*, and* $B_N \stackrel{\text{def}}{=} (\lambda x.Nx) \oplus (\texttt{exptwo}\, (\lambda x.N(x+1)))$*. This is a big-step coupled logical bisimulation up-to contexts. The interesting part is the matching argument for the terms* A_M, B_N*; upon receiving an argument* m *they yield the summed values* $\Sigma_i \langle M(m+1), p_i \rangle$ *and* $\Sigma_i \langle N(m+1), p_i \rangle$ *(for some* p_i*'s), and these are in* $\mathcal{R}_1^{\text{CFS}}$*.*

8 Beyond Call-by-Name Reduction

So far, we have studied the problem of giving sound (and sometime complete) coinductive methods for program equivalence in a probabilistic λ-calculus endowed with *call-by-name* reduction. One may wonder whether what we have obtained can be adapted to other notions of reduction, and in particular to *call-by-value* reduction (e.g., the call-by-value operational semantics of $\Lambda_\oplus$ from [10]).

Since our construction of a labelled Markov chain for $\Lambda_\oplus$ is somehow independent on the underlying operational semantics, defining a call-by-value probabilistic applicative bisimulation is effortless. The proofs of congruence of the bisimilarity and its soundness in this paper can also be transplanted to call-by-value. In defining $\Lambda_\oplus$ as a multisorted labelled Markov chain for the strict regime, one should recall that functions are applied to values only.

Definition 8.1 $\Lambda_\oplus$ *can be seen as a multisorted labelled Markov chain* $(\Lambda_\oplus(\emptyset) \uplus \mathsf{V}\Lambda_\oplus, \mathsf{V}\Lambda_\oplus \uplus \{\tau\}, \mathcal{P}_\oplus)$ *that we denote with* $\Lambda_{\oplus_\mathsf{v}}$*. Please observe that, contrary to how we gave Definition 3.1 for call-by-name semantics, labels here are either values, which model parameter passing, or* τ*, that models evaluation. We define the transition probability matrix* $\mathcal{P}_\oplus$ *as follows:*

- *For every term* M *and for every distinguished value* $\nu x.N$*,*

$$\mathcal{P}_\oplus(M, \tau, \nu x.N) \stackrel{\text{def}}{=} [\![M]\!](\nu x.N);$$

- *For every value* V *and for every distinguished value* $\nu x.N$*,*

$$\mathcal{P}_\oplus(\nu x.N, V, N\{^V/_x\}) \stackrel{\text{def}}{=} 1;$$

- *In all other cases,* $\mathcal{P}_\oplus$ *returns* 0*.*

Then, similarly to the call-by-name case, one can define both probabilistic applicative simulation and bisimulation notions as probabilistic simulation and bisimulation on $\Lambda_{\oplus_\mathsf{v}}$. This way one can define *probabilistic applicative bisimilarity*, which is denoted $\sim_\mathsf{v}$, and *probabilistic applicative similarity*, denoted $\lesssim_\mathsf{v}$.

Proving that $\lesssim_\mathsf{v}$ is a precongruence, follows the reasoning we have outlined for the lazy regime. Of course, one must prove a Key Lemma first.

Lemma 8.2 *If* $M \lesssim_\mathsf{v}^H N$*, then for every* $X \subseteq \Lambda_\oplus(x)$ *it holds that* $[\![M]\!](\lambda x.X) \leq [\![N]\!](\lambda x.(\lesssim_\mathsf{v}^H(X)))$*.*

As the statement, the proof is not particularly different from the one we have provided for Lemma 3.17. The only delicate case is obviously that of application. This is due to its operational semantics that, now, takes into account also the distribution of values the parameter reduces to. Anyway, one can prove $\sim_\mathsf{v}$ of implying context equivalence.

When we restrict our attention to pure λ-terms, as we do in Section 6, we are strongly relying on call-by-name evaluation: LLT's only reflect term equivalence in a call-by-name lazy regime. We

leave the task of generalizing the results to eager evaluation to future work, but we conjecture that, in that setting, probabilistic choice *alone* does not give contexts the same discriminating power as probabilistic bisimulation. Similarly we have not investigated the call-by-value version of coupled logical bisimilarity, as our current proofs rely on the appearance of formal sums only in redex position, a constraint that would probably have to be lifted for call-by-value.

9 A Comparison with Nondeterminism

Syntactically, $\Lambda_\oplus$ is identical to an eponymous language introduced by de'Liguoro and Piperno [13]. The semantics we present here, however, is quantitative, and this has of course a great impact on context equivalence. While in a nondeterministic setting what one observes is the *possibility* of converging (or of diverging, or both), terms with different convergence probabilities are considered different in an essential way here. Actually, nondeterministic context equivalence and probabilistic context equivalence are incomparable. As an example of terms that are context equivalent in the *must* sense but not probabilistically, we can take $I \oplus (I \oplus \Omega)$ and $I \oplus \Omega$. Conversely, I is probabilistically equivalent to any term M that reduces to $I \oplus M$ (which can be defined using fixed-point combinators), while I and M are not equivalent in the *must* sense, since the latter can diverge (the divergence is irrelevant probabilistically because it has probability zero). *May* context equivalence, in contrast, is coarser than probabilistic context equivalence.

Despite the differences, the two semantics have similarities. Analogously to what happens in nondeterministic λ-calculi, applicative bisimulation and context equivalence do *not* coincide in the probabilistic setting, at least if call-by-name is considered. The counterexamples to full abstraction are much more complicated in call-by-value λ-calculi [27], and cannot be easily adapted to the probabilistic setting.

10 Conclusions

This is the first paper in which bisimulation techniques for program equivalence are shown to be applicable to probabilistic λ-calculi.

On the one hand, Abramsky's idea of seeing interaction as application is shown to be amenable to a probabilistic treatment, giving rise to a congruence relation that is sound for context equivalence. Completeness, however, fails: the way probabilistic applicative bisimulation is defined allows one to distinguish terms that are context equivalent, but which behave differently as for *when* choices and interactions are performed. On the other, a notion of coupled logical bisimulation is introduced and proved to precisely characterise context equivalence for $\Lambda_\oplus$. Along the way, applicative bisimilarity is proved to coincide with context equivalence on pure λ-terms, yielding the Levy-Longo tree equality.

The crucial difference between the two main bisimulations studied in the paper is not the style (applicative *vis-à-vis* logical), but rather the fact that while applicative bisimulation insists on relating only individual terms, coupled logical bisimulation is more flexible and allows us to relate formal sums (which we may think as distributions). This also explains why we need distinct reduction rules for the two bisimulations. See examples 3.38 and 7.5. While not complete, applicative bisimulation, as it stands, is simpler to use than coupled logical bisimulation. Moreover it is a natural form of bisimulation, and it should be interesting trying to transport the techniques for handling it onto variants or extensions of the language.

Topics for future work abound — some have already been hinted at in earlier sections. Among the most interesting ones, one can mention the transport of applicative bisimulation onto the language $\Lambda_\oplus^{\mathrm{FS}}$. We conjecture that the resulting relation would coincide with coupled logical bisimilarity and context equivalence, but going through Howe's technique seems more difficult than for $\Lambda_\oplus$, given the infinitary nature of formal sums and their confinement to redex positions.

Also interesting would be a more *effective* notion of equivalence: even if the two introduced notions of bisimulation avoid universal quantifications over all possible contexts, they refer to an

essentially infinitary operational semantics in which the meaning of a term is obtained as the least upper bound of all its finite approximations. Would it be possible to define bisimulation in terms of approximations without getting too fine grained?

Bisimulations in the style of logical bisimulation (or environmental bisimulation) are known to require up-to techniques in order to avoid tedious equality proofs on concrete terms. In the paper we have introduced some up-to techniques for coupled logical bisimilarity, but additional techniques would be useful. Up-to techniques could also be developed for applicative bisimilarity.

More in the long-run, we would like to develop sound operational techniques for so-called *computational indistinguishability*, a key notion in modern cryptography. Computational indistinguishability is defined similarly to context equivalence; the context is however required to work within appropriate resource bounds, while the two terms can have different observable behaviors (although with negligible probability). We see this work as a very first step in this direction: complexity bounds are not yet there, but probabilistic behaviour, an essential ingredient, is correctly taken into account.

References

[1] S. Abramsky. The Lazy λ-Calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–117. Addison Wesley, 1990.

[2] Samson Abramsky and C.-H. Luke Ong. Full abstraction in the lazy lambda calculus. *Inf. Comput.*, 105(2):159–267, 1993.

[3] Egidio Astesiano and Gerardo Costa. Distributive semantics for nondeterministic typed lambda-calculi. *Theor. Comput. Sci.*, 32:121–156, 1984.

[4] Hendrik Pieter Barendregt. *The Lambda Calculus – Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.

[5] Marco Bernardo, Rocco De Nicola, and Michele Loreti. A uniform framework for modeling nondeterministic, probabilistic, stochastic, or mixed processes and their behavioral equivalences. *Inf. Comput.*, 225:29–82, 2013.

[6] G. Boudol and C. Laneve. The discriminating power of the λ-calculus with multiplicities. *Inf. Comput.*, 126(1):83–102, 1996.

[7] Gérard Boudol. Lambda-calculi for (strict) parallel functions. *Inf. Comput.*, 108(1):51–127, 1994.

[8] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Kernel-based object tracking. *IEEE Trans. on Pattern Analysis and Machine Intelligence,*, 25(5):564–577, 2003.

[9] Ugo Dal Lago, Davide Sangiorgi, and Michele Alberti. On coinductive equivalences for probabilistic higher-order functional programs (long version). Available at `http://arxiv...`, 2013.

[10] Ugo Dal Lago and Margherita Zorzi. Probabilistic operational semantics for the lambda calculus. *RAIRO - Theor. Inf. and Applic.*, 46(3):413–450, 2012.

[11] Vincent Danos and Russell Harmer. Probabilistic game semantics. *ACM Trans. Comput. Log.*, 3(3):359–382, 2002.

[12] Rocco De Nicola and Matthew Hennessy. Testing equivalences for processes. *Theor. Comput. Sci.*, 34:83–133, 1984.

[13] Ugo de'Liguoro and Adolfo Piperno. Non deterministic extensions of untyped lambda-calculus. *Inf. Comput.*, 122(2):149–177, 1995.

[14] M. Dezani-Ciancaglini and E. Giovannetti. From bohm's theorem to observational equivalences: an informal account. *Electr. Notes Theor. Comput. Sci.*, 50(2):83–116, 2001.

[15] M. Dezani-Ciancaglini, J. Tiuryn, and P. Urzyczyn. Discrimination by parallel observers: The algorithm. *Inf. Comput.*, 150(2):153–186, 1999.

[16] Thomas Ehrhard, Michele Pagani, and Christine Tasson. The computational meaning of probabilistic coherence spaces. In *LICS*, pages 87–96, 2011.

[17] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[18] Noah D. Goodman. The principles and practice of probabilistic programming. In *POPL*, pages 399–402, 2013.

[19] Andrew D. Gordon. Bisimilarity as a theory of functional programming. *Electr. Notes Theor. Comput. Sci.*, 1:232–252, 1995.

[20] Andrew D. Gordon, Mihhail Aizatulin, Johannes Borgström, Guillaume Claret, Thore Graepel, Aditya V. Nori, Sriram K. Rajamani, and Claudio V. Russo. A model-learner pattern for bayesian reasoning. In *POPL*, pages 403–416, 2013.

[21] Matthew Hennessy. Exploring probabilistic bisimulations, part I. *Formal Asp. Comput.*, 24(4-6):749–768, 2012.

[22] Douglas J. Howe. Proving congruence of bisimulation in functional programming languages. *Inf. Comput.*, 124(2):103–112, 1996.

[23] Radha Jagadeesan and Prakash Panangaden. A domain-theoretic model for a higher-order process calculus. In *ICALP*, pages 181–194, 1990.

[24] C. Jones and Gordon D. Plotkin. A probabilistic powerdomain of evaluations. In *LICS*, pages 186–195, 1989.

[25] V. Koutavas, P.B̃. Levy, and E. Sumii. From applicative to environmental bisimulation. *Electr. Notes Theor. Comput. Sci.*, 276:215–235, 2011.

[26] Kim Guldstrand Larsen and Arne Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991.

[27] S. B. Lassen. *Relational Reasoning about Functions and Nondeterminism.* PhD thesis, University of Aarhus, 1998.

[28] Sergueï Lenglet, Alan Schmitt, and Jean-Bernard Stefani. Howe's method for calculi with passivation. In *CONCUR*, pages 448–462, 2009.

[29] Jean-Jacques Lévy. An algebraic interpretation of equality in some models of the lambda calculus. In C. Böhm, editor, *Lambda Calculus and Computer Science Theory*, volume 37 of *LNCS*, pages 147–165. Springer-Verlag, 1975.

[30] Giuseppe Longo. Set-theoretical models of lambda calculus: Theories, expansions and isomorphisms. *Ann. Pure Appl. Logic*, 24:153–188, 1983.

[31] Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*, volume 999. MIT Press, 1999.

[32] J. Morris. *Lambda Calculus Models of Programming Languages.* PhD thesis, MIT, 1969.

[33] C.-H. Luke Ong. Non-determinism in a functional setting. In *LICS*, pages 275–286, 1993.

[34] Prakash Panangaden. *Labelled Markov Processes.* Imperial College Press, 2009.

[35] Sungwoo Park, Frank Pfenning, and Sebastian Thrun. A probabilistic language based on sampling functions. *ACM Trans. Program. Lang. Syst.*, 31(1), 2008.

[36] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[37] Avi Pfeffer. IBAL: A probabilistic rational programming language. In *IJCAI*, pages 733–740. Morgan Kaufmann, 2001.

[38] A. M. Pitts. Howe's method for higher-order languages. In D. Sangiorgi and J. Rutten, editors, *Advanced Topics in Bisimulation and Coinduction*, pages 197–232. Cambridge University Press, 2011.

[39] Andrew M. Pitts. Operationally-based theories of program equivalence. In *Semantics and Logics of Computation*, pages 241–298. Cambridge University Press, 1997.

[40] Norman Ramsey and Avi Pfeffer. Stochastic lambda calculus and monads of probability distributions. In *POPL*, pages 154–165, 2002.

[41] N. Saheb-Djahromi. Probabilistic LCF. In *MFCS*, volume 64 of *LNCS*, pages 442–451, 1978.

[42] David Sands. From SOS rules to proof principles: An operational metatheory for functional languages. In *POPL*, pages 428–441, 1997.

[43] D. Sangiorgi. The lazy lambda calculus in a concurrency scenario. *Inf. and Comp.*, 111(1):120–153, 1994.

[44] Davide Sangiorgi, Naoki Kobayashi, and Eijiro Sumii. Logical bisimulations and functional languages. In *FSEN*, volume 4767 of *LNCS*, pages 364–379, 2007.

[45] Davide Sangiorgi, Naoki Kobayashi, and Eijiro Sumii. Environmental bisimulations for higher-order languages. *ACM Trans. Program. Lang. Syst.*, 33(1):5, 2011.

[46] Davide Sangiorgi and David Walker. *The pi-Calculus – a theory of mobile processes*. Cambridge University Press, 2001.

[47] Kurt Sieber. Call-by-value and nondeterminism. In *TLCA*, volume 664 of *LNCS*, pages 376–390, 1993.

[48] Sebastian Thrun. Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium*, pages 1–35, 2002.

Wave-Style Token Machines and Quantum Lambda Calculi

Ugo Dal Lago* Margherita Zorzi†

Abstract

Particle-style token machines are a way to interpret proofs and programs, when the latter are written following the principles of linear logic. In this paper, we show that token machines also make sense when the programs at hand are those of a simple quantum λ-calculus. This, however, requires generalizing the concept of a token machine to one in which more than one particle travel around the term *at the same time*. The presence of multiple tokens is intimately related to entanglement and allows to give a simple operational semantics to the calculus, coherently with the principles of quantum computation.

1 Introduction

One of the strongest trends in computer science is the (relatively recent) interest in exploiting new computing paradigms which go beyond the usual, classical one. Among these paradigms, quantum computing plays an important role. In particular, the quantum paradigm is having a deep impact on the notion of a computationally (in)tractable problem. In this respect, two of the most surprising results are due to Peter Shor, who proved that prime factorization of integers and the discrete logarithm can be efficiently solved (i.e. in polynomial time) by a quantum computer [20].

Even if quantum computing has catalyzed the interest of a quite large scientific community, several theoretical aspects are still unexplored. As an example, the definition of a robust theoretical framework for quantum programming is nowadays still a challenge. A number of (paradigmatic) calculi for quantum computing have been introduced in the last ten years. Among them, some functional calculi, typed and untyped, have been proposed [3, 5, 4, 18, 21], but we are still at a stage where it is not clear whether one calculus could be considered *canonical*. Moreover, the meta-theory of most of these formalisms lack the simplicity of the one of their "classical" siblings.

It is clear that linear logic and quantum computing are strongly related: since quantum data have to undergo restrictions such as no-cloning and no-erasing, it is not surprising that in most of the cited quantum calculi the use of resources is controlled. Linear logic therefore provides an ideal framework where rooting quantum data treatment, but also offers another tool which has not been widely exploited in the quantum setting: its mathematical model in terms of operator algebras, i.e. the Geometry of Interaction (GoI in the following). Indeed, the latter provides a dynamical interpretation and a semantic account of the cut-elimination procedure as a flow of information circulating into a net structure. This idea can be formulated both as an algebra of bounded operators on a infinitely dimensional Hilbert space [10] or as a token-based machine (a rewriting automata model with local transition rules) [11, 14]. Both formulations seem to be promising in the quantum setting. On the one hand, the Hilbert space on top of which the first formulation of GoI is given is precisely the canonical

*Università di Bologna & INRIA
†Università di Verona

state space of a quantum Turing machine (see for example [1]). On the other hand, the definition of a token machine provides a mathematically simpler setting, which has already found a role in this context [2, 12].

In this paper, we show that token machines are also a model of a linear quantum λ-calculus $\mathsf{Q}\Lambda$ defined along the lines of van Tonder's λ_q [21]. This allows to give an operational semantics to $\mathsf{Q}\Lambda$ which renders the quantum nature of $\mathsf{Q}\Lambda$ explicit: type derivations become quantum circuits built on exactly the set of gates occurring in the underlying λ-term. This frees us from the burden of having to define the operational semantics of quantum calculi in reduction style, which is known to be technically challenging in a similar setting [21]. On the other hand, the power of β-style axioms is retained in the form of an equational theory for which our operational semantics can be proved sound.

Technically, the design of our token machine for $\mathsf{Q}\Lambda$, called $\mathsf{IAM}_{\mathsf{Q}\Lambda}$ is arguably more challenging than the one of classical token machines. Indeed, the principles of quantum computing, and the so-called *entanglement* in particular, force us to go towards *wave-style* machines, i.e., to machines where more than one particle can travel inside the program at the same time. Moreover, the possibly many tokens at hand are subject to synchronization points, each one corresponding to unitary operators of arity greater than 1. This means that $\mathsf{IAM}_{\mathsf{Q}\Lambda}$, in principle, could suffer from deadlocks, let alone the possibility of non-termination. We here prove that these pathological situations can not happen.

In Section 2, we recall the token machine for multiplicative linear logic. In Section 3 we propose a gentle introduction to quantum computing. The calculus $\mathsf{Q}\Lambda$ and its token machine $\mathsf{IAM}_{\mathsf{Q}\Lambda}$ are introduced in Section 4 and Section 5, respectively. Main results about $\mathsf{IAM}_{\mathsf{Q}\Lambda}$ are in Section 6. Sections 7 and 8 are respectively devoted to related works and conclusion/future plans.

2 Linear Logic and Token Machines

In this section, we give some ideas about the simplest token machine, namely the one for the propositional, multiplicative fragment of linear logic. This not only encourages the unfamiliar reader to understand the basic concepts underlying this concrete approach to the geometry of interaction, but will also be useful in the following, when proving basic results about quantum token machines. More details can be found in [7, 11].

Let $\mathbb{A} = \{\alpha, \beta, \ldots\}$ be a countable set of *propositional atoms*. *Formulas* of Multiplicative Linear Logic (MLL) are given by the following grammar:

$$A, B ::= \alpha \mid \alpha^{\perp} \mid A \otimes B \mid A \parr B.$$

Linear negation can be extended to all formulas in the usual way:

$$\begin{aligned} (\alpha^{\perp})^{\perp} &= \alpha; \\ A \otimes B^{\perp} &= A^{\perp} \parr B^{\perp}; \\ A \parr B^{\perp} &= A^{\perp} \otimes B^{\perp}. \end{aligned}$$

This way, $A^{\perp\perp}$ is just A. The one-sided sequent calculus for MLL is very simple:

$$\frac{}{\vdash A, A^{\perp}}\,\mathsf{ax} \qquad \frac{\vdash \Gamma, A \quad \vdash \Delta, A^{\perp}}{\vdash \Gamma, \Delta}\,\mathsf{cut} \qquad \frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B}\,\otimes \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\,\parr$$

The logic MLL enjoys cut-elimination: there is a terminating algorithm turning any MLL proofs into a cut-free proof of the same conclusion.

Consider the following MLL proof ξ (where different occurrences of the same propositional (co)atom have been numbered):

$$\dfrac{\dfrac{\dfrac{}{\vdash \alpha_4^\perp, \alpha_4}\,\mathsf{ax} \qquad \dfrac{}{\vdash \alpha_5^\perp, \alpha_5}\,\mathsf{ax}}{\vdash \alpha_3^\perp, \alpha_3}\,\mathsf{cut} \qquad \dfrac{}{\vdash \beta_3^\perp, \beta_3}\,\mathsf{ax}}{\dfrac{\vdash \alpha_2^\perp, \beta_2, \alpha_2 \otimes \beta_2^\perp}{\vdash \alpha_1^\perp \parr \beta_1, \alpha_1 \otimes \beta_1^\perp}\,\parr}\,\otimes$$

The token machine for ξ is a simple automaton whose internal state is nothing more than an occurrence of a propositional (co)atom in ξ. This state evolves by "following" this occurrence, keeping in mind that atoms go down, while coatoms go up. A run of the token machine of ξ is, as an example, the following one:

$$\alpha_1^\perp \mapsto_\xi \alpha_2^\perp \mapsto_\xi \alpha_3^\perp \mapsto_\xi \alpha_4^\perp \mapsto_\xi \alpha_4 \mapsto_\xi \alpha_5^\perp \mapsto_\xi \alpha_5 \mapsto_\xi \alpha_3 \mapsto_\xi \alpha_2 \mapsto_\xi \alpha_1.$$

This tells us that the occurrences $\alpha_1^\perp$ and α_1 are somehow related. Similarly, one could find a run relating β_1 to $\beta_1^\perp$. Remarkably, these correspondences survive cut-elimination.

All this can be formalized through the notion of a *context*, which is an MLL formula with a hole:

$$\mathsf{C} ::= [\cdot] \;\Big|\; \mathsf{C} \otimes A \;\Big|\; A \otimes \mathsf{C} \;\Big|\; \mathsf{C} \parr A \;\Big|\; A \parr \mathsf{C}.$$

$\mathsf{C}[A]$ is the formula obtained by replacing the unique occurrence of $[\cdot]$ in C with A. If $A = \mathsf{C}[\alpha]$ ($A = \mathsf{C}[\alpha^\perp]$, respectively), we say that C is a *positive* (*negative*, respectively) *context for* A. If C is positive (negative, respectively) for A, we sometime write it as P_A (as N_A, respectively). An *atom occurrence* in an MLL proof ξ is a pair (A, C) where A is an occurrence of an MLL formula in ξ and C is a context for it. Linear negation can be easily extended to contexts:

$$\begin{aligned}
[\cdot]^\perp &= [\cdot]; \\
(\mathsf{C} \otimes B)^\perp &= \mathsf{C}^\perp \parr B^\perp; & (A \otimes \mathsf{C})^\perp &= A^\perp \parr \mathsf{C}^\perp; \\
(\mathsf{C} \parr B)^\perp &= \mathsf{C}^\perp \otimes B^\perp; & (A \parr \mathsf{C})^\perp &= A^\perp \otimes \mathsf{C}^\perp.
\end{aligned}$$

Please observe that C is a negative context for A iff $\mathsf{C}^\perp$ is a positive context for $A^\perp$. To every proof ξ in MLL, we associate an automaton $\mathcal{M}_\xi$ which consists of:

- The finite set $\mathcal{S}_\xi$ of *states* of $\mathcal{M}_\xi$, which are all the atom occurrences of ξ;
- a *transition relation* $\mapsto_\xi \subseteq \mathcal{S}_\xi \times \mathcal{S}_\xi$, which is described by the rules in Figure 1.

An atom occurrence in ξ is said to be *initial* (respectively, *final*) iff it is in the form (A, N_A) (respectively, in the form (A, P_A)), where A is one among the formulas among the conclusions of ξ. It is easy to verify that:

- for every non-final occurrence O there is exactly one occurrence P such that $O \mapsto_\xi P$;
- for every non-initial occurrence O there is exactly one occurrence P such that $P \mapsto_\xi O$.

As a consequence, every initial occurrence is put in correspondence with a final occurrence in a bijective way — the number of occurrences in ξ is anyway finite, and cycles cannot be reached from initial occurrences. It is this correspondence which is taken as the semantics of ξ, after being shown to be invariant by cut-elimination.

One last observation is now in order. Suppose $O_1, \ldots, O_n$ are *all* the initial occurrences for ξ. Then, every occurrence in ξ is visited *exactly once* along one of the n maximal computations starting in $O_1, \ldots, O_n$. This can be proved as follows:

$$\frac{}{\vdash A, A^{\perp}}\ \mathsf{ax} \qquad \begin{array}{c}(A, \mathsf{N}_A) \mapsto_\xi (A^{\perp}, \mathsf{N}_A^{\perp}) \\ (A^{\perp}, \mathsf{N}_{A^{\perp}}) \mapsto_\xi (A, (\mathsf{N}_{A^{\perp}})^{\perp})\end{array}$$

$$\frac{\vdash \Gamma_1, A \quad \vdash \Delta_1, B}{\vdash \Gamma_2, \Delta_2, A \otimes B}\ \otimes \qquad \begin{array}{c}(A \otimes B, \mathsf{N}_A \otimes B) \mapsto_\xi (A, \mathsf{N}_A) \\ (A \otimes B, A \otimes \mathsf{N}_B) \mapsto_\xi (B, \mathsf{N}_B) \\ (A, \mathsf{P}_A) \mapsto_\xi (A \otimes B, \mathsf{P}_A \otimes B) \\ (B, \mathsf{P}_B) \mapsto_\xi (A \otimes B, A \otimes \mathsf{P}_B) \\ (\Gamma_2, \mathsf{N}) \mapsto_\xi (\Gamma_1, \mathsf{N}) \\ (\Delta_2, \mathsf{N}) \mapsto_\xi (\Delta_1, \mathsf{N}) \\ (\Gamma_1, \mathsf{P}) \mapsto_\xi (\Gamma_2, \mathsf{P}) \\ (\Delta_1, \mathsf{P}) \mapsto_\xi (\Delta_2, \mathsf{P})\end{array}$$

$$\frac{\vdash \Gamma_1, A, B}{\vdash \Gamma_2, A \parr B}\ \parr \qquad \begin{array}{c}(A \parr B, \mathsf{N}_A \parr B) \mapsto_\xi (A, \mathsf{N}_A) \\ (A \parr B, A \parr \mathsf{N}_B) \mapsto_\xi (B, \mathsf{N}_B) \\ (A, \mathsf{P}_A) \mapsto_\xi (A \parr B, \mathsf{P}_A \parr B) \\ (B, \mathsf{P}_B) \mapsto_\xi (A \parr B, A \parr \mathsf{P}_B) \\ (\Gamma_2, \mathsf{N}) \mapsto_\xi (\Gamma_1, \mathsf{N}) \\ (\Gamma_1, \mathsf{P}) \mapsto_\xi (\Gamma_2, \mathsf{P})\end{array}$$

$$\frac{\vdash \Gamma_1, A \quad \vdash \Delta_1, A^{\perp}}{\vdash \Gamma_2, \Delta_2}\ \mathsf{cut} \qquad \begin{array}{c}(A, \mathsf{P}_A) \mapsto_\xi (A^{\perp}, (\mathsf{P}_A)^{\perp}) \\ (A^{\perp}, \mathsf{P}_{A^{\perp}}) \mapsto_\xi (A, (\mathsf{P}_{A^{\perp}})^{\perp}) \\ (\Gamma_2, \mathsf{N}) \mapsto_\xi (\Gamma_1, \mathsf{N}) \\ (\Delta_2, \mathsf{N}) \mapsto_\xi (\Delta_1, \mathsf{N}) \\ (\Gamma_1, \mathsf{P}) \mapsto_\xi (\Gamma_2, \mathsf{P}) \\ (\Delta_1, \mathsf{P}) \mapsto_\xi (\Delta_2, \mathsf{P})\end{array}$$

Figure 1: Defining Rules for $\mapsto_\xi$

- First, prove the statement for any cut-free proof ξ, by induction on the structure of ξ;
- Then show that if ξ has the property and μ reduces to ξ by cut-elimination, μ has the property, too.

Incidentally, this shows that cylic $\mapsto_\xi$ is acyclic.

3 Quantum Computing in a Nutshell

Quantum computing principles are non-standard notions to the largest part of the "lambda community". The aim of this section is to provide to the non-expert reader an overview of quantum computing basic concepts. This will guide her or him in understanding the "quantum content" of our calculus (in particular, the meaning of unitary steps and the linear management of quantum data, see Section 4). Moreover, notions like quantum entanglement, a peculiar feature of quantum data, offers some intuitions about how and why the choice of a wave-style token machine as operational model is the right choice.

The simplest quantum system is a two-dimensional state space whose elements are called *quantum bits* or *qubits* for short. The qubit is the most basic unit of quantum information. The most direct way to represent a quantum bit is as a unitary vector in the 2-dimensional Hilbert space $\ell^2(\{0,1\})$, which is isomorphic to $\mathbb{C}^2$. We will denote with $|0\rangle$ and $|1\rangle$ the elements of the computational basis of $\ell^2(\{0,1\})$. The states $|0\rangle$ and $|1\rangle$ of a qubit correspond to the boolean constants 0 and 1, which are the only possible values of a classical bit. A qubit, however, can assume other values, different from $|0\rangle$ and $|1\rangle$. In fact, every linear combination $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ where $\alpha, \beta \in \mathbb{C}$, and $|\alpha|^2 + |\beta|^2 = 1$, represents a possible qubit state. These states are said to be *superposed*, and the two values α and β are called *amplitudes*. The amplitudes α and β univocally represent the qubit with respect to the computational basis. Given a qubit $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, we commonly denote it by the vectorial notation

$$\psi = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}.$$

In particular, the vectorial representation of the elements of the computational basis $|0\rangle$ and $|1\rangle$ is the following:

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

While we can determine the state of a classical bit, for a qubit we can not establish with the same precision the values α and β: quantum mechanics says that a measurement of a qubit with state $\alpha|0\rangle + \beta|1\rangle$ has the effect of changing the state to $|0\rangle$ with probability $|\alpha|^2$ and to $|1\rangle$ with probability $|\beta|^2$. For example, if $|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$, one can observe 0 or 1 with the same probability $|\frac{1}{\sqrt{2}}|^2 = \frac{1}{2}$. In this brief survey on quantum computing, we will not enter in the details about qubit measurement, since the syntax of the calculus $\mathsf{Q}\Lambda$ does not include an explicit measurement operator (a constant whose — probabilistic — operational semantics mimics the observation of quantum data). This choice is sound from a theoretical viewpoint, since it is possible to assume to have a unique, final measurement, at the end of the computation. Notwithstanding, the measurement operator is a useful programming tool in order to encode quantum algorithms and the extension of the syntax with a measurement operator is one of our planned future works. For a complete overview about measurement of qubits and relationships between different kind of measurement, see [16].

In order to define arbitrary set of quantum data, we need a generalization of the notion of qubit, called *quantum register* or, more commonly, *quantum state* [21, 18, 17]. A quantum register can be viewed as a system of n qubits and, mathematically, it is a normalized vector in the Hilbert space

$\ell^2(\{0,1\}^n)$ ($\{0,1\}^n$ is a compact notation to represent any binary sequence of length n). The standard computational basis for $\ell^2(\{0,1\}^n)$ is $\mathcal{B} = \{|i\rangle \mid i \text{ is a binary string of length } n\}$.

Notation 1 *We use the notation $|b_1 \dots b_k\rangle$ ($b_i \in \{0,1\}$) for $|b_1\rangle \otimes \dots \otimes |b_k\rangle$, where $\otimes$ is the* tensor product *(see below).*

With a little abuse of language, we say that the number of quits n corresponds to the dimension of the space. Notice that if the dimension is n, then the basis $\mathcal{B}$ contains 2^n elements, and each quantum states is a normalized linear combination of these elements:

$$\alpha_1 \underbrace{|00\dots0\rangle}_{n} + \alpha_2|00\dots1\rangle + \dots + \alpha_{2^n}|11\dots1\rangle$$

Example 1 *Let us consider a 2-level quantum system, i.e. a system of two qubits. Each 2-qubit quantum register is a normalized vector in $\ell^2(\{0,1\}^2)$ and the computational basis is $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$. For example, $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{4}}|01\rangle + \frac{1}{\sqrt{8}}|10\rangle + \frac{1}{\sqrt{8}}|11\rangle$ is a quantum register of two qubits and we can represent it as*

$$\psi = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{4}} \\ \frac{1}{\sqrt{8}} \\ \frac{1}{\sqrt{8}} \end{pmatrix}.$$

An Hilbert space of dimension n can be built from smaller Hilbert spaces by means of the *tensor product* $\otimes$. If H_1 is an Hilbert space of dimension k and H_2 is an Hilbert space of dimension m, $H_3 = H_1 \otimes H_2$ is an Hilbert space of dimension km (each element is a vector of km coordinates obtained by "hooking" a vector in H_2 to a vector in H_1). In other words, an n-qubit quantum register with $n \geq 2$ can be viewed as a composite system. It is possible to combine two (or more) distinct physical systems into a composite one. If the first system is in the state $|\phi_1\rangle$ (a vector in a Hilbert Space H_1) and the second system is in the state $|\phi_2\rangle$ (a vector in a Hilbert Space H_1) , then the state of the combined system is $|\phi_1\rangle \otimes |\phi_2\rangle$ (a vector in a Hilbert Space $H_1 \otimes H_2$) .
We will often omit the "$\otimes$" symbol, and will write the joint state as $|\psi_1\rangle|\psi_2\rangle$ or as $|\psi_1\psi_2\rangle$.

Not all quantum states can be viewed as composite systems: this case occurs in presence of *entanglement* phenomena (see below). Since normalized vectors of quantum data represent physical systems, the (discrete) evolution of systems can be viewed as a suitable transformation on Hilbert spaces. The evolution of a quantum register is *linear* and *unitary*. Giving an initial state $|\psi_1\rangle$, for each evolution to a state $|\psi_2\rangle$, there exists a *unitary* operator U such that $|\psi_2\rangle = U|\psi_1\rangle$. Informally, "unitary" referred to an algebraic operator on a suitable space means that the normalization constraint of the amplitudes ($\sum_i |\alpha_i|^2 = 1$) is preserved during the transformation. Thus, a quantum physical system, i.e. a normalized vector which represents our data, can be described in term of linear operators and in a *deterministic* way. In quantum computing we refer to a unitary operator U acting on a n-qubit quantum register as an n-qubit *quantum gate*. We can represent operators on the 2^n-dimensional Hilbert space $\ell^2(\{0,1\}^n)$ with respect to the standard basis of $\mathbb{C}^{2^n}$ as $2^n \times 2^n$ matrices, and it is possible to prove that to each unitary operator on a Hilbert Space it is possible to associate an algebraic representation. Matrices which represent unitary operators enjoy some important property: for example they are easily invertible (reversibility is one of the peculiar features of quantum computing). *The application of quantum gates to quantum registers represents the pure quantum computational step and captures the internal evolution of quantum systems.* The simplest quantum gates act on a single qubit: they are operators on the space $\ell^2(\{0,1\})$, represented in $\mathbb{C}^2$ by 2×2 complex matrices. For

example, the quantum gate X is the unitary operator which maps $|0\rangle$ to $|1\rangle$ and $|1\rangle$ to $|0\rangle$ and it is represented by the matrix

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Being a linear operator, it maps a linear combination of inputs to the corresponding linear combination of outputs, and so X maps the general qubit state $\alpha|0\rangle + \beta|1\rangle$ into the state $\alpha|1\rangle + \beta|0\rangle$ i.e

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix}$$

An interesting unitary gate is the *Hadamard gate* denoted by H which acts on the computational basis in the following way:

$$|0\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \qquad |1\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

The Hadamard gate, which therefore is given by the matrix

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

is useful when we want to create a superposition starting from a classical state. It also holds that $H(H(|c\rangle)) = |c\rangle$ for $c = \{0, 1\}$. 1-qubit quantum gates can be used in order to build gates acting on n-qubit quantum states. If we have a 2-qubit quantum system, we can apply a 1-qubit quantum gate only to one component of the system, and we implicitly apply the identity operator (the identity matrix) to the other one. For example suppose we want to apply X to the first qubit. The 2-qubits input $|\psi_1\rangle \otimes |\psi_2\rangle$ gets mapped to $X|\psi_1\rangle \otimes I|\psi_2\rangle = (X \otimes I)|\psi_1\rangle \otimes |\psi_2\rangle$.

The **CNOT** is one of the most important quantum operators. It is mathematically described by the standard operator $CNOT : \ell^2(\{0,1\}^2) \to \ell^2(\{0,1\}^2)$ defined by

$$\begin{array}{rclrcl} \mathbf{CNOT}|00\rangle & = & |00\rangle & \mathbf{CNOT}|10\rangle & = & |11\rangle \\ \mathbf{CNOT}|01\rangle & = & |01\rangle & \mathbf{CNOT}|11\rangle & = & |10\rangle \end{array}$$

Intuitively, **cnot** acts as follows: it takes two distinct quantum bits as inputs and complements the *target* bit (the second one) if the *control* bit (the first one) is 1; otherwise it does not perform any action. The control qubit is a "master" agent: its evolution in independent from the evolution of the target bit (if the first input of the cnot is $|\phi\rangle$ the output is the same); the target qubit is a "slave" agent: its evolution is controlled by the value of the first qubit. In some sense, a communication between the agents is required and the quantum circuit is a simple distributed system. By adopting this perspective, controlled operators like cnot acts as "synchronization points" between token (ground type occurrences) in our definition of quantum token machine: this is one of the main features of our semantics (see Section 5).

Not all quantum states can be viewed as composite systems. In other words, *if $|\psi\rangle$ is a state of a tensor product space $\mathcal{H}_1 \otimes \mathcal{H}_2$, it is not generally true that there exists $|\psi_1\rangle \in \mathcal{H}_1$ and $|\psi_2\rangle \in \mathcal{H}_2$ such that $|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle$.* Instead, *it is not always possible to decompose an n-qubit register as the tensorial product of n qubits*.

These non-decomposable registers are called *entangled* and enjoy properties that we cannot find in any object of classical physics (and therefore in classical data). If n qubits are entangled, they behave *as if connected*, independently of the real physical distance. The strength of quantum computation is essentially based on the existence of entangled states (see, for example, the *teleportation protocol* [16]).

Example 2 *The 2-qubit states* $|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$ *and* $|\psi\rangle = \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle$ *are entangled. The 2-qubit state* $|\phi\rangle = \alpha|00\rangle + \beta|01\rangle$ *is not entangled. Trivially, notice that it is possible to rewrite it in the mathematically equivalent form* $\phi = |0\rangle \otimes (\alpha|0\rangle + \beta|1\rangle)$.

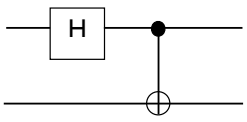
A simple way to create an entangled state is to fed a **CNOT** gate with a target qubit $|c\rangle$ and a particular control qubit, more precisely the output of the Hadamard gate applied to a base qubit, therefore a superposition $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ or $\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$. This composition of quantum gates is actually encoded by the terms defined in the Example 3.

We previously said that each n-ary unitary transformation (or composition of unitary transformations) can be represented by a suitable $n \times n$ matrix. From a computer science viewpoint, it is common to reason about quantum states transformations in terms of *quantum circuits*. Through the paper, we frequently say that "a lambda term encodes a quantum circuit". What does this mean? What is a quantum circuit? One more time, this is a long and complex subject and we refer to [16, 15] for a complete and exhaustive explanation. Since quantum circuits are invoked in the proof of Soundness Theorem 1, we give here some intuitions and a qualitative description (enough to understand the Soundness proof) of quantum circuits. We have introduced qubits to store quantum information, in analogy with the classical case. We have also introduced operations acting on them, i.e. quantum gates, and we can think about quantum gates in analogy with gates in classical logic circuits.

A quantum circuit on n qubits implements an unitary operator on a Hilbert space of dimension $\mathbb{C}^{2^n}$. This can be views as a primitive collection of quantum gates, each implementing a unitary operator on k (small) qubits.

It is useful to graphically represent quantum circuit in terms of sequential and parallel composition of quantum gates and wires, as for boolean circuits (notwithstanding, in the quantum case the graphical representation does not reflect the physical realization of the circuit).

For example, the following diagram represents the quantum circuit implemented by the term in Example 3.



The calculus $\mathsf{Q}\Lambda$ is purely linear (see Section 4). Each (well typed) lambda terms encode a quantum transformation or, equivalently, a quantum circuit built on the set of (the constants representing)

quantum gates occurring in the lambda-term.

One of the primitive operations in information theory is the copy of a datum. When we deal with quantum data as qubits, quantum information suffers from lack of accessibility in comparison to classical one. In fact, a quantum bit *can not be duplicated*. This curious feature is well-know in literature as *no-cloning property*: it does not allow to make a copy of an unknown quantum state (it is only possible to duplicate "trivial" qubits, i.e. basis states $|0\rangle$ and $|1\rangle$). In other words, it is not possible to build a quantum transformation/a quantum circuit able to maps an arbitrary quantum state $|\psi\rangle$ into the state $|\psi\rangle \otimes |\psi\rangle$. No-cloning property is one of the main difference between classical and quantum data and any paradigmatic quantum language has deal with to this fact. Notwithstanding, even if no-cloning property made the design of quantum languages more challenging, quantum data enjoy some properties (which have no classical counterpart) which can be exploited in the design of quantum algorithms.

4 The Calculus $\mathsf{Q}\Lambda$

An essential property of quantum programs is that quantum data, i.e. quantum bits, should always be uniquely referenced. This restriction follows from the well-known *no-cloning* and *no-erasing* properties of quantum physics, which state that a quantum bit cannot be duplicated nor canceled [16]. Syntactically, one captures this restriction by means of linearity: if every abstraction $\lambda x.M$ is such that there is exactly one free occurrence of x in M, then the substitution triggered by firing *any* redex is neither copying nor cancelling and, as a consequence, coherent with the just stated principles.

In this Section, we introduce a quantum linear λ-calculus in the style of van Tonder's λ_q [21] and give an equational theory for it. This is the main object of study of this paper, and is the calculus for which we will give a wave-style token machine in the coming sections.

4.1 The Language of Terms

Let us fix a finite set $\mathscr{U}$ of *unitary operators*, each on a finite-dimensional Hilbert space $\mathbb{C}^{2^n}$, where n can be arbitrary. To each such $\mathbf{U} \in \mathscr{U}$ we associate a symbol U and call n the *arity* of U. The syntactic categories of *patterns*, *bits*, *constants* and *terms* are defined by the following grammar:

$$\begin{array}{lll} \pi & ::= & x \mid \langle x, y\rangle; \qquad \textit{patterns} \\ B & ::= & |0\rangle_n \mid |1\rangle_n; \qquad \textit{bits} \\ C & ::= & B \mid U; \qquad \textit{constants} \\ M, N & ::= & x \mid C \mid M \otimes N \mid MN \mid \lambda\pi.M; \qquad \textit{terms} \end{array}$$

where n ranges over $\mathbb{N}$ and x ranges over a denumerable, totally ordered set of variables $\mathbb{V}$. We always assume that the natural numbers occurring next to bits in any term M are pairwise distinct. This condition, by the way, is preserved by substitution when the substituted variable occurs (free) exactly once. Whenever this does not cause ambiguity, we elide labels and simply write $|b\rangle$ for a bit. Notice that pairs can be formed via the binary operator $\otimes$. We will sometime write $|b_1 b_2 \dots b_k\rangle$ for $|b_1\rangle \otimes |b_2\rangle \otimes \dots \otimes |b_k\rangle$ (where $b_1, \dots, b_n \in \{0, 1\}$). In the following, capital letters such as M, N, L, Q (possibly indexed), denote terms. We work modulo variable renaming; in other words, terms are equivalence classes modulo α-conversion. Substitution up to α-equivalence is defined in the usual way. Observe that the terms of $\mathsf{Q}\Lambda$ are the ones of a λ-calculus with pairs (which are accessed by pattern-matching) endowed with constants for bits and unitary operators. We don't consider measurements here, and discuss the possibility of extending the language of terms in Section 8.

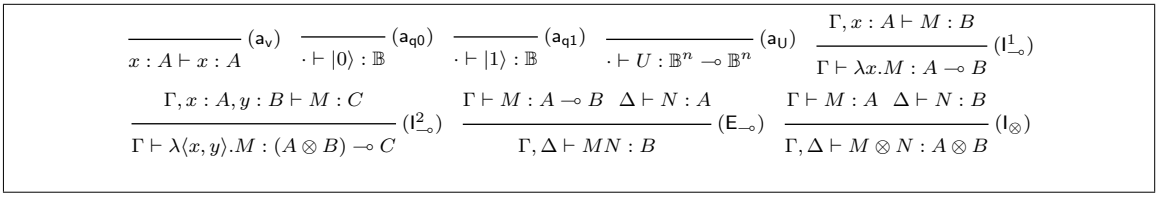


Figure 2: Typing Rules

4.2 Judgements and Typing Rules

Since in QΛ all terms are assumed to be non-duplicable by default, we adopt a linear type-discipline. Formally, the set of types is defined as

$$A ::= \mathbb{B} \mid \ A \multimap B \mid A \otimes B,$$

where $\mathbb{B}$ is the ground type of qubits. We write $\mathbb{B}^n$ for the n-fold tensor product

$$\underbrace{\mathbb{B} \otimes \ldots \otimes \mathbb{B}}_{n \text{ times}}.$$

Judgements are defined from a linear notion of *environment*.

- A *linear environment* Γ is a (possibly empty) finite set of assignments in the form $x : A$. We impose that in a linear environment, each variable x occurs *at most* once.
- If Γ and Δ are two linear environments assigning types to distinct sets of variables, Γ, Δ is their union.
- A *judgement* is an expression $\Gamma \vdash M : A$, where Γ is a linear environment, M is a term, and A is a type in QΛ.

Typing rules are in Figure 2. Observe that contexts are treated multiplicatively and, as a consequence, variables always appear exactly once in terms. In other words, a *strictly linear type discipline* is enforced.

Example 3 *Consider the following term:*

$$M_{EPR} = \lambda\langle x, y\rangle.CNOT(Hx \otimes y).$$

M_{EPR} encodes the quantum circuit which takes two input qubits and returns an entangled *state (a quantum state that cannot in general be expressed as the tensor product of single qubits). It can be given the type $\mathbb{B} \otimes \mathbb{B} \multimap \mathbb{B} \otimes \mathbb{B}$ in the empty context. Indeed, here is a type derivation π_{EPR} for it:*

$$\dfrac{\dfrac{\cdot \vdash CNOT : \mathbb{B} \otimes \mathbb{B} \multimap \mathbb{B} \otimes \mathbb{B} \qquad \dfrac{\dfrac{\cdot \vdash H : \mathbb{B} \multimap \mathbb{B} \quad x : \mathbb{B} \vdash x : \mathbb{B}}{x : \mathbb{B} \vdash Hx : \mathbb{B}}\,(\mathsf{E}_{\multimap}) \qquad y : \mathbb{B} \vdash y : \mathbb{B}}{x : \mathbb{B}, y : \mathbb{B} \vdash Hx \otimes y : \mathbb{B} \otimes \mathbb{B}}\,(\mathsf{I}_{\otimes})}{x : \mathbb{B}, y : \mathbb{B} \vdash CNOT(Hx \otimes y) : \mathbb{B} \otimes \mathbb{B}}\,(\mathsf{E}_{\multimap})}{\cdot \vdash M_{EPR} : \mathbb{B} \otimes \mathbb{B} \multimap \mathbb{B} \otimes \mathbb{B}}\,(\mathsf{I}^2_{\multimap})$$

M_{EPR} and π_{EPR} will be used as running examples in the rest of this paper, together with the following type derivation ρ_{EPR}:

$$\dfrac{\pi_{EPR} \triangleright \cdot \vdash M_{EPR} : \mathbb{B} \otimes \mathbb{B} \multimap \mathbb{B} \otimes \mathbb{B} \qquad \dfrac{\cdot \vdash |0\rangle_1 : \mathbb{B} \quad \cdot \vdash |1\rangle_2 : \mathbb{B}}{\cdot \vdash |0\rangle_1 \otimes |1\rangle_2 : \mathbb{B} \otimes \mathbb{B}}\,(\mathsf{I}_\otimes)}{\cdot \vdash M_{EPR}(|0\rangle_1 \otimes |1\rangle_2) : \mathbb{B} \otimes \mathbb{B}}\,(\mathsf{E}_\multimap)$$

If $\pi \triangleright \Gamma \vdash (\lambda x.M)N : A$, one can build a type derivation $\pi^{\Downarrow}$ with conclusion $\Gamma \vdash M\{x/N\} : A$ in a canonical way, by going through a constructive subsitution lemma. Similarly when $\pi \triangleright \Gamma \vdash (\lambda\langle x, y\rangle.M)(N \otimes L) : A$.

Lemma 1 *If $\pi \triangleright \Gamma, x_1 : A_1, \ldots, x_n : A_n \vdash M : B$ and for every $1 \leq i \leq n$ there is $\rho_i \triangleright \Delta_i \vdash N_i : A_i$, then there is a canonically defined derivation $\pi\{x_1, \ldots, x_n/\rho_1, \ldots, \rho_n\}$ of $\Gamma, \Delta_1, \ldots, \Delta_n \vdash M\{x_1, \ldots, x_n/N_1, \ldots, N_n\} : B$.*

Proof. Just proceed by the usual, simple induction on π. □

The notion of type derivation π of a term M and the related definition of $\pi^{\Downarrow}$, the type derivation of the reduct of M, will be generalized in the following section taking into account quantum superposition.

4.3 An Equational Theory

The λ-calculus is usually endowed with notions of reduction or equality, both centered around the β-rule, according to which a function $\lambda x.M$ applied to an argument N *reduces to* (or *can be considered equal to*) the term $M\{N/x\}$ obtained by replacing all free occurrences of x with N. A reduction relation implicitly provides the underlying calculus with a notion of computation, while an equational theory is more akin to a reasoning technique. Giving a reduction relation on $\mathsf{Q}\Lambda$ terms directly, however, is problematic. What happens when a n-ary unitary operator U is faced with an n-tuple of qubits $|b_1 \ldots b_n\rangle$? Superposition should somehow arise, but how can we capture it?

In this section, an equational theory for $\mathsf{Q}\Lambda$ will be introduced. In the next sections, we will prove that the semantics induced by token machines is *sound* with respect to it. The equational theory we are going to introduce will be a binary relation on formal, weighted sums of type derivations for $\mathsf{Q}\Lambda$ terms.

Definition 1 (Superposed Type Derivation) *A* superposed type derivation *of type (Γ, A) is a formal sum*

$$\mathcal{T} = \sum_{i=1}^{n} \kappa_i \pi_i$$

where for every $1 \leq i \leq n$, $\kappa_i \in \mathbb{C}$ and it holds that $\pi_i \triangleright \Gamma \vdash M_i : A$. In this case, we write $\Gamma \vdash \mathcal{T} : A$. Superposed type derivations will be denoted by metavariables like $\mathcal{T}$ or $\mathcal{S}$.

Please, notice that:

- If $\pi \triangleright \cdot \vdash U|b_1 \ldots b_k\rangle$, then $\pi^{\Downarrow}$ is a superposed type derivation in the form $\sum_{x \in B_k} \kappa_x \pi_x$, where B_k is the set of all binary strings of length k, π_x is the trivial type derivation for $|x\rangle$, and κ_x is the complex number corresponding to $|x\rangle$ in the vector $\mathbf{U}|b_1 \ldots b_k\rangle$.

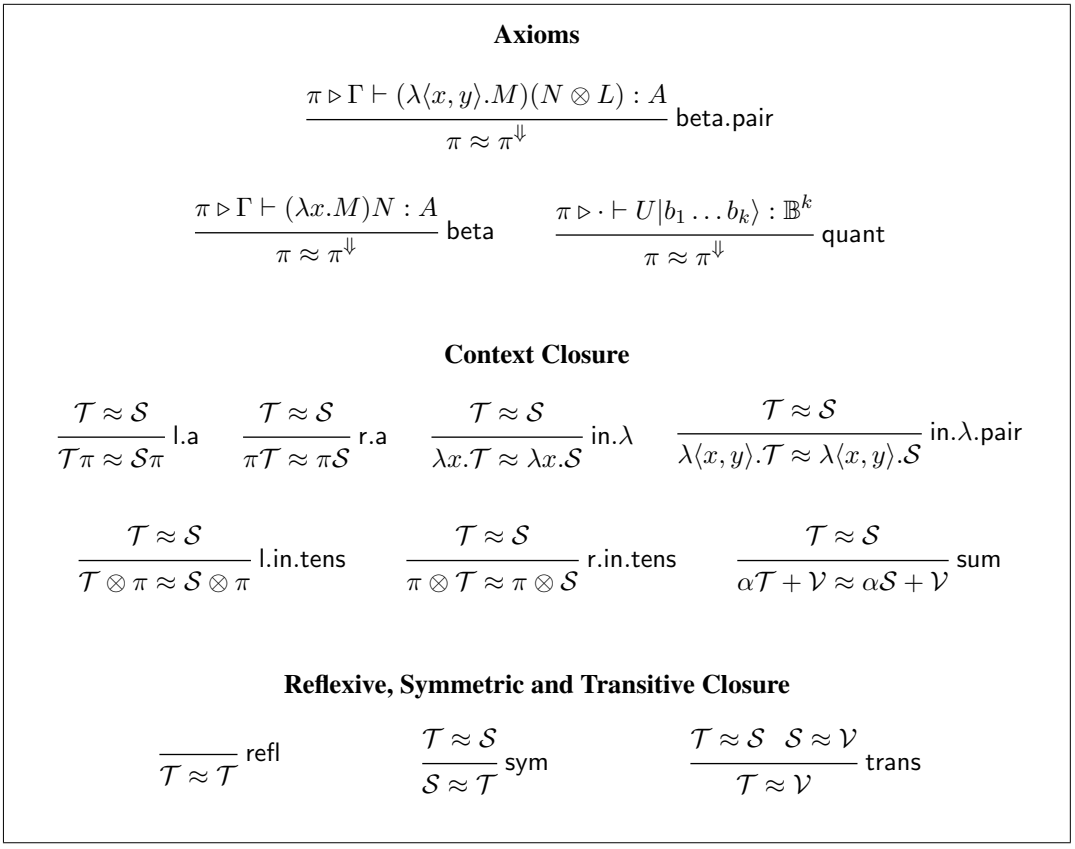


Figure 3: Equational Theory

- If $\pi \triangleright \Gamma \vdash (\lambda x.M)N : A$, $\pi^{\Downarrow}$ is the type derivation with conclusion $\Gamma \vdash M\{x/N\} : A$ built in a canonical way, by going through a constructive subsitution lemma. Similarly when $\pi \triangleright \Gamma \vdash (\lambda\langle x,y\rangle.M)(N \otimes L) : A$.
- All the term constructs can be generalized to operators on superposed type derivations, with the proviso that the types match. As an example if $\mathcal{T} = \sum_i \alpha_i \pi_i$ where $\pi_i \triangleright \Gamma \vdash M_i : A \multimap B$ and $\rho \triangleright \Delta \vdash N : A$, $\mathcal{T}\rho$ denotes the superposed type derivation $\mathcal{S} = \sum_i \alpha_i \sigma_i$ where $\sigma_i \triangleright \Gamma, \Delta \vdash M_i N : B$ and each σ_i is obtained applying the rule $(\mathsf{E}_{\multimap})$ to π_i and ρ.

A binary relation $\approx$ on superposed type derivations having the same type can be given by way of the rules in Figure 3, where we tacitly assume that the involved superposed type derivations have the appropriate type whenever needed. Notice that $\approx$ is by construction an equivalence relation. When the underlying type derivation is clear from the context, we denote superposed derivations simply by superposed *terms*. As an example, consider the term $M_{EPR}(|0\rangle_1 \otimes |1\rangle_2)$ from Example 3 and the corresponding type derivation ρ_{EPR} for it . It is convenient to be able to reason as follows, directly on the former:

$$
\begin{aligned}
M_{EPR}(|0\rangle \otimes |1\rangle) &\approx CNOT(H|0\rangle \otimes |1\rangle) \\
&\approx \frac{1}{\sqrt{2}} CNOT(|0\rangle \otimes |1\rangle) + \frac{1}{\sqrt{2}} CNOT(|1\rangle \otimes |1\rangle) \\
&\approx \frac{1}{\sqrt{2}} |0\rangle \otimes |1\rangle + \frac{1}{\sqrt{2}} CNOT(|1\rangle \otimes |1\rangle) \\
&\approx \frac{1}{\sqrt{2}} |0\rangle \otimes |1\rangle + \frac{1}{\sqrt{2}} |1\rangle \otimes |0\rangle.
\end{aligned}
$$

Please observe that the equational theory we have just defined can *hardly* be seen as an operational semantics for $\mathsf{Q}\Lambda$. Although equations can of course be oriented, it is the very nature of a superposed type derivation which is in principle problematic from the point of view of quantum computation: what is the mathematical nature of a superposed type derivation? Is it an element of an Hilbert Space? And if so, of *which one*? If we consider a simple language such as $\mathsf{Q}\Lambda$, the questions above may appear overly rhetorical, but we claim they are not. For example, what would be the quantum meaning of linear beta-reduction? If we want to design beta-reduction according to the principles of quantum computation, it has to be, at least, easily reversible (unless measurement is implicit in it). Moving towards more expressive languages, this non-trivial issue becomes more difficult and a number of constraints have to be imposed (for example, superposition of terms can be allowed, but only between "homogenous" terms, i.e. terms which have an identical skeleton [21]). This is the reason for which promising calculi [21] fail to be canonical models for quantum programming languages. This issue has been faced in literature without satisfactory answers, yielding a number of convincing arguments in favor of the (implicit or explicit) classical control of quantum data [3, 18].

4.3.1 Equational Theory Derivations in Normal Form

Sometime it is quite useful to assume that a derivation for $\mathcal{T} \approx \mathcal{S}$ is in a peculiar form, defined by giving an order on the rules in Figure 3. More specifically, define the following two sets of rules:

$$
\begin{aligned}
\mathsf{AX} &= \{\mathsf{beta}, \mathsf{beta.pair}, \mathsf{quant}\}; \\
\mathsf{CC} &= \{\mathsf{l.a}, \mathsf{r.a}, \mathsf{in}.\lambda, \mathsf{in}.\lambda.\mathsf{pair}, \mathsf{l.in.tens}, \mathsf{r.in.tens}\}.
\end{aligned}
$$

A derivation of $\mathcal{T} \approx \mathcal{S}$ is said to be *in normal form* (and we write $\mathcal{T} \sim \mathcal{S}$) iff:

- either the derivation is obtained by applying rule refl;
- or any branch in the derivation consists in instances of rules from AX, possibly followed by instances of rules in CC, possibly followed by instances of sum, possibly followed by instances of sym, possibly followed by instances of trans.

In other words, a derivation of $\mathcal{T} \approx \mathcal{S}$ is in normal form iff rules are applied in a certain order. As an example, we cannot apply transitivity or symmetry closure rules too early, i.e., before context closure rules. One may wonder whether this restricts the class of provable equivalences. Infact it does not:

Proposition 1 *$\mathcal{T} \approx \mathcal{S}$ iff $\mathcal{T} \sim \mathcal{S}$.*

Proof. If $\mathcal{T} \sim \mathcal{S}$, then of course $\mathcal{T} \approx \mathcal{S}$. The converce can be showed by induction on the *height* n of a proof of $\mathcal{T} \approx \mathcal{S}$, enriching the thesis by prescribing that the *height* of the obtained proof of $\mathcal{T} \approx \mathcal{S}$ must be at most n:

- If $\mathcal{T} \approx \mathcal{S}$ is proved by rules in AX or by refl, then by definition $\mathcal{T} \sim \mathcal{S}$.

- If $\mathcal{T} \approx \mathcal{S}$ is derived by rules in CC from a proof π, then:
  - If the rules in π are all from AX and CC, then there is nothing to do.
  - If the last rule in π is sum, then we can apply one of the following transformations, so as to be able to apply the induction hypothesis:

$$\dfrac{\dfrac{\mathcal{V} \approx \mathcal{X}}{\alpha\mathcal{V} + \mathcal{W} \approx \alpha\mathcal{X} + \mathcal{W}}\,\mathsf{sum}}{\alpha\mathcal{V}\pi + \mathcal{W}\pi \approx \alpha\mathcal{X}\pi + \mathcal{W}\pi}\,\mathsf{l.a} \quad \Longrightarrow \quad \dfrac{\dfrac{\mathcal{V} \approx \mathcal{X}}{\mathcal{V}\pi \approx \mathcal{X}\pi}\,\mathsf{l.a}}{\alpha\mathcal{V}\pi + \mathcal{W}\pi \approx \alpha\mathcal{X}\pi + \mathcal{W}\pi}\,\mathsf{sum}$$

$$\dfrac{\dfrac{\mathcal{V} \approx \mathcal{X}}{\alpha\mathcal{V} + \mathcal{W} \approx \alpha\mathcal{X} + \mathcal{W}}\,\mathsf{sum}}{\alpha\pi\mathcal{V} + \pi\mathcal{W} \approx \alpha\pi\mathcal{X} + \pi\mathcal{W}}\,\mathsf{r.a} \quad \Longrightarrow \quad \dfrac{\dfrac{\mathcal{V} \approx \mathcal{X}}{\pi\mathcal{V} \approx \pi\mathcal{X}}\,\mathsf{r.a}}{\alpha\pi\mathcal{V} + \pi\mathcal{W} \approx \alpha\pi\mathcal{X} + \pi\mathcal{W}}\,\mathsf{sum}$$

$$\dfrac{\dfrac{\mathcal{V} \approx \mathcal{X}}{\alpha\mathcal{V} + \mathcal{W} \approx \alpha\mathcal{X} + \mathcal{W}}\,\mathsf{sum}}{\alpha\lambda x.\mathcal{V} + \lambda x.\mathcal{W} \approx \alpha\lambda x.\mathcal{X} + \lambda x.\mathcal{W}}\,\mathsf{in.}\lambda \quad \Longrightarrow \quad \dfrac{\dfrac{\mathcal{V} \approx \mathcal{X}}{\lambda x.\mathcal{V} \approx \lambda x.\mathcal{X}}\,\mathsf{in.}\lambda}{\alpha\lambda x.\mathcal{V} + \lambda x.\mathcal{W} \approx \alpha\lambda x.\mathcal{X} + \lambda x.\mathcal{W}}\,\mathsf{sum}$$

$$\dfrac{\dfrac{\dfrac{\mathcal{V} \approx \mathcal{X}}{\alpha\mathcal{V} + \mathcal{W} \approx \alpha\mathcal{X} + \mathcal{W}}\,\mathsf{sum}}{\alpha\lambda\langle x,y\rangle.\mathcal{V} + \lambda\langle x,y\rangle.\mathcal{W} \approx \alpha\lambda\langle x,y\rangle.\mathcal{X} + \lambda\langle x,y\rangle.\mathcal{W}}\,\mathsf{in.}\lambda\mathsf{.pair}}{\alpha\lambda\langle x,y\rangle.\mathcal{V} + \lambda\langle x,y\rangle.\mathcal{W} \approx \alpha\lambda\langle x,y\rangle.\mathcal{X} + \lambda\langle x,y\rangle.\mathcal{W}}\,\mathsf{sum} \quad \Longrightarrow \quad \dfrac{\dfrac{\mathcal{V} \approx \mathcal{X}}{\lambda\langle x,y\rangle.\mathcal{V} \approx \lambda\langle x,y\rangle.\mathcal{X}}\,\mathsf{in.}\lambda\mathsf{.pair}}{\alpha\lambda\langle x,y\rangle.\mathcal{V} + \lambda\langle x,y\rangle.\mathcal{W} \approx \alpha\lambda\langle x,y\rangle.\mathcal{X} + \lambda\langle x,y\rangle.\mathcal{W}}\,\mathsf{sum}$$

$$\dfrac{\dfrac{\mathcal{V} \approx \mathcal{X}}{\alpha\mathcal{V} + \mathcal{W} \approx \alpha\mathcal{X} + \mathcal{W}}\,\mathsf{sum}}{\alpha\mathcal{V} \otimes \pi + \mathcal{W} \otimes \pi \approx \alpha\mathcal{X} \otimes \pi + \mathcal{W} \otimes \pi}\,\mathsf{l.in.tens} \quad \Longrightarrow \quad \dfrac{\dfrac{\mathcal{V} \approx \mathcal{X}}{\mathcal{V} \otimes \pi \approx \mathcal{X} \otimes \pi}\,\mathsf{l.in.tens}}{\alpha\mathcal{V} \otimes \pi + \mathcal{W}\pi\otimes \approx \alpha\mathcal{X} \otimes \pi + \mathcal{W} \otimes \pi}\,\mathsf{sum}$$

$$\dfrac{\dfrac{\mathcal{V} \approx \mathcal{X}}{\alpha\mathcal{V} + \mathcal{W} \approx \alpha\mathcal{X} + \mathcal{W}}\,\mathsf{sum}}{\alpha\pi \otimes \mathcal{V} + \pi \otimes \mathcal{W} \approx \alpha\pi \otimes \mathcal{X} + \pi \otimes \mathcal{W}}\,\mathsf{r.in.tens} \quad \Longrightarrow \quad \dfrac{\dfrac{\mathcal{V} \approx \mathcal{X}}{\pi \otimes \mathcal{V} \approx \pi \otimes \mathcal{X}}\,\mathsf{r.in.tens}}{\alpha\pi \otimes \mathcal{V} + \pi \otimes \mathcal{W} \approx \alpha\pi \otimes \mathcal{X} + \pi \otimes \mathcal{W}}\,\mathsf{sum}$$

  - If the last rule in π is sym or trans, then we can easily apply similar transformations, so as to be able to apply the induction hypothesis.
  - If the last rule in π is refl, then we can derive $\mathcal{T} \approx \mathcal{S}$ by a single application of refl.
- If $\mathcal{T} \approx \mathcal{S}$ is derived by sum from a proof π, then:
  - If the rules in π are all from AX or CC, or are sum, then there is nothing to do.
  - If the last rule in π is sym, then we can apply the following transformation, so as to be able to apply the induction hypothesis:

$$\dfrac{\dfrac{\mathcal{V} \approx \mathcal{X}}{\mathcal{X} \approx \mathcal{V}}\,\mathsf{sym}}{\alpha\mathcal{X} + \mathcal{W} \approx \alpha\mathcal{V} + \mathcal{W}}\,\mathsf{sum} \quad \Longrightarrow \quad \dfrac{\dfrac{\mathcal{V} \approx \mathcal{X}}{\alpha\mathcal{V} + \mathcal{W} \approx \alpha\mathcal{X} + \mathcal{W}}\,\mathsf{sum}}{\alpha\mathcal{X} + \mathcal{W} \approx \alpha\mathcal{V} + \mathcal{W}}\,\mathsf{sym}$$

  - If the last rule in π is trans, then we can apply the following transformation, so as to be able to apply the induction hypothesis

$$\dfrac{\dfrac{\mathcal{V}\approx\mathcal{X}\quad\mathcal{X}\approx\mathcal{Y}}{\mathcal{V}\approx\mathcal{Y}}\,\mathsf{trans}}{\alpha\mathcal{V}+\mathcal{W}\approx\alpha\mathcal{Y}+\mathcal{W}}\,\mathsf{sum}\qquad\Longrightarrow\qquad\dfrac{\dfrac{\mathcal{V}\approx\mathcal{X}}{\alpha\mathcal{V}+\mathcal{W}\approx\alpha\mathcal{X}+\mathcal{W}}\,\mathsf{sum}\qquad\dfrac{\mathcal{X}\approx\mathcal{Y}}{\alpha\mathcal{X}+\mathcal{W}\approx\alpha\mathcal{Y}+\mathcal{W}}\,\mathsf{sum}}{\alpha\mathcal{X}+\mathcal{W}\approx\alpha\mathcal{Y}+\mathcal{W}}\,\mathsf{trans}$$

  - If the last rule in π is refl, then we can derive $\mathcal{T}\approx\mathcal{S}$ by a single application of refl.
- If $\mathcal{T}\approx\mathcal{S}$ is derived by sym from a proof π, then:
  - If the rules in π are all from AX or CC, or are sum or sym, then there is nothing to do.
  - If the last rule in π is trans, then we can apply the following transformation, so as to be able to apply the induction hypothesis:

$$\dfrac{\dfrac{\mathcal{V}\approx\mathcal{X}\quad\mathcal{X}\approx\mathcal{Y}}{\mathcal{V}\approx\mathcal{Y}}\,\mathsf{trans}}{\mathcal{Y}\approx\mathcal{V}}\,\mathsf{sym}\qquad\Longrightarrow\qquad\dfrac{\dfrac{\mathcal{X}\approx\mathcal{Y}}{\mathcal{Y}\approx\mathcal{X}}\,\mathsf{sym}\qquad\dfrac{\mathcal{V}\approx\mathcal{X}}{\mathcal{X}\approx\mathcal{V}}\,\mathsf{sym}}{\mathcal{Y}\approx\mathcal{V}}\,\mathsf{trans}$$

  - If the last rule in π is refl, then we can derive $\mathcal{T}\approx\mathcal{S}$ by a single application of refl.
- If $\mathcal{T}\approx\mathcal{S}$ is derived by trans from two proofs of π and ρ, then if either π or ρ is derived by refl, then the required proof is already in our hand. Otherwise, there is nothing to do.

This concludes the proof. □

5 A Token Machine for $\mathsf{Q}\Lambda$

In this section we describe an interpretation of $\mathsf{Q}\Lambda$ type derivations in terms of a specific token machine called $\mathsf{IAM}_{\mathsf{Q}\Lambda}$.

With a slight abuse of notation, a permutation $\sigma : \{1,\dots,n\} \to \{1,\dots,n\}$ will be often applied to sequences of length n with the obvious meaning: $\sigma(a_1,\dots,a_n) = a_{\sigma(1)},\dots,a_{\sigma(n)}$. Similarly, such a permutation can be seen as the *unique* unitary operator on $\mathbb{C}^{2^n}$ which sends $|b_1\dots b_n\rangle$ to $\left|b_{\sigma(1)}\dots b_{\sigma(n)}\right\rangle$. Suppose given an operator $\mathbf{U} \in \mathscr{U}$ of arity $n \in \mathbb{N}$. Now, take a natural number $m \geq n$ and n distinct natural numbers $j_1,\dots,j_n$, all of them smaller or equal to m. With $\mathbf{U}_m^{j_1,\dots,j_n}$ (or simply with $\mathbf{U}^{j_1,\dots,j_n}$) we indicate the operator of arity m which acts like $\mathbf{U}$ on the qubits indexed with $j_1,\dots,j_n$ and leave all the other qubits unchanged.

In the following, with a slight abuse of notation, occurrences of types in type derivations are confused with types themselves. On the other hand, occurrences of types *inside other types* will be defined quite precisely, as follows. *Contexts* (types with an hole) are denoted by metavariables like C, D. A context C is said to be *a context for a type* A if $C[\mathbb{B}] = A$. *Negative contexts* (i.e., contexts where the hole is in negative position) are denoted by metavariables like N, M. *Positive* ones are denoted by metavariables like P, Q. An *occurrence* of $\mathbb{B}$ in the type derivation π is a pair (A, C), where A is an occurrence of a type in π and C is a context for A. Sequences of occurrences are indicated with metavariables like φ, ψ (possibly indexed). All sequences of occurrences we will deal with do not contain duplicates. Type constructors $\multimap$ and $\otimes$ can be generalized to operators on occurrences and sequences of occurrences, e.g. $(A, C) \multimap B$ is just $(A \multimap B, C \multimap B)$.

Given (an occurrence of) a type A, all positive and negative occurrences of $\mathbb{B}$ inside A can be put

in sequences called $\mathcal{P}(A)$ and $\mathcal{N}(A)$ as follows (where $\cdot$ is sequence concatenation):

$$\begin{aligned}
\mathcal{P}(\mathbb{B}) &= (\mathbb{B}, [\cdot]);\\
\mathcal{N}(\mathbb{B}) &= \varepsilon;\\
\mathcal{P}(A \otimes B) &= (\mathcal{P}(A) \otimes B) \cdot (A \otimes \mathcal{P}(B));\\
\mathcal{N}(A \otimes B) &= (\mathcal{N}(A) \otimes B) \cdot (A \otimes \mathcal{N}(B));\\
\mathcal{P}(A \multimap B) &= (\mathcal{N}(A) \multimap B) \cdot (A \multimap \mathcal{P}(B));\\
\mathcal{N}(A \multimap B) &= (\mathcal{P}(A) \multimap B) \cdot (A \multimap \mathcal{N}(B)).
\end{aligned}$$

As an example, the positive occurrences in the type $\mathbb{B} \multimap \mathbb{B} \otimes \mathbb{B}$ should be the two rightmost ones. And indeed:

$$\begin{aligned}
\mathcal{P}(\mathbb{B} \multimap \mathbb{B} \otimes \mathbb{B}) &= (\mathcal{N}(\mathbb{B}) \multimap \mathbb{B} \otimes \mathbb{B}) \cdot (\mathbb{B} \multimap \mathcal{P}(\mathbb{B} \otimes \mathbb{B}))\\
&= \varepsilon \cdot (\mathbb{B} \multimap \mathcal{P}(\mathbb{B} \otimes \mathbb{B})) = \mathbb{B} \multimap \mathcal{P}(\mathbb{B} \otimes \mathbb{B})\\
&= (\mathbb{B} \multimap (\mathcal{P}(\mathbb{B}) \otimes \mathbb{B})) \cdot (\mathbb{B} \multimap (\mathbb{B} \otimes \mathcal{P}(\mathbb{B})))\\
&= (\mathbb{B}, \mathbb{B} \multimap ([\cdot] \otimes \mathbb{B})), (\mathbb{B}, \mathbb{B} \multimap (\mathbb{B} \otimes [\cdot])).
\end{aligned}$$

For every type derivation π, $\mathcal{B}(\pi)$ is the sequence of all occurrences of $\mathbb{B}$ in π which are introduced by the rules $(\mathsf{a_{q0}})$ and $(\mathsf{a_{q1}})$ (from Figure 2). Similarly, $\mathcal{V}(\pi)$ is the corresponding sequence of binary digits, seen as a vector in $\mathbb{C}^{2^{|\mathcal{B}(\pi)|}}$. Both in $\mathcal{B}(\pi)$ and in $\mathcal{V}(\pi)$, the order is the one induced by the natural number labeling the underlying bit in π. As an example, consider the following type derivation, and call it π:

$$\frac{\cdot \vdash |0\rangle_2 : \mathbb{B}_1 \quad \cdot \vdash |1\rangle_1 : \mathbb{B}_2}{\cdot \vdash |0\rangle_2 \otimes |1\rangle_1 : \mathbb{B}_3 \otimes \mathbb{B}_4}\,(\mathsf{I}_\otimes)$$

There are four occurrences of $\mathbb{B}$ in it, and we have indexed it with the first four positive natural numbers, just to be able to point at them without being forced to use the formal, context machinery. Only two of them, namely the upper ones, are introduced by instances of the rules $(\mathsf{a_{q0}})$ and $(\mathsf{a_{q1}})$. Moreover, the rightmost one serves to type a bit having an index (namely 1) greater than the one in the other instance (namely 2). As a consequence, $\mathcal{B}(\pi)$ is the sequence $\mathbb{B}_2, \mathbb{B}_1$. The two instances introduces bits 0 and 1; then $\mathcal{V}(\pi) = |1\rangle \otimes |0\rangle$. As another example, one can easily compute $\mathcal{B}(\pi_{EPR})$ and $\mathcal{V}(\pi_{EPR})$ (where π_{EPR} is from Example 3), finding out that both are the empty sequence.

Finally, we are able to define, for every π, the abstract machine $\mathcal{A}_\pi$ interpreting it:

- The *states* of $\mathcal{A}_\pi$ form a set $\mathcal{S}_\pi$ and are in the form $(O_1, \ldots, O_n, \mathbf{Q})$ where:
  - $O_1, \ldots, O_n$ are occurrences of the type $\mathbb{B}$ in π;
  - $\mathbf{Q}$ is a *quantum register* on n qubits, i.e. a normalized vector in $\mathbb{C}^{2^n}$ (see Section 3).
- The *transition relation* $\rightarrow_\pi \subseteq \mathcal{S}_\pi \times \mathcal{S}_\pi$ is defined based on π, following Figure 4 and Figure 5. In the latter, each of the $2n$ occurrences of $\mathbb{B}$ in the type of U is simply denoted through its index, and for every $1 \leq k \leq m$, i_k is the position of $\mathbb{B}_k$ in the sequence $(\varphi_1, \mathbb{B}_{j_1}, \varphi_2, \ldots, \varphi_m, \mathbb{B}_{j_m}, \varphi_{m+1})$.

The number of positive (negative, respectively) occurrences of $\mathbb{B}$ in the conclusion of π is said to be the *output arity* (the *input arity*, respectively) of π. Given a type derivation π, the relation $\rightarrow_\pi$ enjoys a strong form of confluence:

Proposition 2 (One-step Confluence of $\rightarrow_\pi$) *Let* $\mathsf{S}, \mathsf{R}, \mathsf{T} \in \mathcal{S}_\pi$ *be such that* $\mathsf{S} \rightarrow_\pi \mathsf{R}$ *and* $\mathsf{S} \rightarrow_\pi \mathsf{T}$. *Then either* $\mathsf{R} = \mathsf{T}$ *or there exists a state* U *such that* $\mathsf{R} \rightarrow_\pi \mathsf{U}$ *and* $\mathsf{T} \rightarrow_\pi \mathsf{U}$.

$$\frac{}{x : A_1 \vdash x : A_2}$$

$$\begin{array}{lcl} ((\varphi, (A_1, P), \psi), \mathbf{Q}) & \to_\pi & ((\varphi, (A_2, P), \psi), \mathbf{Q}) \\ ((\varphi, (A_2, N), \psi), \mathbf{Q}) & \to_\pi & ((\varphi, (A_1, N), \psi), \mathbf{Q}) \end{array}$$

$$\frac{\Gamma_1, x : A_1 \vdash M : B_1}{\Gamma_2 \vdash \lambda x.M : A_2 \multimap B_2}$$

$$\begin{array}{c} ((\varphi, (A_1, N), \psi), \mathbf{Q}) \to_\pi ((\varphi, (A_2 \multimap B_2, N \multimap B_2), \psi), \mathbf{Q}) \\ ((\varphi, (A_2 \multimap B_2, P \multimap B_2), \psi), \mathbf{Q}) \to_\pi ((\varphi, (A_1, P), \psi), \mathbf{Q}) \\ ((\varphi, (B_1, P), \psi), \mathbf{Q}) \to_\pi ((\varphi, (A_2 \multimap B_2, A_2 \multimap P), \psi), \mathbf{Q}) \\ ((\varphi, (A_2 \multimap B_2, A_2 \multimap N), \psi), \mathbf{Q}) \to_\pi ((\varphi, (B_1, N), \psi), \mathbf{Q}) \\ ((\varphi, (\Gamma_2, P), \psi), \mathbf{Q}) \to_\pi ((\varphi, (\Gamma_1, P), \psi), \mathbf{Q}) \\ ((\varphi, (\Gamma_1, N), \psi), \mathbf{Q}) \to_\pi ((\varphi, (\Gamma_2, N), \psi), \mathbf{Q}) \end{array}$$

$$\frac{\Gamma_1, x : A_1, y : B_1 \vdash M : C_1}{\Gamma_2 \vdash \lambda \langle x, y \rangle .M : (A_2 \otimes B_2) \multimap C_2}$$

$$\begin{array}{c} ((\varphi, (A_1, N), \psi), \mathbf{Q}) \to_\pi ((\varphi, (A_2 \otimes B_2 \multimap C_2, N \otimes B_2 \multimap C_2), \psi), \mathbf{Q}) \\ ((\varphi, (A_2 \otimes B_2 \multimap C_2, P \otimes B_2 \multimap C_2), \psi), \mathbf{Q}) \to_\pi ((\varphi, (A_1, P), \psi), \mathbf{Q}) \\ ((\varphi, (B_1, N), \psi), \mathbf{Q}) \to_\pi ((\varphi, (A_2 \otimes B_2 \multimap C_2, A_2 \otimes N \multimap B_2), \psi), \mathbf{Q}) \\ ((\varphi, (A_2 \otimes B_2 \multimap C_2, A_2 \otimes P \multimap C_2), \psi), \mathbf{Q}) \to_\pi ((\varphi, (B_1, P), \psi), \mathbf{Q}) \\ ((\varphi, (C_1, P), \psi), \mathbf{Q}) \to_\pi ((\varphi, (A_2 \otimes B_2 \multimap C_2, A_2 \otimes B_2 \multimap P), \psi), \mathbf{Q}) \\ ((\varphi, (A_2 \otimes B_2 \multimap C_2, A_2 \otimes B_2 \multimap N), \psi), \mathbf{Q}) \to_\pi ((\varphi, (C_1, N), \psi), \mathbf{Q}) \\ ((\varphi, (\Gamma_2, P), \psi), \mathbf{Q}) \to_\pi ((\varphi, (\Gamma_1, P), \psi), \mathbf{Q}) \\ ((\varphi, (\Gamma_1, N), \psi), \mathbf{Q}) \to_\pi ((\varphi, (\Gamma_2, N), \psi), \mathbf{Q}) \end{array}$$

$$\frac{\Gamma_1 \vdash M : A_1 \multimap B_1 \quad \Delta_1 \vdash N : A_2}{\Gamma_2, \Delta_2 \vdash MN : B_2}$$

$$\begin{array}{c} ((\varphi, (A_2, P), \psi), \mathbf{Q}) \to_\pi ((\varphi, (A_1 \multimap B_1, P \multimap B_1), \psi), \mathbf{Q}) \\ ((\varphi, (A_1 \multimap B_1, N \multimap B_1), \psi), \mathbf{Q}) \to_\pi ((\varphi, (A_2, N), \psi), \mathbf{Q}) \\ ((\varphi, (A_1 \multimap B_1, A_1 \multimap P), \psi), \mathbf{Q}) \to_\pi ((\varphi, (B_2, P), \psi), \mathbf{Q}) \\ ((\varphi, (B_2, N), \psi), \mathbf{Q}) \to_\pi ((\varphi, (A_1 \multimap B_1, A \multimap N), \psi), \mathbf{Q}) \\ ((\varphi, (\Gamma_2, P), \psi), \mathbf{Q}) \to_\pi ((\varphi, (\Gamma_1, P), \psi), \mathbf{Q}) \\ ((\varphi, (\Gamma_1, N), \psi), \mathbf{Q}) \to_\pi ((\varphi, (\Gamma_2, N), \psi), \mathbf{Q}) \\ ((\varphi, (\Delta_2, P), \psi), \mathbf{Q}) \to_\pi ((\varphi, (\Delta_1, P), \psi), \mathbf{Q}) \\ ((\varphi, (\Delta_1, N), \psi), \mathbf{Q}) \to_\pi ((\varphi, (\Delta_2, N), \psi), \mathbf{Q}) \end{array}$$

$$\frac{\Gamma_1 \vdash M : A_1 \quad \Delta_1 \vdash N : B_1}{\Gamma_2, \Delta_2 \vdash M \otimes N : A_2 \otimes B_2} (\mathsf{I}_\otimes)$$

$$\begin{array}{c} ((\varphi, (A_2 \otimes B_2, N \otimes B_2), \psi), \mathbf{Q}) \to_\pi ((\varphi, (A_1, N), \psi), \mathbf{Q}) \\ ((\varphi, (A_2 \otimes B_2, A_2 \otimes N), \psi), \mathbf{Q}) \to_\pi ((\varphi, (B_1, N), \psi), \mathbf{Q}) \\ ((\varphi, (A_1, P), \psi), \mathbf{Q}) \to_\pi ((\varphi, (A_2 \otimes B_2, P \otimes B_2), \psi), \mathbf{Q}) \\ ((\varphi, (B_1 P), \psi), \mathbf{Q}) \to_\pi ((\varphi, (A_2 \otimes B_2, A_2 \otimes P), \psi), \mathbf{Q}) \\ ((\varphi, (\Gamma_1, N), \psi), \mathbf{Q}) \to_\pi ((\varphi, (\Gamma_2, N), \psi), \mathbf{Q}) \\ ((\varphi, (\Delta_1, N), \psi), \mathbf{Q}) \to_\pi ((\varphi, (\Delta_2, N), \psi), \mathbf{Q}) \\ ((\varphi, (\Gamma_2, P), \psi), \mathbf{Q}) \to_\pi ((\varphi, (\Gamma_1, P), \psi), \mathbf{Q}) \\ ((\varphi, (\Delta_2, P), \psi), \mathbf{Q}) \to_\pi ((\varphi, (\Delta_1, P), \psi), \mathbf{Q}) \end{array}$$

Figure 4: Quantum GoI Machine — Classical Rules

$$\frac{}{\cdot \vdash U : \mathbb{B}_1 \otimes \ldots \otimes \mathbb{B}_m \multimap \mathbb{B}_{m+1} \otimes \ldots \otimes \mathbb{B}_{2m}}$$

$$\begin{array}{c} ((\varphi_1, \mathbb{B}_{j_1}, \varphi_2, \ldots, \varphi_m, \mathbb{B}_{j_m}, \varphi_{m+1}), \mathbf{Q}) \\ \to_\pi \\ ((\varphi_1, \mathbb{B}_{j_1+m}, \varphi_2, \ldots, \varphi_m, \mathbb{B}_{j_m+m}, \varphi_{m+1}), \mathbf{U}^{i_1,\ldots,i_m}(\mathbf{Q})) \end{array}$$

Figure 5: Quantum GoI Machine — Quantum Rules

Proof. By simply inspecting the various rules. Notice that there are no critical pairs in $\rightarrow_\pi$. □

Suppose, for the sake of simplicity, that π is a type derivation of $\cdot \vdash M : A$. An *initial state* for $\mathbf{Q}$ is a state in the form $(\mathcal{N}(A) \cdot \mathcal{B}(\pi), \mathbf{Q} \otimes \mathcal{V}(\pi))$. Given a permutation σ on n elements, a *final state for* $\mathbf{Q}$ *and* σ is one in the form $(\varphi, \mathbf{Q})$, where $\varphi = \sigma(\mathcal{P}(A))$.

Definition 2 *Given a type derivation π, the partial function* computed by π *is* $[\pi] : \mathbb{C}^{2^n} \rightharpoonup \mathbb{C}^{2^m}$ *(where n and m are the input and output arity of π) and is defined by stipulating that $[\pi](\mathbf{Q}) = \mathbf{R}$ iff any initial state for $\mathbf{Q}$ rewrites into a final state for $\mathbf{S}$ and σ, where $\mathbf{S} = \sigma^{-1}(\mathbf{R})$.*

Given a type derivation π, $[\pi]$ is either always undefined or always defined. Indeed, the fact any initial configuration (for, say, $\mathbf{Q}$) rewrites to a final configuration or not does *not* depend on $\mathbf{Q}$ but only on π:

Lemma 2 (Uniformity) *For every type derivation π and for every occurrences $O_1, \ldots, O_n$, $P_1, \ldots, P_n$, there is a unitary operator $\mathbf{U}$ such that whenever $(O_1, \ldots, O_n, \mathbf{Q}) \rightarrow_\pi (P_1, \ldots, P_n, \mathbf{R})$ it holds that $\mathbf{R} = \mathbf{U}(\mathbf{Q})$.*

Proof. Observe that for every $O_1, \ldots, O_n$, $P_1, \ldots, P_n$ there is *at most* one of the rules defining $\rightarrow_\pi$ which can be applied. Moreover, notice that each rule acts uniformly on the underlying quantum register. □

In the following section, we will prove that $[\pi]$ is always a total function, and that it makes perfect sense from a quantum point of view.

6 Main Properties of $\mathsf{IAM}_{\mathsf{Q}\Lambda}$

In this section, we will prove some crucial results about $\mathsf{IAM}_{\mathsf{Q}\Lambda}$. More specifically, we prove that runs of this token machine are indeed finite and end in final states. We proceed by putting $\mathsf{Q}\Lambda$ in correspondence to MLL, thus inheriting the same kind of very elegant and powerful results enjoyed by MLL token machines.

6.1 A Correspondence Between MLL and $\mathsf{Q}\Lambda$

Any type derivation π can be put in correspondence with *some* MLL proofs. We inductively define the map $(\cdot)^\bullet$ from $\mathsf{Q}\Lambda$ types to MLL formulas as follows:

$$(\mathbb{B})^\bullet = \alpha;$$
$$(A \multimap B)^\bullet = (A)^{\bullet\perp} \parr (B)^\bullet;$$
$$(A \otimes B)^\bullet = (A)^\bullet \otimes (B)^\bullet.$$

Given a judgment $J = \Gamma \vdash M : A$ and a natural number $n \in \mathbb{N}$, the MLL sequent *corresponding* to J and n is the following one:

$$\vdash \underbrace{\alpha^\perp, \ldots, \alpha^\perp}_{n \text{ times}}, ((B_1)^\bullet)^\perp, \ldots, ((B_m)^\bullet)^\perp, (A)^\bullet,$$

where $\Gamma = x_1 : B_1, \ldots, x_m : B_m$. For every π, we define now a set of MLL proofs $\mathscr{I}(\pi)$. This way, every type derivation π for $J = \Gamma \vdash M : A$ such that n bits occur in M, is put in relation to

possibly many MLL proofs of the sequent corresponding to J and n. One among them is called the *canonical proof* for π. The set $\mathscr{I}(\pi)$ and canonical proofs are defined by induction on the structure of the underlying type derivation π:

- If π is the type derivation
$$\frac{}{\cdot \vdash |0\rangle : \mathbb{B}}\,(\mathsf{a_{q0}}),$$
then the only proof ξ in $\mathscr{I}(\pi)$ is an atomic axiom. Similarly if the only rule in π is $(\mathsf{a_{q1}})$. Please notice that π contains *one* bit, and as a consequence ξ has the correct conclusion.
- If π is
$$\frac{}{\cdot \vdash U : \mathbb{B}^n \multimap \mathbb{B}^n}\,(\mathsf{a_U}),$$
then π is in correspondence to all of the $n!$ possible cut-free proofs of the sequent
$$\vdash (\underbrace{(\alpha \otimes \ldots \otimes \alpha)^\perp}_{n \text{ times}}\ \parr\ \underbrace{(\alpha \otimes \ldots \otimes \alpha)}_{n \text{ times}}$$
obtained by starting from n instances of an atomic axiom, gluing them together by the rule $\otimes$, and finally choosing one of the $n!$ possible permutations before applying n times rule $\parr$. The canonical proof is the one corresponding to the identity permutation.
- If π is the type derivation
$$\frac{}{x : A \vdash x : A}\,(\mathsf{a_v})$$
then the only proof corresponding to π is the following
$$\frac{}{\vdash (A)^{\bullet\perp}, (A)^\bullet}\,\mathbf{Ax}$$
- If π is
$$\frac{\rho \triangleright \Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \multimap B}\,(\mathsf{I}^1_\multimap)$$
where $\Gamma = x_1 : A_1, \ldots, x_m : A_m$. Then for all possible MLL proof $\mu \in \mathscr{I}(\rho)$ of the MLL sequent
$$\mathsf{J} = \vdash \underbrace{\alpha^\perp, \ldots, \alpha^\perp}_{n \text{ times}}, ((A_1)^\bullet)^\perp, \ldots, ((A_m)^\bullet)^\perp, ((A)^\bullet)^\perp, (B)^\bullet$$
the following MLL proof is in $\mathscr{I}(\pi)$:
$$\frac{\mu \triangleright \mathsf{J}}{\vdash \underbrace{\alpha^\perp, \ldots, \alpha^\perp}_{n \text{ times}}, ((A_1)^\bullet)^\perp, \ldots, ((A_m)^\bullet)^\perp, (A)^{\bullet\perp} \parr (B)^\bullet}\,\parr$$
- If π is
$$\frac{\rho \triangleright \Gamma, x : A, y : B \vdash M : C}{\Gamma \vdash \lambda\langle x, y\rangle.M : (A \otimes B) \multimap C}\,(\mathsf{I}^2_\multimap)$$
where $\Gamma = z_1 : D_1, \ldots, z_m : D_m, x : A, y : B$, then for all possible MLL proofs $\mu \in \mathscr{I}(\rho)$ of the MLL sequent
$$\mathsf{J} = \vdash \underbrace{\alpha^\perp, \ldots, \alpha^\perp}_{n \text{ times}}, ((D_1)^\bullet)^\perp, \ldots, ((D_m)^\bullet)^\perp, (A)^{\bullet\perp}, (B)^{\bullet\perp}, (C)^\bullet$$

the following MLL proof is in $\mathscr{I}(\pi)$:

$$\dfrac{\dfrac{\mu \triangleright \mathsf{J}}{\vdash \underbrace{\alpha^{\perp}, \ldots, \alpha^{\perp}}_{n \text{ times}}, ((D_1)^{\bullet})^{\perp}, \ldots, ((D_m)^{\bullet})^{\perp}, ((A)^{\bullet\perp} \parr (B)^{\bullet\perp}), (C)^{\bullet}}\,\parr}{\vdash \underbrace{\alpha^{\perp}, \ldots, \alpha^{\perp}}_{n \text{ times}}, ((D_1)^{\bullet})^{\perp}, \ldots, ((D_m)^{\bullet})^{\perp}, ((A)^{\bullet\perp} \parr (B)^{\bullet\perp}) \parr (C)^{\bullet}}\,\parr$$

- If π is

$$\dfrac{\rho \triangleright \Gamma \vdash M : A \multimap B \quad \sigma \triangleright \Delta \vdash N : A}{\Gamma, \Delta \vdash MN : B}\,(\mathsf{E}_{\multimap})$$

where $\Gamma = x_1 : A_1, \ldots, x_m : A_m$ and $\Delta = y_1 : B_1, \ldots, y_k : B_k$ then for all possible MLL proofs $\xi \in \mathscr{I}(\rho)$ and $\mu \in \mathscr{I}(\sigma)$ of the MLL sequents

$$\mathsf{H} \;=\; \vdash \underbrace{\alpha^{\perp}, \ldots, \alpha^{\perp}}_{n_1 \text{ times}}, ((A_1)^{\bullet})^{\perp}, \ldots, ((A_m)^{\bullet})^{\perp}, (A)^{\bullet\perp} \parr (B)^{\bullet}$$
$$\mathsf{G} \;=\; \vdash \underbrace{\alpha^{\perp}, \ldots, \alpha^{\perp}}_{n_2 \text{ times}}, ((B_1)^{\bullet})^{\perp}, \ldots, ((B_k)^{\bullet})^{\perp}, (A)^{\bullet}$$

the following MLL proof is in $\mathscr{I}(\pi)$:

$$\dfrac{\xi \triangleright \mathsf{H} \quad \dfrac{\mu \triangleright \mathsf{G} \quad \overline{\vdash (B)^{\bullet\perp}, (B)^{\bullet}}}{\vdash \underbrace{\alpha^{\perp}, \ldots, \alpha^{\perp}}_{n_2 \text{ times}}, ((B_1)^{\bullet})^{\perp}, \ldots, ((B_k)^{\bullet})^{\perp}, (A)^{\bullet} \otimes (B)^{\bullet\perp}, (B)^{\bullet}}\,\otimes}{\vdash \underbrace{\alpha^{\perp}, \ldots, \alpha^{\perp}}_{n_1 + n_2 \text{ times}}, ((A_1)^{\bullet})^{\perp}, \ldots, ((A_m)^{\bullet})^{\perp}, ((B_1)^{\bullet})^{\perp}, \ldots, ((B_k)^{\bullet})^{\perp}, (B)^{\bullet}}\,\mathsf{cut}$$

- If π is

$$\dfrac{\rho \triangleright \Gamma \vdash M : A \quad \sigma \triangleright \Delta \vdash N : B}{\Gamma, \Delta \vdash M \otimes N : A \otimes B}\,(\mathsf{I}_{\otimes})$$

where $\Gamma = x_1 : A_1, \ldots, x_m : A_m$ and $\Delta = y_1 : B_1, \ldots, y_k : B_k$, then for all possible MLL proofs $\xi \in \mathscr{I}(\rho)$ and $\mu \in \mathscr{I}(\sigma)$ of the MLL sequents

$$\mathsf{H} \;=\; \vdash \underbrace{\alpha^{\perp}, \ldots, \alpha^{\perp}}_{n_1 \text{ times}}, ((A_1)^{\bullet})^{\perp}, \ldots, ((A_m)^{\bullet})^{\perp}, (A)^{\bullet}$$
$$\mathsf{G} \;=\; \vdash \underbrace{\alpha^{\perp}, \ldots, \alpha^{\perp}}_{n_2 \text{ times}}, ((B_1)^{\bullet})^{\perp}, \ldots, ((B_k)^{\bullet})^{\perp}, (B)^{\bullet}$$

π is in correspondence to the MLL proof

$$\dfrac{\xi_1 \triangleright \mathsf{J}_1 \quad \xi_2 \triangleright \mathsf{J}_2}{\vdash \underbrace{\alpha^{\perp}, \ldots, \alpha^{\perp}}_{n_1 + n_2 \text{ times}}, ((A_1)^{\bullet})^{\perp}, \ldots, ((A_m)^{\bullet})^{\perp}, ((B_1)^{\bullet})^{\perp}, \ldots, ((B_k)^{\bullet})^{\perp}, (A)^{\bullet\perp} \otimes (B)^{\bullet\perp}}\,\otimes$$

Observe how $\mathscr{I}(\pi)$ is a singleton whenever π does not contain any unitary operator of arity (strictly) greater than 1.

Given an MLL proof ξ, let us denote as T_ξ the class of all finite sequences of atom occurrences in ξ. The relation $\mapsto_\xi$ can be extended to a relation on T_ξ by stipulating that

$$(O_1, \ldots, O_{n-1}, P, O_{n+1}, \ldots, O_m) \mapsto_\xi (O_1, \ldots, O_{n-1}, R, O_{n+1}, \ldots, O_m)$$

whenever $P \mapsto_\xi R$. As usual, $\mapsto_\xi^+$ is the transitive closure of $\mapsto_\xi$.

Let us now consider a type derivation π in $\mathsf{Q}\Lambda$ and its quantum token machine $\mathcal{A}_\pi$ and any $\xi \in \mathscr{I}(\pi)$. States of $\mathcal{A}_\pi$ can be mapped to T_ξ by simply forgetting the underlying quantum register and mapping any occurrence of π to the corresponding atom occurrence in ξ. This way one gets a map $\mathscr{R}_{\pi,\xi}(\cdot) : \mathcal{S}_\pi \to \mathsf{T}_\xi$ such that, given a state $\mathsf{S} = (O_1, \ldots, O_n, \mathbf{Q})$ in $\mathcal{S}_\pi$, $|\mathscr{R}_{\pi,\xi}(\mathsf{S})| = n$, number of occurrences in S is the same as the length of $\mathscr{R}_{\pi,\xi}(\mathsf{S})$. Each reduction step on the token machine $\mathcal{A}_\pi$ corresponds to *at least one* reduction step in the MLL machine $\mathcal{M}_\xi$, where $\xi \in \mathscr{I}(\pi)$ is the canonical proof:

Lemma 3 *Let us consider a token machine $\mathcal{A}_\pi$ and two states $\mathsf{S}, \mathsf{R} \in \mathcal{S}_\pi$. If $\mathsf{S} \to_\pi \mathsf{R}$ and $\xi \in \mathscr{I}(\pi)$ is canonical, then $\mathscr{R}_{\pi,\xi}(\mathsf{S}) \mapsto_\xi^+ \mathscr{R}_{\pi,\xi}(\mathsf{R})$.*

Proof. This goes by induction on the structure of π. □

Any (possible) pathological situation on the quantum token machine, then, can be brought back to a corresponding (absurd) pathological situation in the MLL token machine. This is the principle that will guide us in the rest of this section.

6.2 Termination

The first property we want to be sure about is that every computation of any token machine $\mathcal{A}_\pi$ always terminates. This is relatively simple to state and prove:

Proposition 3 (Termination) *Given a quantum token machine $\mathcal{A}_\pi$, any sequence $\mathsf{S} \to_\pi \mathsf{R} \to_\pi \ldots$ is finite.*

Proof. Suppose, for the sake of contradiction, than there exists an infinite computation in $\mathcal{A}_\pi$. This implies by Lemma 3 that there exists an infinite path in the token machine $\mathcal{M}_\xi$ where ξ is the canonical MLL proof for π. Absurd. □

6.3 Progress

Progress (i.e. deadlock-freedom) is more difficult to prove than termination. Again, however, we use in an essential way the correspondence between $\mathsf{Q}\Lambda$ and MLL:

Proposition 4 (Progress) *Suppose π is a type derivation in $\mathsf{Q}\Lambda$ and $\mathsf{S} \in \mathcal{S}_\pi$ is initial. Moreover, suppose that $\mathsf{S} \to_\pi^* \mathsf{R}$. Then either R is final or $\mathsf{R} \to_\pi \mathsf{T}$ for some $\mathsf{T} \in \mathcal{S}_\pi$.*

Given a type derivation π, an *argument occurrence* is any negative occurrence (A, N) of $\mathbb{B}$ in a (a_U) axiom. We extend this definition to the corresponding atom occurrence when $\xi \in \mathscr{I}(\pi)$. A *result occurrence* is defined similarly, but the occurrence has to be positive.

Proof. Let us consider a computation $\mathsf{S}_1 \rightarrow_\pi \ldots \rightarrow_\pi \mathsf{S}_k$ on a quantum token machine $\mathcal{A}_\pi$. Suppose that the state S_k is a deadlocked state, i.e. S_k is not a final state, and that there exists no S_m such that $\mathsf{S}_k \rightarrow_\pi \mathsf{S}_m$. The fact S_k is a deadlocked state means that $l \geq 1$ occurrences in S_k are argument occurrences, since the latter are the only points of synchronization of the machine. Let us consider any *maximal* sequence

$$\mathscr{R}_{\pi,\xi}(\mathsf{S}_1) \mapsto_\xi \ldots \mapsto_\xi \mathscr{R}_{\pi,\xi}(\mathsf{S}_k) \mapsto_\xi \mathsf{Q}_1 \mapsto_\xi \ldots \mapsto_\xi \mathsf{Q}_n, \tag{1}$$

where $\xi \in \mathscr{I}(\pi)$ is the canonical proof corresponding to π. Observe that in (1), all occurrences of atoms in ξ are visited exactly once, including those corresponding to argument and result occurrences from π. Notice, however, that the argument and result occurrences of the unitary operators affected by S_k cannot have been visited along the subsequence $\mathscr{R}_{\pi,\xi}(\mathsf{S}_1) \mapsto_\xi \ldots \mapsto_\xi \mathscr{R}_{\pi,\xi}(\mathsf{S}_k)$ (otherwise we would visit the occurrences in S_k at least twice, which is not possible). Now, form a directed graph whose nodes are the unitary constants $U_1, \ldots, U_h$ which block S_k, plus a node F (representing the conclusion of π), and whose edges are defined as follows:

- there is an edge from U_i to U_j iff along $\mathsf{Q}_1 \mapsto_\xi \ldots \mapsto_\xi \mathsf{Q}_n$ one of the l independent computations corresponding to a blocked occurrence in S_k is such that a result occurrence of U_i is followed by an argument occurrence of U_j and the occurrences between them are neither argument nor result occurrences.
- there is an edge from U_i to F iff along $\mathsf{Q}_1 \mapsto_\xi \ldots \mapsto_\xi \mathsf{Q}_n$ one of the l traces is such that a result occurrence of U_i is followed by a final occurrence of an atom and the occurrences between them are neither argument nor result occurrences.

The thus obtained graph has the following properties:

- Every node U_i has at least one incoming edge, because otherwise the configuration S_k would not be deadlocked.
- As a consequence, the graph must be cyclic, because otherwise we could topologically sort it and get a node with no incoming edges (meaning that some of the U_i would not be blocked!). Moreover, the cycle does not include F, because the latter only has incoming nodes.

From any cycle involving the U_j, one can induce the presence of a cycle in the token machine $\mathcal{M}_\mu$ for some $\mu \in \mathscr{I}(\pi)$. Indeed, such a μ can be formed by simply choosing, for each U_j, the "good" permutation, namely the one linking the incoming edge and the outgoing edge which are part of the cycle. This way, we have reached the absurd starting from the existence of a deadlocked computation. □

The token machine $\mathcal{A}_\pi$ can be built by following the structure of π. However, the fact this gives rise to a well-behaved, unitary, function requires proving some properties of $\mathcal{A}_\pi$ (i.e. termination and progress) externally. One may wonder whether this could be avoided by taking a categorical approach and apply the so-called **Int**-Construction [13] to the underlying category. This is not going to work, however, because finite dimensional Hilbert spaces and unitary maps on them are not a *traced* category. Of course, one could switch to linear maps, which indeed turn Hilbert spaces into a traced category; one loses the strong link with quantum computation this way, however.

6.4 Discussion

The immediate consequence of the termination and progress results from Section 6 is that $[\pi]$ is always a *total* function. The way $\mathcal{A}_\pi$ is defined ensures that $[\pi]$ is obtained by feeding some of the input of a unitary operator $\mathbf{U}$ with some bits (namely those occurring in π). $\mathbf{U}$ is itself obtained by composing the unitary operators occurring in π, which can thus be seen as a program computing a quantum

circuit, which we call $\langle \pi \rangle$. Of course, $[\pi]$ is nothing more than the function computed by $\langle \pi \rangle$. In a way, then, token machines both show that $\mathsf{Q}\Lambda$ is a true quantum calculus and can be seen as the right operational semantics for it.

Example 4 *Consider the term $M_{EPR} = \lambda\langle x, y\rangle.CNOT(Hx \otimes y)$ and a type derivation π for it:*

$$\dfrac{\dfrac{\cdot \vdash CNOT : \mathbb{B} \otimes \mathbb{B} \multimap \mathbb{B} \otimes \mathbb{B} \qquad \dfrac{\dfrac{\cdot \vdash H : \mathbb{B} \multimap \mathbb{B} \quad x : \mathbb{B} \vdash x : \mathbb{B}}{x : \mathbb{B} \vdash Hx : \mathbb{B}}(\mathsf{E}_{\multimap}) \qquad y : \mathbb{B} \vdash y : \mathbb{B}}{x : \mathbb{B}, y : \mathbb{B} \vdash Hx \otimes y : \mathbb{B} \otimes \mathbb{B}}(\mathsf{I}_{\otimes})}{x : \mathbb{B}, y : \mathbb{B} \vdash CNOT(Hx \otimes y) : \mathbb{B} \otimes \mathbb{B}}(\mathsf{E}_{\multimap})}{\cdot \vdash M_{EPR} : \mathbb{B} \otimes \mathbb{B} \multimap \mathbb{B} \otimes \mathbb{B}}(\mathsf{I}^2_{\multimap})$$

Forgetting about terms and marking different occurrences of $\mathbb{B}$ with distinct indices, we obtain:

$$\dfrac{\dfrac{\cdot \vdash \mathbb{B}_9 \otimes \mathbb{B}_{10} \multimap \mathbb{B}_{11} \otimes \mathbb{B}_{12} \qquad \dfrac{\dfrac{\cdot \vdash : \mathbb{B}_{21} \multimap \mathbb{B}_{22} \quad \mathbb{B}_{23} \vdash \mathbb{B}_{24}}{\mathbb{B}_{17} \vdash \mathbb{B}_{18}}(\mathsf{E}_{\multimap}) \qquad \mathbb{B}_{19} \vdash \mathbb{B}_{20}}{\mathbb{B}_{13}, \mathbb{B}_{14} \vdash \mathbb{B}_{15} \otimes \mathbb{B}_{16}}(\mathsf{I}_{\otimes})}{\mathbb{B}_5, \mathbb{B}_6 \vdash \mathbb{B}_7 \otimes \mathbb{B}_8}(\mathsf{E}_{\multimap})}{\cdot \vdash \mathbb{B}_1 \otimes \mathbb{B}_2 \multimap \mathbb{B}_3 \otimes \mathbb{B}_4}(\mathsf{I}^2_{\multimap})$$

Now, consider the $\mathsf{IAM}_{\mathsf{Q}\Lambda}$ computation:

$$\begin{aligned}
(\mathbb{B}_1, \mathbb{B}_2, \mathbf{Q}) &\to^*_\pi (\mathbb{B}_5, \mathbb{B}_6, \mathbf{Q}) \to^*_\pi (\mathbb{B}_{13}, \mathbb{B}_{14}, \mathbf{Q}) \\
&\to_\pi (\mathbb{B}_{17}, \mathbb{B}_{19}, \mathbf{Q}) \to^*_\pi (\mathbb{B}_{23}, \mathbb{B}_{20}, \mathbf{Q}) \\
&\to_\pi (\mathbb{B}_{24}, \mathbb{B}_{10}, \mathbf{Q}) \to_\pi (\mathbb{B}_{21}, \mathbb{B}_{10}, \mathbf{Q}) \\
&\to_\pi (\mathbb{B}_{22}, \mathbb{B}_{10}, \mathbf{H}^1(\mathbf{Q})) \to_\pi (\mathbb{B}_{18}, \mathbb{B}_{10}, \mathbf{H}^1(\mathbf{Q})) \\
&\to_\pi (\mathbb{B}_{15}, \mathbb{B}_{10}, \mathbf{H}^1(\mathbf{Q})) \to_\pi (\mathbb{B}_9, \mathbb{B}_{10}, \mathbf{H}^1(\mathbf{Q})) \\
&\to_\pi (\mathbb{B}_{11}, \mathbb{B}_{12}, \mathbf{CNOT}^{1,2}(\mathbf{H}^1(\mathbf{Q}))) \to^*_\pi (\mathbb{B}_7, \mathbb{B}_8, \mathbf{CNOT}^{1,2}(\mathbf{H}^1(\mathbf{Q}))) \\
&\to_\pi (\mathbb{B}_3, \mathbb{B}_4, \mathbf{CNOT}^{1,2}(\mathbf{H}^1(\mathbf{Q}))).
\end{aligned}$$

Notice that $CNOT$ acts as a synchronization operator: the second token is stuck in the occurrence $\mathbb{B}_{10}$ until the first token arrives as a control input of the $CNOT$ and the corresponding reduction step actually occurs.

6.5 Soundness

What is the relation between token machines and the equational theory on superposed type derivations introduced in Section 4.3?

It is easy to extend the definition of $[\cdot]$ to superposed type derivations: if $\mathcal{T} = \sum_{i=1}^n \alpha_i \pi_i$ then $[\mathcal{T}]$ when fed with a vector x returns $\sum_{i=1}^n \alpha_i [\pi_i](x)$. In the rest of this section, we will prove that token machines behave in accordance to the equational theory.

Suppose π is a type derivation for $\Gamma, x_1 : A_1, \ldots, x_m : A_m \vdash M : B$ and that, for every $1 \leq i \leq m$ there is a type derivation ρ_i for $\Delta_i \vdash N_i : A_i$. By induction on the structure of π, one can define a type derivation $\pi\{\rho_1, \ldots, \rho_m / x_1, \ldots, x_m\}$ of $\Gamma, \Delta_1, \ldots, \Delta_m \vdash M\{N_1, \ldots, N_m / x_1, \ldots, x_m\} : B$ (see Lemma 1). Moreover, from $\pi, \rho_1, \ldots, \rho_m$ we can form a machine $\mathcal{A}_\pi^{\rho_1, \ldots, \rho_m}$ as follows:

- The states of $\mathcal{A}_\pi^{\rho_1,\ldots,\rho_m}$ are in the form $(O_1,\ldots,O_n,\mathbf{Q})$ where:
  - $O_1,\ldots,O_n$ are occurrences of the type $\mathbb{B}$ in $\pi,\rho_1,\ldots,\rho_m$;
  - $\mathbf{Q}$ is a quantum register on n qubits;
- The transition function is itself obtained by taking the disjoint union of $\rightarrow_\pi,\rightarrow_{\rho_1},\ldots,\rightarrow_{\rho_n}$, plus
  - transitions of any positive occurrence of $\mathbb{B}$ in A_i (in the conclusion of ρ_i) to the corresponding occurrence of $\mathbb{B}$ in A_i (this time in the conclusion of π);
  - transitions of any negative occurrence of $\mathbb{B}$ in A_i (in the conclusione of π) to the corresponding occurrence of $\mathbb{B}$ in A_i (in the conclusion of ρ_i).
- Initial and final states are defined in the natural way, taking into account occurrences of $\mathbb{B}$ in $\Gamma,\Delta_1,\ldots,\Delta_m,B$, but not those in $A_1,\ldots A_m$.

The just defined machine is equivalent to the one built from the derivation $\pi\{\rho_1,\ldots,\rho_n/x_1,\ldots,x_m\}$. This is stated by the following substitution lemma:

Lemma 4 *Let $\pi \triangleright \Gamma, x_1 : A_1,\ldots,x_m : A_n \vdash M : B$ and for every $1 \leq i \leq m$ let $\rho_i \triangleright \Delta_i \vdash N_i : A_i$. Then the automaton $\mathcal{A}_{\pi\{x_1,\ldots,x_n/\rho_1,\ldots,\rho_n\}}$ is equivalent to $\mathcal{A}_\pi^{\rho_1,\ldots,\rho_n}$.*

It is now possible to prove two key intermediate results towards soundness:

Lemma 5 *Let $\pi \triangleright \Gamma \vdash (\lambda x.M)N : A$. Then $\langle\pi\rangle = \langle\pi^{\Downarrow}\rangle$.*

Lemma 6 *Let $\pi \triangleright \Gamma \vdash (\lambda\langle x,y\rangle.M)(N \otimes L) : A$. Then $\langle\pi\rangle = \langle\pi^{\Downarrow}\rangle$.*

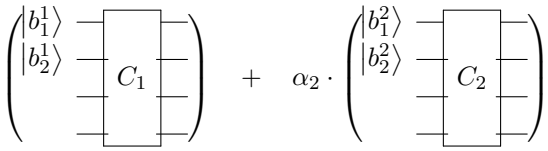
In order to prove Soundness Theorem, we need to introduce the following technical tool:

Definition 3 (Superposed Quantum Circuits) *A* superposed quantum circuits *of arity (n,m) (where $n \leq m$) is a formal sums in the form*

$$\sum_{i=1}^{n} \alpha_i C_i$$

where $\alpha_i \in \mathbb{C}$ and C_i is a quantum circuit on m qubits of which n are assigned a bit.

As an example, a superposed quantum circuit of arity $(2,4)$ looks as follows:



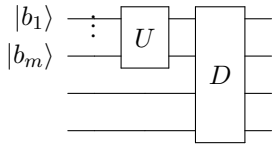
Since every type derivation π computes a quantum circuit $\langle\pi\rangle$, every superposed type derivation $\mathcal{T}$ can be seen as a superposed quantum circuit $\langle\mathcal{T}\rangle$. Moreover, the function $[\sum_{i=1}^n \alpha_i C_i]$ computed by a superposed quantum circuit $\sum_{i=1}^n \alpha_i C_i$ can be defined similarly to what we have done for superposed type derivations. Of course, $[\langle\mathcal{T}\rangle] = [\mathcal{T}]$.

We now define the set of *admissible circuit transformations*.

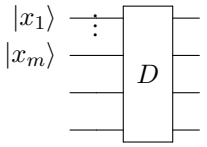
Definition 4 (Admissible Transformations) *Assume $\langle\mathcal{T}\rangle = \sum_{i=1}^n \alpha_i C_i$ is a superposed quantum circuit. The following transformation are called* admissible*:*

1. *One summand αC_i is replaced by $\beta C_i + \gamma C_i$, where $\alpha = \beta + \gamma$;*

2. *One summand αC_i where C_i has the following form*



is replaced by a sum $\sum_{x \in B_m} \alpha \cdot \beta_x \cdot C_x$ where B_m is the set of binary strings of length m, β_x is the coefficient of $|x\rangle$ in $\mathbf{U}|b_1 \ldots b_m\rangle$ and C_x is the following circuit:



Admissible transformations can be applied in both directions. It is easy to prove that admissible transformations, when applied to a superposed circuit $\langle \mathcal{T} \rangle$, leave the underlying function unchanged. We are now ready to prove our soundness result:

Theorem 1 (Soundness) *If $\mathcal{T} \approx \mathcal{S}$, then $[\mathcal{T}] = [\mathcal{S}]$.*

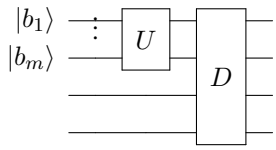
Proof. Since $[\langle \mathcal{T} \rangle] = [\mathcal{T}]$, it is sufficient, by Proposition 1, to show that, if $\mathcal{T} \sim \mathcal{S}$, then $\langle \mathcal{S} \rangle$ can be obtained from $\langle \mathcal{T} \rangle$ by iteratively applying one or more admissible transformations. This is an induction on the structure of a proof d of $\mathcal{T} \sim \mathcal{S}$. Let be r the last rule applied in d, where we enrich the thesis by stipluating that if the rules in d are all from $\mathsf{AX} \cup \mathsf{CC}$, then $\mathcal{T}$ is a single type derivation and that going from $\langle \mathcal{T} \rangle$ to $\langle \mathcal{S} \rangle$ can be done by performing *at most one* admissible transformation of the second kind. Some interesting cases:

- r is $(\mathsf{beta.pair})$. The result follows by means of Lemma 5.
- r is (beta). The result follows by means of Lemma 6.
- r is (quant). Then d is simply

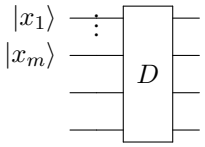
$$\frac{\pi \triangleright \cdot \vdash U|b_1 \ldots b_k\rangle : \mathbb{B}^k}{\pi \approx \mathbf{U}|b_1 \ldots b_k\rangle}\ \mathsf{quant}$$

  and $\langle \pi \rangle$ is simply the quantum circuit built on the unitary operator U, feeded with the input $|b_1 \ldots b_k\rangle$. We know that $\mathbf{U}|b_1 \ldots b_k\rangle$ is a superposed type derivation in the form $\mathcal{S} = \sum_{x \in B_k} \alpha_x \pi_x$, where B_k is the set of all binary strings of length k and π_x is the type derivation for $|x\rangle$ (k applications of the rule $(\mathsf{I}_\otimes)$ starting from the axioms for $|b_1\rangle \ldots |b_k\rangle$). Such a derivation can be seen as the superposed quantum circuit of ariety (k, k) $\langle \mathcal{S} \rangle = \sum_{x \in B_k} \alpha_x |x\rangle$ (where the binary string $|x\rangle$ can also seen as the trivial circuit that act on it as the identity) and the amplitudes α_x are exactly the coefficient of $|x\rangle$ in $U|b_1 \ldots b_k\rangle$. $\langle \mathcal{S} \rangle$ can be plainly obtained from $\langle \pi \rangle$ by means of the admissible transformation of the second kind by replacing the only summand $1 \cdot C$ with the sum $\sum_{x \in B_k} 1 \cdot \alpha_x |x\rangle$.
- r is a reflexive or a symmetric or a transitive closure. Trivial.
- $\mathsf{r} \in \mathsf{CC}$, then we know that $\mathcal{T} \sim \mathcal{S}$ is derived from $\mathcal{V} \sim \mathcal{W}$, where $\mathcal{V}$ is a *single* type derivation and $\langle \mathcal{W} \rangle$ is obtained by applying either zero or one admissible transormations of the second kind

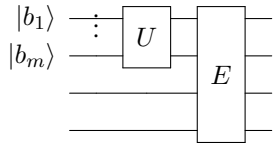
to $\langle \mathcal{V} \rangle$. In other words, $\mathcal{V}$ is



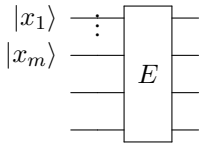
while $\mathcal{W}$ is $\sum_{x \in B_m} \alpha \cdot \beta_x \cdot C_x$ where B_m is the set of binary strings of length m, β_x is the coefficient of $|x\rangle$ in $U|b_1 \dots b_m\rangle$ and C_x is the following circuit:



It is then clear that the effect of r to $\langle \mathcal{V} \rangle$ consists in modifying D, because U cannot be affected. Moreover, the same modification is perfomed by r *uniformly* on D in any C_x. We can then conclude that there exists E such that $\mathcal{T}$ is



while $\mathcal{S}$ is



This concludes the proof. □

7 Related Works

The role of GoI in quantum computing has already been explored in at least two works. In [12] a geometry of interaction model for Selinger and Valiron's quantum lambda calculus [18] is defined. The model is formulated in particle-style. In [2] QMLL, an extension of MLL with quantum modalities is studied. QMLL is sound and complete with respect to quantum circuits, and an interactive, particle-style token machine is defined. The computational meaning of QMLL proofs is given by means of the token machine: each cut-free QMLL proof corresponds to an unique quantum circuit. In both cases, adopting a particle-style approach has a bad consequence: the "quantum" tensor product does *not* coincide with the tensor product in the sense of linear logic. Here we show that adopting the wave-style approach solves the problem.

Quantum extensions of game semantics are partially connected to our subject. In [8] a game semantics for a simply-typed lambda calculus (similar to $\mathsf{Q}\Lambda$) is introduced. The language uses a notion of extended variable, able to deal with tensor products. The game semantics is built around

classical game semantics where, however, quantum operations are the questions and measurements are the answers. A soundness result for the semantics is given. A similar approach for a lambda calculus with quantum stores (i.e. in which quantum data are referred through pointers) has been explored in [9]. Again, two tensor products are needed, unless one wants to drop the possibility of entangling qubits.

Purely linear quantum lambda-calculi (*with* measurements) can be given a fully abstract denotationl semantics, like the one proposed by Selinger and Valiron [19]. In their work, closure (necessary to interpret higher-order functions) is not obtained via traces and is not directly related in any way to the geometry of interaction. Moreover, morphisms are just linear maps, and so the model is far from being an quantum operational semantics like the $\mathsf{IAM}_{\mathsf{Q}\Lambda}$.

8 Conclusions

The definition of an elegant semantics is always a challenge in the case of quantum functional languages. This mainly holds for denotational models, but remains true also for operational, reduction-style semantics. In this paper we introduce $\mathsf{Q}\Lambda$, a linear quantum calculus with explicit qubits, where quantum circuits can be easily encoded. This simple calculus is a good framework to further investigate the (deep) relationships between quantum computing and Girard's Geometry of Interaction. We describe $\mathsf{IAM}_{\mathsf{Q}\Lambda}$, an interactive abstract machine which provides a sound operational characterization of any $\mathsf{Q}\Lambda$'s type derivation. $\mathsf{Q}\Lambda$ quantum features force to move from the (usual) particle-style token machine model to the wave-style one, where different tokens circulate around a net (a type derivation) at the same time. Constants for n-ary unitary operators act as *synchronization* points: every token trips independently since it arrives at a unitary operator constant. In this case, computation takes place only if all input qubits occurrence has reached the unitary operator. $\mathsf{IAM}_{\mathsf{Q}\Lambda}$ is a sound model: critical behaviors potentially introduced by the synchronization mechanism, can not happen in $\mathsf{IAM}_{\mathsf{Q}\Lambda}$ computations. Our contribution can be summarized as follows:

- The $\mathsf{IAM}_{\mathsf{Q}\Lambda}$ provides an elegant model for quantum programs written in $\mathsf{Q}\Lambda$: each type derivation is interpreted as a quantum circuit built on the set of quantum gates occurring in the underlying lambda-term;
- we show that also wave-style token machines are sound with respect to an operational theory of superposed type derivations;
- we give evidence that wave-style provides an original account of the quantum data entanglement phenomenon, since the notion of synchronization we implicitly define is strongly connected to what happens to entangled data.

Our investigation is open to some possible future directions. A natural step will be to extend the syntax of terms and type grammar with an exponential modality. The generalization of the wave-style token machine to this more expressive language would be an interesting and technically challenging subject. Something we see as relatively easy is an extension of this framework to a calculus with measurements: token machines could cope with measurements by evolving probabilistically[6], while adapting the equational theory would probably be nontrivial. Finally, giving a formal status to the connection between wave-style and the presence of entanglement is a fascinating subject which we definitely aim to investigate further.

References

[1] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM J. Comput.*, 26(5):1411–1473, 1997.

[2] U. Dal Lago and C. Faggian. On multiplicative linear logic, modality and quantum circuits. In *QPL*, volume 95 of *Electron. Proc. Theor. Comput. Sci.*, pages 55–66, 2011.

[3] U. Dal Lago, A. Masini, and M. Zorzi. On a measurement-free quantum lambda calculus with classical control. *Math. Structures Comput. Sci.*, 19(2):297–335, 2009.

[4] U. Dal Lago, A. Masini, and M. Zorzi. Confluence results for a quantum lambda calculus with measurements. *Electron. Notes Theor. Comput. Sci.*, 270(2):251–261, 2011.

[5] U. Dal Lago, A. Masini, and M. Zorzi. Quantum implicit computational complexity. *Theoret. Comput. Sci.*, 411(2):377–409, 2011.

[6] U. Dal Lago and M. Zorzi. Probabilistic operational semantics for the lambda calculus. *RAIRO Theor. Inform. Appl.*, 46(03):413–450, 2012.

[7] V. Danos and L. Regnier. Reversible, irreversible and optimal lambda-machines. *Theoret. Comput. Sci.*, 227:79–97, 1996.

[8] Y. Delbecque. Game semantics for quantum data. *Electron. Notes Theor. Comput. Sci.*, 270(1):41–57, 2011.

[9] Y. Delbecque and P. Panagaden. Game semantics for quantum stores. *Electron. Notes Theor. Comput. Sci.*, 218:153–170, 2008.

[10] J.-Y. Girard. Geometry of interaction I: Interpretation of system F. In *Proc. of the Logic Colloquium '88*, pages 221–260, 1989.

[11] G. Gonthier, M. Abadi, and J.-J. Lévy. The geometry of optimal lambda reduction. In *POPL*, pages 15–26, 1992.

[12] I. Hasuo and N. Hoshino. Semantics of higher-order quantum computation via geometry of interaction. In *LICS*, pages 237–246, 2011.

[13] A. Joyal, R. Street, and D. Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119(3):447–468, 1996.

[14] I. Mackie. The geometry of interaction machine. In *POPL*, pages 198–208, 1995.

[15] M. Nakahara and T. Ohmi. *Quantum Computing - From Linear Algebra to Physical Realizations.* CRC Press, 2008.

[16] M. Nielsen and I. Chuang. *Quantum computation and quantum information*. Cambridge University Press, 2000.

[17] H. Nishimura and M. Ozawa. Perfect computational equivalence between quantum turing machines and finitely generated uniform quantum circuit families. *Quantum Inf. Process.*, 8(1):13–24, 2009.

[18] P. Selinger and B. Valiron. A lambda calculus for quantum computation with classical control. *Math. Structures Comput. Sci.*, 16(3):527–552, 2006.

[19] P. Selinger and B. Valiron. On a fully abstract model for a quantum linear functional language. *Electron. Notes Theor. Comput. Sci.*, 210:123–137, 2008.

[20] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.

[21] A. van Tonder. A lambda calculus for quantum computation. *SIAM J. Comput.*, 33(5):1109–1135, 2004.

A Higher-Order Characterization of Probabilistic Polynomial Time

Ugo Dal Lago, Paolo Parisen Toldin

Dipartimento di Informatica, Università di Bologna
Équipe FOCUS, INRIA Sophia Antipolis
Mura Anteo Zamboni 7, 40127 Bologna, Italy

Abstract

We present RSLR, an implicit higher-order characterization of the class **PP** of those problems which can be decided in probabilistic polynomial time with error probability smaller than $1/2$. Analogously, a (less implicit) characterization of the class **BPP** can be obtained. RSLR is an extension of Hofmann's SLR with a probabilistic primitive, which enjoys basic properties such as subject reduction and confluence. Polynomial time soundness of RSLR is obtained by syntactical means, as opposed to the standard literature on SLR-derived systems, which use semantics in an essential way.

Keywords: Implicit computational complexity, Probabilistic classes, Lambda calculus, Linear types

1. Introduction

Implicit computational complexity (ICC) combines computational complexity, mathematical logic, and formal systems to give a machine independent account of complexity phenomena. It has been successfully applied to the characterization of a variety of complexity classes, especially in the sequential and parallel modes of computation (e.g., **FP** [4, 12], **PSPACE** [13], **LOGSPACE** [11], **NC** [5]). Its techniques, however, may be applied also to non-standard paradigms, like quantum computation [7] and concurrency [6]. Among the many characterizations of the class **FP** of functions computable in polynomial time, we can find Hofmann's *safe linear recursion* [10] (SLR in the following), a higher-order generalization of Bellantoni and Cook's *safe recursion* [3] in which linearity plays a crucial role.

Randomized computation is central to several areas of theoretical computer science, including cryptography, analysis of computation dealing with uncertainty and incomplete knowledge agent systems. In the context of computational complexity, probabilistic complexity classes like **BPP** [9] are nowadays considered as very closely corresponding to the

Email address: {dallago,parisent}@cs.unibo.it (Ugo Dal Lago, Paolo Parisen Toldin)
This work has been supported by project ANR 12IS02001 PACE.

informal notion of feasibility, since a solution to a problem in **BPP** can be computed in polynomial time up to any given degree of precision: **BPP** is the set of problems which can be solved by a probabilistic Turing machine working in polynomial time with a probability of error bounded by a constant *strictly* smaller than 1/2.

Probabilistic polynomial time computations, seen as oracle computations, were showed to be amenable to implicit techniques since the early days of ICC, by a relativization of Bellantoni and Cook's safe recursion [3]. They were then studied again in the context of formal systems for security, where probabilistic polynomial time computation plays a major role [14, 16]. These two systems are built on Hofmann's work SLR [10], by adding a random choice operator to the calculus. The system in [14], however, lacks higher-order recursion, and in both papers the characterization of probabilistic classes is obtained by semantic means. While this is fine for completeness, we think it is not completely satisfactory for soundness — we know from the semantics that for any term of a suitable type its normal form *may be* computed within the given bounds, but no notion of evaluation is given for which computation time is *guaranteed* to be bounded.

In this paper we propose RSLR, another probabilistic variation on SLR, and we show that it characterizes the class **PP** of those problems which can be solved in polynomial time by a Turing machine with error probability smaller than $\frac{1}{2}$ [9]. This is carried out by proving that any term in the language can be reduced in polynomial time, but also that any problems in **PP** can be represented in RSLR. A similar result, although in a less implicit form, is proved for **BPP**. Unlike [14], RSLR has higher-order recursion. Unlike [14] and [16], the bound on reduction time is obtained by syntactical means, giving an explicit notion of reduction which realizes that bound.

1.1. Related Work

More than ten years ago, Mitchell, Mitchell, and Scedrov [14] introduced OSLR, a type system that characterizes oracle polynomial time functionals. Even if inspired by SLR, OSLR does not admit primitive recursion on higher-order types, but only on base types. The main theorem shows that terms of type $\Box\mathbf{N}^m \to \mathbf{N}^n \to \mathbf{N}$ define precisely the *oracle polynomial time functionals*, which constitutes a class related but different from the ones we are interested in here. Finally, inclusion in the polynomial time class is proved without studying reduction from an operational viewpoint, but only via semantics: it is not clear for *which* notion of evaluation, computation time is guaranteed to be bounded.

Recently, Zhang's [16] introduced a further system (called CSLR) which builds on OSLR and allows higher-order recursion. The main interest of the paper are applications to the verification of security protocols. It is stated that CSLR defines exactly those functions that can be computed by probabilistic Turing machines in polynomial time, via a suitable variation of Hofmann's techniques as modified by Mitchell et al. This is again a purely semantic proof, whose details are missing in [16].

Finally, both works are derived from Hofmann's one, and as a consequence they both have potential problems with subject reduction. Indeed, as Hofmann showed in his work [10], subject reduction does not hold in SLR, and hence is problematic in both OSLR and CSLR.

1.2. RSLR*: An Informal Account*

Our system is called RSLR, which stands for Random Safe Linear Recursion. RSLR can be thought of as the system obtained by endowing SLR with a new primitive for random binary choice. Some restrictions have to be made to SLR if one wants to be able to prove polynomial time soundness operationally. And what one obtains at the end is indeed quite similar to (a probabilistic variation of) Bellantoni, Niggl and Schwichtenberg calculus RA [2, 15]. Actually, the main difference between RSLR and SLR has to do with linearity: keeping the size of reducts under control during normalization is very difficult in presence of higher-order duplication. For this reason, the two function spaces $A \to B$ and $A \multimap B$ of SLR collapse to just one in RSLR, and arguments of a higher-order type can *never* be duplicated. This constraint allows us to avoid an exponential blowup in the size of terms and results in a reasonably simple system for which polytime soundness can be proved explicitly, by studying the combinatorics of reduction. Another consequence of the just described modification is Subject Reduction, which can be easily proved in our system, contrarily to what happens in SLR [10].

1.3. On the Difficulty of Probabilistic ICC

Differently from most well-known complexity classes such as **P**, **NP** and **LOGSPACE**, interesting probabilistic complexity classes, like **BPP** and **ZPP** [9], are *semantic*. A semantic class is a complexity class defined on top of a class of algorithms which cannot be easily enumerated: a probabilistic polynomial time Turing machine does not *necessarily* solve a problem in **BPP** nor in **ZPP**. For most semantic classes, including **BPP** and **ZPP**, the existence of complete problems and the possibility to prove hierarchy theorems are both open question. Indeed, researchers in the area have proved the existence of such results for other probabilistic classes, but not for those we are interested in [8].

Now, having a "truly implicit" system I for a complexity class C means that we have a way to enumerate programs solving all problems in C (for every problem there is at least one program that solves it). The presence of complete problems, in other words, is deeply linked to the possibility of characterizing the class in the spirit of ICC. In our case the "semantic information" in **BPP** and **ZPP**, i.e., the error probability, seems to be an information that is impossible to capture by way of (recursively enumerable) syntactical restrictions. We need to execute the program on infinitely many inputs in order to check if the error probability is within bounds or not.

2. The Syntax and Basic Properties of RSLR

RSLR is a fairly standard Curry-style λ-calculus with constants for the natural numbers, branching and recursion. Its type system, on the other hand, is based on ideas coming from linear logic (variables of certain types can appear *at most once* in terms) and on a distinction between modal and non modal variables.

Let us introduce the category of types first:

Definition 2.1 (Types)**.** The *types* of RSLR are generated by the following grammar:

$$A ::= \mathbf{N} \mid \Box A \to A \mid \blacksquare A \to A.$$

Types different from $\mathbf{N}$ are denoted with metavariables like H or G. $\mathbf{N}$ is the only *base type*.

There are two function spaces in RSLR. Terms which can be typed with $\blacksquare A \to B$ are such that the result (of type B) can be computed in constant time, independently on the size of the argument (of type A). On the other hand, computing the result of functions in $\Box A \to B$ requires polynomial time in the size of their argument.

A notion of subtyping is used in RSLR to capture the intuition above by stipulating that the type $\blacksquare A \to B$ is a subtype of $\Box A \to B$. Subtyping is best formulated by introducing aspects:

Definition 2.2 (Aspects)**.** An *aspect* is either $\Box$ or $\blacksquare$: the first is the *modal* aspect, while the second is the *non-modal* one. Aspects are partially ordered by the binary relation $\{(\Box, \Box), (\Box, \blacksquare), (\blacksquare, \blacksquare)\}$, noted $<:$.

Defining subtyping, then, merely consists in generalizing $<:$ to a partial order on types in which only structurally identical types can be compared. Subtyping rules are in Figure 1. Please observe that (S-Sub) is contravariant in the aspect a.

$$\frac{}{A <: A}\ (\text{S-Refl}) \qquad \frac{A <: B \qquad B <: C}{A <: C}\ (\text{S-Trans})$$

$$\frac{B <: A \qquad C <: D \qquad b <: a}{aA \to C <: bB \to D}\ (\text{S-Sub})$$

Figure 1: Subtyping Rules.

RSLR's terms are those of an applied λ-calculus with primitive recursion and branching, in the style of Gödel's T:

Definition 2.3 (Terms)**.** *Terms* and *constants* are defined as follows:

$$t ::= x \mid c \mid ts \mid \lambda x : aA.t \mid \mathtt{case}_A\ t\ \mathtt{zero}\ s\ \mathtt{even}\ r\ \mathtt{odd}\ q \mid \mathtt{recursion}_A\ t\, s\, r;$$

$$c ::= n \mid \mathtt{S}_0 \mid \mathtt{S}_1 \mid \mathtt{P} \mid \mathtt{rand}.$$

Here, x ranges over a denumerable set of variables and n ranges over the natural numbers seen as constants of base type. Every constant c has its naturally defined type, that we indicate with $\mathit{type}(c)$. Formally, $\mathit{type}(n) = \mathbf{N}$ for every n, $\mathit{type}(\mathtt{rand}) = \mathbf{N}$, while

$\mathit{type}(\mathtt{S}_0) = \mathit{type}(\mathtt{S}_1) = \mathit{type}(\mathtt{P}) = \blacksquare\mathbf{N} \to \mathbf{N}$. The size $|t|$ of any term t can be easily defined by induction on t (where, by convention, we stipulate that $\log_2(0) = 0$):

$$\begin{aligned}
|x| &= 1;\\
|n| &= \lfloor \log_2(n) \rfloor + 1;\\
|\mathtt{S}_0| = |\mathtt{S}_1| &= |\mathtt{P}| = |\mathtt{rand}| = 1;\\
|ts| &= |t| + |s|;\\
|\lambda x : aA.t| &= |t| + 1;\\
|\mathtt{case}_A\ t\ \mathtt{zero}\ s\ \mathtt{even}\ r\ \mathtt{odd}\ q| &= |t| + |s| + |r| + |q| + 1;\\
|\mathtt{recursion}_A\ t\, s\, r| &= |t| + |s| + |r| + 1.
\end{aligned}$$

Notice that the size of n is exactly the length of the number n in binary representation. Size of 5, as an example, is $\lfloor \log_2(5) \rfloor + 1 = 3$, while 0 only requires one binary digit to be represented, and its size is thus 1. As usual, terms are considered modulo α-conversion. Free (occurrences of) variables and capture-avoiding substitution can be defined in a standard way.

Definition 2.4 (Explicit term)**.** A term is said to be *explicit* if it does not contain any instance of `recursion`.

The main peculiarity of RSLR with respect to similar calculi is the presence of an operator for probabilistic, binary choice, called `rand`, which evolves to either 0 or 1 with probability $\frac{1}{2}$. Although the calculus is in Curry-style, variables are explicitly assigned a type and an aspect in abstractions. This is for technical reasons that will become apparent soon.

Note 2.5. The presence of terms which can (probabilistically) evolve in different ways makes it harder to define a confluent notion of reduction for RSLR. To see why, consider a term like $t = (\lambda x : \blacksquare\mathbf{N}.(t_\oplus x x))\mathtt{rand}$, where $t_\oplus$ is a term computing $\oplus$ on natural numbers seen as booleans (0 stands for "false" and everything else stands for "true"):

$$\begin{aligned}
t_\oplus &= \lambda x : \blacksquare\mathbf{N}.\mathtt{case}_{\blacksquare\mathbf{N}\to\mathbf{N}}\ x\ \mathtt{zero}\ s_\oplus\ \mathtt{even}\ r_\oplus\ \mathtt{odd}\ r_\oplus;\\
s_\oplus &= \lambda y : \blacksquare\mathbf{N}.\mathtt{case}_{\mathbf{N}}\ y\ \mathtt{zero}\ 0\ \mathtt{even}\ 1\ \mathtt{odd}\ 1;\\
r_\oplus &= \lambda y : \blacksquare\mathbf{N}.\mathtt{case}_{\mathbf{N}}\ y\ \mathtt{zero}\ 1\ \mathtt{even}\ 0\ \mathtt{odd}\ 0.
\end{aligned}$$

If we evaluate t in a call-by-value fashion, `rand` will be fired *before* being passed to $t_\oplus$ and, as a consequence, the latter will be fed with two identical natural numbers, returning 0 with probability 1. If, on the other hand, `rand` is passed unevaluated to $t_\oplus$, the four possible combinations on the truth table for $\oplus$ will appear with equal probabilities and the outcome will be 0 or 1 with probability $\frac{1}{2}$. In other words, we need to somehow restrict our notion of reduction if we want it to be consistent, i.e. confluent. For the just explained reasons, arguments are passed to functions following a mixed scheme in RSLR: arguments of base type are evaluated before being passed to functions, while arguments of an higher-order type are passed to functions possibly unevaluated, in a call-by-name fashion. This way,

higher-order terms cannot be duplicated and this guarantees that if a term is duplicated, then it has no `rand` inside. The counterexample above, as a consequence, no longer works.

Let's first of all define the one-step reduction relation:

Definition 2.6 (Reduction)**.** The *one-step reduction relation* $\to$ is a binary relation between terms and sequences of terms. It is defined by the rules in Figure 2, which can be applied in any contexts except in the second and third argument of a recursion. Notice how the last but one rule is defined: in some cases, we allow to swap some arguments. Finally, we say that a term t is in *normal form* if t cannot appear as the left-hand side of a pair in $\to$. *NF* is the set of terms in normal form.

$$
\begin{aligned}
\mathtt{case}_A\ 0\ \mathtt{zero}\ t\ \mathtt{even}\ s\ \mathtt{odd}\ r &\to t;\\
\mathtt{case}_A\ (2\cdot(n+1))\ \mathtt{zero}\ t\ \mathtt{even}\ s\ \mathtt{odd}\ r &\to s;\\
\mathtt{case}_A\ (2\cdot n+1)\ \mathtt{zero}\ t\ \mathtt{even}\ s\ \mathtt{odd}\ r &\to r;\\
\mathtt{recursion}_A\ 0\ g\ f &\to g;\\
\mathtt{recursion}_A\ (n+1)\ g\ f &\to f(n+1)(\mathtt{recursion}_A\ \lfloor\frac{n+1}{2}\rfloor\ g\ f);\\
\mathtt{S}_0 n &\to 2\cdot n;\\
\mathtt{S}_1 n &\to 2\cdot n+1;\\
\mathtt{P} n &\to \lfloor\frac{n}{2}\rfloor;\\
(\lambda x : a\mathbf{N}.t)n &\to t[x/n];\\
(\lambda x : aH.t)s &\to t[x/s];\\
(\lambda x : aA.t)sr &\to (\lambda x : aA.tr)s;\\
\mathtt{rand} &\to 0, 1.
\end{aligned}
$$

Figure 2: One-step Reduction Rules.

Informally, $t \to s_1, \ldots, s_n$ means that t can evolve in one-step to each of $s_1, \ldots, s_n$ with the same probability $\frac{1}{n}$. As a matter of fact, n can be either 1 or 2.

A multistep reduction relation will not be defined by simply taking the transitive and reflective closure of $\to$, since a term can reduce in multiple steps to many terms with different probabilities. Multistep reduction puts in relation a term t to a probability distribution on terms $\mathscr{D}_t$ such that $\mathscr{D}_t(s) > 0$ only if s is a normal form to which t reduces. Of course, if t is itself a normal form, $\mathscr{D}_t$ is well defined, since the only normal form to which t reduces is t itself, so $\mathscr{D}_t(t) = 1$. But what happens when t is *not* in normal form? Is $\mathscr{D}_t$ a well-defined concept? Let us start by formally defining $\rightsquigarrow$:

Definition 2.7 (Multistep Reduction)**.** A *probability distribution* $\mathscr{D}$ should be understood here as a function from NF to $[0,1]$ such that $\sum_{t\in NF}\mathscr{D}(t)=1$. The binary relation $\rightsquigarrow$ between terms and probability distributions is defined by the rules in Figure 3. $\mathscr{I}_t$ is Dirac distribution on $t\in NF$, namely the function returning 1 on t and 0 on any other normal form. A finite distribution on terms $\mathscr{D}$ can be denoted as $\{t_1^{\alpha_1},\ldots,t_n^{\alpha_n}\}$ where $\mathscr{D}(s)=\sum_{t_i=s}\alpha_i$ (observe that the terms $t_1,\ldots,t_n$ are not necessarily distinct).

$$\frac{t\to t_1,\ldots,t_n \qquad t_i\rightsquigarrow\mathscr{D}_i}{t\rightsquigarrow\sum_{i=1}^{n}\frac{1}{n}\mathscr{D}_i} \qquad\qquad \frac{t\in NF}{t\rightsquigarrow\mathscr{I}_t}$$

Figure 3: Multistep Reduction: Inference Rules

In Section 2.2, we will prove that for every t there is at most one $\mathscr{D}$ such that $t\rightsquigarrow\mathscr{D}$. For the sake of clarifying how multistep reduction works, let us consider an example. Let $\mathtt{ifz}_A\ t\ \mathtt{then}\ s\ \mathtt{else}\ r$ be syntactic sugar for $\mathtt{case}_A\ t\ \mathtt{zero}\ s\ \mathtt{even}\ r\ \mathtt{odd}\ r$. Now, consider the term

$$t=\mathtt{ifz_N}\ (\mathtt{rand})\ \mathtt{then}\ (\mathtt{ifz_N}\ (\mathtt{rand})\ \mathtt{then}\ 1\ \mathtt{else}\ 2)\ \mathtt{else}\ 2.$$

The following one-step reduction can be derived from the last rule in Figure 2, applying it in a proper context:

$$\begin{aligned} t\to\ & \mathtt{ifz_N}\ 0\ \mathtt{then}\ (\mathtt{ifz_N}\ (\mathtt{rand})\ \mathtt{then}\ 1\ \mathtt{else}\ 2)\ \mathtt{else}\ 2,\\ & \mathtt{ifz_N}\ 1\ \mathtt{then}\ (\mathtt{ifz_N}\ (\mathtt{rand})\ \mathtt{then}\ 1\ \mathtt{else}\ 2)\ \mathtt{else}\ 2.\end{aligned}$$

Obviously, $\mathtt{ifz_N}\ 1\ \mathtt{then}\ (\mathtt{ifz_N}\ (\mathtt{rand})\ \mathtt{then}\ 1\ \mathtt{else}\ 2)\ \mathtt{else}\ 2\to 2$ (remember that the `ifz` construct is nothing more than syntactic sugar for a `case`). Let us examine the first of the two terms t reduces to. It one-step reduces to $s=\mathtt{ifz_N}\ (\mathtt{rand})\ \mathtt{then}\ 1\ \mathtt{else}\ 2$, and it is quite easy to realize that:

$$s\to(\mathtt{ifz_N}\ 0\ \mathtt{then}\ 1\ \mathtt{else}\ 2),(\mathtt{ifz_N}\ 1\ \mathtt{then}\ 1\ \mathtt{else}\ 2),$$

again by applying the last rule in Figure 2. Finally,

$$\begin{aligned}&\mathtt{ifz_N}\ 0\ \mathtt{then}\ 1\ \mathtt{else}\ 2\to 1;\\ &\mathtt{ifz_N}\ 1\ \mathtt{then}\ 1\ \mathtt{else}\ 2\to 2.\end{aligned}$$

Let us derive appropriate multi-step judgments. Of course, $1\rightsquigarrow\mathscr{I}_1$ and $2\rightsquigarrow\mathscr{I}_2$. As a consequence, $\mathtt{ifz_N}\ 0\ \mathtt{then}\ 1\ \mathtt{else}\ 2\rightsquigarrow\mathscr{I}_1$ and $\mathtt{ifz_N}\ 1\ \mathtt{then}\ 1\ \mathtt{else}\ 2\rightsquigarrow\mathscr{I}_2$. s one-step reduces to the two terms we have just considered, so $s\rightsquigarrow\frac{1}{2}\mathscr{I}_1+\frac{1}{2}\mathscr{I}_2$. Now, t one-step reduces to either s or 2 and, as a consequence,

$$t\rightsquigarrow\frac{1}{2}\left(\frac{1}{2}\mathscr{I}_1+\frac{1}{2}\mathscr{I}_2\right)+\frac{1}{2}\mathscr{I}_2=\frac{1}{4}\mathscr{I}_1+\frac{3}{4}\mathscr{I}_2.$$

We are finally able to present the type system. Preliminary to that is the definition of a proper notion of a context.

Definition 2.8 (Contexts)**.** A *context* Γ is a finite set of assignments of types and aspects to variables, i.e., of expressions in the form $x : aA$. As usual, we require contexts not to contain assignments of distinct types and aspects to the same variable. The union of two disjoint contexts Γ and Δ is denoted as Γ, Δ. In doing so, we implicitly assume that the variables in Γ and Δ are pairwise distinct. The expression $\Gamma; \Delta$ denotes the union Γ, Δ, but is only defined when all types appearing in Γ are base types. As an example, it is perfectly legitimate to write $x : a\mathbf{N}; y : b\mathbf{N}$, while the following is an ill-defined expression:

$$x : a(b\mathbf{N} \to \mathbf{N}); y : c\mathbf{N},$$

the problem being the first assignment, which appears on the left of ";" but which assigns the *higher-order* type $b\mathbf{N} \to \mathbf{N}$ (and the aspect a) to x. This notation is particularly helpful when giving typing rules. With the expression $\Gamma <: a$ we mean that any aspect b appearing in Γ is such that $b <: a$.

Typing rules are in Figure 4.

$$\frac{x : aA \in \Gamma}{\Gamma \vdash x : A}\ (\text{T-Var-Aff}) \qquad \frac{\Gamma \vdash t : A \qquad A <: B}{\Gamma \vdash t : B}\ (\text{T-Sub})$$

$$\frac{\Gamma, x : aA \vdash t : B}{\Gamma \vdash \lambda x : aA.t : aA \to B}\ (\text{T-Arr-I}) \qquad \frac{}{\Gamma \vdash c : \mathit{type}(c)}\ (\text{T-Const-Aff})$$

$$\frac{\begin{array}{cc}\Gamma; \Delta_1 \vdash t : \mathbf{N} & \Gamma; \Delta_3 \vdash r : A \\ \Gamma; \Delta_2 \vdash s : A & \Gamma; \Delta_4 \vdash q : A \qquad A \text{ is } \Box\text{-free}\end{array}}{\Gamma; \Delta_1, \Delta_2, \Delta_3, \Delta_4 \vdash \mathtt{case}_A\ t\ \mathtt{zero}\ s\ \mathtt{even}\ r\ \mathtt{odd}\ q : A}\ (\text{T-Case})$$

$$\frac{\begin{array}{ccc}\Gamma_1; \Delta_1 \vdash t : \mathbf{N} & & \\ \Gamma_1, \Gamma_2; \Delta_2 \vdash s : A & & \Gamma_1, \Delta_1 <: \Box \\ \Gamma_1, \Gamma_2; \vdash r : \Box\mathbf{N} \to \blacksquare A \to A & & A \text{ is } \Box\text{-free}\end{array}}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash \mathtt{recursion}_A\ t\, s\, r : A}\ (\text{T-Rec})$$

$$\frac{\Gamma; \Delta_1 \vdash t : aA \to B \qquad \Gamma; \Delta_2 \vdash s : A \qquad \Gamma, \Delta_2 <: a}{\Gamma; \Delta_1, \Delta_2 \vdash (ts) : B}\ (\text{T-Arr-E})$$

Figure 4: Typing Rules

Note 2.9. Observe how rules with more than one premise are designed in such a way as to guarantee that whenever $\Gamma \vdash t : A$ can be derived and $x : aH$ is in Γ, then x can appear

free *at most once* in t. If $y : a\mathbf{N}$ is in Γ, on the other hand, then y can appear free in t an arbitrary number of times. This can be proved by induction on the structure of any derivation for $\Gamma \vdash t : A$.

Definition 2.10. A *first-order term* of arity k is a closed, well-typed, term of type $a_1\mathbf{N} \to a_2\mathbf{N} \to \ldots a_k\mathbf{N} \to \mathbf{N}$ for some $a_1, \ldots, a_k$.

Example 2.1. Let's see some examples, namely two terms that we are able to type in our system, and one that is *not* possible to type. As we will see in Chapter 4.1 we are able to type addition and multiplication. Addition gives in output a number (recall that we are in unary notation) such that the resulting length is the sum of the input lengths.

$$\begin{aligned}\mathsf{add} =& \lambda x : \square\mathbf{N}.\lambda y : \blacksquare\mathbf{N}.\\ & \mathtt{recursion}_{\mathbf{N}}\ x\, y\, (\lambda x : \square\mathbf{N}.\lambda y : \blacksquare\mathbf{N}.\mathtt{S}_1 y) : \square\mathbf{N} \to \blacksquare\mathbf{N} \to \mathbf{N}\end{aligned}$$

We are also able to define multiplication. The operator is, as usual, defined by iterating addition:

$$\begin{aligned}\mathsf{mult} =& \lambda x : \square\mathbf{N}.\lambda y : \square\mathbf{N}.\\ & \mathtt{recursion}_{\mathbf{N}}\ (\mathtt{P}x)\, y\, (\lambda x : \square\mathbf{N}.\lambda z : \blacksquare\mathbf{N}.\mathsf{add} yz) : \square\mathbf{N} \to \square\mathbf{N} \to \mathbf{N}.\end{aligned}$$

Now that we have multiplication, why not iterate it and get an exponential? As it will be clear from the next example, the restriction on the aspect of the iterated function save us from having an exponential growth. Are we able to type the following term?

$$\lambda h : \square\mathbf{N}.\mathtt{recursion}_{\mathbf{N}}\ h\, (11)\, (\lambda x : \square\mathbf{N}.\lambda y : \blacksquare\mathbf{N}.\mathsf{mult}(y, y))$$

The answer is negative: the operator mult requires an input of aspect $\square$, while the iterated function necessarily has type $\square\mathbf{N} \to \blacksquare\mathbf{N} \to \mathbf{N}$.

2.1. Subject Reduction

The first property we are going to prove about RSLR is preservation of types under reduction, the so-called Subject Reduction Theorem. The proof of it is going to be very standard and, as usual, amounts to proving substitution lemmas. Preliminary to that is a technical lemma saying that weakening is derivable (since the type system is affine):

Lemma 2.1 (Weakening Lemma)**.** *If $\Gamma \vdash t : A$, then $\Gamma, x : bB \vdash t : A$ whenever x does not appear in Γ.*

Proof. By induction on the structure of the typing derivation for t.

- If last rule is (T-Var-Aff) or (T-Const-Aff), we are allowed to add whatever we want to the context.
- If last rule is (T-Sub) or (T-Arr-I), the thesis is proved by applying the induction hypothesis to the premise.

- Suppose that the last rule was:

$$\frac{\begin{array}{ccc}\Gamma;\Delta_1 \vdash u : N & \Gamma;\Delta_3 \vdash r : A & \\ \Gamma;\Delta_2 \vdash s : A & \Gamma;\Delta_4 \vdash q : A & A \text{ is } \Box\text{-free}\end{array}}{\Gamma;\Delta_1,\Delta_2,\Delta_3,\Delta_4 \vdash \mathtt{case}_A\ u\ \mathtt{zero}\ s\ \mathtt{even}\ r\ \mathtt{odd}\ q : A}\ (\text{T-Case})$$

  If $B = \mathbf{N}$ we can easily proceed by applying the induction hypothesis to every premises and add x to Γ. Otherwise, we can proceed by applying induction hypothesis to just one premise.
- Suppose that the last rule is:

$$\frac{\begin{array}{cc}\Gamma_1;\Delta_1 \vdash q : \mathbf{N} & \\ \Gamma_1,\Gamma_2;\Delta_2 \vdash s : A & \Gamma_1,\Delta_1 <: \Box \\ \Gamma_1,\Gamma_2;\vdash r : \Box\mathbf{N} \to \blacksquare A \to A & A \text{ is } \Box\text{-free}\end{array}}{\Gamma_1,\Gamma_2;\Delta_1,\Delta_2 \vdash \mathtt{recursion}_A\ q\ s\ r : A}\ (\text{T-Rec})$$

  Suppose that $B = \mathbf{N}$. We have the following cases:
  - If $b = \Box$, we can do it by applying induction hypothesis to all the premises and add x in Γ_1.
  - If $b = \blacksquare$ we apply induction hypothesis on $\Gamma_1,\Gamma_2;\Delta_2 \vdash s : A$ and on $\Gamma_1,\Gamma_2;\vdash r : \Box\mathbf{N} \to \blacksquare A \to A$.

  Otherwise we apply induction hypothesis on $\Gamma_1;\Delta_1 \vdash q : \mathbf{N}$ or on $\Gamma_1,\Gamma_2;\Delta_2 \vdash s : A$ and we are done.

This concludes the proof. □

Two substitution lemmas are needed in RSLR. The first one applies when the variable to be substituted has a non-modal type:

Lemma 2.2 ($\blacksquare$-Substitution Lemma)**.** *Let* $\Gamma;\Delta \vdash t : A$*. Then*

1. *if* $\Gamma = x : \blacksquare\mathbf{N}, \Theta$*, then* $\Theta;\Delta \vdash t[x/n] : A$ *for every* n*;*
2. *if* $\Delta = x : \blacksquare H, \Theta$ *and* $\Gamma;\Xi \vdash s : H$*, then* $\Gamma;\Theta,\Xi \vdash t[x/s] : A$*.*

Proof. By induction on a type derivation of t. Some interesting cases:

- If the last rule is (T-Rec), our derivation will have the following shape:

$$\frac{\begin{array}{cc}\Gamma_2;\Delta_4 \vdash q : \mathbf{N} & \\ \Gamma_2,\Gamma_3;\Delta_5 \vdash s : B & \Gamma_2,\Delta_4 <: \Box \\ \Gamma_2,\Gamma_3;\vdash r : \Box\mathbf{N} \to \blacksquare B \to B & B \text{ is } \Box\text{-free}\end{array}}{\Gamma_2,\Gamma_3;\Delta_4,\Delta_5 \vdash \mathtt{recursion}_B\ q\ s\ r : B}\ (\text{T-Rec})$$

  By definition, $x : \blacksquare A$ cannot appear in $\Gamma_2;\Delta_4$. If it appears in Δ_5 we can simply apply induction hypothesis and prove the thesis. We will focus on the most interesting case: it appears in Γ_3 and so $A = \mathbf{N}$. In that case, by the induction hypothesis applied to (type derivations for) s and r, we obtain that:

$$\begin{array}{c}\Gamma_2,\Gamma_4;\Delta_5 \vdash s[x/n] : B; \\ \Gamma_2,\Gamma_4;\vdash r[x/n] : \Box\mathbf{N} \to \blacksquare B \to B;\end{array}$$

  where $\Gamma_3 = \Gamma_4, x : \blacksquare\mathbf{N}$.

- If the last rule is (T-Arr-E),

$$\frac{\Gamma;\Delta_4 \vdash t : aC \to B \qquad \Gamma;\Delta_5 \vdash s : C \qquad \Gamma,\Delta_5 <: a}{\Gamma,\Delta_4,\Delta_5 \vdash (ts) : B}\ (\text{T-Arr-E})$$

  If $x : A$ is in Γ then we apply induction hypothesis on both branches, otherwise it is either in Δ_4 or in Δ_5 and we apply induction hypothesis on the corresponding branch. We arrive to the thesis by applying (T-Arr-E) at the end.

This concludes the proof. □

Substituting a variable of a *modal* type requires an additional hypothesis on the term being substituted:

Lemma 2.3 (□-Substitution Lemma)**.** *Let* $\Gamma;\Delta \vdash t : A$*. Then*

1. *if* $\Gamma = x : \Box\mathbf{N}, \Theta$*, then* $\Theta;\Delta \vdash t[x/n] : A$ *for every* n*;*
2. *if* $\Delta = x : \Box H, \Theta$ *and* $\Gamma;\Xi \vdash s : H$ *where* $\Gamma,\Xi <: \Box$*, then* $\Gamma;\Theta,\Xi \vdash t[x/s] : A$*.*

Proof. By induction on the structure of a type derivation for t. Some cases:

- If last rule is (T-Rec), our derivation will have the following shape:

$$\frac{\begin{array}{cc}\Gamma_2;\Delta_4 \vdash q : \mathbf{N} & \\ \Gamma_2,\Gamma_3;\Delta_5 \vdash s : B & \Gamma_2,\Delta_4 <: \Box \\ \Gamma_2,\Gamma_3;\vdash r : \Box\mathbf{N} \to \blacksquare B \to B & B \text{ is } \Box\text{-free}\end{array}}{\Gamma_2,\Gamma_3;\Delta_4,\Delta_5 \vdash \mathtt{recursion}_B\ q\, s\, r : B}\ (\text{T-Rec})$$

  By definition $x : \Box A$ can appear in $\Gamma_1;\Delta_4$. If so, by applying induction hypothesis we can derive easily the proof. In the other cases, we can proceed as in Lemma 2.2. We will focus on the most interesting case, where $x : \Box A$ appears in Γ_2 and so $A = \mathbf{N}$. In that case, by the induction hypothesis applied to (type derivations for) s and r, we obtain that:

$$\begin{array}{c}\Gamma_4,\Gamma_3;\Delta_5 \vdash s[x/n] : B \\ \Gamma_4,\Gamma_3;\vdash r[x/n] : \Box\mathbf{N} \to \blacksquare B \to B\end{array}$$

  where $\Gamma_2 = \Gamma_4, x : \Box\mathbf{N}$.

- If last rule is (T-Arr-E),

$$\frac{\Gamma;\Delta_4 \vdash t : aC \to B \qquad \Gamma;\Delta_5 \vdash s : C \qquad \Gamma,\Delta_5 <: a}{\Gamma,\Delta_4,\Delta_5 \vdash (ts) : B}\ (\text{T-Arr-E})$$

  If $x : A$ is in Γ then we apply induction hypothesis on both branches, otherwise it is either in Δ_4 or in Δ_5 and we apply induction hypothesis on the relative branch. We prove our thesis by applying (T-Arr-E) at the end.

This concludes the proof. □

Substitution lemmas are necessary ingredients when proving Subject Reduction. In particular, they allow to prove that types are preserved along β-reduction steps, the other reduction steps being very easy. We get:

Theorem 2.4 (Subject Reduction)**.** *Suppose that* $\Gamma \vdash t : A$*. If* $t \to t_1 \dots t_j$*, then for every* $i \in \{1, \dots, j\}$*, it holds that* $\Gamma \vdash t_i : A$*.*

Proof. By induction on the structure of a derivation for t. Some interesting cases:

- If last rule is (T-Case).

$$\frac{\begin{array}{cc} \Gamma;\Delta_1 \vdash s : N & \Gamma;\Delta_3 \vdash q : A \\ \Gamma;\Delta_2 \vdash r : A & \Gamma;\Delta_4 \vdash u : A \qquad A \text{ is } \Box\text{-free} \end{array}}{\Gamma;\Delta_1,\Delta_2,\Delta_3,\Delta_4 \vdash \mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u : A}\ (\text{T-Case})$$

  Our final term could reduce in two ways. Either we do β-reduction on s, r, q or u, or we choose one of branches in the case. In all the cases, the proof is trivial.

- If last rule is (T-Rec).

$$\frac{\begin{array}{cc} \rho : \Gamma_1;\Delta_1 \vdash s : \mathbf{N} & \\ \mu : \Gamma_1,\Gamma_2;\Delta_2 \vdash r : A & \Gamma_1,\Delta_1 <: \Box \\ \nu : \Gamma_1,\Gamma_2;\vdash q : \Box\mathbf{N} \to \blacksquare A \to A & A \text{ is } \Box\text{-free} \end{array}}{\Gamma_1,\Gamma_2;\Delta_1,\Delta_2 \vdash \mathtt{recursion}_A\ s\, r\, q : A}\ (\text{T-Rec})$$

  Our term could reduce in three ways. We could evaluate s (trivial), we could be in the case where $s = 0$ (trivial) and the other case is where we unroll the recursion (so, where s is a value $n \geq 1$). We are going to focus on this last option. The term rewrites to $qn(\mathtt{recursion}_\tau\ \lfloor \frac{n}{2} \rfloor\, r\, q)$. We could define the following derivations π and σ:

$$\dfrac{\dfrac{}{\Gamma_1;\Delta_1 \vdash \lfloor \frac{n}{2} \rfloor : \mathbf{N}}\ (\text{T-Const-Aff}) \qquad \mu : \Gamma_1,\Gamma_2;\Delta_2 \vdash r : A}{\nu : \Gamma_1,\Gamma_2;\vdash q : \Box\mathbf{N} \to \blacksquare A \to A \qquad \Gamma_1,\Gamma_2;\Delta_1,\Delta_2 \vdash \mathtt{recursion}_\tau\ \lfloor \frac{n}{2} \rfloor\, r\, q : A}\ (\text{T-Rec})$$

$$\dfrac{\nu : \emptyset;\Gamma_1,\Gamma_2 \vdash q : \Box\mathbf{N} \to \blacksquare A \to A \qquad \dfrac{}{\emptyset;\emptyset \vdash n : \mathbf{N}}\ (\text{T-Const-Aff})}{\emptyset;\Gamma_1,\Gamma_2 \vdash qn : \blacksquare A \to A}\ (\text{T-ARR-E})$$

  By gluing the two derivation with the rule (T-Arr-E) we obtain:

$$\frac{\begin{array}{c} \sigma : \Gamma_1,\Gamma_2;\vdash qn : \blacksquare A \to A \\ \pi : \Gamma_1,\Gamma_2;\Delta_1,\Delta_2 \vdash \mathtt{recursion}_\tau\ \lfloor \frac{n}{2} \rfloor\, r\, q : A \end{array}}{\Gamma_1,\Gamma_2,\Gamma_3;\Delta_1,\Delta_2 \vdash qn(\mathtt{recursion}_\tau\ \lfloor \frac{n}{2} \rfloor\, r\, q) : A}\ (\text{T-Arr-E})$$

  Notice that in the derivation ν we put Γ_1, Γ_2 on the left side of ";" and also on the right side. Recall Definition 2.8.

- If last rule was (T-Sub) we have the following derivation:

$$\frac{\Gamma \vdash s : A \qquad A <: B}{\Gamma \vdash s : B}\ (\text{T-Sub})$$

  If s reduces to r we can apply induction hypothesis on the premises and having the following derivation:

$$\frac{\Gamma \vdash r : A \qquad A <: B}{\Gamma \vdash r : B}\ (\text{T-Sub})$$

- If last rule was (T-Arr-E), we could have different cases.
  - Cases where on the left part of our application we have $\mathtt{S}_i$, $\mathtt{P}$ is trivial.
  - Let's focus on the case where on the left part we find a λ-abstraction. We will only consider the case where we apply the substitution. The other cases are trivial. We could have two possibilities:
    - First of all, we can be in the following situation:

$$\frac{\Gamma;\Delta_1 \vdash \lambda x : \blacksquare A.r : aC \to B \qquad \Gamma;\Delta_2 \vdash s : C \qquad \Gamma,\Delta_2 <: a}{\Gamma,\Delta_1,\Delta_2 \vdash (\lambda x : \blacksquare A.r)s : B}\ \text{(T-Arr-E)}$$

      where $C <: A$ and $a <: \blacksquare$. We have that $(\lambda x : \blacksquare A.r)s$ rewrites to $r[x/s]$. By looking at rules in Figure 4 we can deduce that $\Gamma;\Delta_1 \vdash \lambda x : \blacksquare A.r : aC \to B$ derives from $\Gamma; x : \blacksquare A, \Delta_1 \vdash r : D$ (with $D <: B$). For the reason that $C <: A$ we can apply (T-Sub) rule to $\Gamma;\Delta_2 \vdash s : C$ and obtain $\Gamma;\Delta_2 \vdash s : A$ By applying Lemma 2.2, we get to

$$\Gamma,\Delta_1,\Delta_2 \vdash r[x/s] : D$$

      from which the thesis follows by applying (T-Sub).
    - But we can even be in the following situation:

$$\frac{\Gamma;\Delta_1 \vdash \lambda x : \square A.r : \square C \to B \qquad \Gamma;\Delta_2 \vdash s : C \qquad \Gamma,\Delta_2 <: \square}{\Gamma,\Delta_1,\Delta_2 \vdash (\lambda x : \square A.r)s : B}\ \text{(T-Arr-E)}$$

      where $C <: A$. We have that $(\lambda x : \square A.r)s$ rewrites in $r[x/s]$. We behave as in the previous point, by applying Lemma 2.3, and we are done.
  - Another interesting case of application is where we perform a so-called "swap". $(\lambda x : aA.q)sr$ rewrites in $(\lambda x : aA.qr)s$. From a typing derivation with conclusion $\Gamma,\Delta_1,\Delta_2,\Delta_3 \vdash (\lambda x : aA.q)sr : C$ we can easily extract derivations for the following:

$$\Gamma;\Delta_1, x : aA \vdash q : bD \to E$$
$$\Gamma;\Delta_3 \vdash r : B$$
$$\Gamma;\Delta_2 \vdash s : F$$

    where $B <: D$, $E <: C$ and $A <: F$ and $\Gamma,\Delta_3 <: b$ and $\Gamma,\Delta_2 <: a$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma,\Delta_3 <: b \quad \Gamma;\Delta_3 \vdash r : B \quad \dfrac{\Gamma;\Delta_1, x : aA \vdash q : bD \to E}{\Gamma;\Delta_1,\Delta_3, x : aA \vdash qr : E}\ \text{(T-Arr-E)}}{\Gamma;\Delta_1,\Delta_3,\vdash \lambda x : aA.qr : aA \to E}\ \text{(T-Arr-I)}}{\Gamma;\Delta_1,\Delta_3,\vdash \lambda x : aA.qr : aF \to C}\ \text{(T-Sub)} \qquad \Gamma,\Delta_2 <: a \quad \Gamma;\Delta_2 \vdash s : F}{\Gamma,\Delta_1,\Delta_2,\Delta_3 \vdash (\lambda x : aA.qr)s : C}\ \text{(T-Arr-E)}$$

- All the other cases can be brought back to cases that we have considered.

This concludes the proof. □

Example 2.2. The following example has been proposed by Hofmann [10]. Let f be a variable of type $\blacksquare\mathbf{N} \to \mathbf{N}$. The function $h = \lambda g : \blacksquare(\blacksquare\mathbf{N} \to \mathbf{N}).\lambda x : \blacksquare\mathbf{N}.(f(gx))$ gets type $\blacksquare(\blacksquare\mathbf{N} \to \mathbf{N}) \to \blacksquare\mathbf{N} \to \mathbf{N}$. Thus the function $(\lambda v : \blacksquare(\blacksquare\mathbf{N} \to \mathbf{N}).hv)\mathtt{S}_1$ takes type $\blacksquare\mathbf{N} \to \mathbf{N}$. Let's now fire a β step, by passing the argument $\mathtt{S}_1$ to the function h and we obtain the following term: $\lambda x : \blacksquare\mathbf{N}.(f(\mathtt{S}_1 x))$ It's easy to check that the type has not changed.

2.2. Confluence

In view of the peculiar notion of reduction given in Definition 2.6, let us go back to the counterexample to confluence given in the Introduction. The term $t = (\lambda x : \blacksquare\mathbf{N}.(t_\oplus xx))\mathtt{rand}$ cannot be reduced to $t_\oplus\,\mathtt{rand}\,\mathtt{rand}$ anymore, because only numerals can be passed to functions as arguments of base types. The only possibility is reducing t to the sequence

$$(\lambda x : \blacksquare\mathbf{N}.(t_\oplus xx))0, (\lambda x : \blacksquare\mathbf{N}.(t_\oplus xx))1.$$

Both terms in the sequence can be further reduced to 0. In other words, $t \rightsquigarrow \{0^1\}$.

More generally, the phenomenon of non-convergence of final distributions can no longer happen in RSLR. Technically, this is due to the impossibility of duplicating "probabilistic" terms, i.e., terms containing occurrences of `rand`. In the above example, and in similar cases, we have to evaluate the argument before firing the β-redex — it is therefore not possible to obtain two different distributions. RSLR can also handle correctly the case where `rand` is within an argument t of higher-order type: the only non-normal terms which can be duplicated are the arguments to a recursion, in which reduction cannot take place *by definition*.

Confluence of our system is proved by first showing a strong form of confluence for the single step arrow $\to$, then transferring it to the multistep arrow $\rightsquigarrow$. Showing confluence for $\to$ turns out to be slightly more complicated than expected and is thus split into three separate lemmas.

Lemma 2.5 (Diamond Property, Part I)**.** *Let t be a well-typed term; if $t \to v$ and $t \to z$ (where v and z distinct), then exactly one of the following holds:*

- *$\exists e$ such that $v \to e$ and $z \to e$;*
- *$v \to z$;*
- *$z \to v$.*

Proof. By induction on the structure of the typing derivation for the term t. Some interesting cases:

- If last rule is T-Case, our derivation will have the following shape:

$$\frac{\begin{array}{lll}\Gamma;\Delta_1 \vdash s : N & \Gamma;\Delta_3 \vdash q : A & \\ \Gamma;\Delta_2 \vdash r : A & \Gamma;\Delta_4 \vdash u : A & A \text{ is } \Box\text{-free}\end{array}}{\Gamma;\Delta_1,\Delta_2,\Delta_3,\Delta_4 \vdash \mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u : A}\ (\text{T-Case})$$

  We could have reduced one among s, r, q, u or a combination of them. In the first case we prove by applying induction hypothesis and in the latter case we can easily find e

such that $v \to e$ and $z \to e$: it is the term obtained by applying both reductions. Last case is where from one part we reduce the case, selecting a branch and from the other part we reduce one of the subterms. As can be easily seen, we can find a common confluent term.

- If last rule is T-Rec, our derivation will have the following shape:

$$\frac{\begin{array}{cc}\Gamma_2;\Delta_4 \vdash q : \mathbf{N} & \\ \Gamma_2,\Gamma_3;\Delta_5 \vdash s : B & \Gamma_2;\Delta_4 <: \Box \\ \Gamma_2,\Gamma_3;\vdash r : \Box\mathbf{N} \to \blacksquare B \to B & B \text{ is } \Box\text{-free}\end{array}}{\Gamma_2,\Gamma_3;\Delta_4,\Delta_5 \vdash \mathtt{recursion}_B\ q\ s\ r : B}\ (\text{T-Rec})$$

  By definition, we can have reduction only in q or, if q is a value, we can reduce the recursion by unrolling it. In both cases the proof is trivial.
- If last rule is T-Arr-E, our term could have different shapes but the only interesting cases are the following ones. The other cases can be easily brought back to cases that we have already considered.
  - Our derivation will end in the following way:

$$\frac{\Gamma;\Delta_1 \vdash \lambda x : aA.r : bC \to B \qquad \Gamma;\Delta_2 \vdash s : C \qquad \Gamma,\Delta_2 <: b}{\Gamma,\Delta_1,\Delta_2 \vdash (\lambda x : aA.r)s : B}\ (\text{T-Arr-E})$$

    where $C <: A$ and $b <: a$. We have that $(\lambda x : aA.r)s$ rewrites to $r[x/s]$; if $A = \mathbf{N}$ then s is a value. If we reduce only in s or only in r we can easily prove our thesis by the induction hypothesis. The interesting cases are when we perform the substitution on one hand and on the other hand we make a reduction step on one of the two possible terms s or r. Suppose $(\lambda x : aA.r)s \to r[x/s]$ and $(\lambda x : aA.r)s \to (\lambda x : aA.r)f$, where $s \to f$. Let e be $r[x/f]$. We have that $(\lambda x : aA.r)f \to e$ and $r[x/s] \to e$. Indeed if A is $\mathbf{N}$, s is a value, no reduction could be made on s. Otherwise, there is at most one occurrence of s in $r[x/s]$ and by executing one reduction step we are able to reach e. Suppose $(\lambda x : aA.r)s \to r[x/s]$ and $(\lambda x : aA.r)s \to (\lambda x : aA.g)s$, where $r \to g$. As we have shown in the previous case, we are able to find the required terms.
  - The other interesting case is when we perform the so called "swap". $(\lambda x : aA.q)sr$ rewrites in $(\lambda x : aA.qr)s$. If the reduction steps are made only in q or s or r, by applying induction hypothesis we have the thesis. In all the other cases, where we perform one step on subterms and we perform, on the other hand, the swap, it's easy to find a confluent term e.

This concludes the proof. □

Lemma 2.6 (Diamond Property, Part II)**.** *Let t be a well typed term in* RSLR*; if $t \to v_1, v_2$ and $t \to z$ then one of the following sentence is valid:*

- $\exists e_1, e_2$ *s.t.* $v_1 \to e_1$ *and* $v_2 \to e_2$ *and* $z \to e_1, e_2$
- $\forall i. v_i \to z$
- $z \to v_1, v_2$

Proof. By induction on the structure of a typing derivation for the term t.

- t cannot be a constant or a variable. Indeed if t is `rand`, t reduces in $0, 1$ and this contradict our hypothesis.
- If last rule is T-Sub or T-Arr-I, the thesis is easily proved by applying induction hypothesis.
- If last rule is T-Case, our derivation will have the following shape:

$$\frac{\begin{array}{ccc}\Gamma;\Delta_1 \vdash s : N & \Gamma;\Delta_3 \vdash q : A & \\ \Gamma;\Delta_2 \vdash r : A & \Gamma;\Delta_4 \vdash u : A & A \text{ is } \Box\text{-free}\end{array}}{\Gamma;\Delta_1,\Delta_2,\Delta_3,\Delta_4 \vdash \mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u : A}\ (\text{T-Case})$$

  If we perform the two reductions on the same subterm we could be in the following case (all the other cases are similar). For example, if t reduces to $\mathtt{case}_A\ s_1\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u$ and $\mathtt{case}_A\ s_2\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u$ and also to $t \to \mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ f$, it is easy to check that the two required terms are $e_1 = \mathtt{case}_A\ s_1\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ f$ and $e_2 = \mathtt{case}_A\ s_2\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ f$. Another possible case is where on one hand we perform a reduction by selecting a branch and on the other case we make a reduction on one branch. As example, $t \to q$ and $r \to r_1, r_2$. This case is trivial.
- If last rule was T-Rec, our derivation will have the following shape:

$$\frac{\begin{array}{cc}\Gamma_2;\Delta_4 \vdash q : \mathbf{N} & \\ \Gamma_2,\Gamma_3;\Delta_5 \vdash s : B & \Gamma_2;\Delta_4 <: \Box \\ \Gamma_2,\Gamma_3;\vdash r : \Box\mathbf{N} \to \blacksquare B \to B & B \text{ is } \Box\text{-free}\end{array}}{\Gamma_2,\Gamma_3;\Delta_4,\Delta_5 \vdash \mathtt{recursion}_B\ q\ s\ r : B}\ (\text{T-Rec})$$

  By definition, we can have reduction only in q. By applying induction hypothesis the thesis is proved.
- If last rule was T-Arr-E. Our term could have different shapes but the only interesting cases are the following ones. The other cases can be easily brought back to cases that we have considered.
  - Our derivation will end in the following way:

$$\frac{\Gamma;\Delta_1 \vdash \lambda x : aA.r : bC \to B \qquad \Gamma;\Delta_2 \vdash s : C \qquad \Gamma,\Delta_2 <: b}{\Gamma,\Delta_1,\Delta_2 \vdash (\lambda x : aA.r)s : B}\ (\text{T-Arr-E})$$

    where $C <: A$ and $b <: a$. We have that $(\lambda x : aA.r)s$ rewrites in $r[x/s]$; if $A = \mathbf{N}$ then s is a value, otherwise we are able to make the substitution whenever we want. If we reduce only in s or only in r we can easily prove our thesis by applying induction hypothesis. The interesting cases are when we perform the substitution on one hand and on the other hand we make a reduction step on one of the two possible terms s or r. Suppose $(\lambda x : aA.r)s \to r[x/s]$ and $(\lambda x : aA.r)s \to (\lambda x : aA.r)s_1, (\lambda x : aA.r)s_2$, where $s \to s_1, s_2$. Let e_1 be $r[x/s_1]$ and e_2 be $r[x/s_2]$. We have that $(\lambda x : aA.r)s_1 \to e_1$, $(\lambda x : aA.r)s_2 \to e_2$ and $r[x/s] \to e_1, e_2$. Indeed if A is $\mathbf{N}$ then s is a value (because we are making substitutions) and we cannot have the reductions on s, otherwise there is at least one occurrence of s in $r[x/s]$ and by performing one reduction step on the subterm s we are able to have e_1, e_2. Suppose $(\lambda x : aA.r)s \to r[x/s]$ and $(\lambda x : aA.r)s \to (\lambda x : aA.r_1)s, (\lambda x : aA.r_2)s$, where $r \to r_1, r_2$. This, again, can be easily managed.

- The other interesting case is when we perform the so called "swap". $(\lambda x : aA.q)sr$ rewrites to $(\lambda x : aA.qr)s$. If the reduction steps are made only on q or s or r, applying induction hypothesis suffices. In all the other cases, where we perform one step on subterms and we perform, on the other hand, the swap, it's easy to find the required term e.

This concludes the proof. □

Lemma 2.7 (Diamond Property, Part III)**.** *Let t be a well typed term in* RSLR*; if $t \to v_1, v_2$ and $t \to z_1, z_2$ (v_1, v_2 and z_1, z_2 different) then $\exists e_1, e_2, e_3, e_4$ s.t. $v_1 \to e_1, e_2$ and $v_2 \to e_3, e_4$ and $z_1 \to e_1, e_3$ and $z_2 \to e_2, e_4$.*

Proof. By induction on the structure of a typing derivation for t. Some interesting cases:

- If last rule was (T-Case) our derivation has the following shape:

$$\frac{\begin{array}{ccc}\Gamma;\Delta_1 \vdash s : N & \Gamma;\Delta_3 \vdash q : A & \\ \Gamma;\Delta_2 \vdash r : A & \Gamma;\Delta_4 \vdash u : A & A \text{ is } \Box\text{-free}\end{array}}{\Gamma;\Delta_1,\Delta_2,\Delta_3,\Delta_4 \vdash \mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u : A}\ (\text{T-Case})$$

  Also this case is easy to prove. Indeed if the reduction steps are made only on single subterms: s or r or q or u we can prove by using induction hypothesis. Otherwise we are in the case where one reduction step is made on some subterm and the other is made considering a different subterm. Suppose $s \to s_1, s_2$ and $q \to q_1, q_2$. We could have two possible reduction. One is $t \to \mathtt{case}_A\ s_1\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u, \mathtt{case}_A\ s_2\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u$ and the other is $t \to \mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q_1\ \mathtt{odd}\ u, \mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q_2\ \mathtt{odd}\ u$. It is easy to find the common confluent terms: are the ones in which we have performed both $s \to s_1, s_2$ and $q \to q_1, q_2$.
- If last rule was (T-Rec) our derivation will have the following shape:

$$\frac{\begin{array}{ccc}\Gamma_2;\Delta_4 \vdash q : \mathbf{N} & & \\ \Gamma_2,\Gamma_3;\Delta_5 \vdash s : B & & \Gamma_2;\Delta_4 <: \Box \\ \Gamma_2,\Gamma_3;\vdash r : \Box\mathbf{N} \to \blacksquare B \to B & & B \text{ is } \Box\text{-free}\end{array}}{\Gamma_2,\Gamma_3;\Delta_4,\Delta_5 \vdash \mathtt{recursion}_B\ q\ s\ r : B}\ (\text{T-Rec})$$

  By definition, we can have reduction only in q. By applying induction hypothesis the thesis is proved.
- If last rule was (T-Arr-E). Our term could have different shapes but all of them are trivial or can be easily brought back to cases that we have considered. Also the case where we consider the so called "swap" and the usual application with a lambda abstraction are not interesting in this lemma. Indeed, we cannot consider the "swap" or the substitution case because the reduction relation gives only one term on the right side of the arrow $\to$.

This concludes the proof. □

It is definitely not trivial to prove confluence for $\rightsquigarrow$. For this purpose we will prove our statement on a different definition of *multistep* arrow. This new definition is laxer than the

standard one. Being able to prove our theorems for multistep arrow allows us to conclude that these theorems hold also for $\rightsquigarrow$.

Definition 2.11. The binary relation $\Rightarrow$ is a set of pairs whose first component is a term and whose second component is a distribution on *not-necessarily-normal* terms, namely a function $\mathscr{D} : \Lambda \to [0,1]$ such that $\sum_{t\in\Lambda} \mathscr{D}(t) = 1$. As usual, $\mathscr{I}_t$ is the distribution that maps term t to 1 and any other term to 0. It is easy to check that if $t \rightsquigarrow \mathscr{D}$ then $t \Rightarrow \mathscr{D}$ (but not vice-versa). Formally, rules for $\Rightarrow$ are in Figure 5.

$$\frac{t \to t_1, \ldots, t_n \qquad t_i \Rightarrow \mathscr{D}_i}{t \Rightarrow \sum_{i=1}^{n} \frac{1}{n} \mathscr{D}_i} \qquad\qquad \frac{}{t \Rightarrow \mathscr{I}_t}$$

Figure 5: Generalized Multistep Reduction: Inference Rules

In any formal system, the height $||\pi||$ of a any derivation π is just the height of the derivation, seen as a tree (where by convention leaves have null height). Write $t \stackrel{n}{\Rightarrow} \mathscr{D}$ if the height $||\pi||$ of the derivation $\pi : t \Rightarrow \mathscr{D}$ is *bounded* by n. We can generalize $\stackrel{n}{\Rightarrow}$ to a ternary relation on *distributions* by the rules in Figure 6. When $\mathscr{D} \stackrel{n}{\Rightarrow} \mathscr{E}$ for some n, we simply write $\mathscr{D} \Rightarrow \mathscr{E}$.

$$\frac{\mathscr{D} \stackrel{n}{\Rightarrow} \{t_1^{\alpha_1}, \ldots, t_k^{\alpha_k}\} \qquad t_i \stackrel{m}{\Rightarrow} \mathscr{E}_i}{\mathscr{D} \stackrel{n+m}{\Rightarrow} \sum_{i=1}^{k} \alpha_i \cdot \mathscr{E}_i} \qquad\qquad \frac{}{\mathscr{D} \stackrel{n}{\Rightarrow} \mathscr{D}}$$

Figure 6: Generalized Multistep Reduction on Distributions: Inference Rules

Lemma 2.8. *If $\mathscr{D} \stackrel{n}{\Rightarrow} \mathscr{E}$ and $m \geq n$, then $\mathscr{D} \stackrel{m}{\Rightarrow} \mathscr{E}$.*

Proof. A simple induction on the structure of the proof that $\mathscr{D} \stackrel{n}{\Rightarrow} \mathscr{E}$. □

Lemma 2.9. *If $\mathscr{D} \stackrel{n}{\Rightarrow} \mathscr{E}$ and $\mathscr{E} \stackrel{m}{\Rightarrow} \mathscr{P}$, then $\mathscr{D} \stackrel{n+m}{\Rightarrow} \mathscr{P}$.*

Proof. By induction on the structure of the proof that $\mathscr{E} \stackrel{m}{\Rightarrow} \mathscr{P}$:

- If $\mathscr{E} = \mathscr{P}$, then $\mathscr{D} \stackrel{n}{\Rightarrow} \mathscr{E} = \mathscr{P}$ by hypothesis and, by Lemma 2.8, $\mathscr{D} \stackrel{n+m}{\Rightarrow} \mathscr{P}$;
- If the last inference step in the proof of $\mathscr{E} \stackrel{m}{\Rightarrow} \mathscr{P}$ looks as follows

$$\frac{\mathscr{E} \stackrel{x}{\Rightarrow} \{t_1^{\alpha_1}, \ldots, t_k^{\alpha_k}\} \qquad t_i \stackrel{y}{\Rightarrow} \mathscr{L}_i}{\mathscr{E} \stackrel{x+y}{\Rightarrow} \sum_{i=1}^{k} \alpha_i \cdot \mathscr{L}_i}$$

then we can apply the inductive hypothesis and conclude that $\mathscr{D} \overset{n+x}{\Rightarrow} \{t_1^{\alpha_1}, \ldots, t_k^{\alpha_k}\}$, from which one easily gets the thesis (since $m = x + y$):

$$\frac{\mathscr{D} \overset{n+x}{\Rightarrow} \{t_1^{\alpha_1}, \ldots, t_k^{\alpha_k}\} \qquad t_i \overset{y}{\Rightarrow} \mathscr{L}_i}{\mathscr{E} \overset{n+x+y}{\Rightarrow} \sum_{i=1}^{k} \alpha_i \cdot \mathscr{L}_i}.$$

This concludes the proof. □

Lemma 2.10. *$\mathscr{D} \overset{0}{\Rightarrow} \mathscr{E}$, then $\mathscr{D} = \mathscr{E}$.*

Proof. By an easy induction on the structure of a proof that $\mathscr{D} \overset{0}{\Rightarrow} \mathscr{E}$. □

Lemma 2.11. *If $\mathscr{D} \overset{n+1}{\Rightarrow} \mathscr{E}$, then there is $\mathscr{P}$ such that $\mathscr{D} \overset{1}{\Rightarrow} \mathscr{P} \overset{n}{\Rightarrow} \mathscr{E}$.*

Proof. An induction on the structure of a proof that $\mathscr{D} \overset{n+1}{\Rightarrow} \mathscr{E}$:

- If $\mathscr{D} = \mathscr{E}$, then of course $\mathscr{D} \overset{1}{\Rightarrow} \mathscr{D} \overset{n}{\Rightarrow} \mathscr{D} = \mathscr{E}$.
- If the last inference step in the proof of $\mathscr{D} \overset{n+1}{\Rightarrow} \mathscr{E}$ looks as follows

$$\frac{\mathscr{D} \overset{x}{\Rightarrow} \{t_1^{\alpha_1}, \ldots, t_k^{\alpha_k}\} \qquad t_i \overset{y}{\Rightarrow} \mathscr{L}_i}{\mathscr{D} \overset{x+y}{\Rightarrow} \sum_{i=1}^{k} \alpha_i \cdot \mathscr{L}_i = \mathscr{E}}$$

  and $x \geq 1$, then there is z such that $x = z + 1$. We can apply the induction hypothesis, obtaining a distribution $\mathscr{J}$ such that $\mathscr{D} \overset{1}{\Rightarrow} \mathscr{J} \overset{z}{\Rightarrow} \{t_1^{\alpha_1}, \ldots, t_k^{\alpha_k}\}$. Moreover, we can easily prove that $\mathscr{J} \overset{z+y}{\Rightarrow} \sum_{i=1}^{k} \alpha_i \cdot \mathscr{L}_i$:

$$\frac{\mathscr{J} \overset{z}{\Rightarrow} \{t_1^{\alpha_1}, \ldots, t_k^{\alpha_k}\} \qquad t_i \overset{y}{\Rightarrow} \mathscr{L}_i}{\mathscr{J} \overset{z+y}{\Rightarrow} \sum_{i=1}^{k} \alpha_i \cdot \mathscr{L}_i}$$

- If the last inference step in the proof of $\mathscr{D} \overset{n+1}{\Rightarrow} \mathscr{E}$ looks as follows

$$\frac{\mathscr{D} \overset{x}{\Rightarrow} \{t_1^{\alpha_1}, \ldots, t_k^{\alpha_k}\} \qquad t_i \overset{y}{\Rightarrow} \mathscr{L}_i}{\mathscr{D} \overset{x+y}{\Rightarrow} \sum_{i=1}^{k} \alpha_i \cdot \mathscr{L}_i = \mathscr{E}}$$

  and $x = 0$, then by Lemma 2.8 we can conclude that $\mathscr{D} = \{t_1^{\alpha_1}, \ldots, t_k^{\alpha_k}\}$. For every $1 \leq i \leq n$ there is a sequence $s_{i,1}, \ldots, s_{i,m_i}$ such that:
  - either $m_i = 1$, $t_i = s_{i,1}$ and $\mathscr{L}_i = \{t_i^1\}$;
  - or $t_i \to s_{i,1}, \ldots, s_{i,m_i}$, for every $1 \leq j \leq m_i$ there is $\mathscr{J}_j^i$ such that $s_{i,j} \overset{z}{\Rightarrow} \mathscr{J}_j^i$, where $y = z + 1$ and $\mathscr{L}_i = \sum_{j=1}^{m_i} \frac{1}{m_i} \cdot \mathscr{J}_j^i$.

  The required distribution $\mathscr{P}$, then, is just

$$\{s_{1,1}^{\frac{\alpha_1}{m_1}}, \ldots, s_{1,m_1}^{\frac{\alpha_1}{m_1}}, \ldots, s_{k,1}^{\frac{\alpha_k}{m_k}}, \ldots, s_{k,m_k}^{\frac{\alpha_k}{m_k}}\}.$$

  The fact $\mathscr{D} \overset{1}{\Rightarrow} \mathscr{P} \overset{n}{\Rightarrow} \mathscr{E}$ can be easily derived, since

$$\mathscr{E} = \sum_{i=1}^{k} \alpha_i \cdot \mathscr{L}_i = \sum_{i=1}^{k} \alpha_i \cdot \sum_{j=1}^{m_i} \frac{1}{m_i} \cdot \mathscr{J}_j^i = \sum_{i=1}^{k} \sum_{j=1}^{m_i} \frac{\alpha_i}{m_i} \cdot \mathscr{J}_j^i.$$

This concludes the proof. □

Lemma 2.12. *If $\mathscr{D} \overset{1}{\Rightarrow} \mathscr{E}$ and $\mathscr{D} \overset{1}{\Rightarrow} \mathscr{P}$, then there is $\mathscr{L}$ such that $\mathscr{E} \overset{1}{\Rightarrow} \mathscr{L}$ and $\mathscr{P} \overset{1}{\Rightarrow} \mathscr{L}$.*

Proof. By induction on the structure of the proofs that $\mathscr{D} \overset{1}{\Rightarrow} \mathscr{E}$ and $\mathscr{D} \overset{1}{\Rightarrow} \mathscr{P}$. The only interesting case is the one in which $\mathscr{E} = \sum_{i=1}^{n} \alpha_i \cdot \mathscr{L}_i$ and $\mathscr{P} = \sum_{j=1}^{m} \beta_j \cdot \mathscr{J}_j$:

$$\begin{aligned}
&\mathscr{D} = \{t_1^{\alpha_1}, \ldots, t_n^{\alpha_n}\};\\
&\mathscr{D} = \{s_1^{\beta_1}, \ldots, s_m^{\beta_m}\};\\
&\quad \forall i \in \{1, \ldots, n\}.t_i \overset{1}{\Rightarrow} \mathscr{L}_i;\\
&\quad \forall j \in \{1, \ldots, m\}.s_j \overset{1}{\Rightarrow} \mathscr{J}_j.
\end{aligned}$$

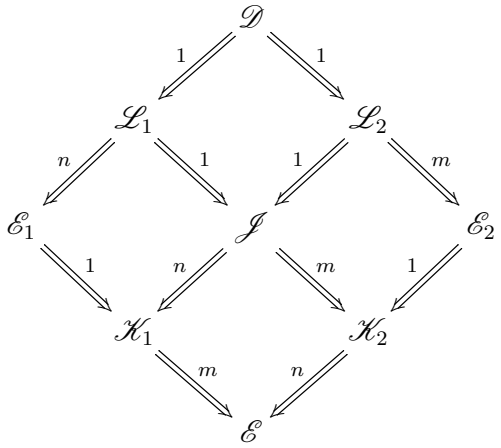
Without losing generality, we can assume that $n = m$, $t_i = s_i$ and $\alpha_i = \beta_i$. Moreover, $\mathscr{E}$ can be written as follows

$$\{r_{1,1}^{\frac{\alpha_1}{m_1}}, \ldots, r_{1,m_1}^{\frac{\alpha_1}{m_1}}, \ldots, r_{n,1}^{\frac{\alpha_n}{m_n}}, \ldots, r_{n,m_n}^{\frac{\alpha_n}{m_n}}\},$$

where either $m_i = 1$ and $r_{i,1} = t_i$ or $t_i \to r_{i,1}, \ldots, r_{i,m_i}$. Similarly for $\mathscr{P}$. By exploiting Lemma 2.5, Lemma 2.6 and Lemma 2.7, the distribution $\mathscr{L}$ can be easily defined. □

Theorem 2.13. *If $\mathscr{D} \Rightarrow \mathscr{E}$ and $\mathscr{D} \Rightarrow \mathscr{P}$, then there is $\mathscr{L}$ such that $\mathscr{E} \Rightarrow \mathscr{L}$ and $\mathscr{P} \Rightarrow \mathscr{L}$.*

Proof. We will actually prove a strengthening of the theorem above, namely the following: If $\mathscr{D} \overset{n}{\Rightarrow} \mathscr{E}_1$ and $\mathscr{D} \overset{m}{\Rightarrow} \mathscr{E}_2$, then there is $\mathscr{P}$ such that $\mathscr{E}_1 \overset{m}{\Rightarrow} \mathscr{P}$ and $\mathscr{E}_2 \overset{n}{\Rightarrow} \mathscr{P}$. This goes by a very simple induction, whose only interesting case is the one where $n, m \geq 1$. In that case, Lemma 2.11 guarantees that there are $\mathscr{L}_1, \mathscr{L}_2$ such that $\mathscr{D} \overset{1}{\Rightarrow} \mathscr{L}_1 \overset{n}{\Rightarrow} \mathscr{E}_1$ and $\mathscr{D} \overset{1}{\Rightarrow} \mathscr{L}_2 \overset{m}{\Rightarrow} \mathscr{E}_2$. By Lemma 2.12, one can build a distribution $\mathscr{J}$ such that $\mathscr{L}_1 \overset{1}{\Rightarrow} \mathscr{J}_1$ and $\mathscr{L}_2 \overset{1}{\Rightarrow} \mathscr{J}_2$. By the induction hypothesis (applied twice) one obtains $\mathscr{K}_1, \mathscr{K}_2$ such that $\mathscr{J} \overset{n}{\Rightarrow} \mathscr{K}_1$, $\mathscr{E}_1 \overset{1}{\Rightarrow} \mathscr{K}_1$, $\mathscr{J} \overset{m}{\Rightarrow} \mathscr{K}_2$, $\mathscr{E}_2 \overset{1}{\Rightarrow} \mathscr{K}_2$. Finally, another application of the induction hypothesis gives us a distribution $\mathscr{P}$ such that $\mathscr{K}_1 \overset{m}{\Rightarrow} \mathscr{P}$ and $\mathscr{K}_2 \overset{n}{\Rightarrow} \mathscr{P}$. Lemma 2.9 is now enough to get the thesis. Summing up, the structure of the proof is the classic one:



□

Corollary 2.14 (Multistep Confluence)**.** *If $t \rightsquigarrow \mathscr{D}$ and $t \rightsquigarrow \mathscr{E}$ then $\mathscr{D} = \mathscr{E}$.*

Proof. An easy consequence of Theorem 2.13: if $t \rightsquigarrow \mathscr{D}$ and $t \rightsquigarrow \mathscr{E}$, then $\{t^1\} \Rightarrow \mathscr{D}$ and $\{t^1\} \Rightarrow \mathscr{E}$. As a consequence, $\mathscr{D} \Rightarrow \mathscr{P}$ and $\mathscr{E} \Rightarrow \mathscr{P}$, but $\mathscr{D} = \mathscr{P} = \mathscr{E}$, because all the terms to which $\mathscr{D}$ and $\mathscr{E}$ attribute a nonnull probability are normal forms. □

Example 2.3. Consider again the term $t = (\lambda x : \blacksquare\mathbf{N}.(t_\oplus x x))\mathtt{rand}$, where $t_\oplus$ is a term computing $\oplus$ on natural numbers seen as booleans (0 stands for "false" and everything else stands for "true"):

$$\begin{aligned} t_\oplus &= \lambda x : \blacksquare\mathbf{N}.\mathtt{case}_{\blacksquare\mathbf{N}\to\mathbf{N}}\ x\ \mathtt{zero}\ s_\oplus\ \mathtt{even}\ r_\oplus\ \mathtt{odd}\ r_\oplus; \\ s_\oplus &= \lambda y : \blacksquare\mathbf{N}.\mathtt{case}_{\mathbf{N}}\ y\ \mathtt{zero}\ 0\ \mathtt{even}\ 1\ \mathtt{odd}\ 1; \\ r_\oplus &= \lambda y : \blacksquare\mathbf{N}.\mathtt{case}_{\mathbf{N}}\ y\ \mathtt{zero}\ 1\ \mathtt{even}\ 0\ \mathtt{odd}\ 0. \end{aligned}$$

In order to simplify reading, let us define:

$$\begin{aligned} f &= (t_\oplus x x); \\ g_0 &= (\mathtt{case}_{\blacksquare\mathbf{N}\to\mathbf{N}}\ 0\ \mathtt{zero}\ s_\oplus\ \mathtt{even}\ r_\oplus\ \mathtt{odd}\ r_\oplus); \\ g_1 &= (\mathtt{case}_{\blacksquare\mathbf{N}\to\mathbf{N}}\ 1\ \mathtt{zero}\ s_\oplus\ \mathtt{even}\ r_\oplus\ \mathtt{odd}\ r_\oplus); \\ h_0 &= \mathtt{case}_{\mathbf{N}}\ 0\ \mathtt{zero}\ 0\ \mathtt{even}\ 1\ \mathtt{odd}\ 1; \\ h_1 &= \mathtt{case}_{\mathbf{N}}\ 1\ \mathtt{zero}\ 1\ \mathtt{even}\ 0\ \mathtt{odd}\ 0. \end{aligned}$$

We can now give the following derivation tree:

$$\dfrac{(\lambda x : \blacksquare\mathbf{N}.f)\mathtt{rand} \to (\lambda x : \blacksquare\mathbf{N}.f)0, (\lambda x : \blacksquare\mathbf{N}.f)1 \qquad \pi : (\lambda x : \blacksquare\mathbf{N}.f)0 \rightsquigarrow \{0^1\} \qquad \rho : (\lambda x : \blacksquare\mathbf{N}.f)1 \rightsquigarrow \{0^1\}}{(\lambda x : \blacksquare\mathbf{N}.(t_\oplus x x))\mathtt{rand} \rightsquigarrow \{0^1\}}$$

where π and ρ are as follows:

$$\pi : \dfrac{(\lambda x : \blacksquare\mathbf{N}.f)0 \to t_\oplus 00 \qquad \dfrac{t_\oplus 0\,0 \to g_0 0 \qquad \dfrac{g_0 0 \to s_\oplus 0 \qquad \dfrac{s_\oplus 0 \to h_0 \qquad \dfrac{h_0 \to 0 \qquad 0 \rightsquigarrow \{0^1\}}{h_0 \rightsquigarrow \{0^1\}}}{s_\oplus 0 \rightsquigarrow \{0^1\}}}{g_0 0 \rightsquigarrow \{0^1\}}}{(\lambda x : \blacksquare\mathbf{N}.\mathtt{case}_{\blacksquare\mathbf{N}\to\mathbf{N}}\ x\ \mathtt{zero}\ s_\oplus\ \mathtt{even}\ r_\oplus\ \mathtt{odd}\ r_\oplus)0\,0 \rightsquigarrow \{0^1\}}}{(\lambda x : \blacksquare\mathbf{N}.f)0 \rightsquigarrow \{0^1\}}$$

$$\rho : \dfrac{(\lambda x : \blacksquare\mathbf{N}.f)1 \to t_\oplus 11 \qquad \dfrac{t_\oplus 1\,1 \to g_1 1 \qquad \dfrac{g_1 1 \to r_\oplus 1 \qquad \dfrac{r_\oplus 1 \to h_1 \qquad \dfrac{h_1 \to 0 \qquad 0 \rightsquigarrow \{0^1\}}{h_1 \rightsquigarrow \{0^1\}}}{r_\oplus 1 \rightsquigarrow \{0^1\}}}{g_1 1 \rightsquigarrow \{0^1\}}}{(\lambda x : \blacksquare\mathbf{N}.\mathtt{case}_{\blacksquare\mathbf{N}\to\mathbf{N}}\ x\ \mathtt{zero}\ s_\oplus\ \mathtt{even}\ r_\oplus\ \mathtt{odd}\ r_\oplus)1\,1 \rightsquigarrow \{0^1\}}}{(\lambda x : \blacksquare\mathbf{N}.f)1 \rightsquigarrow \{0^1\}}$$

3. Probabilistic Polytime Soundness

The most difficult (and interesting!) result about RSLR is definitely polytime soundness: every (instance of) a first-order term can be reduced to a numeral in a polynomial number of steps by a probabilistic Turing machine. Polytime soundness can be proved, following [2], by showing that:

- Any explicit term of base type can be reduced to its normal form with very low time complexity;
- Any term (non necessarily of base type) can be put in explicit form in polynomial time.

By gluing these two results together, we obtain what we need, namely an effective and efficient procedure to compute the normal forms of terms. Formally, two notions of evaluation for terms correspond to the two steps defined above:

- On the one hand, we need a ternary relation $\Downarrow_{\mathsf{nf}}$ between closed terms of type $\mathbf{N}$, probabilities and numerals. Intuitively, $t \Downarrow_{\mathsf{nf}}^{\alpha} n$ holds when t is explicit and rewrites to n with probability α. The inference rules for $\Downarrow_{\mathsf{nf}}$ are defined in Figure 7;
- On the other hand, we need a ternary relation $\Downarrow_{\mathsf{rf}}$ between terms of non modal type, probabilities and terms. We can derive $t \Downarrow_{\mathsf{rf}}^{\alpha} s$ only if t can be transformed into s with probability α consistently with the reduction relation. The inference rules for $\Downarrow_{\mathsf{rf}}$ are in Figure 8.

$$\frac{}{n \Downarrow_{\mathsf{nf}}^{1} n} \qquad \frac{}{\mathtt{rand} \Downarrow_{\mathsf{nf}}^{1/2} 0} \qquad \frac{}{\mathtt{rand} \Downarrow_{\mathsf{nf}}^{1/2} 1}$$

$$\frac{t \Downarrow_{\mathsf{nf}}^{\alpha} n}{\mathtt{S}_0 t \Downarrow_{\mathsf{nf}}^{\alpha} 2 \cdot n} \qquad \frac{t \Downarrow_{\mathsf{nf}}^{\alpha} n}{\mathtt{S}_1 t \Downarrow_{\mathsf{nf}}^{\alpha} 2 \cdot n + 1} \qquad \frac{t \Downarrow_{\mathsf{nf}}^{\alpha} n}{\mathtt{P} t \Downarrow_{\mathsf{nf}}^{\alpha} \lfloor \frac{n}{2} \rfloor}$$

$$\frac{t \Downarrow_{\mathsf{nf}}^{\alpha} 0 \qquad s\overline{u} \Downarrow_{\mathsf{nf}}^{\beta} n}{(\mathtt{case}_A\ t\ \mathtt{zero}\ s\ \mathtt{even}\ r\ \mathtt{odd}\ q)\overline{u} \Downarrow_{\mathsf{nf}}^{\alpha\beta} n} \qquad \frac{t \Downarrow_{\mathsf{nf}}^{\alpha} 2n \qquad r\overline{u} \Downarrow_{\mathsf{nf}}^{\beta} m \qquad n \geq 1}{(\mathtt{case}_A\ t\ \mathtt{zero}\ s\ \mathtt{even}\ r\ \mathtt{odd}\ q)\overline{u} \Downarrow_{\mathsf{nf}}^{\alpha\beta} m}$$

$$\frac{t \Downarrow_{\mathsf{nf}}^{\alpha} 2n+1 \qquad q\overline{u} \Downarrow_{\mathsf{nf}}^{\beta} m}{(\mathtt{case}_A\ t\ \mathtt{zero}\ s\ \mathtt{even}\ r\ \mathtt{odd}\ q)\overline{u} \Downarrow_{\mathsf{nf}}^{\alpha\beta} m}$$

$$\frac{s \Downarrow_{\mathsf{nf}}^{\alpha} n \qquad (t[x/n])\overline{r} \Downarrow_{\mathsf{nf}}^{\beta} m}{(\lambda x : a\mathbf{N}.t)s\overline{r} \Downarrow_{\mathsf{nf}}^{\alpha\beta} m} \qquad \frac{(t[x/s])\overline{r} \Downarrow_{\mathsf{nf}}^{\beta} n}{(\lambda x : aH.t)s\overline{r} \Downarrow_{\mathsf{nf}}^{\beta} n}$$

Figure 7: The Relation $\Downarrow_{\mathsf{nf}}$: Inference Rules

Moreover, a third ternary relation $\Downarrow$ between closed terms of type $\mathbf{N}$, probabilities and numerals can be defined by the rule below:

$$\frac{t \Downarrow_{\mathsf{rf}}^{\alpha} s \qquad s \Downarrow_{\mathsf{nf}}^{\beta} n}{t \Downarrow^{\alpha\beta} n}$$

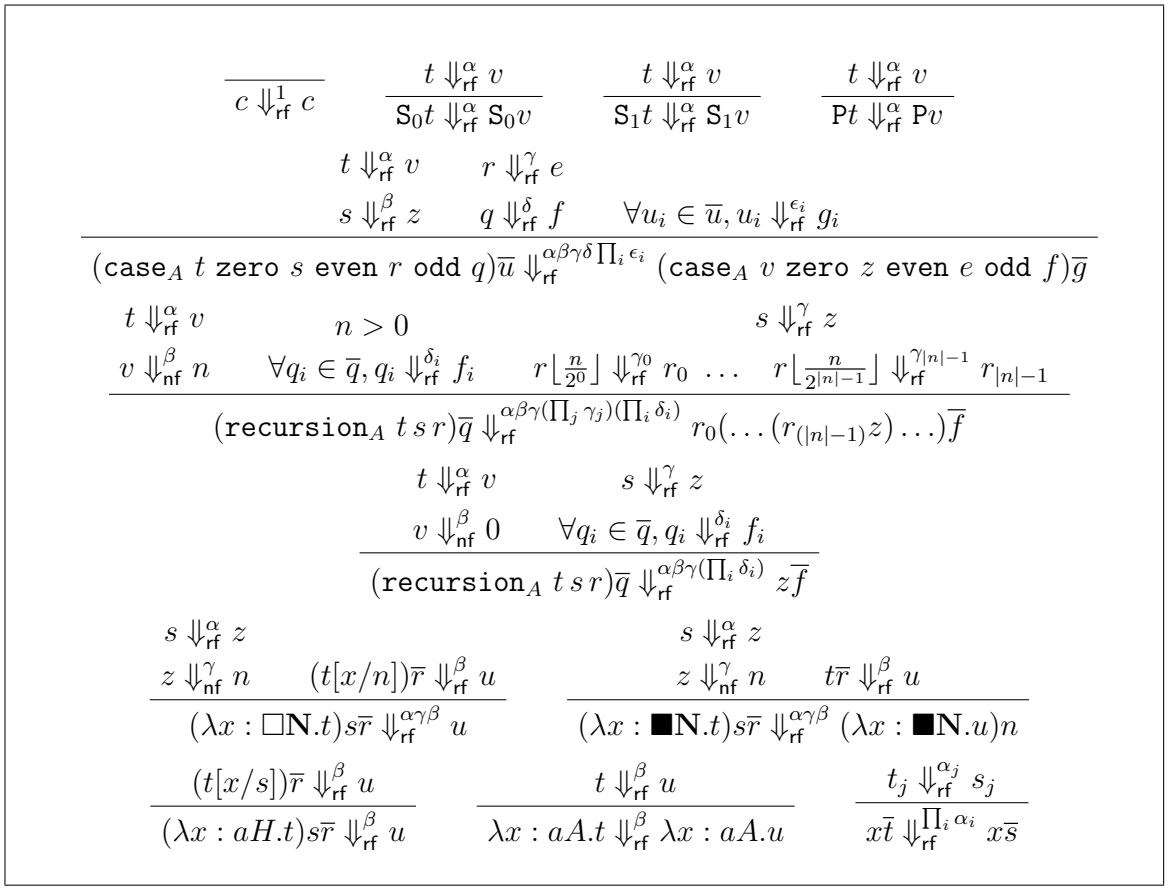


Figure 8: The Relation $\Downarrow_{\mathsf{rf}}$: Inference Rules

A peculiarity of the just introduced relations with respect to similar ones is the following: whenever a statement in the form $t \Downarrow_{\mathsf{nf}}^{\alpha} s$ is an immediate premise of another statement $r \Downarrow_{\mathsf{nf}}^{\beta} q$, then t needs to be structurally smaller than r, provided all numerals are assumed to have the same internal structure. A similar but weaker statement holds for $\Downarrow_{\mathsf{rf}}$. This relies on the peculiarities of RSLR, and in particular on the fact that variables of higher-order types can appear free at most once in terms, and that terms of base type cannot be passed to functions without having been completely evaluated. In other words, the just described operational semantics is structural in a very strong sense, and this allows to prove properties about it by induction on the structure of *terms*, as we will experience in a moment.

Before starting to study the combinatorial properties of $\Downarrow_{\mathsf{rf}}$ and $\Downarrow_{\mathsf{nf}}$, it is necessary to show that, at least, $\Downarrow$ is adequate as a way to evaluate lambda terms. In the following, the size $|\pi|$ of any derivation π (for any formal system) is simply the number of distinct rule occurrences in π.

Theorem 3.1 (Adequacy)**.** *For every term t such that $\vdash t : \mathbf{N}$, the following two conditions are equivalent:*

1. *There are j distinct derivations $\pi_1 : t \Downarrow^{\alpha_1} n_1, \ldots, \pi_j : t \Downarrow^{\alpha_j} n_j$ such that $\sum_{i=1}^{j} \alpha_i = 1$;*
2. *$t \rightsquigarrow \mathscr{D}$, where for every m, $\mathscr{D}(m) = \sum_{n_i = m} \alpha_i$.*

Proof. Implication 1. $\Rightarrow$ 2. can be proved by an induction on $\sum_{k=1}^{j} |\pi_k|$, appropriately enriching the thesis with similar statements for $\Downarrow_{\mathsf{rf}}$ and $\Downarrow_{\mathsf{nf}}$. About the converse, just observe that, *some* derivations like the ones required in Condition 1. need to exist. On $\Downarrow_{\mathsf{nf}}$, this can be formally proved by induction on $|t|_{\mathsf{w}}$, where $|\cdot|_{\mathsf{w}}$ is defined as follows: $|x|_{\mathsf{w}} = 1$, $|ts|_{\mathsf{w}} = |t|_{\mathsf{w}} + |s|_{\mathsf{w}}$, $|\lambda x : aA.t|_{\mathsf{w}} = |t|_{\mathsf{w}} + 1$, $|\mathtt{case}_A\ t\ \mathtt{zero}\ s\ \mathtt{even}\ r\ \mathtt{odd}\ q|_{\mathsf{w}} = |t|_{\mathsf{w}} + |s|_{\mathsf{w}} + |r|_{\mathsf{w}} + |q|_{\mathsf{w}} + 1$, $|\mathtt{recursion}_A\ t\, s\, r|_{\mathsf{w}} = |t|_{\mathsf{w}} + |s|_{\mathsf{w}} + |r|_{\mathsf{w}} + 1$, $|n|_{\mathsf{w}} = 1$, $|\mathtt{S}_0|_{\mathsf{w}} = |\mathtt{S}_1|_{\mathsf{w}} = |\mathtt{P}|_{\mathsf{w}} = |\mathtt{rand}|_{\mathsf{w}} = 1$. On $\Downarrow_{\mathsf{rf}}$, the same can be proved by induction on a lexicographic order taking into account the recursion depth of t and $|t|_{\mathsf{w}}$. Thanks to multistep confluence, we can conclude. □

It's now time to analyse how big derivations for $\Downarrow_{\mathsf{nf}}$ and $\Downarrow_{\mathsf{rf}}$ can be with respect to the size of the underlying term. Let us start with $\Downarrow_{\mathsf{nf}}$ and prove that, since it can only be applied to explicit terms, the sizes of derivations must be very small:

Proposition 3.2. *Suppose that $\vdash t : \mathbf{N}$, where t is explicit. Then for every $\pi : t \Downarrow_{\mathsf{nf}}^{\alpha} m$ it holds that:*

1. *$|\pi| \leq |t|$;*
2. *If $s \in \pi$, then $|s| \leq 2 \cdot |t|^2$.*

Proof. Given any term t, $|t|_{\mathsf{w}}$ and $|t|_{\mathsf{n}}$ are defined, respectively, as the size of t where every numeral counts for 1 and the maximum size of the numerals that occur in t. For a formal definition of $|\cdot|_{\mathsf{w}}$, see the proof of Theorem 3.1. On the other hand, $|\cdot|_{\mathsf{n}}$ is defined as follows: $|x|_{\mathsf{n}} = 1$, $|ts|_{\mathsf{n}} = \max\{|t|_{\mathsf{n}}, |s|_{\mathsf{n}}\}$, $|\lambda x : aA.t|_{\mathsf{n}} = |t|_{\mathsf{n}}$, $|\mathtt{case}_A\ t\ \mathtt{zero}\ s\ \mathtt{even}\ r\ \mathtt{odd}\ q|_{\mathsf{n}} = \max\{|t|_{\mathsf{n}}, |s|_{\mathsf{n}}, |r|_{\mathsf{n}}, |q|_{\mathsf{n}}\}$, $|\mathtt{recursion}_A\ t\, s\, r|_{\mathsf{n}} = \max\{|t|_{\mathsf{n}}, |s|_{\mathsf{n}}, |r|_{\mathsf{n}}\}$, $|n|_{\mathsf{n}} = \lfloor \log_2(n) \rfloor + 1$,

and $|\mathtt{S}_0|_\mathsf{n} = |\mathtt{S}_1|_\mathsf{n} = |\mathtt{P}|_\mathsf{n} = |\mathtt{rand}|_\mathsf{n} = 1$. It holds that $|t| \leq |t|_\mathsf{w} \cdot |t|_\mathsf{n}$. It can be proved by structural induction on term t. We prove the following strengthening of the statements above by induction on $|t|_\mathsf{w}$:

1. $|\pi| \leq |t|_\mathsf{w}$;
2. If $s \in \pi$, then $|s|_\mathsf{w} \leq |t|_\mathsf{w}$ and $|s|_\mathsf{n} \leq |t|_\mathsf{n} + |t|_\mathsf{w}$;

First we prove that this is indeed a strengthening of the thesis. From the first case of the strengthening, we can deduce the first case of the main thesis. Notice indeed that $|t|_\mathsf{w} \leq |t|$. Regarding the latter point, notice that $|s| \leq |s|_\mathsf{w} \cdot |s|_\mathsf{n} \leq |t|_\mathsf{w} \cdot (|t|_\mathsf{n} + |t|_\mathsf{w}) \leq |t|^2 + |t| \leq 2 \cdot |t|^2$. Some interesting cases:

- Suppose t is `rand`. We could have two derivations:

$$\frac{}{\mathtt{rand} \Downarrow_\mathsf{nf}^{1/2} 0} \qquad \frac{}{\mathtt{rand} \Downarrow_\mathsf{nf}^{1/2} 1}$$

  The thesis is easily proved.
- Suppose t is $\mathtt{S}_i s$. Depending on $\mathtt{S}_i$ we could have two different derivations:

$$\frac{\rho : s \Downarrow_\mathsf{nf}^\alpha n}{\mathtt{S}_0 s \Downarrow_\mathsf{nf}^\alpha 2 \cdot n} \qquad \frac{\rho : s \Downarrow_\mathsf{nf}^\alpha n}{\mathtt{S}_1 s \Downarrow_\mathsf{nf}^\alpha 2 \cdot n + 1}$$

  Suppose we are in the case where $\mathtt{S}_i = \mathtt{S}_0$. Then, for every $r \in \pi$,

$$\begin{aligned} |\pi| &= |\rho| + 1 \leq |s|_\mathsf{w} + 1 = |t|_\mathsf{w}; \\ |r|_\mathsf{w} &\leq |s|_\mathsf{w} \leq |t|_\mathsf{w}; \\ |r|_\mathsf{n} &\leq |s|_\mathsf{n} + |s|_\mathsf{w} + 1 = |s|_\mathsf{n} + |t|_\mathsf{w} \ = |t|_\mathsf{n} + |t|_\mathsf{w}. \end{aligned}$$

  The case where $\mathtt{S}_i = \mathtt{S}_1$ is proved in the same way.
- Suppose t is $\mathtt{P}s$.

$$\frac{\rho : s \Downarrow_\mathsf{nf}^\alpha 0}{\mathtt{P}s \Downarrow_\mathsf{nf}^\alpha 0} \qquad \frac{\rho : s \Downarrow_\mathsf{nf}^\alpha n \qquad n \geq 1}{\mathtt{P}s \Downarrow_\mathsf{nf}^\alpha \lfloor \frac{n}{2} \rfloor}$$

  We focus on case where $n > 1$, the other case is similar. For every $r \in \pi$ we have

$$\begin{aligned} |\pi| &= |\rho| + 1 \leq |s|_\mathsf{w} + 1 = |t|_\mathsf{w}; \\ |r|_\mathsf{w} &\leq |s|_\mathsf{w} \leq |t|_\mathsf{w}; \\ |r|_\mathsf{n} &\leq |s|_\mathsf{n} + |s|_\mathsf{w} + 1 = |s|_\mathsf{n} + |t|_\mathsf{w} = |t|_\mathsf{n} + |t|_\mathsf{w}. \end{aligned}$$

- Suppose t is n.

$$\frac{}{n \Downarrow_\mathsf{nf}^1 n}$$

  By knowing $|\pi| = 1$, $|n|_\mathsf{w} = 1$ and $|n|_\mathsf{n} = |n|$, the proof is trivial.
- Suppose that t is $(\lambda y : a\mathbf{N}.s)r\overline{q}$. All derivations π for t are in the following form:

$$\frac{\rho : r \Downarrow_\mathsf{nf}^\alpha o \qquad \mu : (s[y/o])\overline{q} \Downarrow_\mathsf{nf}^\beta m}{t \Downarrow_\mathsf{nf}^{\alpha\beta} m}$$

Then, for every $u \in \pi$,

$$\begin{aligned}
|\pi| &\leq |\rho| + |\mu| + 1 \leq |r|_{\mathsf{w}} + |s[y/o]\overline{q}|_{\mathsf{w}} + 1 \\
&= |r|_{\mathsf{w}} + |s\overline{q}|_{\mathsf{w}} + 1 \leq |t|_{\mathsf{w}}; \\
|u|_{\mathsf{n}} &\leq \max\{|r|_{\mathsf{n}} + |r|_{\mathsf{w}}, |s[y/o]\overline{q}|_{\mathsf{n}} + |s[y/o]\overline{q}|_{\mathsf{w}}\} \\
&= \max\{|r|_{\mathsf{n}} + |r|_{\mathsf{w}}, |s[y/o]\overline{q}|_{\mathsf{n}} + |s\overline{q}|_{\mathsf{w}}\} \\
&= \max\{|r|_{\mathsf{n}} + |r|_{\mathsf{w}}, \max\{|s\overline{q}|_{\mathsf{n}}, |o|\} + |s\overline{q}|_{\mathsf{w}}\} \\
&= \max\{|r|_{\mathsf{n}} + |r|_{\mathsf{w}}, |s\overline{q}|_{\mathsf{n}} + |s\overline{q}|_{\mathsf{w}}, |o| + |s\overline{q}|_{\mathsf{w}}\} \\
&\leq \max\{|r|_{\mathsf{n}} + |r|_{\mathsf{w}}, |s\overline{q}|_{\mathsf{n}} + |s\overline{q}|_{\mathsf{w}}, |r|_{\mathsf{n}} + |r|_{\mathsf{w}} + |s\overline{q}|_{\mathsf{w}}\} \\
&\leq \max\{|r|_{\mathsf{n}}, |s\overline{q}|_{\mathsf{n}}\} + |r|_{\mathsf{w}} + |s\overline{q}|_{\mathsf{w}} \\
&\leq \max\{|r|_{\mathsf{n}}, |s\overline{q}|_{\mathsf{n}}\} + |t|_{\mathsf{w}} \\
&= |t|_{\mathsf{n}} + |t|_{\mathsf{w}}; \\
|u|_{\mathsf{w}} &\leq \max\{|r|_{\mathsf{w}}, |s[y/o]\overline{q}|_{\mathsf{w}}, |t|_{\mathsf{w}}\} \\
&= \max\{|r|_{\mathsf{w}}, |s\overline{q}|_{\mathsf{w}}, |t|_{\mathsf{w}}\} \leq |t|_{\mathsf{w}}.
\end{aligned}$$

If $u \in \pi$, then either $u \in \rho$ or $u \in \mu$ or simply $u = t$. This, together with the induction hypothesis, implies $|u|_{\mathsf{w}} \leq \max\{|r|_{\mathsf{w}}, |s[y/o]\overline{q}|_{\mathsf{w}}, |t|_{\mathsf{w}}\}$. Notice that $|s\overline{q}|_{\mathsf{w}} = |s[y/o]\overline{q}|_{\mathsf{n}}$ holds because any occurrence of y in s counts for 1, but also o itself counts for 1 (see the definition of $|\cdot|_{\mathsf{w}}$ above). More generally, duplication of *numerals* for a variable in t does not make $|t|_{\mathsf{w}}$ bigger.

- Suppose t is $(\lambda y : aH.s)r\overline{q}$. Without loosing generality we can say that it derives from the following derivation:

$$\frac{\rho : (s[y/r])\overline{q} \Downarrow^{\beta}_{\mathsf{nf}} n}{(\lambda y : aH.s)r\overline{q} \Downarrow^{\beta}_{\mathsf{nf}} n}$$

  For the reason that y has type H we can be sure that it appears at most once in s. So, $|s[y/r]| \leq |sr|$ and, moreover, $|s[y/r]\overline{q}|_{\mathsf{w}} \leq |sr\overline{q}|_{\mathsf{w}}$ and $|s[y/r]\overline{q}|_{\mathsf{n}} \leq |sr\overline{q}|_{\mathsf{n}}$. We have, for all $u \in \rho$:

$$\begin{aligned}
|\pi| = |\rho| + 1 &\leq |s[y/r]\overline{q}|_{\mathsf{w}} + 1 \leq |t|_{\mathsf{w}}; \\
|u|_{\mathsf{w}} &\leq |s[y/r]\overline{q}|_{\mathsf{w}} \leq |sr\overline{q}|_{\mathsf{w}} \leq |t|_{\mathsf{w}}; \\
|u|_{\mathsf{n}} &\leq |s[y/r]\overline{q}|_{\mathsf{n}} + |s[y/r]\overline{q}|_{\mathsf{w}} \leq |sr\overline{q}|_{\mathsf{n}} + |sr\overline{q}|_{\mathsf{w}} \leq |t|_{\mathsf{n}} + |t|_{\mathsf{w}};
\end{aligned}$$

  and this means that the same inequalities hold for every $u \in \pi$.

- Suppose t is $\mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u$. We could have three possible derivations:

$$\frac{\rho : s \Downarrow^{\alpha}_{\mathsf{nf}} 0 \qquad \mu : r\overline{v} \Downarrow^{\beta}_{\mathsf{nf}} n}{(\mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u)\overline{v} \Downarrow^{\alpha\beta}_{\mathsf{nf}} n}$$

$$\frac{\rho : s \Downarrow^{\alpha}_{\mathsf{nf}} 2n \qquad \mu : q\overline{v} \Downarrow^{\beta}_{\mathsf{nf}} m \qquad n \geq 1}{(\mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u)\overline{v} \Downarrow^{\alpha\beta}_{\mathsf{nf}} m}$$

$$\frac{\rho : s \Downarrow_{\mathsf{nf}}^{\alpha} 2n+1 \qquad \mu : u\overline{v} \Downarrow_{\mathsf{nf}}^{\beta} m}{(\mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u)\overline{v} \Downarrow_{\mathsf{nf}}^{\alpha\beta} m}$$

we will focus on the case where the value of s is odd. All the other cases are similar. For all $z \in \pi$ we have:

$$\begin{aligned}
|\pi| &\leq |\rho| + |\mu| + 1 \\
&\leq |s|_{\mathsf{w}} + |u\overline{v}|_{\mathsf{w}} + 1 \leq |t|_{\mathsf{w}}; \\
|z|_{\mathsf{w}} &\leq |s|_{\mathsf{w}} + |r|_{\mathsf{w}} + |q|_{\mathsf{w}} + |u\overline{v}|_{\mathsf{w}} \leq |t|_{\mathsf{w}}; \\
|z|_{\mathsf{n}} &= \max\{|s|_{\mathsf{n}} + |s|_{\mathsf{w}}, |u\overline{v}|_{\mathsf{n}} + |u\overline{v}|_{\mathsf{w}}, |r|_{\mathsf{n}}, |q|_{\mathsf{n}}\} \\
&\leq \max\{|s|_{\mathsf{n}}, |u\overline{v}|_{\mathsf{n}}|r|_{\mathsf{n}}, |q|_{\mathsf{n}}\} + |s|_{\mathsf{w}} + |u\overline{v}|_{\mathsf{w}} \\
&\leq |t|_{\mathsf{w}} + |t|_{\mathsf{n}}.
\end{aligned}$$

This concludes the proof. □

As opposed to $\Downarrow_{\mathsf{nf}}$, $\Downarrow_{\mathsf{rf}}$ unrolls instances of primitive recursion, and thus cannot have the very simple combinatorial behaviour of $\Downarrow_{\mathsf{nf}}$. Fortunately, however, everything stays under control:

Proposition 3.3. *Suppose that $x_1 : \Box\mathbf{N}, \ldots, x_i : \Box\mathbf{N} \vdash t : A$, where A is $\Box$-free type. Then there are polynomials p_t and q_t such that for every $n_1, \ldots, n_i$ and for every $\pi : t[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\alpha} s$ it holds that:*

1. *$|\pi| \leq p_t(\sum_i |n_i|)$;*
2. *If $s \in \pi$, then $|s| \leq q_t(\sum_i |n_i|)$.*

Proof. The following strengthening of the result can be proved by induction on the structure of a type derivation μ for t: if $x_1 : \Box\mathbf{N}, \ldots, x_i : \Box\mathbf{N}, y_1 : \blacksquare A_1, \ldots, y_j : \blacksquare A_j \vdash t : A$, where A is positively $\Box$-free and $A_1, \ldots, A_j$ are negatively $\Box$-free. Then there are polynomials p_t and q_t such that for every $n_1, \ldots, n_i$ and for every $\pi : t[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\alpha} s$ it holds that:

1. $|\pi| \leq p_t(\sum_i |n_i|)$;
2. If $s \in \pi$, then $|s| \leq q_t(\sum_i |n_i|)$.

In defining positively and negatively $\Box$-free types, let us proceed by induction on types:

- $\mathbf{N}$ is both positively and negatively $\Box$-free;
- $\Box A \to B$ is *not* positively $\Box$-free, and is negatively $\Box$-free whenever A is positively $\Box$-free and B is negatively $\Box$-free;
- $C = \blacksquare A \to B$ is positively $\Box$-free if A is negatively and B is positively $\Box$-free. C is negatively $\Box$-free if A is positively $\Box$-free and B is negatively $\Box$-free.

Please observe that if A is positively $\Box$-free and $B <: A$, then B is positively $\Box$-free. Conversely, if A is negatively $\Box$-free and $A <: B$, then B is negatively $\Box$-free. This can be easily proved by induction on the structure of A. We are ready to start the proof, now. Let us consider some cases, depending on the shape of μ

- If the only typing rule in μ is (T-Const-Aff), then $t = c$, $p_t(x) = 1$ and $q_t(x) = 1$. The thesis is proved.
- If the last rule was (T-Var-Aff), then $t = x$, $p_t(x) = 1$ and $q_t(x) = x$. The thesis is proved
- If the last rule was (T-Arr-I), then $t = \lambda x : \blacksquare A.s$. Notice that the aspect is $\blacksquare$ because the type of our term has to be positively $\Box$-free. So, we have the following derivation:

$$\frac{\rho : s[\overline{x}/\overline{n}] \Downarrow^{\beta}_{\mathsf{rf}} v}{\lambda x : aA.s[\overline{x}/\overline{n}] \Downarrow^{\beta}_{\mathsf{rf}} \lambda x : aA.v}$$

  If the type of t is positively $\Box$-free, then also the type of s is positively $\Box$-free. We can apply induction hypothesis. Define p_t and q_t as:

$$p_t(x) = p_s(x) + 1;$$
$$q_t(x) = q_s(x) + 1.$$

  Indeed, we have:

$$|\pi| = |\rho| + 1 \leq p_s(\sum_i |n_i|) + 1.$$

- If last rule was (T-Sub) then we have a typing derivation that ends in the following way:

$$\frac{\Gamma \vdash t : A \qquad A <: B}{\Gamma \vdash t : B}$$

  we can apply induction hypothesis on $t : A$ because if B is positively $\Box$-free, then also A will be positively $\Box$-free. Define $p_{t:B}(x) = p_{t:A}(x)$ and $q_{t:B}(x) = q_{t:A}(x)$.
- If the last rule was (T-Case), suppose $t = (\mathtt{case}_A\ s\ \mathtt{zero}\ r\ \mathtt{even}\ q\ \mathtt{odd}\ u)$. The constraints on the typing rule (T-Case) ensure us that the induction hypothesis can be applied to s, r, q, u. The definition of $\Downarrow_{\mathsf{rf}}$ tells us that any derivation of $t[\overline{x}/\overline{n}]$ must have the following shape:

$$\frac{\begin{array}{cc}\rho : s[\overline{x}/\overline{n}] \Downarrow^{\alpha}_{\mathsf{rf}} z & \nu : q[\overline{x}/\overline{n}] \Downarrow^{\gamma}_{\mathsf{rf}} f \\ \mu : r[\overline{x}/\overline{n}] \Downarrow^{\beta}_{\mathsf{rf}} e & \sigma : u[\overline{x}/\overline{n}] \Downarrow^{\delta}_{\mathsf{rf}} g\end{array}}{t[\overline{x}/\overline{n}] \Downarrow^{\alpha\beta\gamma\delta}_{\mathsf{rf}} (\mathtt{case}_A\ z\ \mathtt{zero}\ e\ \mathtt{even}\ f\ \mathtt{odd}\ g)}$$

  Let us now define p_t and q_t as follows:

$$p_t(x) = p_s(x) + p_r(x) + p_q(x) + p_u(x) + 1;$$
$$q_t(x) = q_s(x) + q_r(x) + q_q(x) + q_u(x) + 1.$$

  We have:

$$\begin{aligned}|\pi| &\leq |\rho| + |\mu| + |\nu| + |\sigma| + 1 \\ &\leq p_s(\sum_i |n_i|) + p_r(\sum_i |n_i|) + p_q(\sum_i |n_i|) + p_u(\sum_i |n_i|) + 1 \\ &= p_t(\sum_i |n_i|).\end{aligned}$$

  Similarly, if $z \in \pi$, it is easy to prove that $|z| \leq q_z(\sum_i |n_i|)$.

- If the last rule was (T-Rec),m We consider the most interesting case, where the first term computes to a value greater than 0. Suppose $t = (\texttt{recursion}_A\ s\, r\, q)$. By looking at the typing rule (figure 4) for (T-Rec) we are sure to be able to apply induction hypothesis on s, r, q. Definition of $\Downarrow_{\mathsf{rf}}$ ensure also that any derivation for $t[\overline{x}/\overline{n}]$ must have the following shape:

$$\frac{\begin{array}{c}\rho : s[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\alpha} z \quad \mu : z[\overline{x}/\overline{n}] \Downarrow_{\mathsf{nf}}^{\beta} n \\ \nu : r[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\gamma} e \\ \varrho_0 : qy[\overline{x}, y/\overline{n}, \lfloor \frac{n}{2^0} \rfloor] \Downarrow_{\mathsf{rf}}^{\gamma_0} q_0 \\ \cdots \\ \varrho_{|n|-1} : qy[\overline{x}, y/\overline{n}, \lfloor \frac{n}{2^{|n|-1}} \rfloor] \Downarrow_{\mathsf{rf}}^{\gamma_{|n|-1}} q_{|n|-1}\end{array}}{(\texttt{recursion}_A\ s\, r\, q)[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\alpha\beta\gamma(\prod_j \gamma_j)} q_0(\ldots(q_{(|n|-1)}e)\ldots)}$$

  Notice that we are able to apply $\Downarrow_{\mathsf{nf}}$ on term z because, by definition, s has only free variables of type $\Box\mathbf{N}$ (see figure 4). So, we are sure that z is a closed term of type $\mathbf{N}$ and we are able to apply the $\Downarrow_{\mathsf{nf}}$ algorithm. Let define p_t and q_t as follows:

$$\begin{aligned} p_t(x) &= p_s(x) + 2 \cdot q_s(x) + p_r(x) + 2 \cdot q_s(x)^2 \cdot p_q(x + 2 \cdot q_s(x)^2) + 1; \\ q_t(x) &= q_s(x) + q_r(x) + 2 \cdot q_s(x)^2 + q_q(x + 2 \cdot q_s(x)^2). \end{aligned}$$

  Notice that $|z|$ is bounded by $q_s(x)$. Notice that by applying theorem 3.2 on μ (z has no free variables) we have that every $v \in \mu$ is such that$v \leq 2 \cdot |z|^2$. We have:

$$\begin{aligned} |\pi| &\leq |\rho| + |\mu| + |\nu| + \sum_i (|\varrho_i|) + 1 \\ &\leq p_s(\sum_i |n_i|) + 2 \cdot |z| + p_r(\sum_i |n_i|) + |n| \cdot p_{qy}(\sum_i |n_i| + |n|) + 1 \\ &\leq p_s(\sum_i |n_i|) + 2 \cdot q_s(\sum_i |n_i|) + p_r(\sum_i |n_i|) + \\ &\quad + 2 \cdot q_s(\sum_i |n_i|)^2 \cdot p_{qy}(\sum_i |n_i| + 2 \cdot q_s(\sum_i |n_i|)^2) + 1. \end{aligned}$$

  Similarly, for every $w \in \pi$:

$$\begin{aligned} |w| &\leq q_s(\sum_i |n_i|) + 2 \cdot q_s(\sum_i |n_i|)^2 + q_r(\sum_i |n_i|) + q_{qy}(\sum_i |n_i| + |n|) \\ &\leq q_s(\sum_i |n_i|) + 2 \cdot q_z(\sum_i |n_i|)^2 + q_r(\sum_i |n_i|) + q_{qy}(\sum_i |n_i| + 2 \cdot q_s(\sum_i |n_i|)^2). \end{aligned}$$

- In this and the following cases the last rule is (T-Arr-E). Let's first consider $t = x\overline{s}$. In this case, obviously, the free variable x has type $\blacksquare A_i$ $(1 \leq i \leq j)$. By definition x is negatively $\Box$-free. This means that every term in $\overline{s}$ has a type that is positively $\Box$-free. By knowing that the type of x is negatively $\Box$-free, we conclude that the type of our

term t is $\Box$-free (because is both negatively and positively $\Box$-free at the same time). Definition of $\Downarrow_{\mathsf{rf}}$ ensures us that the derivation will have the following shape:

$$\frac{\rho_i : s_j[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\alpha_j} r_j}{x\overline{s}[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\prod_i \alpha_i} x\overline{r}}$$

We define p_t and q_t as:

$$p_t(x) = \sum_j p_{s_j}(x) + 1;$$
$$q_t(x) = \sum_j q_{s_j}(x) + 1.$$

Indeed we have

$$|\pi| \leq \sum_j |\rho_j| + 1 \leq \sum_j \{p_{s_j}(\sum_i |n_i|)\} + 1.$$

Similarly, if $z \in \pi$, it is easy to prove that $|z| \leq q_z(\sum_i |n_i|)$.

- If $t = \mathsf{S}_0 s$, then s have type $\mathbf{N}$ in the context Γ. The derivation π has the following form

$$\frac{\rho : s[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\alpha} z}{\mathsf{S}_0 s[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\alpha} \mathsf{S}_0 z}$$

Define $p_t(x) = p_s(x) + 1$ and $q_t(x) = q_s(x) + 1$. One can easily check that, by induction hypothesis

$$|\pi| \leq |\rho| + 1 \leq p_s(\sum_i |n_i|) + 1 = p_t(\sum_i |n_i|).$$

Analogously, if $r \in \pi$ then

$$|s| \leq q_s(\sum_i |n_i|) + 1 \leq q_t(\sum_i |n_i|).$$

- If $t = \mathsf{S}_1 s$ or $t = \mathtt{P}s$, then we can proceed exactly as in the previous case.
- Cases where we have on the left side a case or a recursion with some arguments are trivial, since they can be brought back to cases that we have considered.
- If t is $(\lambda x : \Box\mathbf{N}.s)r\overline{q}$, then we have the following derivation:

$$\frac{\begin{array}{ll}\rho : r[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\alpha} e & \\ \mu : e[\overline{x}/\overline{n}] \Downarrow_{\mathsf{nf}}^{\gamma} n & \nu : (s[x/n])\overline{q}[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\beta} v\end{array}}{(\lambda x : \Box\mathbf{N}.s)r\overline{q}[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\alpha\gamma\beta} v}$$

By hypothesis t is positively $\Box$-free and so also r (whose type is $\mathbf{N}$) and $s\overline{q}$ are positively $\Box$-free. So, we are sure that we are able to use induction hypothesis. Let p_t and q_t be:

$$p_t(x) = p_r(x) + 2 \cdot q_r(x) + p_{s\overline{q}}(x + 2 \cdot q_r(x)^2) + 1;$$
$$q_t(x) = q_{s\overline{q}}(x + 2 \cdot q_r(x)^2) + q_r(x) + 2 \cdot q_r(x)^2 + 1.$$

We have:

$$\begin{aligned}|\pi| &= |\rho| + |\mu| + |\nu| + 1\\ &\leq p_r(\sum_i |n_i|) + 2 \cdot |e| + p_{s\overline{q}}(\sum_i |n_i| + |n|) + 1\\ &\leq p_r(\sum_i |n_i|) + 2 \cdot q_r(\sum_i |n_i|) + p_{s\overline{q}}(\sum_i |n_i| + 2 \cdot q_r(\sum_i |n_i|)^2) + 1.\end{aligned}$$

By construction, remember that s has no free variables of type $\blacksquare\mathbf{N}$. By Theorem 3.2 (z has no free variables) we have $v \in \mu$ is such that $|v| \leq 2 \cdot |e|^2$. By applying induction hypothesis we have that every $v \in \rho$ is such that $|v| \leq q_r(\sum_i |n_i|)$, every $v \in \nu$ is such that

$$\begin{aligned}|v| &\leq q_{s\overline{q}}(\sum_i |n_i| + |n|) \leq q_{s\overline{q}}(\sum_i |n_i| + 2 \cdot |e|^2)\\ &\leq q_{s\overline{q}}(\sum_i |n_i| + 2 \cdot q_r(\sum_i |n_i|)^2).\end{aligned}$$

We can prove the second point of our thesis by setting $q_t(\sum_i |n_i|)$ as $q_{s\overline{q}}(\sum_i |n_i| + 2 \cdot q_r(\sum_i |n_i|)^2) + q_r(\sum_i |n_i|) + 2 \cdot q_r(\sum_i |n_i|)^2 + 1$.

- If t is $(\lambda x : \blacksquare\mathbf{N}.s)r\overline{q}$, then we have the following derivation:

$$\frac{\begin{array}{c}\rho : r[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\alpha} e\\ \mu : e[\overline{x}/\overline{n}] \Downarrow_{\mathsf{nf}}^{\gamma} n \qquad \nu : s\overline{q}[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\beta} u\end{array}}{(\lambda x : \blacksquare\mathbf{N}.s)r\overline{q}[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\alpha\gamma\beta} (\lambda x : \blacksquare\mathbf{N}.u)n}$$

By hypothesis we have t that is positively $\Box$-free. So, also r and e (whose type is $\mathbf{N}$) and $s\overline{q}$ are positively $\Box$-free. We define p_t and q_t as:

$$\begin{aligned}p_t(x) &= p_r(x) + 2 \cdot q_r(x) + p_{s\overline{q}}(x) + 1;\\ q_t(x) &= q_r(x) + 2 \cdot q_r(x)^2 + q_{s\overline{q}}(x) + 1.\end{aligned}$$

We have:

$$\begin{aligned}|\pi| &= |\rho| + |\mu| + |\nu| + 1\\ &\leq p_r(\sum_i |n_i|) + 2 \cdot q_r(\sum_i |n_i|) + p_{s\overline{q}}(\sum_i |n_i|) + 1.\end{aligned}$$

Similarly, if $z \in \pi$, it is easy to prove that $|z| \leq q_t(\sum_i |n_i|)$.

- If t is $(\lambda x : aH.s)r\overline{q}$, then we have the following derivation:

$$\frac{\rho : (s[x/r])\overline{q}[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\beta} v}{(\lambda x : aH.s)r\overline{q}[\overline{x}/\overline{n}] \Downarrow_{\mathsf{rf}}^{\beta} v}$$

By hypothesis we have t that is positively $\Box$-free. So, also $s\overline{q}$ is positively $\Box$-free. r has an higher-order type H and so we are sure that $|(s[x/r])\overline{q}| < |(\lambda x : aH.s)r\overline{q}|$. Define p_t and q_t as:

$$p_t(x) = p_{(s[x/r])\overline{q}}(x) + 1;$$
$$q_t(x) = q_{(s[x/r])\overline{q}}(x) + 1.$$

By applying induction hypothesis we have:

$$|\pi| = |\rho| + 1 \leq p_{(s[x/r])\overline{q}}(\sum_i |n_i|) + 1.$$

By induction, we are able to prove the second point of our thesis.
This concludes the proof. $\Box$

Following the definition of $\Downarrow$, it is quite easy to obtain, given a first order term t, of arity k, a probabilistic Turing machine that, when receiving on input (an encoding of) $n_1 \ldots n_k$, produces in output m with probability equal to $\mathscr{D}(m)$, where $\mathscr{D}$ is the (unique!) distribution such that $t \rightsquigarrow \mathscr{D}$. Indeed, $\Downarrow_{\mathsf{rf}}$ and $\Downarrow_{\mathsf{nf}}$ are designed as algorithms. Moreover, the obtained Turing machine works in polynomial time, due to propositions 3.2 and 3.3. Formally:

Theorem 3.4 (Soundness)**.** *Suppose t is a first order term of arity k. Then there is a probabilistic Turing machine M_t running in polynomial time such that M_t on input $n_1 \ldots n_k$ returns m with probability exactly $\mathscr{D}(m)$, where $\mathscr{D}$ is a probability distribution such that $tn_1 \ldots n_k \rightsquigarrow \mathscr{D}$.*

Proof. By propositions 3.2 and 3.3. $\Box$

4. Probabilistic Polytime Completeness

In the previous section, we proved that the behaviour of any RSLR first-order term can be somehow simulated by a probabilistic polytime Turing machine. What about the converse? In this section, we prove that any probabilistic polynomial time Turing machine (PPTM in the following) can be encoded in RSLR. PPTMs are here seen as one-tape Turing machines which are capable at any step during the computation of "tossing a fair coin", and proceeding in two different ways depending on the outcome of the tossing.

To facilitate the encoding, we extend our system with pairs. All the proofs in previous sections remain valid. Types are extended by the following production:

$$A \ ::= \ A \otimes A;$$

Terms now contain two binary constructs $\langle t, s\rangle$ and $\mathtt{let}_A\ t\ \mathtt{be}\ \langle x, y\rangle\ \mathtt{in}\ r$

$$\frac{\Gamma; \Delta_1 \vdash t : A \qquad \Gamma; \Delta_2 \vdash s : B}{\Gamma; \Delta_1, \Delta_2 \vdash \langle t, s\rangle : A \otimes B}$$

$$\frac{\Gamma;\Delta_1 \vdash t : A \otimes B \qquad \Gamma; x : aA, y : aB, \Delta_2 \vdash s : C \qquad \Gamma, \Delta_1 <: a}{\Gamma;\Delta_1,\Delta_2 \vdash \mathtt{let}_C\ t\ \mathtt{be}\ \langle x, y\rangle\ \mathtt{in}\ s : C}$$

We will never make use of the `let` construct, except through projections, which can be easily defined and which have the expected types:

$$\frac{\Gamma \vdash t : A \otimes B}{\Gamma \vdash \pi_1(t) : A} \qquad \frac{\Gamma \vdash t : A \otimes B}{\Gamma \vdash \pi_2(t) : B}$$

As syntactic sugar, we will use $\langle t_1 \ldots, t_i\rangle$ (where $i \geq 1$) for the term

$$\langle t_1, \langle t_2, \ldots \langle t_{i-1}, t_i\rangle \ldots\rangle\rangle.$$

For every $n \geq 1$ and every $1 \leq i \leq n$, we can easily build a term π_i^n which extracts the i-th component from tuples of n elements: this can be done by composing $\pi_1(\cdot)$ and $\pi_2(\cdot)$. With a slight abuse on notation, we sometime write π_i for π_i^n.

4.1. Unary Natural Numbers and Polynomials

Natural numbers are represented in binary in RSLR. In other words, the basic operations allowed on them are $\mathtt{S}_0$, $\mathtt{S}_1$ and $\mathtt{P}$, which correspond to appending a binary digit to the right of the number (seen as a binary string) or stripping the rightmost such digit. This is even clearer if we consider the length $|n|$ of a numeral n, which is only logarithmic in n.

Sometimes, however, it is more convenient to work in unary notation. Given a natural number i, its *unary encoding* is simply the numeral that, written in binary notation, is 1^i. Given a natural number i we will refer to its unary encoding $\underline{i}$. The type in which unary natural numbers will be written, is just $\mathbf{N}$, but for reasons of clarity we will use the symbol $\mathbf{U}$ instead.

From any numeral n, we can extract the unary encoding of its length:

$$\mathsf{encode} = \lambda t : \square\mathbf{N}.\mathtt{recursion}_{\mathbf{U}}\ t\, 0\, (\lambda x : \square\mathbf{U}.\lambda y : \blacksquare\mathbf{U}.\mathtt{S}_1 y) : \square\mathbf{N} \to \mathbf{U}$$

Predecessor and successor functions are simply $\mathtt{P}$ and $\mathtt{S}_1$. We need to show how to express polynomials, and in order to do so we define the operators $\mathsf{add} : \square\mathbf{U} \to \blacksquare\mathbf{U} \to \mathbf{U}$ and $\mathsf{mult} : \square\mathbf{U} \to \square\mathbf{U} \to \mathbf{U}$. We define add as

$$\begin{aligned}\mathsf{add} =& \lambda x : \square\mathbf{U}.\lambda y : \blacksquare\mathbf{U}.\\ & \mathtt{recursion}_{\mathbf{U}}\ x\, y\, (\lambda x : \square\mathbf{U}.\lambda y : \blacksquare\mathbf{U}.\mathtt{S}_1 y) : \square\mathbf{U} \to \blacksquare\mathbf{U} \to \mathbf{U}.\end{aligned}$$

Similarly, we define mult as

$$\begin{aligned}\mathsf{mult} =& \lambda x : \square\mathbf{U}.\lambda y : \square\mathbf{U}.\\ & \mathtt{recursion}_{\mathbf{U}}\ (\mathtt{P}x)\, y\, (\lambda x : \square\mathbf{U}.\lambda z : \blacksquare\mathbf{U}.\mathsf{add} y z) : \square\mathbf{U} \to \square\mathbf{U} \to \mathbf{U}.\end{aligned}$$

The following is quite easy:

Lemma 4.1. *Every polynomial of one variable with natural coefficients can be encoded as a term of type* $\square\mathbf{U} \to \mathbf{U}$.

Proof. Simply, turn add into a term of type $\square\mathbf{U} \to \square\mathbf{U} \to \mathbf{U}$ by way of subtyping and then compose add and mult has much as needed to encode the polynomial at hand. □

4.2. Finite Sets

Any finite, linearly ordered set $F = (|F|, \sqsubseteq_F)$ can be naturally encoded as an "initial segment" of $\mathbf{N}$: if $|F| = \{a_0, \ldots, a_i\}$ where $a_i \sqsubseteq_F a_j$ whenever $i \leq j$, then a_i is encoded simply by the natural number whose binary representation is 10^i. For reasons of clarity, we will denote $\mathbf{N}$ as $\mathbf{F}_F$. We can do some case analysis on an element of $\mathbf{F}_F$ by the combinator

$$\mathsf{switch}_A^F : \blacksquare\mathbf{F}_F \to \underbrace{\blacksquare A \to \ldots \to \blacksquare A}_{i \text{ times}} \to \blacksquare A \to A,$$

where A is a $\square$-free type and i is the cardinality of $|F|$. The term above can be defined by induction on i:

- If $i = 0$, then it is simply $\lambda x : \blacksquare\mathbf{F}_F.\lambda y : \blacksquare A.y$.
- If $i \geq 1$, then it is the following:

$$\begin{array}{l}\lambda x : \blacksquare\mathbf{F}_F.\lambda y_0 : \blacksquare A. \ldots \lambda y_i : \blacksquare A. \\ \quad \mathtt{case}_A\ x\ \mathtt{zero}(\lambda h : \blacksquare A.h) \\ \qquad \mathtt{even}\ (\lambda h : \blacksquare A.\mathsf{switch}_A^E(\mathsf{P}x)y_1 \ldots y_i h) \\ \qquad \mathtt{odd}\ (\lambda h : \blacksquare A.y_0)\end{array}$$

where E is the subset of F of those elements with strictly positive indices.

4.3. Strings

Suppose $\Sigma = \{a_0, \ldots, a_i\}$ is a finite alphabet. Elements of Σ can be encoded following the just described scheme, but how about strings in Σ^*? We can proceed somehow similarly: the string $a_{j_1} \ldots a_{j_k}$ can be encoded as the natural number

$$10^{j_1}10^{j_2} \ldots 10^{j_k}.$$

Whenever we want to emphasize that a natural number is used as a string, we write $\mathbf{S}_\Sigma$ instead of $\mathbf{N}$. It is easy to build a term $\mathsf{append}_\Sigma : \blacksquare(\mathbf{S}_\Sigma \otimes \mathbf{F}_\Sigma) \to \mathbf{S}_\Sigma$ which appends the second argument to the first argument. Similarly, one can define a term $\mathsf{tail}_\Sigma : \blacksquare\mathbf{S}_\Sigma \to \mathbf{S}_\Sigma \otimes \mathbf{F}_\Sigma$ which strips off the rightmost character a from the argument string and returns a together with the rest of the string; if the string is empty, a_0 is returned, by convention.

We also define a function $\mathsf{NtoS}_\Sigma : \square\mathbf{N} \to \mathbf{S}_\Sigma$ that takes a natural number and produce in output an encoding of the corresponding string in Σ^* (where i_0 and i_1 are the indices of 0 and 1 in Σ):

$$\begin{array}{l}\mathsf{NtoS}_\Sigma = \lambda x : \square\mathbf{N}.\mathtt{recursions}_{\mathbf{S}_\Sigma}\ x\ 1 \\ \quad \lambda x : \blacksquare\mathbf{N}.\lambda y : \blacksquare\mathbf{S}. \\ \qquad \mathtt{case}_{\mathbf{N}}\ x\ \mathtt{zero}\ \mathsf{append}_\Sigma\langle y, i_0\rangle \\ \qquad\quad \mathtt{even}\ \mathsf{append}_\Sigma\langle y, i_1\rangle \\ \qquad\quad \mathtt{odd}\ \mathsf{append}_\Sigma\langle y, i_1\rangle : \square\mathbf{N} \to \mathbf{S}.\end{array}$$

Similarly, one can write a term $\mathsf{StoN}_\Sigma : \square\mathbf{S}_\Sigma \to \mathbf{N}$.

4.4. Probabilistic Turing Machines

Let M be a probabilistic Turing machine $M = (Q, q_0, F, \Sigma, \sqcup, \delta)$, where Q is the finite set of states of the machine; q_0 is the initial state; F is the set of final states of M; Σ is the finite alphabet of the tape; $\sqcup \in \Sigma$ is the symbol for empty string; $\delta \subseteq (Q \times \Sigma) \times (Q \times \Sigma \times \{\leftarrow, \downarrow, \rightarrow\})$ is the transition function of M. For each pair $(q, s) \in Q \times \Sigma$, there are exactly two triples (r_1, t_1, d_1) and (r_2, t_2, d_2) such that $((q, s), (r_1, t_1, d_1)) \in \delta$ and $((q, s), (r_2, t_2, d_2)) \in \delta$. Configurations of M can be encoded as follows:

$$\langle t_{\mathit{left}}, t, t_{\mathit{right}}, s\rangle : \mathbf{S}_\Sigma \otimes \mathbf{F}_\Sigma \otimes \mathbf{S}_\Sigma \otimes \mathbf{F}_Q,$$

where t_{left} represents the left part of the main tape, t is the symbol read from the head of M, t_{right} the right part of the main tape; s is the state of our Turing Machine. Let $\mathbf{C}_M$ be a shortcut for $\mathbf{S}_\Sigma \otimes \mathbf{F}_\Sigma \otimes \mathbf{S}_\Sigma \otimes \mathbf{F}_Q$. Let t_C be the term encoding the configuration C.

Suppose that M on input x runs in time bounded by a polynomial $p : \mathbb{N} \to \mathbb{N}$. Then we can proceed as follows:

- encode the polynomial p by using function $\mathsf{encode}, \mathsf{add}, \mathsf{mult}, \mathsf{dec}$ so that at the end we will have a function $\underline{p} : \Box\mathbf{N} \to \mathbf{U}$;
- write a term $\underline{\delta} : \blacksquare\mathbf{C}_M \to \mathbf{C}_M$ which mimicks δ;
- write a term $\mathsf{init}_M : \blacksquare\mathbf{S}_\Sigma \to \mathbf{C}_M$ which returns the initial configuration for M corresponding to the input string.

The term t_M of type $\Box\mathbf{N} \to \mathbf{N}$ which has exactly the same behavior as M is the following:

$$\lambda x : \Box\mathbf{N}.\mathsf{StoN}_\Sigma(\mathtt{recursion}_{\mathbf{C}_M}\ (\underline{p}\ x)\ (\mathsf{init}_M\ (\mathsf{NtoS}_\Sigma(x)))\ (\lambda y : \blacksquare\mathbf{N}.\lambda z : \blacksquare\mathbf{C}_M.\underline{\delta}\ z)).$$

We then get a faithful encoding of PPTM into RSLR, which will be useful in the forthcoming section:

Theorem 4.2. *Suppose M is a probabilistic Turing machine running in polynomial time such that for every n, $\mathscr{D}_n$ is the distribution of possible results obtained by running M on input n. Then there is a first order term t_M such that for every n, tn evaluates to $\mathscr{D}_n$.*

Proof. Proving that t_M does what it is supposed to do can be done as follows:

- First of all, one can show that $\underline{p}\ n$ evaluates to $\mathscr{I}_m$, where m is the unary encoding of the length of $p(n)$;
- Similarly, one can show that init_M correctly computes an initial configuration for M when fed with an input string;
- Finally, $\underline{\delta}$ can be showed to evaluate to $\{t_D^{\frac{1}{2}}, t_E^{\frac{1}{2}}\}$ whenever fed with t_C and whenever C is a configuration which evolves in one step to D and E.

We elide the proof here, since all the steps above are quite simple anyway. □

5. Relations with Complexity Classes

The last two sections established a precise correspondence between RSLR and probabilistic polynomial time Turing machines. But how about probabilistic complexity *classes*,

like **BPP** or **PP**? They are defined on top of probabilistic Turing machines, imposing constraints on the probability of error: in the case of **PP**, the error probability can be anywhere near $\frac{1}{2}$, but not equal to it, while in **BPP** it can be non-negligibly smaller than $\frac{1}{2}$. There are two ways RSLR can be put in correspondence with probabilistic complexity classes, and these are explained in the following two sections.

5.1. Leaving the Error Probability Explicit

Of course, one possibility consists in leaving bounds on the error probability explicit *in the very definition* of what an RSLR term represents:

Definition 5.1 (Recognising a Language with Error ε)**.** A first-order term t of arity 1 recognizes a language $L \subseteq \mathbb{N}$ with probability less than ε if, and only if, both:
- $x \in L$ and $tx \rightsquigarrow \mathscr{D}$ implies $\mathscr{D}(0) > 1 - \varepsilon$;
- $x \notin L$ and $tx \rightsquigarrow \mathscr{D}$ implies $\sum_{s>0} \mathscr{D}(s) > 1 - \varepsilon$.

So, 0 encodes an accepting state of tx and $s > 0$ encodes a reject state of tx. Theorem 3.4, together with Theorem 4.2 allows us to conclude that:

Theorem 5.1 ($\frac{1}{2}$-Completeness for **PP**)**.** *The set of languages which can be recognized with error ε in* RSLR *for some $0 < \varepsilon \leq 1/2$ equals* **PP***.*

But, interestingly, we can go beyond and capture a more interesting complexity class:

Theorem 5.2 ($\frac{1}{2}$-Completeness for **BPP**)**.** *The set of languages which can be recognized with error ε in* RSLR *for some $0 < \varepsilon < 1/2$ equals* **BPP***.*

Observe how ε can be even equal to $\frac{1}{2}$ in Theorem 5.1, while it cannot in Theorem 5.2. This is the main difference between **PP** and **BPP**: in the first class, the error probability can very fast approach $\frac{1}{2}$ when the size of the input grows, while in the second it cannot.

The notion of recognizing a language with an error ε allows to capture complexity classes in RSLR, but it has an obvious drawback: the error probability remains *explicit* and *external* to the system; in other words, RSLR does not characterize *one* complexity class but many, depending on the allowed values for ε. Moreover, given an RSLR term t and an error ε, determining whether t recognizes *any* function with error ε is undecidable. As a consequence, theorems 5.1 and 5.2 do not suggest an enumeration of all languages in either **PP** or **BPP**. This in contrast to what happens with other ICC systems, e.g. SLR, in which all terms (of certain types) compute a function in **FP** (and, *viceversa*, all functions in **FP** are computed this way). As we have already mentioned in the Introduction, this discrepancy between **FP** and **BPP** has a name: the former is a *syntactic* class, while the latter is a *semantic* class (see [1]).

5.2. Getting Rid of the Error Probability

One may wonder whether a more implicit notion of representation can be somehow introduced, and which complexity class corresponds to RSLR this way. One possibility is taking representability by majority:

Definition 5.2 (Representability-by-Majority)**.** Let t be a first-order term of arity 1. Then t is said to *represent-by-majority* a language $L \subseteq \mathbb{N}$ iff:

1. If $n \in L$ and $tn \rightsquigarrow \mathscr{D}$, then $\mathscr{D}(0) \geq \sum_{m>0} \mathscr{D}(m)$;
2. If $n \notin L$ and $tn \rightsquigarrow \mathscr{D}$, then $\sum_{m>0} \mathscr{D}(m) > \mathscr{D}(0)$.

There is a striking difference between Definition 5.2 and Definition 5.1: the latter is asymmetric, while the first is symmetric.

Please observe that any RSLR first order term t represents-by-majority *a* language, namely the language defined from t by Definition 5.2. It is well known that **PP** can be defined by majority itself [1], stipulating that the error probability should be *at most* $\frac{1}{2}$ when handling strings in the language and *strictly smaller than* $\frac{1}{2}$ when handling strings not in the language. As a consequence:

Theorem 5.3 (Completeness-by-Majority for **PP**)**.** *The set of languages which can be represented-by-majority in* RSLR *equals* **PP**.

In other words, RSLR can indeed be considered as a tool to enumerate all functions in a complexity class, namely **PP**. It comes with no surprise, since the latter is a syntactic class.

References

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity — A Modern Approach*. Cambridge University Press, 2009.

[2] S.J. Bellantoni, K.H. Niggl, and H. Schwichtenberg. Higher type recursion, ramification and polynomial time. *Annals of Pure and Applied Logic*, 104(1-3):17–30, 2000.

[3] Stephen Bellantoni. Predicative recursion and the polytime hierarchy. In P. Clote and J.B. Remmel, editors, *Feasible Mathematics II*, pages 15–29. Birkhauser, 1995.

[4] Stephen Bellantoni and Stephen A. Cook. A new recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, 1992.

[5] Guillaume Bonfante, Reinhard Kahle, Jean-Yves Marion, and Isabel Oitavem. Recursion schemata for NC^k. In Michael Kaminski and Simone Martini, editors, *Computer Science Logic, 22nd International Workshop, Proceedings*, volume 5213 of *LNCS*, pages 49–63, 2008.

[6] Ugo Dal Lago, Simone Martini, and Davide Sangiorgi. Light logics and higher-order processes. In Sibylle B. Fröschle and Frank D. Valencia, editors, *17th International Workshop on Expressiveness in Concurrency, Proceedings*, volume 41 of *EPTCS*, 2010.

[7] Ugo Dal Lago, Andrea Masini, and Margherita Zorzi. Quantum implicit computational complexity. *Theoretical Computer Science*, 411(2):377–409, 2010.

[8] Lance Fortnow and Rahul Santhanam. Hierarchy theorems for probabilistic polynomial time. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 316–324, Washington, DC, USA, 2004. IEEE Computer Society.

[9] John Gill. Computational complexity of probabilistic turing machines. *SIAM J. Comput.*, 6(4):675–695, 1977.

[10] Martin Hofmann. A mixed modal/linear lambda calculus with applications to Bellantoni-Cook safe recursion. In Mogens Nielsen and Wolfgang Thomas, editors, *Computer Science Logic, 11th International Workshop, Proceedings*, volume 1414 of *LNCS*, pages 275–294, 1997.

[11] Neil D. Jones. LOGSPACE and PTIME characterized by programming languages. *Theoretical Computer Science*, 228:151–174, 1999.

[12] Daniel Leivant. Stratified functional programs and computational complexity. In *Principles of Programming Languages, 20th International Symposium, Proceedings*, pages 325–333. ACM, 1993.

[13] Daniel Leivant and Jean-Yves Marion. Ramified recurrence and computational complexity II: Substitution and poly-space. In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic, 9th International Workshop, Proceedings*, volume 933 of *LNCS*, pages 486–500. 1995.

[14] John C. Mitchell, Mark Mitchell, and Andre Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *Foundations of Computer Science, 39th Annual Symposium, Proceedings*, pages 725–733. IEEE Computer Society, 1998.

[15] Helmut Schwichtenberg and Steven Bellantoni. Feasible computation with higher types. In *Proof and System-Reliability*, pages 399–415. Kluwer Academic Publisher, 2001.

[16] Yu Zhang. The computational SLR: a logic for reasoning about computational indistinguishability. *Mathematical Structures in Computer Science*, 20(5):951–975, 2010.

Name-passing calculi: from fusions to preorders and types

Daniel Hirschkoff, Jean-Marie Madiot
ENS Lyon, U. de Lyon, CNRS, INRIA, UCBL
{daniel.hirschkoff, jeanmarie.madiot}@ens-lyon.fr

Davide Sangiorgi
University of Bologna and INRIA
davide.sangiorgi@cs.unibo.it

***Abstract*—The fusion calculi are a simplification of the pi-calculus in which input and output are symmetric and restriction is the only binder. We highlight a major difference between these calculi and the pi-calculus from the point of view of types, proving some impossibility results for subtyping in fusion calculi. We propose a modification of fusion calculi in which the name equivalences produced by fusions are replaced by name preorders, and with a distinction between positive and negative occurrences of names. The resulting calculus allows us to import subtype systems, and related results, from the pi-calculus. We examine the consequences of the modification on behavioural equivalence (e.g., context-free characterisations of barbed congruence) and expressiveness (e.g., full abstraction of the embedding of the asynchronous pi-calculus).**

***Index Terms*—process calculus; fusions; types; subtyping;**

I. Introduction

The π-calculus is the paradigmatical name-passing calculus, that is, a calculus where names (a synonym for "channels") may be passed around. Key aspects for the success of the π-calculus are the minimality of its syntax and its expressiveness. Expressiveness comes at a price: often, desirable behavioural properties, or algebraic laws, fail. The reason is that, when employing π-calculus to describe a system, one normally follows a discipline that governs how names can be used. The discipline can be made explicit by means of *types*. Types bring in other benefits, notably the possibility of statically detecting many programming errors. Types are indeed a fundamental aspect of the π-calculus theory, and one of the most important differences between name-passing calculi and process calculi such as CCS in which names may not be passed.

One of the basic elements in type systems for name-passing calculi is the possibility of separating the capabilities for actions associated to a name, e.g., the capability of using a name in input or in output. The control of capabilities has behavioural consequences because it allows one to express constraints on the use of names. For a simple example, consider a process P that implements two distinct services $\mathtt{A}$ and $\mathtt{B}$, accessible using channels a and b that must be communicated to clients of the services. We assume here only two clients, that receive the channels via c_1 and c_2:

$$P \stackrel{\text{def}}{=} (\boldsymbol{\nu} a, b)\left(\overline{c_1}\langle a, b\rangle.\,\overline{c_2}\langle a, b\rangle.\,(\mathtt{A} \mid \mathtt{B})\right) \qquad (1)$$

We expect that outputs at a or b from the clients are eventually received and processed by the appropriate service. But this is not necessarily the case: a malign client can disrupt the expected protocol by simply offering an input at a or b and then throwing away the values received, or forwarding the values to the wrong service. These misbehaviours are ruled out by a capability type system imposing that the clients only obtain the output capability on the names a and b when receiving them from c_1 and c_2. The typing rules are straightforward, and mimic those for the typing of references in imperative languages with subtyping.

Capabilities [1] are at the basis of more complex type systems, with a finer control on names. For instance, type systems imposing constraints on successive usages of the names like usage-based type systems and deadlock-detection systems, session types, and so on [2], [3], [4].

Capabilities are closely related to subtyping. In the example (1), P creates names a and b, and possesses both the input and the output capabilities on them; it however transmits to the clients only a subset of the capabilities (namely the output capability alone). The subset relation on capabilities gives rise to a subtype relation on types. All forms of subtyping for π-calculus or related calculi in the literature require a discipline on capabilities. Subtyping can also be used to recover well-known forms of subtyping in other computational paradigms, e.g., functional languages or object-oriented languages, when an encoding of terms into processes is enhanced with an encoding of types [5].

An interesting family of variants of the π-calculus are — what we call here — the *fusion calculi*: Fusion [6], Update [7], Explicit Fusions [8], Chi [9], Solos [10]. Their beauty is the simplification achieved, with binding removed from the input construct. Thus input prefixing becomes symmetric to output prefixing, and restriction remains as the only binder. The effect of a synchronisation between an output $\overline{a}b.\,P$ and an input $ac.\,Q$ is to fuse the two object names b and c, which are now interchangeable. Thus communications produce, step-by-step, an equivalence relation on names. Different fusion-like calculi differ in the way the name equivalence is handled. The operational theories of these calculi have been widely studied, e.g. [6], [11], [12], [13], [14].

As for the π-calculus (sometimes abbreviated as π in the sequel), however, the expressiveness of fusion calculi makes desirable behavioural properties fail. The same examples for the π-calculus can be used. For instance, the problems of misbehaving clients of the services of (1) remain. Actually, in fusion calculi additional problems arise; for example a client

receiving the two channels a and b along c_i could fuse them using an input $c_i\langle n, n\rangle.\, R$. Now a and b are indistinguishable, and an emission on one of them can reach any of the two services (and if a definition of a service is recursive, a recursive call could be redirected towards the other service).

In the paper we study the addition of types to fusion calculi; more generally, to single-binder calculi, where input is not binding (in fusion calculi, in addition, reductions fuse names). We begin by highlighting a striking difference between π-calculus and fusion calculi, proving some impossibility results for subtyping (and hence for general capability-based type systems, implicitly or explicitly involving subtyping). In the statement of the results, we assume a few basic properties of type systems for name-passing calculi, such as strengthening, weakening and type soundness, and the validity of the ordinary typing rules for the base operators of parallel composition and restriction. These results do not rule out completely the possibility of having subtyping or capabilities in fusion calculi, because of the few basic assumptions we make. They do show, however, that such type systems would have to be more complex than those for ordinary name-passing calculi such as the π-calculus, or require modifications or constraints in the syntax of the calculi.

Intuitively, the impossibility results arise because at the heart of the operational semantics for fusion calculi is an equivalence relation on names, generated through name fusions. In contrast, subtyping and capability systems are built on a preorder relation (be it subtyping, or set inclusion among subsets of capabilities). The equivalence on names forces one to have an equivalence also on types, instead of a preorder.

We propose a solution whose crux is the replacement of the equivalence on names by a preorder, and a distinction on occurrences of names, between 'positive' and 'negative'. In the resulting single-binder calculus, $\pi\mathtt{P}$ ('π with Preorder'), reductions generate a preorder. The basic reduction rule is

$$\overline{c}a.\, P \mid cb.\, Q \longrightarrow P \mid Q \mid a\!\!/\!b \ .$$

The particle $a\!\!/\!b$, called an *arc*, sets a to be above b in the name preorder. Such a process may redirect a prefix at b (which represents a 'positive' occurrence of b) to become a prefix at a. We show that the I/O (input/output) capability systems of the π-calculus can be reused in $\pi\mathtt{P}$, following a generalisation of the typing rules of the π-calculus that takes into account the negative and positive occurrences of names. A better understanding of type systems with subtyping in name-passing calculi is a by-product of this study. For instance, the study suggests that it is essential for subtyping that substitutions produced by communications (in $\pi\mathtt{P}$, the substitutions produced by arcs) only affect the positive occurrences of names.

The modification also brings in behavioural differences. For instance, both in the π-calculus and in $\pi\mathtt{P}$, a process that creates a new name a has the guarantee that a will remain different from all other known names, even if a is communicated to other processes (only the creator of a can break this, by using a in negative position). This is not true in fusion calculi, where the emission of a may produce fusions between a and other names. To demonstrate the proximity with the π-calculus we show that the embedding of the asynchronous π-calculus into $\pi\mathtt{P}$ is fully abstract (full abstraction of the encoding of the π-calculus into fusion calculi fails). We also exhibit an encoding of Explicit Fusions into $\pi\mathtt{P}$, where fusions become bi-directional arcs.

We present two possible semantics for $\pi\mathtt{P}$ that differ on the moment arcs enable substitutions. In the *eager* semantics, arcs may freely act on prefixes; in the *by-need* semantics, arcs act on prefixes only when interactions occur. We provide a characterisation of the reference contextual behavioural equivalence (barbed congruence) as a context-free labelled bisimilarity for the by-need semantics. We also compare and contrast the semantics, both between them and with semantics based on name fusion.

A property of certain fusion calculi (Fusion, Explicit Fusion) is a semantic duality induced by the symmetry between input and output prefixes. In $\pi\mathtt{P}$, the syntax still allows us to swap inputs and outputs, but in general the original and final processes have incomparable behaviours.

We conclude by examining the following syntactic constraint in single-binder calculi: each name, say b, may occur at most once in negative position (this corresponds to input object, as in $ab.\, P$, or to the source of an arc, as in $a\!\!/\!b$). Under this constraint, the two semantics for $\pi\mathtt{P}$, eager and by-need, coincide. In fusion calculi, the constraint allows us to import the π-calculus type systems. The constraint is however rather strong, and, in fusion calculi, breaks the semantic duality between inputs and outputs.

In summary, $\pi\mathtt{P}$, while being syntactically similar to fusion calculi, remains fairly close to the π-calculus (type systems, management of names).

Further related work: Central to $\pi\mathtt{P}$ is the preorder on names, that breaks the symmetry of name equivalence in fusion-like calculi. Another important ingredient for the theory of $\pi\mathtt{P}$ is the distinction between negative and positive occurrences of a name. In Update [7] and (asymmetric versions of) Chi [9], reductions produce ordinary substitutions on names. In practice, however, substitutions are not much different from fusions: a substitution $\{a/b\}$ fuses a with b *and* makes a the representative of the equivalence class. Still, substitutions are directed, and in this sense Update and Chi look closer to $\pi\mathtt{P}$ than the other fusion calculi. For instance Update and Chi, like $\pi\mathtt{P}$, lack the duality property on computations. Update was refined to the Fusion calculus [6] because of difficulties in the extension with polyadicity. Another major difference for Update and Chi with respect to $\pi\mathtt{P}$ is that in the former calculi substitutions replace all occurrences of names, whereas $\pi\mathtt{P}$ takes into account the distinction between positive and negative occurrences.

The question of controlling the fusion of private names has been addressed in [15], in the U-calculus. This calculus makes no distinction between input and output, and relies on two forms of binding to achieve a better control of scope extrusion, thus leading to a sensible behavioural theory that encompasses fusions and π. Thus the calculus is not single-binder. It is

unclear how capability types could be defined in it, as it does not have primitive constructs for input and output.

Paper outline: Section II gives some background. In Section III, we present some impossibility results on type systems for fusion-like calculi. Section IV introduces $\pi\mathtt{P}$ and its type system. The behavioural theory of $\pi\mathtt{P}$ is explored in Section V, and we give some expressiveness results in Section VI. Section VII studies a syntactical restriction that can be applied to $\pi\mathtt{P}$ and fusions, and we discuss future work in Section VIII.

II. Background on name-passing calculi

In this section we group terminology and notation that are common to all the calculi discussed in the paper. For simplicity of presentation, all calculi in the paper are finite. The addition of operators like replication for writing infinite behaviours goes as expected. The results in the paper would not be affected.

We informally call *name-passing* the calculi in the π-calculus tradition, which have the usual constructs of parallel composition and restriction, and in which computation is interaction between input and output constructs. *Names* identify the pairs of matching inputs/outputs, and the values transmitted may themselves be names. Restriction is a binder for the names; in some cases the input may be a binder too. Examples of these calculi are the π-calculus, the asynchronous π-calculus, the Join calculus, the Distributed π-calculus, the Fusion calculus, and so on. Binders support the usual alpha-conversion mechanism, and give rise to the usual definitions of free and bound names.

Convention 1. To simplify the presentation, throughout the paper, in all statements (including rules), we assume that the bound names of the entities in the statements are *different from each other and different from the free names* (*Barendregt convention on names*). Similarly, we say that a name is *fresh* or *fresh for a process*, if the name does not appear in the entities of the statements or in the process. □

We use $a, b, \ldots$ to range over names. In a free input $ab.\,P$, bound input $a(b).\,P$, output $\overline{a}b.\,P$, we call a the *subject* of the prefix, and b the *object*. We sometimes abbreviate prefixes as $a.\,P$ and $\overline{a}.\,P$ when the object carried is not important. We omit trailing $\mathbf{0}$, for instance writing $\overline{a}b$ in place of $\overline{a}b.\,\mathbf{0}$. We write $P\{a/b\}$ for the result of applying the substitution of b with a in P.

When restriction is the only binder (hence the input construct is not binding), we say that the calculus *has a single binder*. If in addition interaction involves fusion between names, so that we have ($\Longrightarrow$ stands for an arbitrary number of reduction steps, and in the right-hand side P and Q can be omitted if they are $\mathbf{0}$)

$$(\boldsymbol{\nu} c)\,(\overline{a}b.\,P \mid ac.\,Q \mid R) \Longrightarrow (P \mid Q \mid R)\{b/c\}\ , \qquad (2)$$

we say that the calculus *has name-fusions*, or, more briefly, has *fusions*. (We are not requiring that (2) is among the rules of the operational semantics of the calculus, just that (2) holds. The shape of (2) has been chosen so to capture the existing calculi; the presence of R allows us to capture also the Solos calculus.) All single-binder calculi in the literature (Update [7], Chi [9], Fusion [6], Explicit Fusion calculus [11], Solos [10]) have fusions. In Section IV we will introduce a single-binder calculus without fusions.

In all calculi in the paper, (reduction-closed) barbed congruence will be our reference behavioural equivalence. Its definition only requires a reduction relation, $\longrightarrow$, and a notion of barb on names, $\downarrow_a$. Intuitively, a barb at a holds for a process if that process can accept an offer of interaction at a from its environment. We omit the definition, which is standard. We write $\simeq_{\mathcal{L}}$ for (strong) reduction-closed barbed congruence in a calculus $\mathcal{L}$. Informally, $\simeq_{\mathcal{L}}$ is the largest relation that is context-closed, barb-preserving, and reduction-closed. Its weak version, written $\cong_{\mathcal{L}}$, replaces the relation $\longrightarrow_{\mathcal{L}}$ with its reflexive and transitive closure $\Longrightarrow_{\mathcal{L}}$, and the barbs $\downarrow_a^{\mathcal{L}}$ with the weak barbs $\Downarrow_a^{\mathcal{L}}$, where $\Downarrow_a^{\mathcal{L}}$ is the composition of the relations $\Longrightarrow_{\mathcal{L}}$ and $\downarrow_a^{\mathcal{L}}$ (i.e., the barb is visible after some internal actions). See [23] for more details.

III. Typing and subtyping with fusions

We consider typed versions of languages with fusions. We show that in such languages it is impossible to have a non-trivial subtyping, assuming a few simple and standard typing properties of name-passing calculi.

We use T, U to range over types, and Γ to range over type environments, i.e., partial functions from names to types. We write $\mathrm{dom}(\Gamma)$ for the set of names on which Γ is defined. In name-passing calculi, a type system assigns a type to each name. Typing judgements are of the form $\Gamma \vdash P$ (process P respects the type assignments in Γ), and $\Gamma \vdash a : T$ (name a can be assigned type T in Γ).[1] The following are the standard typing rules for parallel composition and restriction:

$$\frac{\Gamma \vdash P_1 \qquad \Gamma \vdash P_2}{\Gamma \vdash P_1 \mid P_2} \qquad \frac{\Gamma, x : T \vdash P}{\Gamma \vdash (\boldsymbol{\nu} x : T)\,P} \qquad (3)$$

The first rule says that any two processes typed in the same type environment can be composed in parallel. The second rule handles name restriction.[2]

In name-passing calculi, the basic type construct is the channel (or connection) type $\sharp\, T$. This is the type of a name that may carry, in an input or an output, values of type T. Consequently, we also assume that the following rule for prefixes $ab.\,P$ and $\overline{a}b.\,P$ is *admissible*.

$$\frac{\Gamma(a) = \sharp\, T \qquad \Gamma(b) = T \qquad \Gamma \vdash P}{\Gamma \vdash \alpha.\,P}\ \alpha \in \{ab, \overline{a}b\} \qquad (4)$$

(Prefixes may not have a continuation, in which case P would be missing from the rule.) In the rule, the type of the subject

[1] We consider in this paper basic type systems and basic properties for them; more sophisticated type systems exist where processes have a type too, e.g., behavioural type systems.

[2] In resource-sensitive type systems, i.e., those for linearity [16] and receptiveness [5], where one counts certain occurrences of the names, the rule for parallel composition has to be modified. As mentioned earlier, in this paper we stick to basic type systems, ignoring resource consumption.

and of the object of the prefix are compatible. Again, these need not be the typing rules for prefixes; we are just assuming that the rules are valid in the type system. The standard rule for prefix would have, as hypotheses,

$$\Gamma \vdash a : \sharp\, T \qquad \Gamma \vdash b : T \ .$$

These imply, but are not equivalent to, the hypotheses in (4), for instance in presence of subtyping.

Fundamental properties of type systems are:

- Subject Reduction (or Type Soundness): if $\Gamma \vdash P$ and $P \to P'$, then $\Gamma \vdash P'$;
- Weakening: if $\Gamma \vdash P$ and a is fresh, then $\Gamma, a : T \vdash P$;
- Strengthening: whenever $\Gamma, a : T \vdash P$ and a is fresh for P, then $\Gamma \vdash P$;
- Closure under injective substitutions: if $\Gamma, a : T \vdash P$ and b is fresh, then $\Gamma, b : T \vdash P\{b/a\}$.

Definition 2. *A typed calculus with single binder is* plain *if it satisfies Subject Reduction, Weakening, Strengthening, Closure under injective substitutions, and the typing rules (3) and (4) are admissible.*

If the type system admits subtyping, then another fundamental property is narrowing, which authorises, in a typing environment, the specialisation of types:

- (Narrowing): if $\Gamma, a : T \vdash P$ and $U \leq T$ then also $\Gamma, a : U \vdash P$.

When narrowing holds, we say that the calculus *supports narrowing*.

A typed calculus *has trivial subtyping* if, whenever $T \leq U$, we have $\Gamma, a : T \vdash P$ iff $\Gamma, a : U \vdash P$. When this is not the case (i.e., there are T, U with $T \leq U$, and T, U are not interchangeable in all typing judgements) we say that the calculus has *meaningful* subtyping.

Under the assumptions of Definition 2, a calculus with fusions may only have trivial subtyping.

Theorem 3. *A typed calculus with fusions that is plain and supports narrowing has trivial subtyping.*

In the proof, given in [23], we assume a meaningful subtyping and use it to derive a contradiction from type soundness and the other hypotheses.

One may wonder whether, in Theorem 3, more limited forms of narrowing, or a narrowing in the opposite direction, would permit some meaningful subtyping. Narrowing is interesting when it allows us to modify the type of the values exchanged along a name, that is, the type of the object of a prefix. (In process calculi, communication is the analogous of application for functional languages, and changing the type of an object is similar to changing the type of a function or of its argument.) In other words, disallowing narrowing on objects would make subtyping useless. We show that *any* form of narrowing, on one prefix object, would force subtyping to be trivial.

Theorem 4. *Suppose a typed calculus with fusions is plain and there is at least one prefix α with object b, different from the subject, and there are two types S and T such that $S \leq T$ and one of the following forms of narrowing holds for all Γ:*

1) *whenever $\Gamma, b : T \vdash \alpha.\, \mathbf{0}$, we also have $\Gamma, b : S \vdash \alpha.\, \mathbf{0}$;*
2) *whenever $\Gamma, b : S \vdash \alpha.\, \mathbf{0}$, we also have $\Gamma, b : T \vdash \alpha.\, \mathbf{0}$.*

Then S and T are interchangeable in all typing judgements.

As a consequence, authorising one of the above forms of narrowing for all S and T such that $S \leq T$ implies that the calculus has trivial subtyping. The proof of Theorem 4 is similar to that of Theorem 3.

Remark 5. Theorems 3 and 4 both apply to all fusion calculi: Fusion, Explicit Fusions, Update, Chi, Solos (where the continuation P is $\mathbf{0}$). □

Another consequence of Theorems 3 and 4 is that it is impossible, in plain calculi with fusions, to have an I/O type system; more generally, it is impossible to have any capability-based type system that supports meaningful subtyping.

Actually, to apply the theorems, it is not even necessary for the capability type system to have an explicit notion of subtyping. For Theorem 3, it is sufficient to have sets of capabilities with a non-trivial ordering under inclusion, meaning that we can find two capability types T and U such that whenever $\Gamma, a : U \vdash P$ holds then also $\Gamma, a : T \vdash P$ holds, but not the converse (e.g., T provides more capabilities than U). We could then impose a subtype relation $\leq$ on types, as the least preorder satisfying $T \leq U$. Theorem 3 then tells us that type soundness and the other properties of Definition 2 would require also $U \leq T$ to hold, i.e., T and U are interchangeable in all typing judgements. In other words, the difference between the capabilities in T and U has no consequence on typing. Similarly, to apply Theorem 4 it is sufficient to find two capability types T and U and a single prefix in whose typing U can replace T.

IV. A CALCULUS WITH NAME PREORDERS

A. Preorders, positive and negative occurrences

We now refine the fusion calculi by replacing the equivalence relation on names generated through communication by a preorder, yielding $\pi\mathtt{P}$ ('π with Preorder'). As the preorder on types given by subtyping allows promotions between related types, so the preorder on names of $\pi\mathtt{P}$ allows promotions between related names. Precisely, if a is below a name b in the preorder, then a prefix at a may be promoted to a prefix at b and then interact with another prefix at b. Thus an input $av.\, P$ may interact with an output $\overline{b}w.\, Q$; and, if also c is below b, then $av.\, P$ may as well interact with an output $\overline{c}z.\, R$.

The ordering on names is introduced by means of the *arc* construct, $a/\!\!/b$, that declares the *source* b to be below the *target* a. The remaining operators are as for fusion calculi (i.e., those of the π-calculus with bound input replaced by free input).

$$P ::= \mathbf{0} \ \Big|\ P \mid P \ \Big|\ \overline{a}b.\, P \ \Big|\ ab.\, P \ \Big|\ \boldsymbol{\nu} aP \ \Big|\ a/\!\!/b \ .$$

The semantics of the calculus is given in the reduction style. Structural congruence, $\equiv$, is defined as the usual congruence produced by the monoidal rules for parallel composition and

the rules for commuting and extruding restriction (see [23] for a complete definition). We explain the effect of reduction by means of contexts, rather than separate rules for each operator. Contexts allow us a more succinct presentation, and a simpler comparison with an alternative semantics (Section V). An *active context* is one in which the hole may reduce. Thus the only difference with respect to ordinary contexts is that the hole may not occur underneath a prefix. We use C to range over (ordinary) contexts, and E for active contexts. The rules for reduction are as follows (the subscript in $\longrightarrow_{\mathrm{ea}}$, for "eager", will distinguish this from the alternative semantics in Section V-A):

$$\text{R-SCON}: \quad \frac{P \equiv E[Q] \qquad Q \longrightarrow_{\mathrm{ea}} Q' \qquad E[Q'] \equiv P'}{P \longrightarrow_{\mathrm{ea}} P'}$$

$$\text{R-INTER}: \quad \overline{a}b.\,P \mid ac.\,Q \longrightarrow_{\mathrm{ea}} P \mid Q \mid b\!/\!c$$

$$\text{R-SUBOUT}: \quad a\!/\!b \mid \overline{b}c.\,Q \longrightarrow_{\mathrm{ea}} a\!/\!b \mid \overline{a}c.\,Q$$

$$\text{R-SUBINP}: \quad a\!/\!b \mid bc.\,Q \longrightarrow_{\mathrm{ea}} a\!/\!b \mid ac.\,Q$$

Rule R-INTER shows that communication generates an arc. Rules R-SUBOUT and R-SUBINP show that arcs only act on the subject of prefixes; moreover, they only act on *unguarded* prefixes (i.e., prefixes that are not underneath another prefix). The rules also show that arcs are persistent processes. Acting only on prefix subjects, arcs can be thought of as particles that "redirect prefixes": an arc $a\!/\!b$ redirects a prefix at b towards a higher name a. (Arcs remind us of special π-calculus processes, called forwarders or wires [17], which under certain hypotheses allow one to model substitutions; as for arcs, so the effect of forwarders is to replace the subject of prefixes.)

We write $\Longrightarrow_{\mathrm{ea}}$ for the reflexive and transitive closure of $\longrightarrow_{\mathrm{ea}}$. Here are some examples of reduction.

$$\begin{array}{rcl}
 & & \overline{a}c.\overline{c}a.e.P \mid ad.de.\overline{a}.Q \\
\text{R-INTER} & \longrightarrow_{\mathrm{ea}} & \overline{c}a.e.P \mid \quad de.\overline{a}.Q \mid c\!/\!d \\
\text{R-SUBINP} & \longrightarrow_{\mathrm{ea}} & \overline{c}a.e.P \mid \quad ce.\overline{a}.Q \mid c\!/\!d \\
\text{R-INTER} & \longrightarrow_{\mathrm{ea}} & e.P \mid \quad \overline{a}.Q \mid c\!/\!d \mid a\!/\!e \\
\text{R-SUBINP} & \longrightarrow_{\mathrm{ea}} & a.P \mid \quad \overline{a}.Q \mid c\!/\!d \mid a\!/\!e \\
\text{R-INTER} & \longrightarrow_{\mathrm{ea}} & P \mid \quad Q \mid c\!/\!d \mid a\!/\!e
\end{array}$$

Reductions can produce multiple arcs that act on the same name. This may be used to represent certain forms of choice, as in the following processes:

$$\begin{array}{l}
(\boldsymbol{\nu} h, k)\,(bu.\,cu.\,\overline{u} \mid \overline{b}h.\,h.\,P \mid \overline{c}k.\,k.\,Q) \\
\Longrightarrow_{\mathrm{ea}}\ (\boldsymbol{\nu} h, k)\,(\overline{u} \mid h\!/\!u \mid k\!/\!u \mid h.\,P \mid k.\,Q)\ .
\end{array}$$

Both arcs may act on $\overline{u}$, and are therefore in competition with each other. The outcome of the competition determines which process between P and Q is activated. For instance, reduction may continue as follows:

$$\begin{array}{rcl}
\text{R-SUBOUT} & \longrightarrow_{\mathrm{ea}} & (\boldsymbol{\nu} h, k)\,(\overline{k} \mid h\!/\!u \mid k\!/\!u \mid h.\,P \mid k.\,Q) \\
\text{R-INTER} & \longrightarrow_{\mathrm{ea}} & (\boldsymbol{\nu} h, k)\,(h\!/\!u \mid k\!/\!u \mid h.\,P \mid Q)\ .
\end{array}$$

Definition 6 (Positive and negative occurrences)**.** *In an input $ab.\,P$ and an arc $a\!/\!b$, the name b has a* negative occurrence*. All other occurrences of names in input, output and arcs are* positive occurrences*.*

An occurrence in a restriction ($\boldsymbol{\nu}$a) is neither negative nor positive, intuitively because restriction acts only as a binder, and does not stand for an usage of the name (in particular, it does not take part in a substitution).

Negative occurrences are particularly important, as by properly tuning them, different usages of names may be obtained. For instance, a name with zero negative occurrence is a constant (i.e., it is a channel, and may not be substituted); and a name that has a single negative occurrence is like a π-calculus name bound by an input (see Section VI-B).

The number of negative occurrences of a name is invariant under reduction.

Lemma 7. *If $P \longrightarrow_{\mathrm{ea}} P'$ then for each b, the number of negative occurrences of b in P and P' is the same.*

B. Types

We now show that the I/O capability type system and its subtyping can be transplanted from π to $\pi\mathtt{P}$. In all typed calculi in the paper, binding occurrences of names are annotated with their type — we are not concerned with type inference.

In the typing rules for I/O-types in the (monadic) π-calculus [1], two additional types are introduced: $\mathtt{o}\,T$, the type of a name that can be used only in output and that carries values of type T; and $\mathtt{i}\,T$, the type of a name that can be used only in input and that carries values of type T. The subtyping rules stipulate that $\mathtt{i}$ is covariant, $\mathtt{o}$ is contravariant, and $\sharp$ is invariant. Subtyping is brought up into the typing rules through the subsumption rule. The most important typing rules are those for input and output prefixes; for input we have:

$$\text{T-INPBOUND}: \quad \frac{\Gamma \vdash a : \mathtt{i}\,T \qquad \Gamma, b : T \vdash P}{\Gamma \vdash a(b : T).\,P}$$

The π-calculus supports narrowing, and this is essential in the proof of subject reduction.

The type system for $\pi\mathtt{P}$ is presented in Table I. With respect to the π-calculus, only the rule for input needs an adjustment, as $\pi\mathtt{P}$ uses free, rather than bound, input. The idea in rule T-INPFREE of $\pi\mathtt{P}$ is however the same as in rule T-INPBOUND of π: we look up the type of the object of the prefix, say T, and we require $\mathtt{i}\,T$ as the type for the subject of the prefix. To understand the typing of an arc $a\!/\!b$, recall that such an arc allows one to replace b with a. Rule T-ARC essentially checks that a has at least as many capabilities as b, in line with the intuition for subtyping in capability type systems.

Common to all premises of T-INPBOUND, T-INPFREE and T-ARC is the look-up of the type of names that occur negatively (the source of an arc and the object of an input prefix): the type that appears for b in the hypothesis is precisely the type found in the conclusion (within the process or in Γ). In contrast, the types for positive occurrences may be different (e.g., because of subsumption $\Gamma \vdash a : \mathtt{i}\,T$ may hold even if $\Gamma(a) \neq \mathtt{i}\,T$). We cannot type inputs like outputs: consider

$$\text{T-INPFREE2-WRONG}: \quad \frac{\Gamma \vdash a : \mathtt{i}\,T \qquad \Gamma \vdash b : T}{\Gamma \vdash ab}$$

Rule T-INPFREE2-WRONG would accept, for instance, an input ab in an environment Γ where $a : \mathtt{i}\,\mathtt{i}\,\mathbf{1}$ and $b : \sharp\,\mathbf{1}$. By

Types ($\mathbf{1}$ is the unit type): $\quad T ::= \mathtt{i}\,T \mid \mathtt{o}\,T \mid \sharp\,T \mid \mathbf{1}$

Subtyping rules:

$$\frac{}{\sharp\,T \leq \mathtt{i}\,T} \qquad \frac{}{\sharp\,T \leq \mathtt{o}\,T} \qquad \frac{S \leq T}{\mathtt{i}\,S \leq \mathtt{i}\,T} \qquad \frac{S \leq T}{\mathtt{o}\,T \leq \mathtt{o}\,S} \qquad \frac{}{T \leq T} \qquad \frac{S \leq T \qquad T \leq U}{S \leq U}$$

Typing rules:

$$\text{TV-NAME}\ \frac{}{\Gamma, a : T \vdash a : T} \qquad \text{SUBSUMPTION}\ \frac{\Gamma \vdash a : S \qquad S \leq T}{\Gamma \vdash a : T} \qquad \text{T-RES}\ \frac{\Gamma, a : T \vdash P}{\Gamma \vdash \boldsymbol{\nu} a P} \qquad \text{T-PAR}\ \frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \mid Q} \qquad \text{T-NIL}\ \frac{}{\Gamma \vdash \mathbf{0}}$$

$$\text{T-OUT}\ \frac{\Gamma \vdash a : \mathtt{o}\,T \qquad \Gamma \vdash b : T \qquad \Gamma \vdash P}{\Gamma \vdash \overline{a} b.\, P} \qquad \text{T-INPFREE}\ \frac{\Gamma \vdash a : \mathtt{i}\,\Gamma(b) \qquad \Gamma \vdash P}{\Gamma \vdash a b.\, P} \qquad \text{T-ARC}\ \frac{\Gamma \vdash a : \Gamma(b)}{\Gamma \vdash a/\!\!/b}$$

TABLE I
THE TYPE SYSTEM OF $\pi\mathtt{P}$

subtyping and subsumption, we could then derive $\Gamma \vdash b : \mathtt{i}\,\mathbf{1}$. In contrast, rule T-INPFREE, following the input rule of the π-calculus, makes sure that the object of the input does not have too many capabilities with respect to what is expected in the type of the subject of the input. This constraint is necessary for subject reduction. As a counterexample, assuming rule T-INPFREE2-WRONG, we would have $a : \sharp\,\mathtt{i}\,\mathbf{1}, b : \sharp\,\mathbf{1}, c : \mathtt{i}\,\mathbf{1} \vdash P$, for $P \stackrel{\text{def}}{=} ab \mid \overline{a}c \mid \overline{b}$. However, $P \longrightarrow_{\text{ea}} c/\!\!/b \mid \overline{b} \longrightarrow_{\text{ea}} c/\!\!/b \mid \overline{c}$, and the final derivative is not typable under Γ (as Γ only authorises inputs at c).

In $\pi\mathtt{P}$, the direction of the narrowing is determined by the negative or positive occurrences of a name.

Theorem 8 (Polarised narrowing)**.** *Let T_1 and T_2 be two types such that $T_1 \leq T_2$.*

1) *If a occurs only positively in P, then $\Gamma, a : T_2 \vdash P$ implies $\Gamma, a : T_1 \vdash P$.*
2) *If a occurs only negatively in P, then $\Gamma, a : T_1 \vdash P$ implies $\Gamma, a : T_2 \vdash P$.*
3) *If a occurs both positively and negatively in P, then it is in general unsound to replace, in a typing $\Gamma \vdash P$, the type of a in Γ with a subtype or supertype.*

Theorem 8 (specialised to prefixes) does not contradict Theorem 4, because in $\pi\mathtt{P}$, reduction does not satisfy (2) (from Section II). Our system enjoys subject reduction:

Theorem 9. *If $\Gamma \vdash P$ and $P \longrightarrow_{\text{ea}} P'$ then also $\Gamma \vdash P'$.*

Remark 10. Theorem 8 may be seen as a refinement of the standard narrowing result for name-passing calculi. In the π-calculus, for instance, a free name only has positive occurrences. Hence the usual narrowing corresponds to Theorem 8(1). And in an input $a(b : T).\, P$, the binder for b represents a negative occurrence, so that if b is free in P then b has both positive and negative occurrences, which means that the type T may not be modified, as by Theorem 8(3). In contrast, Theorem 8(2) is vacuous in π, as a name b with only negative occurrences is found in an input $a(b : T).\, P$ where b is not free in P.

In general, in a name-passing calculus, if a name has only positive occurrences, then its type (be it declared in the typing environment, or in the binding occurrence of that name within the process) may be replaced by a subtype, and conversely for names with only negative occurrences, whereas the type of names with both positive and negative occurrences may not be changed. Defining rules that distinguish between negative and positive occurrences in name-passing calculi is beyond the scope of this paper. A rule of thumb however seems that if the occurrence of a name generates a substitution acting on that name (i.e., a replacement of the name), then the occurrence is negative; if it does not, then it is positive. Thus in a fusion $a = b$ of the Explicit Fusion calculus, the occurrences of a and b are both positive and negative, as a fusion may produce a substitution $a/\!\!/b$ or a substitution $b/\!\!/a$ (which, incidentally, gives another explanation of the impossibility of narrowing in presence of an explicit fusion construct). □

Remark 11. For the Subject Reduction theorem for $\pi\mathtt{P}$ it is critical that an arc $a/\!\!/b$ only acts on positive occurrences of b. Provided this is respected, the theorem remains valid under different behaviours for arcs (e.g., simultaneously replacing all positive occurrences of b, not only at top-level). □

V. BEHAVIOURS

A. An alternative semantics

The operational semantics given to $\pi\mathtt{P}$ in Section IV allows arcs to act locally, at any time. The effect of an arc is irreversible: the application of an arc $a/\!\!/b$ to a prefix at b commits that prefix to interact along a name that is greater than, or equal to, a in the preorder among names. A commitment may disable certain interactions, even block a prefix for ever. Consider, e.g.,

$$(\boldsymbol{\nu} a, c)\,(bv.\, P \mid \overline{c}w.\, Q \mid a/\!\!/b \mid c/\!\!/b) \tag{5}$$

There is a competition between the two arcs; if the first wins, the process is deadlocked:

$$\longrightarrow_{\mathrm{ea}} (\boldsymbol{\nu} a, c)\ (av.\, P \mid \overline{c}w.\, Q \mid a/b \mid c/b)$$

since a and c are unrelated in the preorder.

We consider here an alternative semantics, in which the action of arcs is not a commitment: arcs come about only when interaction occurs. For this reason we call the new semantics *by-need* (arcs act only when 'needed'), whereas we call *eager* the previous semantics (arcs act regardless of matching prefixes). In this semantics, as in the π-calculus, an interaction involves both a synchronisation and a substitution; however unlike in the π-calculus where the substitution is propagated to the whole term, here substitution only replaces the subject of the interacting prefixes.

The formalisation of the new semantics makes use of the partial order on names induced by arcs. In a process, an arc is *active* if it is unguarded, i.e., it is not underneath a prefix. We write $\mathtt{preor}(P)$ for the preorder on names produced by the active arcs in P (i.e., the least preorder $\leq$ that includes $b \leq a$ for each active arc a/b in P). Similarly, $\mathtt{preor}(C)$ is the preorder produced by the active arcs of the context C. Note that this definition relies on the Barendregt convention on names (Convention 1), as it is purely syntactic, i.e., if P and P' are alpha convertible then $\mathtt{preor}(P)$ and $\mathtt{preor}(P')$ may be different. A definition that does not rely on the convention is given in [23].

We write $P \triangleright a \curlyvee b$ if $\{a, b\}$ has an upper bound in the preorder $\mathtt{preor}(P)$, that is, there is a name that is above both a and b; in this case we also say that a and b are *joinable*. Similarly we write $C \triangleright a \curlyvee b$ for contexts. For instance, we have $\boldsymbol{\nu} u(u/a \mid u/b \mid Q) \triangleright a \curlyvee b$, and $\boldsymbol{\nu} v(\overline{v}t \mid (\boldsymbol{\nu} w)(w/v \mid a/w \mid [\cdot]) \triangleright a \curlyvee v$. We have $P \triangleright a \curlyvee b$ iff $P' \triangleright a \curlyvee b$ if P and P' are alpha convertible and a and b occur free in P.

Example 12. *A process $M_{fg} = (\boldsymbol{\nu} c)(c/f \mid c/g)$ acts like a* mediator*: it joins names f and g (we have $M_{fg} \triangleright f \curlyvee g$). Mediators remind us of equators in the π-calculus, or of fusions in the Explicit Fusion calculus, but lack the transitivity property (e.g., $M_{fg} \mid M_{gh} \triangleright f \curlyvee h$ does not hold).*

Definition 13 (By-need reduction)**.** *The* by-need *reduction relation, $P \longrightarrow_{\mathrm{bn}} P'$, is defined by the following rules, where $\equiv$ is as in the eager semantics:*

$$\text{BN-SCON}: \quad \frac{P \equiv E[Q] \qquad Q \longrightarrow_{\mathrm{bn}} Q' \qquad E[Q'] \equiv P'}{P \longrightarrow_{\mathrm{bn}} P'}$$

$$\text{BN-RED}: \quad \frac{E \triangleright a \curlyvee b}{E[ac.\, P \mid \overline{b}d.\, Q] \longrightarrow_{\mathrm{bn}} E[P \mid d/c \mid Q]}$$

Relation $\Longrightarrow_{\mathrm{bn}}$ is the reflexive transitive closure of $\longrightarrow_{\mathrm{bn}}$.

While the eager semantics has simpler rules, the by-need semantics avoids 'too early commitments' on prefixes. For instance, the only immediate reduction of the process in (5) is

$$\longrightarrow_{\mathrm{bn}} (\boldsymbol{\nu} a, c)\ (P \mid w/v \mid Q \mid a/b \mid c/b)$$

where prefixes $bv.\, P$ and $\overline{c}w.\, Q$ interact because their subjects are joinable in the preorder generated by the two arcs.

Lemma 14 (Eager and by-need)**.** *$P \longrightarrow_{\mathrm{bn}} P'$ (by-need semantics) implies $P \Longrightarrow_{\mathrm{ea}} P'$ (eager semantics).*

Corollary 15. *Theorem 9 holds for the by-need semantics.*

B. Behavioural equivalence

We contrast barbed congruence in $\pi\mathtt{P}$ under the two semantics we have given, eager and by-need. We have already defined reduction relations, we only need to define barbs. This requires some care, as the interaction of a process with its environment may be mediated by arcs. For this, and to have a uniform definition of barbs under the eager and by-need semantics, we follow the definition of success in testing equivalence [18], using a special signal ω that we assume may not appear in processes: thus for any name a, the barb $\downarrow_a$ holds for a process P if there is a prefix α with subject a such that $P \mid \alpha.\, \omega$ reduces in one step to a process in which ω is unguarded (i.e., the offer of the environment of an action at a may be accepted by P). Weak barbs and barbed congruence are then defined in the standard way, as outlined in Section II. We write $\simeq_{\mathrm{ea}}$ and $\cong_{\mathrm{ea}}$ (resp. $\simeq_{\mathrm{bn}}$ and $\cong_{\mathrm{bn}}$) for the strong and weak versions of eager (resp. by-need) barbed congruence.

The eager and by-need semantics of $\pi\mathtt{P}$ yield incomparable equivalences. The two following laws are valid in the by-need case, and fail in the eager case:

$$(\boldsymbol{\nu} a)a/c = \mathbf{0} \qquad\qquad a \mid a = a.\, a \ .$$

To see the failure of the first law in the eager semantics, consider a context $C \stackrel{\text{def}}{=} [\cdot] \mid (\boldsymbol{\nu} b)(b/c) \mid c \mid \overline{c}.\, \overline{w}$; then $C[(\boldsymbol{\nu} a)(a/c)]$ can lose the possibility of emitting at w, by reducing in two steps to $(\boldsymbol{\nu} a)(a/c \mid a) \mid (\boldsymbol{\nu} b)(b/c \mid \overline{b}.\, \overline{w})$, because of a commitment determined by arcs; this cannot happen for $C[\mathbf{0}]$. There are no early commitments in the by-need semantics, for which the two processes are hence equal.

Similarly, in the eager semantics, it is possible to put $a \mid a$ in a context where two arcs rewrite each a prefix differently, while one can only rewrite the topmost prefix in $a.\, a$. This scenario cannot be played in the by-need semantics.

On the other hand, the following law is valid for strong (and weak) eager equivalence, but fails to hold in the by-need case:

$$(\boldsymbol{\nu} abu)(a/u \mid b/u \mid \overline{u} \mid a.\, \overline{w}) = (\boldsymbol{\nu} v)(\overline{v} \mid v.\, \tau.\, \overline{w} \mid v.\, \mathbf{0}) \ .$$

($\tau.\, \overline{w}$ stands for $\boldsymbol{\nu} c(c \mid \overline{c}.\, \overline{w})$). The intuition is that concurrent substitutions are used on the left-hand side to implement internal choice. As a consequence of the law $(\boldsymbol{\nu} a)a/c = \mathbf{0}$, in the by-need case, process b/u can be disregarded on the left, so that the process on the left *must* do the output on w.

We have introduced $\pi\mathtt{P}$ with the eager semantics for reasons of simplicity, but we find the by-need semantics more compelling. Below, unless otherwise stated, we work under by-need, though we also indicate what we know under eager.

C. Context-free characterisations of barbed congruence

When it comes to proving behavioural equalities, the definition of barbed congruence is troublesome, as it involves a heavy quantification on contexts. One therefore looks for context-free coinductive characterisations, as labelled bisimilarities that take into account not only reductions within a process, but also the potential interactions between the process and its environment (e.g., input and output actions). We present such characterisation for the by-need equivalence; currently we do not have one for the eager.

As actions for the by-need labelled bisimilarity, we use, besides τ-actions, only free input and free output:

$$\mu \quad ::= \quad \tau \mid ab \mid \overline{a}b \ .$$

In by-need, labelled transitions are written $P \xrightarrow{\mu}_{\rm bn} P'$. Internal transitions have already been defined, in the reduction semantics, thus we can take relation $\xrightarrow{\tau}_{\rm bn}$ to coincide with the reduction relation $\longrightarrow_{\rm bn}$. Input and output transitions are defined by these rules:

$$\text{BN-INP}: \quad \frac{E \rhd a \curlyvee b \qquad E \text{ does not bind } b \text{ and } d}{E[ac.\, P] \xrightarrow{bd}_{\rm bn} E[d/c \mid P]}$$

$$\text{BN-OUT}: \quad \frac{E \rhd a \curlyvee b \qquad E \text{ does not bind } b \text{ and } d}{E[\overline{a}c.\, P] \xrightarrow{\overline{b}d}_{\rm bn} E[c/d \mid P]}$$

The purpose of the two rules is to define the input and output transitions, with labels as simple as possible, with which to derive a labelled bisimilarity. The two rules are not supposed to be composed together to derive τ-actions (which are computed from the rules of reduction). We leave the definition of a pure SOS semantics, which avoids the structural manipulations of structural congruence, for future work.

To understand rules BN-INP and BN-OUT, suppose the environment is offering an action at b. Since a and b are joinable, there is a name, say e, that is above both a and b in the preorder; hence the prefix at a in the process and the prefix at b in the environment can be transformed into prefixes at e, and can interact. The need for the preorder explains why we found it convenient to express actions via active contexts. In the action, the use of a free object d allows us to ignore name extrusion and thus simplifies the bisimulation checks. As an example of BN-OUT, we have (similar observations can be made for BN-INP):

$$\begin{array}{l}(\boldsymbol{\nu} u)\left(\, u/b \mid (\boldsymbol{\nu} a, c)(u/a \mid \overline{a}c.\, P)\,\right) \\ \quad \xrightarrow{\overline{b}d}_{\rm bn} \ (\boldsymbol{\nu} u)\left(\, u/b \mid (\boldsymbol{\nu} a, c)(u/a \mid c/d \mid P)\,\right) \ .\end{array}$$

Here the process can interact with the environment at b (and hence perform a transition where b is the subject), because a and b are joinable. Name c is not extruded; instead the arc c/d redirects interactions on d to c.

The labelled bisimulation requires, besides the invariance for actions, invariance under the addition of arcs; moreover a check is made on the visible effects of arcs. In the clause for actions, no extrusion or binding on names is involved; further, it is sufficient that the objects of the actions are *fresh names*.

Definition 16 (Bisimulation)**.** *A* by-need *bisimulation $\mathcal{R}$ is a set of pairs (P, Q) s.t. $P\mathcal{R}Q$ implies:*

1) *$P \mid a/b \ \mathcal{R} \ Q \mid a/b$, for each name a, b (invariance under arcs);*
2) *if a and b appear free in P, then $P \rhd a \curlyvee b$ implies $Q \rhd a \curlyvee b$;*
3) *if $P \xrightarrow{\mu}_{\rm bn} P'$, then $Q \xrightarrow{\mu}_{\rm bn} Q'$ and $P'\mathcal{R}Q'$ (where the object part of μ is fresh);*
4) *the converse of clauses (2) and (3).*

Bisimilarity, written $\sim_{\rm bn}$, is the largest bisimulation.

We now present some examples and laws that are proved using the coinductive proof method of labelled bisimilarity. All equalities and inequalities also hold under the eager semantics, though for some equalities only in the weak case (e.g., Lemma 19).

Any input and output of πP can be transformed into a bound prefix, by introducing a new restricted name:

Lemma 17. *We have $ax.\, P \sim_{\rm bn} (\boldsymbol{\nu} x')ax'.\, (x'/x \mid P)$ and $\overline{b}y.\, Q \sim_{\rm bn} (\boldsymbol{\nu} y')\overline{b}y'.\, (y/y' \mid Q)$, for fresh x' and y'.*

If these laws are applied to all inputs and outputs of a process P, then the result is a process P' that is behaviourally the same as P, and in which all names exchanged in an interaction are fresh. Thus P' reminds us of a variant of π that achieves symmetry between input and output constructs, namely πI, the π-calculus with internal mobility [19].

Lemma 18. *We have $(\boldsymbol{\nu} b, c)\overline{a}c.\, \overline{a}b.\, \mathbf{0} \not\sim_{\rm bn} (\boldsymbol{\nu} c)\overline{a}c.\, \overline{a}c.\, \mathbf{0}$, and $(\boldsymbol{\nu} b, c)ac.\, ab.\, \mathbf{0} \sim_{\rm bn} (\boldsymbol{\nu} c)ac.\, ac.\, \mathbf{0}$.*

These laws show a difference between input and output in behavioural equalities. The reason for the inequality is that the first process can produce two transitions with objects e, f yielding $P \stackrel{\text{def}}{=} \boldsymbol{\nu} c\, (c/f \mid c/e)$, and then $P \rhd e \curlyvee f$.

Lemma 19 (Substitution and polarities)**.**

1) *If name a has only positive occurrences in P, then $(\boldsymbol{\nu} a)(P \mid b/a) \sim_{\rm bn} P\{b/a\}$;*
2) *if name a has only negative occurrences in P, then $(\boldsymbol{\nu} a)(P \mid a/b) \sim_{\rm bn} P\{b/a\}$;*
3) *$(\boldsymbol{\nu} a)(P \mid b/a \mid a/b) \sim_{\rm bn} P\{b/a\}$.*

For the comparison between labelled bisimilarity and barbed congruence, the most delicate part is the proof of congruence for bisimilarity. This is due to the shape of visible transitions, where an arc is introduced and the object part is always a fresh name, and to the use of $\equiv$ in the definition of transitions. The proof can be found in [23].

Theorem 20. *Bisimilarity is a congruence.*

Theorem 21 (Characterisation of barbed congruence)**.** *In πP, relations $\sim_{\rm bn}$ and $\simeq_{\rm bn}$ coincide.*

Hence all the laws stated above for $\sim_{\rm bn}$ hold for $\simeq_{\rm bn}$.

VI. EXPRESSIVENESS OF πP

We compare πP with a few other calculi, both as examples of the use of the calculus and as a test for its expressiveness.

When useful, we work in a *polyadic* version of πP; the addition of polyadicity goes as for other name-passing calculi in the literature. All results in this section use the by-need semantics; we do not know their status under the eager semantics.

A. Explicit Fusions

Bi-directional arcs, e.g., $a/b \mid b/a$, work as name fusions (cf, Lemma 19(3)). We thus can encode calculi based on name fusion into πP. As an example, we consider the Explicit Fusion calculus [8]. Its syntax extends the Fusion calculus with a fusion construct $a = b$. The encoding is defined as follows for prefixes and explicit fusions, the other constructs being encoded homomorphically:

$$\begin{array}{rcl} [\![\overline{a}\langle v\rangle.\, P]\!] & = & (\boldsymbol{\nu} w)\overline{a}\langle v, w\rangle.\, wv.\, [\![P]\!] \\ [\![ax.\, Q]\!] & = & (\boldsymbol{\nu} y)a\langle x, y\rangle.\, \overline{y}\langle x\rangle.\, [\![Q]\!] \\ [\![a = b]\!] & = & a/b \mid b/a \end{array}$$

In Explicit Fusions, an interaction introduces a name fusion. In the πP encoding, this is mimicked in two steps so to be able to produce bidirectional arcs. The second step is the reverse of the original interaction, and is realised by means of an extra private name. We have operational correspondence for the encoding (we do not know whether it is is fully abstract).

Theorem 22. *Let P, Q be processes of the Explicit Fusion calculus, and $\longrightarrow_{\mathrm{EF}}$ the reduction relation in the calculus.*

1) *If $P \equiv Q$ then $[\![P]\!] \simeq_{\mathrm{bn}} [\![Q]\!]$;*
2) *if $P \longrightarrow_{\mathrm{EF}} P'$ then $[\![P]\!] \longrightarrow_{\mathrm{bn}} \cong_{\mathrm{bn}} [\![P']\!]$;*
3) *conversely, if $[\![P]\!] \longrightarrow_{\mathrm{bn}} Q$, then $Q \cong_{\mathrm{bn}} [\![P']\!]$ for some P' such that $P \longrightarrow_{\mathrm{EF}} P'$.*

A similar result holds for the Fusion calculus, though for Explicit Fusions the statement is simpler because in the latter calculus a restriction is not necessary for fusions to act.

B. π-calculus

The embedding of the π-calculus into a fusion calculus is defined by translating the bound input construct as follows:

$$[\![a(x).\, P]\!] = (\boldsymbol{\nu} x)\, ax.\, [\![P]\!]$$

(the other constructs being translated homomorphically). The same encoding can be used for πP.

The encoding of π-calculus into Fusions is not fully abstract for barbed congruence. For instance, in the π-calculus, a new channel is guaranteed to remain different from all other existing channels. Thus in a process $\boldsymbol{\nu} a\,(\overline{b}a.\, (a.\, P \mid \overline{c}.\, Q))$, the two prefixes $a.\, P$ and $\overline{c}.\, Q$ may never interact with each other, in any context, even if a is exported. This property does not hold in the Fusion calculus, as a recipient of the newly created name a could equate it with any other name (e.g., using the context $bc.\, \mathbf{0} \mid [\cdot]$).

We do not know whether the encoding of the full π-calculus into πP is fully abstract. However, at least the encoding is fully abstract on the asynchronous subset (where no continuation is allowed after the output prefix).

Theorem 23. *Suppose P, Q are processes from the asynchronous π-calculus, $\mathrm{A}\pi$. Then $P \simeq_{\mathrm{A}\pi} Q$ iff $[\![P]\!] \simeq_{\mathrm{bn}} [\![Q]\!]$.*

In the theorem, $\simeq_{\mathrm{A}\pi}$ could be replaced by $\simeq_{\pi}$ (barbed congruence in the full π-calculus). Note that $\simeq_{\mathrm{A}\pi}$ is the standard barbed congruence, as opposed to *asynchronous* barbed congruence, where output barbs are visible but input barbs are not. We believe the theorem also holds under asynchronous barbed congruence.

For the proof of the theorem, we first establish results of operational correspondence between source and target terms of the encoding. Then the direction from right to left is easy because contexts of the π-calculus are also contexts of πP (under the encoding). The delicate direction is the opposite. Here we use Theorem 21, and the characterisation of π-calculus barbed congruence on the subset of asynchronous processes as ground bisimilarity [5]. We also make use of some up-to techniques, notably 'by-need bisimulation up to $\sim_{\mathrm{bn}}$ and restriction' whose soundness is proved along the lines of soundness proofs of similar techniques for other forms of bisimilarity. We finally consider the relation defined as $\{([\![P]\!] \mid \sigma, [\![Q]\!] \mid \sigma) \;\big|\; P \sim_{\mathrm{g}} Q\}$, where σ is a parallel composition of arcs, and prove that it is a by-need bisimulation up to $\sim_{\mathrm{bn}}$ and up to restriction.

Regarding translations in the opposite direction, both for fusion calculi and for πP, the encoding into π is not possible in general. However, for πP some results can be obtained under constraints such as *asynchrony* and *locality*. Something similar has been done by Merro [20] for the Fusion calculus.

VII. Unique negative occurrences of names

In this section we consider a constrained version of the calculi discussed in the paper, where each name may have at most one negative occurrence in a process. In the fusion calculus [6] the constraint means that each name appears at most once as the object of an input. In πP, the constraint affects also arcs (as their source is a negative occurrence).

The constraint is rather draconian, bringing the calculi closer to the π-calculus (where the constraint is enforced by having binding input). Still, the constraint is more generous than tying the input to a binder as in π. For instance, we have more complex forms of causality involving input, as in $\boldsymbol{\nu} x(ax.\, \overline{w}t \mid \overline{b}x)$, where the input at a blocks the output at w, and can be triggered before or after the output at b takes place. We call πP1 and FU1 the constrained versions of πP and Fusions; in both languages the constraint is preserved by reduction.

We show that the constraint makes certain differences between calculi or semantics disappear. In πP1 the eager and the by-need semantics of πP coincide, at least in a weak semantics.

Theorem 24. *In πP1, relations $\cong_{\pi\mathrm{P1ea}}$ and $\cong_{\pi\mathrm{P1bn}}$ coincide.*

The following property is useful in the proof (see [23]).

Lemma 25. *For $P \in \pi$P1, suppose $P \longrightarrow_{\mathrm{ea}} P'$ where the reduction is a rewrite step involving an arc. Then $P \cong_{\pi\mathrm{P1ea}} P'$.*

The calculi πP1 and FU1 resulting from the constraint are behaviourally similar. For instance, in πP1 the directionality of arcs is irrelevant, as shown by the following law (where we omit the subscripts 'ea' and 'bn' in the light of Theorem 24).

Lemma 26. $a/b \cong_{\pi\mathrm{P1}} b/a$.

Another difference that disappears under the constraint of unique negative occurrences of names is the one concerning capabilities and subtyping in fusion calculi with respect to π and $\pi\mathtt{P}$, exposed in Sections III and IV. Indeed, to equip FU1 with an I/O type system and subtyping, we can use exactly the rules of $\pi\mathtt{P}$ in Section IV-B — with the exception of T-ARC as FU1 does not have arcs. This intuitively because FU1 is, syntactically, a subset of $\pi\mathtt{P}$ (each process of FU1 is also a process of $\pi\mathtt{P}$), and the Subject Reduction theorem for $\pi\mathtt{P}$ in Section IV-B holds regardless of when and how arcs generate substitutions (Remark 11); making an arc a/b act immediately and on all positive occurrences of b is similar to substitution as in FU1. This may however involve changing the type of a name c into a smaller type when c is used in input object; e.g., in $ac \mid (\boldsymbol{\nu} b : T)\overline{a}b.\, P \longrightarrow_{\mathrm{FU1}} P\{c/b\}$ (where $\longrightarrow_{\mathrm{FU1}}$ is reduction in FU1), name c is used at type T, which is a smaller type than $\Gamma(c)$.

Theorem 27. *Let P be a* FU1 *process. If $\Gamma \vdash P$ and $P \longrightarrow_{\mathrm{FU1}} P'$, then $\Gamma' \vdash P'$, where for at most one name c, $\Gamma'(c) \leq \Gamma(c)$; for other names b, $\Gamma'(b) = \Gamma(b)$.*

Note that FU1 does not satisfy the conditions of Definition 2 because well-typed processes may not be freely put in parallel, as this could break the constraint on unique input objects.

We leave for future work a thorough comparison between $\pi\mathtt{P1}$, FU1, and π-calculus.

VIII. FUTURE WORK

Here we mention some lines for future work, in addition to those already mentioned in the main text.

The coinductive characterisation of behavioural equivalence in $\pi\mathtt{P}$ has been presented in the strong case, and should be extended to the weak case. We have presented and compared two semantics for $\pi\mathtt{P}$, eager and by-need. While we tend to consider the advantages so far uncovered for the by-need superior, more work is needed to draw more definite conclusions. For instance, it would also be interesting to contrast axiomatisations of the semantics, rules for pure SOS presentations of the operational semantics, the expressiveness of the subcalculus in which the two semantics agree, and implementations. We do not expect, in contrast, significant differences to arise from type systems.

Another possible advantage of by-need is a smoother extension with dynamic operators like guarded choice, in which an action may discard a component. (In the eager case it is unclear what should be the effect of an arc that acts on one of the summands of a choice.) Choice would be useful for axiomatisations. In by-need, we would have for instance

$$(\boldsymbol{\nu} b, c)\overline{a}b.\, \overline{a}c.\, (\overline{b} | c) \sim (\boldsymbol{\nu} b, c)\overline{a}b.\, \overline{a}c.\, (\overline{b}.\, c + c.\, \overline{b}).$$

The law, valid in both $\pi\mathtt{P}$ and π, illustrates the possibility of generating fresh names that cannot be identified with other names even if exported. The law fails in fusion calculi as a recipient might decide to equate b and c (cf. Section VI-B).

Solos calculus is the polyadic Fusion calculus without continuations. Solos can encode continuations [10]. We believe the same machinery would work for the 'Solos version' of $\pi\mathtt{P}$.

It could also be interesting to study the representation of $\pi\mathtt{P}$ into Psi calculi [21]. This may not be immediate because the latter make use of on an equivalence relation on channels, while the former uses a preorder. One could then see whether the move from Fusions and π to $\pi\mathtt{P}$ in this paper, and the corresponding results on types, can be lifted at the level of Psi calculi, by comparing them with variants based on preorders. [24] presents type systems for Psi calculi, and for explicit fusions, but does not address subtyping.

ACKNOWLEDGEMENT

The authors acknowledge support from the ANR projects 2010-BLAN-0305 PiCoq and 12IS02001 PACE.

REFERENCES

[1] B. Pierce and D. Sangiorgi, "Typing and subtyping for mobile processes," *Math. Str. in Comp. Sci.*, vol. 6, no. 5, pp. 409–453, 1996.

[2] N. Kobayashi, "Type systems for concurrent programs," in *10th Anniversary Colloquium of UNU/IIST*, ser. LNCS, vol. 2757. Springer, 2003, pp. 439–453.

[3] ——, "A new type system for deadlock-free processes," in *CONCUR*, ser. LNCS, vol. 4137. Springer, 2006, pp. 233–247.

[4] K. Honda, V. T. Vasconcelos, , and M. Kubo, "Language primitives and type discipline for structured communication-based programming," in *ESOP*, ser. LNCS, vol. 1381. Springer, 1998, pp. 122—-138.

[5] D. Sangiorgi and D. Walker, *The Pi-Calculus: a theory of mobile processes*. Cambridge University Press, 2001.

[6] J. Parrow and B. Victor, "The fusion calculus: expressiveness and symmetry in mobile processes," in *LICS*. IEEE, 1998, pp. 176 –185.

[7] ——, "The update calculus (extended abstract)," in *AMAST*, ser. LNCS, vol. 1349. Springer, 1997, pp. 409–423.

[8] L. Wischik and P. Gardner, "Explicit fusions," *Theor. Comput. Sci.*, vol. 340, no. 3, pp. 606–630, 2005.

[9] Y. Fu, "The χ-calculus," in *APDC*. IEEE Comp. Soc., 1997, pp. 74–81.

[10] C. Laneve and B. Victor, "Solos in concert," *Math. Str. in Comp. Sci.*, vol. 13, no. 5, pp. 657–683, 2003.

[11] P. Gardner and L. Wischik, "Explicit fusions," in *MFCS*, ser. LNCS, vol. 1893. Springer, 2000, pp. 373–382.

[12] J. Parrow and B. Victor, "The tau-laws of fusion," in *CONCUR*, ser. LNCS, vol. 1466. Springer, 1998, pp. 99–114.

[13] G. L. Ferrari, U. Montanari, E. Tuosto, B. Victor, and K. Yemane, "Modelling Fusion Calculus using HD-Automata," in *CALCO*, ser. LNCS, vol. 3629. Springer, 2005, pp. 142–156.

[14] F. Bonchi, M. G. Buscemi, V. Ciancia, and F. Gadducci, "A presheaf environment for the explicit fusion calculus," *J. Autom. Reasoning*, vol. 49, no. 2, pp. 161–183, 2012.

[15] M. Boreale, M. G. Buscemi, and U. Montanari, "A general name binding mechanism," in *TGC*, ser. LNCS, vol. 3705. Springer, 2005, pp. 61–74.

[16] N. Kobayashi, B. Pierce, and D. Turner, "Linearity and the pi-calculus," *TOPLAS*, vol. 21, no. 5, pp. 914–947, 1999.

[17] K. Honda and N. Yoshida, "On reduction-based process semantics," *Theor. Comp. Sci.*, vol. 152, no. 2, pp. 437–486, 1995.

[18] R. De Nicola and M. Hennessy, "Testing equivalences for processes," *Theor. Comput. Sci.*, vol. 34, pp. 83–133, 1984.

[19] D. Sangiorgi, "Pi-calculus, internal mobility, and agent-passing calculi," *Theor. Comput. Sci.*, vol. 167, no. 1&2, pp. 235–274, 1996.

[20] M. Merro, "Locality in the pi-calculus and applications to distributed objects," Ph.D. dissertation, École des Mines, France, 2000.

[21] J. Bengtson, M. Johansson, J. Parrow, and B. Victor, "Psi-calculi: Mobile processes, nominal data, and logic," in *LICS*. IEEE, 2009, pp. 39—48.

[22] B. Victor, "The fusion calculus : Expressiveness and symmetry in mobile processes," Ph.D. thesis, Uppsala University, 1998.

[23] Web appendix to this paper, available from http://hal.inria.fr/hal-00818068, 2013.

[24] H. Hüttel, "Typed ψ-calculi," in *CONCUR*, ser. LNCS, vol. 6901. Springer, 2011, pp. 265–279.

Under consideration for publication in Math. Struct. in Comp. Science

Compositional Methods for Information-Hiding[†]

Konstantinos Chatzikokolakis, Catuscia Palamidessi and Christelle Braun

INRIA, CNRS and École Polytechnique.
Email: {kostas,catuscia,braun}@lix.polytechnique.fr

Received 27 June 2012

Systems concerned with information hiding often use randomization to obfuscate the link between the observables and the information to be protected. The degree of protection provided by a system can be expressed in terms of the probability of error associated to the inference of the secret information. We consider a probabilistic process calculus to specify such systems, and we study how the operators affect the probability of error. In particular, we characterize constructs that have the property of not decreasing the degree of protection, and that can therefore be considered safe in the modular construction of these systems. As a case study, we apply these techniques to the Dining Cryptographers, and we derive a generalization of Chaum's strong anonymity result.

1. Introduction

During the last decade, internet activities have become an important part of many people's lives. As the number of these activities increases, there is a growing amount of personal information about the users that is stored in electronic form and that is usually transferred using public electronic means. This makes it feasible and often easy to collect, transfer and process a huge amount of information about a person. As a consequence, the need for mechanisms to protect such information is compelling.

A recent example of such privacy concerns are the so-called "biometric" passports. These passports, used by many countries and required by all visa waiver travelers to the United States, include a RFID chip containing information about the passport's owner. These chips can be read wirelessly without any contact with the passport and without the owner even knowing that his passport is being read. It is clear that such devices need protection mechanisms to ensure that the contained information will not be revealed to any non-authorized person.

In general, privacy can be defined as the ability of users to prevent information about themselves from becoming known to people other than those they choose to give the

[†] This work has been partially supported by the project ANR-09-BLAN-0169-01 PANDA and by the INRIA DRI Equipe Associée PRINTEMPS. A preliminary version of this work appeared in the proc. of FOSSACS 2008.

information to. We can further categorize privacy properties based on the nature of the hidden information. *Data protection* usually refers to confidential data like the credit card number. *Anonymity*, on the other hand, concerns the identity of the user who performed a certain action. *Unlinkability* refers to the link between the information and the user, and *unobservability* regards the actions of a user.

Information-hiding protocols aim at ensuring a privacy property during an electronic transaction. For example, the voting protocol Foo 92 ((Fujioka, Okamoto & Ohta 1993)) allows a user to cast a vote without revealing the link between the voter and the vote. The anonymity protocol Crowds ((Reiter & Rubin 1998)) allows a user to send a message on a public network without revealing the identity of the sender. These kinds of protocols often use *randomization* to introduce *noise*, thus limiting the inference power of a malicious observer.

1.1. *Information theory*

At an abstract level information-hiding protocols can be viewed as *information-theoretic channels*. A channel consists of a set of input values $\mathcal{S}$, a set of output values $\mathcal{O}$ (the observables) and a transition matrix which gives the conditional probability $p(o|s)$ of producing o as the output when s is the input. In the case of privacy preserving protocols, $\mathcal{S}$ contains the secret information that we want to protect and $\mathcal{O}$ the facts that the attacker can observe. This framework allows us to apply concepts from information theory to reason about the knowledge that the attacker can gain about the input by observing the output of the protocol (*information leakage*). This leakage is usually expressed in terms of *mutual information*, that is the difference between the a priori entropy (the initial uncertainty of the attacker) and the a posteriori entropy (the uncertainty of the attacker after the observation). The channel *capacity*, that is defined as the maximum mutual information under all possible a priori distributions, represents the worst case of leakage.

1.2. *Hypothesis testing*

Information theory is parametric on the notion of entropy. The most popular one is Shannon entropy, because of its relation with the channel's transmission rate. With respect to the problem of information-hiding, however, one of the most natural notion is arguably the Rényi min entropy (Rényi 1961). As discussed by Smith (Smith 2009), this notion represents well the *one-try attacks*, and it is strictly related to the problem of *hypothesis testing* and to the *Bayes risk*.

In information-hiding systems the attacker finds himself in the following scenario: he cannot directly detect the information of interest, namely the actual value of the random variable $S \in \mathcal{S}$, but he can discover the value of another random variable $O \in \mathcal{O}$ which depends on S according to a known conditional distribution. This kind of situation is quite common also in other disciplines, like medicine, biology, and experimental physics, to mention a few. The attempt to infer S from O is called *hypothesis testing* (the "hy-

pothesis" to be validated is the actual value of S), and it has been widely investigated in statistics.

One of the most used approaches to this problem is the Bayesian method, which consists in assuming known the a priori probability distribution of the hypotheses, and deriving from that (and from the matrix of the conditional probabilities) the a posteriori distribution after a certain fact has been observed. It is well known that the best strategy for the adversary is to apply the MAP (Maximum Aposteriori Probability) criterion, which, as the name says, dictates that one should choose the hypothesis with the maximum a posteriori probability for the given observation. "Best" means that this strategy induces the smallest probability of error in the guess of the hypothesis. The probability of error, in this case, is called *Bayes risk*. The a posteriori Rényi min entropy is the logarithm of the converse of the Bayes risk[†]. In (Chatzikokolakis, Palamidessi & Panangaden 2008*b*), we proposed to define the *degree of protection* provided by a protocol as the Bayes risk associated to the matrix. McIver and al. (McIver, Meinicke & Morgan 2010) have shown that the Bayes risk is the maximally discriminating among various notions of entropy, when compositionality is taken into account.

A major problem with the Bayesian method is that the a priori distribution is not always known. This is particularly true in security applications. In some cases, it may be possible to approximate the a priori distribution by statistical inference, but in most cases, especially when the input information changes over time, it may not. Thus other methods need to be considered, which do not depend on the a priori distribution. One such method is the one based on the so-called *Maximum Likelihood* criterion.

1.3. *Contribution*

In this paper we focus on the hypothesis testing approach to the one-try attacks, and consider both the scenario in which the input distribution is known, in which case we consider the Bayes risk, and the one in which we have no information on the input distribution, or it changes over time. In this second scenario, we consider as degree of protection the probability of error associated to the Maximum Likelihood rule, averaged on all possible input distributions. It turns out that such average is equal to the value of the probability of error on the point of uniform distribution, which is much easier to compute.

Next, we consider a probabilistic process algebra for the specification of information-hiding protocols, and we investigate which constructs in the language can be used safely in the sense that by applying them to a term, the degree of protection provided by the term does not decrease. This provides a criterion to build specifications in a compositional way, while preserving the degree of protection. We do this study for both the Bayesian and the Maximum Likelihood approaches.

We apply these compositional methods to the example of the Dining Cryptographers, and we are able to strengthen the strong anonymity result by Chaum. Namely we show

[†] There are other possible definitions of the a posteriori Rényi min entropy. Smith proposed to use this one because of its suitability for the information-hiding problem.

that we can have strong anonymity even if some coins are unfair, provided that there is a spanning tree of fair ones. This result is obtained by adding processes representing coins to the specification and using the fact that this can be done with a safe construct.

1.4. *Plan of the paper*

In the next section we recall some basic notions. Section 3 introduces the language CCS_p. Section 4 shows how to model protocols and process terms as channels. Section 5 discusses hypothesis testing and presents some properties of the probability of error. Section 6 characterizes the constructs of CCS_p which are safe. Section 7 applies previous results to find a new property of the Dining Cryptographers. Section 8 discusses related work. Section 9 concludes.

2. Preliminaries

In this section we give a brief overview of the technical concepts from the literature that will be used through the paper. More precisely, we recall here some basic notions of probability theory and probabilistic automata ((Segala 1995, Segala & Lynch 1995)).

2.1. *Probability spaces*

Let Ω be a set. A *σ-field* over Ω is a collection $\mathcal{F}$ of subsets of Ω closed under complement and countable union and such that $\Omega \in \mathcal{F}$. If $\mathcal{B}$ is a collection of subsets of Ω then *the σ-field generated by $\mathcal{B}$* is defined as the smallest σ-field containing U (its existence is ensured by the fact that the intersection of an arbitrary set of σ-fields containing $\mathcal{B}$ is still a σ-field containing $\mathcal{B}$).

A *measure* on $\mathcal{F}$ is a function $\mu : \mathcal{F} \to [0, \infty]$ such that

1 $\mu(\emptyset) = 0$ and
2 $\mu(\bigcup_i C_i) = \sum_i \mu(C_i)$ if $\{C_i\}_i$ is a countable collection of pairwise disjoint elements of $\mathcal{F}$.

A *probability measure* on $\mathcal{F}$ is a measure μ on $\mathcal{F}$ such that $\mu(\Omega) = 1$. A *probability space* is a tuple $(\Omega, \mathcal{F}, \mu)$ where Ω is a set, called the *sample space*, $\mathcal{F}$ is a σ-field on Ω and μ is a probability measure on $\mathcal{F}$. The elements of a σ-field $\mathcal{F}$ are also called *events*.

We will denote by $\delta(x)$ (called the *Dirac measure* on x) the probability measure s.t. $\delta(x)(\{y\}) = 1$ if $y = x$, and $\delta(x)(\{y\}) = 0$ otherwise. If $\{c_i\}_i$ are convex coefficients, and $\{\mu_i\}_i$ are probability measures, we will denote by $\sum_i c_i\mu_i$ the probability measure defined as $(\sum_i c_i\mu_i)(A) = \sum_i c_i\mu_i(A)$.

If A, B are events then $A \cap B$ is also an event. If $\mu(A) > 0$ then we can define the *conditional probability* $p(B|A)$, meaning "the probability of B given that A holds", as

$$p(B|A) = \frac{\mu(A \cap B)}{\mu(A)}$$

Note that $p(\cdot|A)$ is a new probability measure on $\mathcal{F}$. In continuous probability spaces,

where many events have zero probability, it is possible to generalize the concept of conditional probability to allow conditioning on such events. However, this is not necessary for the needs of this paper. Thus we will use the above "traditional" definition of conditional probability and make sure that we never condition on events of zero probability.

A probability space and the corresponding probability measure are called *discrete* if Ω is countable and $\mathcal{F} = 2^{\Omega}$. In this case, we can construct μ from a function $p : \Omega \to [0, 1]$ satisfying $\sum_{x \in \Omega} p(x) = 1$ by assigning $\mu(\{x\}) = p(x)$. The set of all discrete probability measures with sample space Ω will be denoted by $Disc(\Omega)$.

2.2. *Probabilistic automata*

A *probabilistic automaton* $\mathcal{M}$ is a tuple $(St, T_{init}, Act, \mathcal{T})$ where St is a set of states, $T_{init} \in St$ is the *initial state*, Act is a set of actions and $\mathcal{T} \subseteq St \times Act \times Disc(St)$ is a *transition relation*. Intuitively, if $(T, a, \mu) \in \mathcal{T}$ then there is a transition from the state T performing the action a and leading to a distribution μ over the states of the automaton. (We use T for states instead of s because later in the paper states will be (process) terms, and s will be used for sequences of actions.) We also write $T \xrightarrow{a} \mu$ if $(T, a, \mu) \in \mathcal{T}$. The idea is that the choice of transition among the available ones in $\mathcal{T}$ is performed nondeterministically, and the choice of the target state among the ones allowed by μ (i.e. those states T' such that $\mu(T') > 0$) is performed probabilistically. A probabilistic automaton $\mathcal{M}$ is *fully probabilistic* if from each state of $\mathcal{M}$ there is at most one transition available.

An *execution fragment* α of a probabilistic automaton is a (possibly infinite) sequence $T_0 a_1 T_1 a_2 T_2 \ldots$ of alternating states and actions, such that for each i there is a transition $(T_i, a_{i+1}, \mu_i) \in \mathcal{T}$ and $\mu_i(T_{i+1}) > 0$. We will use $fst(\alpha), lst(\alpha)$ to denote the first and last state of a finite execution fragment α respectively. An *execution* (or *history*) is an execution fragment such that $fst(\alpha) = T_{init}$. An execution α is maximal if it is infinite or there is no transition from $lst(\alpha)$ in $\mathcal{T}$. We denote by $exec^*(\mathcal{M})$, $exec^{\perp}(\mathcal{M})$, and $exec(\mathcal{M})$ the set of all the finite, all the non-maximal, and all executions of $\mathcal{M}$, respectively.

A *scheduler* of a probabilistic automaton $\mathcal{M} = (St, T_{init}, Act, \mathcal{T})$ is a function

$$\zeta : exec^{\perp}(\mathcal{M}) \to \mathcal{T}$$

such that $\zeta(\alpha) = (T, a, \mu) \in \mathcal{T}$ implies that $T = lst(\alpha)$.

The idea is that a scheduler selects a transition among the ones available in $\mathcal{T}$ and it can base his decision on the history of the execution. The *execution tree* of $\mathcal{M}$ relative to the scheduler ζ, denoted by $etree(\mathcal{M}, \zeta)$, is a fully probabilistic automaton $\mathcal{M}' = (St', T_{init}, Act, \mathcal{T}')$ such that $St' \subseteq exec^*(\mathcal{M})$, and $(\alpha, a, \mu') \in \mathcal{T}'$ if and only if $\zeta(\alpha) = (lst(\alpha), a, \mu)$ for some μ, and $\mu'(\alpha a T) = \mu(T)$. Intuitively, $etree(M, \zeta)$ is produced by unfolding the executions of $\mathcal{M}$ and resolving the nondeterminism using ζ.

Given a fully probabilistic automaton $\mathcal{M} = (St, T_{init}, Act, \mathcal{T})$ we can define a probability space $(\Omega_{\mathcal{M}}, \mathcal{F}_{\mathcal{M}}, p_{\mathcal{M}})$ on the space of executions of $\mathcal{M}$ as follows:

— $\Omega_{\mathcal{M}} \subseteq exec(\mathcal{M})$ is the set of maximal executions of $\mathcal{M}$.

— If α is a finite execution of $\mathcal{M}$ we define the cone with prefix α as $C_\alpha = \{\alpha' \in \Omega_{\mathcal{M}} | \alpha \leq$

$\alpha'\}$. Let $\mathcal{C}_{\mathcal{M}}$ be the collection of all cones of $\mathcal{M}$. Then $\mathcal{F}$ is the σ-field generated by $\mathcal{C}_{\mathcal{M}}$ (by closing under complement and countable union).

— We define the probability of a cone C_α where $\alpha = T_0 a_1 T_1 \dots a_n T_n$ as

$$p(C_\alpha) = \prod_{i=1}^{n} \mu_i(T_i)$$

where μ_i is the (unique because the automaton is fully probabilistic) measure such that $(T_{i-1}, a_i, \mu_i) \in \mathcal{T}$. We define $p_{\mathcal{M}}$ as the measure extending p to $\mathcal{F}$ (see (Segala 1995) for more details about this construction).

3. CCS with internal probabilistic choice

We consider an extension of standard CCS ((Milner 1989)) obtained by adding internal probabilistic choice. The resulting calculus CCS_p can be seen as a simplified version of the probabilistic π-calculus presented in (Herescu & Palamidessi 2000, Palamidessi & Herescu 2005) and it is similar to the one considered in (Deng, Palamidessi & Pang 2005). Like in those calculi, computations have both a probabilistic and a nondeterministic nature. The main conceptual novelty is a distinction between *observable* and *secret* actions, introduced for the purpose of specifying information-hiding protocols.

We assume a countable set *Act* of actions a, and we assume that it is partitioned into a set *Sec* of *secret actions* s, a set *Obs* of *observable actions* o, and the silent action τ. For each $s \in Sec$ we assume a complementary action $\overline{s} \in Sec$ such that $\overline{\overline{s}} = s$, and the same for *Obs*. The silent action τ does not have a complementary action, so the notation $\overline{a}$ will imply that $a \in Sec$ or $a \in Obs$.

The syntax of CCS_p is the following:

T	::=		*process term*
		$\sum_i p_i T_i$	*probabilistic choice*
	|	$\boxplus_i s_i.T_i$	*secret choice* $(s_i \in Sec)$
	|	$+_i r_i.T_i$	*nondeterministic choice* $(r_i \in Obs \cup \{\tau\})$
	|	$T \mid T$	*parallel composition*
	|	$(\nu a)\, T$	*restriction*
	|	$!T$	*replication*

All the summations in the syntax are finite. We will use the notation $T_1 \oplus_p T_2$ to represent a binary probabilistic choice $\sum_i p_i T_i$ with $p_1 = p$ and $p_2 = 1 - p$. Similarly we will use $a_1.T_1 \boxplus a_2.T_2$ to represent a binary secret or nondeterministic choice.

The semantics of a given CCS_p term is a probabilistic automaton whose states are process terms, whose initial state is the given term, and whose transitions are those derivable from the rules in Table 1. We will use the notations (T, a, μ) and $T \xrightarrow{a} \mu$ interchangeably. We denote by $\mu \mid T$ the measure μ' such that $\mu'(T' \mid T) = \mu(T')$

$$\text{PROB}\ \frac{}{\sum_i p_i T_i \xrightarrow{\tau} \sum_i p_i \delta(T_i)} \qquad \text{ACT}\ \frac{j \in I}{\lfloor\!+\!\rfloor_I a_i.T_i \xrightarrow{a_j} \delta(T_j)}$$

$$\text{PAR1}\ \frac{T_1 \xrightarrow{a} \mu}{T_1 \mid T_2 \xrightarrow{a} \mu \mid T_2} \qquad \text{PAR2}\ \frac{T_2 \xrightarrow{a} \mu}{T_1 \mid T_2 \xrightarrow{a} T_1 \mid \mu} \qquad \text{REP}\ \frac{T \mid !T \xrightarrow{a} \mu}{!T \xrightarrow{a} \mu \mid !T}$$

$$\text{COM}\ \frac{T_1 \xrightarrow{a} \delta(T_1') \quad T_2 \xrightarrow{\overline{a}} \delta(T_2')}{T_1 \mid T_2 \xrightarrow{\tau} \delta(T_1' \mid T_2')} \qquad \text{RES}\ \frac{T \xrightarrow{b} \mu \qquad \alpha \neq a, \overline{a}}{(\nu a)T \xrightarrow{b} (\nu a)\mu}$$

Table 1. *The semantics of CCS_p.*

for all processes T' and $\mu'(T'') = 0$ if T'' is not of the form $T' \mid T$, and similarly for $T \mid \mu$. Furthermore we denote by $(\nu a)\mu$ the measure μ' such that $\mu'((\nu a)T) = \mu(T)$, and $\mu'(T') = 0$ if T' is not of the form $(\nu a)T$.

Note that in the produced probabilistic automaton, all transitions to non-Dirac measures are silent. Note also that a probabilistic term generates exactly one (probabilistic) transition.

A transition of the form $T \xrightarrow{a} \delta(T')$, i.e. a transition having for target a Dirac measure, corresponds to a transition of a non-probabilistic automaton (a standard labeled transition system). Thus, all the rules of CCS_p specialize to the ones of CCS except from PROB. The latter models the internal probabilistic choice: a silent τ transition is available from the sum to a measure containing all of its operands, with the corresponding probabilities.

A secret choice $\lfloor\!+\!\rfloor_i s_i.T_i$ produces the same transitions as the nondeterministic term $\lfloor\!+\!\rfloor_i r_i.T_i$, except for the labels.

The distinction between the two kind of labels influences the notion of scheduler for CCS_p: the secret actions are assumed to be *inputs* of the system, namely they can only be performed if the input matches them. Hence some choices are determined, or influenced, by the input. In particular, a secret choice with different guards is entirely decided by the input. The scheduler has to resolve only the residual nondeterminism.

In the following, we use the notation $X \rightharpoonup Y$ to represent the partial functions from X to Y, and $\alpha_{|Sec}$ represents the projection of α on Sec.

Definition 3.1. Let T be a process in CCS_p and $\mathcal{M}$ be the probabilistic automaton generated by T. A scheduler is a function

$$\zeta : Sec^* \to exec^*(\mathcal{M}) \rightharpoonup \mathcal{T}$$

such that:

(i) if $s = s_1 s_2 \dots s_n$ and $\alpha_{|Sec} = s_1 s_2 \dots s_m$ with $m \leq n$, and
(ii) there exists a transition $(lst(\alpha), a, \mu)$ such that, if $a \in Sec$ then $a = s_{m+1}$

then $\zeta(s)(\alpha)$ is defined, and it is one of such transitions. We will write $\zeta_s(\alpha)$ for $\zeta(s)(\alpha)$.

Note that this definition of scheduler is different from the one used in probabilistic automaton, where the scheduler can decide to stop, even if a transition is allowed. Here the scheduler must proceed whenever a transition is allowed (provided that if it is labeled by a secret, that secret is the next one in the input string s).

We now adapt the definition of *execution tree* from the notion found in probabilistic automata. In our case, the execution tree depends not only on the scheduler, but also on the input.

Definition 3.2. Let $\mathcal{M} = (St, T, Act, \mathcal{T})$ be the probabilistic automaton generated by a CCS_p process T, where St is the set of processes reachable from T. Given an input s and a scheduler ζ, the *execution tree* of T for s and ζ, denoted by $etree(T, s, \zeta)$, is a fully probabilistic automaton $\mathcal{M}' = (St', T, Act, \mathcal{T}')$ such that $St' \subseteq exec(\mathcal{M})$, and $(\alpha, a, \mu') \in \mathcal{T}'$ if and only if $\zeta_s(\alpha) = (lst(\alpha), a, \mu)$ for some μ, and $\mu'(\alpha a T) = \mu(T)$.

4. Modeling protocols for information-hiding

In this section we propose an abstract model for information-hiding protocols, and we show how to represent this model in CCS_p. An extended example is presented in Section 7.

4.1. *Protocols as channels*

We view protocols as *channels* in the information-theoretic sense (Cover & Thomas 1991). The secret information that the protocol is trying to conceal constitutes the input of the channel, and the observables constitute the outputs. The set of the possible inputs and that of the possible outputs will be denoted by $\mathcal{S}$ and $\mathcal{O}$ respectively. We assume that $\mathcal{S}$ and $\mathcal{O}$ are of finite cardinality m and n respectively. We also assume a discrete probability distribution over the inputs, which we will denote by $\vec{\pi} = (\pi_{s_1}, \pi_{s_2}, \ldots, \pi_{s_m})$, where π_s is the probability of the input s.

To fit the model of the channel, we assume that at each run, the protocol is given exactly one secret s_i to conceal. This is not a restriction, because the s_i's can be complex information like sequences of keys or tuples of individual data. During the run, the protocol may use randomized operations to increase the level of uncertainty about the secrets and obfuscate the link with the observables. It may also have internal interactions between internal components, or other forms of nondeterministic behavior, but let us rule out this possibility for the moment, and consider a purely probabilistic protocol. We also assume there is exactly one output from each run of the protocol, and again, this is not a restrictive assumption because the elements of $\mathcal{O}$ can be structured data.

Given an input s, a run of the protocol will produce each $o \in \mathcal{O}$ with a certain probability $p(o|s)$ which depends on s and on the randomized operations performed by the protocol. Note that $p(o|s)$ depends only on the probability distributions on the mechanisms of the protocol, and not on the input distribution. The probabilities $p(o|s)$, for $s \in \mathcal{S}$ and $o \in \mathcal{O}$, constitute a $m \times n$ array M which is called the *matrix* of the channel, where the rows are indexed by the elements of $\mathcal{S}$ and the columns are indexed by the elements of $\mathcal{O}$. We will use the notation $(\mathcal{S}, \mathcal{O}, M)$ to represent the channel.

Note that the input distribution $\vec{\pi}$ and the probabilities $p(o|s)$ determine a distribution on the output. We will represent by $p(o)$ the probability of $o \in \mathcal{O}$. Thus both the input and the output can be considered *random variables*. We will denote these random variables by S and O.

If the protocol contains some forms of nondeterminism, like internal components giving rise to different interleaving and interactions, then the behavior of the protocol, and in particular the output, will depend on the scheduling policy. We can reduce this case to previous (purely probabilistic) scenario by assuming a scheduler ζ which resolves the nondeterminism entirely. Of course, the conditional probabilities, and therefore the matrix, will depend on ζ, too. We will express this dependency by using the notation M_ζ.

4.2. *Process terms as channels*

A given CCS_p term T can be regarded as a protocol in which the input is constituted by sequences of secret actions, and the output by sequences of observable actions. We assume that only a finite set of such sequences is relevant. This is certainly true if the term is terminating, which is usually the case in security protocols, as each session is supposed to terminate in finite time.

Thus the set S could be, for example, the set of all sequences of secret actions up to a certain length (for example, the maximal length of executions) and analogously O could be the set of all sequences of observable actions up to a certain length. To be more general, we will just assume $\mathcal{S} \subseteq_{fin} Sec^*$ and $\mathcal{O} \subseteq_{fin} Obs^*$.

Definition 4.1. Given a term T and a scheduler $\zeta : \mathcal{S} \to exec^*(\mathcal{M}) \to \mathcal{T}$, the matrix $M_\zeta(T)$ associated to T under ζ is defined as the matrix such that, for each $s \in \mathcal{S}$ and $o \in \mathcal{O}$, $p(o|s)$ is the probability of the set of the maximal executions in $etree(T, s, \zeta)$ whose projection in Obs is o.

The following remark may be useful to understand the nature of the above definition:

Remark 4.2. Given a sequence $s = s_1 s_2 \dots s_h$, consider the term

$$T' = (\nu Sec)(\bar{s}_1.\bar{s}_2.\dots.\bar{s}_h.0 \mid T)$$

Given a scheduler ζ for T, let ζ' be the scheduler on T' that chooses the transition

$$((\nu Sec)(\bar{s}_j.\bar{s}_2.\dots.\bar{s}_h.0 \mid U), r, (\nu Sec)(\bar{s}_j.\bar{s}_2.\dots.\bar{s}_h.0 \mid \mu))$$

if ζ_s chooses (U, r, μ), with $(r \notin Sec)$, and it chooses

$$((\nu Sec)(\bar{s}_j.\bar{s}_2.\dots.\bar{s}_h.0 \mid U), \tau, (\nu Sec)(\delta(\bar{s}_{j+1}.\bar{s}_2.\dots.\bar{s}_h.0 \mid (U'))))$$

if ζ_s chooses $(U, s_j, \delta(U'))$.

Note that ζ' is a "standard" scheduler, i.s. it does not depend on an input sequence.

We have that each element $p(o|s)$ in $M_\zeta(T)$ is equal to the probability of the set of all the maximal executions of T', under ζ', whose projection in Obs gives o.

5. Inferring the secrets from the observables

In this section we discuss possible methods by which an adversary can try to infer the secrets from the observables, and consider the corresponding probability of error, that is,

the probability that the adversary draws the wrong conclusion. We regard the probability of error as a representative of the degree of protection provided by the protocol, and we study its properties with respect to the associated matrix.

We start by defining the notion of *decision function*, which represents the guess the adversary makes about the secrets, for each observable. This is a well-known concept, particularly in the field of *hypothesis testing*, where the purpose is to try to discover the valid hypothesis from the observed facts, knowing the probabilistic relation between the possible hypotheses and their consequences. In our scenario, the hypotheses are the secrets.

Definition 5.1. A decision function for a channel $(\mathcal{S}, \mathcal{O}, M)$ is any function $f : \mathcal{O} \to \mathcal{S}$.

Given a channel $(\mathcal{S}, \mathcal{O}, M)$, an input distribution $\vec{\pi}$, and a decision function f , the *probability of error* $\mathcal{P}(f, M, \vec{\pi})$ is the average probability of guessing the wrong hypothesis by using f, weighted on the probability of the observable (see for instance (Cover & Thomas 1991)). The probability that, given o, s is the wrong hypothesis is $1 - p(s|o)$ (with a slight abuse of notation, we use $p(\cdot|\cdot)$ to represent also the probability of the input given the output). Hence we have:

Definition 5.2 ((Cover & Thomas 1991)). The probability of error is defined by

$$\mathcal{P}(f, M, \vec{\pi}) \;=\; 1 - \sum_{\mathcal{O}} p(o)p(f(o)|o)$$

Given a channel $(\mathcal{S}, \mathcal{O}, M)$, the best decision function that the adversary can use, namely the one that minimizes the probability of error, is the one associated to the so-called MAP rule, which prescribes choosing the hypothesis s which has *Maximum Aposteriori Probability* (for a given $o \in \mathcal{O}$), namely the s for which $p(s|o)$ is maximum. The fact that the MAP rule represent the 'best bet' of the adversary is rather intuitive, and well known in the literature. We refer to (Cover & Thomas 1991) for a formal proof.

The MAP rule is used in the so-called *Bayesian approach* to hypothesis testing, and the corresponding probability of error is also known as *Bayes risk*. We will denote it by $\mathcal{P}_{MAP}(M, \vec{\pi})$. The following characterization is an immediate consequence of Definition 5.2 and of the Bayes theorem $p(s|o) = p(o|s)\pi_s/p(o)$.

$$\mathcal{P}_{MAP}(M, \vec{\pi}) \;=\; 1 - \sum_{\mathcal{O}} \max_s (p(o|s)\pi_s)$$

It is natural then to define the degree of protection associated to a process term as the infimum probability of error that we can obtain from this term under every compatible scheduler (in a given class).

In the following, we assume the class of schedulers $\mathcal{A}$ to be the set of all the schedulers compatible with the given input $\mathcal{S}$.

It turns out that the infimum probability of error on $\mathcal{A}$ is actually a minimum. In order to prove this fact, let us first define a suitable metric on $\mathcal{A}$.

Definition 5.3. Consider a CCS_p process T, and let $\mathcal{M}$ be the probabilistic automaton

generated by T. We define a distance d between schedulers in $\mathcal{A}$ as follows:

$$d(\zeta,\zeta') = \begin{cases} 2^{-m} & \text{if } m = \min\{|\alpha| \mid \alpha \in exec^*(\mathcal{M}) \text{ and } \zeta(\alpha) \neq \zeta'(\alpha)\} \\ 0 & \text{if } \zeta(\alpha) = \zeta'(\alpha) \text{ for all } \alpha \in exec^*(\mathcal{M}) \end{cases}$$

where $|\alpha|$ represents the length of α.

Note that $\mathcal{M}$ is finitely branching, both in the nondeterministic and in the probabilistic choices, in the sense that from every node T' there is only a finite number of transitions (T', a, μ) and μ is a finite summation of the form $\mu = \sum_i p_i\ \delta(T_i)$. Hence we have the following (standard) result:

Proposition 5.4. $(\mathcal{A}, d)$ is a *sequentially compact* metric space, i.e. every sequence has a convergent subsequence (namely a subsequence with a limit in $\mathcal{A}$).

We are now ready to show that there exists a scheduler that gives the minimum probability of error:

Proposition 5.5. For every CCS$_p$ process T we have

$$\inf_{\zeta\in\mathcal{A}}\ \mathcal{P}_{MAP}(M_\zeta(T),\vec{\pi})\ =\ \min_{\zeta\in\mathcal{A}}\ \mathcal{P}_{MAP}(M_\zeta(T),\vec{\pi})$$

Proof. By Proposition 5.4, $(\mathcal{A}, d)$ is sequentially compact. By definition, $\mathcal{P}_{MAP}(M_\zeta(T),\vec{\pi})$ is a continuous function from $(\mathcal{A}, d)$ to $([0,1], d')$, where d' is the standard distance on real numbers. Consequently, $(\{\mathcal{P}_{MAP}(M_\zeta(T),\vec{\pi}) \mid \zeta \in \mathcal{A}\}, d')$ is also sequentially compact. Let $\{\zeta_n\}_n$ be a sequence such that for all n

$$\mathcal{P}_{MAP}(M_{\zeta_n}(T),\vec{\pi}) - \inf_{\mathcal{A}} \mathcal{P}_{MAP}(M_\zeta(T),\vec{\pi}) \leq 2^{-n}$$

We have that $\{\mathcal{P}_{MAP}(M_{\zeta_n}(T),\vec{\pi})\}_n$ is convergent and

$$\lim_n \mathcal{P}_{MAP}(M_{\zeta_n}(T),\vec{\pi})\ =\ \inf_{\mathcal{A}} \mathcal{P}_{MAP}(M_\zeta(T),\vec{\pi})$$

Consider now a convergent subsequence $\{\zeta_{n_j}\}_j$ of $\{\zeta_n\}_n$. By continuity of $\mathcal{P}_{MAP}$, we have

$$\lim_n \mathcal{P}_{MAP}(M_{\zeta_n}(T),\vec{\pi})\ =\ \lim_j \mathcal{P}_{MAP}(M_{\zeta_{n_j}}(T),\vec{\pi})\ = \mathcal{P}_{MAP}(\lim_j M_{\zeta_{n_j}}(T),\vec{\pi})$$

which concludes the proof. □

Thanks to previous proposition, we can define the degree of protection provided by a protocols in terms of the minimum probability of error.

Definition 5.6. Given a CCS$_p$ process T, the protection $Pt_{MAP}(T)$ provided by T, in the Bayesian approach, is given by

$$Pt_{MAP}(T,\vec{\pi})\ =\ \min_{\zeta\in\mathcal{A}}\ \mathcal{P}_{MAP}(M_\zeta(T),\vec{\pi})$$

The problem with the MAP rule is that it assumes that the input distribution is known to the adversary. This is often not the case, so it is natural to try to approximate it with some other rule. One such rule is the so-called ML rule, which prescribes the choice of the s which has *Maximum Likelihood* (for a given $o \in \mathcal{O}$), namely the s for which

$p(o|s)$ is maximum. The name comes from the fact that $p(o|s)$ is called the *likelihood* of s given o. We will denote the corresponding probability of error by $\mathcal{P}_{ML}(M, \vec{\pi})$. The following characterization is an immediate consequence of Definition 5.2 and of the Bayes theorem.

$$\mathcal{P}_{ML}(M, \vec{\pi}) \;=\; 1 - \sum_{\mathcal{O}} \max_{s}(p(o|s))\pi_s$$

It has been shown (see for instance (Chatzikokolakis, Palamidessi & Panangaden 2008*a*)) that under certain conditions on the matrix, the ML rule approximates indeed the MAP rule, in the sense that by repeating the protocol the adversary can make the probability of error arbitrarily close to 0, with either rule.

We could now define the degree of protection provided by a term T under the ML rule as the minimum $\mathcal{P}_{ML}(M_\zeta(T), \vec{\pi})$, but it does not seem reasonable to give a definition that depends on the input distribution, since the main reason to apply a non-Bayesian approach is that indeed we do not know the input distribution. Instead, we define the degree of protection associated to a process term as the *average* probability of error with respect to all possible distributions $\vec{\pi}$:

Definition 5.7. Given a CCS_p process T, the protection $Pt_{ML}(T)$ provided by T, in the Maximum Likelihood approach, is given by

$$Pt_{ML}(T) \;=\; \min_{\zeta \in \mathcal{A}} \; (m-1)! \int_{\vec{\pi}} \mathcal{P}_{ML}(M_\zeta(T), \vec{\pi}) \, d\vec{\pi}$$

In the above definition, $(m-1)!$ represents a normalization function: $\frac{1}{(m-1)!}$ is the hypervolume of the domain of all possible distributions $\vec{\pi}$ on $\mathcal{S}$, namely the $(m-1)$-dimensional space of points $\vec{\pi}$ such that $0 \leq \pi_s \leq 1$ and $0 \leq \sum_{s \in S} \pi_s = 1$ (where m is the cardinality of $\mathcal{S}$).

Fortunately, it turns out that this definition is equivalent to a much simpler one: the average value of the probability of error, under the Maximum Likelihood rule, can be obtained simply by computing $\mathcal{P}_{ML}$ on the uniform distribution $\vec{\pi}_u = (\frac{1}{m}, \frac{1}{m}, \ldots, \frac{1}{m})$.

Theorem 5.8. $Pt_{ML}(T) \;=\; \min_{\zeta \in \mathcal{A}} \; \mathcal{P}_{ML}(M_\zeta(T), \vec{\pi}_u)$

Proof.

SimplificationsGiven a channel $(\mathcal{S}, \mathcal{O}, M)$ and an input distribution $\vec{\pi} = (\pi_1, \ldots, \pi_m)$ of cardinality m, the probability of error is characterized by the expression:

$$f_m(\vec{\pi}) \;=\; 1 - \sum_{\mathcal{O}} \max_{s}(p(o|s))\pi_s \;=\; \mathcal{P}_{ML}(M, \vec{\pi})$$

$f_m(\vec{\pi})$ is a linear function of the input distribution $\vec{\pi}$ of the form:

$$f_m(\vec{\pi}) = 1 - a_1\pi_1 - \ldots - a_m\pi_m$$

where $\forall i, a_i \in \mathbb{R}$.

With the additional constraint $\sum_{i=1\ldots m} \pi_i = 1$, the dependency on one of the m variables $\pi_1, \ldots, \pi_m$, for instance π_m, can be removed. Replacing π_m by the equivalent

expression $1 - \sum_{i=1}^{m-1} \pi_i$ yields:

$$f_m(\vec{\pi}) \;=\; c_1\pi_1 + \ldots + c_{m-1}\pi_{m-1} + c_m$$

with

$$\begin{array}{rcl} c_1 & = & a_m - a_1 \\ c_2 & = & a_m - a_2 \\ \ldots & & \\ c_{m-1} & = & a_m - a_{m-1} \\ c_m & = & 1 - a_m \end{array}$$

Expression of the normalization function The hyper-volume $V_m(X)$ of the domain $D_m(X)$ of all possible distributions $\vec{\pi}$ on $\mathcal{S}$, i.e. the $(m-1)$-dimensional space of points $\vec{\pi}$ such that $0 \leq \pi_s \leq X$ and $0 \leq \sum_{s \in S} \pi_s = X$ (where m is the cardinality of $\mathcal{S}$) is given by:

$$V_m(X) \;=\; \frac{X^{m-1}}{(m-1)!}$$

Induction hypothesis We will show by induction on m that following equality $\mathcal{H}_m$ holds for all m:

$$\int\limits_{D_m(X)} f_m(\vec{\pi})d\vec{\pi} \;=\; V_m(X) f_m(\vec{\pi}_u(X))$$

where $\vec{\pi}_u(X) = (\frac{X}{m}, \frac{X}{m}, \ldots, \frac{X}{m})$. Theorem 5.8 then follows by taking $X = 1$.
According to the aforementioned notations, $\mathcal{H}_m$ can be written as:

$$L_m(X) \;=\; R_m(X)$$

where

$$L_m(X) \;=\; \int\limits_{x_{m-1}=0}^{X} \int\limits_{x_{m-2}=0}^{X-x_{m-1}} \ldots \int\limits_{x_1=0}^{X-x_{m-1}-\ldots-x_2} f_m(x_1, x_2, \ldots, x_{m-1})dx_1dx_2\ldots dx_{m-1}$$

and

$$R_m(X) \;=\; \frac{X^{m-1}}{(m-1)!}(\sum_{i=1}^{m-1} c_i\frac{X}{m} + c_m)$$

Base step: $m = 2$We have:

$$\begin{array}{rcl} L_2(X) & = & \int_{x_1=0}^{x_1=X} (c_1x_1 + c_2)dx_1 \\ & = & \frac{c_1X^2}{2} + c_2X \\ & = & X(\frac{c_1X}{2} + c_2) \\ & = & R_2(X) \end{array}$$

Induction step: $\mathcal{H}_m \Rightarrow \mathcal{H}_{m+1}$Consider

$$\begin{aligned} f_{m+1}(x) &= c_1x_1 + \ldots + c_mx_m + c_{m+1} \\ &= \textstyle\sum_{i=1}^{m} c_ix_i + c_{m+1} \\ &= f_m(x) - c_m + c_mx_m + c_{m+1} \end{aligned}$$

The left-hand side of $\mathcal{H}_{m+1}$ is given by:

$$L_{m+1}(X) = \textstyle\int_{x_m=0}^{x_m=Y} \ldots \int_{x_1=0}^{x_1=Y-x_m-\ldots-x_2} f_{m+1}(x_1,\ldots,x_m)dx_1\ldots dx_m$$

The $m-1$ inner-most integrations can be resolved according to $\mathcal{H}_m$. Replacing X by $Y - x_m$ leads to:

$$\begin{aligned} L_{m+1}(X) &= \textstyle\int_{x_m=0}^{x_m=Y} V_m(Y-x_m) f_{m+1}(\frac{Y-x_m}{m},\ldots,\frac{Y-x_m}{m})dx_m \\ &= \textstyle\int_{x_m=0}^{x_m=Y} \frac{(Y-x_m)^{m-1}}{(m-1)!}(\sum_{i=1}^{m-1} c_i\frac{Y-x_m}{m} + c_mx_m + c_{m+1})dx_m \end{aligned}$$

Replacing $Y - x_m$ by Z leads to:

$$\begin{aligned} L_{m+1}(X) &= \textstyle\int_{Z=0}^{Z=Y} \frac{Z^{m-1}}{(m-1)!}(\sum_{i=1}^{m-1} c_i\frac{Z}{m} + c_m(Y-Z) + c_{m+1})dZ \\ &= \textstyle\int_{Z=0}^{Z=Y} ((\frac{1}{m!}\sum_{i=1}^{m-1} c_i - \frac{c_m}{(m-1)!})Z^m + \frac{c_mY+c_{m+1}}{(m-1)!}Z^{m-1})dZ \\ &= (\frac{1}{(m+1)!}\textstyle\sum_{i=1}^{m-1} c_i)Y^{m+1} - \frac{c_m}{(m-1)!(m+1)}Y^{m+1} + \frac{c_m}{m!}Y^{m+1} + \frac{c_{m+1}}{m!}Y^m \\ &= (\frac{1}{(m+1)!}\textstyle\sum_{i=1}^{m-1} c_i)Y^{m+1} + \frac{c_m}{(m+1)!}Y^{m+1} + \frac{c_{m+1}}{m!}Y^m \\ &= \frac{Y^m}{m!}(\textstyle\sum_{i=1}^{m} c_i\frac{Y}{m+1} + c_{m+1}) = R_{m+1}(Y) \end{aligned}$$

This completes the proof for Theorem 5.8. □

The next corollary follows immediately from Theorem 5.8 and from the definitions of $\mathcal{P}_{MAP}$ and $\mathcal{P}_{ML}$.

Corollary 5.9. $Pt_{ML}(T) = \min_{\zeta\in\mathcal{A}} \mathcal{P}_{MAP}(M_\zeta(T), \vec{\pi}_u)$

We conclude this section with some properties of $\mathcal{P}_{MAP}$. Note that the same properties hold also for $\mathcal{P}_{ML}$ on the uniform distribution, because $\mathcal{P}_{ML}(M, \vec{\pi}_u) = \mathcal{P}_{MAP}(M, \vec{\pi}_u)$.

The next proposition shows that the probabilities of error are *concave* functions with respect to the space of matrices.

Proposition 5.10. Consider a family of channels $\{(\mathcal{S}, \mathcal{O}, M_i)\}_{i\in I}$, and a family $\{c_i\}_{i\in I}$ of convex coefficients, namely $0 \le c_i \le 1$ for all $i \in I$, and $\sum_{i\in I} c_i = 1$. Then:

$$\mathcal{P}_{MAP}(\sum_{i\in I} c_i\, M_i, \vec{\pi}) \;\ge\; \sum_{i\in I} c_i\, \mathcal{P}_{MAP}(M_i, \vec{\pi})$$

Proof. Consider $\forall i \in I, M_i = (p_i(o|s))_{s\in\mathcal{S},o\in\mathcal{O}}$. Then:

$$\begin{aligned}
\mathcal{P}_{MAP}(\textstyle\sum_i c_i\, M_i, \vec{\pi}) &= 1 - \textstyle\sum_o \max_s(\sum_i c_i\, p_i(o|s)\pi_s) \\
&\geq 1 - \textstyle\sum_o \sum_i c_i\ \max_s(p_i(o|s)\pi_s) \\
&= 1 - \textstyle\sum_i \sum_o c_i\ \max_s(p_i(o|s)\pi_s) \ \text{(since the summands are positive)} \\
&= 1 - \textstyle\sum_i c_i\ \sum_o \max_s(p_i(o|s)\pi_s) \\
&= \textstyle\sum_{i\in I} c_i - \sum_{i\in I} c_i\ \sum_{o\in\mathcal{O}} \max_s(p_i(o|s)\pi_s) \ \text{(since } \textstyle\sum_{i\in I} c_i \ = \ 1) \\
&= \textstyle\sum_{i\in I} c_i(1 - \sum_{o\in\mathcal{O}} \max_s(p_i(o|s)\pi_s)) \\
&= \textstyle\sum_{i\in I} c_i \mathcal{P}_{MAP}(M_i, \vec{\pi})
\end{aligned}$$

□

Corollary 5.11. Consider a family of channels $\{(\mathcal{S}, \mathcal{O}, M_i)\}_{i\in I}$, and a family $\{c_i\}_{i\in I}$ of convex coefficients. Then:

$$\mathcal{P}_{MAP}(\sum_{i\in I} c_i\, M_i, \vec{\pi}) \ \geq \ \min_{i\in I}\ \mathcal{P}_{MAP}(M_i, \vec{\pi})$$

The next proposition shows that if we transform the observables, and collapse the columns corresponding to observables which have become the same after the transformation, the probability of error does not decrease.

Proposition 5.12. Consider a channel $(\mathcal{S}, \mathcal{O}, M)$, where M has conditional probabilities $p(o|s)$, and a transformation of the observables $f : \mathcal{O} \to \mathcal{O}'$. Let M' be the matrix whose conditional probabilities are $p'(o'|s) = \sum_{f(o)=o'} p(o|s)$ and consider the new channel $(\mathcal{S}, \mathcal{O}', M')$. Then:

$$\mathcal{P}_{MAP}(M', \vec{\pi}) \ \geq \ \mathcal{P}_{MAP}(M, \vec{\pi})$$

Proof. The result derives from:

$$\begin{aligned}
\textstyle\sum_{o'\in\mathcal{O}'} \max_s(p'(o'|s)\pi_s) &= \textstyle\sum_{o'\in\mathcal{O}'} \max_s(\sum_{f(o)=o'} p(o|s)\pi_s) \\
&\leq \textstyle\sum_{o'\in\mathcal{O}'} \sum_{f(o)=o'} \max_s(p(o|s)\pi_s) \\
&= \textstyle\sum_{o\in\mathcal{O}} \max_s(p(o|s)\pi_s)
\end{aligned}$$

□

The following propositions are from the literature.

Proposition 5.13 ((Chatzikokolakis et al. 2008*a*)). Given $\mathcal{S}$, $\mathcal{O}$, let M be a matrix indexed on $\mathcal{S}$, $\mathcal{O}$ such that all the rows of M are equal, namely $p(o|s) = p(o|s')$ for all $o \in \mathcal{O}$, $s, s' \in \mathcal{S}$. Then,

$$\mathcal{P}_{MAP}(M, \vec{\pi}) \ = \ 1 - \max_s\ \pi_s$$

Furthermore $\mathcal{P}_{MAP}(M, \vec{\pi})$ is the maximum probability of error, i.e. for every other matrix

M' indexed on $\mathcal{S}$, $\mathcal{O}$ we have:

$$\mathcal{P}_{MAP}(M, \vec{\pi}) \;\geq\; \mathcal{P}_{MAP}(M', \vec{\pi})$$

Proposition 5.14 ((Bhargava & Palamidessi 2005)). Given a channel $(\mathcal{S}, \mathcal{O}, M)$, the rows of M are equal (and hence the probability of error is maximum) if and only if $p(s|o) = \pi_s$ for all $s \in \mathcal{S}$, $o \in \mathcal{O}$.

The condition $p(s|o) = \pi_s$ means that the observation does not give any additional information concerning the hypothesis. In other words, the *a posteriori* probability of s coincides with its *a priori* probability. The property $p(s|o) = \pi_s$ for all $s \in \mathcal{S}$ and $o \in \mathcal{O}$ was used as a definition of (strong) anonymity by Chaum (Chaum 1988) and was called *conditional anonymity* by Halpern and O'Neill (Halpern & O'Neill 2005).

6. Safe constructs

In this section we investigate constructs of the language CCS_p which are *safe* with respect to the protection of the secrets.

We start by giving some conditions that will allow us to ensure the safety of the parallel and the restriction operators.

Definition 6.1. Consider process term T, and the observables $o_1, o_2, \ldots, o_k$ such that

(i) T does not contain any secret action, and
(ii) the observable actions of T are included in $o_1, o_2, \ldots, o_k$.

Then we say that T is safe outside of $o_1, o_2, \ldots, o_k$.

The following theorem states our main results for Pt_{MAP}. Note that they are also valid for Pt_{ML}, because $Pt_{ML}(T) = Pt_{MAP}(T, \vec{\pi}_u)$.

Theorem 6.2. The probabilistic choice, the nondeterministic choice, and a restricted form of parallel composition are safe constructs, namely, for every input probability π, and any terms $T_1, T_2, \ldots, T_h$, we have

(1) $$Pt_{MAP}(\sum_i p_i\, T_i, \vec{\pi}) \;\geq\; \sum_i p_i\, Pt_{MAP}(T_i, \vec{\pi}) \;\geq\; \min_i\, Pt_{MAP}(T_i, \vec{\pi})$$

(2) $$Pt_{MAP}(\Big[\!\!+\!\!\Big]_i\, o_i.T_i, \vec{\pi}) \;=\; \min_i Pt_{MAP}(T_i, \vec{\pi})$$

(3) $$Pt_{MAP}((\nu\ o_1, o_2, \ldots, o_k)\,(T_1 \mid T_2)) \;\geq\; Pt_{MAP}(T_2, \vec{\pi})$$
if T_1 is safe outside of $o_1, o_2, \ldots, o_k$.

Proof.

1 By definition $Pt_{MAP}(\sum_i p_i\, T_i, \vec{\pi}) = \min_{\zeta \in \mathcal{A}}\ \mathcal{P}_{MAP}(M_\zeta(\sum_i p_i\, T_i), \vec{\pi})$.
Let $\zeta_m = \mathrm{minarg}_{\mathcal{A}}\, \mathcal{P}_{MAP}(M_\zeta(\sum_i p_i\, T_i), \vec{\pi})$. Hence

$$Pt_{MAP}(\sum_i p_i\, T_i, \vec{\pi}) \;=\; \mathcal{P}_{MAP}(M_{\zeta_m}(\sum_i p_i\, T_i), \vec{\pi})$$

Consider, for each i, the scheduler ζ_{m_i} defined as ζ_m on the i-th branch, except for

the removal of the first state and the first τ-step from the execution fragments in the domain. It's easy to see that

$$M_{\zeta_m}(\sum_i p_i\, T_i) = \sum_i p_i M_{\zeta_{m_i}}(T_i)$$

From Proposition 5.10 we derive

$$\mathcal{P}_{MAP}(M_{\zeta_m}(\sum_i p_i\, T_i), \vec{\pi}) \geq \sum_i p_i \mathcal{P}_{MAP}(M_{\zeta_{m_i}}(T_i), \vec{\pi})$$

Finally, observe that ζ_{m_i} is still compatible with $\mathcal{S}$, hence we have

$$\mathcal{P}_{MAP}(M_{\zeta_{m_i}}(T_i), \vec{\pi}) \;\geq\; Pt_{MAP}(T_i, \vec{\pi}) \quad \text{for all } i$$

which concludes the proof in this case.

2 Let $\zeta_m = \mathrm{minarg}_{\mathcal{A}}\, \mathcal{P}_{MAP}(M_\zeta(\lfloor\!+\!\rfloor_i\, o_i.T_i), \vec{\pi})$. Let $\mathcal{A}_i$ be the class of schedulers that choose the i-th branch at the beginning of the execution, and define

$$\zeta_{n_i} = \operatorname*{minarg}_{\mathcal{A}_i} \mathcal{P}_{MAP}(M_\zeta(\Big\lfloor\!\!+\!\!\Big\rfloor_i\, o_i.T_i), \vec{\pi})$$

Obviously we have

$$Pt_{MAP}(\Big\lfloor\!\!+\!\!\Big\rfloor_i\, o_i.T_i, \vec{\pi}) \;=\; \min_i\; \mathcal{P}_{MAP}(M_{\zeta_{n_i}}(\Big\lfloor\!\!+\!\!\Big\rfloor_i\, o_i.T_i), \vec{\pi})$$

Consider now, for each i, the scheduler ζ_{m_i} defined as as ζ_{n_i}, except for the removal of the first state and the first step from the execution fragments in the domain. Obviously ζ_{m_i} is still compatible with $\mathcal{S}$, and the observables of T_i are i one-to one correspondence with those of $\lfloor\!+\!\rfloor_i\, o_i.T_i$ via the bijective function $f_i(o_i o_{j_1} \ldots o_{j_k}) = o_{j_1} \ldots o_{j_k}$. Furthermore, all the probabilities of the channel $M_{\zeta_{n_i}}(\lfloor\!+\!\rfloor_i\, o_i.T_i)$ are the same as those of $M_{\zeta_{m_i}}(T_i)$ modulo the renaming of o into $f(o)$. Hence we have

$$\mathcal{P}_{MAP}(M_{\zeta_{n_i}}(\Big\lfloor\!\!+\!\!\Big\rfloor_i\, o_i.T_i), \vec{\pi}) \;=\; \mathcal{P}_{MAP}(M_{\zeta_{m_i}}(T_i), \vec{\pi}) \;=\; Pt_{MAP}(T_i, \vec{\pi})$$

which concludes the proof of this case.

3 Let $\zeta_m = \mathrm{minarg}_{\mathcal{A}}\, \mathcal{P}_{MAP}(M_\zeta((\nu o_1, o_2, \ldots, o_k)\,(T_1 \mid T_2)), \vec{\pi})$. Hence

$$Pt_{MAP}((\nu o_1, o_2, \ldots, o_k)\,(T_1 \mid T_2), \vec{\pi}) \;=\; \mathcal{P}_{MAP}(M_{\zeta_m}((\nu o_1, o_2, \ldots, o_k)\,(T_1 \mid T_2)), \vec{\pi})$$

The proof proceeds by constructing a set of series of schedulers whose limit with respect to the metric d in Definition 5.3 correspond to schedulers on the execution tree of T_2. Consider a generic node in the execution tree of $(\nu o_1, o_2, \ldots, o_k)\,(T_1 \mid T_2)$ under ζ_m, and let $(\nu o_1, o_2, \ldots, o_k)\,(T_1' \mid T_2')$ be the new term in that node. Assume α to be the execution history up to that node. Let us consider separately the three possible kinds of transitions derivable from the operational semantics:

(a) $(\nu o_1, o_2, \ldots, o_k)\,(T_1' \mid T_2') \xrightarrow{a} (\nu o_1, o_2, \ldots, o_k)\,(\mu \mid T_2')$ due to a transition $T_1' \xrightarrow{a} \mu$. In this case a must be τ because of the assumption that T_1 does not contain secret actions and all its observable actions are included in $\{o_1, o_2, \ldots, o_k\}$. Assume that $\mu = \sum_i p_i\, \delta(T_{1i}')$. Then we have $(\nu o_1, o_2, \ldots, o_k)\,(\mu \mid T_2') = \sum_i p_i\, \delta((\nu o_1, o_2, \ldots, o_k)\,(T_{1i}' \mid T_2'))$. Let us consider the tree obtained by replacing this distribution with $\delta((\nu o_1, o_2, \ldots, o_k)\,(T_{1i}' \mid T_2'))$

(i.e. the tree obtained by pruning all alternatives except $(\nu o_1, o_2, \ldots, o_k)\,(T'_{1i} \mid T'_2)$, and assigning to it probability 1). Let ζ_{mi} be the projection of ζ_m on the new tree (i.e. ζ_{mi} is defined as the projection of ζ_m on the histories α' such that if α is a proper prefix of α' then $\alpha\tau(\nu o_1, o_2, \ldots, o_k)\,(T'_{1i} \mid T'_2)$ is a prefix of α'). We have

$$\begin{gathered}\mathcal{P}_{MAP}(M_{\zeta_m}((\nu o_1, o_2, \ldots, o_k)\,(T_1 \mid T_2)), \vec{\pi}) \\ = \\ \mathcal{P}_{MAP}(\textstyle\sum_i p_i\; M_{\zeta_{mi}}((\nu o_1, o_2, \ldots, o_k)\,(T_1 \mid T_2)), \vec{\pi}) \\ \geq \qquad \text{(by Proposition 5.10)} \\ \textstyle\sum_i p_i\; \mathcal{P}_{MAP}(M_{\zeta_{mi}}((\nu o_1, o_2, \ldots, o_k)\,(T_1 \mid T_2)), \vec{\pi})\end{gathered}$$

In the execution tree of T_2 the above transition does not have a correspondent, but it obliges us to consider all different schedulers that are associated to the various ζ_{mi}'s for different i's.

(b) $(\nu o_1, o_2, \ldots, o_k)\,(T'_1 \mid T'_2) \xrightarrow{a} (\nu o_1, o_2, \ldots, o_k)\,(T'_1 \mid \mu)$ due to a transition $T'_2 \xrightarrow{a} \mu$, with a not included in $\{o_1, o_2, \ldots, o_k\}$. In this case, the corresponding scheduler for T_2 must choose the same transition, i.e. $T'_2 \xrightarrow{a} \mu$.

(c) $(\nu o_1, o_2, \ldots, o_k)\,(T'_1 \mid T'_2) \xrightarrow{\tau} (\nu o_1, o_2, \ldots, o_k)\,\delta(T''_1 \mid T''_2)$ due to the transitions $T'_1 \xrightarrow{a} \delta(T''_1)$ and $T'_2 \xrightarrow{\bar{a}} \delta(T''_2)$. In this case a must be an observable o because of the assumption that T_1 does not contain secret actions. The corresponding scheduler for T_2 must choose the transition $T'_2 \xrightarrow{\bar{a}} \delta(T''_2)$.

By considering the inequalities given by the transitions of type (a), we obtain

$$\begin{gathered}\mathcal{P}_{MAP}(M_{\zeta_m}((\nu o_1, o_2, \ldots, o_k)\,(T_1 \mid T_2)), \vec{\pi}) \\ \geq \\ \textstyle\sum_i p_i\; \mathcal{P}_{MAP}(M_{\zeta_{mi}}((\nu o_1, o_2, \ldots, o_k)\,(T_1 \mid T_2)), \vec{\pi}) \\ \geq \\ \textstyle\sum_i p_i\; \sum_j q_j\; \mathcal{P}_{MAP}(M_{\zeta_{mij}}((\nu o_1, o_2, \ldots, o_k)\,(T_1 \mid T_2)), \vec{\pi}) \\ \geq \\ \textstyle\sum_i p_i\; \sum_j q_j\; \sum_h r_h\; \mathcal{P}_{MAP}(M_{\zeta_{mijh}}((\nu o_1, o_2, \ldots, o_k)\,(T_1 \mid T_2)), \vec{\pi}) \\ \geq \\ \ldots\end{gathered}$$

Observe now that $\{\zeta_m, \zeta_{mi}, \zeta_{mij}, \zeta_{mijh}, \ldots\}$ is a converging series of schedulers whose limit $\zeta_{mijh\ldots}$ is isomorphic to a scheduler for T_2, except that some of the observable transitions in T_2 may be removed due to the restriction on $o_1, o_2, \ldots, o_k$. This removal

determines a (usually non injective) mapping f on the observables. Hence:

$$\mathcal{P}_{MAP}(M_{\zeta_m}((\nu o_1, o_2, \ldots, o_k)\,(T_1 \mid T_2)), \vec{\pi})$$
$$\geq$$
$$\textstyle\sum_i p_i \ \sum_j q_j \ \sum_h r_h \ \ldots \mathcal{P}_{MAP}(M_{\zeta_{mijh\ldots}}((\nu o_1, o_2, \ldots, o_k)\,(T_1 \mid T_2)), \vec{\pi})$$
$$\geq \qquad \text{(by Proposition 5.12)}$$
$$\textstyle\sum_i p_i \ \sum_j q_j \ \sum_h r_h \ \ldots \mathcal{P}_{MAP}(M_{\zeta_{mijh\ldots}}(T_2), \vec{\pi})$$
$$\geq$$
$$\textstyle\sum_i p_i \ \sum_j q_j \ \sum_h r_h \ \ldots \min_{\zeta \in \mathcal{A}} \mathcal{P}_{MAP}(M_\zeta(T_2), \vec{\pi})$$

Finally, observe that $\sum_i p_i = \sum_j q_j = \sum_h r_h = \ldots = 1$, hence

$$\mathcal{P}_{MAP}(M_{\zeta_m}((\nu o_1, o_2, \ldots, o_k)\,(T_1 \mid T_2)), \vec{\pi}) \geq \min_{\zeta \in \mathcal{A}} \mathcal{P}_{MAP}(M_\zeta(T_2), \vec{\pi})$$

which concludes the proof.

□

Unfortunately the safety property does not hold for the secret choice. The following is a counterexample.

Example 6.3. Let $Sec = \{s_1, s_2\}$ and assume that $\mathcal{S}$ does not contain the empty sequence. Let $T = o_1.0 \boxplus o_2.0$. Then $Pt_{MAP}(T, \vec{\pi})$ is maximum (i.e. $Pt_{MAP}(T, \vec{\pi}) = 1 - \max \vec{\pi}$) because for every sequence $s \in \mathcal{S}$ we have $p(o_1|s) = p(o_2|s)$. Let $T' = s_1.T \boxplus s_2.T$. We can now define a scheduler such that, if the secret starts with s_1, it selects o_1, and if the secret starts with s_2, it selects o_2. Hence, under this scheduler, $p(o_1|s_1 s) = p(o_2|s_2 s) = 1$ while $p(o_1|s_2 s) = p(o_2|s_1 s) = 0$. Therefore $Pt_{MAP}(T', \vec{\pi}) = 1 - p_1 - p_2$ where p_1 and p_2 are the maximum probabilities of the secrets of the form $s_1 s$ and $s_2 s$, respectively. Note now that either $\max \vec{\pi} = p_1$ or $\max \vec{\pi} = p_2$ because of the assumption that $\mathcal{S}$ does not contain the empty sequence. Let $\vec{\pi}$ be such that both p_1 and p_2 are positive. Then $1 - p_1 - p_2 < 1 - \max \vec{\pi}$, hence $Pt_{MAP}(T', \vec{\pi}) < Pt_{MAP}(T, \vec{\pi})$.

The reason why we need the condition *(i)* in Definition 6.1 for the parallel operator is analogous to the case of secret choice. The following is a counterexample.

Example 6.4. Let *Sec* and $\mathcal{S}$ be as in Example 6.3. Define $T_1 = s_1.0 \boxplus s_2.0$ and $T_2 = o_1.0 \boxplus o_2.0$. Clearly, $Pt_{MAP}(T_2, \vec{\pi}) = 1 - \max \vec{\pi}$. Consider now the term $T_1 \mid T_2$ and define a scheduler that first executes an action s in T_1 and then, if s is s_1, it selects o_1, while if s is s_2, it selects o_2. The rest proceeds like in Example 6.3, where $T' = T_1 \mid T_2$ and $T = T2$.

The reason why we need the condition *(ii)* in Definition 6.1 is that without it the parallel operator may create different interleavings, thus increasing the possibility of an adversary discovering the secrets. The following is a counterexample.

Example 6.5. Let *Sec* and $\mathcal{S}$ be as in Example 6.3. Define $T_1 = o.0$ and $T_2 =$

$s_1.(o_1.0 \oplus_{.5} o_2.0) \boxplus s_2.(o_1.0 \oplus_{.5} o_2.0)$. It is easy to see that $Pt_{MAP}(T_2, \vec{\pi}) = 1 - \max \vec{\pi}$. Consider the term $T_1 \mid T_2$ and define a scheduler that first executes an action s in T_2 and then, if s is s_1, it selects first T_1 and then the continuation of T_2, while if s is s_2, it selects first the continuation of T_2 and then T_1. Hence, under this scheduler, $p(oo_1|s_1s) = p(oo_2|s_1s) = .5$ and also $p(o_1o|s_2s) = p(o_2o|s_2s) = .5$ while $p(oo_1|s_2s) = p(oo_2|s_2s) = 0$ and $p(o_1o|s_1s) = p(o_2o|s_1s) = 0$. Therefore $Pt_{MAP}(T, \vec{\pi}) = 1 - p_1 - p_2$ where p_1 and p_2 are the maximum probabilities of the secrets of the form s_1s and s_2s, respectively. Following the same reasoning as in Example 6.3, we have that for certain $\vec{\pi}$, $Pt_{MAP}(T', \vec{\pi}) < Pt_{MAP}(T, \vec{\pi})$.

7. A case study: the Dining Cryptographers

In this section we consider the Dining Cryptographers (DC) protocol proposed by Chaum (Chaum 1988), we show how to describe it in CCS_p, and we apply the results of previous section to obtain a generalization of Chaum's strong anonymity result.

In its most general formulation, the DC consists of a multigraph where one of the nodes (cryptographers) may be secretly designated to pay for the dinner. The cryptographers would like to find out whether there is a payer or not, but without either discovering the identity of the payer, nor revealing it to an external observer. The problem can be solved as follows: we put on each edge a probabilistic coin, which can give either 0 or 1. The coins get tossed, and each cryptographer computes the binary sum of all (the results of) the adjacent coins. Furthermore, it adds 1 if it is designated to be the payer. Finally, all the cryptographers declare their result.

It is easy to see that this protocol solves the problem of figuring out the existence of a payer: the binary sum of all declarations is 1 if and only if there is a payer, because all the coins get counted twice, so their contribution to the total sum is 0.

The property we are interested in, however, is the anonymity of the system. Chaum proved that the DC is strongly anonymous if all the coins are fair, i.e. they give 0 and 1 with equal probability, and the multigraph is connected, namely there is a path between each pair of nodes. To state formally the property, let us denote by s the secret identity of the payer, and by o the collection of the declarations of the cryptographers.

Theorem 7.1 ((Chaum 1988)). If the multigraph is connected, and the coins are fair, then DC is strongly anonymous, namely for every s and o, $p(s|o) = p(s)$ holds.

We are now going to show how to express the DC in CCS_p. We start by introducing a notation for value-passing in CCS_p, following standard lines.

$$\begin{array}{rrcl} \textit{Input} & c(x).T & = & \left\lfloor\!\!\left\lfloor + \right\rfloor\!\!\right\rfloor_v c_v.T[v/x] \\ \textit{Output} & \bar{c}\langle v\rangle & = & \bar{c}_v \end{array}$$

The protocol can be described as the parallel composition of the cryptographers processes $Crypt_i$, of the coin processes $Coin_h$, and of a process *Collect* whose purpose is to collect all the declarations of the cryptographers, and output them in the form of a tuple.

$$
\begin{array}{rcl}
\mathit{Crypt}_i & = & c_{i,i_1}(x_1) \,.\, \ldots \,.\, c_{i,i_k}(x_k) \,.\, \mathit{pay}_i(x) \,.\, \bar{d}_i\langle x_1 + \ldots + x_k + x\rangle \\
\mathit{Coin}_h & = & \bar{c}_{\ell,h}\langle 0\rangle \,.\, \bar{c}_{r,h}\langle 0\rangle.0 \;\oplus_{p_h}\; \bar{c}_{\ell,h}\langle 1\rangle \,.\, \bar{c}_{r,h}\langle 1\rangle.0 \\
\mathit{Collect} & = & d_1(y_1) \,.\, d_2(y_2) \,.\, \ldots \,.\, d_n(y_n) \,.\, \overline{\mathit{out}}\langle y_1, y_2, \ldots, y_n\rangle \\
\mathit{DC} & = & (\nu \vec{c})(\nu \vec{d})(\prod_i \mathit{Crypt}_i \mid \prod_h \mathit{Coin}_h \mid \mathit{Collect})
\end{array}
$$

Table 2. *The dining cryptographers protocol expressed in CCS_p.*

See Table 2. In this protocol, the secret actions are pay_i. All the others are observable actions.

Each coin communicates with two cryptographers. $c_{i,h}$ represents the communication channel between Coin_h and Crypt_i if h is indeed the index of a coin, otherwise it represents a communication channel "with the environment". We will call the latter *external*. In the original definition of the DC there are no external channels, we have added them to prove a generalization of Chaum's result. They could be interpreted as a way for the environment to influence the computation of the cryptographers and hence test the system, for the purpose of discovering the secret.

We are now ready to state our generalization of Chaum's result:

Theorem 7.2. A DC is strongly anonymous if it has a spanning tree consisting of fair coins only.

Proof. Consider the term DC in Table 2. Remove all the coins that do not belong to the spanning tree, and the corresponding restriction operators. Let T be the process term obtained this way. Let $\mathcal{A}$ be the class of schedulers which select the value 0 for all the external channels. This situation corresponds to the original formulation of Chaum and so we can apply Chaum's result (Theorem 7.1) and Proposition 5.14 to conclude that all the rows of the matrix M are the same and hence, by Proposition 5.13, $\mathcal{P}_{\mathit{MAP}}(M, \vec{\pi}) = 1 - \max_i \pi_i$.

Consider now one of the removed coins, h, and assume, without loss of generality, that $c_{\ell,h}(x)$, $c_{r,h}(x)$ are the first actions in the definitions of Crypt_ℓ and Crypt_r. Consider the class of schedulers $\mathcal{B}$ that selects value 1 for x in these channels. The matrix M' that we obtain is isomorphic to M: the only difference is that each column o is now mapped to a column $o + w$, where w is a tuple that has 1 in the ℓ and r positions, and 0 in all other positions, and $+$ represents the componentwise binary sum. Since this map is a bijection, we can apply Proposition 5.12 in both directions and derive that $\mathcal{P}_{\mathit{MAP}}(M', \vec{\pi}) = 1 - \max_i \pi_i$.

By repeating the same reasoning on each of the removed coins, we can conclude that $\mathit{Pt}_{\mathit{MAP}}(T, \vec{\pi}) = 1 - \max_i \pi_i$ for any scheduler ζ of T.

Consider now the term T' obtained from T by adding back the coin h:

$$T' = (\nu c_{\ell,h} c_{r,h})(\mathit{Coin}_h \mid T)$$

By applying Theorem 6.2 we can deduce that

$$Pt_{MAP}(T', \vec{\pi}) \geq Pt_{MAP}(T, \vec{\pi})$$

By repeating this reasoning, we can add back all the coins, one by one, and obtain the original DC. Hence we can conclude that

$$Pt_{MAP}(DC, \vec{\pi}) \geq Pt_{MAP}(T, \vec{\pi}) = 1 - \max_i \pi_i$$

and, since $1 - \max_i \pi_i$ is the maximum probability of error,

$$Pt_{MAP}(DC, \vec{\pi}) = 1 - \max_i \pi_i$$

which concludes the proof. □

Interestingly, also the other direction of Theorem 7.2 holds. We report this result for completeness, however we have proved it by using traditional methods, not by applying the compositional methods of Section 6.

Theorem 7.3. A DC is strongly anonymous only if it has a spanning tree consisting of fair coins only.

Proof. By contradiction. Let G be the multigraph associated to the DC and let n be the number of vertices in G. Assume that G does not have a spanning tree consisting only of fair coins. Then it is possible to split G in two non-empty subgraphs, G_1 and G_2, such that all the edges between G_1 and G_2 are unfair. Let $(c_1, c_2, \ldots, c_m)$ be the vector of coins corresponding to these edges. Since G is connected, we have that $m \geq 1$.

Let a_1 be a vertex in G_1 and a_2 be a vertex in G_2. By strong anonymity, for every observable o we have

$$p(o \mid a_1) = p(o \mid a_2) \tag{1}$$

Where a_1, resp. a_2, represents the event that the chryptographer a_1, resp. a_2, is the payer.

Observe now that $p(o \mid a_1) = p(o + w \mid a_2)$ where w is a binary vector of dimension n containing 1 exactly twice, in correspondence of a_1 and a_2, and $+$ is the binary sum. Hence (1) becomes

$$p(o + w \mid a_2) = p(o \mid a_2) \tag{2}$$

Let d be the binary sum of all the elements of o in G_1, and d' be the binary sum of all the elements of $o + w$ in G_1. Since in G_1 w contains 1 exactly once, we have $d' = d + 1$. Hence (2), being valid for all o's, implies

$$p(d + 1 \mid a_2) = p(d \mid a_2) \tag{3}$$

Because of the way o, and hence d, are calculated, and since the contribution of the edges internal to G_1 is 0, and a_2 (the payer) is not in G_1, we have that

$$d = \sum_{i=1}^{m} c_i$$

from which, together with (3), and the fact that the coins are independent from the choice of the payer, we derive

$$p(\sum_{i=1}^{m} c_i = 0) = p(\sum_{i=1}^{m} c_i = 1) = 1/2 \tag{4}$$

The last step is to prove that $p(\sum_{i=1}^{m} c_i = 0) = 1/2$ implies that one of the c_i's is fair, which will give us a contradiction. We prove this by induction on m. The property obviously holds for $m = 1$. Let us now assume that we have proved it for the vector $(c_1, c_2, \ldots, c_{m-1})$. Observe that $p(\sum_{i=1}^{m} c_i = 0) = p(\sum_{i=1}^{m-1} c_i = 0)p(c_m = 0) + p(\sum_{i=1}^{m-1} c_i = 1)p(c_m = 1)$. From (4) we derive

$$p(\sum_{i=1}^{m-1} c_i = 0)p(c_m = 0) + p(\sum_{i=1}^{m-1} c_i = 1)p(c_m = 1) = 1/2 \tag{5}$$

Now, it is easy to see that (5) has only two solutions: one in which $p(c_m = 0) = 1/2$, and one in which $p(\sum_{i=1}^{m-1} c_i = 1) = 1/2$. In the first case we are done, in the second case we apply the induction hypothesis. □

8. Related work

In the field of information flow there have been various works (McLean 1990, Gray, III 1991, Lowe 2002, Clark, Hunt & Malacaria 2001, Clark, Hunt & Malacaria 2005, Malacaria 2007, Malacaria & Chen 2008, Heusser & Malacaria 2009, Smith 2009, Andrés, Palamidessi, van Rossum & Smith 2010, Alvim, Andrés & Palamidessi 2010, Boreale, Pampaloni & Paolini 2011) in which the *high information* and the *low information* are seen as the input and output respectively of a (noisy) channel. Information leakage is formalized in this setting as the channel mutual information or the channel capacity. The idea is that the leakage represents the difference between the a priori uncertainty about the (secret) high information, and the a posteriori uncertainty, after the low information has become pubblically known. The uncertainty is expressed in terms of entropy, and there are various alternative notions depending on the notion of attack that one wishes to model, as discussed in (Köpf & Basin 2007). Most of the above approaches are based either on Shannon entropy or on the Rényi min entropy.

Channel capacity has been also used in relation to anonymity in (Moskowitz, Newman, Crepeau & Miller 2003*b*, Moskowitz, Newman & Syverson 2003*a*). These works propose a method to create covert communication by means of non-perfect anonymity.

A related line of work is (Serjantov & Danezis 2002, Díaz, Seys, Claessens & Preneel 2002), where the main idea is to express the lack of (probabilistic) information in terms of entropy.

A different information-theoretic approach is taken in (Clarkson, Myers & Schneider 2009). In this paper, the authors define as information leakage the difference between the a priori accuracy of the guess of the attacker, and the a posteriori one, after the attacker has made his observation. The accuracy of the guess is defined as the Kullback-Leibler distance between the *belief* (which is a weight attributed by the attacker to each

input hypothesis) and the true distribution on the hypotheses. This approach, that was Shannon-based in (Clarkson et al. 2009), was later applied by Hamadou et al. (Hamadou, Palamidessi & Sassone 2010) also to the case of the Rényi min entropy.

The problem of preservation of information protection under program composition was considered also by McIver et al (McIver et al. 2010). In that paper, the author define an order $\preceq$ on specifications based on Bayes Risk, and they identify a compositional subset of it: a refinement order $\sqsubseteq$ such that $S \sqsubseteq I$ implies $C[S] \preceq C[I]$ for all contexts C. They also show that $\sqsubseteq$ is the *compositional closure* of $\preceq$, in the sense that $S \not\sqsubseteq I$ only when $C[S] \not\preceq C[I]$ for some C. Finally, they prove that $\sqsubseteq$ is sound for other three, competing notions of elementary test and that therefore Bayes-Risk testing, with context, is maximally discriminating among them.

Desharnais et al. (Desharnais, Jagadeesan, Gupta & Panangaden 2002) defined a notion of metric between probabilistic processes and proved that the Shannon capacity associated to a protocol associated to a process is continuous with respect to this metric.

Deng et al. (Deng, Pang & Wu 2006) consider a probabilistic calculus similar to CCS_p, and define a relation between traces based on the Kullback-Leibler divergence. They show that certain constructs of their calculus preserve the relation, in the sense that they do not increase the divergence.

9. Conclusion and future work

In this paper we have investigated the properties of the probability of error associated to a given information-hiding protocol, and we have investigated CCS_pconstructs that are safe, i.e. that are guaranteed not to decrease the protection of the protocol. Then we have applied these results to strengthen a result of Chaum: the dining cryptographers are strongly anonymous if and only if they have a spanning tree of fair coins.

In the future, we would like to extend our results to other constructs of the language. This is not possible in the present setting, as the examples after Theorem 6.2 show. The problem is related to the scheduler: the standard notion of scheduler is too powerful and can leak secrets, by depending on the secret choices that have been made in the past. All the examples after Theorem 6.2 are based on this kind of problem. In (Chatzikokolakis & Palamidessi 2010), we have studied the problem and we came out with a language-based solution to restrict the power of the scheduler. We are planning to investigate whether such approach could be exploited here to guarantee the safety of more constructs.

References

Alvim, M. S., Andrés, M. E. & Palamidessi, C. (2010), Information Flow in Interactive Systems, *in* P. Gastin & F. Laroussinie, eds, 'Proceedings of the 21th International Conference on Concurrency Theory (CONCUR 2010), Paris, France, August 31-September 3', Vol. 6269 of *Lecture Notes in Computer Science*, Springer, pp. 102–116.

Andrés, M. E., Palamidessi, C., van Rossum, P. & Smith, G. (2010), Computing the leakage of information-hiding systems, *in* J. Esparza & R. Majumdar, eds, 'Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2010)', Vol. 6015 of *Lecture Notes in Computer Science*, Springer, pp. 373–389.

Bhargava, M. & Palamidessi, C. (2005), Probabilistic anonymity, *in* M. Abadi & L. de Alfaro, eds, 'Proceedings of CONCUR', Vol. 3653 of *Lecture Notes in Computer Science*, Springer, pp. 171–185.

Boreale, M., Pampaloni, F. & Paolini, M. (2011), Asymptotic information leakage under one-try attacks, *in* M. Hofmann, ed., 'Proceedings of the 14th International Conference on the Foundations of Software Science and Computational Structures (FOSSACS)', Vol. 6604 of *Lecture Notes in Computer Science*, Springer, pp. 396–410.

Chatzikokolakis, K. & Palamidessi, C. (2010), 'Making random choices invisible to the scheduler', *Information and Computation* **208**(6), 694–715.

Chatzikokolakis, K., Palamidessi, C. & Panangaden, P. (2008*a*), 'Anonymity protocols as noisy channels', *Inf. and Comp.* **206**(2–4), 378–401.

Chatzikokolakis, K., Palamidessi, C. & Panangaden, P. (2008*b*), 'On the Bayes risk in information-hiding protocols', *Journal of Computer Security* **16**(5), 531–571.

Chaum, D. (1988), 'The dining cryptographers problem: Unconditional sender and recipient untraceability', *Journal of Cryptology* **1**, 65–75.

Clark, D., Hunt, S. & Malacaria, P. (2001), Quantitative analysis of the leakage of confidential data, *in* 'Proceedings of the Workshop on Quantitative Aspects of Programming Languages', Vol. 59 (3) of *Electr. Notes Theor. Comput. Sci*, Elsevier Science B.V., pp. 238–251.

Clark, D., Hunt, S. & Malacaria, P. (2005), Quantified interference for a while language, *in* 'Proceedings of the Second Workshop on Quantitative Aspects of Programming Languages (QAPL 2004)', Vol. 112 of *Electronic Notes in Theoretical Computer Science*, Elsevier Science B.V., pp. 149–166.

Clarkson, M. R., Myers, A. C. & Schneider, F. B. (2009), 'Belief in information flow', *Journal of Computer Security* **17**(5), 655–701.

Cover, T. M. & Thomas, J. A. (1991), *Elements of Information Theory*, John Wiley & Sons, Inc.

Deng, Y., Palamidessi, C. & Pang, J. (2005), Compositional reasoning for probabilistic finite-state behaviors, *in* A. Middeldorp, V. van Oostrom, F. van Raamsdonk & R. C. de Vrijer, eds, 'Processes, Terms and Cycles: Steps on the Road to Infinity', Vol. 3838 of *Lecture Notes in Computer Science*, Springer, pp. 309–337. `http://www.lix.polytechnique.fr/~catuscia/papers/Yuxin/BookJW/par.pdf`.

Deng, Y., Pang, J. & Wu, P. (2006), Measuring anonymity with relative entropy, *in* T. Dimitrakos, F. Martinelli, P. Y. A. Ryan & S. A. Schneider, eds, 'Proc. of the of the 4th Int. Worshop on Formal Aspects in Security and Trust', Vol. 4691 of *LNCS*, Springer, pp. 65–79.

Desharnais, J., Jagadeesan, R., Gupta, V. & Panangaden, P. (2002), The metric analogue of weak bisimulation for probabilistic processes, *in* 'Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science', IEEE Computer Society, pp. 413–422.

Díaz, C., Seys, S., Claessens, J. & Preneel, B. (2002), Towards measuring anonymity, *in* R. Dingledine & P. F. Syverson, eds, 'Proceedings of the workshop on Privacy Enhancing Technologies (PET) 2002', Vol. 2482 of *Lecture Notes in Computer Science*, Springer, pp. 54–68.

Fujioka, A., Okamoto, T. & Ohta, K. (1993), A practical secret voting scheme for large scale elections, *in* 'ASIACRYPT '92: Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques', Springer-Verlag, London, UK, pp. 244–251.

Gray, III, J. W. (1991), Toward a mathematical foundation for information flow security, *in* 'Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy (SSP '91)', IEEE, Washington - Brussels - Tokyo, pp. 21–35.

Halpern, J. Y. & O'Neill, K. R. (2005), 'Anonymity and information hiding in multiagent systems', *Journal of Computer Security* **13**(3), 483–512.

Hamadou, S., Palamidessi, C. & Sassone, V. (2010), Reconciling belief and vulnerability in information flow, *in* 'Proceedings of the 31st IEEE Symposium on Security and Privacy', IEEE Computer Society, pp. 79–92.

Herescu, O. M. & Palamidessi, C. (2000), Probabilistic asynchronous π-calculus, *in* J. Tiuryn, ed., 'Proceedings of FOSSACS 2000 (Part of ETAPS 2000)', Vol. 1784 of *Lecture Notes in Computer Science*, Springer, pp. 146–160. `http://www.lix.polytechnique.fr/~catuscia/papers/Prob_asy_pi/fossacs.ps`.

Heusser, J. & Malacaria, P. (2009), Applied quantitative information flow and statistical databases, *in* P. Degano & J. D. Guttman, eds, 'Proceedings of the International Workshop on Formal Aspects in Security and Trust', Vol. 5983 of *Lecture Notes in Computer Science*, Springer, pp. 96–110.

Köpf, B. & Basin, D. A. (2007), An information-theoretic model for adaptive side-channel attacks, *in* P. Ning, S. D. C. di Vimercati & P. F. Syverson, eds, 'Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007', ACM, pp. 286–296.

Lowe, G. (2002), Quantifying information flow, *in* 'Proc. of CSFW 2002', IEEE Computer Society Press, pp. 18–31.

Malacaria, P. (2007), Assessing security threats of looping constructs, *in* M. Hofmann & M. Felleisen, eds, 'Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007', ACM, pp. 225–235.

Malacaria, P. & Chen, H. (2008), Lagrange multipliers and maximum information leakage in different observational models, *in* Úlfar Erlingsson and Marco Pistoia, ed., 'Proceedings of the 2008 Workshop on Programming Languages and Analysis for Security (PLAS 2008)', ACM, Tucson, AZ, USA, pp. 135–146.

McIver, A., Meinicke, L. & Morgan, C. (2010), Compositional closure for bayes risk in probabilistic noninterference, *in* S. Abramsky, C. Gavoille, C. Kirchner, F. M. auf der Heide & P. G. Spirakis, eds, 'Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP). Part II.', Vol. 6199 of *Lecture Notes in Computer Science*, Springer, pp. 223–235.

McLean, J. (1990), Security models and information flow, *in* 'SSP'90', IEEE, pp. 180–189.

Milner, R. (1989), *Communication and Concurrency*, International Series in Computer Science, Prentice Hall.

Moskowitz, I. S., Newman, R. E. & Syverson, P. F. (2003*a*), Quasi-anonymous channels, *in* 'Proc. of CNIS', IASTED, pp. 126–131.

Moskowitz, I. S., Newman, R. E., Crepeau, D. P. & Miller, A. R. (2003*b*), Covert channels and anonymizing networks., *in* S. Jajodia, P. Samarati & P. F. Syverson, eds, 'Workshop on Privacy in the Electronic Society 2003', ACM, pp. 79–88.

Palamidessi, C. & Herescu, O. M. (2005), 'A randomized encoding of the π-calculus with mixed choice', *Theoretical Computer Science* **335**(2-3), 373–404. `http://www.lix.polytechnique.fr/~catuscia/papers/prob_enc/report.pdf`.

Reiter, M. K. & Rubin, A. D. (1998), 'Crowds: anonymity for Web transactions', *ACM Transactions on Information and System Security* **1**(1), 66–92.

Rényi, A. (1961), On Measures of Entropy and Information, *in* 'Proceedings of the 4th Berkeley Symposium on Mathematics, Statistics, and Probability', pp. 547–561.

Segala, R. (1995), Modeling and Verification of Randomized Distributed Real-Time Systems, PhD thesis. Tech. Rep. MIT/LCS/TR-676.

Segala, R. & Lynch, N. (1995), 'Probabilistic simulations for probabilistic processes', *Nordic Journal of Computing* **2**(2), 250–273. An extended abstract appeared in *Proceedings of CONCUR '94*, LNCS 836: 481-496.

Serjantov, A. & Danezis, G. (2002), Towards an information theoretic metric for anonymity., *in* R. Dingledine & P. F. Syverson, eds, 'Proceedings of the workshop on Privacy Enhancing Technologies (PET) 2002', Vol. 2482 of *Lecture Notes in Computer Science*, Springer, pp. 41–53.

Smith, G. (2009), On the foundations of quantitative information flow, *in* L. de Alfaro, ed., 'Proc. of the 12th Int. Conf. on Foundations of Software Science and Computation Structures', Vol. 5504 of *LNCS*, Springer, York, UK, pp. 288–302.

Models and Emerging Trends of Concurrent Constraint Programming *

Carlos Olarte† Camilo Rueda ‡ Frank D. Valencia §

June 2013

Abstract

Concurrent Constraint Programming (CCP) has been used over the last two decades as an elegant and expressive model for concurrent systems. It models systems of agents communicating by posting and querying partial information, represented as constraints over the variables of the system. This covers a vast variety of systems as those arising in biological phenomena, reactive systems, net-centric computing and the advent of social networks and cloud computing. In this paper we survey the main applications, developments and current trends of CCP.

1 Introduction

Concurrent Constraint Programming (CCP) [181, 186, 187] is a well-established formalism for concurrency based upon the shared-variables communication model. The CCP model traces its origins back to the ideas of *computing with constraints* [142, 200, 202, 210], *Concurrent Logic Programming* [194, 123] and *Constraint Logic Programming* (CLP) [134, 116, 117]. It was designed for giving programmers and models explicit access to the concept of partial information, traditionally referred to as *constraints*. CCP is intended for reasoning, modeling and programming concurrent and reactive agents (or processes) that interact with each other and their environment by posting and asking information in a medium, a so-called *store*. The CCP formalism enables a unified representation of agents: they can be seen as both computing processes (behavioral imperative style) and as formulas in logic (logical declarative style). Due to this dual view of processes, CCP can benefit from the development and mathematical apparatus of well-established formalisms such as process calculi and logic. In addition, it has been shown as a flexible and versatile framework where several variants can be experimented with and validated.

*Preprint of the paper: *Carlos Olarte, Camilo Rueda, Frank D. Valencia: Models and emerging trends of concurrent constraint programming. Constraints 18(4): 535-578 (2013)*

†Pontificia Universidad Javeriana Cali, Colombia. carlosolarte@puj.edu.co

‡Pontificia Universidad Javeriana Cali, Colombia. crueda@puj.edu.co

§CNRS, LIX, École Polytechnique, France. frank.valencia@lix.polytechnique.fr

The CCP model has received a significant theoretical and implementational attention over the last two decades. In this survey, we shall provide a unified presentation of the most important developments, applications and state of the art in CCP. We aim at giving a broad perspective to the reader about the calculus, its variants, applications and reasoning techniques, as well as recent trends in the area. We start by recalling in Section 2.1 the notion of constraint system that makes CCP a flexible model able to adapt to different application domains. Thereafter, Section 2.2 presents the core language of CCP.

Similar to other mature models of concurrency, such as Petri nets [162, 170] and process calculi [47, 19, 140], CCP has been extended also to capture the behavior of different phenomena in emerging systems. Section 3 presents the developments of CCP calculi in order to deal with: discrete and continuous time and asynchrony for reactive systems; stochastic behavior for physical and biological systems; and linearity (consumption of constraints) for imperative data structures. Furthermore, we shall show the extensions of CCP to cope with: soft constraints where agents can express preferences; communication of local names or links (i.e. mobility) for net-centric computing and protocols; and epistemic and spatial modalities for social networks and cloud computing. In order to give a unified view of these developments, we shall use the same notation of the core language in Section 2.2 that may marginally differ from the original publications. We also present the operational semantics of the calculi and we follow a simple running example to improve the understanding.

Section 4 is devoted to summarize different programming languages whose bases are the CCP model and we also refer some implementations of interpreters for CCP calculi. In Section 5 we show the applicability of CCP and its extensions to model systems in a wide spectrum of scenarios including physical and reactive systems, biological phenomena, multimedia interaction, and net-centric computing.

An appealing feature of the concurrent constraint programming model is that it offers a large set of reasoning techniques for studying the modeled systems. Such techniques come from the theory of process calculi as well as from logic thanks to the declarative reading of CCP agents as logical formulas. In Section 6 we describe the elegant denotational semantics for CCP based on closure operators and the program analysis techniques developed for the language. Furthermore, we summarize the developments of logical characterizations of processes, inference systems and model checking techniques for proving properties of CCP models. We also describe the current efforts to endow CCP with bisimulation reasoning techniques as standardly done in process calculi. Section 7 concludes the paper.

The reader may also refer [95], an article included in the book that celebrates the 25th anniversary of the Italian Association for Logic Programming (GULP) [68]. It presents an overview of the contributions of the Italian research community to CCP.

2 Constraints and Agents

Concurrent Constraint Programming (CCP) [181, 186, 187] has emerged as a simple but powerful paradigm for concurrency tied to logic. Under this paradigm, the conception of *store as valuation* in the von Neumann model is replaced by the notion of

store as constraint, and *processes* are seen as *information transducers*. The CCP model of computation makes use of *ask* and *tell* operations instead of the classical *read* and *write*. An ask operation tests if a given piece of information (i.e., a constraint as in $x > 42$) can be deduced from the store. A *tell* operation conjoins the store with a new constraint to augment the information in it. This is a very general paradigm that extends and subsumes both Concurrent Logic Programming [194, 123] and Constraint Logic Programming [134, 116, 117].

A fundamental issue in CCP is then the specification of concurrent systems by means of constraints that represent partial information about certain variables. The state of the system is specified by the *store* (i.e., a constraint) that is *monotonically refined* by adding new information.

In the spirit of process calculi [47, 19, 140], the language of processes in the CCP model is given by a small number of primitive operators or combinators. A typical CCP process language is equipped with the following operators:

- A *tell* operator adding a constraint to the store.
- An *ask* operator querying if a constraint can be deduced from the store.
- *Parallel Composition* combining processes concurrently.
- A *hiding* operator (also called *restriction* or *locality*) introducing local variables and thus restricting the interface a process can use to interact with others.

Additionally, infinite computations can be described by means of *recursion* or *replication* as we shall see later.

Central to CCP is the notion of *constraint system*. Roughly, a constraint system specifies the basic constraints agents can tell and ask during computation. This makes the language parametric and hence, versatile to be used in different contexts. Next we recall this idea to later introduce the language of CCP processes.

2.1 Constraint Systems

CCP calculi are parametric in a *constraint system*. A constraint system provides a signature from which the constraints can be constructed as well as an entailment relation, notation $\vdash$, specifying inter-dependencies between these constraints. Recall that a constraint represents a piece of information (or partial information) upon which processes may act.

In the literature, the notion of constraint system has been set up in two alternative ways: 1) in terms of Scott's information systems [193] without consistency structure as in [187] and 2) as first-order logic (FOL) formulas as in [180, 195]. In the following we explain these two formulations.

2.1.1 Cylindric Constraint Systems

Processes in CCP can add constraints that lead to an *inconsistent* store. Therefore, it is necessary to represent the possibility of inconsistent information. Constraint systems in this approach are then formalized as algebraic structures with operators to express

conjunction of constraints, absence of information, inconsistent information, hiding of information and parameter passing. More precisely, following the presentation in [41]:

Definition 1 (Constraint System) *A cylindric constraint system is a structure* $\mathbf{C} = \langle \mathcal{C}, \leq, \sqcup, \mathtt{true}, \mathtt{false}, \mathit{Var}, \exists, D \rangle$ *such that*

- $\langle \mathcal{C}, \leq, \sqcup, \mathtt{true}, \mathtt{false} \rangle$ *is a lattice with* $\sqcup$ *the lub operation (representing the logical* and*), and* `true`*,* `false` *the least and the greatest elements in* $\mathcal{C}$*, respectively. Elements in* $\mathcal{C}$ *are called* constraints *with typical elements* $c, d \ldots$.
- Var *is a denumerable set of variables and for each* $x \in \mathit{Var}$ *the function* $\exists x : \mathcal{C} \to \mathcal{C}$ *is a cylindrification operator satisfying: (1)* $\exists x(c) \leq c$*; (2) if* $c \leq d$ *then* $\exists x(c) \leq \exists x(d)$*; (3)* $\exists x(c \sqcup \exists x(d)) = \exists x(c) \sqcup \exists x(d)$*; and (4)* $\exists x \exists y(c) = \exists y \exists x(c)$.
- *For each* $x, y \in \mathit{Var}$*,* $d_{xy} \in D$ *is a* diagonal element *and it satisfies: (1)* $d_{xx} =$ `true`*; (2) if* z *is different from* x, y *then* $d_{xy} = \exists z(d_{xz} \sqcup d_{zy})$ *and (3) If* x *is different from* y *then* $c \leq d_{xy} \sqcup \exists x(c \sqcup d_{xy})$.

The concepts of *cylindrification operators* and *diagonal elements* were borrowed from the theory of cylindric algebras [110]. The cylindrification operator models a sort of existential quantification, helpful for defining the CCP hiding (local) operator and it is assumed to be a continuous function [187, 41]. The diagonal elements are useful to model parameter passing in procedure calls The constraint d_{xy} can be thought of as the equality $x = y$.

The entailment relation ($\vdash$) is the reverse of the ordering $\leq$, i.e., we say that d *entails* c iff $c \leq d$ and we write $d \vdash c$. For operational reasons $\vdash$ is often required to be decidable. The entailment relation is defined in [187] only on the compact elements of $\mathcal{C}$ (finite sets of *tokens*, or basic constraints) while this restriction is not required in [41]. Nevertheless, in both cases, processes are only allowed to ask and tell basic (finite) constraints.

2.1.2 Constraint System as FOL formulas

The notion of constraint system as first order logic (FOL) formulas can be seen as an instance of the previous definition. It gives a logical flavor that has found widespread use in the literature (see e.g., [195, 77, 153, 157]).

Definition 2 (Constraint System as FOL formulas) *A constraint system is a pair* (Σ, Δ) *where* Σ *is a signature of constant, function and predicate symbols, and* Δ *is a first order theory over* Σ *(i.e., a set of non-logical axioms). Let* $\mathcal{L}$ *be the underlying first-order language under* Σ *with variables* $x, y, \ldots$*, and logic symbols* $\wedge, \exists,$ `true` *and* `false`*. Constraints are first-order formulas over* $\mathcal{L}$.

Under this definition, we say that c entails d iff the implication $c \Rightarrow d$ is true in all models of Δ. In the rest of the paper, for the sake of presentation, we shall make use only of this definition of constraint system.

Other realizations of constraint systems have given rise to CCP idioms. For instance, constraints as Girard's linear logic [97] formulas allowed to develop the theory

of linear CCP [77] where agents can *consume* information from the store. Semiring-based constraints [25] allowed to define Soft Concurrent Constraint Programming [29] where agents can post and ask *soft constraints* in order to express preferences, fuzziness, probabilities, etc. Moreover, by adding space functions into the structure of the constraint system, CCP calculi devised for epistemic and spatial reasoning where studied in [124]. We will come back to these definitions in Section 3 where we give an account of different CCP-based calculi.

2.1.3 Examples of Constraint Systems

Let P be a process modeling a temperature controller. Hence, P may have to deal with partial information such as $tsensor > 42$ expressing that the sensor registers an unknown (or not precisely determined) temperature value above 42. Furthermore, P may query if $tsensor > 0$. Then, the constraint system must provide an entailment relation to assert that this information can be inferred, for instance, from the information $tsensor > 42$ (i.e., $tsensor > 42 \vdash tsensor > 0$). In order to deal with such type of constraints, one can consider the finite domain constraint system (FD) [111]. FD variables are assumed to range over finite domains and, in addition to equality, one may have predicates that restrict the possible values of a variable to some finite set.

Another example is the Herbrand constraint system [187] underlying logic programming where a first-order language with equality is assumed. The entailment relation is the one we expect from equality, for instance, $f(x, y) = f(a, g(z))$ must entail $x = a$ and $y = g(z)$.

The reader may find in [187] more examples of constraint systems such as the Kahn Constraint System underlying data-flow languages and the Rational Interval Constraint System. We also point to [65] and [149] that describe, respectively, an implementation of a real interval constraint system and a constraint system to deal with finite sets. Modern constraint solvers as Gecode (`http://www.gecode.org/`) can indeed be used as basis for the implementation of interpreters for CCP languages (see Section 4.7).

2.2 The Language of CCP Processes

Now that we have set up the notion of constraint system, we are ready to define the language of processes. By using the notation in [153]:

Definition 3 (Syntax of CCP [187]) *Processes $P, Q, \ldots$ in CCP are built from constraints in the underlying constraint system by the following syntax:*

$$P, Q := \mathbf{skip} \mid \mathbf{tell}(c) \mid \mathbf{when}\ c\ \mathbf{do}\ P \mid P \parallel Q \mid (\mathbf{local}\ \overline{x}; c)\ P \mid q(\overline{x})$$

The process $\mathbf{skip}$ does nothing and represents inaction. The process $\mathbf{tell}(c)$ adds the constraint c to the store. The process $\mathbf{when}\ c\ \mathbf{do}\ P$ *asks* if c can be deduced from the store. If so, it behaves as P. In other case, it waits until the store contains at least as much information as c. This way, ask agents define a powerful synchronization mechanism based on entailment of constraints.

The parallel composition of the processes P and Q is represented as $P \parallel Q$. The process $(\mathbf{local}\,\overline{x}; c)\,P$ behaves like P, except that all the information on the variables $\overline{x}$ produced by P (represented as c) can only be seen by P and the information on $\overline{x}$ produced by other processes cannot be seen by P. In fact, what the other processes can observe from c is $\exists\overline{x}(c)$ and what P can observe from the store d is $\exists\overline{x}(d)$. If there is no local information (i.e., $c = \mathtt{true}$), it is customary to write $(\mathbf{local}\,\overline{x})\,P$ instead of $(\mathbf{local}\,\overline{x}; \mathtt{true})\,P$.

The process $q(\overline{y})$ is an *identifier* with arity $|\overline{y}|$. We assume that every such an identifier has a unique (recursive) definition of the form $q(\overline{x}) \stackrel{\mathtt{def}}{=} Q$ where $\overline{x}$ are pairwise distinct and $|\overline{x}| = |\overline{y}|$. Then, $q(\overline{y})$ behaves as $Q[\overline{y}/\overline{x}]$. The process $Q[\overline{y}/\overline{x}]$ is actually modeled, with the help of diagonal elements, as $\Delta^{\overline{y}}_{\overline{x}} = (\mathbf{local}\,\overline{z}; d_{\overline{yz}})\,(\mathbf{local}\,\overline{x}; d_{\overline{xz}})\,(Q)$ where $d_{\overline{xy}}$ is the constraint $\bigsqcup_{i\in 1\ldots|\overline{x}|} d_{x_i y_i}$.

2.3 Operational Semantics

The structural operational semantics (SOS) of CCP can be described by means of configurations of the form $\langle P, c\rangle$ where P is a process and c represents the current store. Then, a transition of the form $\langle P, c\rangle \longrightarrow \langle P', c'\rangle$ dictates that P under the store c evolves into P' and produces the store c'. Remember that processes in CCP can only add information. Then, by monotonicity, it must be the case that $c' \vdash c$. Figure 1 shows the SOS for the CCP processes in Definition 3.

Let us explain the Rule $\mathrm{R_L}$ as it may seem somewhat complex. Consider the process $Q = (\mathbf{local}\,x; c)\,P$. The global store is d and the local store is c. We distinguish between the *external* (corresponding to Q) and the *internal* points of view (corresponding to P). From the internal point of view, the information about x, possibly appearing in the "global" store d, cannot be observed. Thus, before reducing P we first hide the information about x that Q may have in d by existentially quantifying x in d. Similarly, from the external point of view, the observable information about x that the reduction of internal agent P may produce (i.e., c') cannot be observed. Thus we hide it by existentially quantifying x in c' before adding it to the global store. Additionally, we make c' the new private store of the evolution of the internal process.

The semantics of the local operator can also be given by using *fresh* variables, i.e., variables that do not appear elsewhere in the processes as in [138, 77, 105]. In this case, configurations are augmented with a set of *eigenvariables*.

Observe that parallel composition is defined above in terms of two rules, noted $\mathrm{R_{PL}}$ and $\mathrm{R_{PR}}$. It is also possible to define parallel composition with a single rule: this is done by internalizing into the semantics a structural congruence relation which equates processes with minor syntactic differences, such as $P \parallel Q$ and $Q \parallel P$ (see e.g., [153]).

An observable (or output) of a process P under input c is then a constraint d such that $\langle P, c\rangle \longrightarrow^* \langle P', d\rangle \not\longrightarrow$ where $\longrightarrow^*$ is the transitive and reflexive closure of the relation $\longrightarrow$. Depending on the semantic framework, one may be interested only in finite computations and/or the limit of infinite ones [187, 41].

We conclude this section with the well known example of concatenation of lists written in CCP taken from [187]:

$$\mathrm{R_T}\ \frac{}{\langle \mathbf{tell}(c), d\rangle \longrightarrow \langle \mathbf{skip}, d \wedge c\rangle}$$

$$\mathrm{R_A}\ \frac{d \vdash c}{\langle \mathbf{when}\ c\ \mathbf{do}\ P, d\rangle \longrightarrow \langle P, d\rangle}$$

$$\mathrm{R_{PL}}\ \frac{\langle P, c\rangle \longrightarrow \langle P', d\rangle}{\langle P \parallel Q, c\rangle \longrightarrow \langle P' \parallel Q, d\rangle}$$

$$\mathrm{R_{PR}}\ \frac{\langle Q, c\rangle \longrightarrow \langle Q', d\rangle}{\langle P \parallel Q, c\rangle \longrightarrow \langle P \parallel Q', d\rangle}$$

$$\mathrm{R_L}\ \frac{\langle P, c \wedge (\exists \overline{x} d)\rangle \longrightarrow \langle P', c' \wedge (\exists \overline{x} d)\rangle}{\langle (\mathbf{local}\, \overline{x}; c)\, P, d\rangle \longrightarrow \langle (\mathbf{local}\, \overline{x}; c')\, P', d \wedge \exists \overline{x} c'\rangle}$$

$$\mathrm{R_C}\ \frac{q(\overline{x}) \stackrel{\mathtt{def}}{=} Q \text{ is a process definition}}{\langle q(\overline{y}), c\rangle \longrightarrow \langle Q[\overline{y}/\overline{x}], c\rangle}$$

Figure 1: Operational semantics for the CCP language in Definition 3.

Example 1 (Append) *Assume the Herbrand constraint system where constraints are existentially quantified conjunctions of equations over a given set of terms (e.g., $L = []$ represents that L is the empty list and $\exists x \exists y (L = [x \mid y])$ that L is a list with head x and tail y). The process below concatenates the lists L_1 and L_2 into L_3:*

$$\begin{array}{rcl} \mathtt{append}(L_1, L_2, L_3) & \stackrel{\mathtt{def}}{=} & \mathbf{when}\ L_1 = [] \ \mathbf{do}\ \mathbf{tell}(L_3 = L_2) \ \parallel \\ & & \mathbf{when}\ \exists x \exists y (L_1 = [x \mid y])\ \mathbf{do}\ \ (\mathbf{local}\, x, y, z)\, (\\ & & \quad \mathbf{tell}(L_1 = [x \mid y]) \parallel \mathbf{tell}(L_3 = [x \mid z]) \parallel \mathtt{append}(y, L_2, z)) \end{array}$$

2.4 Non-determinism

Non-determinism arises in CCP by introducing *guarded choices*.

Definition 4 (Non-deterministic CCP [187]) *Non-determinism is obtained by replacing the ask operator* $\mathbf{when}\ c\ \mathbf{do}\ P$ *with the guarded choice* $\sum_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i$ *where I is a finite set of indexes.*

The process $\sum_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i$ non-deterministically chooses one of the P_j whose corresponding guard (constraint) c_j is entailed by the store. The chosen alternative, if any, precludes the others. If no choice is possible then the summation remains blocked until more information is added to the store:

$$\mathrm{R_{SUM}}\ \frac{d \vdash c_j \quad j \in I}{\langle \sum_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i, d\rangle \rightarrow \langle P_j, d\rangle}$$

Example 2 (Consumer-producer streams) *Let* `append` *be as in the Example 1 and assume the following process definitions*

$$\begin{array}{lcl} \mathtt{prod}_a(x) & \stackrel{\mathtt{def}}{=} & \mathbf{when}\ \mathtt{true}\ \mathbf{do}\ (\mathbf{local}\, x')\,(\mathbf{tell}(x = [a|x']) \parallel \mathtt{prod}_a(x')) + \\ & & \mathbf{when}\ \mathtt{true}\ \mathbf{do}\ x = [] \\ \mathtt{prod}_b(x) & \stackrel{\mathtt{def}}{=} & \mathbf{when}\ \mathtt{true}\ \mathbf{do}\ (\mathbf{local}\, x')\,(\mathbf{tell}(x = [b|x']) \parallel \mathtt{prod}_b(x')) + \\ & & \mathbf{when}\ \mathtt{true}\ \mathbf{do}\ x = [] \end{array}$$

The process $\mathtt{prod}_a(x) \parallel \mathtt{prod}_b(y) \parallel \mathtt{append}(x, y, z)$ *binds* x *and* y *to a (possibly infinite) list of* a*'s and* b*'s respectively and* z *to the concatenation of* x *and* y*.*

3 Constraint-Based Concurrent Calculi

Several extensions of the CCP model have been studied in order to provide frameworks for the programming, specification and verification of systems with the declarative flavor of concurrent constraint programming. This section is devoted to describe these developments.

3.1 Timed Concurrent Constraint Programming

Reactive systems [20, 107] are those that react continuously with their environment at a rate controlled by the environment. For example, a controller or a signal-processing system, receives a stimulus (input) from the environment, computes an output and then waits for the next interaction with the environment.

In CCP, the shared store of constraints grows monotonically, i.e., agents cannot drop information (constraints) from it. Then, a system that changes the state of a signal (i.e., the value of a variable) cannot be straightforwardly modeled: the conjunction of the constraints "$signal = on$" and "$signal = off$" leads to an inconsistent store [1].

Timed CCP (`tcc`) [182] extends CCP for reactive systems and elegantly combines CCP with ideas from the paradigm of Synchronous Languages [20]. Time in `tcc` is conceptually divided into *time intervals* (or *time-units*). In a particular time interval, a CCP process P gets an input c from the environment, it executes with this input as the initial *store*, and when it reaches its resting point, it *outputs* the resulting store d to the environment. The resting point determines also a residual process Q that is then executed in the next time-unit. The resulting store d is not automatically transferred to the next time-unit. This way, computations during a time-unit proceed monotonically but outputs of two different time-units are not supposed to be related. Therefore, the variable $signal$ in the example above may change its value when passing from one time-unit to the next one.

This view of reactive computation is particularly appropriate for programming reactive systems such as robotic devices and micro-controllers. These systems typically operate in a cyclic fashion, i.e., in each cycle they receive an input from the environment, compute on this input, and then return the corresponding output to the environment.

[1] It is possible however to make use of streams to represent changes in the value of a variable.

The tcc calculus extends the language of CCP processes with constructs that: (1) *delay* the execution of a process; and (2) *time-out* (or weak pre-emption) operations that wait during the current time interval for a given piece of information to be present, if it is not, they trigger a process in the next time interval.

Definition 5 (Deterministic tcc [182]) *The syntax of* tcc *is obtained by adding to Definition 3 the processes* $\mathbf{next}\ P$ *and* $\mathbf{unless}\ c\ \mathbf{next}\ P$.

The process $\mathbf{next}\ P$ delays the execution of P to the next time interval. The *time-out* $\mathbf{unless}\ c\ \mathbf{next}\ P$ is also a unit-delay, but P is executed in the next time-unit only if c is not entailed by the final store at the current time interval. Notice that, in general, $P = \mathbf{unless}\ c\ \mathbf{next}\ R$ is not the same as $Q = \mathbf{when}\ \neg c\ \mathbf{do}\ \mathbf{next}\ R$. Take for instance $x = 1$ as the constraint c. From the store true, one cannot deduce neither $x = 1$ nor $x \neq 1$. In this particular case, the process P executes R (provided that the final store is true) while Q remains blocked.

In tcc, recursive calls must be guarded by a **next** operator to avoid infinite computations during a time-unit. If the temporal language does not consider recursive definitions, it is customary to add the *replication* $!\,P$ in order to give processes the possibility to be executed in infinitely many time-units. The *replication* $!\,P$ means $P \parallel \mathbf{next}\ P \parallel \mathbf{next}^2 P \parallel \ldots$, i.e., unboundedly many copies of P but one at a time. The reader may refer to [152] for a detailed account of the expressiveness power of temporal CCP languages with and without recursion and replication.

In spite of its simplicity, the tcc extension to CCP is far-reaching. Many interesting temporal constructs can be expressed as it is shown in [182]. As an example, tcc allows processes to be "clocked" by other processes. This provides meaningful pre-emption constructs and the ability of defining multiple forms of time instead of only having a unique global clock.

The SOS of tcc considers *internal* and *observable* transitions. The internal transitions are similar to those in Figure 1 and correspond to the operational steps that take place during a time-unit. As for the timed operators, the **unless** process evolves into **skip** if its guard can be entailed from the current store:

$$\mathrm{R_U}\ \frac{d \vdash c}{\langle \mathbf{unless}\ c\ \mathbf{next}\ P, d\rangle \longrightarrow \langle \mathbf{skip}, d\rangle}$$

The seemingly missing rule for the **next** operator will be clarified soon.

The *observable transition* $P \overset{(c,d)}{\Longrightarrow} Q$ should be read as "P on input c, reduces in one *time-unit* to Q and outputs d". The observable transitions are obtained from finite sequences of internal ones, i.e., $P \overset{(c,d)}{\Longrightarrow} Q$ whenever $\langle P, c\rangle \longrightarrow^* \langle R, d\rangle \not\longrightarrow$. The process Q (the continuation for the next time-unit) is obtained from:

$$F(R) = \begin{cases} \mathbf{skip} & \text{if } R = \mathbf{skip} \text{ or } R = \mathbf{when}\ c\ \mathbf{do}\ R' \\ F(R_1) \parallel F(R_2) & \text{if } R = R_1 \parallel R_2 \\ (\mathbf{local}\ \overline{x})\ F(R') & \text{if } P = (\mathbf{local}\ \overline{x}; c)\ R' \\ Q & \text{if } R = \mathbf{next}\ R' \text{ or } R = \mathbf{unless}\ c\ \mathbf{next}\ R' \end{cases}$$

The function $F(R)$ (the future of R) unfolds *next* and *unless* expressions. Notice that an *ask* process reduces to **skip** if its guard was not entailed by the final store d. Notice

also that F is not defined for $\mathbf{tell}(c)$, $!\,Q$ or $p(\overline{x})$ processes since they must occur within a $\mathbf{next}$ or $\mathbf{unless}$ expression.

3.1.1 Strong Pre-Emption in Timed CCP

Modeling some systems may require detecting negative information (i.e., the *absence* of information) and react *instantaneously*. Take for instance a process P that needs to be aborted to *immediately* start a process Q when a specific event occurs. This is usually called pre-emption of P. The work in [183] introduces the notion of strong pre-emption in `tcc`: on the absence of a constraint, time-out operations can trigger activity in the current time interval rather than delaying it to the next interaction with the environment as in `tcc`. Borrowing ideas from default logics [171], **Default** `tcc` is defined as follows:

Definition 6 (Default `tcc` [183]) *The syntax of* **Default** `tcc` *is obtained by adding to Definition 5 the constructor* $\mathbf{when}\ c\ \mathbf{else}\ P$.

Unlike $\mathbf{unless}\ c\ \mathbf{next}\ P$ in `tcc`, the process $\mathbf{when}\ c\ \mathbf{else}\ P$ reduces to P at the present time instant if c has not been produced in the store, and will not be produced throughout system execution. The transition system is then parametrized with a constraint e, representing the final "guess" that is used to evaluate defaults:

$$\frac{d \vdash c}{\langle \mathbf{when}\ c\ \mathbf{else}\ P, d\rangle \longrightarrow_e \langle \mathbf{skip}, d\rangle}\ \mathrm{R_{DA}} \qquad \frac{e \not\vdash c}{\langle \mathbf{when}\ c\ \mathbf{else}\ P, d\rangle \longrightarrow_e \langle P, d\rangle}\ \mathrm{R_{DA'}}$$

Example 3 (Default values for variables [183]) *The following program:*

$$\mathtt{default}(x, v) \stackrel{\mathtt{def}}{=} \mathbf{when}\ x \neq v\ \mathbf{else}\ x = v$$

sets the value of x to be v unless the current value of x is different from v.

3.1.2 Non-Determinism and Asynchrony in `tcc`

The notion of *non-determinism* and *asynchrony* is ubiquitous in concurrent systems. As for non-determinism, a system may exhibit different behaviors when reacting to the same input. As for asynchrony, we may have processes that occur at some point in the future time but we do not know exactly when as in a failure model for a component that is doomed to fail. The `ntcc` calculus [153] extends `tcc` by adding guarded-choices for modeling non-deterministic behavior and an unbounded finite-delay operator for asynchronous behavior. Computation in this language progresses as in `tcc`, except for the non-determinism induced by the new constructs:

Definition 7 (`ntcc` Processes [153]) *The `ntcc` processes result from adding to the syntax in Definition 5 two constructs,* $\sum_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i$ *and* $\star P$.

The guarded-choice is similar to that in Definition 4. The operator "$\star$" corresponds to the unbounded but finite delay operator ϵ for synchronous CCS [139] and it allows

to express asynchronous behavior through the time intervals. Intuitively, the process $\star P$ represents $P + \mathbf{next}\, P + \mathbf{next}\,^2 P + ...$, i.e., an arbitrary long but finite delay for the activation of P:

$$\mathrm{R_S}\ \frac{n \geq 0}{\langle \star P, d\rangle \rightarrow \langle \mathbf{next}\,^n P, d\rangle}$$

Example 4 (Controller for a Robot) *Assume a controller for a robot where the environment sends signals when the device has to turn. Let*

$$\texttt{TurnR} \stackrel{\text{def}}{=} \mathbf{when}\ dir = \texttt{N}\ \mathbf{do}\ \mathbf{next}\ \mathbf{tell}(dir = \texttt{E}) + \mathbf{when}\ dir = \texttt{E}\ \mathbf{do}\ \mathbf{next}\ \mathbf{tell}(dir = \texttt{S}) + \mathbf{when}\ dir = \texttt{S}\ \mathbf{do}\ \mathbf{next}\ \mathbf{tell}(dir = \texttt{W}) + \mathbf{when}\ dir = \texttt{W}\ \mathbf{do}\ \mathbf{next}\ \mathbf{tell}(dir = \texttt{N})$$

be a process that changes in the next time-unit the value of dir according to its current value. The process `TurnL` *can be defined similarly. The process below specifies that the robot turns left or right when the environment provides the signal* `turn`*:*

$$\texttt{Robot} \stackrel{\text{def}}{=} \mathbf{next}\ \texttt{Robot} \parallel \mathbf{when}\ \texttt{turn}\ \mathbf{do}\ \texttt{TurnR} + \mathbf{when}\ \texttt{turn}\ \mathbf{do}\ \texttt{TurnL} \parallel \sum_{i \in \{\texttt{N},\texttt{E},\texttt{S},\texttt{W}\}} \mathbf{when}\ dir = i\ \mathbf{do}\ \mathbf{unless}\ \texttt{turn}\ \mathbf{next}\ dir = i$$

The summation above specifies that the direction in the next time-unit is the same as in the current one unless the constraint `turn` *can be deduced.*

3.1.3 The `tccp` Language

In the `tcc` and `ntcc` models of computation, stores are not automatically transferred to the next time-unit. Therefore, the store is monotonically refined in each time-unit but outputs of two different time-units are not supposed to be related to each other. The `tccp` process calculus [34] is an orthogonal timed non-deterministic extension of CCP. In this language, time is identified with the time needed to ask and tell information to the store. Furthermore, unlike `tcc`, the information in the store is carried through the time-units. Then, streams are used in `tccp` to model the evolution of variable values along the time.

Definition 8 (`tccp` Processes [34]) *The syntax of* `tccp` *is obtained by adding the construct* $\mathbf{when}\ c\ \mathbf{then}\ P\ \mathbf{else}\ Q$ *to the language in Definition 4.*

The `tccp` calculus introduces a discrete global clock and assumes that ask and $tell$ actions take one time-unit. Computation evolves in steps of one time-unit, so called clock-cycles, that are syntactically separated by action prefixing. Moreover, maximal parallelism is assumed, that is, at each moment every enabled agent of the system is activated:

$$\mathrm{R_P}\ \frac{\langle P_1, c\rangle \longrightarrow \langle P_1', c_1\rangle \quad \langle P_2, c\rangle \longrightarrow \langle P_2', c_2\rangle}{\langle P_1 \parallel P_2, c\rangle \longrightarrow \langle P_1' \parallel P_2', c_1 \wedge c_2\rangle}$$

Let $P = \mathbf{when}\ c\ \mathbf{then}\ Q\ \mathbf{else}\ R$. This process queries if c can be deduced from the current store. If so, Q is executed. Otherwise, unlike the **unless** process in `tcc`,

P reduces to R *instantaneously*:

$$\mathrm{R}_{\mathrm{NE}}\ \frac{\langle Q, d\rangle \longrightarrow \langle Q', d'\rangle \quad d \vdash c}{\langle \mathbf{when}\ c\ \mathbf{then}\ Q\ \mathbf{else}\ R, d\rangle \longrightarrow \langle Q', d'\rangle}$$

$$\mathrm{R}'_{\mathrm{NE}}\ \frac{\langle R, d\rangle \longrightarrow \langle R', d'\rangle \quad d \nvdash c}{\langle \mathbf{when}\ c\ \mathbf{then}\ Q\ \mathbf{else}\ R, d\rangle \longrightarrow \langle R', d'\rangle}$$

Analogous rules are given when Q and R cannot evolve on the store d.

Example 5 (Time-outs) *Assume now that the controller in Example 4 must emit the signal* `stop` *when the environment does not provide the signal* `turn` *after* n *time-units. This can be done by defining inductively* $\sum_{i\in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i\ \mathbf{time\text{-}out}(\mathbf{m})\ Q$ *that behaves as* Q *if after* m *time-units none of the guards* c_i *can be entailed. Let* $P = \sum_{i\in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i$. *When* $m = 0$, *the time-out constructor is defined as*

$$\mathbf{when}\ c_1\ \mathbf{then}\ P\ \mathbf{else}\ \mathbf{when}\ c_2\ \mathbf{then}\ P\ \mathbf{else}\ ...\mathbf{when}\ c_n\ \mathbf{then}\ P\ \mathbf{else}\ Q$$

As for $m > 0$, *we have:*

$$\sum_{i\in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i\ \mathbf{time\text{-}out}(\mathbf{m})\ (\sum_{i\in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i\ \mathbf{time\text{-}out}(\mathbf{m-1})\ Q)$$

In our particular case, I *is a singleton,* $c_i = \mathtt{turn}$, $m = n$ *and* $Q = \mathbf{tell}(\mathtt{stop})$.

`tccp` programs usually require to perform arithmetic calculations in order to output a signal to the environment. Even though the underlying constraint system may support some basic arithmetic, it is not realistic to assume that it implements any computable function. One may think of implementing such functions as `tccp` processes. Nevertheless, these computations would consume an unspecified amount of time-units, making the synchronization of processes more difficult. In [4], `tccp` is extended to consider *external* functions written in a functional language. Processes can evaluate *instantaneously* a function by means of an external implementation, thus avoiding the burden of computing it as a `tccp` process.

3.1.4 Continuous Time in CCP

The Hybrid concurrent constraint programming model, *Hybrid cc* (`hcc`) [101], is a calculus that can express discrete and continuous evolution of time. More precisely, there are points at which discontinuous change may occur (i.e. the execution proceeds as a burst of activity) and open intervals in which the state of the system changes continuously (i.e. the system evolves continuously and autonomously).

In `hcc` the notion of constraint system is extended with differential equations and the entailment relation is defined in order to solve initial value problems. Hence, *continuous constraint systems* allows for describing the continuous evolution of time. For instance, the constraint $\mathtt{init}(x = 0)$ means that the initial value of x is 0 and $\mathtt{cont}(dot(x) = 1)$ that the first derivative of x is 1. Then, it is possible to infer that $x = t$ in time t.

Definition 9 (hcc Processes [101]) *The* hcc *processes result from adding to the syntax in Definition 6 the construct* **hence** P.

The constructor **hence** P specifies a process where P holds continuously beyond the current instant. Therefore, if **hence** P is invoked at time t, a copy of P is created at each time in the time interval (t, ∞). In combination with the other constructs in **Default** tcc, various patterns of temporal activity can be generated.

Example 6 (Controlling the speed of the robot) *Assume that the variable v controls the speed of the robot in Example 4. The following process reduces the speed of the robot when it receives the signal* stop*:*

$$\mathit{speed\text{-}control} \stackrel{\text{def}}{=} \textbf{when } \mathit{stop} \textbf{ do hence when } v > 0 \textbf{ do tell}(\mathit{cont}(\mathit{dot}(v) = -1))$$

The reader can find in [99] a survey of the developments of the temporal extensions of tcc, **Default** tcc and hcc and the relationships between their semantic models. We also point the reader to [189] where a CCP calculus based on the notion of real time is proposed. The rtcc calculus allows to deal with strong preemption and delay declarations in the lines of ntcc. Furthermore, the transition system is extended to specify true concurrency and the resources that processes require to be executed.

3.2 Probabilistic Behavior

Non-deterministic choices in a model leave completely unspecified the way the system may react to a given input. In some cases, models can be refined when information about the propensity of an action to occur is available. For instance, we could have information about the probability of a system component to fail. In [100] a probabilistic model for CCP and tcc is studied where random variables with a given probability distribution are introduced. The resulting languages, *probabilistic* CCP and *probabilistic* tcc (pcc, ptcc), allow programs to make stochastic moves during execution, so that they may be seen as stochastic processes.

Definition 10 (Probabilistic CCP and tcc [100]) *The syntax of* pcc *(resp.* ptcc*) are obtained by adding to Definition 3 (resp. 5) the constructor* **new** (x, f) **in** P *where x is a variable and f is its probability mass function.*

The constructor introduced above chooses a value for x according to f:

$$\mathrm{R}_{\mathrm{PROB}} \ \frac{f(r) > 0 \qquad y \text{ is a fresh variable}}{\langle \textbf{new } (x, f) \textbf{ in } P, d\rangle \longrightarrow \langle P[y/x], d \wedge y = r\rangle}$$

Example 7 (Random Zigzag) *Assume that the robot in Example 4 takes autonomously the decision of turning right or left:*

$$\begin{aligned} \mathit{choice} \stackrel{\text{def}}{=} \ & \textbf{new } (x, f : f(0) = f(1) = 0.25, f(2) = 0.5) \textbf{ in} \\ & \textbf{when } x = 0 \textbf{ do } \mathit{TurnR} \parallel \textbf{when } x = 1 \textbf{ do } \mathit{TurnL} \end{aligned}$$

The agent above with a probability of 0.25 calls the procedure TurnR*. Similarly for* TurnL*. Moreover, with a probability of 0.5, the directions does not change.*

An interesting feature of pcc is the use of constraints to add information about random variables. This can be used to declaratively modify the probabilities of the possible execution paths as the following example shows.

Example 8 (Random Zigzag Revisited) *Consider the process*

$$\texttt{choice} \stackrel{\text{def}}{=} \mathbf{new}\ (x, f : f(0) = f(1) = 0.25, f(2) = 0.5)\ \mathbf{in}\ \mathbf{tell}(x \in \{0,1\}) \parallel \mathbf{when}\ x = 0\ \mathbf{do}\ \texttt{TurnR} \parallel \mathbf{when}\ x = 1\ \mathbf{do}\ \texttt{TurnL}$$

Here we restrict the variable x to take a value in the set $\{0,1\}$. Then, the choice $x = 2$ is precluded and, normalizing f, we observe the calls TurnL *and* TurnR *with the same probability of* 0.5.

The results of [100] are extended in [98] where pcc processes with recursion are considered. The authors show that due to the combination of probabilities and constraints (as in Example 8), the interpretations of procedure calls are not necessarily monotonic and then, the semantics cannot be compositional. The authors extend the operational semantics with labels indicating the needed constraint to perform a transition and a denotational semantics based on weak bisimulation on such transition system is shown to be a congruence. The works in [161, 44, 9] study also stochastic extensions for CCP. Although the calculi developed in those works differ in the constructs provided and the semantic treatment, all of them aim at representing the stochastic nature of the modeled system. The semantics of probabilistic extensions of CCP has been also studied in [164, 165]. Furthermore, a framework for the analysis of probabilistic programs is developed in [166].

3.3 Linear Concurrent Constraint Programming

Linear CCP (lcc) [179, 77, 22, 23, 105] is a variant of CCP where constraints are built from a *linear constraint system* based on Girard's intuitionistic linear logic (ILL) [97].

Definition 11 (Linear Constraint Systems [77]) *A linear cs is a pair $(\mathcal{C}, \vdash)$ where $\mathcal{C}$ is a set of formulas (linear constraints) built from a signature Σ, a denumerable set of variables $\mathcal{V}$ and the following ILL operators:* multiplicative *conjunction ($\otimes$) and its neutral element (1), the existential quantifier ($\exists$) and the exponential bang ($!$). Let Δ be a (possibly empty) subset of $\mathcal{C} \times \mathcal{C}$ defining the non-logical axioms of the constraint system (i.e, a theory). Then the entailment relation $\vdash$ is the least set containing Δ and closed by the rules of ILL.*

The bang connective allows to recover the classical constraint system by writing all constraints preceded by "!".

The language of lcc processes is similar to that of CCP but variables in ask agents can be universally quantified.

Definition 12 (Linear CCP [77]) *Processes in* lcc *are built from constraints in a linear constraint system and the syntax in Definition 4 where ask agents are of the form $\forall \overline{x}(\mathbf{when}\ c_i\ \mathbf{do}\ P_i)$.*

The linear tell operator $\mathbf{tell}(c)$ augments the current store d to $d \otimes c$. Furthermore, the linear ask $\mathbf{when}\ c\ \mathbf{do}\ P$ evolves into P whenever there exists e s.t. d entails $c \otimes e$. When this happens, the constraint c is *consumed*:

$$\mathrm{R_{LA}}\ \frac{d \vdash c[t/x] \otimes d'}{\langle \forall x(\mathbf{when}\ c\ \mathbf{do}\ P), d\rangle \longrightarrow \langle P[t/x], d'\rangle}$$

Due to the removal of information, `lcc` is intrinsically non-deterministic and the constraint d' above has to be carefully chosen to avoid the unwanted weakening of the store as in $!c \vdash c$ [106, 137, 105]. Nevertheless, since `lcc` is not monotonic (in the sense that the information in the store can be dropped), this model introduces some forms of imperative programming particularly useful for reactive systems. For instance, imperative data structures are encoded directly with linear constraints instead of streams.

Example 9 (Access Permissions) *Assume a constraint system with the constant symbols* `unq` *(unique) and* `shr` *(share) and with the ternary predicate* $\mathtt{acc}(x, r, p)$ *(agent* x *is currently accessing the resource* r *with permission* $p \in \{\mathtt{unq}, \mathtt{shr}\}$*). Assume also that the constraint system is equipped with the axiom* $\Delta = \mathtt{acc}(x, r, \mathtt{unq}) \vdash \mathtt{acc}(x, r, \mathtt{shr})$*. In words, if* x *makes use of* r *with permission* `unq`*, it can* downgrade *this permission to* `shr`*. Let* $R = P \parallel Q$ *where*

$$P \quad = \quad \mathbf{tell}(\mathtt{acc}(x, r, \mathtt{unq})) \qquad\qquad Q \quad = \quad \forall y(\mathbf{when}\ \mathtt{acc}(x, y, \mathtt{shr})\ \mathbf{do}\ Q')$$

Roughly speaking, P *adds to the store the information required to state that* x *uses* r *and has a unique permission on it. Thereafter,* Q consumes *this information (by using* Δ*), leading to the store* $\top$ *(i.e.,* x *has no longer access to* r*) where the agent* $Q'[r/y]$ *is executed.*

3.4 Soft Concurrent Constraint Programming

Soft constraints [25, 30] have been introduced in constraint programming in order to deal with different levels of consistency. The framework of semiring-based constraints [25] gives a general setting where, according to an algebraic structure, it is possible to represent preferences, fuzziness, probabilities and uncertainty in constraint satisfaction problems (CSP).

The authors in [29] propose Soft Concurrent Constraint Programming (`scc`). The key idea is to replace the (crisp) constraint system in Definition 1 by a semiring-based constraint system. Let us first recall the notion of c-semirings.

Definition 13 (C-Semiring [25]) *A semiring is a tuple* $\langle \mathcal{A}, +, \times, 0, 1\rangle$ *where (1)* $\mathcal{A}$ *is a set and* $0, 1 \in \mathcal{A}$*. (2)* $+$ *is commutative, associative and* 0 *is its unit element. (3)* $\times$ *is associative, distributes over* $+$*,* 1 *is its unit element and* 0 *is its absorbing element. A c-semiring is a semiring such that* $+$ *is idempotent,* 1 *is its absorbing element and* $\times$ *is commutative.*

Intuitively, $+$ is the lub operator and it is used to choose the *best* constraint: $a \leq b$ iff $a + b = b$. The $\times$ operator allows for combining constraints. Here it is important to notice that combining more constraints leads to a *worse* level of consistency, that is,

$c \times c' \leq c$ unlike in Definition 1 where $c \leq c \sqcup c'$ (i.e., $c \sqcup c' \vdash c$). As an example, fuzzy constraint satisfaction problems [69] can be modeled and solved by using the c-semirings $\langle [0,1], max, min, 0, 1 \rangle$.

By defining suitable operations on the semiring, it is possible to define a cylindric algebra leading to a *soft constraint system* as shown in [29]. Then, CCP agents can tell and ask soft constraints. Furthermore, such agents can define thresholds that express the level of consistency of the store. This allows the programmer to specify that an action (tell or ask) is executed only if it does not decrease the consistency level to a given lower bound.

3.4.1 Timed Soft Concurrent Constraint Programming

Following the approach of `tccp`, a timed extension of `scc` (`stcc`) was proposed in [28]. As in `tccp`, action-prefixing in `stcc` is interpreted as the next-time operator and the parallel execution of agents follows the scheduling policy of maximal parallelism.

Definition 14 (`stcc` Processes [28]) *Agents in `stcc` are built from:*

$$\begin{array}{rcl} P, Q & := & \mathbf{skip} \mid \mathbf{tell}(c) \rightsquigarrow P \mid \sum_{i \in I} \mathbf{when}\ c_i \ \rightsquigarrow\ P_i \mid P \parallel Q \mid (\mathbf{local}\, \overline{x}; c)\, P \mid q(\overline{x}) \\ & & \mathbf{when}_{\Phi}\ c\ \mathbf{then}\ P\ \mathbf{else}\ Q \mid \mathbf{when}^{a}\ c\ \mathbf{then}\ P\ \mathbf{else}\ Q \end{array}$$

Here, $\rightsquigarrow$ can be either $\rightarrow_{\Phi}$ or $\rightarrow^{a}$ where a is a semiring element and Φ a constraint.

Intuitively, the semiring value a and the constraint Φ are used as a cut level to prune branches of computation that are not satisfactory. For instance, the process $\mathbf{tell}(c) \rightarrow^{a} P$ adds c to the current store d, if the conjunction (based on the $\times$ operator of the semiring) of d and c is *better* (with respect to $+$) than a.

Example 10 (Bounded Zigzag) *Let us choose the Fuzzy c-semiring and assume now that the robot in Example 7 zigzags according to* preferences*:*

$$\begin{array}{rcl} \mathit{choice} & \stackrel{\text{def}}{=} & \mathbf{when}\ \mathtt{true} \rightarrow^{0.1} \mathit{TurnR} + \mathbf{when}\ \mathtt{true} \rightarrow^{0.3} \mathit{TurnL} + \\ & & \mathbf{when}\ \mathtt{true} \rightarrow\ \mathbf{skip} \end{array}$$

The process above can always do nothing ($\mathbf{skip}$*) and then, the direction of the robot remains the same. Moreover, assume that each time it turns left or right, we add a constraints and a* penalization *(in terms of the semiring value) is paid. Then, the number of times the robot chooses to turn is confined according to the thresholds* 0.1 *and* 0.3*.*

3.5 Mobile Behavior

Process calculi such as the π-calculus [140] allow to specify *mobile* systems, i.e., systems where agents can communicate their local names. Unlike the π-calculus (that is based on point-to-point communication), interaction in CCP is *asynchronous* as communication takes place thorough the shared store. In the CCP model it is possible to specify *mobility* in the sense of *reconfiguration* of the communication structure of the program. This can be done by using logical variables that represent communication channels and unification to bind messages to channels [181]. Since logical variables

can be bound to a value only once, if two messages are sent through the same channel, then they must be equal to avoid an inconsistent store. This problem was addressed in [127] by considering *atomic* tells where the constraint c in $\mathbf{tell}(c)$ is added to the store d if the conjunction $c \wedge d$ is consistent. Channels are then represented as imperative style variables by binding them to streams. Therefore, a protocol is required since messages must compete for a position in such a stream.

The following two sections describe two alternative approaches to endow CCP with mechanisms to communicate private channels or links.

3.5.1 The `cc-pi` Calculus

The `cc-pi` calculus [50] results from the combination of the CCP model with a name-passing calculi. More precisely, `cc-pi` extends CCP by adding synchronous communication and by providing a treatment of names in terms of restriction and structural axioms closer to nominal calculi than to variables with existential quantification.

Definition 15 (`cc-pi` Processes [50]) *Processes in `cc-pi` are built from c-semiring based constraints as follows:*

$$\begin{array}{lcl} P, Q & := & \mathbf{skip} \mid P \parallel Q \mid \sum_{i \in I} \pi_i.P_i \mid (\mathbf{local}\, x; c)\, P \mid p(x) \\ \pi & := & \tau \mid \overline{x}\langle \tilde{y} \rangle \mid x(\tilde{y}) \mid \mathbf{tell}(c) \mid \mathbf{when}\ c \mid \mathbf{retract}\ c \mid \mathbf{check}\ c \end{array}$$

In this calculus, tell and ask actions are prefixing much like in `stcc`. The name passing discipline of `cc-pi` is reminiscent to that in the pi-F calculus [211] whose synchronization mechanism is global and, instead of binding formal names to actual names, it yields explicit fusions, i.e., simple constraints expressing name equalities.

Example 11 (Name passing) *Assume two components P and Q of a system such that P creates a local variable that must be shared with Q. This system can be modeled as:*

$$P = (\mathbf{local}\, y)\, (\bar{x}\langle y \rangle.P') \qquad\qquad Q = x\langle z \rangle.Q'$$

In $P \parallel Q$, P sends *the* private *name y on channel x and synchronizes with Q leading to $(\mathbf{local}\, x)\, (P' \parallel Q' \parallel \mathbf{tell}(y = z))$.*

Similar to the non-monotonic extension of `scc` reported in [31], `cc-pi` also introduces retraction of constraints ($\mathbf{retract}\ c$) whose effect is to erase a previously told constraints. Furthermore, it is possible to check if a given constraint c is consistent with the current store though the prefix $\mathbf{check}\ c$.

Another line of development in this direction was the π^+-calculus [66]. This language is an extension of the π-calculus [140] with constraint agents that can perform tell and ask actions. Similarly as in `cc-pi`, mobility of π^+comes from the operands inherited from the π-calculus.

3.5.2 Universal Timed CCP

Universal Timed CCP (`utcc`) was proposed in [158] as an orthogonal extension of `tcc` for the specification of mobile reactive systems as security protocols. Basically,

utcc replaces the ask operation $\mathbf{when}\ c\ \mathbf{do}\ P$ by a *parametric ask* constructor of the form $(\mathbf{abs}\ \overline{x}; c)\ P$. This process can be viewed as an *abstraction* of the process P on the variables $\overline{x}$ under the constraint (or with the *guard*) c.

Definition 16 (utcc Processes [158]) *The utcc processes result from replacing in the syntax in Definition 5 the expression* $\mathbf{when}\ c\ \mathbf{do}\ P$ *with* $(\mathbf{abs}\ \overline{x}; c)\ P$ *where variables in* $\overline{x}$ *are pairwise distinct.*

Operationally, $(\mathbf{abs}\ \overline{x}; c)\ P$ executes $P[\overline{t}/\overline{x}]$ in the current time interval for *all the terms* $\overline{t}$ s.t $c[\overline{t}/\overline{x}]$ is entailed by the current store. This construct is akin to replicated asks in lcc that can replace process declarations [106, 105].

Example 12 (Mobile behavior in utcc) *Assume an uninterpreted binary predicate* out*. The system in Example 11 can be specified in* utcc *as:*

$$P = (\mathbf{local}\, y)\,(\mathbf{tell}(\mathtt{out}(x, y)) \parallel P') \qquad\qquad Q = (\mathbf{abs}\ z; \mathtt{out}(x, z))\, Q'$$

The SOS of utcc *dictates that the process above evolves into* $(\mathbf{local}\, x)\,(P' \parallel Q'[y/z])$ *where* P' *and* Q' *share the local variable* y *created by* P*. Then, any information produced by* P' *on* y *can be seen by* Q' *and viceversa.*

The reader may also refer [172] where CCP is endowed with an asynchronous message-based communication mechanism to model distributed systems. Agents can then communicate messages by using *send* and *receive* primitives. The work in [96] defines a model of *process mobility* for CCP. In this context, localities (or sites) are defined and agents are allowed to have their own local store. Sites are organized in a hierarchical way and then, it is possible for an agent to have sub-agents. A primitive *migrate* is added to the calculus in order to allow processes to move to another location and carry their local store. A distributed and probabilistic extension of CCP where a network of computational nodes, each of them with their own local store, is studied in [45]. Nodes can send and receive through communication channels constraints, agents (processes) and channels themselves. We also point the reader to [132] where the authors study an encoding of the utcc calculus into tccp. Such encoding has to deal with the **abs** operator in utcc and also with the fact that the notion of time differs from the source and target calculi.

3.6 Epistemic and Spatial Modalities in CCP

In some situations, the centralized notion of store makes CCP unsuitable for systems where information and processes can be shared or spatially distributed among certain groups of agents. In particular, agents posting and querying information in the presence of spatial hierarchies for sharing information and knowledge. For instance, friend circles and shared albums in social networks or shared folders in cloud storage provide natural examples of managing information access. These domains raise important problems such as the design of models to predict and prevent privacy breaches, which are commonplace nowadays.

In [124] the authors enhance and generalize the theory of CCP for systems with spatial distribution of information. More precisely, the underlying theory of constraint

systems is extended by adding space functions to their structure. Take for instance the constraint $d = \mathfrak{s}_i(c) \wedge \mathfrak{s}_j(c')$. Intuitively, d asserts that the local store of the agent i (resp. j) is c (resp. c'). Functions $\mathfrak{s}_i$, $\mathfrak{s}_j$,... can be seen as topological and closure operators and they allow for specifying spatial and epistemic information.

Definition 17 (Spatial-Epistemic CCP [124]) *Processes in Spatial-Epistemic CCP are obtained by adding the operator $[P]_i$ in the Syntax of Definition 3.*

The spatial operator can specify a process, or a local store of information, that resides within the space of the agent i (e.g., an application in some user account, or some private data shared with a specific group). This operator can also be interpreted as an epistemic construction to specify that the information computed by a process will be known to a given agent. It is worth noticing that the CCP concept of local variables cannot faithfully model local spaces since in the spatial constraint systems defined in [124], it is possible to have inconsistent local stores without propagating their inconsistencies towards the global store.

4 Programming Languages Based on the CCP Model

Several CCP programming languages have been designed. These cover a wide spectrum going from syntactic sugar over a particular CCP calculus, to graphical representations of the calculus primitives and to full fledged general purpose multiparadigm languages. Early CCP languages took inspiration in constraint logic programming (CLP), where unification was replaced by constraint solving. An example is the language $cc(FD)$ [111] that implements an efficient *finite domains* constraint system. The CLP model was implemented in various languages, each with a suitable constraint system (finite domains, booleans, real numbers). Nevertheless, these languages lacked flexibility since problem solving had to be tailored to the specific fix set of predefined constraints. In contrast, $cc(FD)$ offered general-purpose combinators, applicable to any constraint system, such as constructive disjunction and blocking implication [111]. These, together with entailment, allow the language to be tailored to specific user domains without losing the "naturalness" of specifications.

Programs in $cc(FD)$ are written in a Prolog-like syntax and user constraints are translated into canonical forms called *indexicals* [56] that can be implemented very efficiently.

Even though solving constraint problems remains an important goal of CCP languages, they have mostly evolved into powerful ways to define complex synchronization schemes in concurrent and distributed settings. In the rest of this section we describe some of the systems and programming languages built on the ideas of the CCP model. We also point to some implementation of frameworks and interpreters for the calculi described in Section 3. The reader may also refer to [88], a survey of the developments of (concurrent) languages that integrate constraint reasoning and solving.

4.1 Janus

Lucy [185] is a simple language were agents communicate by posting constraints over a mailbox called a "bag". Processes can merge and pass around bags in a kind of distributed version of the CCP model. Janus is an extension of Lucy intended for distributed programming and it resembles a concurrent logic programming language. A program is a network of agents that communicate by passing messages over a channel. Agents consist of *rules* of the form $p(t_1, \ldots, t_n) \leftarrow C \mid C_1, B$ where $t_1, \ldots, t_n$ are terms, C are ask and C_1 $tell$ constraints, and B is a conjunction of *goals*. When a message matches the pattern on the left, it triggers the behavior determined by the right hand side of the rule. In accordance with the CCP model, variables are logical and they have two aspects or *faces*, corresponding to *ask* and *tell* annotations. Any of these can also be passed around. Careful design of restrictions on *askers* and *tellers* ensure the *failure-free property*: Janus computations never abort due to the store becoming inconsistent.

A graphical representation of Janus, *Pictorial Janus* was proposed in [122]. The basic elements of a Pictorial Janus program are graphical primitives, i.e., closed contours, connections and links between objects. Rules like the one above can be specified inside agents. A visual debugging environment for Pictorial Janus, providing real-time animation of programs can be found in [72].

4.2 JCC

The language `jcc` [184] was designed as an integration of **Default** `tcc` into Java. `jcc` is intended for embedded reactive systems and for simulation and modeling in robotics and system biology. It implements bounded-time execution of the `tcc` calculus constructs. In `jcc` users can define their own constraint system and thus tune the language to particular domains. The main purpose of the language is to provide a model of loosely-coupled concurrent programming in Java. The model introduces the notion of a vat. A vat may be thought of as encapsulating a single synchronous, reactive `tcc` computation. A computation consists of a dynamically changing collection of interacting $vats$, communicating with each other through shared, mutable objects called $ports$. *Asker* and *teller* objects read from and write into the port. Constructs from the underlying `tcc` model allows an object to specify code that should be executed in the future.

4.3 LMNtal

The goal of LMNtal [205] is to provide a scalable and uniform view of concurrent programming concepts such as processes, messages, synchronous and asynchronous computation. It inherits ideas from the concurrent constraint language of Guarded Horn Clauses (GHC) [93] and from Janus. Basic components of the language are *links*, *multisets*, *nodes* and *transformations*. Links represent both communication channels between logically neighboring processes and logical neighborhood relations between data cells. Links are bi-directional.

Communication is based on constraints over logical variables. Processes sharing variables are thought of as been "connected", as in the CCP model. Multisets of nested nodes and links are a first-class notion in LMNtal. These organize into a hierarchy (called a *membrane* structure) and thus provide a kind of *ambient*, as in the Ambient calculus [55]. Transformations are rules, much like in Janus. LMNtal provides both channel mobility and process mobility: it allows dynamic reconfiguration of process structures as well as the migration of nested computations. An expression $p(x_1, ..., x_m)$ defines an atomic process. Variables x_i are its links. LMNtal makes no distinction between processes and data. Atom $x = y$ denotes a $connector$ between one side of the link x and one side of the link y. $\{P\}$ denotes a process enclosed within the membrane $\{\}$; and $T :- T$ a rewrite rule for processes. Links in the left part of the rule are $consumed$ and on the right hand are $produced$. Complex patterns can thus be defined for rule triggering and concepts such as mobility can be easily expressed. An implementation of LMNtal is available at `http://www.ueda.info.waseda.ac.jp/lmntal/`.

4.4 Constraint Handling Rules

Constraint Handling Rules (CHR) [89, 90] was originally conceived as a language for extending CLP systems with elegant mechanisms to define new constraint solvers, thus fulfilling the aim of CLP languages to be truly parametric in (user-built) constraint systems. CHR found afterwards widespread use as a general purpose multi-paradigm programming language. CHR provides users with declarative (multi-headed) rules for implementing simplification, propagation and so-called *simpagation* of constraints. Rules can be guarded with conditions that must hold for them to be applied. The rules act on a (multiset) constraint store. Simplification rules replace constraints in the store by simpler ones, propagation rules add redundant constraints (useful for additional simplifications) and simpagation rules do both. CHR is embedded in some host language $\mathcal{H}$, written CHR($\mathcal{H}$). The host language provides data types and some primitive constraints.

The declarative (classical logic) semantics of CHR [199] is given in terms of the constraint theory of the host language together with the logical formulas for each rule. A simplification rule $H \iff G \mid B$, for example, corresponds to the formula $\tilde{\forall}(G \rightarrow (H \leftrightarrow \exists \bar{x}.B))$, where $\tilde{\forall}$ quantifies over all free variables. This semantics, however, does not comply with the intended meaning of some CHR programs. The problem comes from the multiset store and from the unidirectionality and committed-choice nature of CHR rules. A better declarative semantics based on intuitionistic linear logic [97] was later proposed in [24].

CHR can encode a basic CCP language, as shown in [91]. The committed-choice feature of CHR is used to represent each $\mathbf{when}\ c_i\ \mathbf{do}\ P_i$ process in a non-deterministic choice $\sum_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i$ as a simplification rule of the form $summation(\sum(...)) \iff c_i \mid P_i$. The committed choice ensures that when this rule is chosen (c_i holds) none of the other rules for *ask* constructs in the summation can be used.

As mentioned, a key feature of CHR is the possibility of its embedding in many different types of programming languages (logic, functional, explicit-state), such as CHR(Prolog), CHR(Haskell), CHR(C). Various applications, ranging from Multi-Agent

systems to language processing (CHR grammars) or software testing have been developed in CHR. A summary of these can be found in [199].

The simplicity and expressivity of CHR have attracted the attention of several researchers. The meaning of CHR has evolved from *theoretical* semantics [90], which are highly non-deterministic, to more refined notions of transition systems that eliminate some of the non-determinism [70]. The latter is the basis for implementing systems based on CHR. Similar to the developments of probabilistic extensions of CCP, CHR has been also extended to replace non-deterministic choices (in the rule to be applied) with probabilistic choices [92]. Probabilistic CHR (PCHR) allows for an explicit control of the rule to be applied and fairness can be directly expressed by choosing an appropriate probability distribution on the rules. Other extensions dealing with distributed constraint stores [188] and user-definable rule priorities [125] have been also proposed. In [137] the connection between `lcc` and CHR is studied. The authors show that a semantic preserving encoding in both directions is possible. Moreover, properties as confluence [71], termination [168] as well as general abstract interpretation techniques [192] for the analysis and optimization of CHR programs have been established. The reader may refer [199] and [91] for a more complete account on these topics.

4.5 Oz

Arguably the CCP-based language that has found more widespread use is Oz [196, 197]. The Oz model takes inspiration from CCP and CLP and from its predecessor AKL [87]. It builds up from a kernel language consisting of a first-order structure defining the values and constraints Oz computes with, a CCP calculus (called the Oz calculus) over this structure and the *actor model*, a (non-formal) computation model introducing high level concurrent notions such as computation spaces (for speculative computation) and threads. In this model the usual store of constraints coexists with a so-called *predicate store* that includes a non-monotonic mutable store. The last one is used to model shared state and message passing concurrent computation via the notions of *cells* and *ports*.

Although the Oz kernel is small and conceptually simple, its rich semantics allows for the implementation of several computation models: declarative, stateful, lazy declarative, lazy stateful, eager, in both sequential and concurrent settings. The Oz kernel is based on a calculus comprising the ρ calculus [151] plus some additions for modeling functional, object-oriented, constraint-based and logic programming. The ρ calculus is a relational calculus parametrized by a logical constraint system. Its basic constructs are constraints, parallel composition, local declarations, conditionals and abstractions. Constraints are taken from the underlying constraint system, conditionals test for entailment of constraints from the store and abstractions are used to encode procedures.

As mentioned, traditional programming styles such as imperative, functional or object-oriented can homogeneously coexist within the Oz language. Furthermore, constraint programming, message passing concurrency and an asynchronous distribution model are also supported. A complete presentation and analysis of all Oz computation models, together with programming strategies for each, can be found in [173].

By default, Oz relies on a constraint system over (infinite) feature trees for value assignment to logical variables. Extensions to finite domains, finite sets and real intervals are also provided. Oz has been successfully used in many different problem domains. A strong point of the language is the coherent combination of the declarative pure CCP model with the traditional shared state scheme. A cell variable is assigned to a unique name in the monotonic (i.e. CCP) store and this name is associated with another variable in the mutable store. The latter represents the value of the cell. Changing the value of a cell amounts to changing the association of its name to a different variable. This variable, in turn, may have a different value from the previous one in the monotonic store. That is, cell values do not really change. What changes is the variable associated to its name.

4.6 CORDIAL

CORDIAL [174] is a visual language based on the π^+-calculus [66] and provides transparent integration of constraints and objects. Objects within methods are represented by closed contours. Object methods launch CCP processes that, in addition to the usual ask and $tell$ operations, can send messages to other objects. Messages are objects connected by links to object mailboxes. In CORDIAL objects are not located at some reference but "float" over a constraints medium and they are identified by an associated constraint parametrized on the local variable $self$. Senders willing to invoke some object method post a constraint involving some variable, say x, and then send the message to x. Any object such that its associated constraint can be entailed by the store conjoined with the constraint $self = x$, is eligible to accept the message. Some eligible object is then non-deterministically chosen to handle the message. This scheme allows for very complex patterns of communication and mobility.

4.7 Interpreters for CCP Calculi

Similar to `jcc`, some other interpreters for the calculi described in Section 3 have been implemented. In the context of `lcc`, SiLCC (`http://contraintes.rocq.inria.fr/~tmartine/silcc/`) is an implementation of the language with a module system as described in [106]. Furthermore, ALCOVE *Aeminium Linear COnstraints VErifier* (`http://avispa.puj.edu.co`) is a `lcc`-based tool for the analysis of access permissions in concurrent-by-default programs written in Aeminium [201].

The `ntcc` calculus has been implemented as `ntccSim` (`http://avispa.puj.edu.co`), an interpreter written in the Oz language. An important feature of this tool is that several constraint systems can be included in the same model. For instance, constraints over finite domains (FD) and real intervals (XRI) have been used to implement computational models of biological system (see Section 5.3). The `ntccrt` interpreter for the `ntcc` calculus (`http://sourceforge.net/projects/ntccrt/`) is written in C++ and specifications can be made in Common Lisp or in *OpenMusic* (`http://repmus.ircam.fr/openmusic/`), and then translated to C++. This framework can be also integrated as a plugin for either *Pure Data (*PD*) or Max/MSP* [169] to take advantage of the facilities offered by those languages to implement, for

example, sound processors. ntccrt makes use of the state of the art propagation techniques in Gecode (http://www.gecode.org/) to implement the underlying constraint system. Hence, the system is able to deal with real-time requirements for the execution of ntcc models in the context of computer-based musical improvisation (see Section 5.2). In [130] an interpreter for tccp written in Maude is described (http://users.dsic.upv.es/~villanue/tccp-func/). As for hcc, at http://www-cs-students.stanford.edu/~vgupta/hcc/ the reader may find an interpreter for this language. Finally, k-stores [16] is an interpreter for the epistemic and spatial calculus in Definition 17.

In the context of CHR, implementations of Prolog such as SWI-Prolog (http://www.swi-prolog.org) and SICStus Prolog (http://sicstus.sics.se) feature modules for CHR. WebCHR is a web tool that allows the execution of Prolog and CHR programs (http://chr.informatik.uni-ulm.de/~webchr/). Implementations of CHR for different programming languages such as Java, C and Haskell can be found at http://chr.informatik.uni-ulm.de/. The system CHRat [76] implements a modular version of CHR that allows for reusing built-in constraints, defined in a constraint system, as a constraint solver in another CHR program (http://contraintes.inria.fr/~tmartine/chrat/). CHRiSM [198] (http://people.cs.kuleuven.be/~jon.sneyers/chrism/) integrates CHR and PRISM (PRogramming In Statistical Modeling) [190], a probabilistic extension of Prolog for symbolic-statistical modeling.

5 Emergent Applications for CCP

Nowadays concurrent systems are ubiquitous in several domains and applications. They pervade different areas in science (e.g. biological and chemical systems), engineering (e.g., security protocols, mobile and service oriented computing and social networks) and even the arts (e.g. tools for multimedia interaction). CCP based languages and calculi have been extensively used to model, analyze and verify concurrent systems in different scenarios such as the aforementioned. The simplicity and the expressivity of this model attracts the attention of modelers mainly due to: (1) the parameterization of CCP in a constraint system provides a very flexible way to tailor data structures to specific domains and applications; (2) The declarative synchronization mechanism based on entailment of constraints eases the modeling of complex interactions between subsystems; (3) The ability to deal with partial information allows for modeling and studying such systems even when one is not fully aware of the behavior of all the components involved; (4) As we shall show in Section 6, CCP enjoys several reasoning techniques; finally, (5) the underlying model of CCP based on a common store of partial information is akin to several systems where components post information asynchronously.

This section is devoted to show some of the most relevant applications of CCP. We shall show that the reactive model of temporal CCP allows for the declarative specification of reactive systems such as electromechanical devices, software control and multimedia interaction systems. The temporal and probabilistic extensions of CCP have found application in system biology and physical systems. Finally, the declara-

tive nature of CCP and its reasoning techniques have been used to specify and verify, for instance, security protocols and service oriented computing systems.

5.1 Physical Systems

The work in [212] shows the applicability of `tcc` as programming language to specify controllers for electromechanical systems. In this setting, `tcc` provides a declarative model for the components that comprise the device. The authors show that the timing constructs in `tcc` can neatly express the pattern of interaction over time between the controller and the environment. Furthermore, since `tcc` programs can be compiled into finite-state machines [182], the implementation of the system is straightforward and efficient. The strong connection of CCP calculi and logic is also an advantage in this context since it is possible to use standard techniques for proving properties over the software constructed.

A natural application of the `hcc` calculus is the modeling of physical systems. In this scenario, one is interested in observing the change of the state of the system when interacting with the environment (discrete change) and also when evolving autonomously (continuous change). In [103], a compositional model of a photocopier paper path in `hcc` is presented. The declarative nature of `hcc` is particularly useful in this setting, since for each fragment of the model, it is only necessary to state the laws of physics applicable, e.g. equilibrium laws, boundary condition, etc.

The work in [207] makes use of CCP for the design of reprographic machines. In this case, the CCP model allows to capture in a declarative and compositional way the model of the machine in an appropriate level of abstraction, thus providing support for the requirement specification and design activities.

Finally, in [153, 167] the authors show how tasks for an RCX programmable microcontroller can be specified in the `ntcc` calculus.

5.2 Music Interaction and Composition

Many systems for music composition and interaction have been proposed in the past. These are based in general either on dataflow models and languages inspired in digital sound processing systems, for interaction, and on existing (mostly functional) general-purpose programming languages, for music composition. The purpose of the former is controlling musical devices (e.g. sound synthesizers) in real-time performance settings. The latter aim at providing composers with tools for supporting the structuring and controlled evolution of complex musical material. CCP-based calculi have been proposed recently in both domains. What is intended in the first case is to take advantage of the natural synchronization mechanism provided by blocking ask processes to model complex concurrent interactions in a precise and simple way. In the second case, the logical nature of the calculus is used to verify musical properties of a system before launching costly constraint processes.

In [175], `ntcc` models of various musical problems are described. These problems involve relations between harmonic and rhythmic properties. What (harmonic) information is output at each time unit determines rhythm properties. The problem consists in finding out whether two musical voices with specific melodic evolution rules can

comply with some given harmonic relations when played together. This problem pops up frequently in music composition in many different forms. The particular instance of this problem described in [175] is the following: two voices are constructed in such a way that the second one reproduces the first (up to transposition) with a time gap of p. The upper and lower voices play notes in the sets S_1 and S_2, respectively. A transposition function f gives for each upper voice note the lower note that is to be played p time units later. Additional constraints state that time units that are either contiguous or separated by p units should not play the same two notes (chords). Finally, all chords thus formed between the two voices must be chosen among the elements of a given set C. The strategy is to construct a weaker `ntcc` model of the problem and then, use the linear temporal logic associated with `ntcc` to find conditions for the problem to be solvable.

In [13] an entirely different domain is explored using `ntcc`, that of live improvisation of an interpreter and the computer. The computer must first learn the musical *style* of the human interpreter and then begin to play jointly in the same style. A *style* in this case means some set of meaningful sequences of musical material (notes, durations, etc.) the interpreter has played. A graph structure called *factor oracle* (FO) is used to efficiently represent this set. The `ntcc` models define processes that construct in real-time the FO (i.e. learn the style) and then synchronize with the interpreter to travel through different paths in the FO graph (i.e. improvise).

Interactive Multimedia deals with scenarios where multimedia content and interactive events are related via computer programs. A recent trend is to express the relationships between contents and interactions in a precise way by integrating both in a general model, thus providing a kind of enriched score for composers. One such system, called *interactive scores* [2], allows the specification of contents whose temporal occurrence is not given in advance but is the result of temporal constraints (in the form of Allen interval relations) defined for them. The occurrence of external interactions (whose window of observation is also subject to constraints) is thought to transform (or instantiate) the temporal structure of the piece by imposing further constraints. The score thus defines a collection of possible temporal occurrences of the audio/visual events in a performance. In [204], `ntcc` has been proposed to extend this model so that external interactions can condition paths in the score. The calculus is also used to implement a precise synchronization of processes in this model. In [156] the ability of `utcc` to express mobile behavior was exploited to define interactive scores where interactive points can be defined to adapt the hierarchical structure of the score depending on the information inferred from the environment.

The `ntcc` system in [176] is proposed as a framework for constructing sound processing models in a precise and compact way. Processes in a given time unit define (data flow) transformations of a sound sample supplied by the environment (or a past process). The resulting sample is then output and can also be transmitted to the next time unit. Shared variables are used to represent links between processes. Sample delay units are straightforwardly represented with the `ntcc` *next* primitive. The compositionality of the calculus is used to represent hierarchies of sound processing boxes.

5.3 Biological Systems

The study of biological systems has found a fertile substrate in the CCP model, mainly due to: (1) constraints can naturally express quantitative information as well as partial information on the available reactants in the system; (2) synchronization via constraint entailment allows for triggering actions when some information can be derived from the system. For instance, it is natural to express that a given reaction occurs only when certain component is present in the system; (3) the ability of CCP to build up models (i.e., components) by parallel composition leads to a robust modeling strategy: one can study separately components of a system and then observe the behavior of the whole system; (4) timed operators in temporal extensions of CCP allows for describing actions (more precisely reaction in this context) that can take several time-units to be completed; and (5), probabilistic constructs as in `pcc` allow for choosing, according to a given probability distribution function, among different reaction that may have different propensities to occur.

In [10, 104] the authors propose a model in `ntcc` of a mechanism for cellular transport: the Sodium-Potassium pump. In the same work, the connection of `ntcc` and linear temporal logic is exploited to facilitate reachability analysis.

BioWayS (`http://avispa.javerianacali.edu.co`) [57] is a web tool for the modeling of biochemical networks based on the `ntcc` calculus. In [112], it is shown that BioWayS allows for the compositional modeling and simulation of biological systems.

The stochastic extension `sCCP` proposed in [44] allows for describing stochastic duration by means of functional rates. This calculus has been used to describe biological networks and it has been shown to be a general and extensible framework to describe a wide class of dynamical behaviors and kinetic laws.

The discrete and continuous nature of `hcc` has been exploited to model dynamic biological systems, e.g. in [74, 32]. For instance, in [32] it is shown that `hcc` can naturally model a variety of biological phenomena, such as reaching thresholds, kinetics, genetic interaction and biological pathways.

In [46], the authors carry out a comparative study of `sCCP`, `hcc`, and Biocham [75] as languages for the modeling of biochemical reactions. In [26] the authors compare the `sCCP`, `ntcc`, and `hcc` models for the blood coagulation process. Experimental results are shown when using the interpreters of `hcc` and `ntcc` to simulate the system.

Finally, [178] makes use of a linear CCP language to model protein interaction. The work in [160] uses CCP techniques for the protein structure prediction problem, which consists in predicting the 3D native conformation of a protein, when its sequence of amino acids is known. The authors also provide a prototype in the Oz language showing the feasibility of the approach proposed.

5.4 Security and Service Oriented Computing

Due to technological advances such as the Internet and mobile computing, security has become a serious challenge in Computer Science. Several process calculi have been proposed in order to deal with the verification of security protocols. Some of the features of those calculi are reminiscent of CCP. For instance, SPL [61], the spi calculus

variants in [1] and the calculi presented in [8] and [42] are all operationally defined in terms of configurations containing items of information (messages) which can only increase during evolution. Such monotonic evolution of information is akin to the notion of monotonic store in CCP. Moreover, the calculi in [8, 42, 86, 18] are parametric in an entailment relation over a logic for reasoning about protocol properties very much like CCP is parametric in an entailment relation over an underlying constraint system.

The notion of mobility in utcc is used in [158] to model and exhibit the secrecy flaw of the well known Needham-Schroeder [150] security protocol. The cryptographic primitives and the messages an attacker may infer are specified in a suitable constraint system. In [157] the authors describe an encoding of a simple language for security into monotonic utcc processes (i.e. processes not including the **unless** constructor). Then, by using the denotational semantics of utcc, the authors show that it is possible to give a closure operator semantics to languages for security. Moreover, [82] develops an abstract interpretation framework to approximate the semantics of a security protocol for verification purposes.

A type system for restricting the behavior of agents in utcc is studied in [113]. This system gives guarantees that a channel name and encrypted values are only extracted by agents that are able to infer the channel or the non-encrypted value from the store.

An extension of tccp [34] is studied in [129] as a language for modeling security protocols. The authors show how the language can naturally express the behavior of the principals in the protocol and a Dolev-Yao attacker [67] for verification purposes.

In [118], a policy language for role-based access control in distributed systems along the lines of **Default** tcc is proposed. The authors combine constraint reasoning and temporal logic model-checking to verify whether a given resource (e.g. a directory in a file system) can be accessed.

Soft Concurrent Constraint Programming based languages have been used in the modeling and verification of service oriented computing systems. The cc-pi calculus, for instance, has been used to specify Quality of Service (QoS) and to conclude Service Level Agreement (SLA) contracts [50]. The language is equipped with mechanisms for resource allocation and for joining different SLA requirements to reach an agreement between agents (clients and servers) in a service oriented computing scenario. In [53], the non-deterministic choice in cc-pi is replaced by a prioritized guarded choice. Alternatives are labeled with constraints and the chosen one corresponds to the constraint with a higher priority over the constraints of the alternative branches. The prioritized calculus is then used to express richer QoS negotiations where agents may state preferences between a set of possible alternatives. See also [52] where an overview of cc-pi and prioritized cc-pi is given. That work describes also the application of these languages in the specification of service negotiation in telecommunication and financial domains in the context of the SENSORIA project (http://www.sensoria-ist.eu).

The work in [31] shows the application of a non-monotonic extension of scc (where constraints can be *retracted* from the store) in the modeling of negotiation processes where different parties have to agree on a contract specifying QoS requirements expressed as semiring values. Moreover, in [27], the authors specify with the same language an access control mechanism with granularity at the level of constraints. Then,

it is possible to control the way the different agents of the system post and consume information.

Sessions and sessions types [203, 64] were introduced with the aim of guaranteeing structured communication between agents. Type rules statically ensure that a predefined communication scheme/protocol (typically based on duality) is respected along process execution. The work in [133] studies an encoding of the language for structured communications proposed in [115] into `utcc`. The framework allows for the declarative analysis of sessions in network protocols. Furthermore, due to the timed nature of `utcc`, it is possible to reason about session duration and expiration in structured communications. In [59], the authors combine CCP and name passing in the style of `cc-pi` together with sessions. The resulting calculus aims at specifying QoS requirements where safe interactions between clients and servers are assured. The primitive used to open a session makes use of constraints whose satisfaction is necessary for starting and conducting the session interaction. Hence, constraints and the underlying type system guarantee bilinearity, i.e., channels are private and they are exclusively used to carry on the communication prescribed by the session. The calculus is also endowed with a primitive for delegation (also restricted by constraints) that allows agents in a protocol to delegate a service to a third party.

In [49] a different approach for guaranteeing structured communication is devised. In this work, the client-service interaction is decomposed in three phases: *negotiation* where agents negotiate certain desired behaviors, but without any guarantee of success; once the agents agree, they *commit* and the choosing behaviors must be respected; finally, the protocol is *executed* upon the agreed properties and deadlock-freeness is guaranteed. The communication scheme is specified in a source calculus close to the π-calculus establishing communication patterns between clients and services. This model is then compiled into a target calculus close to the prioritized `cc-pi` calculus [53] where named constraint semirings [50] encode the behaviors of agents. More precisely, the choices in the compiled model are guarded by check (ask) constructs that enable the corresponding continuation only if the global store allows it. The novelty of this approach relies on the fact that constraints are used to choose the right interaction and to avoid deadlock in the execution phase. That is, the combination of the constraints of the client and the server leads only to executions of the client-service system that do not stuck.

Propositional Contract Logic (PCL) [17] extends intuitionistic logic with a *contractual* form of implication. This logic aims to model SLA contracts to formalize the duties of the client and the server in a service oriented scenario. The execution model of this logic is based on a calculus of contracting processes which relies on a CCP language (plus primitives for a name passing discipline) where agents tell and asks formulas in PCL. The calculus provides also a *fuse* operation that, unlike `cc-pi`, allows for simultaneous multiparty agreements.

5.5 Other Applications and Results

In [206], a Büchi finite state automata characterization of the strongest postcondition of the local independent fragment of the `ntcc` calculus is given. Using this characterization, the author proves the decidability of the satisfaction problem for the restricted

negation formulas without rigid variables in linear temporal logic [135].

The work in [209] studies the execution of formal specifications in SPECS-C++, a model-based formal specification language designed for specifying the interfaces of C++ classes. Since this specification language was not designed to be executed, the approach proposed by the authors is to translate such a formal specifications into the CCP based language AKL [109]. A subset of the specifications in SPECS-C++ can be then executed in AKL. If the specification is consistent (and executable), it is possible to find the set of post-states satisfying the specification.

The connection between CCP languages parametrized with a finite domain (FD) constraint system and query languages in finite model theory such as first-order logic over relational vocabulary, fixpoint logics, and Datalog is studied in [78]. This work presents complexity results for CCP(FD) when considering complete and open constraint systems (those that do not fix the interpretation of all relation symbols); flat and deep guards, i.e. guards that can be an arbitrary process instead of a constraint; and terminal and success observables (where there are no blocking asks in the final configuration).

The work in [191] introduces a constraint system to handle equations and inequations over real numbers. This constraint system along with the model of `lcc` provides a general and extensible foundation for linear programming algorithms design. The authors show that it is possible to build a version of the (constraint solver) simplex algorithm in this framework and additionally, that it is possible to specify non-trivial concurrent algorithms on it.

In [141] the authors experiment with the use of `ntcc` as a language to describe dynamic enumeration strategies to solve constraint satisfaction problems. In this case, the reactivity of the calculus allows to design enumeration strategies that adapt themselves according to information issued from the resolution process and from external solvers such as an incomplete solver (e.g. local search).

A `lcc` model of the access permission mechanism in Aeminium [201], an object-oriented concurrent-by-default programming language, is presented in [155]. The logical interpretation of `lcc` as formulas in intuitionistic linear logic allows for the automatic verification of Aeminium's program properties such as deadlock detection, correctness with respect to the access permission specification, and the ability of methods to run concurrently.

6 Verification and Reasoning Techniques

In this section we survey different reasoning techniques developed for CCP. As we stated before, CCP is a model of concurrency tied to logic. Then, CCP benefits from verification techniques coming from both process calculi and logic. In the forthcoming section, we give an account of semantics, program analyses, logic characterizations, model checking and equivalences developed for CCP calculi.

$$
\begin{array}{lll}
\mathrm{D}_{\mathrm{SKIP}} & [\![\mathbf{skip}]\!]_I & = \mathcal{C} \\
\mathrm{D}_{\mathrm{TELL}} & [\![\mathbf{tell}(c)]\!]_I & = \{d \mid d \vdash c\} \\
\mathrm{D}_{\mathrm{ASK}} & [\![\mathbf{when}\ c\ \mathbf{do}\ P]\!]_I & = \{d \mid d \nvdash c\} \cup [\![P]\!]_I \\
\mathrm{D}_{\mathrm{PAR}} & [\![P \parallel Q]\!]_I & = [\![P]\!]_I \cap [\![Q]\!]_I \\
\mathrm{D}_{\mathrm{LOCAL}} & [\![(\mathbf{local}\,\overline{x}; c)\ P]\!]_I & = \{d \mid \text{there is } d' \text{ s.t. } d' \vdash c, \exists\overline{x}(d) = \exists\overline{x}(d') \text{ and } d' \in [\![P]\!]_I\} \\
\mathrm{D}_{\mathrm{CALL}} & [\![q(\overline{x})]\!]_I & = I(q(\overline{x}))
\end{array}
$$

Figure 2: Semantic equations for the CCP calculus in Definition 3. $\mathcal{C}$ denotes the set of constraints in the underlying constraint system.

6.1 Semantic Frameworks and Program Analysis

The first semantic characterizations of CCP were inspired by methods and techniques from concurrency theory such as failure sets and bisimulation. For instance, [37] and [38] made use of tree-like structures labeled with functions on substitutions. Simpler tree-like structures labeled by constraints are used in [94], and in [186], similar structures modulo equivalence relations based on bisimulation are considered. The work in [39] showed that it is possible to give a simpler compositional semantics for CCP since communication is asynchronous and actions are triggered only depending on the current store. The proposed semantics consists of sequences of constraints labeled by assume/tell modes.

In [187], deterministic CCP processes are identified with Scott's closure operators (idempotent, extensive and monotonic functions). Such functions can be retrieved from their set of fixpoints and then, the meaning of a process is given by the set of constraints upon which the process cannot add any information. This set is also known as the *strongest postcondition* and it corresponds to the *quiescent* constraints for the process. This semantics characterization is quite elegant and simple. Figure 2 shows the semantic equations for the CCP calculus in Definition 3. Let us elaborate on them. Notice that all constraints are quiescent for $\mathbf{skip}$ ($\mathrm{D}_{\mathrm{SKIP}}$), i.e., $\mathbf{skip}$ cannot add any information to any constraint. A process $\mathbf{tell}(c)$ cannot add any information to d if d entails c ($\mathrm{D}_{\mathrm{TELL}}$). A constraint d is quiescent for $\mathbf{when}\ c\ \mathbf{do}\ P$ either if d does not entail c or d is quiescent for P ($\mathrm{D}_{\mathrm{ASK}}$). The constraint d is quiescent for $P \parallel Q$ if neither P nor Q can add any information to d ($\mathrm{D}_{\mathrm{PAR}}$). If d' cannot add any information to P and d and d' differ only on the information about the variables in $\overline{x}$ (i.e., $\exists\overline{x}(d) = \exists\overline{x}(d')$), then d cannot add any information to $(\mathbf{local}\,\overline{x}; c)\ P$ ($\mathrm{D}_{\mathrm{LOCAL}}$). Process calls are interpreted according to the *interpretation* I that gives meaning to the process definition ($\mathrm{D}_{\mathrm{CALL}}$). The semantics is then obtained by a standard fixpoint construction.

For the case of non-deterministic CCP, [187] denotes processes with bounded traced operators that recall the path the processes followed to reach the fixpoint, i.e., the sequence of tell/ask interactions with the environment. This semantics was proven to be equivalent [40] to that in [39]. The result follows from the fact that both semantics are fully abstract with respect to the notion of observables for finite computations. Furthermore, [80] studied restricted fragments of (non-deterministic) CCP that can be characterized as closure operators on sets of constraints. More precisely, the authors show that this is possible for *structurally confluent* CCP processes, i.e., processes whose out-

puts do not depend on the scheduling policy of the system. This fragment includes, for instance, *angelic*-CCP where only *local choices* are allowed (i.e, all the guards are the same) and *mutually exclusive choice* (i.e., one guard excludes the others). The semantic characterization for CCP with local choice found application in [36, 81] to establish the semantic foundations and a verification system for CLP with delay by means of closure operators. Confluence in CCP has been also studied in [148] where a confluent operational semantics for CCP with blind and angelic choice is proposed. In the same lines, [136] proposes a confluent calculus for CCP that considers blind and guarded choices. This calculus is later used for the analysis of CCP programs. To effectively deal with guarded choices, the semantics in [136] keeps the precluded alternatives when selecting a branch of execution. Those alternatives are guarded in such a way that they reduce to "failure" on termination.

For the case of infinite computations and non-deterministic behavior, the semantic foundations in [187] were extended in [119] and [126] to give meaning to *angelic*-CCP processes. Later on, [41] showed that the domain used in [119, 126] is not closed under set intersection and then, the semantics of parallel composition is not well defined. In [154] the authors considered the Lehmann's powerdomain [128] over set of traces. Then, a (compositional) fixpoint semantics can be obtained in order to retrieve the outputs of infinite computations where fairness is assumed, i.e., all enabled agents are eventually executed. Relying on these ideas, [41] shows that the construction in [154] can be used to capture both infinite computations and non-determinism when considering sets of constraints (instead of traces). Fairness requirements have been also studied in [54], where the operational semantics of the parallel operator makes use of quantitative metrics to provide a more accurate way to establish which of the processes in a parallel composition can succeed. This thus guarantees a fair criterion on the selection of processes.

The elegant semantic characterization of CCP has been extended to its subcalculi. In [182], `tcc` processes are denoted as closure operators on sequences of constraints. Then, for instance, the sequence $c_1.c_2.c_3....$ is quiescent for $\mathbf{next}\ P$ only if the subsequent $c_2.c_3...$ is quiescent for P. Similarly, [153] gives a denotational semantics to the `ntcc` calculus. This idea is also present in [157] where processes are identified with closure operators on sequences of linear temporal logic formulas [135] for the case of the `utcc` calculus. As for `tccp`, in [34] it is shown that the full abstraction problem for this language cannot be reduced to that one of CCP. Then, a semantics based on reactive sequences is proposed to be correct with respect to the notion of observables and fully abstract. Following these ideas, [28] proposes a semantics for `stcc`.

The aforementioned semantic characterizations of CCP rely on the idea of the strongest postcondition, i.e., the quiescent inputs of a process. A finer-grain characterization consists in determining the *minimal* requirements from the environment to produce an output, this is, observe the *causality* relation between inputs and outputs. These ideas have been developed to endow CCP with true concurrency semantics, i.e., semantics that interpret the parallel operator as a concurrent execution (instead of an interleaving execution) of processes. In [143, 144, 145], the operational rules of CCP are augmented with the context required for the reaction to occur. From such rules, the authors show that a contextual net [146] (a Petri Net that considers contexts representing the conditions required for an event to occur) can be constructed to capture

all possible computations of a given program. It allows also to capture causal dependencies, mutual exclusion and the concurrency among processes. The contextual net semantics of CCP has been extended in [147, 48] with an inconsistency dependency relations to deal with atomic tells (see Section 3.5). This semantics is exploited to derive safe parallelization of CLP computation steps. The ideas in [143, 145] found also application in [102] where a semantics for CCP based on *contexted tokens* of the form c^d is proposed. Intuitively, c^d means d is the cause for the effect c. In this work it is also shown that the contexted tokens a process may output can be retrieved compositionally.

In the context of program analysis and transformations, unfold/fold transformations in CCP have been studied in [73]. The proposed transformation system, besides folding and unfolding, includes other new operations, namely backward instantiation, ask and tell simplification, branch elimination, conservative ask elimination and distribution. This framework has found application for proving deadlock freeness of CCP programs. Furthermore, [21] investigated transformation techniques for CCP based on the *replacement* (see [163] for a survey of transformation techniques in logic languages). Abstract semantic characterizations for CCP have been studied in [79, 213]. Those works proved that it is not possible to give a sound approximation, in the sense of abstract interpretation [60], for the ask operator if one considers only abstract domain values. The main problem is that *weaker* constraints are needed to over-approximate the program outputs but ask-synchronization requires *stronger* constraints to guarantee that suspension in the abstract model implies suspension in the concrete model. Then, an entailment relation between abstract and concrete constraints is used to give a safe approximation of the semantics. These ideas have been extended in [82] to consider temporal extensions of CCP. The proposed abstract semantics have been used to prove properties such as groundness and suspension freeness and they have served as the foundation for abstract diagnosis and debugging techniques for `tccp` [58] and `ntcc` programs [83].

6.2 Logics, Specifications and Verification

CCP-based languages have been shown to have a strong connection to logic that distinguishes this model from other formalisms for concurrency. The work in [138] shows that CCP processes can be viewed as logic formulas, constructs in the language as logical connectives and simulations (runs) as proofs. In [126], the semantic characterization of CCP processes is used to show that the logical view of the program and its denotation correspond to each other. Then, proving that P satisfies a given property F amounts to show that the semantics of P is included in the semantic objects satisfying F. Here the author considered infinite computations and then, liveness properties (i.e., something *good* eventually happens) may be proved. In [33], a calculus for proving correctness of CCP programs is introduced. In this framework, the specification of the program is given in terms of a first-order formula. The authors pointed out that some problems arise when representing non-deterministic choices by disjunction and when considering the representation of this logical connective in the constraint system. For example, assume that process P satisfies certain property F and consider the agent $Q = \mathbf{when}\ x = 0\ \mathbf{do}\ P + \mathbf{when}\ x > 0\ \mathbf{do}\ P$. One would like to state that the above process satisfies the formula F' defined as $(x = 0 \vee x > 0) \Rightarrow F$. Logically speaking,

$x \geq 0$ implies $x = 0 \vee x > 0$. Nevertheless, if we run Q in parallel with $\mathbf{tell}(x \geq 0)$, none of the guards of Q is enabled and then, P is not executed. Therefore, the authors enrich the logic of the constraint system and a property (represented by a constraint) is thus interpreted as the set of constraints that entails it. Consequently, logical operators are interpreted in terms of the corresponding set-theoretic operations. For instance, the formula F' above is interpreted as the union of the set of constraints that entails $x = 0$ and F and the set of constraints that entails $x > 0$ and F.

As for `lcc`, [77] shows that the observable behavior of CCP and `lcc` processes can be characterized as proofs in intuitionistic linear logic. This characterization is shown to be useful to verify *liveness* properties of systems. Furthermore, the language is endowed with a phase semantics to verify *safety* properties.

In the context of timed CCP calculi, [182] proposes a proof system for `tcc` based on an intuitionistic logic enriched with a next operator. Judgments in the proof system have the form $P_1, ..., P_n \vdash P$ where $P_1, ..., P_n$ and P are processes. Such judgments are valid if and only if the intersection of the denotations of the agents $P_1, ..., P_n$ is contained in the denotation of P; equivalently, any observation that can be made from the parallel system of agents $P_1, ..., P_n$ can also be made from P. The results in [33] are extended and strengthened in [153], where a linear temporal logic characterization of `ntcc` is studied along with a proof system to verify properties. In the same lines, [158] gives a logic characterization for the `utcc` calculus. A temporal logic based on the epistemic modalities of *knowledge* and *belief* is proposed in [35]. The assertions in the logic capture what a process *assumes* from the input (the environment) and what a process *commits* to, i.e., the outputs in a given time-unit. Then, this logic provides a language for the specification of the reactive behavior of processes. A sound a complete proof system to reason about the correctness of `tccp` programs based on this logic is reported in [35].

Model checking techniques have been also explored in the context of temporal CCP languages. In [3, 85], the behavior of `tccp` processes is modeled by means of a so called `tccp` structure. Such structure is like a Kripke structure where, following the CCP philosophy, the state of the system is represented as a conjunction of constraints. This allows for the specification of the possible states and transitions the system may exhibit. Then, the modal logic in [35] is used to specify the property to be verified and the model checking graph is analyzed to decide if the process satisfies or not the property. A tool implementing the construction of the `tccp` structure is described in [131]. The work of [85] was based on the ideas developed for the automatic verification of `tcc` programs in [84].

In order to mitigate the state explosion problem, [5] considers an abstract model-checking technique where both the model and the property to be verified are abstracted. Following a data-abstraction approach, abstract operations are defined to over approximate the behavior of the original program. The authors identify that an over-approximation is not sufficient to give an accurate approximation of the synchronization and timed mechanisms of `tccp`. Then, under-approximations of the semantics are considered to improve accuracy. The authors prove the total correctness of the abstract semantics which models precisely the suspension of processes. The abstract semantics is also implemented as a source-to-source transformation that compiles the abstract program back into `tccp`. Abstract refinements are also proposed to improve

accuracy in the verification process. The work in [5] is extended in [7] where the authors study a general framework for abstract verification and analysis of concurrent programs. The programs considered in this study are imperative (states as variable valuations) and declarative (states as conjunction of constraints as in `tccp`) style. The semantics is approximated and implemented as a source-to-source transformation that interprets the abstract actions into the original language. Finally, we mention the work of [6] that proposes a new semantics for `tccp` able to recognize the time instant when some piece of information is added to the store. The logic in [35] is also extended with discrete-time marks to model synchronous real-time properties. Thus, real time is introduced in `tccp` and it is shown how model checkers for this language can be extended to deal with real-time properties.

6.3 Equivalences

Bisimilarity is one of the main representative of process equivalences. It captures our intuitive notion of process equivalence; two processes are equivalent if they can match each other moves. Furthermore, it provides an elegant co-inductive proof technique based on bisimulation [177]. Despite the relevance of bisimilarity, there have been few attempts to define a notion of bisimilarity for CCP. The work in [11] provides a labeled transition semantics and a notion of bisimilarity for CCP. A labeled transition $\langle P, c\rangle \xrightarrow{d} \langle Q, e\rangle$ says that d is the minimal piece of information that needs to be joint with c to perform a reduction from P. From these transitions, a derived notion of bisimulation following standard lines is obtained: Two configurations $\langle P, c\rangle$ and $\langle P', c'\rangle$ are bisimilar iff whenever $\langle P, c\rangle \xrightarrow{d} \langle Q, e\rangle$ then must exist a transition $\langle P', c' \wedge d\rangle \rightarrow \langle Q', e'\rangle$ so that $\langle Q, e\rangle$ and $\langle Q', e'\rangle$ are also bisimilar. The authors also showed a strong correspondence with existing CCP notions by providing a fully-abstract characterization of a standard observable behavior. Furthermore, in [12] the authors provided an algorithm for the automatic verification of bisimilarity. In the same lines, [105] studies a labeled bisimulation for Linear CCP processes. The latter equivalence is shown to coincide with a may-testing equivalence and the barbed congruence. Finally, a notion of open bisimulation is proposed in [51] for `cc-pi`. Essentially, two processes are open bisimilar if they have the same stores of constraints - which can be statically checked - and if their moves can be mutually simulated.

6.4 CCP and other Computational Models

The Fusion calculus [208] is a π-calculus variant that, rather than substitution, uses an implicit notion of equality, a fusion, between variables/names. So, in the Fusion calculus instead of replacing a parameter x of an input with that of an output, say z, an implicit fusion is given between the parameters involved in the communication by imposing $x = z$. This idea results in a calculus that is simpler and yet as expressive as the π-calculus. In fact, the Fusion calculus has only one binding operator where the π-calculus has two (input and restriction) and it has a complete symmetry between input and output actions. The authors in [208] gave an encoding from CCP into Fusion calculus as a compelling demonstration of the expressivity of their calculus. This makes

the reasoning techniques and tools of the Fusion calculus available for CCP. However, the encoding is only intended for equality (inequality) based constraint systems such as the Herbrand constraint system. This may not come as a surprise since a fusion can be thought of as an equality constraint between two variables. In fact, an encoding from CCP with equality constraints into Explicit Fusion [211], an alternative presentation of the Fusion calculus, should be almost immediate.

CCP offers reasoning techniques substantially different from those from π-based calculi. CCP also focuses on the notion of partial information while it abstracts away from channel and point-to-point communication. It is worth noticing that some variants of the π-calculus include logic assertions in their process language (see e.g., the ψ-calculus [18] and `cc-pi` [50]) as well the use of parametric signatures (see e.g., the applied π-calculus [86]). These recent additions to the machinery of the π-calculus variants bear witness to the importance of the concepts singled out in CCP. We note also that CCP has been shown to be expressive enough to encode other models of computation, different from its predecessors Concurrent Logic Programming and Constraint Logic Programming. For instance, different (fragments of) asynchronous concurrent formalisms such as the asynchronous π-calculus, Actor models, Linda, Petri nets have been encoded as (linear and higher order) CCP processes [121, 181, 179, 127, 105]. Furthermore, sequential models of computations such as the the untyped λ-calculus and Minsky machines have been encoded into CCP [179, 157].

7 Concluding Remarks

The simplicity and elegance of the CCP model have attracted the attention of both practitioners and theoreticians in Computer Science. It can be seen in the large number of extensions proposed in the literature to cope with different notions such as time, non-determinism, mobility, etc. Being parametric in an underlying constraint system, the CCP model has offered the flexibility needed to be adopted as a formal basis for several programming languages and practical applications.

Another appealing feature of CCP is the set of reasoning techniques the model offers. For instance, closure operator semantics, logical characterization, proof and type systems, model checking, and more recently, equivalences and bisimulation techniques.

A current line of research is the development of a more principled notion of time in CCP. This is central to applications that require to impose real-time constraints on the execution of processes as in multimedia interaction systems. In the same lines, building interpreters that guarantee reliable responses in time is required.

From the verification point of view, there are ongoing works in defining more robust proof systems for CCP calculi and static analyzers for CCP programs. Developing machine-assisted tools relying on those techniques for the automatic verification of system properties is also desirable.

Finally, epistemic and spatial constraint systems open a new window for the specification of emergent systems such as cloud computing and social networks in ways that provide more flexible views of information hiding/sharing and where properties such as privacy could be specified.

Acknowledgments. We thank the anonymous reviewers for their detailed comments that helped us to improve this paper. Special thanks to Jorge A. Perez for his careful remarks and suggestions. This work has been partially supported by grant 1251-521-28471 from Colciencias (Colombia), and by Digiteo and DGAR (École Polytechnique) funds for visitors.

References

[1] Abadi, M., Gordon, A.D.: A calculus for cryptographic protocols: The spi calculus. Inf. Comput. **148**(1), 1–70 (1999)

[2] Allombert, A., Desainte-Catherine, M., Assayag, G.: Iscore: a system for writing interaction. In: S. Tsekeridou, A.D. Cheok, K. Giannakis, J. Karigiannis (eds.) DIMEA, *ACM International Conference Proceeding Series*, vol. 349, pp. 360–367. ACM (2008)

[3] Alpuente, M., Falaschi, M., Villanueva, A.: A symbolic model checker for tccp programs. In: N. Guelfi (ed.) RISE, *LNCS*, vol. 3475, pp. 45–56. Springer (2004)

[4] Alpuente, M., Gramlich, B., Villanueva, A.: A framework for timed concurrent constraint programming with external functions. ENTCS **188**, 143–155 (2007)

[5] Alpuente, M., del Mar Gallardo, M., Pimentel, E., Villanueva, A.: A semantic framework for the abstract model checking of tccp programs. Theor. Comput. Sci. **346**(1), 58–95 (2005)

[6] Alpuente, M., del Mar Gallardo, M., Pimentel, E., Villanueva, A.: Verifying real-time properties of tccp programs. J. UCS **12**(11), 1551–1573 (2006)

[7] Alpuente, M., del Mar Gallardo, M., Pimentel, E., Villanueva, A.: An abstract analysis framework for synchronous concurrent languages based on source-to-source transformation. ENTCS **206**, 3–21 (2008)

[8] Amadio, R.M., Lugiez, D., Vanackère, V.: On the symbolic reduction of processes with cryptographic functions. Theor. Comput. Sci. **290**(1), 695–740 (2003)

[9] Aranda, J., Pérez, J.A., Rueda, C., Valencia, F.D.: Stochastic behavior and explicit discrete time in concurrent constraint programming. In: de la Banda and Pontelli [14], pp. 682–686

[10] Arbelaez, A., Gutierrez, J., Perez, J.A.: Timed concurrent constraint programming in systems biology. Newsletter of the ALP **19**(4) (2006)

[11] Aristizábal, A., Bonchi, F., Palamidessi, C., Pino, L.F., Valencia, F.D.: Deriving labels and bisimilarity for concurrent constraint programming. In: M. Hofmann (ed.) FOSSACS, *LNCS*, vol. 6604, pp. 138–152. Springer (2011)

[12] Aristizábal, A., Bonchi, F., Valencia, F.D., Pino, L.F.: Partition refinement for bisimilarity in ccp. In: Ossowski and Lecca [159], pp. 88–93

[13] Assayag, G., Dubnov, S., Rueda, C.: A concurrent constraints factor oracle model for music improvisation. In: Proc. of CLEI 2006 (2006)

[14] de la Banda, M.G., Pontelli, E. (eds.): Logic Programming, 24th Int. Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings, *LNCS*, vol. 5366. Springer (2008)

[15] Barahona, P., Felty, A.P. (eds.): Proceedings of the 7th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 11-13 2005, Lisbon, Portugal. ACM (2005)

[16] Barco, A., Knight, S., Valencia, F.: K-stores A spatial and epistemic concurrent constraint interpreter. In: Proc. of WFLP'12 (2012)

[17] Bartoletti, M., Zunino, R.: A calculus of contracting processes. In: LICS, pp. 332–341. IEEE Computer Society (2010)

[18] Bengtson, J., Johansson, M., Parrow, J., Victor, B.: Psi-calculi: a framework for mobile processes with nominal data and logic. Logical Methods in Computer Science **7**(1) (2011)

[19] Bergstra, J.A., Klop, J.W.: Algebra of communicating processes with abstraction. Theor. Comput. Sci. **37**, 77–121 (1985)

[20] Berry, G., Gonthier, G.: The esterel synchronous programming language: Design, semantics, implementation. Sci. Comput. Program. **19**(2), 87–152 (1992)

[21] Bertolino, M., Etalle, S., Palamidessi, C.: The replacement operation for ccp programs. In: A. Bossi (ed.) LOPSTR, *LNCS*, vol. 1817, pp. 216–233. Springer (1999)

[22] Best, E., de Boer, F., Palamidessi, C.: Concurrent constraint programming with information removal. In: First Workshop on Concurrent Constraint Programming (1995)

[23] Best, E., de Boer, F.S., Palamidessi, C.: Partial order and sos semantics for linear constraint programs. In: D. Garlan, D.L. Métayer (eds.) COORDINATION, *LNCS*, vol. 1282, pp. 256–273. Springer (1997)

[24] Betz, H., Frühwirth, T.W.: A linear-logic semantics for constraint handling rules. In: P. van Beek (ed.) CP, *LNCS*, vol. 3709, pp. 137–151. Springer (2005)

[25] Bistarelli, S.: Semirings for Soft Constraint Solving and Programming, *LNCS*, vol. 2962. Springer (2004)

[26] Bistarelli, S., Bottalico, M., Santini, F.: Constraint-based languages to model the blood coagulation cascade. In: S. Ferilli, D. Malerba (eds.) Logic-Based Approaches in Bioinformatics, pp. 32–41 (2009)

[27] Bistarelli, S., Campli, P., Santini, F.: A secure coordination of agents with non-monotonic soft concurrent constraint programming. In: Ossowski and Lecca [159], pp. 1551–1553

[28] Bistarelli, S., Gabbrielli, M., Meo, M.C., Santini, F.: Timed soft concurrent constraint programs. In: D. Lea, G. Zavattaro (eds.) COORDINATION, *LNCS*, vol. 5052, pp. 50–66. Springer (2008)

[29] Bistarelli, S., Montanari, U., Rossi, F.: Soft concurrent constraint programming. ACM Trans. Comput. Log. **7**(3), 563–589 (2006)

[30] Bistarelli, S., Montanari, U., Rossi, F., Schiex, T., Verfaillie, G., Fargier, H.: Semiring-based CSPs and valued CSPs: Frameworks, properties, and comparison. Constraints **4**(3), 199–240 (1999)

[31] Bistarelli, S., Santini, F.: A nonmonotonic soft concurrent constraint language to model the negotiation process. Fundam. Inform. **111**(3), 257–279 (2011)

[32] Bockmayr, A., Courtois, A.: Using hybrid concurrent constraint programming to model dynamic biological systems. In: P.J. Stuckey (ed.) ICLP, *LNCS*, vol. 2401, pp. 85–99. Springer (2002)

[33] de Boer, F.S., Gabbrielli, M., Marchiori, E., Palamidessi, C.: Proving concurrent constraint programs correct. ACM Trans. Program. Lang. Syst. **19**(5), 685–725 (1997)

[34] de Boer, F.S., Gabbrielli, M., Meo, M.C.: A timed concurrent constraint language. Inf. Comput. **161**(1), 45–83 (2000)

[35] de Boer, F.S., Gabbrielli, M., Meo, M.C.: Proving correctness of timed concurrent constraint programs. ACM Trans. Comput. Log. **5**(4), 706–731 (2004)

[36] de Boer, F.S., Gabbrielli, M., Palamidessi, C.: Proving correctness of constraint logic programs with dynamic scheduling. In: R. Cousot, D.A. Schmidt (eds.) SAS, *LNCS*, vol. 1145, pp. 83–97. Springer (1996)

[37] de Boer, F.S., Kok, J.N., Palamidessi, C., Rutten, J.J.M.M.: Control flow versus logic: A denotational and a declarative model for guarded horn clauses. In: A. Kreczmar, G. Mirkowska (eds.) MFCS, *LNCS*, vol. 379, pp. 165–176. Springer (1989)

[38] de Boer, F.S., Kok, J.N., Palamidessi, C., Rutten, J.J.M.M.: Semantic models for a version of parlog. In: ICLP, pp. 621–636 (1989)

[39] de Boer, F.S., Palamidessi, C.: A fully abstract model for concurrent constraint programming. In: S. Abramsky, T.S.E. Maibaum (eds.) TAPSOFT, Vol.1, *LNCS*, vol. 493, pp. 296–319. Springer (1991)

[40] de Boer, F.S., Palamidessi, C.: On the semantics of concurrent constraint programming. In: ALPUK, pp. 145–173 (1992)

[41] de Boer, F.S., Pierro, A.D., Palamidessi, C.: Nondeterminism and infinite computations in constraint programming. Theor. Comput. Sci. **151**(1), 37–78 (1995)

[42] Boreale, M.: Symbolic trace analysis of cryptographic protocols. In: F. Orejas, P.G. Spirakis, J. van Leeuwen (eds.) ICALP, *LNCS*, vol. 2076, pp. 667–681. Springer (2001)

[43] Borning, A. (ed.): Principles and Practice of Constraint Programming, Second International Workshop, PPCP'94, Rosario, Orcas Island, Washington, USA, May 2-4, 1994, Proceedings, *LNCS*, vol. 874. Springer (1994)

[44] Bortolussi, L., Policriti, A.: Modeling biological systems in stochastic concurrent constraint programming. Constraints **13**(1-2), 66–90 (2008)

[45] Bortolussi, L., Wiklicky, H.: A distributed and probabilistic concurrent constraint programming language. In: M. Gabbrielli, G. Gupta (eds.) ICLP, *Lecture Notes in Computer Science*, vol. 3668, pp. 143–158. Springer (2005)

[46] Bottalico, M., Bistarelli, S.: Constraint based languages for biological reactions. In: Hill and Warren [114], pp. 561–562

[47] Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A theory of communicating sequential processes. J. ACM **31**(3), 560–599 (1984)

[48] Bueno, F., Hermenegildo, M.V., Montanari, U., Rossi, F.: Partial order and contextual net semantics for atomic and locally atomic cc programs. Sci. Comput. Program. **30**(1-2), 51–82 (1998)

[49] Buscemi, M.G., Coppo, M., Dezani-Ciancaglini, M., Montanari, U.: Constraints for service contracts. In: R. Bruni, V. Sassone (eds.) TGC, *LNCS*, vol. 7173, pp. 104–120. Springer (2011)

[50] Buscemi, M.G., Montanari, U.: CC-Pi: A constraint-based language for specifying service level agreements. In: R. De Nicola (ed.) ESOP, *LNCS*, vol. 4421, pp. 18–32. Springer (2007)

[51] Buscemi, M.G., Montanari, U.: Open bisimulation for the concurrent constraint pi-calculus. In: S. Drossopoulou (ed.) ESOP, *LNCS*, vol. 4960, pp. 254–268. Springer (2008)

[52] Buscemi, M.G., Montanari, U.: CC-Pi: A constraint language for service negotiation and composition. In: M. Wirsing, M.M. Hölzl (eds.) Results of the SENSORIA Project, *LNCS*, vol. 6582, pp. 262–281. Springer (2011)

[53] Buscemi, M.G., Montanari, U.: Qos negotiation in service composition. J. Log. Algebr. Program. **80**(1), 13–24 (2011)

[54] Campli, P., Bistarelli, S.: Capturing fair computations on concurrent constraint language. In: Hill and Warren [114], pp. 559–560

[55] Cardelli, L., Gordon, A.D.: Mobile ambients. In: M. Nivat (ed.) FoSSaCS, *LNCS*, vol. 1378, pp. 140–155. Springer (1998)

[56] Carlson, B., Haridi, S., Janson, S.: AKL(FD) - a concurrent language for FD programming. In: SLP, pp. 521–535 (1994)

[57] Chiarugi, D., Falaschi, M., Olarte, C., Palamidessi, C.: Compositional modelling of signalling pathways in timed concurrent constraint programming. In: A. Zhang, M. Borodovsky, G. Özsoyoglu, A.R. Mikler (eds.) BCB, pp. 414–417. ACM (2010)

[58] Comini, M., Titolo, L., Villanueva, A.: Abstract diagnosis for timed concurrent constraint programs. TPLP **11**(4-5), 487–502 (2011)

[59] Coppo, M., Dezani-Ciancaglini, M.: Structured communications with concurrent constraints. In: C. Kaklamanis, F. Nielson (eds.) TGC, *LNCS*, vol. 5474, pp. 104–125. Springer (2008)

[60] Cousot, P., Cousot, R.: Abstract interpretation and application to logic programs. J. Log. Program. **13**(2&3), 103–179 (1992)

[61] Crazzolara, F., Winskel, G.: Petri nets in cryptographic protocols. In: IPDPS, p. 149. IEEE Computer Society (2001)

[62] Dahl, V., Niemelä, I. (eds.): Logic Programming, 23rd International Conference, ICLP 2007, Porto, Portugal, September 8-13, 2007, Proceedings, *LNCS*, vol. 4670. Springer (2007)

[63] Demoen, B., Lifschitz, V. (eds.): Logic Programming, 20th International Conference, ICLP 2004, Saint-Malo, France, September 6-10, 2004, Proceedings, *LNCS*, vol. 3132. Springer (2004)

[64] Dezani-Ciancaglini, M., de'Liguoro, U.: Sessions and session types: An overview. In: C. Laneve, J. Su (eds.) WS-FM, *LNCS*, vol. 6194, pp. 1–28. Springer (2009)

[65] Díaz, J.F., Gutierrez, G., Olarte, C.A., Rueda, C.: Using constraint programming for reconfiguration of electrical power distribution networks. In: P.V. Roy (ed.) MOZ, *LNCS*, vol. 3389, pp. 263–276. Springer (2004)

[66] Díaz, J.F., Rueda, C., Valencia, F.D.: Pi+- calculus: A calculus for concurrent processes with constraints. CLEI Electron. J. **1**(2) (1998)

[67] Dolev, D., Yao, A.C.C.: On the security of public key protocols. IEEE Transactions on Information Theory **29**(2), 198–207 (1983)

[68] Dovier, A., Pontelli, E. (eds.): A 25-Year Perspective on Logic Programming: Achievements of the Italian Association for Logic Programming, GULP, *LNCS*, vol. 6125. Springer (2010)

[69] Dubois, D., Fargier, H., Prade, H.: The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In: Proc. 2nd IEEE Conference on Fuzzy Systems, pp. 1131–1136 vol.2. San Francisco, CA (1993)

[70] Duck, G.J., Stuckey, P.J., de la Banda, M.J.G., Holzbaur, C.: The refined operational semantics of constraint handling rules. In: Demoen and Lifschitz [63], pp. 90–104

[71] Duck, G.J., Stuckey, P.J., Sulzmann, M.: Observable confluence for constraint handling rules. In: Dahl and Niemelä [62], pp. 224–239

[72] Dücker, M., Lehrenfeld, G., Müller, W., Tahedl, C.: A generic system for interactive real-time animation. In: ECBS, pp. 263–270. IEEE Computer Society (1997)

[73] Etalle, S., Gabbrielli, M., Meo, M.C.: Transformations of CCP programs. ACM Trans. Program. Lang. Syst. **23**(3), 304–395 (2001)

[74] Eveillard, D., Ropers, D., de Jong, H., Branlant, C., Bockmayr, A.: A multiscale constraint programming model of alternative splicing regulation. Theor. Comput. Sci. **325**(1), 3–24 (2004)

[75] Fages, F., Batt, G., Maria, E.D., Jovanovska, D., Rizk, A., Soliman, S.: Computational systems biology in biocham. ERCIM News **2010**(82), 36 (2010)

[76] Fages, F., de Oliveira Rodrigues, C.M., Martinez, T.: Modular CHR with ask and tell. In: Proc. of Fifth Workshop on Constraint Handling Rules (2008)

[77] Fages, F., Ruet, P., Soliman, S.: Linear concurrent constraint programming: Operational and phase semantics. Inf. Comput. **165**(1), 14–41 (2001)

[78] Fages, F., Soliman, S., Vianu, V.: Expressiveness and complexity of concurrent constraint programming: a finite model theoretic approach. Tech. Rep. 98-14, LIENS (1998)

[79] Falaschi, M., Gabbrielli, M., Marriott, K., Palamidessi, C.: Compositional analysis for concurrent constraint programming. In: LICS, pp. 210–221. IEEE Computer Society (1993)

[80] Falaschi, M., Gabbrielli, M., Marriott, K., Palamidessi, C.: Confluence in concurrent constraint programming. Theor. Comput. Sci. **183**(2), 281–315 (1997)

[81] Falaschi, M., Gabbrielli, M., Marriott, K., Palamidessi, C.: Constraint logic programming with dynamic scheduling: A semantics based on closure operators. Inf. Comput. **137**(1), 41–67 (1997)

[82] Falaschi, M., Olarte, C., Palamidessi, C.: A framework for abstract interpretation of timed concurrent constraint programs. In: A. Porto, F.J. López-Fraguas (eds.) PPDP, pp. 207–218. ACM (2009)

[83] Falaschi, M., Olarte, C., Palamidessi, C., Valencia, F.: Declarative diagnosis of temporal concurrent constraint programs. In: Dahl and Niemelä [62], pp. 271–285

[84] Falaschi, M., Policriti, A., Villanueva, A.: Modeling concurrent systems specified in a temporal concurrent constraint language-i. ENTCS **48**, 197–210 (2001)

[85] Falaschi, M., Villanueva, A.: Automatic verification of timed concurrent constraint programs. TPLP **6**(3), 265–300 (2006)

[86] Fournet, C., Abadi, M.: Hiding names: Private authentication in the applied pi calculus. In: M. Okada, B.C. Pierce, A. Scedrov, H. Tokuda, A. Yonezawa (eds.) ISSS, *LNCS*, vol. 2609, pp. 317–338. Springer (2002)

[87] Franzén, T., Haridi, S., Janson, S.: An overview of the andorra kernel language. In: L.H. Eriksson, L. Hallnäs, P. Schroeder-Heister (eds.) ELP, *LNCS*, vol. 596, pp. 163–179. Springer (1991)

[88] Frühwirth, T., Michel, L., Schulte, C.: Chapter 13 - constraints in procedural and concurrent languages. In: F. Rossi, P. van Beek, T. Walsh (eds.) Handbook of Constraint Programming, *Foundations of Artificial Intelligence*, vol. 2, pp. 453 – 494. Elsevier (2006)

[89] Frühwirth, T.W.: Constraint handling rules. In: Constraint Programming, pp. 90–107 (1994)

[90] Frühwirth, T.W.: Theory and practice of constraint handling rules. J. Log. Program. **37**(1-3), 95–138 (1998)

[91] Frühwirth, T.W.: Constraint Handling Rules. Cambridge University Press (2009)

[92] Frühwirth, T.W., Pierro, A.D., Wiklicky, H.: Probabilistic constraint handling rules. ENTCS **76**, 115–130 (2002)

[93] Furukawa, K., Ueda, K.: Ghc - a language for a new age of parallel programming. In: K.V. Nori, S. Kumar (eds.) FSTTCS, *Lecture Notes in Computer Science*, vol. 338, pp. 364–376. Springer (1988)

[94] Gabbrielli, M., Levi, G.: Unfolding and fixpoint semantics of concurrent constraint logic programs. In: H. Kirchner, W. Wechler (eds.) ALP, *LNCS*, vol. 463, pp. 204–216. Springer (1990)

[95] Gabbrielli, M., Palamidessi, C., Valencia, F.D.: Concurrent and reactive constraint programming. In: Dovier and Pontelli [68], pp. 231–253

[96] Gilbert, D., Palamidessi, C.: Concurrent constraint programming with process mobility. In: J.W. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.K. Lau, C. Palamidessi, L.M. Pereira, Y. Sagiv, P.J. Stuckey (eds.) Computational Logic, *LNCS*, vol. 1861, pp. 463–477. Springer (2000)

[97] Girard, J.Y.: Linear logic. Theor. Comput. Sci. **50**, 1–102 (1987)

[98] Gupta, V., Jagadeesan, R., Panangaden, P.: Stochastic processes as concurrent constraint programs. In: A.W. Appel, A. Aiken (eds.) POPL, pp. 189–202. ACM (1999)

[99] Gupta, V., Jagadeesan, R., Saraswat, V.A.: Models for concurrent constraint programming. In: U. Montanari, V. Sassone (eds.) CONCUR, *LNCS*, vol. 1119, pp. 66–83. Springer (1996)

[100] Gupta, V., Jagadeesan, R., Saraswat, V.A.: Probabilistic concurrent constraint programming. In: A.W. Mazurkiewicz, J. Winkowski (eds.) CONCUR, *LNCS*, vol. 1243, pp. 243–257. Springer (1997)

[101] Gupta, V., Jagadeesan, R., Saraswat, V.A.: Computing with continuous change. Sci. Comput. Program. **30**(1-2), 3–49 (1998)

[102] Gupta, V., Jagadeesan, R., Saraswat, V.A.: Truly concurrent constraint programming. Theor. Comput. Sci. **278**(1-2), 223–255 (2002)

[103] Gupta, V., Saraswat, V., Struss, P.: A model of a photocopier paper path. In: Proceedings of the 2nd IJCAI Workshop on Engineering Problems for Qualitative Reasoning (1995)

[104] Gutierrez, J., Pérez, J.A., Rueda, C., Valencia, F.D.: Timed concurrent constraint programming for analysing biological systems. ENTCS **171**(2), 117–137 (2007)

[105] Haemmerlé, R.: Observational equivalences for linear logic concurrent constraint languages. TPLP **11**(4-5), 469–485 (2011)

[106] Haemmerlé, R., Fages, F., Soliman, S.: Closures and modules within linear logic concurrent constraint programming. In: V. Arvind, S. Prasad (eds.) FSTTCS, *LNCS*, vol. 4855, pp. 544–556. Springer (2007)

[107] Halbwachs, N.: Synchronous programming of reactive systems. In: A.J. Hu, M.Y. Vardi (eds.) CAV, *LNCS*, vol. 1427, pp. 1–16. Springer (1998)

[108] Hankin, C. (ed.): Programming Languages and Systems - ESOP'98, *LNCS*, vol. 1381. Springer (1998)

[109] Haridi, S., Janson, S., Montelius, J., Franzén, T., Brand, P., Boortz, K., Danielsson, B., Carlson, B., Keisu, T., Sahlin, D., Sjöland, T.: Concurrent constraint programming at sics with the andorra kernel language (extended abstract). In: PPCP, pp. 107–116 (1993)

[110] Henkin, L., J.D., M., Tarski, A.: Cylindric Algebras, Part I. North-Holland (1971)

[111] Hentenryck, P.V., Saraswat, V.A., Deville, Y.: Design, implementation, and evaluation of the constraint language cc(FD). J. Log. Program. **37**(1-3), 139–164 (1998)

[112] Hermith, D., Olarte, C., Rueda, C., Valencia, F.D.: Modeling cellular signaling systems: An abstraction-refinement approach. In: M.P. Rocha, J.M.C. Rodríguez, F. Fdez-Riverola, A. Valencia (eds.) PACBB, *Advances in Intelligent and Soft Computing*, vol. 93, pp. 321–328. Springer (2011)

[113] Hildebrandt, T., López, H.A.: Types for secure pattern matching with local knowledge in universal concurrent constraint programming. In: Hill and Warren [114], pp. 417–431

[114] Hill, P.M., Warren, D.S. (eds.): Logic Programming, 25th International Conference, ICLP 2009, Pasadena, CA, USA, July 14-17, 2009. Proceedings, *LNCS*, vol. 5649. Springer (2009)

[115] Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: Hankin [108], pp. 122–138

[116] Jaffar, J., Lassez, J.L.: Constraint logic programming. In: POPL, pp. 111–119. ACM Press (1987)

[117] Jaffar, J., Maher, M.J.: Constraint logic programming: A survey. J. Log. Program. **19/20**, 503–581 (1994)

[118] Jagadeesan, R., Marrero, W., Pitcher, C., Saraswat, V.A.: Timed constraint programming: a declarative approach to usage control. In: Barahona and Felty [15], pp. 164–175

[119] Jagadeesan, R., Saraswat, V., Shanbhogue, V.: Angelic non-determinism in concurrent constraint programming. Tech. rep., Xerox Parc (1991)

[120] Jouannaud, J.P. (ed.): Constraints in Computational Logics, First International Conference, CCL'94, Munich, Germant, September 7-9, 1994, *LNCS*, vol. 845. Springer (1994)

[121] Kahn, K.M., Saraswat, V.A.: Actors as a special case of concurrent constraint programming. In: A. Yonezawa (ed.) OOPSLA/ECOOP, pp. 57–66. ACM (1990)

[122] Kahn, K.M., Saraswat, V.A.: Complete visualization of concurrent programs and their executions. In: LPE, pp. 30–34 (1990)

[123] de Kergommeaux, J.C., Codognet, P.: Parallel logic programming systems. ACM Comput. Surv. **26**(3), 295–336 (1994)

[124] Knight, S., Palamidessi, C., Panangaden, P., Valencia, F.D.: Spatial and epistemic modalities in constraint-based process calculi. In: M. Koutny, I. Ulidowski (eds.) CONCUR, *LNCS*, vol. 7454, pp. 317–332. Springer (2012)

[125] Koninck, L.D., Schrijvers, T., Demoen, B.: User-definable rule priorities for chr. In: M. Leuschel, A. Podelski (eds.) PPDP, pp. 25–36. ACM (2007)

[126] Kwiatkowska, M.Z.: Infinite behaviour and fairness in concurrent constraint programming. In: J.W. de Bakker, W.P. de Roever, G. Rozenberg (eds.) REX Workshop, *LNCS*, vol. 666, pp. 348–383. Springer (1992)

[127] Laneve, C., Montanari, U.: Mobility in the cc-paradigm. In: I.M. Havel, V. Koubek (eds.) MFCS, *LNCS*, vol. 629, pp. 336–345. Springer (1992)

[128] Lehmann, D.J.: Categories for fixpoint-semantics. In: FOCS, pp. 122–126. IEEE Computer Society (1976)

[129] Lescaylle, A., Villanueva, A.: Using tccp for the Specification and Verification of Communication Protocols. In: Proc. of WFLP 07 (2007)

[130] Lescaylle, A., Villanueva, A.: The tccp interpreter. ENTCS **258**(1), 63–77 (2009)

[131] Lescaylle, A., Villanueva, A.: A tool for generating a symbolic representation of tccp executions. ENTCS **246**, 131–145 (2009)

[132] Lescaylle, A., Villanueva, A.: Bridging the gap between two concurrent constraint languages. In: J. Mariño (ed.) WFLP, *LNCS*, vol. 6559, pp. 155–173. Springer (2010)

[133] López, H.A., Olarte, C., Pérez, J.A.: Towards a unified framework for declarative structured communications. In: A.R. Beresford, S.J. Gay (eds.) PLACES, *EPTCS*, vol. 17, pp. 1–15 (2009)

[134] Maher, M.J.: Logic semantics for a class of committed-choice programs. In: ICLP, pp. 858–876 (1987)

[135] Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer-Verlag (1991)

[136] Marriott, K., Odersky, M.: A confluent calculus for concurrent constraint programming. Theor. Comput. Sci. **173**(1), 209–233 (1997)

[137] Martinez, T.: Semantics-preserving translations between linear concurrent constraint programming and constraint handling rules. In: T. Kutsia, W. Schreiner, M. Fernández (eds.) PPDP, pp. 57–66. ACM (2010)

[138] Mendler, N.P., Panangaden, P., Scott, P.J., Seely, R.A.G.: A logical view of concurrent constraint programming. Nord. J. Comput. **2**(2), 181–220 (1995)

[139] Milner, R.: A finite delay operator in synchronous CCS. Tech. Rep. CSR-116-82, University of Edinburgh (1992)

[140] Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, Parts I and II. Inf. Comput. **100**(1), 1–40 (1992)

[141] Monfroy, E., Olarte, C., Rueda, C.: Process calculi for adaptive enumeration strategies in constraint programming. Research in Computer Science (2007)

[142] Montanari, U.: Networks of constraints: Fundamental properties and applications to picture processing. Inf. Sci. **7**, 95–132 (1974)

[143] Montanari, U., Rossi, F.: True concurrency in concurrent constraint programming. In: ISLP, pp. 694–713 (1991)

[144] Montanari, U., Rossi, F.: Contextual occurence nets and concurrent constraint programming. In: H.J. Schneider, H. Ehrig (eds.) Dagstuhl Seminar on Graph Transformations in Computer Science, *LNCS*, vol. 776, pp. 280–295. Springer (1993)

[145] Montanari, U., Rossi, F.: A concurrenct semantics for concurrent constraint programming via contextual nets. In: V. Saraswat, P.V. Hentenryck (eds.) Principles and Practice of Constraint Programming, pp. 3–27. MIT Press (1995)

[146] Montanari, U., Rossi, F.: Contextual nets. Acta Inf. **32**(6), 545–596 (1995)

[147] Montanari, U., Rossi, F., Bueno, F., de la Banda, M.J.G., Hermenegildo, M.V.: Towards a concurrent semantics based analysis of cc and clp. In: Borning [43], pp. 151–161

[148] Montanari, U., Rossi, F., Saraswat, V.A.: Cc programs with both in- and nondeterminism: A concurrent semantics. In: Borning [43], pp. 162–172

[149] Müller, T., Müller, M.: Finite set intervals in oz. In: WLP, pp. 17–19 (1997)

[150] Needham, R.M., Schroeder, M.D.: Using encryption for authentication in large networks of computers. Commun. ACM **21**(12), 993–999 (1978)

[151] Niehren, J., Smolka, G.: A confluent relational calculus for higher-order programming with constraints. In: Jouannaud [120], pp. 89–104

[152] Nielsen, M., Palamidessi, C., Valencia, F.D.: On the expressive power of temporal concurrent constraint programming languages. In: PPDP, pp. 156–167. ACM (2002)

[153] Nielsen, M., Palamidessi, C., Valencia, F.D.: Temporal concurrent constraint programming: Denotation, logic and applications. Nord. J. Comput. **9**(1), 145–188 (2002)

[154] Nyström, S.O., Jonsson, B.: Indeterminate concurrent constraint programming: A fixpoint semantics for non-terminating computations. In: ILPS, pp. 335–352 (1993)

[155] Olarte, C., Pimentel, E., Rueda, C., Cataño, N.: A linear concurrent constraint approach for the automatic verification of access permissions. In: D.D. Schreye, G. Janssens, A. King (eds.) PPDP, pp. 207–216. ACM (2012)

[156] Olarte, C., Rueda, C.: A declarative language for dynamic multimedia interaction systems. In: E. Chew, A. Childs, C.H. Chuan (eds.) Mathematics and Computation in Music, *Communications in Computer and Information Science*, vol. 38, pp. 218–227. Springer Berlin Heidelberg (2009)

[157] Olarte, C., Valencia, F.D.: The expressivity of universal timed ccp: undecidability of monadic fltl and closure operators for security. In: S. Antoy, E. Albert (eds.) PPDP, pp. 8–19. ACM (2008)

[158] Olarte, C., Valencia, F.D.: Universal concurrent constraint programing: symbolic semantics and applications to security. In: R.L. Wainwright, H. Haddad (eds.) SAC, pp. 145–150. ACM (2008)

[159] Ossowski, S., Lecca, P. (eds.): Proceedings of the ACM Symposium on Applied Computing, SAC 2012, Riva, Trento, Italy, March 26-30, 2012. ACM (2012)

[160] Palù, A.D., Dovier, A., Fogolari, F.: Protein folding simulation in ccp. In: Demoen and Lifschitz [63], pp. 452–453

[161] Pérez, J.A., Rueda, C.: Non-determinism and probabilities in timed concurrent constraint programming. In: de la Banda and Pontelli [14], pp. 677–681

[162] Petri, C.A.: Fundamentals of a theory of asynchronous information flow. In: IFIP Congress, pp. 386–390 (1962)

[163] Pettorossi, A., Proietti, M.: Transformation of logic programs: Foundations and techniques. J. Log. Program. **19/20**, 261–320 (1994)

[164] Pierro, A.D., Wiklicky, H.: A banach space based semantics for probabilistic concurrent constraint programming. In: X. Lin (ed.) CATS, *Australian Computer Science Communications*, vol. 20, pp. 245–260. Springer-Verlag Singapore Pte. Ltd. (1998)

[165] Pierro, A.D., Wiklicky, H.: Probabilistic concurrent constraint programming: Towards a fully abstract model. In: L. Brim, J. Gruska, J. Zlatuska (eds.) MFCS, *LNCS*, vol. 1450, pp. 446–455. Springer (1998)

[166] Pierro, A.D., Wiklicky, H.: Concurrent constraint programming: towards probabilistic abstract interpretation. In: PPDP, pp. 127–138. ACM (2000)

[167] del Pilar Muñoz, M., Hurtado, A.R.: Programming robotic devices with a timed concurrent constraint language. In: M. Wallace (ed.) CP, *LNCS*, vol. 3258, p. 803. Springer (2004)

[168] Pilozzi, P., Schreye, D.D.: Improved termination analysis of chr using self-sustainability analysis. In: G. Vidal (ed.) LOPSTR, *LNCS*, vol. 7225, pp. 189–204. Springer (2011)

[169] Puckette, M., Apel, T., Zicarelli, D.: Real-time audio analysis tools for Pd and MSP. In: Proceedings, International Computer Music Conference., pp. 109–112 (1998)

[170] Reisig, W.: Petri Nets: An Introduction, *Monographs in Theoretical Computer Science. An EATCS Series*, vol. 4. Springer (1985)

[171] Reiter, R.: A logic for default reasoning. Artif. Intell. **13**(1-2), 81–132 (1980)

[172] Réty, J.H.: Distributed concurrent constraint programming. Fundam. Inform. **34**(3), 323–346 (1998)

[173] Roy, P.V., Haridi, S.: Concepts, Techniques, and Models of Computer Programming. MIT Press (2004)

[174] Rueda, C., Alvarez, G., Quesada, L., Tamura, G., Valencia, F.D., Díaz, J.F., Assayag, G.: Integrating constraints and concurrent objects in musical applications: A calculus and its visual language. Constraints **6**(1), 21–52 (2001)

[175] Rueda, C., Valencia, F.: On validity in modelization of musical problems by CCP. Soft Comput. **8**(9) (2004)

[176] Rueda, C., Valencia, F.D.: A temporal concurrent constraint calculus as an audio processing framework. In: Sound and Music Computing conference (2005)

[177] Sangiorgi, D.: Introduction to Bisimulation and Coinduction. Cambridge Universtity Press (2012)

[178] Saraswat, V.: Euler: an applied lcc language for graph rewriting. Tech. rep., IBM TJ Watson Research Center (2004)

[179] Saraswat, V., Lincoln, P.: Higher-order Linear Concurrent Constraint Programming. Tech. rep., Xerox Parc (1992)

[180] Saraswat, V.A.: The category of constraint systems is cartesian-closed. In: LICS, pp. 341–345. IEEE Computer Society (1992)

[181] Saraswat, V.A.: Concurrent Constraint Programming. MIT Press (1993)

[182] Saraswat, V.A., Jagadeesan, R., Gupta, V.: Foundations of timed concurrent constraint programming. In: LICS, pp. 71–80. IEEE Computer Society (1994)

[183] Saraswat, V.A., Jagadeesan, R., Gupta, V.: Timed default concurrent constraint programming. J. Symb. Comput. **22**(5/6), 475–520 (1996)

[184] Saraswat, V.A., Jagadeesan, R., Gupta, V.: jcc: Integrating timed default concurrent constraint programming into java. In: F. Moura-Pires, S. Abreu (eds.) EPIA, *LNCS*, vol. 2902, pp. 156–170. Springer (2003)

[185] Saraswat, V.A., Kahn, K.M., Levy, J.: Janus: A step towards distributed constraint programming. In: NACLP, pp. 431–446 (1990)

[186] Saraswat, V.A., Rinard, M.C.: Concurrent constraint programming. In: F.E. Allen (ed.) POPL, pp. 232–245. ACM Press (1990)

[187] Saraswat, V.A., Rinard, M.C., Panangaden, P.: Semantic foundations of concurrent constraint programming. In: D.S. Wise (ed.) POPL, pp. 333–352. ACM Press (1991)

[188] Sarna-Starosta, B., Ramakrishnan, C.R.: Compiling constraint handling rules for efficient tabled evaluation. In: M. Hanus (ed.) PADL, *LNCS*, vol. 4354, pp. 170–184. Springer (2007)

[189] Sarria, G.: Real-time concurrent constraint calculus: The complete operational semantics. Engineering Letters **19**(1), 38–45 (2011)

[190] Sato, T.: A glimpse of symbolic-statistical modeling by prism. J. Intell. Inf. Syst. **31**(2), 161–176 (2008)

[191] Schächter, V.: Linear concurrent constraint programming over reals. In: M.J. Maher, J.F. Puget (eds.) CP, *LNCS*, vol. 1520, pp. 400–416. Springer (1998)

[192] Schrijvers, T., Stuckey, P.J., Duck, G.J.: Abstract interpretation for constraint handling rules. In: Barahona and Felty [15], pp. 218–229

[193] Scott, D.S.: Domains for denotational semantics. In: M. Nielsen, E.M. Schmidt (eds.) ICALP, *LNCS*, vol. 140, pp. 577–613. Springer (1982)

[194] Shapiro, E.: The family of concurrent logic programming languages. ACM Comput. Surv. **21**(3) (1989)

[195] Smolka, G.: A foundation for higher-order concurrent constraint programming. In: Jouannaud [120], pp. 50–72

[196] Smolka, G.: The Oz programming model. In: J. van Leeuwen (ed.) Computer Science Today, *LNCS*, vol. 1000, pp. 324–343. Springer (1995)

[197] Smolka, G.: Concurrent constraint programming based on functional programming (extended abstract). In: Hankin [108], pp. 1–11

[198] Sneyers, J., Meert, W., Vennekens, J., Kameya, Y., Sato, T.: Chr(prism)-based probabilistic logic learning. TPLP **10**(4-6), 433–447 (2010)

[199] Sneyers, J., Weert, P.V., Schrijvers, T., Koninck, L.D.: As time goes by: Constraint handling rules. TPLP **10**(1), 1–47 (2010)

[200] Stallman, R.M., Sussman, G.J.: Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. Artif. Intell. **9**(2), 135–196 (1977)

[201] Stork, S., Marques, P., Aldrich, J.: Concurrency by default: using permissions to express dataflow in stateful programs. In: S. Arora, G.T. Leavens (eds.) OOPSLA Companion, pp. 933–940. ACM (2009)

[202] Sussman, G.J., Jr., G.L.S.: Constraints - a language for expressing almost-hierarchical descriptions. Artif. Intell. **14**(1), 1–39 (1980)

[203] Takeuchi, K., Honda, K., Kubo, M.: An interaction-based language and its typing system. In: C. Halatsis, D.G. Maritsas, G. Philokyprou, S. Theodoridis (eds.) PARLE, *LNCS*, vol. 817, pp. 398–413. Springer (1994)

[204] Toro-Bermúdez, M., Desainte-Catherine, M.: Concurrent constraints conditional-branching timed interactive scores. In: Sound and Music Computing conference. Barcelona, Spain (2010)

[205] Ueda, K., Kato, N., Hara, K., Mizuno, K.: Lmntal as a unifying declarative language: Live demonstration. In: S. Etalle, M. Truszczynski (eds.) ICLP, *LNCS*, vol. 4079, pp. 457–458. Springer (2006)

[206] Valencia, F.D.: Decidability of infinite-state timed ccp processes and first-order ltl. Theor. Comput. Sci. **330**(3), 577–607 (2005)

[207] Varejao, F.M., Fromherz, M.P., Garcia, A.C.B., de Souza, C.S.: An integrated framework for the specification and design of reprographic machines. In: Thirteenth Int. Conf. on Applications of Artificial Intelligence in Engineering. Computational Mechanics Publications (1998)

[208] Victor, B., Parrow, J.: Concurrent constraints in the fusion calculus. In: K.G. Larsen, S. Skyum, G. Winskel (eds.) ICALP, *LNCS*, vol. 1443, pp. 455–469. Springer (1998)

[209] Wahls, T., Leavens, G.T., Baker, A.L.: Executing formal specifications with concurrent constraint programming. Autom. Softw. Eng. **7**(4), 315–343 (2000)

[210] Waltz, D.L.: Gene freuder and the roots of constraint computation. Constraints **11**(2-3), 87–89 (2006)

[211] Wischik, L., Gardner, P.: Explicit fusions. Theor. Comput. Sci. **340**(3), 606–630 (2005)

[212] Wong, H.C., Fromherz, M., Gupta, V., Saraswat, V.: Control-based programming of electro-mechanical controllers. In: IJCAI Workshop on Executable Temporal Logics (1995)

[213] Zaffanella, E., Giacobazzi, R., Levi, G.: Abstracting synchronization in concurrent constraint programming. Journal of Functional and Logic Programming **1997**(6) (1997)

Coalgebraic up-to techniques

Damien Pous[1⋆]

CNRS, LIP, ENS Lyon, France
damien.pous@ens-lyon.fr

1 The concrete case of finite automata

A simple algorithm for checking language equivalence of finite automata consists in trying to compute a *bisimulation* that relates them. This is possible because language equivalence can be characterised coinductively, as the largest bisimulation.

More precisely, consider an automaton $\langle S, t, o\rangle$, where S is a (finite) set of states, $t : S \to \mathcal{P}(S)^A$ is a non-deterministic transition function, and $o : S \to 2$ is the characteristic function of the set of accepting states. Such an automation gives rise to a determinised automaton $\langle \mathcal{P}(S), t^\sharp, o^\sharp\rangle$, where $t^\sharp : \mathcal{P}(S) \to \mathcal{P}(S)^A$ and $o^\sharp : \mathcal{P}(S) \to 2$ are the natural extensions of t and o to sets. A *bisimulation* is a relation R between sets of states such that for all sets of states X, Y, $X\ R\ Y$ entails:

1. $o^\sharp(X) = o^\sharp(Y)$, and
2. for all letter a, $t^\sharp_a(X)\ R\ t^\sharp_a(Y)$.

The coinductive characterisation is the following one: *two sets of states recognise the same language if and only if they are related by some bisimulation.*

Taking inspiration from concurrency theory [4,5], one can improve this proof technique by weakening the second item in the definition of bisimulation: given a function f on binary relations, a *bisimulation up to* f is a relation R between states such that for all sets X, Y, $X\ R\ Y$ entails:

1. $o^\sharp(X) = o^\sharp(Y)$, and
2. for all letter a, $t^\sharp_a(X)\ f(R)\ t^\sharp_a(Y)$.

For well-chosen functions f, bisimulations up to f are contained in a bisimulation, so that the improvement is sound. So is the function mapping each relation to its equivalence closure. In this particular case, one recover the standard algorithm by Hopcroft and Karp [2]: two sets can be skipped whenever they can already be related by a sequence of pairwise related states.

One can actually do more, by considering the function c mapping each relation to its congruence closure: the smallest equivalence relation which contains

⋆ Work partially funded by the PiCoq and PACE projects, ANR-10-BLAN-0305 and ANR-12IS02001

the argument, and which is compatible w.r.t. set union:

$$\frac{}{X\ c(R)\ X} \qquad \frac{Y\ c(R)\ X}{X\ c(R)\ Y} \qquad \frac{X\ c(R)\ Y \qquad Y\ c(R)\ Z}{X\ c(R)\ Z}$$

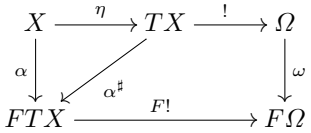
$$\frac{X\ R\ Y}{X\ c(R)\ Y} \qquad \frac{X_1\ c(R)\ Y_1 \qquad X_2\ c(R)\ Y_2}{X_1 \cup X_2\ c(R)\ Y_1 \cup Y_2}\ .$$

This is how we obtained HKC [1], an algorithm that can be exponentially faster than Hopcroft and Karp's algorithm or more recent antichain algorithms [7].

2 Generalisation to coalgebra

The above ideas generalise nicely, using the notion of λ-bialgebras [3].

Let T be a monad, F an endofunctor, and λ a distributive law $TF \Rightarrow FT$, a λ-*bialgebra* is a triple $\langle X, \alpha, \beta\rangle$, where $\langle X, \alpha\rangle$ is a F-coalgebra, $\langle X, \beta\rangle$ a T-algebra, and $\alpha \circ \beta = F\beta \circ \lambda_X \circ T\alpha$. Given such a λ-bialgebra, FT-algebra generalise non-deterministic automata: take $X \mapsto 2 \times X^A$ for F, and $X \mapsto \mathcal{P}_f X$ for T. Determinisation through the powerset construction can be generalised as follows [6], when the functor F has a final coalgebra $\langle \Omega, \omega\rangle$:



Bisimulations up-to can be expressed in a natural way in such a framework. One can in particular consider bisimulations up to congruence, where the congruence is taken w.r.t. the monad T: the fact that λ is a distributive law ensures that this improvement is always sound.

References

1. F. Bonchi and D. Pous. Checking NFA equivalence with bisimulations up to congruence. In *POPL*, pages 457–468. ACM, 2013.
2. J. E. Hopcroft and R. M. Karp. A linear algorithm for testing equivalence of finite automata. Technical Report 114, Cornell University, December 1971.
3. B. Klin. Bialgebras for structural operational semantics: An introduction. *TCS*, 412(38):5043–5069, 2011.
4. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
5. D. Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8:447–479, 1998.
6. A. Silva, F. Bonchi, M. Bonsangue, and J. Rutten. Generalizing the powerset construction, coalgebraically. In *Proc. FSTTCS*, volume 8 of *LIPIcs*, pages 272–283. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.
7. M. De Wulf, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Antichains: A new algorithm for checking universality of finite automata. In *Proc. CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 17–30. Springer, 2006.

Saturated Semantics for Coalgebraic Logic Programming

Filippo Bonchi and Fabio Zanasi

ENS Lyon, U. de Lyon, CNRS, INRIA, UCBL

Abstract. A series of recent papers introduces a coalgebraic semantics for logic programming, where the behavior of a goal is represented by a *parallel* model of computation called coinductive tree. This semantics fails to be compositional, in the sense that the coalgebra formalizing such behavior does not commute with the substitutions that may apply to a goal. We suggest that this is an instance of a more general phenomenon, occurring in the setting of interactive systems (in particular, nominal process calculi), when one tries to model their semantics with coalgebrae on presheaves. In those cases, compositionality can be obtained through *saturation*. We apply the same approach to logic programming: the resulting semantics is compositional and enjoys an elegant formulation in terms of coalgebrae on presheaves and their right Kan extensions.

1 Introduction

Coalgebrae on presheaves have been successfully employed to provide semantics to *nominal* calculi: sophisticated process calculi with complex mechanisms for variable binding, like the π-calculus [17, 14, 37, 16, 15]. The idea is to have an index category $\mathbf{C}$ of interfaces (or names), and encode as a presheaf $\mathcal{F}\colon \mathbf{C} \to \mathbf{Set}$ the mapping of any object i of $\mathbf{C}$ to the set of states having i as interface, and any arrow $f\colon i \to j$ to a function switching the interface of states from i to j. The operational semantics of the calculus will arise as a notion of transition between states, that is, as a coalgebra $\alpha\colon \mathcal{F} \to \mathcal{B}(\mathcal{F})$, where $\mathcal{B}\colon \mathbf{Set}^{\mathbf{C}} \to \mathbf{Set}^{\mathbf{C}}$ is a functor on presheaves encoding the kind of behavior that we want to express.

As an arrow in a presheaf category, α has to be a natural transformation, i.e. it should commute with arrows $f\colon i \to j$ in the index category $\mathbf{C}$. Unfortunately, this naturality requirement may fail when the structure of $\mathbf{C}$ is rich enough, as for instance when non-injective substitutions [18, 33, 38] or name fusions [32, 5] occur. As a concrete example, consider the π-calculus term $t = \bar{a}\langle x\rangle | b(y)$ consisting of a process $\bar{a}\langle x\rangle$ sending a message x on a channel named a, in parallel with $b(y)$ receiving a message on a channel named b. Since the names a and b are different, the two processes cannot synchronize. Conversely the term $t\theta = \bar{a}\langle x\rangle | a(y)$, that is obtained by applying the substitution θ mapping b to a, can synchronize. If θ is an arrow of the index category $\mathbf{C}$, then the operational semantics α is not natural since $\alpha(t\theta) \neq \alpha(t)\overline{\theta}$, where $\overline{\theta}$ denotes the application of θ to the transitions of t. As a direct consequence, also the unique morphism to the terminal coalgebra is not natural: this means that the abstract semantics of π-calculus is not *compositional* - in other words, bisimilarity is not a congruence

w.r.t. name substitutions. In order to make bisimilarity a congruence, Sangiorgi introduced in [36] *open bisimilarity*, that is defined by considering the transitions of processes under *all* possible name substitutions θ.

The approach of *saturated semantics* [6, 8] can be seen as a generalization of open bisimilarity, relying on analogous principles: the operational semantics α is "saturated" w.r.t. the arrows of the index category $\mathbf{C}$, resulting in a natural transformation $\alpha^\sharp$ in $\mathbf{Set}^{\mathbf{C}}$. In [5, 34], this is achieved by first shifting the definition of α to the category $\mathbf{Set}^{|\mathbf{C}|}$ of presheaves indexed by the discretization $|\mathbf{C}|$ of $\mathbf{C}$. Since $|\mathbf{C}|$ does not have other arrow than the identities, α is trivially a natural transformation in this setting. The source of α is $\mathcal{U}(\mathcal{F}) \in \mathbf{Set}^{|\mathbf{C}|}$, where $\mathcal{U}\colon \mathbf{Set}^{\mathbf{C}} \to \mathbf{Set}^{|\mathbf{C}|}$ is a forgetful functor defined by composition with the inclusion $\iota\colon |\mathbf{C}| \to \mathbf{C}$. The functor $\mathcal{U}$ has a right adjoint $\mathcal{K}\colon \mathbf{Set}^{|\mathbf{C}|} \to \mathbf{Set}^{\mathbf{C}}$ sending a presheaf to its *right Kan extension* along ι. The adjoint pair $\mathcal{U} \dashv \mathcal{K}$ induces an isomorphism $(-)^\sharp_{X,Y}\colon \mathbf{Set}^{|\mathbf{C}|}[\mathcal{U}(X), Y] \to \mathbf{Set}^{\mathbf{C}}[X, \mathcal{K}(Y)]$ mapping α to $\alpha^\sharp$. The latter is a natural transformation in $\mathbf{Set}^{\mathbf{C}}$ and, consequently, the abstract semantics results to be compositional.

In this paper, we show that the saturated approach can be fruitfully instantiated to *coalgebraic logic programming* [23, 25, 24, 26], which consists of a novel semantics for logic programming and a parallel resolution algorithm based on *coinductive trees*. These are a variant of and-or trees [21] modeling *parallel* implementations of logic programming, where the soundness of the derivations represented by a tree is guaranteed by the restriction to *term-matching* (whose algorithm, differently from unification, is parallelizable [13]).

There are two analogies with the π-calculus: (a) the state space is modeled by a presheaf on the index category $\mathbf{L}_{\mathbf{\Sigma}}^{op}$, that is the (opposite) *Lawvere Theory* associated with some signature Σ; (b) the operational semantics given in [25] fails to be a natural transformation in $\mathbf{Set}^{\mathbf{L}_{\mathbf{\Sigma}}^{op}}$: Example 2 provides a counter-example which is similar to the π-calculus term t discussed above. The authors of [25] obviate to (b) by relaxing naturality to *lax naturality*: the operational semantics p of a logic program is given as an arrow in the category $Lax(\mathbf{L}_{\mathbf{\Sigma}}^{op}, \mathbf{Poset})$ of locally ordered functors $\mathcal{F}\colon \mathbf{L}_{\mathbf{\Sigma}}^{op} \to \mathbf{Poset}$ and lax natural transformations between them. They show the existence of a cofree comonad that induces a morphism $[\![-]\!]_p$ mapping atoms (i.e., atomic formulae) to coinductive trees. Since $[\![-]\!]_p$ is not natural but lax natural, the semantics provided by coinductive trees is not compositional, in the sense that, for some atoms A and substitution θ,

$$[\![A\theta]\!]_p \neq [\![A]\!]_p\overline{\theta}$$

where $[\![A\theta]\!]_p$ is the coinductive tree associated with $A\theta$ and $[\![A]\!]_p\overline{\theta}$ denotes the result of applying θ to each atom occurring in the tree $[\![A]\!]_p$.

Instead of introducing laxness, we propose to tackle the non-naturality of p with a saturated approach. It turns out that, in the context of logic programming, the saturation map $(-)^\sharp$ has a neat description in terms of substitution mechanisms: while p performs *term-matching* between the atoms and the heads of clauses of a given logic program, its saturation $p^\sharp$ (given as a morphism in $\mathbf{Set}^{\mathbf{L}_{\mathbf{\Sigma}}^{op}}$) performs *unification*. It is worth to remark here that not only most general unifiers are considered but *all* possible unifiers.

A cofree construction leading to a map $[\![-]\!]_{p^\sharp}$ can be obtained by very standard categorical tools, such as terminal sequences [1]. This is possible because, as **Set**, both $\mathbf{Set}^{\mathbf{L}_\Sigma^{op}}$ and $\mathbf{Set}^{|\mathbf{L}_\Sigma^{op}|}$ are (co)complete categories, whereas in the lax approach, $Lax(\mathbf{L}_\Sigma^{op}, \mathbf{Poset})$ not being (co)complete, more indirect and more sophisticated categorical constructions are needed [24, Sec. 4]. By naturality of $p^\sharp$, the semantics given by $[\![-]\!]_{p^\sharp}$ turns out to be compositional, as in the desiderata. Analogously to $[\![-]\!]_p$, also $[\![-]\!]_{p^\sharp}$ maps atoms to tree structures, which we call *saturated trees*. They generalize coinductive trees, in the sense that the latter can be seen as a "desaturation" of saturated trees, where all unifiers that are not term-matchers have been discarded. This observation leads to a *translation* from saturated to coinductive trees, based on the counit ϵ of the adjunction $\mathcal{U} \dashv \mathcal{K}$. It follows that our framework encompasses the semantics in [25, 24].

Analogously to what is done in [24], we propose a notion of *refutation subtree* of a given saturated tree, intuitively corresponding to an SLD-refutation of an atomic goal in a program. This leads to a result of soundness and completeness of our semantics with respect to SLD-resolution, crucially using both compositionality and the translation into coinductive trees.

Related works. Apart from [23, 25, 24], there exist other categorical perspectives on (extensions of) logic programming, such as [12, 22, 31, 3]. Amongst these, the most relevant for us is [8] since it exploits a form of saturation: states representing formulae are both instantiated by substitution and contextualized by other formulae in "and". Beyond logic programming, the idea of exploiting saturation to achieve compositionality is even older than [36] (see e.g. [35]). As far as we know, [11] is the first work where saturation is explored in terms of coalgebrae. It is interesting to note that, in [10], a subset of the same authors also proposed laxness as a solution for the lack of compositionality of Petri-nets.

A third approach, alternative to laxness and saturation, may be possible by taking a special kind of "powerobject" functor as done in [32, 38] for giving a coalgebraic semantics to fusion and open π-calculus. We have chosen saturated semantics for its generality: it works for any behavioral functor $\mathcal{B}$ and it models a phenomenon that occurs in many different computational models (see e.g. [4]).

Synopsis. Section 2 introduces the background on logic programming and its coalgebraic semantics. In Section 3 we propose saturated semantics as an alternative to laxness and we show that it is compositional. Section 4 builds a bridge between the different approaches. We define a translation from saturated to coinductive trees. In Section 5 the results of the previous two sections are used to deduce soundness and completeness of saturated semantics with respect to SLD-resolution.

2 Coalgebraic Logic Programming

In this section we recall the framework of coalgebraic logic programming, as introduced in [23, 25, 24]. For this purpose, we first fix some terminology and notation, mainly concerning category theory and logic programming.

Given a (small) category $\mathbf{C}$, $|\mathbf{C}|$ denotes the category with the same objects as $\mathbf{C}$ but no other arrow than the identities. With a little abuse of notation, $o \in |\mathbf{C}|$ indicates that o is an object of $\mathbf{C}$ and $\mathbf{C}[o_1, o_2]$ the set of arrows from o_1 to o_2. A $\mathbf{C}$-indexed *presheaf* is any functor $\mathcal{G} \colon \mathbf{C} \to \mathbf{Set}$. We write $\mathbf{Set}^{\mathbf{C}}$ for the category of $\mathbf{C}$-indexed presheaves and natural transformations between them. Given a functor $\mathcal{B} \colon \mathbf{C} \to \mathbf{C}$, a $\mathcal{B}$-*coalgebra* on $o \in |\mathbf{C}|$ is an arrow $p \colon o \to \mathcal{B}(o)$.

We fix a *signature* Σ of function symbols, each equipped with a fixed arity, and a countably infinite set $\mathit{Var} = \{x_1, x_2, x_3, \dots\}$ of variables. We model substitutions and unification of terms over Σ and Var according to the categorical perspective of [19, 9]. To this aim, let the (opposite) *Lawvere Theory* of Σ be a category $\mathbf{L}_{\mathbf{\Sigma}}^{op}$ where objects are natural numbers, with $n \in |\mathbf{L}_{\mathbf{\Sigma}}^{op}|$ intuitively representing variables $x_1, x_2, \dots, x_n$ from Var. For any two $n, m \in |\mathbf{L}_{\mathbf{\Sigma}}^{op}|$, the set $\mathbf{L}_{\mathbf{\Sigma}}^{op}[n, m]$ consists of all n-tuples $\langle t_1, \dots, t_n \rangle$ of terms where only variables among $x_1, \dots, x_m$ occur. The identity on $n \in |\mathbf{L}_{\mathbf{\Sigma}}^{op}|$, denoted by id_n, is given by the tuple $\langle x_1, \dots, x_n \rangle$. The composition of $\langle t_1^1, \dots, t_n^1 \rangle \colon n \to m$ and $\langle t_1^2, \dots, t_m^2 \rangle \colon m \to m'$ is the tuple $\langle t_1, \dots, t_n \rangle \colon n \to m'$, where t_i is the term t_i^1 in which every variable x_j has been replaced with t_j^2, for $1 \leq j \leq m$ and $1 \leq i \leq n$.

We call *substitutions* the arrows of $\mathbf{L}_{\mathbf{\Sigma}}^{op}$ and use Greek letters θ, σ and τ to denote them. Given $\theta_1 \colon n \to m_1$ and $\theta_2 \colon n \to m_2$, a *unifier* of θ_1 and θ_2 is a pair of substitutions $\sigma \colon m_1 \to m$ and $\tau \colon m_2 \to m$, where m is some object of $\mathbf{L}_{\mathbf{\Sigma}}^{op}$, such that $\sigma \circ \theta_1 = \tau \circ \theta_2$. The *most general unifier* of θ_1 and θ_2 is a unifier with a universal property, i.e. a pushout of the diagram $m_1 \xleftarrow{\theta_1} n \xrightarrow{\theta_2} m_2$.

An *alphabet* $\mathcal{A}$ consists of a signature Σ, a set of variables Var and a set of predicate symbols $P, P_1, P_2, \dots$ each assigned an arity. Given P of arity n and Σ-terms $t_1, \dots, t_n$, $P(t_1, \dots, t_n)$ is called an *atom*. We use Latin capital letters $A, B, \dots$ for atoms. Given a substitution $\theta = \langle t_1, \dots, t_n \rangle \colon n \to m$ and an atom A with variables among $x_1, \dots, x_n$, we adopt the standard notation of logic programming in denoting with $A\theta$ the atom obtained by replacing x_i with t_i in A, for $1 \leq i \leq n$. The atom $A\theta$ is called a *substitution instance* of A. The notation $\{A_1, \dots, A_m\}\theta$ is a shorthand for $\{A_1\theta, \dots, A_m\theta\}$. Given atoms A_1 and A_2, we say that A_1 *unifies* with A_2 (equivalently, they are *unifiable*) if they are of the form $A_1 = P(t_1, \dots, t_n)$, $A_2 = P(t'_1, \dots, t'_n)$ and a unifier $\langle \sigma, \tau \rangle$ of $\langle t_1, \dots, t_n \rangle$ and $\langle t'_1, \dots, t'_n \rangle$ exists. Observe that, by definition of unifier, this amounts to saying that $A_1\sigma = A_2\tau$. *Term matching* is a particular case of unification, where σ is the identity substitution. In this case we say that $\langle \sigma, \tau \rangle$ is a *term-matcher* of A_1 and A_2, meaning that $A_1 = A_2\tau$.

A *logic program* $\mathbb{P}$ consists of a finite set of *clauses* C written as $H \leftarrow B_1, \dots, B_k$. The components H and $B_1, \dots, B_k$ are atoms, where H is called the *head* of C and $B_1, \dots, B_k$ form the *body* of C. One can think of $H \leftarrow B_1, \dots, B_k$ as representing the first-order formula $(B_1 \wedge \dots \wedge B_k) \to H$. We say that $\mathbb{P}$ is *ground* if only ground atoms (i.e. without variables) occur in its clauses. The central algorithm of logic programming is SLD-resolution [27, 28], checking whether a finite set of atoms (called a *goal*) is *refutable* in $\mathbb{P}$ and giving a substitution called *computed answer* as output: we refer to Appendix A for more details. Relevant for our exposition are *and-or trees* [21], which represent

executions of SLD-resolution exploiting two forms of parallelism: *and-parallelism*, corresponding to simultaneous refutation-search of multiple atoms in a goal, and *or-parallelism*, exploring multiple attempts to refute the same goal.

Definition 1. *Given a logic program $\mathbb{P}$ and an atom A, the* (parallel) and-or tree *for A in $\mathbb{P}$ is the possibly infinite tree T satisfying the following properties:*

1. *Each node in T is either an and-node or an or-node.*
2. *Each and-node is labeled with one atom and its children are or-nodes.*
3. *The root of T is an and-node labeled with A.*
4. *Each or-node is labeled with $\bullet$ and its children are and-nodes.*
5. *For every and-node s in T, let A' be its label. For every clause $H \leftarrow B_1, \ldots, B_k$ of $\mathbb{P}$ and most general unifier $\langle \sigma, \tau \rangle$ of A' and H, s has exactly one child t, and viceversa. For each atom B in $\{B_1, \ldots, B_k\}\tau$, t has exactly one child labeled with B, and viceversa.*

As standard for any tree, we have a notion of *depth*: the root is at depth 0 and depth $i+1$ is given by the children of nodes at depth i.

2.1 The Ground Case

We recall the coalgebraic semantics of ground logic programs introduced in [23]. For the sequel we fix an alphabet $\mathcal{A}$, a set At of ground atoms and a ground logic program $\mathbb{P}$. The behavior of $\mathbb{P}$ is represented by a coalgebra $p\colon At \to \mathcal{P}_f\mathcal{P}_f(At)$ on **Set**, where $\mathcal{P}_f$ is the finite powerset functor and p is defined as follows:

$$p\colon\ A \ \mapsto\ \{\{B_1, \ldots, B_k\} \ \mid\ H \leftarrow B_1, \ldots, B_k \text{ is a clause of } \mathbb{P} \ \text{ and } A = H\}.$$

The idea is that p maps an atom $A \in At$ to the set of bodies of clauses of $\mathbb{P}$ whose head H unifies with A, i.e. (in the ground case) $A = H$. Therefore $p(A) \in \mathcal{P}_f\mathcal{P}_f(At)$ can be seen as representing the and-or tree of A in $\mathbb{P}$ up to depth 2, according to Definition 1: each element $\{B_1, \ldots, B_k\}$ of $p(A)$ corresponds to a child of the root, whose children are labeled with $B_1, \ldots, B_k$. The full tree is recovered as an element of $\mathcal{C}(\mathcal{P}_f\mathcal{P}_f)(At)$, where $\mathcal{C}(\mathcal{P}_f\mathcal{P}_f)$ is the *cofree comonad* on $\mathcal{P}_f\mathcal{P}_f$, standardly provided by the following construction [1, 39].

Construction 1 *The terminal sequence for the functor $At \times \mathcal{P}_f\mathcal{P}_f(-)\colon \mathbf{Set} \to \mathbf{Set}$ consists of sequences of objects X_α and arrows $\delta_\alpha\colon X_{\alpha+1} \to X_\alpha$, defined by induction on α as follows.*

$$X_\alpha := \begin{cases} At & \alpha = 0 \\ At \times \mathcal{P}_f\mathcal{P}_f(X_\beta) & \alpha = \beta + 1 \end{cases} \qquad \delta_\alpha := \begin{cases} \pi_1 & \alpha = 0 \\ id_{At} \times \mathcal{P}_f\mathcal{P}_f(\delta_\beta) & \alpha = \beta + 1 \end{cases}$$

For α a limit ordinal, X_α is given as a limit of the sequence and a function $\delta_\alpha\colon X_\alpha \to X_\beta$ is given for each $\beta < \alpha$ by the limiting property of X_α.

By [39] it follows that the sequence given above converges to a limit X_γ such that $X_\gamma \cong X_{\gamma+1}$. Since $X_{\gamma+1}$ is defined as $At \times \mathcal{P}_f\mathcal{P}_f(X_\gamma)$, there is a projection function $\pi_2\colon X_{\gamma+1} \to \mathcal{P}_f\mathcal{P}_f(X_\gamma)$ which makes $\pi_2 \circ \delta_\gamma^{-1}\colon X_\gamma \to \mathcal{P}_f\mathcal{P}_f(X_\gamma)$ the cofree $\mathcal{P}_f\mathcal{P}_f$-coalgebra *on At. This induces the* cofree comonad $\mathcal{C}(\mathcal{P}_f\mathcal{P}_f)\colon \mathbf{Set} \to \mathbf{Set}$ *on $\mathcal{P}_f\mathcal{P}_f$ as a functor mapping At to X_γ.*

As the elements of the cofree comonad on $\mathcal{P}_f$ are standardly presented as finitely branching trees [39], those for $\mathcal{P}_f\mathcal{P}_f$ can be seen as finitely branching trees with two sorts of nodes occurring at alternating depth. We now define a $\mathcal{C}(\mathcal{P}_f\mathcal{P}_f)$-coalgebra $[\![-]\!]_p \colon At \to \mathcal{C}(\mathcal{P}_f\mathcal{P}_f)(At)$.

Construction 2 *Given a ground program $\mathbb{P}$, let $p \colon At \to \mathcal{P}_f\mathcal{P}_f(At)$ be the coalgebra associated with $\mathbb{P}$. We define a cone $\{p_\alpha \colon At \to X_\alpha\}_{\alpha<\gamma}$ on the terminal sequence of Construction 1 as follows:*

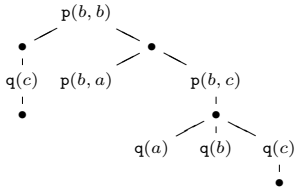
$$p_\alpha := \begin{cases} id_{At} & \alpha = 0 \\ \langle id_{At}, (\mathcal{P}_f\mathcal{P}_f(p_\beta) \circ p)\rangle & \alpha = \beta + 1. \end{cases}$$

For α a limit ordinal, $p_\alpha \colon At \to X_\alpha$ is provided by the limiting property of X_α. Then in particular $X_\gamma = \mathcal{C}(\mathcal{P}_f\mathcal{P}_f)(At)$ yields a function $[\![-]\!]_p \colon At \to \mathcal{C}(\mathcal{P}_f\mathcal{P}_f)(At)$.

Given an atom $A \in At$, the tree $[\![A]\!]_p \in \mathcal{C}(\mathcal{P}_f\mathcal{P}_f)(At)$ is built by iteratively applying the map p, first to A, then to each atom in $p(A)$, and so on. For each natural number m, p_m maps A to its and-or tree up to depth m. As shown in [23], the limit $[\![-]\!]_p$ of all such approximations provides the full and-or tree of A.

Example 1. Consider the ground logic program on the left-hand side, based on an alphabet consisting of a signature $\{a^0, b^0, c^0\}$ and predicates $\mathtt{p}(-,-)$, $\mathtt{q}(-)$. The and-or tree $[\![\mathtt{p}(b,b)]\!]_p \in \mathcal{C}(\mathcal{P}_f\mathcal{P}_f)(At)$ is depicted on the right-hand side.

$\mathtt{p}(b,c) \leftarrow \mathtt{q}(a), \mathtt{q}(b), \mathtt{q}(c)$
$\mathtt{p}(b,b) \leftarrow \mathtt{p}(b,a), \mathtt{p}(b,c)$
$\mathtt{p}(b,b) \leftarrow \mathtt{q}(c)$
$\mathtt{q}(c) \leftarrow$



2.2 The General Case

We recall the extension of the coalgebraic semantics to arbitrary (i.e. possibly non-ground) logic programs presented in [25, 24]. In presence of variables, and-or trees are not guaranteed to represent sound derivations, whence *coinductive trees* are introduced as a sound variant of and-or trees, where unification is restricted to term-matching. We refer to [25, 24] and Appendix A for more details.

Before formally defining coinductive trees, it is worth recalling that, in [25], the collection of atoms (based on an alphabet $\mathcal{A}$) is modeled as a presheaf $At \colon \mathbf{L}_{\mathbf{\Sigma}}^{op} \to \mathbf{Set}$. The index category is the (opposite) *Lawvere Theory* $\mathbf{L}_{\mathbf{\Sigma}}^{op}$ of Σ, as defined above. For each natural number $n \in |\mathbf{L}_{\mathbf{\Sigma}}^{op}|$, $At(n)$ is defined as the set of atoms with variables among $x_1, \ldots, x_n$. Given an arrow $\theta \in \mathbf{L}_{\mathbf{\Sigma}}^{op}[n, m]$, the function $At(\theta) \colon At(n) \to At(m)$ is defined by substitution, i.e. $At(\theta)(A) := A\theta$. By definition, whenever an atom A belongs to $At(n)$, then it also belongs to $At(n')$, for all $n' \geq n$. However, the occurrences of the same atom in $At(n)$ and $At(n')$ (for $n \neq n'$) are considered distinct: the atoms $A \in At(n)$ and $A \in At(n')$ can be thought of as two states $x_1, \ldots, x_n \vdash A$ and $x_1, \ldots, x_{n'} \vdash A$ with two different interfaces $x_1, \ldots, x_n$ and $x_1, \ldots, x_{n'}$. For this reason, when referring to an atom A, it is important to always specify the set $At(n)$ to which it belongs.

Definition 2. *Given a logic program $\mathbb{P}$, a natural number n and an atom $A \in At(n)$, the* n-coinductive tree *for A in $\mathbb{P}$ is the possibly infinite tree T satisfying properties 1-4 of Definition 1 and property 5 replaced by the following*[1]*:*

5. *For every and-node s in T, let $A' \in At(n)$ be its label. For every clause $H \leftarrow B_1, \ldots, B_k$ of $\mathbb{P}$ and term-matcher $\langle id_n, \tau \rangle$ of A' and H, with $B_1\tau, \ldots, B_k\tau \in At(n)$, s has exactly one child t, and viceversa. For each atom B in $\{B_1, \ldots, B_k\}\tau$, t has exactly one child labeled with B, and viceversa.*

We recall from [25] the categorical formalization of this class of trees. The first step is to generalize the definition of the coalgebra p associated with a program $\mathbb{P}$. Definition 2 suggests how p should act on an atom $A \in At(n)$, for a fixed n:

$$A \mapsto \{\{B_1, \ldots, B_k\}\tau \mid H \leftarrow B_1, \ldots, B_k \text{ is a clause of } \mathbb{P}, \\ A = H\tau \text{ and } B_1\tau, \ldots, B_k\tau \in At(n)\}. \qquad (1)$$

For each clause $H \leftarrow B_1, \ldots, B_k$, there might be infinitely (but countably) many substitutions τ such that $A = H\tau$ (see e.g. [25]). Thus the object on the right-hand side of (1) will be associated with the functor $\mathcal{P}_c\mathcal{P}_f \colon \mathbf{Set} \to \mathbf{Set}$, where $\mathcal{P}_c$ and $\mathcal{P}_f$ are respectively the countable powerset functor and the finite powerset functor. In order to formalize this as a coalgebra on $At \colon \mathbf{L}_{\mathbf{\Sigma}}^{op} \to \mathbf{Set}$, consider liftings $\widetilde{\mathcal{P}_c} \colon \mathbf{Set}^{\mathbf{L}_{\mathbf{\Sigma}}^{op}} \to \mathbf{Set}^{\mathbf{L}_{\mathbf{\Sigma}}^{op}}$ and $\widetilde{\mathcal{P}_f} \colon \mathbf{Set}^{\mathbf{L}_{\mathbf{\Sigma}}^{op}} \to \mathbf{Set}^{\mathbf{L}_{\mathbf{\Sigma}}^{op}}$, standardly defined on presheaves $\mathcal{F} \colon \mathbf{L}_{\mathbf{\Sigma}}^{op} \to \mathbf{Set}$ by postcomposition respectively with $\mathcal{P}_c$ and $\mathcal{P}_f$. Then one would like to fix (1) as the definition of the n-component of a natural transformation $p \colon At \to \widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_f}(At)$. The key problem with this formulation is that p would *not* be a natural transformation, as shown by the following example.

Example 2. Consider the signature $\Sigma = \{cons^2, succ^1, zero^0, nil^0\}$ and the predicates $\mathtt{List}(-)$, $\mathtt{Nat}(-)$. The program `NatList`, encoding the definition of lists of natural numbers, will be our running example of a non-ground logic program.

$$\begin{array}{rcl@{\qquad\qquad}rcl} \mathtt{List}(cons(x_1, x_2)) & \leftarrow & \mathtt{Nat}(x_1), \mathtt{List}(x_2) & \mathtt{List}(nil) & \leftarrow & \\ \mathtt{Nat}(succ(x_1)) & \leftarrow & \mathtt{Nat}(x_1) & \mathtt{Nat}(zero) & \leftarrow & \end{array}$$

Fix a substitution $\theta = \langle nil \rangle \colon 1 \to 0$ and, for each $n \in |\mathbf{L}_{\mathbf{\Sigma}}^{op}|$, suppose that $p(n) \colon At(n) \to \widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_f}(At)(n)$ is defined according to (1). Then the square

[1] The notion of coinductive tree that we define here, differently from the one given in [24, Def.4.1], allows at a given depth multiple or-nodes to represent the same clause (with different term-matchers). In fact, it corresponds to the notion of coinductive forest of breadth n [24, Def.4.4], the only difference being that we "glue" together all trees of the forest into a single tree. This formulation is more convenient for our presentation. Observe that the adequacy theorem for the coalgebraic semantics [24, Th.4.5] is formulated in terms of coinductive forests and their breadth, whence one can also express it in terms of our notion of n-coinductive tree.

$$\begin{array}{ccc} At(1) & \xrightarrow{p(1)} & \widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_f}(At)(1) \\ {\scriptstyle At(\theta)}\downarrow & & \downarrow{\scriptstyle \widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_f}(At)(\theta)} \\ At(0) & \xrightarrow[p(0)]{} & \widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_f}(At)(0) \end{array}$$

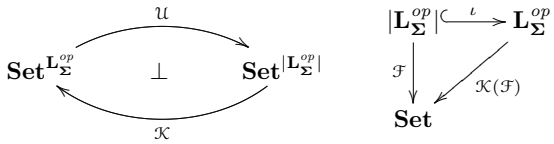
does not commute. A counterexample is provided by the atom $\texttt{List}(x_1) \in At(1)$. Passing through the bottom-left corner of the square, $\texttt{List}(x_1)$ is mapped first to $\texttt{List}(nil) \in At(0)$ and then to $\{\emptyset\} \in \widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_f}(At)(0)$ - intuitively, this yields a refutation of the goal $\{\texttt{List}(x_1)\}$ with substitution of x_1 with nil. Passing through the top-right corner, $\texttt{List}(x_1)$ is mapped first to $\emptyset \in \widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_f}(At)(1)$ and then to $\emptyset \in \widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_f}(At)(0)$, i.e. the computation ends up in a failure.

In [25, Sec.4] the authors overcome this difficulty by relaxing the naturality requirement. The morphism p is defined as a $\breve{\mathcal{P}}_c\breve{\mathcal{P}}_f$-coalgebra in the category $Lax(\mathbf{L}_{\mathbf{\Sigma}}^{op}, \mathbf{Poset})$ of locally ordered functors $\mathcal{F}\colon \mathbf{L}_{\mathbf{\Sigma}}^{op} \to \mathbf{Poset}$ and *lax* natural transformations, with each component $p(n)$ given according to (1) and $\breve{\mathcal{P}}_c\breve{\mathcal{P}}_f$ the extension of $\widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_f}$ to an endofunctor on $Lax(\mathbf{L}_{\mathbf{\Sigma}}^{op}, \mathbf{Poset})$.

The lax approach fixes the problem, but presents also some drawbacks. Unlike the categories $\mathbf{Set}$ and $\mathbf{Set}^{\mathbf{L}_{\mathbf{\Sigma}}^{op}}$, $Lax(\mathbf{L}_{\mathbf{\Sigma}}^{op}, \mathbf{Poset})$ is neither complete nor cocomplete, meaning that a cofree comonad on $\breve{\mathcal{P}}_c\breve{\mathcal{P}}_f$ cannot be retrieved through the standard Constructions 1 and 2 that were used in the ground case. Moreover, the category of $\breve{\mathcal{P}}_c\breve{\mathcal{P}}_f$-coalgebrae becomes problematic, because coalgebra maps are subject to a commutativity property stricter than the one of lax natural transformations. These two issues force the formalization of non-ground logic program to use quite different (and more sophisticated) categorical tools than the ones employed for the ground case. Finally, as stressed in the Introduction, the laxness of p makes the resulting semantics not compositional.

3 Saturated Semantics

Motivated by the observations of the previous section, we propose a *saturated approach* to the semantics of logic programs. For this purpose, we consider an adjunction between presheaf categories as depicted on the left.



The left adjoint $\mathcal{U}$ is the forgetful functor, given by precomposition with the inclusion functor $\iota\colon |\mathbf{L}_{\mathbf{\Sigma}}^{op}| \hookrightarrow \mathbf{L}_{\mathbf{\Sigma}}^{op}$. As shown in [29, Th.X.1], $\mathcal{U}$ has a right adjoint

$\mathcal{K}\colon \mathbf{Set}^{|\mathbf{L}_\Sigma^{op}|} \to \mathbf{Set}^{\mathbf{L}_\Sigma^{op}}$ sending $\mathcal{F}\colon |\mathbf{L}_\Sigma^{op}| \to \mathbf{Set}$ to its *right Kan extension* along ι. This is a presheaf $\mathcal{K}(\mathcal{F})\colon \mathbf{L}_\Sigma^{op} \to \mathbf{Set}$ mapping an object n of $\mathbf{L}_\Sigma^{op}$ to

$$\mathcal{K}(\mathcal{F})(n) := \prod_{\theta \in \mathbf{L}_\Sigma^{op}[n,m]} \mathcal{F}(m)$$

where m is any object of $\mathbf{L}_\Sigma^{op}$. Intuitively, $\mathcal{K}(\mathcal{F})(n)$ is a set of tuples indexed by arrows with source n and such that, at index $\theta\colon n \to m$, there are elements of $\mathcal{F}(m)$. We use $\dot{x}\ \dot{y}, \ldots$ to denote such tuples and we write $\dot{x}(\theta)$ to denote the element at index θ of the tuple $\dot{x}$. Alternatively, when it is important to show how the elements depend from the indexes, we use $\langle x \rangle_{\theta:n\to m}$ (or simply $\langle x \rangle_\theta$) to denote the tuple having at index θ the element x. With this notation, we can express the behavior of $\mathcal{K}(\mathcal{F})\colon \mathbf{L}_\Sigma^{op} \to \mathbf{Set}$ on an arrow $\theta\colon n \to m$ as

$$\mathcal{K}(\mathcal{F})(\theta)\colon \dot{x} \mapsto \langle \dot{x}(\sigma \circ \theta) \rangle_{\sigma:m\to m'}. \tag{2}$$

The tuple $\langle \dot{x}(\sigma \circ \theta) \rangle_\sigma$ in $\mathcal{K}(\mathcal{F})(m)$ can be intuitively read as follows: for each $\sigma \in \mathbf{L}_\Sigma^{op}[m, m']$, we let the element indexed by σ be the one which was indexed by $\sigma \circ \theta \in \mathbf{L}_\Sigma^{op}[n, m']$ in the input tuple $\dot{x}$.

All this concerns the behavior of $\mathcal{K}$ on the objects of $\mathbf{Set}^{|\mathbf{L}_\Sigma^{op}|}$. For an arrow $f\colon \mathcal{F} \to \mathcal{G}$ in $\mathbf{Set}^{|\mathbf{L}_\Sigma^{op}|}$, the natural transformation $\mathcal{K}(f)$ is defined as an indexwise application of f on tuples from $\mathcal{K}(\mathcal{F})$. For all $n \in |\mathbf{L}_\Sigma^{op}|$, $\dot{x} \in \mathcal{K}(\mathcal{F})(n)$,

$$\mathcal{K}(f)(n)\colon \dot{x} \mapsto \langle f(m)(\dot{x}(\theta)) \rangle_{\theta:n\to m}.$$

For any presheaf $\mathcal{F}\colon \mathbf{L}_\Sigma^{op} \to \mathbf{Set}$, the unit η of the adjunction is instantiated to a morphism $\eta_\mathcal{F}\colon \mathcal{F} \to \mathcal{KU}(\mathcal{F})$ given as follows: for all $n \in |\mathbf{L}_\Sigma^{op}|$, $X \in \mathcal{F}(n)$,

$$\eta_\mathcal{F}(n)\colon X \mapsto \langle \mathcal{F}(\theta)(X) \rangle_{\theta:n\to m}.$$

When taking $\mathcal{F}$ to be At, $\eta_{At}\colon At \to \mathcal{KU}(At)$ maps an atom to its *saturation*: for each $A \in At(n)$, the tuple $\eta_{At}(n)(A)$ consists of all substitution instances $At(\theta)(A) = A\theta$ of A, each indexed by the corresponding $\theta \in \mathbf{L}_\Sigma^{op}[n, m]$.

As shown in Example 2, given a program $\mathbb{P}$, the family of functions p defined by (1) fails to be a morphism in $\mathbf{Set}^{\mathbf{L}_\Sigma^{op}}$. However, it forms a morphism in $\mathbf{Set}^{|\mathbf{L}_\Sigma^{op}|}$

$$p\colon \mathcal{U}At \to \widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}(\mathcal{U}At)$$

where $\widehat{\mathcal{P}_c}$ and $\widehat{\mathcal{P}_f}$ denote the liftings of $\mathcal{P}_c$ and $\mathcal{P}_f$ to $\mathbf{Set}^{|\mathbf{L}_\Sigma^{op}|}$. The naturality requirement is trivially satisfied in $\mathbf{Set}^{|\mathbf{L}_\Sigma^{op}|}$, since $|\mathbf{L}_\Sigma^{op}|$ is discrete. The adjunction induces a morphism $p^\sharp\colon At \to \mathcal{K}\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}(At)$ in $\mathbf{Set}^{\mathbf{L}_\Sigma^{op}}$, defined as

$$At \xrightarrow{\eta_{At}} \mathcal{KU}(At) \xrightarrow{\mathcal{K}(p)} \mathcal{K}\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}(At). \tag{3}$$

In the sequel, we write $\mathcal{S}$ for $\mathcal{K}\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}$. The idea is to let $\mathcal{S}$ play the same role as $\mathcal{P}_f\mathcal{P}_f$ in the ground case, with the coalgebra $p^\sharp\colon At \to \mathcal{S}(At)$ encoding the program $\mathbb{P}$. An atom $A \in At(n)$ is mapped to $\langle p(m)(A\sigma) \rangle_{\sigma:n\to m}$, that is:

$$\begin{aligned} p^\sharp(n)\colon A \mapsto \langle \{\{B_1, \ldots, B_k\}\tau \mid H \leftarrow B_1, \ldots, B_k \text{ is a clause of } \mathbb{P}, \\ A\sigma = H\tau \text{ and } B_1\tau, \ldots, B_k\tau \in At(m)\} \rangle_{\sigma:n\to m}. \end{aligned} \tag{4}$$

Intuitively, $p^\sharp(n)$ retrieves all unifiers $\langle \sigma, \tau \rangle$ of A and heads of $\mathbb{P}$: first, $A\sigma \in At(m)$ arises as a component of the saturation of A, according to $\eta_{At}(n)$; then, the substitution τ is given by term-matching on $A\sigma$, according to $K(p)(m)$.

By naturality of $p^\sharp$, we achieve the property of "commuting with substitutions" that was precluded by the term-matching approach, as shown by the following rephrasing of Example 2.

Example 3. Consider the same square of Example 2, with $p^\sharp$ in place of p and $\mathcal{S}$ in place of $\widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_f}$. The atom $\mathtt{List}(x_1) \in At(1)$ together with the substitution $\theta = \langle nil \rangle \colon 1 \to 0$ does not constitute a counterexample to commutativity anymore. Indeed $p^\sharp(1)$ maps $\mathtt{List}(x_1)$ to the tuple $\langle p(n)(\mathtt{List}(x_1)\sigma) \rangle_{\sigma \colon 1 \to n}$, which is then mapped by $\mathcal{S}(At)(\theta)$ to $\langle p(n)(\mathtt{List}(x_1)\sigma' \circ \theta) \rangle_{\sigma' \colon 0 \to n}$ according to (2). Observe that the latter is just the tuple $\langle p(n)(\mathtt{List}(nil)\sigma') \rangle_{\sigma' \colon 0 \to n}$ obtained by applying first $At(\theta)$ and then $p^\sharp(0)$ to the atom $\mathtt{List}(x_1)$.

Another benefit of saturated semantics is that $p^\sharp \colon At \to \mathcal{S}(At)$ lives in a (co)complete category which behaves (pointwise) as **Set**. This allows us to follow the same steps as in the ground case, constructing a coalgebra for the cofree comonad $\mathcal{C}(\mathcal{S})$ as a straightforward generalization of Constructions 1 and 2.

Construction 3 *The terminal sequence for the functor $At \times \mathcal{S}(-) \colon \mathbf{Set}^{\mathbf{L}_\Sigma^{op}} \to \mathbf{Set}^{\mathbf{L}_\Sigma^{op}}$ consists of a sequence of objects X_α and arrows $\delta_\alpha \colon X_{\alpha+1} \to X_\alpha$, which are defined just as in Construction 1, with $\mathcal{S}$ replacing $\mathcal{P}_f\mathcal{P}_f$. As shown in Appendix B, this sequence converges to a limit X_γ such that $X_\gamma \cong X_{\gamma+1}$ and X_γ is the carrier of the cofree $\mathcal{S}$-coalgebra on At.*

Since $\mathcal{S}$ is accessible, the *cofree comonad* $\mathcal{C}(\mathcal{S})$ exists and maps At to X_γ given as in Construction 3. A $\mathcal{C}(\mathcal{S})$-coalgebra $[\![-]\!]_{p^\sharp} \colon At \to \mathcal{C}(\mathcal{S})(At)$ is given below.

Construction 4 *The terminal sequence for $At \times \mathcal{S}(-)$ induces a cone $\{p^\sharp_\alpha \colon At \to X_\alpha\}_{\alpha < \gamma}$ as in Construction 2 with $p^\sharp$ and $\mathcal{S}$ replacing p and $\mathcal{P}_f\mathcal{P}_f$. This yields a natural transformation $[\![-]\!]_{p^\sharp} \colon At \to X_\gamma$, where $X_\gamma = \mathcal{C}(\mathcal{S})(At)$.*

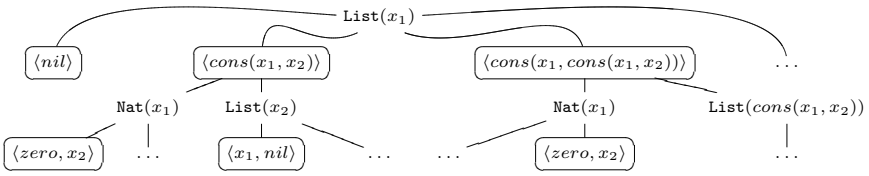
As in the ground case, the coalgebra $[\![-]\!]_{p^\sharp}$ is constructed as an iterative application of $p^\sharp$: we call *saturated tree* the associated tree structure.

Definition 3. *Given a logic program $\mathbb{P}$, a natural number n and an atom $A \in At(n)$, the* saturated tree *for A in $\mathbb{P}$ is the possibly infinite tree T satisfying properties 1-3 of Definition 1 and properties 4 and 5 replaced by the following:*

4. *Each or-node is labeled with a substitution σ and its children are and-nodes.*
5. *For every and-node s in T, let $A' \in At(n')$ be its label. For every clause $H \leftarrow B_1, \ldots, B_k$ of $\mathbb{P}$ and unifier $\langle \sigma, \tau \rangle$ of A' and H, with $\sigma \colon n' \to m'$ and $B_1\tau, \ldots, B_k\tau \in At(m')$, s has exactly one child t labeled with σ, and viceversa. For each atom B in $\{B_1, \ldots, B_k\}\tau$, t has exactly one child labeled with B, and viceversa.*

We have now seen three kinds of tree, exhibiting different substitution mechanisms. In saturated trees one considers all the unifiers, whereas in and-or trees and coinductive trees one restricts to most general unifiers and term-matchers respectively. Moreover, in a coinductive tree each and-node is labeled with an atom in $At(n)$ for a fixed n, while in a saturated tree n can dynamically change.

Example 4. Part of the infinite saturated tree of $\texttt{List}(x_1) \in At(1)$ in `NatList` is depicted below. Note that not all labels of and-nodes belong to $At(1)$, as it would be the case for a coinductive tree: such information is inherited from the label of the parent or-node, which is now a substitution. For instance, both $\texttt{Nat}(x_1)$ and $\texttt{List}(x_2)$ belong to $At(2)$, since their parent is labeled with $\langle cons(x_1, x_2)\rangle \colon 1 \to 2$ (using the convention that the target of a substitution is the largest index appearing among its variables).



We can generalize these observations to the following adequacy theorem.

Theorem 1. *Let $[\![-]\!]_{p^\sharp}$ be defined from a program $\mathbb{P}$ according to Construction 4. Then, for all n and $A \in At(n)$, the saturated tree of A in $\mathbb{P}$ is $[\![A]\!]_{p^\sharp}$.*

In the above theorem and in the rest of the paper, with an abuse of notation we use $[\![A]\!]_{p^\sharp}$ to denote the application of $[\![-]\!]_{p^\sharp}(n)$ to $A \in At(n)$ without mentioning the object $n \in |\mathbf{L}_{\mathbf{\Sigma}}^{op}|$. For an arrow $\theta \in \mathbf{L}_{\mathbf{\Sigma}}^{op}[n, m]$, we write $\overline{\theta}$ for $\mathcal{C}(\mathcal{S})(At)(\theta) \colon \mathcal{C}(\mathcal{S})(At)(n) \to \mathcal{C}(\mathcal{S})(At)(m)$. With this notation, we can state the following theorem that is an immediate consequence of the naturality of $[\![-]\!]_{p^\sharp}$.

Theorem 2 (Compositionality). *For all atoms $A \in At(n)$ and substitutions $\theta \in \mathbf{L}_{\mathbf{\Sigma}}^{op}[n, m]$,*

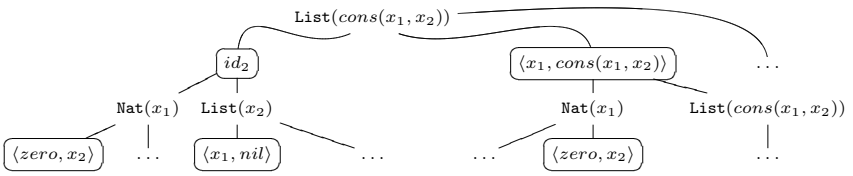
$$[\![A\theta]\!]_{p^\sharp} = [\![A]\!]_{p^\sharp}\overline{\theta}.$$

We conclude this section with a concrete description of the behavior of the operator $\overline{\theta}$, for a given substitution $\theta \in \mathbf{L}_{\mathbf{\Sigma}}^{op}[n, m]$. Let r be the root of a tree $T \in \mathcal{C}(\mathcal{S})(At)(n)$ and r' the root of $T\overline{\theta}$. Then

1. the node r has label A iff r' has label $A\theta$;
2. the node r has a child t with label $\sigma \circ \theta$ and children $t_1, \ldots, t_n$ iff r' has a child t' with label σ and children $t_1 \ldots t_n$.

Note that the children $t_1, \ldots, t_n$ are exactly the same in both trees: $\overline{\theta}$ only modifies the root and the or-nodes at depth 1 of T, while it leaves untouched all the others. This peculiar behavior can be better understood by observing that the definition of $\mathcal{K}(\mathcal{F})(\theta)$, as in (2), is independent of the presheaf $\mathcal{F}$. As a result, $\overline{\theta} = X_\gamma(\theta)$ is independent of all the X_αs built in Construction 3.

Example 5. Recall from Example 4 the saturated tree $[\![\texttt{List}(x_1)]\!]_{p^\sharp}$. For $\theta = \langle cons(x_1, x_2)\rangle$, the tree $[\![\texttt{List}(x_1)]\!]_{p^\sharp}\overline{\theta}$ is depicted below.



4 Desaturation

One of the main features of coinductive trees is to represent (sound) and-or parallel derivations of goals. This leads the authors of [24] to a resolution algorithm exploiting the two forms of parallelism [26]. Motivated by these developments, we include coinductive trees in our framework, showing how they can be obtained as a "desaturation" of saturated trees.

For this purpose, the key ingredient is given by the *counit* ϵ of the adjunction $\mathcal{U} \dashv \mathcal{K}$. Given a presheaf $\mathcal{F}\colon |\mathbf{L}_{\mathbf{\Sigma}}^{op}| \to \mathbf{Set}$, the morphism $\epsilon_{\mathcal{F}}\colon \mathcal{UK}(\mathcal{F}) \to \mathcal{F}$ is defined as follows: for all $n \in |\mathbf{L}_{\mathbf{\Sigma}}^{op}|$ and $\dot{x} \in \mathcal{UK}(\mathcal{F})(n)$,

$$\epsilon_{\mathcal{F}}(n)\colon \dot{x} \mapsto \dot{x}(id_n) \tag{5}$$

where $\dot{x}(id_n)$ is the element of the input tuple $\dot{x}$ which is indexed by the identity substitution $id_n \in \mathbf{L}_{\mathbf{\Sigma}}^{op}[n,n]$. In the logic programming perspective, the intuition is that, while the unit of the adjunction provides the saturation of an atom, the counit reverses the process. It takes the saturation of an atom and gives back the substitution instance given by the identity, that is, the atom itself.

The next construction defines a morphism $\overline{d}\colon \mathcal{U}\big(\mathcal{C}(\mathcal{S})(At)\big) \to \mathcal{C}(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f})(\mathcal{U}At)$ where $\mathcal{C}(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f})\colon \mathbf{Set}^{|\mathbf{L}_{\mathbf{\Sigma}}^{op}|} \to \mathbf{Set}^{|\mathbf{L}_{\mathbf{\Sigma}}^{op}|}$ is the cofree comonad on $\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}$, obtained through a terminal sequence analogously to Construction 3. The idea is that $\overline{d}$ acts on saturated trees as the depthwise application of $\epsilon_{\mathcal{U}At}$.

Construction 5 *For α an ordinal, let us note by Y_α the objects occurring in the construction of $\mathcal{C}(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f})(\mathcal{U}At)$ and with X_α the ones in the construction of $\mathcal{C}(\mathcal{S})(At)$, converging to $X_\gamma = \mathcal{C}(\mathcal{S})(At)$. We define a sequence $\{d_\alpha\colon \mathcal{U}(X_\alpha) \to Y_\alpha\}_{\alpha<\gamma}$ in $\mathbf{Set}^{|\mathbf{L}_{\mathbf{\Sigma}}^{op}|}$ as follows:*

$$d_\alpha := \begin{cases} id_{\mathcal{U}At} & \alpha = 0 \\ id_{\mathcal{U}At} \times \big(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}(d_\beta) \circ \epsilon_{\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}(X_\beta)}\big) & \alpha = \beta + 1. \end{cases}$$

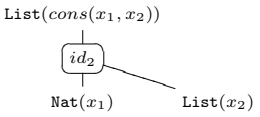
For $\alpha < \gamma$ a limit ordinal, $d_\alpha\colon \mathcal{U}(X_\alpha) \to Y_\alpha$ is provided by the limiting property of Y_α. This sequence induces a morphism $\overline{d}\colon \mathcal{U}\big(\mathcal{C}(\mathcal{S})(At)\big) \to \mathcal{C}(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f})(\mathcal{U}At)$. We refer to Appendix C for more details.

The next theorem shows that $\overline{d}$ is a translation from saturated to coinductive trees: given an atom $A \in At(n)$, it maps $[\![A]\!]_{p^\sharp}$ to the n-coinductive tree of A. The key intuition is that n-coinductive trees can be seen as saturated trees where the labeling of or-nodes has been restricted to the identity substitution id_n, represented as $\bullet$ (see Definition 2). The operation of pruning all or-nodes (and their descendants) in $[\![A]\!]_{p^\sharp}$ which are not labeled with id_n is precisely what is provided by Construction 5, in virtue of the definition of the counit ϵ given in (5).

Theorem 3 (Desaturation). *Let $[\![-]\!]_{p^\sharp}\colon At \to \mathcal{C}(\mathcal{S})(At)$ be defined for a logic program $\mathbb{P}$ according to Construction 4 and $\overline{d}\colon \mathcal{U}\big(\mathcal{C}(\mathcal{S})(At)\big) \to \mathcal{C}(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f})(\mathcal{U}At)$ be defined according to Construction 5. Then for all $n \in |\mathbf{L}_{\mathbf{\Sigma}}^{op}|$ and $A \in \mathcal{U}At(n)$, the n-coinductive tree of A in $\mathbb{P}$ is $\big(\overline{d} \circ \mathcal{U}([\![-]\!]_{p^\sharp})\big)(n)(A)$.*

Theorem 3 also provides an alternative formalization for the coinductive tree semantics [25], given by composition of the saturated semantics with desaturation. In fact it represents a different approach to the non-compositionality problem: instead of relaxing naturality to lax naturality, we simply forget about all the arrows of the index category $\mathbf{L}_{\mathbf{\Sigma}}^{op}$, shifting the framework from $\mathbf{Set}^{\mathbf{L}_{\mathbf{\Sigma}}^{op}}$ to $\mathbf{Set}^{|\mathbf{L}_{\mathbf{\Sigma}}^{op}|}$. The substitutions on trees (that are essential, for instance, for the resolution algorithm given in [24, 26]) exist at the saturated level, i.e. in $\mathcal{C}(\mathcal{S})(At)$, and they are given precisely as the operator $\overline{\theta}$ described at the end of Section 3.

Example 6. The coinductive tree for $\mathtt{List}(cons(x_1, x_2))$ in `NatList` is depicted on the right. It is constructed by desaturating the tree $[\![\mathtt{List}(cons(x_1, x_2))]\!]_{p^\sharp}$ in Example 5, i.e., by pruning all the or-nodes (and their descendants) that are not labeled with id_2.



5 Soundness and Completeness

The notion of coinductive tree leads to a semantics that is sound and complete with respect to SLD-resolution [24, Th.4.8]. To this aim, a key role is played by *derivation subtrees* of a given coinductive tree.

Definition 4. *Let T be the n-coinductive tree for an atom A in a program $\mathbb{P}$. A subtree T' of T is a* derivation subtree *if it satisfies the following conditions:*

1. *the root of T' is the root of T;*
2. *if an and-node of T belongs to T', then just one of its children belongs to T';*
3. *if an or-node of T belongs to T', then all its children belong to T'.*

A refutation subtree *(called success subtree in [24]) is a finite derivation subtree with only or-nodes as leaves.*

In analogy with coinductive trees, we want to define a notion of subtree for saturated semantics. This requires care: saturated trees are associated with unification, which is more liberal than term-matching. In particular, similarly to and-or trees, they may represent unsound derivation strategies. However, in saturated trees *all* unifiers, and not just the most general ones, are taken into account. This gives enough flexibility to shape a sound notion of subtree, based on an implicit synchronization of the substitutions used in different branches.

Definition 5. *Let T be the saturated tree for an atom A in a program $\mathbb{P}$. A subtree T' of T is called a* synched derivation subtree *if it satisfies properties 1-3 of Definition 4 and the following condition:*

4. *all or-nodes of T' which are at the same depth are labeled with the same substitution.*

A synched refutation subtree *is a finite synched derivation subtree with only or-nodes as leaves. Its* answer *is the substitution $\theta_{2k+1} \circ \ldots \theta_3 \circ \theta_1$, where θ_i is the (unique) substitution labeling the or-nodes of depth i and $2k+1$ is its maximal depth.*

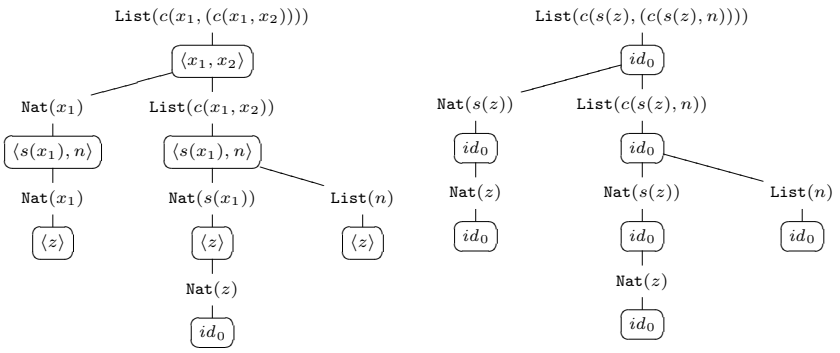


Fig. 1. Successful synched derivation subtrees for $\texttt{List}(cons(x_1, (cons(x_1, x_2))))$ (left) and $\texttt{List}(cons(succ(zero), (cons(succ(zero), nil))))$ (right) in `NatList`. The symbols *cons*, *nil*, *succ* and *zero* are abbreviated to c, n, s and z respectively.

The prefix "synched" emphasizes the restriction to and-parallelism which is encoded in Definition 5. Intuitively, we force all subgoals at the same depth to proceed with *the same* substitution. For instance, this rules out the unsound derivation of [24, Ex.5.2] (reported in Appendix A, Example 7).

Note that derivation subtrees can be seen as special instances of synched derivation subtrees where all the substitutions are forced to be identities.

Theorem 4 (Soundness and Completeness). *Let $\mathbb{P}$ be a logic program and $A \in At(n)$ an atom. The following are equivalent.*

1. *The saturated tree for A in $\mathbb{P}$ has a synched refutation subtree with answer θ.*
2. *There is some natural number m such that the m-coinductive tree for $A\theta$ in $\mathbb{P}$ has a refutation subtree.*
3. *There is an SLD-refutation for $\{A\}$ in $\mathbb{P}$ with computed answer τ such that there exists a substitution σ with $\sigma \circ \tau = \theta$.*

The statement $(2 \Leftrightarrow 3)$ is a rephrasing of [24, Th.4.8], while $(1 \Leftrightarrow 2)$ is proved in Appendix D by using compositionality and desaturation (Theorems 2 and 3).

Figure 1 provides an example of the argument for direction $(1 \Rightarrow 2)$. Note that the root of the rightmost tree is labeled with an atom of the form $A\theta$, where θ and A are respectively the answer and the label of the root of the leftmost tree. The key observation is that the rightmost tree is a refutation subtree of the 0-coinductive tree for $A\theta$ and can be obtained from the leftmost tree by a procedure involving the operator $\overline{\theta}$ discussed at the end of Section 3.

6 Conclusions

This work proposed a coalgebraic semantics for logic programming, extending the framework introduced in [23] for the case of ground logic programs. Our approach has been formulated in terms of coalgebrae on presheaves, whose nice

categorical properties made harmless to reuse the very same constructions as in the ground case. We stressed how the critical point of this generalization was to achieve compositionality, which we obtained by employing *saturation* techniques. Starting from the operational semantics p proposed in [25], we characterized its saturation $p^\sharp$ in terms of substitution mechanisms, showing that the latter corresponds to unification, whereas the former is associated with term-matching. The map $p^\sharp$ gave rise to the notion of *saturated tree*, as the model of computation represented in our semantics. We observed that coinductive trees, introduced in [25], can be seen as a desaturated version of saturated trees, and we compared the two notions with a translation. Eventually, we tailored a notion of subtree (of a saturated tree), called *synched derivation subtree*, representing a sound derivation of a goal in a program. This led to a result of soundness and completeness of our semantics with respect to SLD-resolution. A next step in this direction would be to investigate *infinite* computations and the semantics of coinductive logic programming [20]. These have been fruitfully explored within the approach based on coinductive trees [24, 26], and we expect the notion of synched derivation subtree to bring further insights on the question.

Another line of research concerns a deeper understanding of saturation. A drawback of saturated semantics is that one has to take into account *all* the arrows in the index category, which are usually infinitely many. This problem has been tackled in [7] for powerset-like functors, by employing *normalized coalgebrae*: one considers only a *minimal* set of arrows, as an "optimal" representation of all the arrows in the index category. It is not immediate to see how this approach can be generalized to arbitrary functors, as it would be in the formulation of saturation in terms of right Kan extension. We believe that the case of logic programming represents an ideal benchmark to investigate this question, since minimal arrows have a very intuitive characterization as most general unifiers.

References

1. J. Adámek and V. Koubek. On the greatest fixed point of a set functor. *Theor. Comput. Sci.*, 150:57–75, 1995.
2. J. Adámek and J. Rosický. *Locally Presentable and Accessible Categories*. Cambridge University Press, 1994.
3. G. Amato, J. Lipton, and R. McGrail. On the algebraic structure of declarative programming languages. *Theor. Comput. Sci.*, 410(46):4626–4671, 2009.
4. F. Bonchi. *Abstract Semantics by Observable Contexts*. PhD thesis, Department of Computer Science of Pisa, 2008.
5. F. Bonchi, M. G. Buscemi, V. Ciancia, and F. Gadducci. A presheaf environment for the explicit fusion calculus. *J. Autom. Reason.*, 49(2):161–183, 2012.
6. F. Bonchi, B. König, and U. Montanari. Saturated semantics for reactive systems. In *LICS*, pages 69–80. IEEE, 2006.
7. F. Bonchi and U. Montanari. Coalgebraic symbolic semantics. In *CALCO*, pages 173–190, 2009.
8. F. Bonchi and U. Montanari. Reactive systems, (semi-)saturated semantics and coalgebras on presheaves. *Theor. Comput. Sci.*, 410(41):4044–4066, 2009.
9. R. Bruni, U. Montanari, and F. Rossi. An interactive semantics of logic programming. *Theory and Practice of Logic Programming*, 1(6):647–690, 2001.
10. A. Corradini, M. Große-Rhode, and R. Heckel. A coalgebraic presentation of structured transition systems. *Theor. Comput. Sci.*, 260(1-2):27–55, 2001.
11. A. Corradini, R. Heckel, and U. Montanari. From sos specifications to structured coalgebras: How to make bisimulation a congruence. *ENTCS*, 19(0):118 – 141, 1999.

12. A. Corradini and U. Montanari. An algebraic semantics for structured transition systems and its application to logic programs. *Theor. Comput. Sci.*, 103(1):51 – 106, 1992.
13. C. Dwork, P. C. Kanellakis, and J. C. Mitchell. On the sequential nature of unification. *The Journal of Logic Programming*, 1(1):35 – 50, 1984.
14. M. P. Fiore, E. Moggi, and D. Sangiorgi. A fully abstract model for the π-calculus. *Inf. Comput.*, 179(1):76–117, 2002.
15. M. P. Fiore and S. Staton. Comparing operational models of name-passing process calculi. *Inf. Comput.*, 204(4):524–560, 2006.
16. M. P. Fiore and S. Staton. A congruence rule format for name-passing process calculi from mathematical structural operational semantics. In *LICS*, pages 49–58. IEEE, 2006.
17. M. P. Fiore and D. Turi. Semantics of name and value passing. In *LICS*, pages 93–104. IEEE, 2001.
18. N. Ghani, K. Yemane, and B. Victor. Relationally staged computations in calculi of mobile processes. volume 106, pages 105–120, 2004.
19. J. A. Goguen. What is unification? - a categorical view of substitution, equation and solution. In *Resolution of Equations in Algebraic Structures, Volume 1: Algebraic Techniques*, pages 217–261. Academic, 1989.
20. G. Gupta, A. Bansal, R. Min, L. Simon, and A. Mallya. Coinductive logic programming and its applications. In *ICLP*, pages 27–44, 2007.
21. G. Gupta and V. S. Costa. Optimal implementation of and-or parallel prolog. In *PARLE*, pages 71–92, 1994.
22. Y. Kinoshita and A. J. Power. A fibrational semantics for logic programs. In *Proceedings of the 5th International Workshop on Extensions of Logic Programming*, ELP '96, pages 177–191, London, UK, UK, 1996. Springer-Verlag.
23. E. Komendantskaya, G. McCusker, and J. Power. Coalgebraic semantics for parallel derivation strategies in logic programming. In *AMAST*, pages 111–127, 2010.
24. E. Komendantskaya and J. Power. Coalgebraic derivations in logic programming. In *CSL*, pages 352–366, 2011.
25. E. Komendantskaya and J. Power. Coalgebraic semantics for derivations in logic programming. In *CALCO*, pages 268–282, 2011.
26. E. Komendantskaya, J. Power, and M. Schmidt. Coalgebraic logic programming: from semantics to implementation. *Submitted to the Journal of Logic and Computation*, 2012.
27. R. Kowalski. *Logic for problem-solving*. North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands, 1986.
28. J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2nd edition, 1993.
29. S. Mac Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer, 2nd edition, Sept. 1998.
30. S. MacLane and I. Moerdijk. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*. Springer, corrected edition, May 1992.
31. Z. Majkic. Coalgebraic semantics for logic programs. In *Proceedings of the 18th Workshop W(C)LP on Constraint Logic Programming*, pages 76–87, 2004.
32. M. Miculan. A categorical model of the fusion calculus. *ENTCS*, 218:275–293, 2008.
33. M. Miculan and K. Yemane. A unifying model of variables and names. In *FOSSACS*, volume 3441, pages 170–186, 2005.
34. U. Montanari and M. Sammartino. A network-conscious pi-calculus and its coalgebraic semantics. *Submitted to TCS (Festschrift for Glynn Winskel)*.
35. U. Montanari and V. Sassone. Dynamic congruence vs. progressing bisimulation for ccs. *FI*, 16(1):171–199, 1992.
36. D. Sangiorgi. A theory of bisimulation for the pi-calculus. *Acta Inf.*, 33(1):69–97, 1996.
37. I. Stark. A fully abstract domain model for the π-calculus. In *LICS*, pages 36–42. IEEE, 1996.
38. S. Staton. Relating coalgebraic notions of bisimulation. In *CALCO*, pages 191–205, 2009.
39. J. Worrell. Terminal sequences for accessible endofunctors. *ENTCS*, 19, 1999.

A More on SLD-Resolution and And-Or Trees

In this appendix we provide a more detailed description of SLD-resolution and the problem of unsoundness in and-or trees.

As mentioned in Section 2, SLD-resolution is an algorithm to check whether a goal G (which we represent as a finite set of atoms) is *refutable* in a program $\mathbb{P}$. A run of the algorithm on inputs G and $\mathbb{P}$ gives rise to an *SLD-derivation*, whose steps of computation can be sketched as follows. At the initial step 0, a set of atoms G_0 (the current goal) is initialized as G. At each step i, an atom A_i is selected in the current goal G_i and one checks whether A_i is unifiable with the head of some clause of the program. If not, the computation terminates with a failure. Otherwise, one such clause $C_i \;=\; H \leftarrow B_1, \ldots, B_k$ is selected: by a classical result, since A_i and H unify, they have also a most general unifier $\langle \sigma_i, \tau_i \rangle$. The goal G_{i+1} for the step $i+1$ is given as

$$\{B_1, \ldots, B_k\}\tau_i \;\cup\; (G_i \setminus \{A_i\})\sigma_i.$$

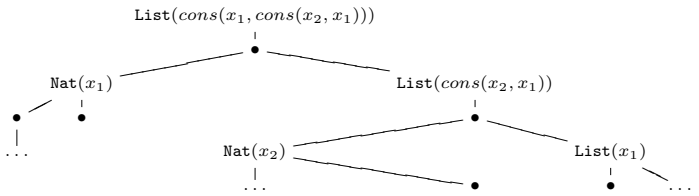
Such a computation is called an *SLD-refutation* if it terminates in a finite number (say n) of steps with $G_n = \emptyset$. In this case one calls *computed answer* the substitution given by composing the first projections $\sigma_n, \ldots, \sigma_0$ of the most general unifiers associated with each step of the computation. The goal G is *refutable* in $\mathbb{P}$ if an SLD-refutation for G in $\mathbb{P}$ exists[2].

As reported in Section 2, implementations of SLD-resolution exploiting *and-or parallelism* have a standard representation of computations as *and-or trees* (Definition 1). Unfortunately, in presence of variables these trees are not guaranteed to represent sound derivations. The problem lies in the interplay between variable dependencies and unification, which makes SLD-derivations for logic programs inherently *sequential* processes [13].

Example 7. Let `NatList` be the same program of Example 2 and consider the atom $A = \texttt{List}(cons(x_1, cons(x_2, x_1)))$. It is intuitively clear that there is no substitution of variables making A represent a list of natural numbers: we should replace x_1 with a "number" (for instance *zero*) in its first occurrence and with a "list" (for instance *nil*) in its second occurrence. Consequently, there is no SLD-refutation for $\{A\}$ in `NatList`. However, consider the and-or tree of A in

[2] In any derivation of G in $\mathbb{P}$, the standard convention is that the variables occurring in the clause C_i considered at step i do not appear in goals $G_{i-1}, \ldots, G_0$. This guarantees that the computed answer is a well-defined substitution and may require a dynamic (i.e. at step i) renaming of variables appearing in C_i. The associated procedure is called *standardizing the variables apart* and we assume it throughout the paper without explicit mention. It also justifies our definition (Section 2) of the most general unifier as pushout of two substitutions *with different target*, whereas it is also modeled in the literature as the coequalizer of two substitutions *with the same target*, see for instance [19]. The different target corresponds to the two substitutions depending on disjoint sets of variables.

NatList, for which we provide a partial representation as follows.



The above tree seems to yield an SLD-refutation: $\texttt{List}(cons(x_1, cons(x_2, x_1)))$ is refuted by proving $\texttt{Nat}(x_1)$ and $\texttt{List}(cons(x_2, x_1))$. However, the associated computed answer would be ill-defined, as it is given by substituting x_2 with $zero$ and x_1 both with $zero$ and with nil (the computed answer of $\texttt{Nat}(x_1)$ maps x_1 to $zero$ and the computed answer of $\texttt{List}(cons(x_2, x_1))$ maps x_1 to nil).

In Section 2 we recall *coinductive trees* [25]: a variant of and-or trees where unification is restricted to the case of term-matching. This constraint is sufficient to guarantee that coinductive trees only represent sound derivations: the key intuition is that a term-matcher is a unifier that leaves untouched the current goal, meaning that the "previous history" of the derivation remains uncorrupted.

For instance, the coinductive tree for the atom A of Example 7 in NatList is the subtree of the represented and-or tree having the nodes labeled with $\texttt{Nat}(x_1)$, $\texttt{Nat}(x_2)$ and $\texttt{List}(x_1)$ as leaves. Despite of the restriction to term-matching, one can show correctness and completeness of coinductive tree semantics with respect to SLD-resolution, as reported in Theorem 4 and originally shown in [24].

B Convergence of a Terminal Sequence

This appendix is devoted to the following proposition, which is propaedeutic to Construction 3.

Proposition 1. *The terminal sequence for $At \times \mathcal{S}(-)$ converges to a terminal coalgebra.*

Proof. By [39, Th.7], it suffices to prove that $\mathcal{S}$ is an accessible mono-preserving functor. Since these properties are preserved by composition, we show them separately for each component of $\mathcal{S}$:

- Being adjoint functors between accessible categories, $\mathcal{K}$ and $\mathcal{U}$ are accessible themselves [2, Prop.2.23]. Moreover, they are both right adjoints: in particular, $\mathcal{U}$ is right adjoint to the left Kan extension functor along $\iota\colon |\mathbf{C}| \hookrightarrow \mathbf{C}$. It follows that both preserve limits, whence they preserve monos.
- The functors $\widehat{\mathcal{P}_c}\colon \mathbf{Set}^{|\mathbf{L}_\Sigma^{op}|} \to \mathbf{Set}^{|\mathbf{L}_\Sigma^{op}|}$ and $\widehat{\mathcal{P}_f}\colon \mathbf{Set}^{|\mathbf{L}_\Sigma^{op}|} \to \mathbf{Set}^{|\mathbf{L}_\Sigma^{op}|}$ are defined as liftings of $\mathcal{P}_c\colon \mathbf{Set} \to \mathbf{Set}$ and $\mathcal{P}_f\colon \mathbf{Set} \to \mathbf{Set}$. It is well-known that $\mathcal{P}_c$ and $\mathcal{P}_f$ are both mono-preserving accessible functors on $\mathbf{Set}$. It follows that $\widehat{\mathcal{P}_c}$ and $\widehat{\mathcal{P}_f}$ also have these properties, because (co)limits in presheaf categories are computed objectwise and monos are exactly the objectwise injective morphisms (as shown for instance in [30, Ch.6]).

C A Closer Look at Desaturation

In this appendix we spell out the details of the construction of $\overline{d}$, as presented in Section 4. For this purpose, first we state the construction of the cofree comonad on $\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}$ for later reference.

Construction 6 *The terminal sequence for* $\mathcal{U}At \times \widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}(-)\colon \mathbf{Set}^{|\mathbf{L}_\Sigma^{op}|} \to \mathbf{Set}^{|\mathbf{L}_\Sigma^{op}|}$ *consists of sequences of objects* Y_α *and arrows* $\lambda_\alpha\colon Y_{\alpha+1} \to Y_\alpha$*, defined by induction on* α *as follows.*

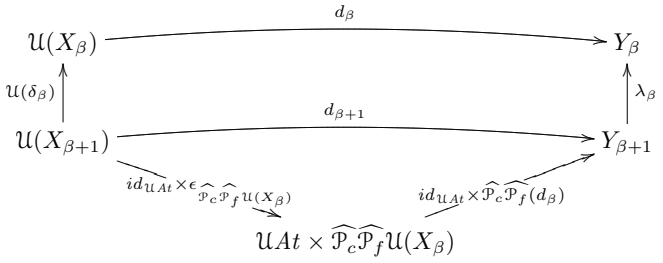
$$Y_\alpha := \begin{cases} \mathcal{U}At & \alpha = 0 \\ \mathcal{U}At \times \widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}(Y_\beta) & \alpha = \beta + 1 \end{cases} \quad \lambda_\alpha := \begin{cases} \pi_1 & \alpha = 0 \\ id_{\mathcal{U}At} \times \widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}(\lambda_\beta) & \alpha = \beta + 1 \end{cases}$$

For α *a limit ordinal,* Y_α *and* λ_α *are defined as expected. As stated in the proof of Proposition 1,* $\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}$ *is a mono-preserving accessible functors. Then by [39, Th.7] we know that the sequence given above converges to a limit* Y_χ *such that* $Y_\chi \cong Y_{\chi+1}$ *and* Y_χ *is the value of* $\mathcal{C}(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f})\colon \mathbf{Set}^{|\mathbf{L}_\Sigma^{op}|} \to \mathbf{Set}^{|\mathbf{L}_\Sigma^{op}|}$ *on* $\mathcal{U}At$*, where* $\mathcal{C}(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f})$ *is the cofree comonad on* $\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}$ *induced by the terminal sequence given above, analogously to Construction 1.*

Now, we can define the desaturation map $\overline{d}\colon \mathcal{U}\big(\mathcal{C}(\mathcal{S})(At)\big) \to \mathcal{C}(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f})(\mathcal{U}At)$, filling in the details of Construction 5.

Construction 7 *Consider the image of the terminal sequence converging to* $\mathcal{C}(\mathcal{S})(At) = X_\gamma$ *(Construction 3) under the forgetful functor* $\mathcal{U}\colon \mathbf{Set}^{\mathbf{L}_\Sigma^{op}} \to \mathbf{Set}^{|\mathbf{L}_\Sigma^{op}|}$*. We define a sequence of natural transformations* $\{d_\alpha\colon \mathcal{U}(X_\alpha) \to Y_\alpha\}_{\alpha<\gamma}$ *as follows*[3]*:*

$$d_\alpha := \begin{cases} id_{\mathcal{U}At} & \alpha = 0 \\ id_{\mathcal{U}At} \times \big(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}(d_\beta) \circ \epsilon_{\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}(X_\beta)}\big) & \alpha = \beta + 1. \end{cases}$$



[3] Concerning the successor case, observe that $id_{\mathcal{U}At} \times \big(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}(d_\beta) \circ \epsilon_{\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}(X_\beta)}\big)$ is in fact an arrow from $\mathcal{U}At \times \mathcal{U}\mathcal{K}\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}(X_\beta)$ to $Y_{\beta+1}$. However, the former is isomorphic to $\mathcal{U}(X_{\beta+1}) = \mathcal{U}\big(At \times \mathcal{K}\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}(X_\beta)\big)$, because $\mathcal{U}$ is a right adjoint (as observed in Proposition 1) and thence it commutes with products.

For $\alpha < \gamma$ a limit ordinal, a natural transformation $d_\alpha \colon \mathcal{U}(X_\alpha) \to Y_\alpha$ is provided by the limiting property of Y_α. In order to show that the limit case is well defined, observe that, for every $\beta < \alpha$, the above square commutes, that is, $\lambda_\beta \circ d_{\beta+1} = d_\beta \circ \mathcal{U}(\delta_\beta)$. This can be easily checked by ordinal induction, using the fact that $\epsilon_{\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}(X_\beta)}$ is a natural transformation, for each such β.

We now turn to the definition of a natural transformation $\overline{d} \colon \mathcal{U}\big(\mathcal{C}(\mathcal{S})(At)\big) \to \mathcal{C}(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f})(\mathcal{U}At)$. If $\chi \leq \gamma$, then this is provided by $d_\chi \colon \mathcal{U}(X_\chi) \to Y_\chi$ together with the limiting property of $\mathcal{U}(X_\gamma)$ on $\mathcal{U}(X_\chi)$. In case $\gamma < \chi$, observe that, since X_γ is isomorphic to $X_{\gamma+1}$, then X_γ is isomorphic to X_ζ for all $\zeta > \gamma$, and in particular $X_\gamma \cong X_\chi$. Then we can suitably extend the sequence to have a natural transformation $d_\chi \colon \mathcal{U}(X_\chi) \to Y_\chi$. The morphism $\overline{d}$ is given as the composition of d_χ with the isomorphism between $\mathcal{U}(X_\gamma)$ and $\mathcal{U}(X_\chi)$.

D Proof of Theorem 4

In this appendix we provide more details on the proof of Theorem 4. To this aim, a key role is played by the following lemma that generalizes the example of Figure 1.

Lemma 1. *Let $\mathbb{P}$ be a logic program and $A \in At(n)$ an atom. If $[\![A]\!]_{p^\sharp}$ has a synched refutation subtree with answer $\theta \colon n \to m$, then $[\![A]\!]_{p^\sharp}\overline{\theta}$ has a synched refutation subtree whose or-nodes are all labeled with id_m.*

Proof. First, we observe that the two following properties hold for all the and-nodes of $[\![A]\!]_{p^\sharp}$.

(†) Let θ, θ' be two arrows in $\mathbf{L}_{\mathbf{\Sigma}}^{op}$ such that $\theta' \circ \theta$ is defined. If an and-node s has a child t such that (a) the label of t is θ and (b) t has children labeled with $B_1, \ldots, B_n$, then s has also another child t' such that (a) the label of t' is $\theta' \circ \theta$ and (b) t' has children labeled with $B_1\theta', \ldots, B_n\theta'$.

(‡) Let $\theta, \theta', \sigma, \sigma'$ be four arrows in $\mathbf{L}_{\mathbf{\Sigma}}^{op}$ such that $\sigma \circ \theta = \sigma' \circ \theta'$. If an and-node labeled with A' has a child t such that (a) the label of t is θ and (b) t has children labeled with $B_1, \ldots, B_n$, then each node labeled with $A'\theta'$ has a child t' such that (a) the label of t' is σ' and (b) t' has children labeled with $B_1\sigma, \ldots, B_n\sigma$.

Assume that $[\![A]\!]_{p^\sharp}$ has a synched refutation subtree T whose or-nodes are labeled with $\theta_1, \theta_3, \ldots, \theta_{2k+1}$ (where θ_i is the substitution labeling the or-nodes of depth i). We prove that $[\![A]\!]_{p^\sharp}$ has another synched refutation subtree T' whose first or-node is labeled with $\theta = \theta_{2k+1} \circ \theta_{2k-1} \circ \cdots \circ \theta_1$ and all the other or-nodes are labeled with identities.

By assumption, the root r has a child (in T) that is labeled with θ_1. Assume that its children are labeled with $B_1^2 \ldots B_{n_2}^2$. By (†), r has another child t' (in $[\![A]\!]_{p^\sharp}$), that (a) is labeled with θ and (b) has children labeled with $B_1^2\sigma_3 \ldots B_n^2\sigma_3$ where $\sigma_3 = \theta_{k+1} \circ \theta_{k-1} \circ \cdots \circ \theta_3$. These children form depth 2 of T' (the root r and t form, respectively, depth 0 and 1).

We now build the other depths. For an even $i \leq 2k$, let σ_{i+1} denote $\theta_{2k+1} \circ \theta_{2k-1} \circ \cdots \circ \theta_{i+1}$ and let $B^i_1, \ldots, B^i_{n_i}$ be the labels of the and-nodes of T at depth i. The depth i of T' is given by and-nodes labeled with $B^i_1\sigma_{i+1}, \ldots, B^i_{n_i}\sigma_{i+1}$; the depth $i+1$ by or-nodes all labeled with id_m. It is easy to see that T' is a subtree of $[\![A]\!]_{p^\sharp}$: by assumption the nodes labeled with $B^i_1, \ldots, B^i_{n_i}$ have children in T all labeled with θ_{i+1}; since $\sigma_{i+3} \circ \theta_{i+1} = id_m \circ \sigma_{i+1}$, by property ($\ddagger$), the nodes labeled with $B^i_1\sigma_{i+1}, \ldots, B^i_{n_i}\sigma_{i+1}$ have children (in $[\![A]\!]_{p^\sharp}$) that (a) are labeled with id_m and (b) have children with labels $B^{i+2}_1\sigma_{i+3}, \ldots, B^{i+2}_{n_{i+2}}\sigma_{i+3}$.

Once we have built T', we can easily conclude. Recall that t' (the first or-node of T') is labeled with θ. Following the construction at the end of Section 3, the root of $[\![A]\!]_{p^\sharp}\overline{\theta}$ has a child that is labeled with id_m and that has the same children as t'. Therefore $[\![A]\!]_{p^\sharp}\overline{\theta}$ has a synched refutation subtree with answer id_m.

We are now ready to provide a proof of Theorem 4.

Proof (Proof of Theorem 4). The statement ($2 \Leftrightarrow 3$) is a rephrasing of [24, Th.4.8], whence we focus on proving ($1 \Leftrightarrow 2$), where m is always the target object of θ.

($1 \Rightarrow 2$). If $[\![A]\!]_{p^\sharp}$ has a synched refutation subtree with answer θ, then by Lemma 1, $[\![A]\!]_{p^\sharp}\overline{\theta}$ has a synched refutation subtree T whose or-nodes are all labeled with id_m. Compositionality (Theorem 2) guarantees that T is a synched refutation subtree of $[\![A\theta]\!]_{p^\sharp}$. Since all the or-nodes of T are labeled with id_m, T is preserved by desaturation. This means that T is a refutation subtree of $\overline{d}(\mathcal{U}([\![-]\!]_{p^\sharp}))(m)(A\theta)$ which, by Theorem 3, is the m-coinductive tree for $A\theta$ in $\mathbb{P}$.

($2 \Leftarrow 1$). If the m coinductive tree for $A\theta$ has a refutation subtree T then, by Theorem 3, this is also the case for $\overline{d}(\mathcal{U}([\![-]\!]_{p^\sharp}))(m)(A\theta)$. This means that T is a synched derivation subtree of $[\![A\theta]\!]_{p^\sharp}$ whose or-nodes are all labeled by id_m. By compositionality, T is also a subtree of $[\![A]\!]_{p^\sharp}\overline{\theta}$. Let t be the or-node at the first depth of T. By construction of the operator $\overline{\theta}$, the root of $[\![A]\!]_{p^\sharp}$ has a child t' labeled with θ having the same children as t. Therefore $[\![A]\!]_{p^\sharp}$ has a synched refutation subtree with answer θ.

On Context Bisimulation for Parameterized Higher-order Processes

Xian Xu *
Department of Computer Science and Technology
East China University of Science and Technology, Shanghai, China P.R. (200237)
xuxian@ecust.edu.cn, xuxian2004@gmail.com

This paper studies context bisimulation for higher-order processes, in the presence of parameterization (viz. abstraction). We show that the extension of higher-order processes with process parameterization retains the characterization of context bisimulation by a much simpler form of bisimulation called normal bisimulation (viz. they are coincident), in which universal quantifiers are eliminated; whereas it is not the same with name parameterization. These results clarify further the bisimulation theory of higher-order processes, and also shed light on the essential distinction between the two kinds of parameterization.

Keywords: Context Bisimulation, Normal Bisimulation, Higher-order, Processes

1 Introduction

Higher-order processes differ from first-order (name-passing) ones in that they can transmit themselves (i.e. integral programs) in communication. This mechanism of process-passing provides an alternative yet essentially distinct way from name-passing to achieve mobility. That distinction lies in several aspects, among which the behavioral theory is of pivotal importance. As a basis of behavioral theory, bisimulation theory has been studied since the very early work on higher-order processes. The most well-know (and probably standard) bisimulation is context bisimulation [7]. It was proposed to improve on the previous forms of bisimulation, including (applicative) higher-order bisimulation [14] [15], by considering the sent process and residual process at the same time. It then continued to draw attention [9] [16] [2] [1] [5] [12].

Related work and motivation

Context bisimulation, in its original form, is not so convenient in that, for example, it involves universal quantifiers when comparing output (as well as input), i.e. in the output clause of the definition of context bisimulation one has to check the matching of the output action with respect to all possible contexts. Here matching means the output action of one process can be simulated with an output by the other and the resulting derivatives are bisimilar again. That is a particularly heavy burden. This situation is improved by the so-called normal bisimulation, which characterizes context bisimulation (i.e. coincident with it) but requires only the checking with some special context (see an example below) [7] [9] [1].

A common methodology for establishing such a characterization in [7] [9] [1] consists of two main ingredients: (1) Showing a factorization theorem using triggers. The factorization theorem provides a mechanism of relocating a (sub)process to a new place, and setting up a pointer to the new place for

*The author acknowledges the support by the National Nature Science Foundation of China (61202023,61261130589,61173048), and the ANR project 12IS02001 PACE.

M. Carbone, I. Lanese, A. Lluch-Lafuente, A. Sokolova (Eds.):
6th Interaction and Concurrency Experience (ICE 2013)
EPTCS 131, 2013, pp. 37–51, doi:10.4204/EPTCS.131.5

potential use. Such a pointer is represented by a trigger. (2) Showing the coincidence of context and normal bisimulation with the help of an intermediate bisimulation equivalence called triggered bisimulation, which is defined on a subclass of higher-order processes communicating only triggers. This design of normal bisimulation, particularly the special context used in it, is guided by the factorization theorem as described above.

We exemplify below the simplification of context bisimulation by normal bisimulation in basic higher-order processes [9], equipped with the basic operators including input prefix ($a(X).P$ in which X is a bound variable), output prefix ($\overline{a}A.P$), parallel composition ($P \mid Q$) and restriction ($(c)P$ in which c is a bound name). Notice that $E[X]$ denotes a process with the free occurrence of X (i.e. not bound by an input prefix $a(X).P$), $E[A]$ denotes substituting A for all free occurrences of X in $E[X]$, output action $(\widetilde{c})\overline{a}A$ means sending over (port) name a a process A containing a set $\widetilde{c}$ of bound names, and the replication operation $!m.A$ is a derivable one in a higher-order setting [15]. We use $\approx$ for context bisimilarity (i.e. the equivalence induced by context bisimulation), and $\cong$ for normal bisimilarity (complete definitions are given in Definitions 1,2). First of all, the factorization theorem is stated as below (m is fresh, i.e. it does not occur in $E[A]$).

$$E[A] \approx (m)(E[Tr_m] \mid !m.A), \qquad \text{where the trigger } Tr_m \equiv \overline{m}.0$$

Intuitively, using factorization theorem one can extract from $E[A]$ the process A that might cause difference to its behavior. For instance,

$$A \mid Q \approx (m)((\overline{m}.0 \mid Q) \mid !m.A)$$

Now suppose P and Q are basic higher-order processes, and are bisimilar with respect to either bisimulation equivalence (i.e. context bisimulation or normal bisimulation). We focus on the output clauses of context bisimulation and normal bisimulation, since the input case is similar. Note $fn(E[X])$ returns the free names of $E[X]$ (i.e. not bound by restriction), and for simplicity we do not claim the existence of $\widetilde{d}, B, Q'$ below.

- The output clause of context bisimulation is
  If $P \xrightarrow{(\widetilde{c})\overline{a}A} P'$, then $Q \overset{(\widetilde{d})\overline{a}B}{\Longrightarrow} Q'$ s.t. for every $E[X]$ with $fn(E[X]) \cap (\widetilde{c} \cup \widetilde{d}) = \emptyset$, $(\widetilde{c})(P' \mid E[A]) \approx (\widetilde{d})(Q' \mid E[B])$.
- The output clause of normal bisimulation is
  If $P \xrightarrow{(\widetilde{c})\overline{a}A} P'$, then $Q \overset{(\widetilde{d})\overline{a}B}{\Longrightarrow} Q'$ s.t. for some fresh m, $(\widetilde{c})(P' \mid !m.A) \cong (\widetilde{d})(Q' \mid !m.B)$.

The intuition behind the simplification is that using the factorization theorem one can extract process A (respectively B) in $E[A]$ (respectively $E[B]$), so in context bisimulation one obtains the following, due to the congruence property of $\approx$,

$$\begin{array}{ccccc} R_1 \overset{def}{=} (\widetilde{c})(P' \mid (m)(E[Tr_m] \mid !m.A)) & \cdots\approx\cdots & (\widetilde{d})(Q' \mid (m)(E[Tr_m] \mid !m.B)) & \overset{def}{=} & R_2 \\ \vdots\approx & & \vdots\approx & & \\ (\widetilde{c})(P' \mid E[A]) & \cdots\approx\cdots & (\widetilde{d})(Q' \mid E[B]) & & \end{array}$$

where each dotted line connects two processes that are related by context bisimilarity ($\approx$). Then by the congruence property of $\approx$, one can eliminate the common part in processes R_1 and R_2 (i.e. $E[Tr_m]$ and restriction on m), and thus arrive at the form of the output clause in normal bisimulation.

The above has explained how the normal bisimulation works in basic higher-order processes [9] [1]. This paper concentrates on the extension of normal bisimulation, including its relevant technique, to higher-order processes with parameterization (also known as abstraction, akin to that in lambda-calculus; we postpone some examples until before stating the contribution), which still lacks full discussion.

Although parameterization is considered in [7], the definition of higher-order processes in [7] includes both name-passing and process-passing, and the coincidence between context bisimulation and normal bisimulation is studied in such a mixed language. So one does not know clearly whether the normal bisimulation can be extended to a pure higher-order setting with parameterization. To the best of our knowledge, it has not been discussed thoroughly whether such characterization, together with related technique such as trigges, is still applicable in a pure higher-order calculus extended with the two kinds of parameterization (on names and on processes) respectively. In general, it is still not clear whether the characterization result can be extended to an enriched higher-order setting. Yet the question whether the context bisimulation has such a convenient characterization of normal bisimulation in a parameterized setting is significant in broader research, as well as in improving the bisimulation theory itself (i.e. no universal checking is demanded). It would be interesting to know the answer to this question also because it is already known that parameterization can strictly enhance the expressiveness power of mere process-passing [3]; thus potential difference in the behavior theory of a more expressive calculus can be revealed. Relevantly in the study of expressiveness, a simpler yet equivalent form of context bisimulation would probably render much easier the proof of soundness (i.e. two processes are equivalent only if their encodings are equivalent), which requires the establishing of context bisimulation in a handy way. For instance, when comparing name-passing and processing [8] [17], the soundness demands that two name-passing processes are equivalent (e.g. early bisimilar) only if their translations to process-passing processes are context bisimilar. In this case, establishing normal bisimulation instead of context bisimulation would be much more convenient.

In this paper, we explicitly examine the normal characterization of context bisimulation in pure higher-order processes with parameterization; this time the language is not mixed, that is we study each parameterization separately, so that the essential difference can be revealed. We use Π to denote the basic (strict) higher-order pi-calculus, with the elementary operators (higher-order input and output, parallel composition, restriction) and without any first-order fragment. Notation Π^D_n (respectively Π^d_n) stands for the calculus extending Π with process parameterization (respectively name parameterization) of arity n ($n \in \mathbb{N}$ and $\mathbb{N}$ is the set of natural numbers); Π^D (respectively Π^d) is the union of Π^D_n (respectively Π^d_n). For example, the process P_1 below is a Π process (it means outputting X on name a and continuing as null); P_2 is a Π^D_1 process; P_3 is a Π^d_1 process. The parameterization operation is denoted by (leftmost) $\langle\cdot\rangle$. In the case of P_2, the X is a parameterized variable that can be instantiated with an application (rightmost $\langle\cdot\rangle$), for instance $P_2\langle A\rangle$ results in the process $\overline{a}A.0$. We will formally define the calculi in section 2.

$$P_1 \stackrel{def}{=} \overline{a}X.0 \qquad P_2 \stackrel{def}{=} \langle X\rangle\overline{a}X.0 \qquad P_3 \stackrel{def}{=} \langle x\rangle\overline{x}X.0$$

More often than not, process (respectively name) parameterization is understood as certain abstraction on processes (respectively names) in the literature. So parameterization and abstraction may be used interchangeably.

Contribution

The main contribution of this paper can be summarized as below.

- In the calculus Π^D_n (Π^D as well), we have normal bisimulation and it indeed characterizes context bisimulation. We prove the coincidence by re-exploiting the available method from [7] [9].

Moreover, a technical novelty here is that the proof is given in a more direct way, rather than first resorting to an intermediate bisimulation called triggered bisimulation by restricting to a sub-class of processes as described above.

- The approach of normal bisimulation (i.e. simple characterization of context bisimulation with normal bisimulation) cannot be extended to the calculus Π_n^d (Π^d either). That is, one cannot reuse the explicit technique of 'normal' bisimulation from [7] [9]. We provide a counterexample and some detailed discussion.

Since bisimulation theory often serves as the basis (or tool) for more advanced studies, these results provide a reference point for related work on higher-order calculi. They also shed light on the expressiveness of parameterization, because the contrast between the two results above reveals an essential separation between parameterization on process and names. Furthermore, such distinction also stresses the intrinsic complexity of name-handling (in a strictly higher-order setting without name-passing). So it will be interesting to look for some other kind of simpler characterization of context bisimulation in presence of name parameterization.

Organization

The paper is organized as follows. In section 2, we define the calculi Π, Π_n^D, Π_n^d, and the standard form of context bisimulation. Section 3 proves that there indeed exists the normal bisimulation that characterizes context bisimulation in Π_n^D ($n \in \mathbb{N}$). On the other hand, Π_n^d does not have such characterization in general, which we show in section 4. In section 5, we provide the conclusions and some further discussions.

2 Calculi

In this section, we define the calculus Π, and its variants Π_n^D, Π_n^d within which the context bisimulation is examined.

2.1 Calculus Π

The Π processes, denoted by uppercase letters ($A, B, E, F, P, Q, R, T...$) and their variant forms (e.g. T'), are defined by the following grammar (Π_{seg} is used to provide convenience for defining the variants of Π). Lowercase letters represent channel names, whereas X, Y, Z stand for process variables.

$$\begin{array}{l} T ::= \Pi_{seg} \\ \Pi_{seg} ::= 0 \;\Big|\; X \;\Big|\; u(X).T \;\Big|\; \overline{u}T'.T \;\Big|\; T \,|\, T' \;\Big|\; (c)T \end{array}$$

The operators are: prefix ($u(X).T, \overline{u}T'.T$), parallel composition ($T \,|\, T'$), restriction ($(c)T$) in which c is bound (or local); they have their standard meaning, and parallel composition has the least precedence. As usual some convenient notations are: a for $a(X).0$; $\overline{a}$ for $\overline{a}0.0$; $\overline{m}A$ for $\overline{m}A.0$; $\tau.P$ for $(a)(a.P \,|\, \overline{a})$; sometimes $\overline{a}[A].T$ for $\overline{a}A.T$ (for the sake of clarity); $\widetilde{\cdot}$ for a finite sequence of some items (e.g. names, processes), and $\widetilde{c}\widetilde{d}$ for the concatenation of $\widetilde{c}$ and $\widetilde{d}$. By standard definition, $fn(\widetilde{T})$, $bn(\widetilde{T})$, $n(\widetilde{T})$; $fv(\widetilde{T})$, $bv(\widetilde{T})$, $v(\widetilde{T})$ respectively denote free names, bound names, names; free variables, bound variables and variables in $\widetilde{T}$. Closed processes contain no free variables, and are studied by default. A fresh name or variable is one that does not occur in the processes under consideration. Name substitution $T\{\widetilde{n}/\widetilde{m}\}$

and higher-order substitution $T\{\widetilde{A}/\widetilde{X}\}$ are defined structurally in the standard way. $E[\widetilde{X}]$ denotes E with variables $\widetilde{X}$, and $E[\widetilde{A}]$ stands for $E\{\widetilde{A}/\widetilde{X}\}$. We work up-to α-conversion and always assume no capture. We use the following version of replication as a derived operator [15] [5]: $!\phi.P \stackrel{def}{=} (c)(Q_c \mid \overline{c}Q_c)$, $Q_c \stackrel{def}{=} c(X).(\phi.(X \mid P) \mid \overline{c}X)$, where ϕ is a prefix.

The operational semantics is given in Figure 1. Symmetric rules are omitted. The rules are mostly self-explanatory. For example, in higher-order input ($a(A)$), the received process A becomes part of the receiving environment through a substitution; in higher-order output ($(\widetilde{c})\overline{a}A$), the process A is sent with a set of local names for prospective use in further communication. In the first rule of the second row, we slightly abuse the notation, i.e. $fn(A)-\{\widetilde{c},a\}$ means the free names of A minus the names in $\widetilde{c}$ and a, which excludes the possibility of d=a. Symbols $\alpha,\beta,\lambda,...$ denote actions, whose subject (e.g. a in action $a(A)$) indicates the channel name on which it happens. Operations $fn(),bn(),n()$ can be similarly defined on actions. $\Longrightarrow$ is the reflexive transitive closure of the internal transition ($\stackrel{\tau}{\rightarrow}$), and $\stackrel{\lambda}{\Longrightarrow}$ is $\Longrightarrow\stackrel{\lambda}{\rightarrow}\Longrightarrow$. $\stackrel{\hat{\lambda}}{\Longrightarrow}$ is $\Longrightarrow$ when λ is τ and $\stackrel{\lambda}{\Longrightarrow}$ otherwise. $\stackrel{\tau}{\rightarrow}_k$ means k consecutive τ's. $P\Longrightarrow\cdot\mathscr{R}\,Q$ means $P\Longrightarrow Q'$ for some Q' and $Q'\,\mathscr{R}\,Q$ (i.e. $(Q',Q)\in\mathscr{R}$), where $\mathscr{R}$ is a binary relation. We say relation $\mathscr{R}$ is closed under (variable) substitution if $(E\{A/X\},F\{A/X\})\in\mathscr{R}$ for any A,X whenever $(E,F)\in\mathscr{R}$, in which E,F (possibly) have free occurrence of variable X. A process diverges if it can perform an infinite τ sequence.

$$\frac{}{a(X).T\stackrel{a(A)}{\longrightarrow}T\{A/X\}}\qquad\frac{}{\overline{a}A.T\stackrel{\overline{a}A}{\longrightarrow}T}\qquad\frac{T\stackrel{\lambda}{\rightarrow}T'}{(c)T\stackrel{\lambda}{\rightarrow}(c)T'}c\notin n(\lambda)\qquad\frac{T_1\stackrel{a(A)}{\longrightarrow}T_1',T_2\stackrel{(\widetilde{c})\overline{a}[A]}{\longrightarrow}T_2'}{T_1\mid T_2\stackrel{\tau}{\rightarrow}(\widetilde{c})(T_1'\mid T_2')}$$

$$\frac{T\stackrel{(\widetilde{c})\overline{a}[A]}{\longrightarrow}T'}{(d)T\stackrel{(d\widetilde{c})\overline{a}[A]}{\longrightarrow}T'}d\in fn(A)-\{\widetilde{c},a\}\qquad\frac{T\stackrel{\lambda}{\rightarrow}T'}{T\mid T_1\stackrel{\lambda}{\rightarrow}T'\mid T_1}bn(\lambda)\cap fn(T_1)=\emptyset$$

Figure 1: Semantics of Π

2.2 Calculi Π_n^D and Π_n^d

Parameterization extends Π with the syntax and semantics in Figure 2. $\langle U_1,U_2,...,U_n\rangle T$ is a parameterization where $U_1,U_2,...,U_n$ are the pairwise distinct formal parameters to be instantiated by the application $T\langle K_1,K_2,...,K_n\rangle$ where the parameters are instantiated by concrete objects $K_1,K_2,...,K_n$. Application binds tighter than prefixes and restriction. In the rule of Figure 2, $|\widetilde{U}|=|\widetilde{K}|$ requires the sequence of parameters and the sequence of instantiating objects should be equally sized. It also expresses that the parameterized process can do an action only after the application happens. Calculus Π_n^D, which has process parameterization (or higher-order abstraction), is defined by setting $\widetilde{U},\widetilde{K}$ to be $\widetilde{X},\widetilde{T'}$ respectively. Calculus Π_n^d, which has name parameterization (or first-order abstraction), is defined by setting $\widetilde{U},\widetilde{K}$ to be $\widetilde{x},\widetilde{u}$ respectively. For convenience, names (ranged over by u,v,w) are divided into two disjoint subsets: name constants (ranged over by $a,b,c,...,m,n$); name variables (ranged over by x,y,z). Process

$$T ::= \Pi_{seg} \;\Big|\; \langle U_1,U_2,...,U_n\rangle T \;\Big|\; T\langle K_1,K_2,...,K_n\rangle$$

$$\frac{T\{\widetilde{K}/\widetilde{U}\}\stackrel{\lambda}{\longrightarrow}T'}{F\langle\widetilde{K}\rangle\stackrel{\lambda}{\longrightarrow}T'}\quad\text{if } F\stackrel{def}{=}\langle\widetilde{U}\rangle T\;\;(|\widetilde{U}|=|\widetilde{K}|=n)$$

Figure 2: Π with parameterization

expressions (or terms) of the form $\langle\widetilde{X}\rangle P$ or $\langle\widetilde{x}\rangle P$, in which $\widetilde{X}$ and $\widetilde{x}$ are not empty, are *parameterized processes*. Terms without outmost parameterization are *non-parameterized processes*, or simply processes.

We mainly focus on (closed) processes. Only free variables can be effectively parameterized (i.e. it does not make sense to parameterize a bound variable); they become *bound* after parameterization. In the syntax, redundant parameterizations, for example $\langle X_1, X_2\rangle P$ in which $X_2 \notin fv(P)$, are allowed. A Π^D_n ($n \geq 1$) process is therefore definable in Π^D_{n+1} (similar for Π^d_n). In the case of $\langle \widetilde{X}\rangle P$, it can be coded up by setting an additional fresh dummy variable Y (of no use) to obtain $\langle \widetilde{X}, Y\rangle P$. Sometimes related notations are slightly abused if no confusion is caused.

Type systems for the processes of Π^D_n and Π^d_n can be routinely defined in a similar way to that in [7]. We do not present the type system and always assume type consistency, since such a type system is not important for the discussion in this paper. Without loss of generality, we stipulate that the processes of Π^D_n and Π^d_n are strictly abstraction-passing, i.e. all the transmitted objects are parameterized processes; accordingly, it is assumed that all the process variables have the type of abstractions, so an occurrence of a variable X typically takes the form $X\langle T'\rangle$ in some context. This can be justified by two facts: firstly a non-parameterized process can be treated as a special case of parameterization; secondly to the aim of this paper, the characterization of context bisimulation in the case of non-parameterized processes has been examined in depth by Sangiorgi [9].

$$
\begin{array}{l}
T \equiv T', \text{ if they are } \alpha\text{-convertible to each other,} \\
\qquad i.e.,\ a(X).T \equiv a(Z).T\{Z/X\}, (c)T \equiv (d)T\{d/c\} \\
T \,|\, 0 \equiv T,\ T \,|\, (T' \,|\, T'') \equiv (T \,|\, T') \,|\, T'',\ T \,|\, T' \equiv T' \,|\, T \\
(c)(d)T \equiv (d)(c)T,\ (c)\lambda.T \equiv 0 \text{ whenever the subject of } \lambda \text{ is } c \\
(c)(T \,|\, T') \equiv (c)T \,|\, T' \text{ whenever } c \notin fn(T') \\
(\langle \widetilde{U}\rangle T)\langle \widetilde{K}\rangle \equiv T\{\widetilde{K}/\widetilde{U}\}, |\widetilde{U}| = |\widetilde{K}| = n
\end{array}
$$

Figure 3: Structural congruence

The semantics of parameterization renders it somewhat natural to deem application as some (extra) rule of structural congruence, denoted by $\equiv$, which is defined in Figure 3 in a standard way [6] [9]. In addition to the standard algebraic laws (concerning parallel composition and restriction), the last rule of application is included. So the rule below can used in place of that in Figure 2.

$$\frac{T \equiv T_1, T_1 \xrightarrow{\alpha} T_2, T_2 \equiv T'}{T \xrightarrow{\alpha} T'}$$

2.3 Context Bisimulation

The bisimulation equivalence of higher-order processes we intend to examine is the context bisimulation. The form of its definition is the same for the calculi defined above.

Definition 1 (Context bisimulation)**.** A symmetric binary relation $\mathscr{R}$ on closed processes is a context bisimulation, if whenever $P \mathscr{R} Q$ the following properties hold:

1. If $P \xrightarrow{\alpha} P'$ and α is τ or $a(A)$, then $Q \stackrel{\hat{\alpha}}{\Longrightarrow} Q'$ for some Q' and $P' \mathscr{R} Q'$;
2. If $P \xrightarrow{(\widetilde{c})\overline{a}A} P'$ then $Q \stackrel{(\widetilde{d})\overline{a}B}{\Longrightarrow} Q'$ for some $\widetilde{d}, B, Q'$, and for every process $E[X]$ s.t. $\{\widetilde{c}, \widetilde{d}\} \cap fn(E) = \emptyset$ it holds that $(\widetilde{c})(E[A] \,|\, P') \mathscr{R} (\widetilde{d})(E[B] \,|\, Q')$.

Process P is context bisimilar to Q, written $P \approx, Q$, if $P \mathscr{R} Q$ for some context bisimulation $\mathscr{R}$. Relation $\approx$ is called context bisimilarity.

Context bisimulation can be extended to general (open) processes in a standard way, i.e. suppose $fv(T,T') \subseteq \widetilde{X}$, then $T \approx T'$ if $T\{\widetilde{A}/\widetilde{X}\} \approx T'\{\widetilde{A}/\widetilde{X}\}$ for all closed $\widetilde{A}$. Similarly the extension to parameterized processes is: $\langle\widetilde{X}\rangle T \approx \langle\widetilde{X}\rangle T'$ if $T\{\widetilde{A}/\widetilde{X}\} \approx T'\{\widetilde{A}/\widetilde{X}\}$ all closed $\widetilde{A}$.

Relation $\sim$ is the strong version of $\approx$. For clarity, notation $\approx_{\mathscr{L}}$ (resp. $\sim_{\mathscr{L}}$) indicates the context bisimilarity (resp. strong context bisimilarity) of calculus $\mathscr{L}$ (Π, Π_n^D, or Π_n^d); we simply use $\approx$ (resp. $\sim$) when it is clear from context. It is well-known that context bisimilarity is *an equivalence and a congruence* with respect to prefixing, parallel composition and restriction; see [7] [8] [9] [10]. Notice $E[X]$ is different from the well-known concept of contexts, which neglect name capture (i.e. the case a free name falls into the scope of some restriction of the same name). That is $E[X]$ is sensitive to name capture and should use α-conversion to avoid that. Otherwise, such two α-convertible processes like $(m)\overline{a}[m.0].\overline{m}.b$ and $(n)\overline{a}[n.0].\overline{n}.b$ would be distinguishable by context bisimilarity using $E[X] \equiv (m)X$ as the receiving environment (the latter can produce a visible action on b while the former does not), which is contradictory because α-convertibility shall entail context bisimilarity.

3 Normal bisimulation in Π_n^D

In this section, we show that in Π_n^D we have normal bisimulation that characterizes context bisimulation. For convenience, we focus on Π_1^D; the result can be readily extended to Π_n^D. We first define normal bisimulation, and then prove the coincidence theorem. Hereinafter $Tr_m \stackrel{def}{=} \langle Z\rangle\overline{m}Z$ denotes a trigger with (trigger name) m.

The form of normal bisimulation and the proof schema stem from the result in [7] [9] [1]. However, as mentioned, we go beyond those works in several respects. Firstly, the processes under inspection here are purely higher-order without name-passing, and capable of parameterization on processes only. Secondly, although the schema for the proof of the characterization of context bisimulation using normal bisimulation exploits those works, the technical details are not the same. More specifically, different from the approaches in [1], where the so-called index technique is essentially used to deal with actions, and in [7] [9], where an intermediate equivalence called triggered bisimulation is used and processes are first transformed into a subclass called triggered processes, we prove the coincidence of normal bisimulation and context bisimulation in a more *direct* and easier way.

3.1 Definition of normal bisimulation

Definition 2. A symmetric binary relation $\mathscr{R}$ on closed processes of Π_1^D is a normal bisimulation, if whenever $P\mathscr{R}Q$ the following properties hold:

1. If $P \xrightarrow{\tau} P'$, then $Q \Longrightarrow Q'$ for some Q' s.t. $P'\mathscr{R}Q'$;
2. If $P \xrightarrow{a(Tr_m)} P'$ and $Tr_m \equiv \langle Z\rangle\overline{m}Z$ (m is fresh), then $Q \stackrel{a(Tr_m)}{\Longrightarrow} Q'$ for some Q' s.t. $P'\mathscr{R}Q'$;
3. If $P \xrightarrow{(\widetilde{c})\overline{a}A} P'$ then $Q \stackrel{(\widetilde{d})\overline{a}B}{\Longrightarrow} Q'$ for some $\widetilde{d}, B, Q'$, and it holds that (m is fresh)

$$(\widetilde{c})(P' \mid !m(Z).A\langle Z\rangle)\ \mathscr{R}\ (\widetilde{d})(Q' \mid !m(Z).B\langle Z\rangle)$$

which can be rephrased as $(\widetilde{c})(P' \mid E'[A])\ \mathscr{R}\ (\widetilde{d})(Q' \mid E'[B])$ where the $E'[X] \stackrel{def}{=} !m(Z).X\langle Z\rangle$ holding A and B is special, in contrast to the general requirement in context bisimulation.

Process P is normal bisimilar to Q, written $P \cong, Q$, if $P\mathscr{R}Q$ for some normal bisimulation $\mathscr{R}$. Relation $\cong$ is called normal bisimilarity.

Remark The strong version of $\cong$ is denoted by $\simeq$. Notation $\cong_{\mathscr{L}}$ (resp. $\simeq_{\mathscr{L}}$) indicates the normal bisimilarity (resp. strong normal bisimilarity) of calculus $\mathscr{L}$, if any; we simply use $\cong$ (resp. $\simeq$) when there is no confusion. It can be shown in a standard way that normal bisimilarity is *an equivalence and a congruence*; see [7] [9] for a reference.

The rest of this section is mainly devoted to the (technical) proof of the following theorem.

Theorem 3. *In Π_1^D, normal bisimilarity coincides with context bisimilarity; that is $\cong = \approx$.*

3.2 Normal bisimulation characterizes context bisimulation

A key step in proving Theorem 3 is to prove the Factorization theorem (Theorem 6). First we need some preparation, i.e. the two lemmas below. Intuitively, these two lemmas state some distributive laws concerning the replication $!m(Z).A\langle Z\rangle$ (m is fresh) that are useful in the proof of the Factorization theorem.

Lemma 4. *Suppose $E[X], E_1[X], E_2[X]$ belong to Π_1^D, and let $m \notin fn(E, E_1, E_2, A)$.*

(1) If $m \notin fn(\alpha)$, then

$$(m)(\alpha.E[Tr_m] \mid !m(Z).A\langle Z\rangle) \approx \alpha.(m)(E[Tr_m] \mid !m(Z).A\langle Z\rangle)$$

(2) It holds for output prefix that

$$(m)(\overline{a}B_1.E_1[Tr_m] \mid !m(Z).A\langle Z\rangle) \approx (m)(\overline{a}B_2.E_1[Tr_m] \mid !m(Z).A\langle Z\rangle)$$

where $B_1 \equiv E_2[Tr_m]$, $B_2 \equiv (m)(E_2[Tr_m] \mid !m(Z).A\langle Z\rangle)$.

(3) It holds for parallel composition that

$$\begin{array}{rl} & (m)(E_1[Tr_m] \mid E_2[Tr_m] \mid !m(Z).A\langle Z\rangle) \\ \approx & (m)(E_1[Tr_m] \mid !m(Z).A\langle Z\rangle) \mid (m)(E_2[Tr_m] \mid !m(Z).A\langle Z\rangle) \end{array}$$

Proof. The proof is based on a standard argument on establishing bisimulations, where the scope of restriction concerning m is the critical part. The details are thus omitted. □

Lemma 5. *Suppose $E_1[X]$ belongs to Π_1^D, and let m be fresh. It holds for every A, B (of the type of abstraction) that*

$$B\langle (m)(E_1[Tr_m] \mid !m(Z).A\langle Z\rangle)\rangle \approx (m)(B\langle E_1[Tr_m]\rangle \mid !m(Z).A\langle Z\rangle)$$

Proof. Suppose $B \equiv \langle Y\rangle T$, it amounts to prove

$$\begin{array}{rl} & T\{T_1/Y\} \approx (m)(T\{T_2/Y\} \mid !m(Z).A\langle Z\rangle) \\ \text{where} & T_1 \equiv (m)(E_1[Tr_m] \mid !m(Z).A\langle Z\rangle) \text{ and } T_2 \equiv E_1[Tr_m] \end{array}$$

This can be done by a simple induction on the structure of T in a routine way. The details pertaining to establishing bisimulation during the induction would not raise major obstacle. □

Now we are ready to show the Factorization theorem. As mentioned, this theorem offers some method to relocate a subprocess, which might cause difference in behavior, and set up a reference to it with the help of a trigger, while maintaining the equivalence with respect to context bisimilarity. The proof of Theorem 6 is put in appendix A.

Theorem 6 (Factorization)**.** *Given a $E[X]$ of Π_1^D, let m be fresh and notice $Tr_m \equiv \langle Z\rangle\overline{m}Z$. It holds for every A (of the type of abstraction) that*

(1) if $E[Tr_m]$ is non-parameterized, i.e. not an abstraction, then

$$E[A] \approx (m)(E[Tr_m] \mid !m(Z).A\langle Z\rangle)$$

(2) else if $E[Tr_m] \equiv \langle Y_1\rangle \cdots \langle Y_k\rangle E'$ for some $k \geq 1$ and non-parameterized E', then

$$E[A] \approx \langle Y_1\rangle \cdots \langle Y_k\rangle((m)(E' \mid !m(Z).A\langle Z\rangle))$$

With the help of factorization theorem (Theorem 6), below we give the proof of Theorem 3.

Proof of Theorem 3. The fact that $\approx$ implies $\cong$ barely needs argument, because the former demands more and the latter is actually a special case of it. So we focus on the other direction. To achieve this, we show the relation $\mathscr{R} \stackrel{def}{=} \{(P,Q) \mid P \cong Q\}$ (i.e. normal bisimilarity $\cong$) is a context bisimulation up-to $\approx$ [11] [13] (the definition is standard and thus omitted), by using mainly the factorization theorem.

There are several cases to analyze in terms of the definition of context bisimulation. Notice we use weak transitions in the bisimulation, which is somewhat a standard variant of the corresponding bisimulation. Moreover we focus on the case when the first result (1) of theorem 6 applies, the case when the other applies can be handled in a similar (and easier) way.

- Internal action. This case is trivial, because the clauses in context bisimulation and normal bisimulation are the same.
- Input. If $P \stackrel{a(A)}{\Longrightarrow} P'$, then we want to show that

$$Q \stackrel{a(A)}{\Longrightarrow} Q' \text{ for some } Q' \tag{1}$$
$$\text{and } P' \approx \mathscr{R} \approx Q' \tag{2}$$

W.l.o.g. suppose $P' \equiv E[A]$ for some $E[X]$ (i.e. from $P \stackrel{a(X)}{\Longrightarrow}$ intuitively). So $P \stackrel{a(Tr_m)}{\Longrightarrow} E[Tr_m]$ for some fresh m. Since $P \cong Q$, we know

$$Q \stackrel{a(Tr_m)}{\Longrightarrow} F[Tr_m] \text{ for some } F$$

and

$$E[Tr_m] \cong F[Tr_m] \tag{3}$$

Thus

$$Q \stackrel{a(A)}{\Longrightarrow} F[A] \stackrel{def}{=} Q'$$

which fulfills (1). We know from (3) and the congruence properties of $\cong$ that

$$(m)(E[Tr_m] \mid !m(Z).A\langle Z\rangle) \cong (m)(F[Tr_m] \mid !m(Z).A\langle Z\rangle)$$

Now by factorization theorem (Theorem 6), we have

$$E[A] \approx (m)(E[Tr_m] \mid !m(Z).A\langle Z\rangle)\,\mathscr{R}\,(m)(F[Tr_m] \mid !m(Z).A\langle Z\rangle) \approx F[A]$$

which arrives at (2).

- Output. If $P \stackrel{(\widetilde{c})\overline{a}A}{\Longrightarrow} P'$, then we want to show that (notice $\{\widetilde{c},\widetilde{d}\} \cap fn(E) = \emptyset$)

$$Q \stackrel{(\widetilde{d})\overline{a}B}{\Longrightarrow} Q' \text{ for some } \widetilde{d}, B, Q' \tag{4}$$

$$\text{and for every } E[X],\ (\widetilde{c})(E[A] \mid P') \approx \mathscr{R} \approx (\widetilde{d})(E[B] \mid Q') \tag{5}$$

The argument can be conducted in a pretty similar way to that in the previous case for input; this time one attaches a process $E[Tr_m]$ and also uses the congruence properties of $\cong$.

Since $P \cong Q$, we have $Q \stackrel{(\widetilde{d})\overline{a}B}{\Longrightarrow} Q'$ which fulfills (4), and also

$$(\widetilde{c})(P' \mid !m(Z).A\langle Z\rangle) \cong (\widetilde{d})(Q' \mid !m(Z).B\langle Z\rangle) \tag{6}$$

We know from the congruence properties and (6) that

$$(m)((\widetilde{c})(P' \mid !m(Z).A\langle Z\rangle) \mid E[Tr_m]) \cong (m)((\widetilde{d})(Q' \mid !m(Z).B\langle Z\rangle) \mid E[Tr_m]) \tag{7}$$

By some simple structural adjustment and the factorization theorem (Theorem 6), we know the lhs (left hand side) and rhs (right hand side) of (7) are equivalent to $(\widetilde{c})(E[A] \mid P')$ and $(\widetilde{d})(E[B] \mid Q')$ respectively. So we have

$$(\widetilde{c})(E[A] \mid P') \approx (\text{lhs of (7)})\mathscr{R}(\text{rhs of (7)}) \approx (\widetilde{d})(E[B] \mid Q')$$

This is exactly what (5) says.

The proof is completed now. □

Remark.

- The success of the characterization of context bisimulation using normal bisimulation in Π_n^D can be attributed to the fact that Π_n^D processes are purely higher-order, i.e. no names can be passed but only (parameterized) processes (carrying names), and moreover no names can be parameterized. Thus one can delay the instantiation of a process parameterization by moving it elsewhere with the help of triggers.
- We mentioned in the introduction that in [1], the index technique is utilized to show that strong normal bisimulation characterizes strong context bisimulation, within a calculus capable of both name-passing and process-passing but without any parameterization. A critical point there is that indices can be used to precisely pinpoint the matching of actions from the processes when going through some intermediate transformation (e.g. factorization). We believe, by combining the technique in that paper with the approach in this section, one can further show that $\sim$ and $\simeq$ coincide in the calculus Π_n^D. The key point in this combination is that we can reuse the approach in this section for proving the coincidence in the strong case, except that we need to harness indices to mark and filter out the extra τ actions brought about by the operation of factorization, and thus the precise matching of strong actions can be established.

4 Normal bisimulation in Π_n^d

In this section, we examine context bisimulation in Π_n^d, particularly Π_1^d for simplicity. We show that, unlike that in Π_1^D, the technique of normal bisimulation [7] [9] cannot be extended to Π_1^d that is endowed

with name parameterization instead of process parameterization. To this end, we show the negative fact in two steps. Firstly, we discuss the possible form of normal bisimulation, toward giving some intuition. Then we provide a counterexample, to show that the expected form of normal bisimulation does not work out. Therefore finding a useful characterization of context bisimulation may amount to exploiting further the essence (e.g. expressiveness) of Π_n^d.

Possible form of normal bisimulation

Along the line of the very original idea of normal bisimulation [7] [9], we pretend having the 'normal bisimulation', among which the largest one is denoted by $\cong'$. This would lead to the argument below.

- A trigger now should be defined as $Tr_m \stackrel{def}{=} \langle z\rangle \overline{m}z$, because it is supposed to carry names rather than processes. This immediately brings about a critical problem. That is, name-passing is not allowed in Π_1^d, and we only admit abstraction-passing.
- In the definition of $\cong'$, the output clause should take the following form.

  If $P \xrightarrow{(\widetilde{c})\overline{a}A} P'$ then $Q \stackrel{(\widetilde{d})\overline{a}B}{\Longrightarrow} Q'$ for some $\widetilde{d}, B, Q'$, and it holds that (m is fresh)

  $$(\widetilde{c})(P' \,|\, E[A])\ \mathscr{R}\ (\widetilde{d})(Q' \,|\, E[B])$$

  where particularly it should be that $E[X] \stackrel{def}{=} !m(z).X\langle z\rangle$ in line with the trigger form. Again, the special environment $E[X]$ does not belong to the calculus Π_1^d.
- As for the input clause of $\cong'$, one meets with similar obstacle.

So intuitively, the failure of trigger technique, and thus the failure of the factorization theorem, deprives Π_1^d of the normal bisimulation. Furthermore, below we provide a counterexample to exhibit the deprival of normal bisimulation in Π_1^d.

A counterexample

We examine the following example: $W \stackrel{def}{=} A\langle d\rangle$, in which $A \stackrel{def}{=} \langle x\rangle \overline{x}$. Obviously W belongs to Π_1^d and is able to fire an action on d, i.e. $W \equiv \overline{d} \xrightarrow{\overline{d}} 0$. However if one tries to factorize out the subprocess A, some contradiction arises by the examining below.

1) One is supposed to replace A with a trigger of certain (general) form, say T, which has to be an abstraction *on a name* (to remain well-typed); so T must take the shape $\langle z\rangle T'$, and we have after a substitution

   $$W\{T/A\} \equiv T'\{d/z\} \tag{8}$$

   Now some sugar should be added, i.e. some context F is needed to contain $W\{T/A\}$ so that the resulting process bi-simulates W. Let us suppose F is of the form

   $$(\widetilde{c})([\cdot] \,|\, G[A])$$

   in which G should have A, in conformance to the rationale of factorization. Then generally, in terms of bisimulation, $F[T'\{d/z\}]$ should engage in some internal moves between $T'\{d/z\}$ and $G[A]$, so that in its current (different) position, A can do the same action $\overline{d}$ after an instantiation on x. It is the responsibility of T' to convey the particular information of d to G.

2) Holding back a little bit, a crux is that there is no way to transmit, with the help of any process (let alone a trigger), a concrete name for instantiation of the abstraction (e.g. in (8)) to the newly-assigned place for future access, because all the processes here are strictly higher-order.

Hence we conclude that the technique of normal bisimulation does not work for Π_n^d. This result also sheds light on the expressiveness of Π_n^d. It reflects that name parameterization offers more flexibility than process parameterization in expressiveness so that its bisimulation does not have a similar (simpler) characterization. We give more discussion in the conclusion below.

5 Conclusion

This paper provides some results on the characterization of context bisimulation in parameterized higher-order processes, and thus offers some tool as well as some insights for the bisimulation theory in higher-order paradigm. Firstly, we show that when extended with process parameterization, higher-order processes possess the characterization of context bisimulation by normal bisimulation. Secondly, we show that this technique of normal bisimulation, at least in its original form, cannot be extended to characterizing context bisimulation in presence of name parameterization. This separation result implies that the two kinds of parameterization have some essential difference. It will offer some potential reference for relevant research, for instance expressiveness studies.

There is some work worth further consideration.

- Finding some appropriate proof technique for the context bisimulation in Π_n^d. To some extent, Π_n^d is more useful than Π_n^D, which can strictly enhance the expressiveness of Π [3], because it consists of some name-handling primitive, though it is still a higher-order language (i.e. no name-passing). So in order to make Π_n^d more convenient when studying related topics, for example expressiveness and applications like modeling-verifying, there should be some proof technique for its canonical bisimulation equivalence, i.e. the context bisimulation. Since a standard normal characterization based on triggers is not available, one has to search for other ways to simplify the work on proving/checking whether two processes are context bisimilar. This may need some restriction on name abstraction or some other novel technique catering for certain motivation.

- Extension of the available characterization and related proof technique to more general higher-order models. For a long time, normal bisimulation and trigger technique has seen successful application and been deemed as a somewhat general method of dealing with higher-order processes (even applicable in some first-order process models). However, the result here implies that the normal approach, i.e. proof of context bisimulation up-to trigger and context, deserves more examination. Sometimes it would be better to study a general form of bisimulation in some general higher-order model. This is advantageous in that the usefulness of the available technique can be further tested, and also more essence of the behavior equivalence in higher-order processes can be hopefully revealed. A possible direction is to follow the idea in [12] where environmental bisimulation is proposed for higher-order models, and examine the proof technique of environmental bisimulation that is shown to coincide with canonical bisimulation in various higher-order models. In such a more general model, one may exploit some general proof technique (e.g. up-to certain normal feature) of different nature.

Acknowledgement The author thanks Davide Sangiorgi for many constructive comments, and Qiang Yin for much helpful discussion. He is also grateful to the comments and suggestions from the anonymous referees.

References

[1] Z. Cao (2006): *More on Bisimulations for Higher Order π-Calculus*. In L. Aceto & A. Ingólfsdóttir, editors: *Proceedings of FOSSACS2006*, *LNCS* 3921, pp. 63–78, doi:10.1007/11690634_5. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25-31, 2006.

[2] A. Jeffrey & J. Rathke (2005): *Contextual equivalence for higher-order pi-calculus revisited*. *Logical Methods in Computer Science* 1(1:4), doi:10.2168/LMCS-1(1:4)2005.

[3] I. Lanese, J. A. Pérez, D. Sangiorgi & A. Schmitt (2010): *On the Expressiveness of Polyadic and Synchronous Communication in Higher-Order Process Calculi*. In: *Proceedings of ICALP 2010*, LNCS, Springer Verlag, pp. 442–453, doi:10.1007/978-3-642-14162-1_37.

[4] I. Lanese, J. A. Pérez, D. Sangiorgi & A. Schmitt (2011): *On the Expressiveness and Decidability of Higher-Order Process Calculi*. *Information and Computation* 209(2), pp. 198–226, doi:10.1016/j.ic.2010.10.001.

[5] I. Lanese, J.A. Perez, D. Sangiorgi & A. Schmitt (2008): *On the Expressiveness and Decidability of Higher-Order Process Calculi*. In: *Proceedings of the 23rd Annual IEEE Symposium on Logic in Computer Science (LICS 2008)*, IEEE Computer Society, pp. 145–155, doi:10.1109/LICS.2008.8. Journal version in [4].

[6] R. Milner (1992): *Functions as Processes*. *Mathematical Structures in Computer Science* 2(2), pp. 119–141, doi:10.1017/S0960129500001407. Research Report 1154, INRIA, Sofia Antipolis, 1990.

[7] D. Sangiorgi (1992): *Expressing Mobility in Process Algebras: First-order and Higher-order Paradigms*. Phd thesis, University of Edinburgh.

[8] D. Sangiorgi (1992): *From π-Calculus to Higher-Order π-Calculus—and Back*. In: *Proceedings of TAPSOFT '93*, *LNCS* 668, Springer Verlag, pp. 151–166, doi:10.1007/3-540-56610-4_62.

[9] D. Sangiorgi (1996): *Bisimulation for Higher-order Process Calculi*. *Information and Computation* 131(2), pp. 141–178, doi:10.1006/inco.1996.0096. Preliminary version in proceedings of PROCOMET'94 (IFIP Working Conference on Programming Concepts, Methods and Calculi), pages 207-224, North Holland, 1994.

[10] D. Sangiorgi (1996): *Pi-calculus, Internal Mobility and Agent-Passing Calculi*. *Theoretical Computer Science* 167(2), doi:10.1016/0304-3975(96)00075-8. Extracts of parts of the material contained in this paper can be found in the Proceedings of TAPSOFT'95 and ICALP'95.

[11] D. Sangiorgi (1998): *On the Bisimulation Proof Method*. *Mathematical Structures in Computer Science* 8(6), pp. 447–479, doi:10.1017/S0960129598002527. An extended abstract in Proceedings of MFCS'95, LNCS 969, pp. 479-488, Springer Verlag.

[12] D. Sangiorgi, N. Kobayashi & E. Sumii (2011): *Environmental bisimulations for higher-order languages*. *ACM Transactions on Programming Languages and Systems* 33(1), p. 5, doi:10.1145/1889997.1890002.

[13] D. Sangiorgi & D. Walker (2001): *The Pi-calculus: a Theory of Mobile Processes*. Cambridge Universtity Press.

[14] B. Thomsen (1990): *Calculi for Higher Order Communicating Systems*. Phd thesis, Department of Computing, Imperial College.

[15] B. Thomsen (1993): *Plain CHOCS, a Second Generation Calculus for Higher-Order Processes*. *Acta Informatica* 30(1), pp. 1–59, doi:10.1007/BF01200262.

[16] J.-L. Vivas & M. Dam (1998): *From Higher-Order Pi-Calculus to Pi-Calculus in the Presence of Static Operators*. In: *Proceedings of the 9th International Conference on Concurrency Theory*, *LNCS* 1466, pp. 115–130, doi:10.1007/BFb0055619. Nice, France, September 8-11, 1998.

[17] Xian Xu (2012): *Distinguishing and Relating Higher-order and First-order Processes by Expressiveness*. *Acta Informatica* 49(7-8), pp. 445–484, doi:10.1007/s00236-012-0168-9.

Appendix

A Proof of Section 3

In this appendix, we give the proof of the factorization theorem in section 3.

Proof of Theorem 6: Factorization. The proof is by induction on the structure of E. The cases 1,2,3 are the base cases.

1. E is 0 or E is Y and $Y \neq X$. These cases are trivial.
2. E is X. Notice this time $E[Tr_m]$ is $Tr_m \equiv \langle Z\rangle \overline{m}Z \equiv \langle Y\rangle \overline{m}Y$ (up-to alpha-conversion), so the second statement of this theorem applies and the two terms of interest are

$$A \quad \text{and} \quad \langle Y\rangle((m)(\overline{m}Y \mid !m(Z).A\langle Z\rangle)$$

   Suppose $A \equiv \langle Z'\rangle F$, then it is easy to verify that for each B one has

$$F\langle B\rangle \approx (m)(\overline{m}B \mid !m(Z).F\{Z/Z'\}$$

   which completes this case.
3. E is $X\langle E_1\rangle$. This can be proven by showing the following relation $\mathscr{R}$ is a context bisimulation up-to $\sim$ (this technique is standard [9] [13] and we omit its definition here).

$$\mathscr{R} \stackrel{def}{=} \{(A\langle E_1\rangle, (m)(Tr_m\langle E_1\rangle \mid !m(Z).A\langle Z\rangle) \mid m \text{ is fresh } \} \cup \approx$$

   Let $(A\langle E_1\rangle, (m)(Tr_m\langle E_1\rangle \mid !m(Z).A\langle Z\rangle) \in \mathscr{R}$ and $A \equiv \langle Z'\rangle T$; so the pair is actually

$$(T\{E_1/Z'\}, (m)(\overline{m}E_1 \mid !m(Z).T\{Z/Z'\})$$

   There are mainly two cases to consider.

   - $(m)(\overline{m}E_1 \mid !m(Z).T\{Z/Z'\}) \xrightarrow{\alpha} T'$. Then α must be τ, and thus

$$T' \equiv (m)(T\{E_1/Z'\} \mid !m(Z).T\{Z/Z'\})$$

     By $T\{E_1/Z'\} \Longrightarrow T\{E_1/Z'\}$ (null transition), since m is fresh, it can be easily seen that,

$$T\{E_1/Z'\} \sim T\{E_1/Z'\}\,\mathscr{R}\,T\{E_1/Z'\} \sim T'$$

   - $T\{E_1/Z'\} \xrightarrow{\alpha} T_1$. Then this is simulated by

$$\begin{array}{ll} & (m)(\overline{m}E_1 \mid !m(Z).T\{Z/Z'\}) \\ \xrightarrow{\tau} & (m)(T\{E_1/Z'\} \mid !m(Z).T\{Z/Z'\}) \\ \xrightarrow{\alpha} & (m)(T_1 \mid !m(Z).T\{Z/Z'\}) \stackrel{def}{=} T_2 \end{array}$$

     So it holds in a straightforward way that

$$T_1 \sim T_1\,\mathscr{R}\,T_1 \sim T_2$$

The following cases 4,5,6,7,8,9 are cases involving the induction hypothesis (ind. hyp. for short). Notice that we will only consider the case when statement (1) in the theorem applies, and the case for (2) can be dealt with similarly.

4. E is $\langle Y\rangle E_1$. Then we have by ind. hyp.

$$\begin{array}{rl} & E[A] \equiv \langle Y\rangle E_1[A] \\ \approx & \langle Y\rangle((m)(E_1[Tr_m] \mid !m(Z).A\langle Z\rangle)) \end{array}$$

To conclude this case, we have to show

$$\langle Y\rangle((m)(E_1[Tr_m] \mid !m(Z).A\langle Z\rangle)) \approx \langle Y\rangle((m)(E_1[Tr_m] \mid !m(Z).A\langle Z\rangle))$$

This is immediate and the right-hand-side is exactly the second claim (2) of this theorem.

5. E is $\overline{a}E_2.E_1$. Then we have

$$\begin{array}{rll} & E[A] \equiv \overline{a}E_2[A].E_1[A] & \\ \approx & \overline{a}[(m)(E_2[Tr_m] \mid !m(Z).A\langle Z\rangle)].((m)(E_1[Tr_m] \mid !m(Z).A\langle Z\rangle)) & \text{(ind. hyp.)} \\ \approx & (m)(\overline{a}[(m)(E_2[Tr_m] \mid !m(Z).A\langle Z\rangle)].E_1[Tr_m] \mid !m(Z).A\langle Z\rangle) & \text{(Lemma 4(1))} \\ \approx & (m)(\overline{a}E_2[Tr_m].E_1[Tr_m] \mid !m(Z).A\langle Z\rangle) & \text{(Lemma 4(2))} \end{array}$$

The last equation gives exactly what we expect.

6. E is $a(Y).E_1$. This is similar (and easier) than the previous case.

7. E is $E_1 \mid E_2$. Then we have

$$\begin{array}{rll} & E[A] \equiv E_1[A] \mid E_2[A] & \\ \approx & (m)(E_1[Tr_m] \mid !m(Z).A\langle Z\rangle) \mid (m)(E_2[Tr_m] \mid !m(Z).A\langle Z\rangle) & \text{(ind. hyp.)} \\ \approx & (m)(E_1[Tr_m] \mid E_2[Tr_m] \mid !m(Z).A\langle Z\rangle) & \text{(Lemma 4(3))} \end{array}$$

The last equation completes this case.

8. E is $(c)E_1$. This is similar to the previous case.

9. E is $E_2\langle E_1\rangle$. This case can be reduced to the case E is $Y\langle E_1\rangle$ and $Y \neq X$, because if E_2 is not a variable (i.e. an abstraction) or is X, then it can be handled in a way that falls into one of the previous cases (up-to structural congruence). So we have

$$\begin{array}{rll} & E[A] & \\ \equiv & Y\langle E_1[A]\rangle & \\ \approx & Y\langle(m)(E[Tr_m] \mid !m(Z).A\langle Z\rangle)\rangle \stackrel{def}{=} T & \text{(ind. hyp.)} \end{array}$$

We want to show that $T \approx (m)(Y\langle E_1[Tr_m]\rangle \mid !m(Z).A\langle Z\rangle)$ i.e. for every B,

$$B\langle(m)(E_1[Tr_m] \mid !m(Z).A\langle Z\rangle)\rangle \approx (m)(B\langle E_1[Tr_m]\rangle \mid !m(Z).A\langle Z\rangle)$$

This is immediately from Lemma 5.

The proof is now completed. □

Preserving differential privacy under finite-precision semantics *

Ivan Gazeau, Dale Miller, and Catuscia Palamidessi
INRIA and LIX, Ecole Polytechnique

The approximation introduced by finite-precision representation of continuous data can induce arbitrarily large information leaks even when the computation using exact semantics is secure. Such leakage can thus undermine design efforts aimed at protecting sensitive information. We focus here on differential privacy, an approach to privacy that emerged from the area of statistical databases and is now widely applied also in other domains. In this approach, privacy is protected by the addition of noise to a true (private) value. To date, this approach to privacy has been proved correct only in the ideal case in which computations are made using an idealized, infinite-precision semantics. In this paper, we analyze the situation at the implementation level, where the semantics is necessarily finite-precision, i.e. the representation of real numbers and the operations on them, are rounded according to some level of precision. We show that in general there are violations of the differential privacy property, and we study the conditions under which we can still guarantee a limited (but, arguably, totally acceptable) variant of the property, under only a minor degradation of the privacy level. Finally, we illustrate our results on two cases of noise-generating distributions: the standard Laplacian mechanism commonly used in differential privacy, and a bivariate version of the Laplacian recently introduced in the setting of privacy-aware geolocation.

Keywords: Differential privacy, floating-point arithmetic, robustness to errors.

1 Introduction

It is well known that, due to the physical limitations of actual machines, in particular the finiteness of their memory, real numbers and their operations cannot be implemented with full precision. While for traditional computation getting an approximate result is not critical when a bound on the error is known, we argue that, in security applications, the approximation error can became a fingerprint potentially causing the disclosure of secrets.

Obviously, the standard techniques to measure the security breach do not apply, because an analysis of the system in the *ideal* (aka *exact*) semantics does not reveal the information leaks caused by the implementation. Consider, for instance, the following simple program

$$\text{if } f(h) > 0 \text{ then } \ell = 0 \text{ else } \ell = 1$$

where h is a high (i.e., confidential) variable and ℓ is a low (i.e., public) variable. Assume that h can take two values, v_1 and v_2, and that both $f(v_1)$ and $f(v_2)$ are strictly positive. Then, in the ideal semantics, the program is perfectly secure, i.e. it does not leak any information. However, in the implementation, it could be the case that the test succeeds in the case of v_1 but not in the case of v_2 because, for instance, the value of $f(v_2)$ is below the smallest representable positive number. Hence, we would have a total disclosure of the secret value.

*This work has been partially supported by the project ANR-09-BLAN-0345-02 CPP and by the INRIA Action d'Envergure CAPPRIS.

Submitted to:
QAPL 2013

The example above is elementary but it should give an idea of the pervasive nature of the problem, which can have an impact in any confidentiality setting, and should therefore receive attention by those researchers interested in (quantitative) information flow. In this paper, we initiate this investigation with an in-depth study of the particular case of *differential privacy*.

Differential privacy [9, 10] is an approach to the protection of private information that originated in the field of statistical databases and is now investigated in many other domains, ranging from programming languages [3, 11] to social networks [18] and geolocation [15, 13, 2]. The key idea behind differential privacy is that whenever someone queries a dataset, the reported answer should not allow him to distinguish whether a certain individual record is in the dataset or not. More precisely, the presence or absence of the record should not change significantly the probability of obtaining a given answer. The standard way of achieving such a property is by using an *oblivious mechanism*[1] which consists in adding some noise to the true answer. Now the point is that, even if such a mechanism is proved to provide the desired property in the ideal semantics, its implementation may induce errors that alter the least significant digits of the reported answer and cause significant privacy breaches. Let us illustrate the problem with an example.

Example 1.1. Consider the simplest representation of reals: the fixed-point numbers. This representation is used on low-cost processors which do not have floating-point arithmetic module. Each value is stored in a memory cell of fixed length. In such cells, the last d digits represent the fractional part. Thus, if the value (interpreted as an integer) stored in the cell is z, its semantics (i.e., the true real number being represented) is $z \cdot 2^{-d}$.

To grant differential-privacy, the standard technique consists of returning a random value with probability $p(x) = 1/2b \cdot e^{-|x-r|/b}$ where r is the true result and b is a scale parameter which depends on the degree of privacy to be obtained and on the sensitivity of the query. To get a random variable with any specific distribution, in general, we need to start with an initial random variable provided by a primitive of the machine with a given distribution. To simplify the example, we assume that the machine already provides a Laplacian random variable X with a scale parameter 1. The probability distribution of such an X is $p_X(x) = 1/2e^{-|x|}$. Hence, if we want to generate the random variable bX with probability distribution $p_{bX}(x) = 1/2b \cdot e^{-|x|/b}$, we can just multiply by b the value $x = z \cdot 2^{-d}$ returned by the primitive.

Assume that we want to add noise with a scale parameter $b = 2^n$ for some fixed integer n (b can be big when the sensitivity of the query and the required privacy degree are high). In this case, the multiplication by 2^n returns a number $2^n z \cdot 2^{-d}$ that, in the fixed-point representation terminates with n zeroes. Hence, when we add this noise to the true result, we return a value whose representation has the same n last digits as the secret. For example, assume $b = 2^2 = 4$ and $d = 6$. Consider that the true answers are $r_1 = 0$ and $r_2 = 1 + 2^{-5}$. In the fixed-point representation, the last two digits of r_1 are 00, and the last two digits of r_2 are 10. Hence, even after we add the noise, we can always tell whether the true value was r_1 or r_2. Note that the same example holds for every $b = 2^n$ and every pair of true values r_1 and r_2 which differ by $(2^n k+h)/2^d$ where k is any integer and h is any integer between 1 and $2^n - 1$. Figure 1 illustrates the situation for $b = 4$, $k = 3$ and $h = 2$.

Another attack, based on the IEEE standard floating-point representation [14], was presented in [17]. In contrast to [17], we have chosen an example based on the fixed point representation because it allows to illustrate more distinctively a problem for privacy which rises from the finite precision[2] and which

[1] The name "oblivious" comes from the fact that the final answer depends only on the answer to the query and not on the dataset.

[2] More precisely, the problem is caused by scaling a finite set of randomly generated numbers. It is easy to prove that the problem raises for any implementation of numbers, although it may not raise *for every point* like in the case of the fixed-point representation.

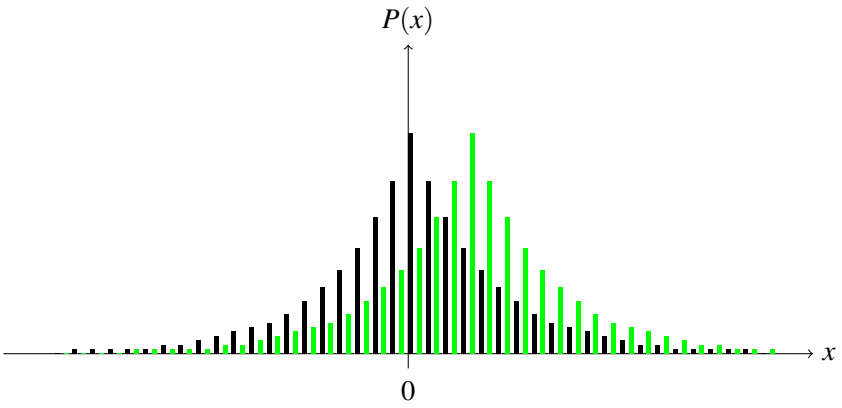


Figure 1: The probability distribution of the reported answers after the addition of Laplacian noise for the true answer $r_1 = 0$ (black) and $r_2 = 3 \cdot 2^{-4} + 2^{-5}$ (green).

is, therefore, pandemic. (This is not the case for the example in [17]: fixed-point and integer-valued algorithms are immune to that attack.)

In this paper, we propose a solution to fix the privacy breach induced by the finite-precision implementation of a differentially-private mechanism for any kind of implementation. Our main concern is to establish a bound on the degradation of privacy induced by both the finite representation and by the computational errors in the generation of the noise. In order to achieve this goal, we use the concept of *closeness* introduced by the authors in [12], which allows us to reason about the approximation errors and their accumulation. We make as few assumptions as possible about the procedure for generating the noise. In particular, we do not assume that the noise has a linear Laplacian distribution: it can be any noise that provides differential privacy and whose implementation satisfies a few properties (normally granted by the implementation of real numbers) which ensure its closeness. We illustrate our method with two examples: the classic case of the univariate (i.e., linear) Laplacian, and the case of the bivariate Laplacian. The latter distribution is used, for instance, to generate noise in privacy-aware geolocation mechanisms [2].

1.1 Related work

As far as we know, the only other work that has considered the problem introduced by the finite precision in the implementation of differential privacy is [17]. As already mentioned, that paper showed an attack on the Laplacian-based implementation of differential privacy within the IEEE standard floating-point representation[3]. To thwart such an attack, the author of [17] proposed a method that avoids using the standard uniform random generator for floating point (because it does not draw all representable numbers but only multiple of 2^{-53}). Instead, his method generates two integers, one for the mantissa and one for the exponent in such a way that every representable number is drawn with its correct probability. Then it computes the linear Laplacian using a logarithm implementation (assumed to be full-precision), and finally it uses a snapping mechanism consisting in truncating large values and then rounding the final result.

The novelties of our paper, w.r.t. [17], consist in the fact that we deal with a general kind of noise, not

[3]We discovered our attack independently, but [17] was published first.

necessarily the linear Laplacian, and with any kind of implementation of real numbers, not necessarily the IEEE floating point standard. Furthermore, our kind of analysis allows us to measure how safe an existing solution can be and what to do if the requirements needed for the safety of this solution are not met. Finally, we consider our correct implementation of the bivariate Laplacian also as a valuable contribution, given its practical usefulness for location-based applications.

The only other work we are aware of, considering both computational error and differential privacy, is [7]. However, that paper does not consider at all the problem of the loss of privacy due to implementation error: rather, they develop a technique to establish a bound on the error, and show that this technique can also be used to compute the sensitivity of a query, which is a parameter of the Laplacian noise.

1.2 Plan of the paper

This paper is organized as follow. In section 2, we recall some mathematical definitions and introduce some notation. In section 3, we describe the standard Laplacian-based mechanism that provides differential privacy in a theoretical setting. In section 4, we discuss the errors due to the implementation, and we consider a set of assumptions which, if granted, allows us to establish a bound on the irregularities of the noise caused by the finite-precision implementation. Furthermore we propose a correction to the mechanism based on rounding and truncating the result. Section 5 contains our main theorem, stating that with our correction the implementation of the mechanism still preserves differential privacy, and establishing the precise degradation of the privacy parameter. The next two sections propose some applications of our result: Section 6 illustrates the technique for the case of Laplacian noise in one dimension and section 7 shows how our theorem applies to the case of the Euclidean bivariate Laplacian. Section 8 concludes and discusses some future work.

2 Preliminaries and notation

In this section, we recall some basic mathematical definitions and we introduce some notation that will be useful in the rest of the paper. We will assume that the the queries give answers in $\mathbb{R}^m$. Examples of such queries are the tuples representing, for instance, the average height, weight, and age. Another example comes from geolocation, where the domain is $\mathbb{R}^2$.

2.1 Distances and geometrical notations

There are several natural definitions of distance on $\mathbb{R}^m$ [19]. For $m \in \mathbb{N}$ and $x = (x_1, \ldots, x_m) \in \mathbb{R}^m$, the L_p norm of x, which we will denote by $\|x\|_p$, is defined as

$$\|x\|_p = \sqrt[p]{\sum_{i=1}^{m} |x_i|^p}$$

The corresponding distance function is

$$d_p(x,y) = \|x-y\|_p$$

We extend this norm and distance to $p = \infty$ in the usual way: $\|x\|_\infty = \max_{i\in\{1,\ldots,m\}} |x_i|$ and $d_\infty(x,y) = \|x-y\|_\infty$. The notion of L_∞ norm is extended to functions in the following way: given $f : A \to \mathbb{R}^m$, we define $\|f\|_\infty = \max_{x\in A} \|f(x)\|$. When clear from the context, we will omit the parameter p and write simply $\|x\|$ and $d(x,y)$ for $\|x\|_p$ and $d_p(x,y)$, respectively.

Let $S \subseteq \mathbb{R}^m$. We denote by S^c the *complement of* S, i.e., $S^c = \mathbb{R}^m \setminus S$. The *diameter of* S is defined as

$$\varnothing(S) = \max_{x,y \in S} d(x,y).$$

For $\varepsilon \in \mathbb{R}^+$, the $+\varepsilon$*-neighbor* and the $-\varepsilon$*-neighbor* of S are defined as

$$S^{+\varepsilon} = \{x \mid \exists s \in S, d(x,s) \leq \varepsilon\} \qquad S^{-\varepsilon} = \{x \mid \forall s \in \mathbb{R}^m, d(x,s) \leq \varepsilon \implies s \notin S\} = ((S^c)^{+\varepsilon})^c$$

For $x \in \mathbb{R}^m$, the *translations* of S by x and $-x$ are defined as

$$S + x = \{y + x \mid y \in S\} \qquad S - x = \{y - x \mid y \in S\}$$

2.2 Measure theory

We recall here some basic notions of measures theory that will be used in this paper.

Definition 2.1 (σ-algebra and measurable space)**.** *A σ-algebra $\mathscr{T}$ for a set M is a nonempty set of subsets of M that is closed under complementation (wrt to M) and (potentially empty) enumerable union. The tuple $(M, \mathscr{T})$ is called a* measurable space.

Definition 2.2 (Measure)**.** *A* positive measure μ *on a measurable space $(M, \mathscr{T})$ is a function $\mathscr{T} \to \mathbb{R}^+ \cup \{0\}$ such that $\mu(\emptyset) = 0$ and whenever (S_i) is a enumerable family of disjoint subset of M then*

$$\sum \mu(S_i) = \mu(\bigcup S_i).$$

A positive measure μ where $\mu(X) = 1$ is called a probability measure.

A tuple $(M, \mathscr{T}, P)$ where $(M, \mathscr{T})$ is a measurable space and P a probability measure is called *probability space*.

In this paper we will make use of the Lebesgue measure λ on $(\mathbb{R}^m, \mathscr{S})$ where $\mathscr{S}$ is the Lebesgue σ-algebra. The Lebesgue measure is the standard way of assigning a measure to subsets of $\mathbb{R}^m$.

Definition 2.3 (Measurable function)**.** *Let $(M, \mathscr{T})$ and (V, Σ) be two measurable spaces. A function $f : M \to V$ is measurable if $f^{-1}(v) \in \mathscr{T}$ for all $v \in \Sigma$.*

Definition 2.4 (Absolutely continuous)**.** *A measure ν is absolutely continuous according to a measure μ, if for all $M \in \mathscr{S}$, $\mu(M) = 0$ implies $\nu(M) = 0$.*

If a measure is absolutely continuous according to the Lebesgue measure then by the Radon-Nikodym theorem, we can express it as an integration of a density function f:

$$\mu(M) = \int_M f(x)\, d\lambda$$

2.3 Probability theory

Definition 2.5 (Random variable)**.** *Let $(\Omega, \mathscr{F}, P)$ be a probability space and $(E, \mathscr{E})$ a measurable space. Then a random variable is a measurable function $X : \Omega \to E$. We shall use the expression $P[X \in B]$ to denote $P\left(X^{-1}(B)\right)$.*

Let $f : \mathbb{R}^m \to \mathbb{R}^m$ be a measurable function and let $X : \Omega \to \mathbb{R}^m$ be a random variable. In this paper, we will use the notation $f(X)$ to denote the random variable $Y : \Omega \to \mathbb{R}^m$ such that $f(X)(\omega) = f(X(\omega))$. In particular, for $m \in \mathbb{R}^m$ we denote by $m + X$ the random variable $Y : \Omega \to \mathbb{R}^m$ such that $\omega \mapsto X(\omega) + m$.

Definition 2.6 (Density function)**.** *Let $X : \Omega \to E$ be a random variable. If there exists a function f such that, for all $S \in \mathscr{S}$, $P[X \in S] = \int_S f(u)\,du$, then f is called the* density function *of X.*

In this paper, we use the following general definition of the Laplace distribution (centered at zero).

Definition 2.7 (Laplace distribution)**.** *The density function F of a Laplace distribution with scale parameter b is $F_b(x) = K(b)e^{-b\|x\|}$ where $K(b)$ is a normalization factor which is determined by imposing $\int_S F_b(x)\,dx = 1$.*

Definition 2.8 (Joint probability)**.** *Let (X,Y) be a pair of random variable on $\mathbb{R}^m$. The joint probability on (X,Y) is defined for all $I,J \in \mathscr{S}$ as: $P[(X,Y) \in (I,J)] = P[X \in I \wedge Y \in J]$.*

Definition 2.9 (Marginals)**.** *Let X and $Y : (\Omega,\mathscr{F},P) \to (\mathbb{R}^m,\mathscr{S})$. The marginal probability of the random variable (X,Y) for X is defined as:*

$$P[X \in B] = \int_{\mathbb{R}^m} P[(X,Y) \in (B,\mathrm{d}y)].$$

3 Differential privacy in the exact semantics

In this section we recall the definition of differential privacy and of the standard mechanisms to achieve it, and we discuss its correctness.

3.1 Differential privacy

We denote by $\mathscr{D}$ the set of databases and we assume that the domain of the answers of the query is $\mathbb{R}^m$ for some $n \geq 1$. We denote by $D_1 \sim D_2$ the fact that D_1 and D_2 differ by at most one element. Namely, D_2 is obtained from D_1 by adding or removing one element.

Definition 3.1 (ε-differential privacy)**.** *A randomized mechanism $\mathscr{A} : \mathscr{D} \to \mathbb{R}^m$ is ε-differentially private if for all databases D_1 and D_2 in $\mathscr{D}$ with $D_1 \sim D_2$, and all $S \in \mathscr{S}$ (the Lebegue σ-algebra), we have :*

$$P[\mathscr{A}(D_1) \in S] \leq e^{\varepsilon} P[\mathscr{A}(D_2) \in S]$$

Definition 3.2 (sensitivity)**.** *The sensitivity Δ_f of a function $f : \mathscr{D} \to \mathbb{R}^m$ is*

$$\Delta_f = \sup_{D_1,D_2 \in \mathscr{D}, D_1 \sim D_2} d(f(D_1), f(D_2)).$$

3.2 Standard technique to implement differential privacy

The standard technique to grant differential privacy is to add random noise to the true answer to the query. In the following, we denote the query by $f : \mathscr{D} \to \mathbb{R}^m$. This is usually a deterministic function. We represent the noise as a random variable $X : \Omega \to \mathbb{R}^m$. The standard mechanism, which we will denote by $\mathscr{A}_0$, returns a probabilistic value which is the sum of the true result and of a random variable X, namely:

Mechanism 1.

$$\mathscr{A}_0(D) = f(D) + X$$

3.3 Error due to the implementation of the query

The correctness of a mechanism $\mathscr{A}$, if we do not take the implementation error into account, consists in $\mathscr{A}$ being ε-differentially private. However, we are interested in analyzing the correctness of the implemented mechanism. We start here by discussing the case in which, in mechanism 1, the noise X is exact but we take into account the approximation error in the implementation of f.

Notation Given a function g, we will indicate by g' its implementation, i.e. the function that, for any x, gives as result the value actually computed for $g(x)$, with all the approximation an representation errors.

The first thing we observe is that the implementation of f can give a sensitivity $\Delta_{f'}$ greater than Δ_f and we need to take that into account. In fact, in the exact semantics the correctness of the mechanism relies on the fact that $d(f(D_1), f(D_2)) \leq \Delta_f$. However, with rounding errors, we may have $d(f'(D_1), f'(D_2)) > \Delta_f$. Hence we need to require the following property, usually stronger than differential privacy.

Condition 1. *Given a mechanism $\mathscr{A}(D) = f'(D) + X$, we say that $\mathscr{A}$ satisfies Condition 1 with degree ε (the desired degree of differential privacy) if the random variable X has a probability distribution which is absolutely continuous according to the Lebesgue measure, and*

$$\forall S \in \mathscr{S}, r_1, r_2 \in \mathbb{R}^m, P[r_1 + X \in S] \leq e^{\varepsilon \frac{d(r_1, r_2)}{\Delta_{f'}}} P[r_2 + X \in S]$$

Remark 1. *In general we expect that an analysis of the implementation of f will provide some bound on the difference between f and f', and that will allow us to provide a bound on $\Delta_{f'}$ in terms of Δ_f. For instance, if $\|f - f'\| \leq \delta_f$ then we get $\Delta_{f'} \leq \Delta_f + 2\delta_f$.*

Proposition 3.1. *Condition 1 implies that the mechanism $\mathscr{A}(D) = f'(D) + X$ is ε-differentially private (w.r.t. f').*

Proof Let D_1 and D_2 be two databases such that $D_1 \sim D_2$. Let $r_1 = f'(D_1)$ and $r_2 = f'(D_2)$ be two answers. By definition of sensitivity, $d(r_1, r_2) \leq \Delta_{f'}$ so $e^{\varepsilon \frac{d(r_1, r_2)}{\Delta_{f'}}} \leq e^{\varepsilon}$. Hence,

$$P[\mathscr{A}(D_1) \in S] \leq e^{\varepsilon} P[\mathscr{A}(D_2) \in S]$$

□

The following theorem shows that Condition 1 is actually equivalent to differential privacy in the case of Laplacian noise.

Theorem 3.1. *Let $\mathscr{A}(D) = f'(D) + X$ be a mechanism, and assume that X is Laplacian. If $\mathscr{A}$ is ε-differentially private (w.r.t. f'), then Condition 1 holds.*

Proof First, we show that if $\mathscr{A}$ is ε-differentially private then $b \leq \frac{\varepsilon}{\Delta_{f'}}$ holds for the scale parameter b of X. Let $D_1 \sim D_2$ with $d(f'(D_1), f'(D_2)) = \Delta_{f'}$. By ε-differential privacy we have, for any $S \in \mathscr{S}$:

$$P[f'(D_1) + X \in S] \leq e^{\varepsilon} P[f'(D_2) + X \in S]$$

From the density function of the Laplace noise (Definition 2.7), we derive:

$$K(n,d) d\lambda \leq e^{\varepsilon} K(n,d) e^{-b\Delta_{f'}} d\lambda$$

Hence,

$$b \leq \frac{\varepsilon}{\Delta_{f'}}. \tag{1}$$

Now, by definition of the density function, we have

$$P[r_2 + X \in S] = \int_{x \in S} K(n,d) e^{-bd(x,r_2)} d\lambda$$

From the triangular inequality, we derive:

$$P[r_2 + X \in S] \geq \int_{x \in S} K(n,d) e^{-b(d(x,r_1)+d(r_1,r_2))} d\lambda$$

Hence,

$$P[r_2 + X \in S] \geq e^{-bd(r_2,r_1)} \int_{x \in S} e^{-bd(r_1,x)} d\lambda$$

From inequality (1), we derive:

$$P[r_2 + X \in S] \geq e^{-\frac{\varepsilon d(r_2,r_1)}{\Delta_{f'}}} \int_{x \in S} e^{-bd(r_1,x)} d\lambda$$

Finally,

$$P[r_2 + X \in S] \geq e^{-\frac{\varepsilon d(r_2,r_1)}{\Delta_{f'}}} P[r_1 + X \in S]$$

□

4 Error due to the implementation of the noise

In this section we consider the implementation error in the noise, trying to make as few assumptions as possible about the implementation of real numbers and of the noise function.

We start with example which shows that any finite implementation makes it impossible for a mechanism to achieve the degree of privacy predicted by the theory (i.e. the degree of privacy it has in the exact semantics). This example is more general than the one in the introduction in the sense that it does not rely on any particular implementation of the real numbers, just on the (obvious) assumption that in a physical machine the representation of numbers in memory is necessarily finite. On the other hand it is less "dramatic" than the one in the introduction, because it only shows that the theoretical degree of privacy degrades in the implementation, while the example in the introduction shows a case in which ε-differential privacy does not hold (in the implementation) for any ε.

Example 4.1. Consider the standard way to produce a random variable with a given probability law, such as the Laplace distribution. Randomness on most computers is generated with integers. When we call a function that returns a uniform random value on the representation of reals, the function generates a random integer z (with uniform law) between 0 and N (in practice $N \geq 2^{32}$) and returns $u = z/N$. From this uniform random generator, we compute $n(z/N)$ where n depends on the probability distribution we want to generate. For instance, to generate the Laplace distribution we have $n(u) = -b\,\mathrm{sgn}(u - 1/2)\ln(1 - 2|u - 1/2|)$ which is the inverse of the cumulative function of the Laplace distribution. However the computation of n is performed in the finite precision semantics, i.e. n is a function $\mathbb{F} \to \mathbb{F}$ where $\mathbb{F}$ is the finite set of the representable numbers. In this setting, the probability of getting some value x for our noise depends on the number of integers z such that $n(z/N) = x$: if there are k values for z such

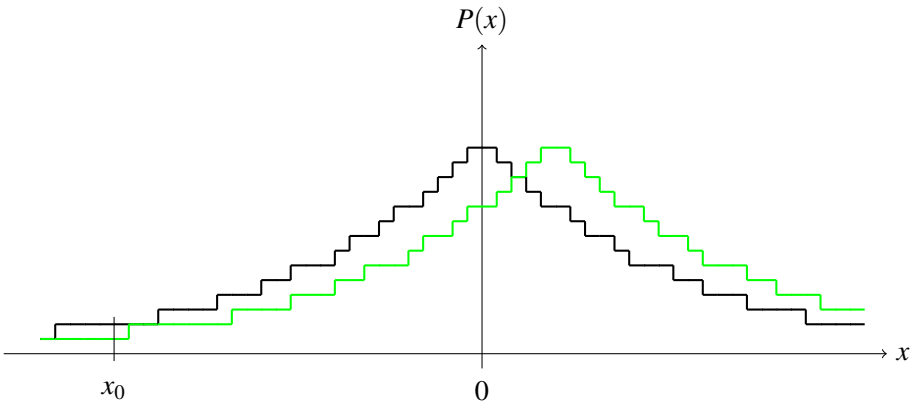


Figure 2: The probability distributions of Laplace noises generated from a discretized uniform generator

that $n(z/N) = x$ then the probability of getting x is k/N. This means, in particular, that, if the theoretical probability for a value x is $1.5/N$, then the closest probability actually associated with the drawing of x is either $1/N$ or $2/N$ and in both cases the error is at least 33%. In figure 2, we illustrate how the error on the distribution breaks the differential-privacy ratio that holds for the theoretical distribution. The ratio between the two theoretical Laplacian distributions is $4/3$. However, since the actual distribution is issued from a discretization of the uniform generator, the resulting distribution is a step function. So when the theoretical probability is very low like in x_0, the discretization creates an artificial ratio of 2 instead of $4/3$.

4.1 The initial uniform random generator

To generate a random variable, programing languages have only one primitive that generates a random value between 0 and 1 that aims to be uniform and independent across several calls.Hence, to get a random variable with a non uniform distribution, we generate it with a function that makes calls to this random generator. For instance, to draw a value from a random variable X on $\mathbb{R}$ distributed according to the cumulative function $C : \mathbb{R} \to]0,1]$, it is sufficient to pick a value u from the uniform generator in $]0,1]$ and then return $C^{-1}(u)$.

We identify three reasons why a uniform random generator may induce errors. The first has been explained in the introduction: finite precision allows generating only N different numbers such that when we apply a function on the value picked some values are missing and other are over represented. The second reason comes from the generator itself which can returns the N values with different probabilities even though we might assume that they are returned with probability $1/N$: furthermore, some values may not even be returned at all. A third error is due to the dependence of returned results when we pick several random values. Indeed most of the generator implementations are indeed pseudo generators: when a value is picked the next one is generated as a hash function of the first one. This means that if we have N possibilities for one choice then we also have N possible pairs of successive random values.

To reason about implementation leakage, we have to take into account all of these sources of errors. We propose the following model. In the exact semantics, the uniform random variable U^q is generated from a cross product of q uniform independent variables U (with $q \geq m$). We denote by $u_1, \ldots, u_q \in [0,1]^q$ the values picked by our perfect random generator. Then we consider the random variable $U^{q\prime}$ actually provided as generated from a function $n_0 : \mathbb{R}^q \to \mathbb{R}^q$, $(u'_1, \ldots, u'_q) = n_0(u_1, \ldots, u_q)$. We assume that the

bias, i.e., the difference between n_0 and the identity, is bounded by some $\delta_0 \in \mathbb{R}^+$:

$$\|n_0 - \mathrm{Id}\|_\infty \leq \delta_0. \tag{2}$$

4.2 The function n for generating the noise

From the value u (resp. u') drawn according to the distribution U^q in the exact semantics (resp. according to $U^{q'}$ in the actual implementation), we generate another value by applying the functions n and n' respectively. Let $X = n(U^q)$ be the random variable with the exact distribution and $X' = n'(U^{q'})$ the random variable with the actual one.

Definition 4.1. *We denote by μ and ν the probability measure of X and X', respectively: for all $S \in \mathscr{S}$, $\mu(S) = P[n(U^q) \in S] = \lambda(n^{-1}(S))$ and $\nu(S) = P[n'(U^{q'}) \in S] = \lambda(n_0^{-1}(n^{-1}(S)))$.*

In order to establish a bound on the difference between the probability distribution of X and X' we need some condition on the implementation n' of n. For this purpose we use the notion of closeness that we defined in [12].

Definition 4.2 ((k,δ)-close, [12])**.** *Let A and B be metric spaces with distance d_A and d_B, respectively. Let n and n' be two functions from A to B and let $k, \delta \in \mathbb{R}^+$. We say that n' is (k,δ)-close to n if*

$$\forall u, v \in A, d_B(n(u), n'(v)) \leq k\, d_A(u,v) + \delta.$$

This condition is a combination of the k-Lipschitz property, that states a bound between the error on the output and the error on the input, see below, and the implementation errors of n:

Definition 4.3 (k-Lipschitz)**.** *Let (A, d_A) and (B, d_B) be two metrics spaces and $k \in \mathbb{R}$: A function $n : A \to B$ is k-Lipschitz if:*

$$\forall u, v \in A, d_B(n(u), n(v)) \leq k d_A(u,v)$$

In [12], we have proven the following relation between the properties of being k-Lipschitz and of being (k,δ)-close.

Theorem 4.1 ([12])**.** *If n is k-Lipschitz and $\|n - n'\|_\infty \leq \delta$ then n and n' are (k,δ)-close.*

We strengthen now the relation by proving that (a sort of) the converse is also true.

Theorem 4.2. *If there exist u, v, $d(n(u), n(v)) > kd(u,v) + 2\delta$ then there exist no function n' such that n and n' are (k,δ)-close.*

Proof Let u, v such that $d(n(u), n(v)) > kd(u,v) + 2\delta$. Let n' such that n and n' are (k,δ)-close. From the definition of closeness, we get $d(n(u), n'(u)) \leq \delta$ and $d(n'(u), n(v)) \leq kd(u,v) + \delta$. From a triangular inequality, we derive $d(n(u), n(v)) \leq kd(u,v) + 2\delta$. Hence, we obtain a contradiction. □

Now, we would like n and n' to be (k,δ)-close on $\mathbb{R}^m$. However, this implies that $\mathscr{A}_0$ (mechanism 1) cannot be ε-differentially-private. In fact, the latter would imply $\|n([0,1]^q)\|_\infty = \infty$ otherwise certain answers could be reported (with non-null probability) only in correspondence with certain true answers and not with others. However, $\|n([0,1]^q)\|_\infty = \infty$ and $[0,1]^q$ bounded implies there exist u, v, such that $d(n(u), n(v)) > kd(u,v) + 2\delta$: we derive from theorem 4.2 that n' cannot exist.

In order to keep computed results in a range where we are able to bound the computational errors, one possible solution consists of a truncation of the result. The traditional truncation works as follows: choose a subset $\mathbb{M}_r \subset \mathbb{R}^m$ and, whenever the reported answer x is outside $\mathbb{M}_r$ return the closest point to x in $\mathbb{M}_r$. However, while such a procedure is safe in the exact semantics because remapping does not alter differential privacy, problems might appears when n and n' are not close. Furthermore, while in the

uni-dimensional case there are two disjoints set that are mapped one on the minimal value and the other on the maximal value, in higher dimensions we have a connected set that is mapped on several points, and on which the error is not bounded.

Therefore, to remain in a general framework where we do not have any additional knowledge about computational errors for large numbers, we decide here to return an exception value when the result is outside of some compact subset $\mathbb{M}_r$ of $\mathbb{R}^m$. We denote by ∞ the value returned by the mechanism when $f'(D)+X' \notin \mathbb{M}_r$. Hence, the truncated mechanism $\mathscr{A}$ returns the randomized value or ∞:

Mechanism 2.

$$\mathscr{A}(D) = \begin{cases} f'(D)+X' & \text{if } f'(D)+X' \in \mathbb{M}_r \\ \infty & \text{otherwise} \end{cases}$$

We truncate the result because we want to exclude non-robust computations from our mechanism. However, such a procedure is effective only if unsafe computations remain outside the safe domain. To grant this property we need two more conditions. One requires the implementation to respect the monotonicity of the computed functions:

Condition 2. *We say that a function $g : \mathbb{R}^m \to \mathbb{R}^k$ satisfies Condition 2 if, for all $x, y \in \mathbb{R}^m$, $\|g(x)\| \leq \|g(y)\|$ implies $\|g'(x)\| \leq \|g'(y)\|$.*

With this property, even if the implementation is not robust for large values, if we know some result is not in $\mathbb{M}_r$ then the result for any greater value is not in $\mathbb{M}_r$ either.

The other condition is about the closeness of the implementation of the noise and its exact semantics in a safe area. For any $\delta_r \in \mathbb{R}^+$, we consider the set $U_r \subset U^q$ defined as $\forall u \in U_r \ \|n(u)\| \leq \varnothing(\mathbb{M}_r) + \delta_r$ i.e. : $U_r = n^{-1}(\{y \,|\, \|y\| \leq \varnothing(\mathbb{M}_r) + \delta_r\})$.

Condition 3. *We say that a noise n satisfies Condition 3 if n and n' are (k, δ_n)-close on a set U_r such that*

$$\forall u \in U_r^c \ f'(D) + n(u) \notin \mathbb{M}_r^{+k\delta_0+\delta_n}$$

To find such a set U_r, one possible way is by a fix point construction. We begin by finding the smallest k_0 and δ_{n0} such that n and n' are (k_0, δ_{n0})-close on $\mathbb{M}_r$. Then for the generic step $m > 0$, we compute the smallest k_{m+1} and δ_{nm+1} such that n and n' are (k_{m+1}, δ_{nm+1})-close on $\mathbb{M}_r^{k_m\delta_0+\delta_{nm}}$.

If Conditions 2 and 3 hold, then from (2) we derive

$$\forall u \in U_r^c \ f'(D) + n'(n_0(u)) \notin \mathbb{M}_r \tag{3}$$

So whatever happens outside of U_r, the result will be truncated. We can then consider that there is no implementation error outside U_r. Finally, we have a bound δ_t for the maximal shift between the exact and the actual semantics:

$$\delta_t = k\delta_0 + \delta_n \tag{4}$$

4.3 A distance between distributions

Given that we are in a probabilistic setting, the round-off errors cannot be measured in terms of numerical difference as they can be in the deterministic case, they should rather be measured in terms of distance between the theoretical distribution and the actual distribution. Hence, we need a notion of distance between distributions. We choose to use the ∞-Wassertein distance [5] which, as we will show, is the natural metric to measure our deviation.

Definition 4.4 (∞-Wassertein distance)**.** *Let μ, ν two probability measures on $(\mathbb{R}^m, \mathscr{S})$ such that there exist a compact Ω, $\mu(\Omega) = \nu(\Omega) = 1$, the ∞-Wassertein distance between μ and ν is defined as follows:*

$$W_\infty(\mu,\nu) = \inf_{\gamma\in\Gamma(\mu,\nu)} \left(\inf_{t\geq 0} \left(\gamma\left(\left\{ (x,y) \in (\mathbb{R}^m)^2 \middle| d(x,y) > t \right\} \right) \right) = 0 \right)$$

Where $\Gamma(\mu,\nu)$ denotes the collection of all measure on $M \times M$ with marginals μ and ν respectively.

If we denote by $\mathrm{Supp}(x,y)$, the support where $\gamma(x,y)$ is non zero, we have an equivalent definition [5] for the ∞-Wassertein distance:

$$W_\infty(\mu,\nu) = \inf_{\gamma\in\Gamma(\mu,\nu)} \left(\sup_{\mathrm{Supp}(x,y)} d(x,y) \right)$$

We extend this definition to any pair of measures that differ only on a compact ($\mathbb{M}_r$ in our case) by considering the subset of $\Gamma(\mu,\nu)$ containing only measure $\gamma(x,y)$ with $\gamma(x,y) = 0$ if $x \neq y$ and either $x \in \mathbb{M}_r^c$ or $y \in \mathbb{M}_r^c$.

We have introduced this measure because it has a direct link with the computational error as expressed by the following theorem.

Theorem 4.3. *Let X and X' be two random variables with distribution μ and ν respectively. We have that $\|X - X'\|_\infty \leq \delta$ implies $d(\mu,\nu) \leq \delta$.*

Proof We consider the measure γ on $M \times M$, $\forall A, B \in \mathscr{S}, \gamma(A,B) = P(X \in A \wedge X' \in B)$. The marginals of γ are μ and ν. Moreover, the support of γ is δ since $P(X \in A \wedge X' \in B) = 0$ when A and B are distant by more than δ. Since we have such a γ the minimum on all the $\gamma \in \Gamma(\mu,\nu)$ is less than δ. □

In our case, according to (4), we have $d(\mu,\nu) \leq \delta_t$. The following theorem allow us to bound the μ measure of some set with the measure ν.

Theorem 4.4.

$$d(\mu,\nu) \leq \varepsilon \implies \forall S \in \mathbb{R}^m, \nu(S^{-\varepsilon}) \leq \mu(S) \leq \nu(S^\varepsilon)$$

Proof The property of marginals is $\nu(S) = \int_{\mathbb{R}^m \times S} d\gamma(x,y)$. Since $\gamma(x,y) = 0$ if $d(x,y) > \varepsilon$, we derive $\nu(S) = \int_{S^\varepsilon \times S} d\gamma(x,y)$. Then we get $\nu(S) \leq \int_{S^\varepsilon \times \mathbb{R}^m} d\gamma(x,y)$. The last expression is the marginal of γ in S^ε, hence by definition of marginal: $\nu(S) \leq \mu(S^\varepsilon)$. The other inequality is obtain by considering the complement set of S ($\mathbb{R}^m \setminus S$). □

4.4 Rounding the answer

Once the computation of $\mathscr{A}(D)$ is achieved, we cannot yet return the answer, because it could still leak some information. Indeed, the distribution of X and X' are globally the same, but, on a very small scale, the distributions could differ a lot. We prevent this problem by rounding the result:

Mechanism 3. *The mechanism rounds the result by returning the value closest to $f(D) + n'$ in some discrete subset S'. So $\mathscr{K}(D) = r(\mathscr{A}(D))$ where r is the rounding function.*

From the above rounding function we define the set $\mathscr{S}'_0$ of all sets that have the same image under r. Then we define the σ-algebra $\mathscr{S}'$ generated by $\mathscr{S}'_0$: it is the closure under union of all these sets. Observe now that it is not possible for the user to measure the probability that the answer belongs to a set which is not in $\mathscr{S}'$. Hence our differential privacy property becomes:

$$\forall S \in \mathscr{S}', P[\mathscr{A}(D_1) \in S] \leq e^\varepsilon P[\mathscr{A}(D_2) \in S] \tag{5}$$

In this way we grant that any measurable set has a minimal measure and we prevent the inequality from being violated when probabilities are small. The following value R represents the robustness of the rounding.

$$R = \max_{S \in \mathscr{S}'_0, S \neq \emptyset} \frac{\lambda(S^{\varepsilon} \setminus S^{-\delta_t})}{\lambda(S^{-\delta_t})} \tag{6}$$

5 Preserving differential privacy

In this section, we prove that if all conditions are met, then the implementation of the mechanism satisfies differential privacy.

Theorem 5.1. *Any mechanism that respects Conditions 1–3 is ε'-differentially private, with:*

$$\forall S \in \mathscr{S}, P[\mathscr{A}'(D_1) \in S] \leq e^{\varepsilon'} P[\mathscr{A}'(D_2) \in S']$$

where $\varepsilon' = \varepsilon + \ln(1 + Re^{\varepsilon \frac{L+\delta_t}{\Delta_{f'}}})$, $\delta_t = k\delta_0 + \delta_n$ *and* $L = \max_{S \in \mathscr{S}'_0} \varnothing S$.

Proof Let S in $\mathscr{S}$. We first consider the case $S \neq \infty$.

Define $P_1 = P[\mathscr{A}'(D_1) \in S_a]$ and $P_2 = P[\mathscr{A}'(D_2) \in S_a]$. Since the result has been rounded (Definition 3), it is equivalent to consider the set $S' \in \mathscr{S}'$ with $S' = r^{-1}(S)$ instead of S_a.

Now we have $P_i = P[f'(D_i) + n'(X) \in S'] = P[n'(X) \in S' - f'(D_i)]$ where i is 1 or 2. Since ν is the measure associated to n', we have

$$P_i = \nu(S' - f'(D_i))$$

From (4) and Theorem 4.3, $d(\nu, \mu) \leq \delta_t$. From Theorem 4.4 we derive

$$P_1 \leq \mu(S^{\delta_t} - f'(D_1)) \qquad \text{and} \qquad P_2 \geq \mu(S^{-\delta_t} - f'(D_2)).$$

The additivity property of measures grants us $\mu(S^{\delta_t}) = \mu(S^{-\delta_t}) + \mu(S^{\delta_t} - S^{-\delta_t})$. Condition 1 can be expressed in term of the measure as:

$$\forall S \in \mathscr{S}, r \in \mathbb{R}^m \|r\|, \mu(S) \leq e^{\varepsilon \frac{\|r\|}{\Delta_{f'}}} \mu(S - r)$$

From this inequality, we can derive, since $\|r\| = \Delta_{f'}$:

$$\mu(S^{\varepsilon}) \leq e^{\varepsilon} P_2 + \mu(S^{\delta_t} \setminus S^{-\delta_t})$$

Since the probability is absolutely continuous according to the Lebesgue measure (Condition 1), we can express the probability with a density function p:

$$\forall S \in \mathscr{S}, \mu(S) = \int_S p(x) d\lambda$$

We derive:

$$\forall S \in \mathscr{S}, \min_{x \in S} p(x) \leq \frac{\mu(S)}{\lambda(S)}$$

By applying this property on $S^{-\delta_t} - f'(D_2)$, we get:

$$\min_{x \in S^{-\delta_t} - f'(D_2)} p(x) \leq \frac{\mu(S^{-\delta_t} - f'(D_2))}{\lambda(S^{-\delta_t} - f'(D_2))}$$

We derive:

$$\exists x_0 \in S - f'(D_2), p(x_0) \leq \frac{P_2}{\lambda(S)}$$

By the triangular inequality, we can bound the distance between x_0 and any point of S^{δ_t} by $\Delta_{f'} + L + \delta_t$ Hence, from Condition 1 we derive:

$$\forall x \in S^{\delta_t} - f'(D_1), p(x) \leq e^{\varepsilon \frac{\Delta_{f'}+L+\delta_t}{\Delta_{f'}}} p(x_0)$$

Then by integration:

$$\mu(S^{\delta_t} - f'(D_1) \setminus S^{-\delta_t}) \leq e^{\varepsilon \frac{\Delta_{f'}+L+\delta_t}{\Delta_{f'}}} \frac{\lambda(S^{\delta_t} \setminus S^{-\delta_t})}{\lambda(S^{-\delta_t})} P_2$$

We apply the condition 6:

$$\mu(S^{\delta_t} - f'(D_1) \setminus S^{-\delta_t}) \leq e^{\varepsilon \frac{\Delta_{f'}+L+\delta_t}{\Delta_{f'}}} RP_2$$

Finally we obtain :

$$P_1 \leq (1 + Re^{\varepsilon \frac{L+\delta_t}{\Delta_{f'}}}) e^{\varepsilon} P_2$$

In case S is ∞, due to (3), $P[\mathscr{A}'(D) = \infty]$ is the same as $P[f'(D) + X' \in \mathbb{M}_r^c]$ where $d(\mu, \nu) \leq \delta_t$. Moreover, $\mathbb{M}_r^c$ can be decomposed in a enumerable disjoint union of element of $\mathscr{S}_0$. Therefore, the first part of the proof applies: ε'-differential privacy holds for all these elements. By the additivity of the measure of disjoint union we conclude.

□

6 Application to the Laplacian noise in one dimension

In this section we illustrate how to use our result in the case in which the domain of the answers is $\mathbb{R}$. The noise added for the protocol, stated in the mechanism 1, is the Laplacian centered in 0 with scale parameter $\Delta_{f'}/\varepsilon$. Theorem 3.1 implies that Condition 1 holds for ε. We truncate the result outside of some interval $\mathbb{M}_r = [m, M]$.

Implementation of the n function To generate a centered Laplacian distribution from a uniform random variable U in $]0,1]$, a standard method consists in using the inverse of the cumulative function, i.e. $X = n(U) = -b\,\mathrm{sgn}(U - 1/2)\ln(1 - 2|U - 1/2|)$, where b is the intended scale parameter ($\frac{\Delta_{f'}}{\varepsilon}$ in our case). Hence our exact function n is

$$n(u) = \frac{\Delta_{f'}}{\varepsilon} \mathrm{sgn}(x - 1/2)\ln(1 - 2|x - 1/2|). \tag{7}$$

Closeness of n and n' In order to apply our theorem, we need to prove that Condition 3 is satisfied. By theorem 4.1, it is sufficient to prove that, in the interval of interest, $n(u)$ is k-Lipschitz and that $|n(u) - n'(u)| \leq \delta_n$. Note that the values of δ_n and k in general depend on n and on its implementation (often the logarithm is implemented by the CORDIC algorithm).

The logarithm function used by n is not k-Lipschitz for any k. However, we are interested in the behavior of n when $|n(u)| \leq M - m$. From the definition of n in (7), we have:

$$\frac{dn}{du}(u) \leq dn_{\max} = \frac{2\Delta_{f'}}{\varepsilon} e^{\frac{\varepsilon \varnothing(\mathbb{M}_r)}{\Delta_{f'}}}$$

in $U_r = \{u | n(u) \leq M - m\}$. So our function n is $dn_{\max}$-Lipschitz. Finally, our global error is

$$\delta_t = \frac{2\Delta_{f'}}{\varepsilon} e^{\frac{\varepsilon \varnothing(\mathbb{M}_r)}{\Delta_{f'}}} \delta_0 + \delta_n$$

Rounding the result The rounding process generates a σ-algebra $\mathscr{S}'$ composed by small intervals of length L where L is the accuracy step of the rounding. In that case, the value defined in (6) is $R = \frac{L+2\delta_t}{L-2\delta_t}$.

Differential privacy By Theorem 5.1, the implementation of our mechanism is ε'-differentially private with

$$\varepsilon' = \varepsilon + \ln(1 + \frac{L+2\delta_t}{L-2\delta_t} e^{\varepsilon \frac{L+\delta_t}{\Delta_{f'}}})$$

Remark 2. *In case our answer is not in $[m,M]$, we can return $-\infty$ or $+\infty$ instead of ∞. The reason is that even if the algorithm is not robust when $|u-0.5|$ is small the sign is still correct. Then we can remap $-\infty$ to m and $+\infty$ to M to get the usual truncation procedure.*

7 Application to the Laplacian noise in $\mathbb{R}^2$

When the domain of the answers are the points of a map, like in the case of location-based applications, it is natural to formalize it as the space $\mathbb{R}^2$ equipped with the Euclidean distance.

According to the protocol, we sanitize the results by adding a random variable X. In this case, we will use for X the bivariate Laplacian defined for the Euclidean metric [2] whose density function is:

$$p(x,y) = Ke^{b\sqrt{|x-x_0|^2+|y-y_0|^2}}$$

where K is the normalization constant and b the scale parameter. Since we are using a Laplacian noise, by Theorem 3.1, Condition 1 holds.

Truncation Since most of the time the domain studied is bound (for instance the public transportation of a city is inside the limit of the city), we can do a truncation. However, we recall that our truncation is made for robustness purpose and not just for utility reasons. Hence, if our domain of interest is a circle, we will not choose $\mathbb{M}_r$ to be the same circle because the probability the truncation would return an exception would be too high (more than one half if the true result is on the circumference).

Implementation of the n function Following [2], we compute the random variable by drawing an angle and a distance independently. The angle θ is uniformly distributed in $[-\pi, \pi[$. The radius r has a probability density $D_{\varepsilon,R}(r) = \varepsilon^2 r e^{-\varepsilon r}$ and cumulative function $C_\varepsilon(r) = 1 - (1+\varepsilon r)e^{-\varepsilon r}$. The radius can therefore be drawn by setting $r = C_\varepsilon^{-1}(u)$ where u is generated uniformly in $]0,1]$.

Robustness of n As in the previous section, we do not analyze an actual implementation but we care about the k factor used for Condition 3. First, we analyze for which $k_C(\varepsilon, \varnothing(\mathbb{M}_r))$ the function C_ε^{-1} is k-Lipschitz in $[0, \varnothing(\mathbb{M}_r)]$. Since C is differential, this question is equivalent to find the inverse of the minimal value taken by its derivative function on the interval $C_\varepsilon^{-1}([0, \varnothing(\mathbb{M}_r)])$. By computing this minimum value, we get:

$$K_C(\varepsilon, \varnothing(\mathbb{M}_r)) = \frac{e^{\varepsilon \varnothing(\mathbb{M}_r)}}{2\varepsilon + r\varepsilon^2}$$

On the other hand, the computation of θ is just a multiplication by 2π of the uniform generator hence $k_\theta = 2\pi$. Then, with the conversion $(r, \theta) \mapsto (r\cos(\theta), r\sin(\theta))$ from polar coordinates to Cartesian coordinates we obtain the global k factor:

$$k = \sqrt{K_C(\varepsilon, \varnothing(\mathbb{M}_r))^2 + 2\pi\varnothing(\mathbb{M}_r)}$$

Let δ_n be the distance between n and n', and δ_0 be the error of the uniform generator. From (4) we get:

$$\delta_t = \sqrt{K_C(\varepsilon, \varnothing(\mathbb{M}_r))^2 + 2\pi\varnothing(\mathbb{M}_r)}\delta_0 + \delta_n.$$

Rounding the answer We now compute the parameter R in (6). The rounding is made in the Cartesian coordinates, hence the inverse image of any returned value is a square S of length L. Note that S^{δ_t} is included in the square of length $L + 2\delta_t$ and $S^{-\delta_t}$ is a square of length $L - 2\delta_t$. Hence the ratio value is smaller than $R = (\frac{L+2\delta_t}{L-2\delta_t})^2$.

Differential privacy By Theorem 5.1 we get that (the implementation of) our mechanism is ε'-differentially private with

$$\varepsilon' = \varepsilon + \ln(1 + (\frac{L+2\delta_t}{L-2\delta_t})^2 e^{\varepsilon \frac{L+\delta_t}{\Delta_{f'}}})$$

8 Conclusion and future work

In this paper we have shown that, in any implementation of mechanisms for differential privacy, the finite precision representation of numbers in any machine induces approximation errors that cause the loss of the privacy property. To solve this problem, we have proposed a method based on rounding the answer and raising an exception when the result is outside some values. The main result of our paper is that the above method is sound in the sense that it preserves differential privacy at the price of a degradation of the privacy degree. To prove this result, we needed to pay special attention at expressing the problem in terms of probability theory and at defining the link between computational error and distance between probability distributions. Finally, we have shown how to apply our method to the case of the linear Laplacian and to that of bivariate Laplacian.

As future developments of this work, we envisage two main lines of research:

- Deepening the study of the implementation error in differential privacy: there are several directions that seem interesting to pursue, including:
  - Improving the mechanisms for generating basic random variables. For instance, when generating a one-dimensional random variable, it may have some advantage to pick more values from the uniform random generator, instead than just one (we recall that the standard method

is to draw one uniformly distributed value in $]0,1]$ and then apply the inverse of the cumulative function). For instance, $u_1 + u_2$ has a density function with a triangular shape and cost only one addition. The other advantage is due to the finite representation: if the uniform random generator can pick N different values then two calls of it generate N^2 possibilities, which enlarge considerably the number of possibilities, and therefore reduce the "holes" in the distribution.

– Considering more relaxed versions of differential privacy, for instance the (ε, δ)-differential privacy allows for a (small) additive shift δ between the two likelihoods in Definition 3.1 and it is therefore more tolerant to the implementation error. It would be worth investigating for what values of δ (if any) the standard implementation of differential privacy is safe.

- Enlarging the scope of this study to the more general area of quantitative information flow. There are various notions of information leakage that have been considered in the computer security literature; the one considered in differential privacy is just one particular case. Without the pretense of being exhaustive, we mention the information-theoretic approaches based on Shannon entropy [8, 16, 6] and those based on Rényi min-entropy [20, 4] and the more recent approach based on decision theory [1]. The main difference between differential privacy and these other notions of leakage is that in the former any violation of the bound in the likelihood ratio is considered catastrophic, while the latter focuses on the average amount of leakage, and it is therefore less sensitive to the individual violations. However, even though the problem of the implementation error may be attenuated in general by the averaging, we expect that there are cases in which it may still represent a serious problem.

References

[1] Mário S. Alvim, Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Geoffrey Smith. Measuring information leakage using generalized gain functions. In *Proc. of CSF*, pages 265–279, 2012.

[2] Miguel E. Andrés, Nicolás E. Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Geo-indistinguishability: Differential privacy for location-based systems. Technical report, 2012. Available at `arXiv:1212.1984`.

[3] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic relational reasoning for differential privacy. In *Proc. of POPL*. ACM, 2012.

[4] Christelle Braun, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Quantitative notions of leakage for one-try attacks. In *Proc. of MFPS*, volume 249 of *ENTCS*, pages 75–91. Elsevier, 2009.

[5] T. Champion, L. De Pascale, and P. Juutinen. The ∞-wasserstein distance: Local solutions and existence of optimal transport maps. *SIAM Journal on Mathematical Analysis*, 40(1):1–20, 2008.

[6] Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Prakash Panangaden. Anonymity protocols as noisy channels. *Inf. and Comp.*, 206(2–4):378–401, 2008.

[7] Swarat Chaudhuri, Sumit Gulwani, Roberto Lublinerman, and Sara NavidPour. Proving programs robust. In *Proc. of ESEC-13*, pages 102–112. ACM, 2011.

[8] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantified interference for a while language. In *Proc. of QAPL*, volume 112 of *ENTCS*, pages 149–166. Elsevier, 2005.

[9] Cynthia Dwork. Differential privacy. In *Proc. of ICALP*, volume 4052 of *LNCS*, pages 1–12. Springer, 2006.

[10] Cynthia Dwork, Frank Mcsherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Proc. of TCC*, volume 3876 of *LNCS*, pages 265–284. Springer, 2006.

[11] Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Ravi Narayan, and Benjamin C. Pierce. Linear dependent types for differential privacy. In *Proc. of POPL 2013. To appear.*

[12] Ivan Gazeau, Dale Miller, and Catuscia Palamidessi. A non-local method for robustness analysis of floating point programs. Technical report, INRIA, June 2012. Available at `http://hal.inria.fr/hal-00665995`.

[13] Shen-Shyang Ho and Shuhua Ruan. Differential privacy for location pattern mining. In *Proc. of SPRINGL*, pages 17–24. ACM, 2011.

[14] IEEE Task P754. *IEEE 754-2008, Standard for Floating-Point Arithmetic*. IEEE, pub-IEEE-STD:adr, August 2008.

[15] Ashwin Machanavajjhala, Daniel Kifer, John M. Abowd, Johannes Gehrke, and Lars Vilhuber. Privacy: Theory meets practice on the map. In *Proc. of ICDE*, pages 277–286. IEEE, 2008.

[16] Pasquale Malacaria. Assessing security threats of looping constructs. In *Proc. of POPL*, pages 225–235. ACM, 2007.

[17] Ilya Mironov. On significance of the least significant bits for differential privacy. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 650–661, New York, NY, USA, 2012. ACM.

[18] Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *Proc. of S&P*, pages 173–187. IEEE, 2009.

[19] Walter Rudin. *Real and Complex Analysis*. McGraw-Hill, 3rd edition, 1986.

[20] Geoffrey Smith. On the foundations of quantitative information flow. In *Proc. of FOSSACS*, volume 5504 of *LNCS*, pages 288–302. Springer, 2009.

A differentially private mechanism of optimal utility for a region of priors*

Ehab ElSalamouny[1,2], Konstantinos Chatzikokolakis[1], and Catuscia Palamidessi[1]

[1] INRIA and LIX, Ecole polytechnique, France
[2] Faculty of Computer and Information Science, Suez Canal University, Egypt

Abstract. The notion of differential privacy has emerged in the area of statistical databases as a measure of protection of the participants' sensitive information, which can be compromised by selected queries. Differential privacy is usually achieved by using mechanisms that add random noise to the query answer. Thus, privacy is obtained at the cost of reducing the accuracy, and therefore the *utility*, of the answer. Since the utility depends on the user's side information, commonly modelled as a prior distribution, a natural goal is to design mechanisms that are optimal *for every prior*. However, it has been shown that such mechanisms *do not exist* for any query other than (essentially) counting queries ([1]).
Given the above negative result, in this paper we consider the problem of identifying *a restricted class of priors* for which an optimal mechanism *does exist*. Given an arbitrary query and a privacy parameter, we geometrically characterise a special region of priors as a convex polytope in the priors space. We then derive upper bounds for utility as well as for min-entropy leakage for the priors in this region. Finally we define what we call the *tight-constraints mechanism* and we discuss the conditions for its existence. This mechanism reaches the bounds for all the priors of the region, and thus it is optimal on the whole region.

1 Introduction

Statistical databases are commonly used to provide aggregate information about the individuals of a certain population, to attain a social benefit. In general, certain data of the participants in the database may be confidential, and we should not allow queries that can reveal them. On the other hand we would like to allow global queries, like, for instance, the average salary of the inhabitants of a certain region, the percentage of individuals having a certain disease, or the cities with the highest rates of crime. This kind of information can be extremely useful for e.g. financial planning, medical research, and anti-crime measures.

Unfortunately, even though these kinds of queries do not refer directly to the individual data, they still represent a major threat to the privacy of the participants in the databases. To illustrate the problem, consider a database whose records contain personal data, among which the salary, regarded as confidential. Suppose we are allowed to query the number of participants and their average salary. Then, by querying the

* This work is partially funded by the Inria large scale initiative CAPPRIS, the EU FP7 grant no. 295261 (MEALS), and the project ANR-09-BLAN-0169-01 (PANDA).

database before and after the insertion of a new record "Bob", we can easily infer, by an easy calculation, the exact salary of Bob.

A successful approach to solve the above problem is to report to the user an approximate answer instead of the exact one. The approximate answer is produced by adding controlled *random noise* to the exact answer. The overall procedure, representing the sanitized query, is a (probabilistic) *mechanism* $\mathcal{K}$ which takes as input the database v and reports to the user an output o in some domain O, according to some probabilistic distribution. Intuitively, the uncertainty introduced at the level of the global answer induces uncertainty about the value of the individual data in the database, thus making it difficult for an attacker to guess such value. However it is crucial to know *exactly* what kind of protection is achieved this way. *Differential privacy*, introduced by Dwork ([2–5]), is a formalization of the privacy property that can be guaranteed by such mechanism. It is a quantitative notion, in the sense that it depends on a parameter ϵ representing the provided level of privacy.

Following common lines (e.g. [6–8]), in this paper we assume that the mechanism $\mathcal{K}$ is *oblivious* with respect to the given query f. Namely, its output depends only on the exact query result and not on the underlying database. Furthermore, we consider only the case in which the domains of the answers (exact and reported) are finite. Under these assumptions, the mechanism $\mathcal{K}$ is determined by an underlying stochastic *noise matrix* X whose generic element x_{io} is the conditional probability of reporting the answer o when the exact query answer is i.

Besides guaranteeing differential privacy, a mechanism should of course provide an answer which is still "useful" enough to the user asking the query. This second goal is measured in terms of *utility*, which represents the average gain that a rational user obtains from the reported answer. More precisely, on the basis of the reported answer o the user can make a guess k (remapping) about the exact hidden query result i. His gain $g(i, k)$ is established by a given function g. The utility is then defined as *the expected gain under the best possible remapping*. While the gain function can take various forms, in this paper we restrict our analysis to the *binary* gain function, which evaluates to 1 when the user's guess is the same as the query result ($k = i$) and evaluates to 0 otherwise.

The utility of a mechanism depends on the side-information which the user may have about the database. This knowledge induces a probability distribution, called 'prior', over the possible query results. Suppose for example that a user "Alice" knows that all people in the database have a salary of at least 20K €. Thus Alice expects the average of the salaries to be at least 20K €. This is reflected on Alice's prior over the average-salary query results: the total probability mass is distributed on the range of values $\geq$ 20K, while it is 0 on lower values. Given this prior, a mechanism X producing only outputs $\geq$ 20K is intuitively more useful to Alice than another one generating also values $<$ 20K, which are less informative for Alice.

The *optimal* mechanism for a given prior and level of privacy ϵ is defined as the mechanism which maximises the utility function, while satisfying ϵ-differential privacy. Naturally, we do not want to change the mechanism depending on the user, so we would like to devise mechanisms which are *universally optimal*, i.e. optimal for *any* prior. A famous result by Gosh et al. [6] states that this is possible for the so-called *counting queries*, which are queries concerned with questions of the kind "how many

records in the database have the property $\mathcal{P}$?" (for some $\mathcal{P}$). In [6] it was proved that the *truncated geometric mechanism* is optimal, for this type of queries, for all priors. Of course the question immediately arises whether we can obtain a similar result for other queries as well. Unfortunately Brenner and Nissim answered this question negatively, by showing that for any query other than (essentially) counting queries a universally optimal mechanism does not exist [1]. However, one can still hope that, also for other queries, by restricting the class of users (i.e. the domain of priors), one could find mechanisms that are optimal for all the users of the class. This is exactly the objective of the present paper: given a query, we aim at identifying a mechanism, and a class of users, for whom that same mechanism provides ϵ-differential privacy and maximal utility at the same time.

Given an arbitrary query and a privacy level $\epsilon > 0$, we call *ϵ-regular* the priors, for which, the probabilities of two adjacent answers (i.e. answers obtained from databases that differ for only one record) are not very different (their ratio is bounded by e^ϵ). At the same time, they may assign significantly different probabilities to "distant" answers. As an example of such prior, consider a researcher "Alice" in a medical school who is interested in the incidence of a certain disease in a statistical medical database containing 1000 records. (Each record represents a person and contains a field saying whether or not the person is infected.) Assume that Alice's side knowledge lets her to expect that the percentage of infected people is likely to be, say, between 1% and 2%, while it is highly unlikely to be higher than 5%. Also, assume that Alice does not have "sharp" enough information to assign significantly different probabilities to adjacent answers, e.g. 1.5% (15 people affected) and 1.6% (16 people affected). It is precisely this kind of users that we target in this paper: we will see that, under certain conditions, we can design a mechanism which maximises the utility for all of them.

A related issue that we consider in this paper is the amount of information leaked by a mechanism, from the point of view of the so-called *quantitative information flow* framework. There have been various proposal for quantifying the information flow; we consider here the *information-theoretic approach*, in which the system (in this case the mechanism) is regarded as a *noisy channel*, and the leakage is defined as the difference between the a priori *entropy* of the input (the secret – in this case the database entries), and the a posteriori one, after revealing the output (in this case the reported answer). Depending on the notion of entropy adopted one can model different kinds of adversaries [9]. In particular, Shannon entropy (used, for instance, in [10–13]) is suitable for adversaries who can probe the secret repeatedly, while Rényi min-entropy (used, for instance, in [14, 15]) is suitable for one-try attacks. In both cases, the main difference with differential privacy is that the information-theoretic approaches measure the *expected* threat to confidentiality (i.e. the average amount of leakage, where each leak is weighted by its probability to occur), while differential privacy considers catastrophic any disclosure of confidential information, no matter how unlikely it is.

Computing and bounding the information leakage has been pursued in several papers, we mention for instance [16, 17]. Recently, researchers have investigated the relation between differential privacy and information leakage [18–20, 8], and in particular it has been proved in [20] that differential privacy induces a bound on the min-entropy leakage, which is met by a certain mechanism for the uniform prior (for which min-

entropy leakage is always maximum). In this paper, we extend the above result so to provide a more accurate bound for any fixed ϵ-regular prior distribution. More precisely, we provide a bound to the leakage specific to the prior and that can be met, under a certain condition, by a suitable mechanism. It is worth noting that this mechanism is defined similarly to the one that is optimal for the ϵ-regular priors. In fact, min-entropy leakage and utility are strongly related: the main difference is what we regard as the input of the channel. For the former is the database, for the latter the exact answer to the query. Correspondingly, min-entropy leakage measures the correlation between the reported answer and the database entries, while utility measures the correlation between the reported answer and the exact answer.

Contribution

- We identify, for an arbitrary query and a privacy parameter ϵ, the class of the ϵ-regular prior distributions on the exact answers. The interest of this class is that for each prior in it we are able to provide a specific upper bound to the utility of any ϵ-differentially-private mechanism. We characterise this class as a geometric region, and we study its properties.
- We describe an ϵ-differentially-private mechanism, called "tight-constraints mechanism", which meets those upper bounds for every ϵ-regular prior, and is therefore universally optimal in this region. We provide necessary and sufficient conditions for the existence of such mechanism, and an effective method to test the conditions and to construct the mechanism.
- Switching view, and considering the correlation between the databases and the reported answers (instead than between the exact and reported answers) we recast the above definitions and results in terms of quantitative information flow. The outcome is that we are able to improve the upper bounds for the min-entropy leakage of an ϵ-differentially-private mechanism, for all the ϵ-regular prior distributions on the databases. A construction similar to the one in previous point yields the tight-constraints mechanism which reaches those upper bounds.

Plan of the paper In the next section we recall the basic definitions of differential privacy and utility. Section 3 introduces the notion of ϵ-regular prior, investigates the properties of these priors, and gives a geometric characterisation of their region. Section 4 shows that for all ϵ-regular priors on the exact answers (resp. databases), ϵ-differential privacy induces an upper bound on the utility (resp. on the min-entropy leakage). Section 5 identifies a mechanism which reaches the above bounds for every ϵ-regular prior, and that is therefore the universally optimal mechanism (resp. the maximally leaking mechanism) in the region. Section 6 illustrates our methodology and results using the example of the sum queries. Section 7 concludes and proposes some directions for future research.

For reason of space we have omitted several proofs from the body of the paper. The interested reader can find them in the appendix.

2 Preliminaries

2.1 Differential privacy

The notion of ϵ-differential privacy, introduced by Dwork in [2], imposes constraints on data reporting mechanisms so that the user is unable to distinguish, from an output, between two databases differing only for one record. This indistinguishability property represents a protection for the individual corresponding to that record. In the following, the mechanism is represented as a probabilistic function $\mathcal{K}$ from the set of possible databases $\mathcal{V}$ to the set of possible reported outputs $\mathcal{O}$. The relation of 'differing only for one record' for two databases v and v' is represented by the *adjacency* relation and written as $v \sim v'$.

Definition 1 (Differential privacy [2]). *A probabilistic mechanism $\mathcal{K}$ from $\mathcal{V}$ to $\mathcal{O}$ satisfies ϵ-differential privacy if for all pairs $v, v' \in \mathcal{V}$, with $v \sim v'$, and all $S \subseteq \mathcal{O}$, it holds that*

$$P(\mathcal{K}(v) \in S) \leq e^{\epsilon} P(\mathcal{K}(v') \in S).$$

Note that the indistinguishability property is independent from the a priori knowledge the user may have about the database.

Consider a query $f : \mathcal{V} \to \mathcal{R}_f$, where $\mathcal{R}_f$ is the set of the query results. Then a mechanism $\mathcal{K}$ is said to be *oblivious* if for every database $v \in \mathcal{V}$, the output of the mechanism, $\mathcal{K}(v)$, depends only on $f(v)$, the result of applying the query to the database v, regardless of v itself. More formally,

Definition 2 ([1]). *Let $f : \mathcal{V} \to \mathcal{R}_f$ be a query. A mechanism $\mathcal{K} : \mathcal{V} \to \mathcal{O}$ is* oblivious *if there exists a randomised function $\mathcal{M} : \mathcal{R}_f \to \mathcal{O}$ such that, for all $v \in \mathcal{V}$, and all $S \subseteq \mathcal{O}$, it holds that*

$$P(\mathcal{K}(v) \in S) = P(\mathcal{M}(f(v)) \in S).$$

According to the above definition, any oblivious mechanism $\mathcal{K}$ can be seen as a cascade of two functions: the deterministic query f and a randomised function $\mathcal{M}$. The role of $\mathcal{M}$ is to add random noise to the exact query result $f(v)$ and produce a 'noisy' output $o \in \mathcal{O}$ to the user. The privacy guarantees are therefore provided by the function $\mathcal{M}$ which we implement by a stochastic matrix $X = (x_{io})$, called the *noise matrix*. The rows of X are indexed by the elements of $\mathcal{R}_f$ and the columns are indexed by the elements of $\mathcal{O}$. With this representation x_{io} is the probability of giving the output o when the exact query result is i. In this paper, we consider only oblivious mechanisms and therefore our results concern the design of the noise matrix X. Similarly, the query function f and the mechanism can be represented as matrices and hence it holds by Def. 2 that $\mathcal{K} = f\,X$.

Given a query f, The adjacency relation on databases $\mathcal{V}$ induces another adjacency relation on the set of query results $\mathcal{R}_f$ as follows.

Definition 3 (Adjacent query results). *Given a query function f with a range $\mathcal{R}_f$, two different results $i, h \in \mathcal{R}_f$ are said to be 'adjacent', and written as $i \sim_f h$, if and only if there exists two databases v, v' such that $f(v) = i$ and $f(v') = h$, and $v \sim v'$.*

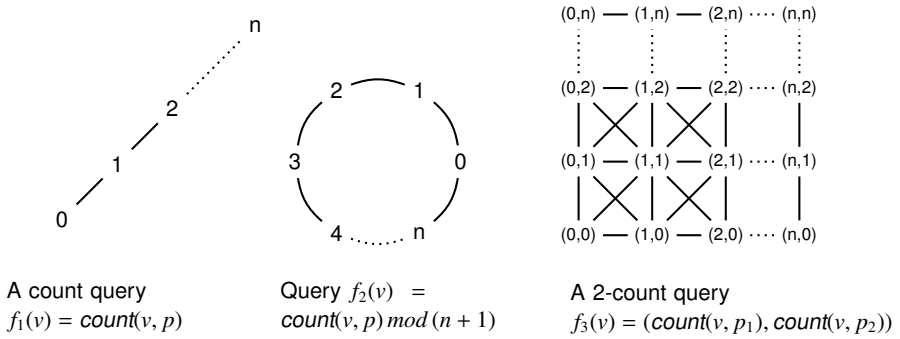


Fig. 1. Examples for the graph structures of different queries

Informally, $i, h \in \mathcal{R}_f$ are adjacent if they discriminate between two adjacent databases. Using the introduced notion of adjacency between query results, a graph structure can be used to model these results along with their adjacency relationship. More precisely, the set of nodes in this graph represents the set of query results $\mathcal{R}_f$, while edges represent the adjacency relationship among them. It is worth noting that this graph structure of queries have been used also in [8, 1] to analyse the differentially private mechanisms. Figure 1 shows examples of the graph structures of different queries. In these examples $count(v, p)$ refers to a counting query which returns the number of records in the database v which satisfy a certain property p. Other queries in the figure are expressed using the *count* function.

Let $\mathcal{K}$ be an oblivious mechanism for which X is the underlying noise matrix. It is intuitive to see that satisfying the indistinguishability between adjacent databases (i.e. satisfying differential privacy) corresponds to the satisfying indistinguishability (by means of X) between adjacent query results. Formally,

Lemma 1. *Given a noise matrix X, An oblivious mechanism $\mathcal{K}$ satisfies ϵ-differential privacy if and only if for all query results i, h where $i \sim_f h$ and all outputs $o \in \mathcal{O}$, it holds that $x_{io} \leq e^{\epsilon}\, x_{ho}$.*

Note that Lemma 1 provides an equivalent characterisation for differential privacy in terms of adjacent query results rather than adjacent databases.

With the graph structure of a query, the 'distance' between two query results i, h, denoted by $d(i, h)$ is defined as the shortest graph distance between i and h. Using this distance measure, differential privacy constraints can be further lifted from conditions on pairs of adjacent query results (Lemma 1) to a general condition on any pair of query results according to the following proposition.

Proposition 1. *Given a noise matrix X, the oblivious mechanism $\mathcal{K}$ satisfies ϵ-differential privacy if and only if for all query results i, h and all outputs $o \in \mathcal{O}$, it holds that $x_{io} \leq e^{\epsilon\, d(i,h)}\, x_{ho}$.*

That is, the ratio between the probability of reporting an answer o given that the query result is r and the probability of reporting the same output o given that the query result

is h does not exceed $e^{\epsilon\, d(i,h)}$. We call the noise matrix that satisfies this condition ϵ-differentially private. Note that, while Lemma 1 describes differential privacy in terms of only adjacent query results, the equivalent characterisation given by Proposition 1 specifies the privacy constraints imposed on any pair of results (whether or not they are adjacent to each other). This feature abstracts our analysis to arbitrary pairs of graph nodes rather than reasoning about only adjacent ones.

2.2 Utility model

For an oblivious mechanism $\mathcal{K}$, the objective of the underlying noise matrix X is to guarantee the differential privacy of the database, while providing the user with 'useful' information about the true query result. That is to satisfy a trade-off between the privacy and utility. For quantifying the utility of $\mathcal{K}$ we follow the model adopted in [6]. Given a query f, let $i \in \mathcal{R}_f$ be the result of executing f on some database. After processing i by the noise matrix X, let o be the reported output to the user. In practice, the user may use the output o, to 'guess' the value of the real query result. Therefore she may apply a *remap* (or guess) function which maps the mechanism output o to a guess $k \in \mathcal{R}_f$ for the exact query answer. The remap function (or simply 'remap') can be described as a stochastic matrix R, where its entry r_{ok} is the probability of guessing k when the observed mechanism output is o. With this representation, it can be easily seen that the probabilities of the user's guesses given individual query results are described by the matrix product XR. We say here that X is remapped to XR by the remap R. Note that this remapping procedure models the post-processing done by the user for the mechanism output o. Now, with the user's guessed value k, a real-valued *gain function* $g : (\mathcal{R}_f \times \mathcal{R}_f) \to \mathbb{R}$ quantifies how informative k is compared to the real result i.

The *utility* of a given mechanism to the user is described as the expected value of the gain function g. The evaluation of this expected value depends on the a priori probability distribution $\boldsymbol{\pi}$ over the real query results, which models the side knowledge of the user about the database. The utility of the mechanism depends therefore on the definition of the gain function g, the mechanism's underlying noise matrix X, the user's remap R, and also the probability distribution $\boldsymbol{\pi}$ over the real query results.

One choice for the gain function is the binary gain defined as $g_b(i, j) = 1$ iff $i = j$ and 0 otherwise. The binary gain function formalises the requirement of a user to guess the exact query result using the mechanism output. In the current work we restrict our analysis to this gain function. An important feature of this function, is that it is applicable to the ranges of various queries including numerical and non-numerical one. Moreover, it will be shown that this gain function is strongly connected to the information theoretic notions of conditional entropy and information leakage. Hence, our results about the utility of private mechanism imply corresponding results regarding quantifying information leaked by these mechanisms. These results go inline with a recent trend of research aiming at quantifying information leaked by security protocols, and privacy mechanisms specifically (see e.g. [16, 17, 8, 18]). We leave considering other gain functions to future work.

Now, for formulating the utility we represent the a priori probability distribution (called the 'prior') over the real query results by a row vector $\boldsymbol{\pi}$, indexed by $\mathcal{R}_f$, where

π_i is the probability that the query in hand yields the result i. The prior is therefore relative to the user and depends on her knowledge. With a generic gain function g, the utility of a mechanism for a prior $\boldsymbol{\pi}$ using the remap R is denoted by $\mathcal{U}(X, \boldsymbol{\pi}, R)$, and defined as follows.

$$\mathcal{U}(X, \boldsymbol{\pi}, R) = \mathbf{E}\left[g(i,k)\right] = \sum_{i,k} \pi_i \, (X\,R)_{ik} \;\; g(i,k), \tag{1}$$

where X is the noise matrix of the given mechanism. In our case, where the binary gain function g_b is used, the utility reduces to a convex combination of the diagonal elements of $X\,R$ as follows.

$$\mathcal{U}(X, \boldsymbol{\pi}, R) = \sum_i \pi_i \, (X\,R)_{ii}. \tag{2}$$

Accordingly, for a given prior $\boldsymbol{\pi}$, an oblivious ϵ-differentially private mechanism, with a noise matrix X, is said to be *optimal* if and only if there is a remap R such that the above function is maximised over all ϵ-differentially private mechanisms and all remaps [1]. As exemplified in the introduction, the optimality of a mechanism depends, in general, on the prior (user); that is a mechanism can be optimal for a prior while it is not for another one. It has been proved by [1] that for arbitrary queries (except the counting ones), there is no such a mechanism that is optimal for all priors simultaneously. Nevertheless, we identify in the following section a region of priors, where it is possible to find a single mechanism which is optimal to all of them.

3 ϵ-Regular priors

In this section we describe a region of priors, called 'ϵ-regular'. These priors are determined by the given query f and privacy parameter ϵ. In our way to specify these priors, we first represent the ϵ-differential privacy constraints in a matrix form. By Proposition 1, observe that each ϵ-differential privacy constraint imposed on a noise matrix X can be written as $x_{io}/x_{ho} \geq e^{-\epsilon d(i,h)}$. Since the lower bound $e^{-\epsilon d(i,h)}$ depends only on i, h, all constraints can be described altogether by a square matrix Φ formed by such lower bounds. We refer to this matrix as the *privacy-constraints* matrix. Note that the rows, and also columns of Φ are indexed by the elements of $\mathcal{R}_f$, the set of query results.

Definition 4 (privacy-constraints matrix). *The* privacy-constraints matrix Φ *of a query* f *with a range* $\mathcal{R}_f$*, and a privacy parameter* $\epsilon > 0$ *is a square matrix, indexed by* $\mathcal{R}_f \times \mathcal{R}_f$*, where* $\phi_{ih} = e^{-\epsilon\, d(i,h)}$ *for all* $i, h \in \mathcal{R}_f$*.*

Note that Φ is symmetric ($\phi_{ih} = \phi_{hi}$) due to the symmetry of the distance function $d(i,h)$. Observe that when $\epsilon \to \infty$, i.e. exclude privacy at all, Φ converges to the identity matrix where each diagonal entry is 1 and other entries are zeros. In terms of the privacy-constraints matrix of a query and ϵ, we define now the ϵ-regular priors as follows (note that we use $\boldsymbol{y} \geq 0$ to denote $\forall i : y_i \geq 0$).

Definition 5 (ϵ-regular prior). *For a given query* f *and a privacy parameter* $\epsilon > 0$*, a prior* $\boldsymbol{\pi}$ *is called* ϵ-regular *iff there exists a row vector* $\boldsymbol{y} \geq 0$ *such that* $\boldsymbol{\pi} = \boldsymbol{y}\,\Phi$*.*

[1] Note that there may exist many optimal mechanism for a given prior.

In the following we describe the common properties of these priors and also give a geometric characterisation for their region comparing it to the whole prior space. As the first observation, note that, as privacy is excluded ($\epsilon \to \infty$), this region converges to the entire prior space. This is because Φ approaches the identity matrix where the vector $\boldsymbol{y}$ exists for each prior.

An important property of any ϵ-regular prior is that the ratio between any two of its entries π_i, π_j is always bound as follows, depending on ϵ and the distance $d(i, j)$. Because of this property, such a prior is called ϵ-regular.

Proposition 2. *Consider a query f and $\epsilon > 0$. Then for any ϵ-regular prior $\boldsymbol{\pi}$, it holds for all $i, j \in \mathcal{R}_f$: $\pi_i/\pi_j \leq e^{\epsilon\, d(i,j)}$.*

While the above property restricts the ratio between probabilities of adjacent query results, this restriction, in practice, holds for a large class of users who have no sharp information suggesting discrimination between adjacent results. This class is exemplified in the introduction. Note that the above property is not equivalent to Definition 5. Namely, it is not true that all priors having such a property are ϵ-regular.

A consequence of the above proposition is that for any ϵ-regular prior $\boldsymbol{\pi}$, the probability π_i associated with any query result i is restricted by upper and lower bounds as follows.

Proposition 3. *Consider a query f and $\epsilon > 0$. Then for any ϵ-regular prior $\boldsymbol{\pi}$, it holds for all $i \in \mathcal{R}_f$ that*

$$1 \Big/ \sum_{j \in \mathcal{R}_f} e^{\epsilon\, d(i,j)} \leq \pi_i \leq 1 \Big/ \sum_{j \in \mathcal{R}_f} e^{-\epsilon\, d(i,j)}.$$

One implication is that any ϵ-regular prior must have full support, that is $\pi_i > 0$ for all $i \in \mathcal{R}_f$.

In the following we go further and describe the region of ϵ-regular priors as a region of points in the prior space, where each point represents a member in this region. For doing so, we identify by the following definition a set of priors which describe the 'corner points' or vertices of the region.

Definition 6 (corner priors). *Given a query f and a privacy parameter $\epsilon > 0$, then for each query result $i \in \mathcal{R}_f$, a corresponding corner prior, denoted by $\boldsymbol{c}^i$, is defined as*

$$c^i_j = \frac{\phi_{ij}}{\sum_{k \in \mathcal{R}_f} \phi_{ik}} \qquad \forall\, j \in \mathcal{R}_f.$$

Note that the above definition is sound, i.e. $\boldsymbol{c}^i$ is a probability distribution. By the above definition, for a given query with the domain $\mathcal{R}_f$ of results, the region of ϵ-regular priors has $|\mathcal{R}_f|$ corner priors. Each one corresponds to a query result $i \in \mathcal{R}_f$. Note that each corner prior $\boldsymbol{c}^i$ is maximally biased (relative to the region) to the query result i; that is the entry c^i_i meets its maximum value given in Proposition 3. It can be seen that each corner prior is ϵ-regular. Namely for any corner $\boldsymbol{c}^i$, define the vector $\boldsymbol{y}$ as $y_i = 1/\sum_{k \in \mathcal{R}_f} \phi_{ik}$ and $y_j = 0$ for all $j \neq i$; thus it holds that $\boldsymbol{c}^i = \boldsymbol{y}\, \Phi$.

The region of the ϵ-regular priors can be characterised in terms of the corner priors. More precisely, this region consists of all priors that can be composed as a convex combination of the corner priors.

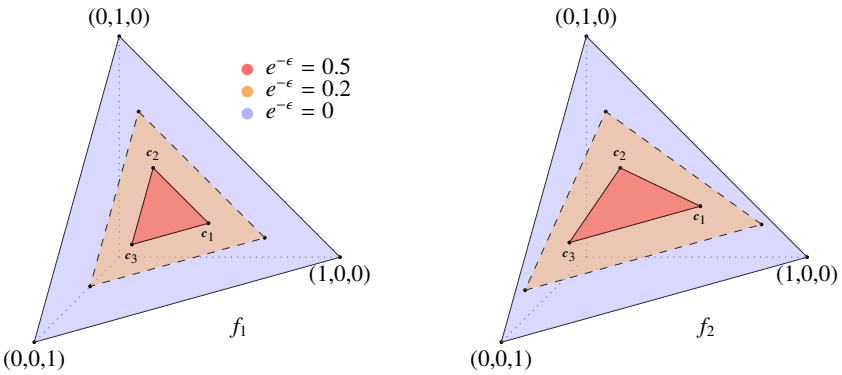


Fig. 2. Regions of ϵ-regular priors for queries described in Example 1

Proposition 4 (convexity). *For a given a query f and privacy parameter $\epsilon > 0$, a prior $\boldsymbol{\pi}$ is ϵ-regular iff there exist real numbers $\gamma_i \geq 0$, $i \in \mathcal{R}_f$ such that*

$$\boldsymbol{\pi} = \sum_{i \in \mathcal{R}_f} \gamma_i \, \boldsymbol{c^i}.$$

It is easy to see that it must hold that $\sum_{i \in \mathcal{R}_f} \gamma_i = 1$ for any ϵ-regular prior. This is obtained by summing the components of the $\boldsymbol{\pi}$ as follows.

$$\sum_{j \in \mathcal{R}_f} \pi_j = \sum_i \gamma_i \sum_j c^i_j \quad \text{and} \quad \sum_j \pi_j = 1, \; \forall \boldsymbol{\pi}.$$

From Proposition 4 and the above observation, the region of ϵ-regular priors is a convex set, where each point (prior) in this region is a convex combination of the corner priors. This region is therefore geometrically regarded as a convex polytope in the prior space. Since the corner points always exists, this region is never empty.

For a prior $\boldsymbol{\pi}$ in this region, the coefficients γ_i model the 'proximity' of $\boldsymbol{\pi}$ to each corner prior $\boldsymbol{c^i}$. Observe that $0 \leq \gamma_i \leq 1$, and $\gamma_i = 1$ iff $\boldsymbol{\pi} = \boldsymbol{c^i}$. We demonstrate this geometric interpretation using the following examples.

Example 1. Priors having 3 entries can be represented as points in the 3-dimensional euclidean space. These priors correspond to queries whose graph structures contain 3 nodes. These nodes can be arranged in either a sequence or a cycle, corresponding to queries f_1 and f_2 respectively shown in Figure 1, with $n = 2$ in both cases. Figure 2 shows - for each of these queries - the region of ϵ-regular priors. The corner priors of each region are represented by points $\boldsymbol{c^1}, \boldsymbol{c^2}, \boldsymbol{c^3}$. For each query in Fig. 2, we depict the regions for $e^{-\epsilon} = 0.5$ and $e^{-\epsilon} = 0.2$. Note that the level of privacy set by ϵ imposes a restriction on the region of ϵ-regular priors. With $e^{-\epsilon} = 0.2$ (less privacy), this region is larger than the one with $e^{-\epsilon} = 0.5$. In fact, as $e^{-\epsilon} \to 0$ (i.e. no privacy), the region of ϵ-regular priors converges to the entire region of priors defined by the corner points $\{(0,0,1),(0,1,0),(0,0,1)\}$.

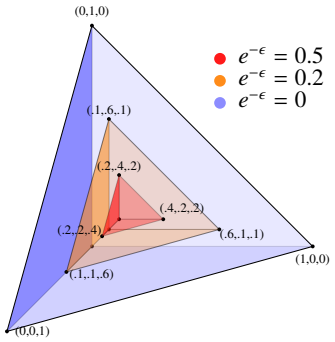


Fig. 3. Regions of ϵ-regular priors for the query described in Example 2

Example 2. Let v be database containing at most one record. Consider a bundle of two counting queries $f_3 = (count(v, p_1), count(v, p_2))$ which counts the records satisfying properties p_1 and p_2 respectively in the database v. The graph structure of this query is depicted in Figure 1 (with $n = 1$). Note that in this case the adjacency graph (and also the set $\mathcal{R}_f$ of query results) consists of 4 nodes: $\{(0,0),(1,0),(0,1),(1,1)\}$. Any prior $\boldsymbol{\pi}$ corresponds therefore to a point in a 4-dimensional space. However, since the 4th component of the prior is redundant ($\sum_i \pi_i = 1$), each prior is defined by its 'projection' onto the 3- dimensional subspace. Given this observation, Figure 3 shows the projection of the ϵ-regular prior region for different values of $e^{-\epsilon}$. It is again seen that the region is getting larger as the level of privacy $e^{-\epsilon}$ decreases, and coincides with the full space of priors when $e^{-\epsilon} \to 0$ (i.e. when no privacy is provided).

4 Upper bounds for utility and min-mutual information

In this section, we further describe the ϵ-regular priors in terms of the utility that can be achieved for these priors by ϵ-differentially private mechanisms. We also describe the amount of information that can be conveyed by these mechanisms to users with such priors. More precisely, we identify for any ϵ-regular prior $\boldsymbol{\pi}$ upper bounds for the utility and min-mutual information, considering all ϵ-differentially private mechanisms and all possible remaps. These bounds are indeed induced by the privacy constraints parameterised by ϵ and the query f as stated by Proposition 1. They also depend on the given prior $\boldsymbol{\pi}$.

4.1 Utility

Given a query f and a privacy parameter $\epsilon > 0$, let $\boldsymbol{\pi}$ be a prior on the set R_f of the query results. For any noise matrix X satisfying ϵ-differential privacy (as in Proposition 1), and a remap R, we derive in the following a linear algebraic expression for $\mathcal{U}(X, \boldsymbol{\pi}, R)$, the utility of X for $\boldsymbol{\pi}$ using the remap R. Such an expression will play the main role in the

subsequent results. We start by observing that the matrix product of the noise matrix X and the remap R describes an ϵ-differentially private noise matrix $XR : \mathcal{R}_f \rightarrow \mathcal{R}_f$. Hence the entries of XR satisfy (by Proposition 1) the following subset of constraints.

$$e^{-\epsilon\, d(i,k)}\,(XR)_{kk} \;\leq\; (XR)_{ik}$$

for all $i, k \in \mathcal{R}_f$. Using Definition 4 of the privacy-constraints matrix Φ, and taking into account that $\sum_{k\in\mathcal{R}_f}(XR)_{ik} = 1$ for all i (as both X and R are stochastic), we imply the following inequalities.

$$\textstyle\sum_{k\in\mathcal{R}_f} \phi_{ik}\,(XR)_{kk} \leq 1, \quad \forall i \in \mathcal{R}_f.$$

The inequality operators can be replaced by equalities while introducing *slack* variables $0 \leq s_i \leq 1$ for all $i \in \mathcal{R}_f$. The above inequalities can therefore be written as follows.

$$\textstyle\sum_{k\in\mathcal{R}_f} \phi_{ik}\,(XR)_{kk} + s_i = 1, \quad \forall i \in \mathcal{R}_f.$$

Let the slack variables s_i form a column vector $\boldsymbol{s}$ indexed by $\mathcal{R}_f$. Let also $\mathbf{1}$ denote another column vector of the same size having all entries equal to 1. Using these vectors and the privacy-constraints matrix Φ (for the given query and ϵ), the above equations can be rewritten in the following matrix form.

$$\Phi\ \operatorname{diag}(XR) + \boldsymbol{s} = \mathbf{1}, \tag{3}$$

where $\operatorname{diag}(XR)$ is the column vector consisting of the diagonal entries of XR. Now, for any noise matrix $X : \mathcal{R}_f \rightarrow \mathcal{O}$ and a remap $R : \mathcal{O} \rightarrow \mathcal{R}_f$ satisfying Eq. (3), and for a prior $\boldsymbol{\pi}$, we want to refine the generic expression (2) of the utility by taking Eq. (3) into account. We start by rewriting Eq. (2) in the following matrix form.

$$\mathcal{U}(X, \boldsymbol{\pi}, R) = \boldsymbol{\pi}\ \operatorname{diag}(XR). \tag{4}$$

Now, let $\boldsymbol{y}$ be a row vector such that

$$\boldsymbol{\pi} = \boldsymbol{y}\,\Phi. \tag{5}$$

Note that, the above matrix equation is in fact a system of $|\mathcal{R}_f|$ linear equations. The kth equation in this system is formed by the kth column of Φ, and the kth entry of $\boldsymbol{\pi}$ as follows.

$$\boldsymbol{y}\,\Phi_k = \pi_k \qquad \forall k \in \mathcal{R}_f.$$

Solving this system of equations for the row vector $\boldsymbol{y}$ has the following possible outcomes: If the matrix Φ is invertible, then, for any prior $\boldsymbol{\pi}$, Eq. (5) has exactly one solution. If Φ is not invertible (i.e. it contains linearly dependent columns), then there are either 0 or an infinite number of solutions, depending on the prior $\boldsymbol{\pi}$: If the entries of $\boldsymbol{\pi}$ respect the linear dependence relation then are infinitely many solutions. Otherwise, the equations are '*inconsistent*', in which case there are no solutions.

Since the matrices Φ have a precise format, one may wonder whether it could be that they are all invertible or all non invertible. In fact, this is not the case: In Appendix B

we show an example of a matrix Φ that, for certain values of ϵ is invertible, while for others is non invertible.

Whether Φ is invertible or not, we consider here only the priors where the matrix equation (5) has at least one solution $\boldsymbol{y}$. Note that, by definition, all the ϵ-regular priors have this property, but there can be others for which the solution $\boldsymbol{y}$ has some negative components. In some of the results below (in particular in Lemma 2) we consider this larger class of priors, for the sake of generality.

Multiplying Equation (3) by $\boldsymbol{y}$ yields

$$\boldsymbol{y}\,\Phi\,\mathrm{diag}(X\,R) + \boldsymbol{y}\;\boldsymbol{s} = \boldsymbol{y}\,\mathbf{1}. \tag{6}$$

Substituting Equations (5) and (4) in the above equation consecutively provides the required expression for the utility and therefore proves the following lemma.

Lemma 2. *For a given query f and a privacy parameter $\epsilon > 0$, let $\boldsymbol{\pi}$ be any prior. Then for every row vector $\boldsymbol{y}$ satisfying $\boldsymbol{\pi} = \boldsymbol{y}\,\Phi$, the* utility *of any ϵ-differentially private mechanism with a noise matrix X for the prior $\boldsymbol{\pi}$ using a remap R is given by*

$$\mathcal{U}(X, \boldsymbol{\pi}, R) = \boldsymbol{y}\,\mathbf{1} - \boldsymbol{y}\,\boldsymbol{s}, \tag{7}$$

for a vector $\boldsymbol{s}$ satisfying $0 \le s_i \le 1$ for all $i \in \mathcal{R}_f$.

Lemma 2 expresses the utility function for any ϵ-private noise matrix X for a prior $\boldsymbol{\pi}$ with a remap R as a function of the vector $\boldsymbol{y}$ and the slack vector $\boldsymbol{s}$. Although the matrix X and the remap R do not explicitly appear on the right hand side of Equation (7), the utility still depends on them indirectly through the vector $\boldsymbol{s}$. Namely, according to Equation (3), the choice of X and R determines the slack vector $\boldsymbol{s}$. The utility function depends also on the prior $\boldsymbol{\pi}$, because the choice of $\boldsymbol{\pi}$ determines the set of vectors satisfying Eq. (5). Substituting any of these vectors $\boldsymbol{y}$ in Eq. (7) yields the same value for $\mathcal{U}(X, \boldsymbol{\pi}, R)$.

By Definition 5, of ϵ-regular priors, the above lemma specifies the utility for any of them. Therefore, we use Lemma 2, and obtain an upper bound for the utility of ϵ-differentially private mechanisms for ϵ-regular priors.

Theorem 1 (utility upper bound). *For a given query f and a privacy parameter $\epsilon > 0$, let $\boldsymbol{\pi}$ be an ϵ-regular prior and X be an ϵ-differentially private noise matrix. Then for all row vectors $\boldsymbol{y} \ge 0$ satisfying $\boldsymbol{y}\,\Phi = \boldsymbol{\pi}$, it holds for any remap R that*

$$\mathcal{U}(X, \boldsymbol{\pi}, R) \le \textstyle\sum_{i \in \mathcal{R}_f} y_i, \tag{8}$$

where the equality holds iff $\Phi\ \mathrm{diag}(X\,R) = \mathbf{1}$.

The above result can be also seen from the geometric perspective. As shown by Proposition 4, each member in the region of ϵ-regular priors is described as a convex combination of the corner priors. That is there are coefficients $\gamma_i \ge 0$ for $i \in \mathcal{R}$ which form this combination. It can be shown (as in the proof of Proposition 4) that $\gamma_i = y_i \left(\sum_{k \in \mathcal{R}_f} \phi_{ik}\right)$. Hence, the upper bound given by Theorem 1 can be written as follows using the coefficients γ_i.

$$\mathcal{U}(X, \boldsymbol{\pi}, R) \le \sum_{i \in \mathcal{R}_f} \frac{\gamma_i}{\sum_{k \in \mathcal{R}_f} \phi_{ik}}.$$

Inspecting the above result for corner priors, recall that for a corner $\boldsymbol{c}^i$, γ_j is 1 for $j = i$ and is 0 otherwise; thus, the utility upper bound for $\boldsymbol{c}^i$ is therefore $1/\sum_k \phi_{ik}$. Moreover, the upper bound for each ϵ-regular prior $\boldsymbol{\pi}$ can be regarded (according to the above equation) as a convex combination of the upper bounds for the corner priors. That is, from the geometric perspective, the utility upper bound for $\boldsymbol{\pi}$ linearly depends on its proximity to the corner priors.

4.2 Min-mutual information

In this section, we employ an information-theoretic notion, namely mutual information, to quantify the amount of information conveyed by a noise matrix X as an information theoretic channel. We use this notion in two distinct ways: first, mutual information is used to measure the information conveyed about the result of a specific query, similarly to the use of "utility" in the previous section. Mutual information and utility (under the binary gain function) are closely related, which allows us to transfer the bound obtained in the previous section to the information-theoretic setting.

Second, we use mutual information to quantify the information *about the database* that is revealed by a mechanism, a concept known in the area of quantitative information flow as "information leakage". This allows us to obtain bounds on the information leaked by any mechanism, even non-oblivious ones, independently from the actual query. For arbitrary priors, we obtain in a more natural way the bound conjectured in [18] and proven in [8]. Moreover, if we restrict to specific (ϵ-regular) priors, then we are able to provide more accurate bounds.

Following recent works in the are of quantitative information flow ([14–17, 8, 18]), we adopt *Rényi's min-entropy* ([21]) as our measure of uncertainly. The min-entropy $\mathcal{H}_\infty(\boldsymbol{\pi})$ of a prior $\boldsymbol{\pi}$, defined as $\mathcal{H}_\infty(\boldsymbol{\pi}) = -\log_2 \max_i \pi_i$, measures the user's uncertainty about the query result. Then, the corresponding notion of *conditional* min-entropy, defined as $\mathcal{H}_\infty(X, \boldsymbol{\pi}) = -\log_2 \sum_o \max_i \pi_i\, x_{io}$, measures the uncertainty about the query result after observing the output of the noise matrix X. Finally, subtracting the latter from the former brings us to the notion of min-mutual information:

$$\mathcal{L}(X, \boldsymbol{\pi}) = \mathcal{H}_\infty(\boldsymbol{\pi}) - \mathcal{H}_\infty(X, \boldsymbol{\pi})$$

which measures the amount of information about the query result conveyed by the noise matrix. In the area of quantitative information flow this quantity is known as *min-entropy leakage*; the reader is referred to [14] for more details about this notion.

Min-mutual information is closely related to the notion of utility under the binary gain function and using an *optimal* remap. A remap $\hat{R}$ is optimal for $X, \boldsymbol{\pi}$ if it gives the best utility among all possible remaps for this noise matrix and prior. The following result from [8] connects min-mutual information and utility:

Proposition 5. *Given a noise matrix X and a prior $\boldsymbol{\pi}$, let $\hat{R}$ be an optimal remap for $\boldsymbol{\pi}, X$. Then, it holds*

$$\mathcal{L}(X, \boldsymbol{\pi}) = \log_2 \frac{\mathcal{U}(X, \boldsymbol{\pi}, \hat{R})}{\max_i \pi_i}$$

This connection allows us to transfer the upper-bound given by Theorem 1 to min-mutual information.

Proposition 6 (min-mutual information upper bound). *Let f be a query, let $\epsilon > 0$, let $\boldsymbol{\pi}$ be an ϵ-regular prior and let X be the noise matrix of any ϵ-differentially private mechanism. Then for all row vectors $\boldsymbol{y} \geq 0$ satisfying $\boldsymbol{y}\,\Phi = \boldsymbol{\pi}$, it holds that:*

$$\mathcal{L}(X, \boldsymbol{\pi}) \leq \log_2 \frac{\sum_{i \in \mathcal{R}_f} y_i}{\max_i \pi_i} \tag{9}$$

The above bound holds only for ϵ-regular priors. However, it is well-known ([15]) that min-mutual information is maximised by the uniform prior $\boldsymbol{u}$, i.e. $\mathcal{L}(X, \boldsymbol{\pi}) \leq \mathcal{L}(X, \boldsymbol{u})$ for all $X, \boldsymbol{\pi}$. Thus, in cases when $\boldsymbol{u}$ is ϵ-regular, we can extend the above bound to *any* prior.

Corollary 1. *Let f be a query, let $\epsilon > 0$ such that the uniform prior $\boldsymbol{u}$ is ϵ-regular, and let X be the noise matrix of any ϵ-differentially private mechanism. Then for all row vectors $\boldsymbol{y} \geq 0$ satisfying $\boldsymbol{y}\,\Phi = \boldsymbol{u}$, and for all priors $\boldsymbol{\pi}$, it holds that:*

$$\mathcal{L}(X, \boldsymbol{\pi}) \leq \log_2(|\mathcal{R}_f| \sum_{i \in \mathcal{R}_f} y_i)$$

4.3 Quantifying the leakage about the database

In the previous section we considered the information about the query result conveyed by an oblivious mechanism. This information was measured by the min-mutual information $\mathcal{L}_(X, \boldsymbol{\pi})$, where X is noise matrix, mapping query results $\mathcal{R}_f$ to outputs.

We now turn our attention to quantifying the information about the *database* that is conveyed by the complete mechanism $\mathcal{K}$ (even in the case of non-oblivious mechanisms). Intuitively, we wish to minimise this information to protect the privacy of the users, contrary to the utility which we aim at maximising. Quantifying this information can be done in a way very similar to the previous section. The only difference is that we use a stochastic matrix Y that models the mechanism $\mathcal{K}$, mapping databases $\mathcal{V} = V^u$ to outputs (recall that u is the number of individuals in the database and V the set of possible values for each individual). Moreover, the underlying graph $\sim$ is the *Hamming graph*, induced by the adjacency relation on databases, and ϵ-regularity concerns priors $\boldsymbol{\pi}$ on databases.

In this case, $\mathcal{L}(Y, \boldsymbol{\pi})$ measures the information about the database conveyed by the mechanism, which we refer to as "min-entropy leakage", and the bounds from the previous section can be directly applied. However, since we now work on a specific graph $(\mathcal{V}, \sim)$, we can obtain a closed expression for the bound of Corollary 1. We start by observing that due to the symmetry of the graph, the uniform prior $\boldsymbol{u}$ is ϵ-regular for all $\epsilon > 0$. More precisely, we can show that the vector $\boldsymbol{y}$, defined as

$$y_i = \left(\frac{e^\epsilon}{|V|(|V| - 1 + e^\epsilon)} \right)^u \qquad i \in \mathcal{V}$$

satisfies $\boldsymbol{y}\Phi = \boldsymbol{u}$ and $\boldsymbol{y} \geq 0$. Thus, applying Corollary 1 we get the following result.

Theorem 2 (min-entropy leakage upper bound). *Let $\mathcal{V} = V^u$ be a set of databases, let $\epsilon > 0$, and let Y be an ϵ-differentially private mechanism. Then for all priors $\boldsymbol{\pi}$, it holds that:*

$$\mathcal{L}(Y, \boldsymbol{\pi}) \leq u \log_2 \frac{|V|\, e^\epsilon}{|V| - 1 + e^\epsilon}$$

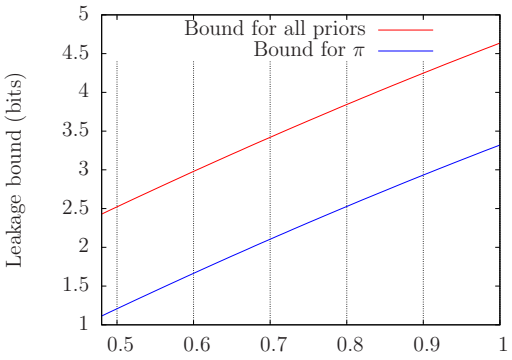


Fig. 4. Leakage bounds for various values of ϵ

This bound determines the maximum amount of information that *any* differentially privacy mechanism can leak about the database (independently from the underlying query). The bound was first conjectured in [18] and independently proven in [8]; our technique gives an alternative an arguably more intuitive proof of this result.

Note that the above bound holds for *all priors*. If we restrict to a *specific ϵ-regular prior* $\boldsymbol{\pi}$, then we can get better results by using the bound of Proposition 6 which depends on the actual prior. This is demonstrated in the following example.

Example 3. Consider a database of 5 individuals, each having one of 4 possible values, i.e. $\mathcal{V} = V^u$ with $V = \{1, 2, 3, 4\}$ and $u = 5$. Assume that each individual selects a value independently from the others, but not all values are equally probable; in particular the probabilities of values $1, 2, 3, 4$ are $0.3, 0.27, 0.23, 0.2$ respectively. Let $\boldsymbol{\pi}$ be the corresponding prior on $\mathcal{V}$ that models this information. We have numerically verified that for all $0.48 \leq \epsilon \leq 1$ (with step 0.01) $\boldsymbol{\pi}$ is ϵ-regular. Thus we can apply Proposition 6 to get an upper bound of $\mathcal{L}(Y, \boldsymbol{\pi})$ for this prior.

The resulting bound, together with the general bound for all priors from Theorem 2, are shown in Figure 4. We see that restricting to a specific prior provides a significantly better bound for all values of ϵ. For instance, for $\epsilon = 0.5$ we get that $\mathcal{L}(Y, \boldsymbol{\pi}) \leq 1.2$ for this $\boldsymbol{\pi}$, while $\mathcal{L}(Y, \boldsymbol{\pi}) \leq 2.5$ for all priors $\boldsymbol{\pi}$.

Note that, in general, the above bounds for the utility and the min-mutual information are not tight. For a given query and a privacy parameter ϵ, there may be no noise matrix X that meets these bounds. Nevertheless, they provide ultimate limits, induced by the privacy constraints, for all ϵ-differential private mechanisms and ϵ-regular priors. Note also that these bounds are simultaneously tight if the *common* condition $\Phi\ \mathrm{diag}(XR) = \mathbf{1}$ is satisfied (note that this condition is independent of the underlying prior). From this point we investigate the mechanisms that, whenever exist, they satisfy such a condition and are therefore optimal for the entire class of ϵ-regular priors.

5 Tight-constraints mechanisms

In this section, we introduce the notion of *tight-constraints* mechanism. We start by giving the definition of these mechanisms for a given query f and privacy parameter $\epsilon > 0$. Then we describe their properties in terms of privacy guarantees and optimality for ϵ-regular priors.

Definition 7 (A tight-constraints mechanism). *For a given query f with range $\mathcal{R}_f$, and a given privacy parameter $\epsilon > 0$, an oblivious mechanism with a noise matrix $X : \mathcal{R}_f \to \mathcal{R}_f$ is called a* tight-constraints *mechanism iff it satisfies the following conditions for all $i, k \in \mathcal{R}_f$.*

$$e^{-\epsilon\, d(i,k)}\, x_{kk} \;=\; x_{ik}. \tag{10}$$

It is important to note that, in general, there may exist zero, one or more tight-constraints mechanisms for a given query f and a privacy parameter $\epsilon > 0$. The above definition enforces $|\mathcal{R}_f|\,(|\mathcal{R}_f| - 1)$ linearly independent equations, referred to as the '*tight constraints*'. Additionally it must also hold that $\sum_{k \in \mathcal{R}_f} x_{ik} = 1$ for all $i \in \mathcal{R}_f$. Thus we have, in total, $|\mathcal{R}_f|\,|\mathcal{R}_f|$ equations. If these equations are linearly independent, then they solve to unique values. If these values are non-negative, then they determine a *unique* tight-constraints mechanism. On the other hand, if these equations are not linearly independent, then there may be multiple solutions with non-negative entries, in which case we have multiple tight-constraints mechanisms for the given query and privacy parameter ϵ.

5.1 Properties

It has been seen from Definition 7, that the choice of a query f and a value $\epsilon > 0$ correspond to a set of tight-constraints mechanisms. The first important feature of these mechanisms is that they satisfy ϵ-differential privacy as confirmed by the following proposition.

Proposition 7 (differential privacy). *For a given query f and a privacy parameter $\epsilon > 0$, every tight-constraints mechanism is ϵ-differentially private.*

Thanks to the above fact, we can give a further useful characterisation of the tight-constraints mechanisms in comparison to other ϵ-differentially private mechanisms. More precisely, the following proposition identifies a linear algebraic condition that is satisfied *only by* the tight-constraints mechanisms for given f, ϵ:

Lemma 3 (diagonal characterisation). *Let f be a query and let $\epsilon > 0$. Then for any oblivious ϵ-differentially private mechanism $\mathcal{K}$ with a noise matrix $X : \mathcal{R}_f \to \mathcal{R}_f$, the following equation holds iff $\mathcal{K}$ is a tight-constraints mechanism.*

$$\Phi \;\operatorname{diag}(X) = \mathbf{1}. \tag{11}$$

Observe that the above proposition provides a way to check the existence of, and also compute, the tight-constraints mechanisms for given f, ϵ. Since Condition (11) is satisfied only by these mechanisms, then there is at least one tight-constraints mechanism if

there is a vector z, with non-negative entries, that satisfies the equation $\Phi\ z = \mathbf{1}$. In this case the noise matrix $\hat{X}$ of a tight-constraints mechanism is obtained by setting its diagonal to z, and evaluating the non-diagonal entries from the diagonal using Equations (10).

Now we turn our attention to the region of ϵ-regular priors and we identify the oblivious mechanisms which are optimal wrt both utility and min-mutual information in this region. It turns out that the set of these optimal mechanisms consists exactly of all the mechanisms that can be *mapped to* a tight-constraints mechanism using a remap R.

Theorem 3 (Optimality). *Let f be a query and let $\epsilon > 0$ such that at least one tight-constraints mechanism exists. Then any oblivious mechanism $\mathcal{K} : \mathcal{V} \to \mathcal{O}$ is optimal (wrt both utility and min-mutual information) for every ϵ-regular prior $\boldsymbol{\pi}$ iff there is a remap $R : \mathcal{O} \to \mathcal{R}_f$ such that $\mathcal{K}R$ is a tight-constraints mechanism for f, ϵ.*

Proof. If there exists a tight-constraints mechanism for given f, ϵ, then its noise matrix $\hat{X}$ must satisfy Eq (11). This implies that the upper-bound in Theorem 1 is reachable by $\hat{X}$ and the identity remap. Thus that upper-bound, in this case, is tight. By Theorem 1, a mechanism $\mathcal{K}$ with a noise matrix X meets such an upper bound for the utility (and therefore is optimal) iff it satisfies the condition $\Phi\,\mathrm{diag}(XR) = \mathbf{1}$, with some remap R. Since any mechanism with noise matrix XR is ϵ-differentially private, then by Lemma 3, this condition is satisfied iff XR is the noise matrix of a tight-constraints mechanism (for f, ϵ). That is iff $fXR = \mathcal{K}R$ is a tight-constraints mechanism. Using the relation, given by Proposition 5, between utility and min-mutual information, the same argument holds for the latter. □

Note that tight-constrains mechanisms are themselves optimal as they are mapped to themselves by the identity remap.

As a consequence of the above general result, we consider the special case of the uniform prior, denoted by $\boldsymbol{u}$, where all exact query results in $\mathcal{R}_f$ are equally likely. Note that this prior corresponds to users having unbiased knowledge about the query results, i.e. they assume that all the exact results $\mathcal{R}_f$ are yielded, by executing the query, with the same probability. Firstly, the following lemma proves an equivalence between the existence of at least one tight-constraints mechanism on one hand and the uniform prior $\boldsymbol{u}$ being ϵ-regular on the other hand.

Lemma 4. *For a given query f and privacy parameter $\epsilon > 0$, there exists at least one tight-constraints mechanism iff the uniform prior $\boldsymbol{u}$ is ϵ-regular.*

It is worth noticing that in general the region of ϵ-regular priors may or may not include the uniform prior. However, as shown earlier in Section 3, this region is enlarged and converges to the entire prior space as less privacy is imposed (that is as ϵ increases). This means that for the values of ϵ above certain threshold ϵ^*, depending on the query, the region of ϵ-regular priors accommodates the uniform prior $\boldsymbol{u}$, and therefore (by Lemma 4), there is at least one tight-constraints mechanism. This provides a design criteria to *select* a setting for ϵ such that we have an optimal mechanism for the whole region.

Using Lemma 4, we can describe the optimal mechanisms for the uniform prior as a corollary of Theorem 3.

(a) Sum query (b) 2-count query

Fig. 5. Adjacency graphs

Corollary 2. *Let f be a query and let $\epsilon > 0$ such that there exists at least one tight-constrains mechanism. Then a mechanism $\mathcal{K} : \mathcal{V} \to \mathcal{O}$ is optimal for the uniform prior iff $\mathcal{K}R$ is a tight-constraints mechanism for some remap $R : \mathcal{O} \to \mathcal{R}_f$.*

In fact when we consider arbitrary queries, we find that our specification for the tight-constraints mechanisms covers other well known differentially-private mechanisms. In particular, when we consider a counting query, we find that a tight-constraints mechanism for this query is exactly the *truncated-geometric mechanism*, which is shown by [6] to be optimal for every prior. Furthermore, we are able to show that this mechanism, as a tight-constraints one, exists for the selected query with any $\epsilon > 0$.

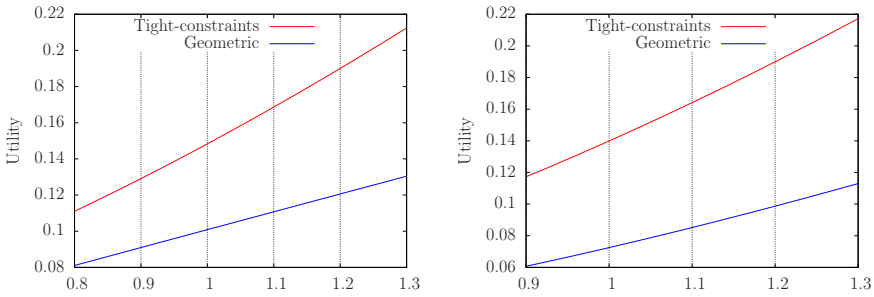
Another class of queries, studied in [8] are the ones whose graph structures are either *vertex-transitive* or *distance-regular*. The authors in [8] were able to construct a mechanism which is optimal for the uniform prior for any $\epsilon > 0$. In the context of our results, when we consider a query f in this class, it is easy to show that such an optimal mechanism is in fact a tight-constraints mechanism for f. We can also show that this tight-constraints mechanism exists for all $\epsilon > 0$.

6 Case-study: sum and 2-count queries

In this section we show the usefulness of the tight-constraints mechanism by applying it to two particular families of queries, namely sum and 2-count queries. For each family, we evaluate the tight-constraint mechanism on databases consisting of u individuals each having an integer value between 0 and v, and we compare its utility to the geometric mechanism.

It is well-known that no universally optimal mechanism exists for these families; in particular, the geometric mechanism, known to be optimal for a single counting query, is not guaranteed to be optimal for sum or multiple counting queries. On the other hand, as discussed in the previous section, tight-constraints mechanisms, whenever they exist, are guaranteed to be optimal within the region of ϵ-regular priors.

The comparison is made as follows: for each query, we numerically compute the smallest ϵ (using a step of 0.01) for which a tight-constraints mechanism exists (i.e. for which the uniform prior $\boldsymbol{u}$ is ϵ-regular, see Lemma 4). Then we compute the utility (using an optimal remap) of both the tight constraints and the geometric mechanisms,



(a) Sum query (b) 2-count query

Fig. 6. Utility for various values of ϵ

for a range of ϵ starting from the minimum one (note that the tight constraint mechanism exists for any ϵ greater than the minumum one).

Sum query Let f be the query returning the sum of the value for all individuals, thus it has range $\mathcal{R}_f = \{0, \ldots, vu\}$. By modifying the value of a single individual, the outcome of the query can be altered by at most v (when changing the value from 0 to v), thus two elements $i, j \in \mathcal{R}_f$ are adjacent iff $|i - j| \leq v$. The induced graph structure on $\mathcal{R}_f$ is shown in Figure 5(a) (for the case $v = 3$).

For our case-study we numerically evaluate this query for $u = 150, v = 5$ and for the uniform prior. We found that the minumum ϵ for which a tight-constraints mechanism exists (and is in fact unique since Φ is invertible) is 0.8. Figure 6(a) shows the utility of the tight-constraint mechanism, as well as that of the geometric mechanism, for values of ϵ between 0.8 and 1.3, the uniform prior and using and optimal remap. We see that the tight-constraint mechanism provides significantly higher utility than the geometric mechanism in this case.

2-count query Consider now the query f consisting of 2 counting queries (i.e. reporting the number of users satisfying properties p_1 and p_2), thus it has range $\mathcal{R}_f = \{0, \ldots, u\} \times \{0, \ldots, u\}$. By modifying the value of a single individual, the outcome of each counting query can be altered by at most 1, thus two anwers $(i_1, i_2), (j_1, j_2) \in \mathcal{R}_f$ are adjacent iff $|i_1 - j_1| \leq 1$ and $|i_2 - j_2| \leq 1$. The induced graph structure on $\mathcal{R}_f$ is shown in Figure 5(b).

We evaluate this query for $u = 30$ and for the uniform prior. We found that the minumum ϵ for which a tight-constraints mechanism exists is 0.9. Figure 6(b) shows the utility of the tight-constraint mechanism, as well as that of the geometric mechanism (applied independently to each counting query), for values of ϵ between 0.9 and 1.3 and the uniform prior. Similarly to the sum query, we see that the tight-constraint mechanism provides significantly higher utility than the geometric mechanism in this case.

7 Conclusion and future work

In this paper we have continued the line of research initiated by [6, 1] about the existence of universally-optimal differentially-private mechanisms. While the positive result of [6] (for counting queries) and the negative one of [1] (for essentially all other queries) answer the question completely, the latter sets a rather dissatisfactory scenario for differential privacy and the typical mechanisms used in the community, since counting queries are just one of the (infinitely many) kinds of queries one can be interested in. In practice the result of [1] says that for any query other than counting queries one cannot devise a mechanism that gives the best trade-off between privacy and utility for all users. Hence one has to choose: either design a different mechanism for every user, or be content with a mechanism that, depending on the user, can be far from providing the best utility. We have then considered the question whether, for a generic query, the optimality is punctual or can actually be achieved with the same mechanism for a class of users. Fortunately the answer is positive: we have identified a class of priors, called ϵ-regular, and a mechanism which is optimal for all the priors in this class. We have also provided a complete and effectively checkable characterisation of the conditions under which such mechanism exists, and an effective method to construct it. As a side result, we have improved on the existing bounds for the min-entropy leakage induced by differential privacy. More precisely, we have been able to give specific and tight bounds for each ϵ-regular prior, in general smaller than the bound existing in literature for the worst-case leakage (achieved by the uniform prior [20]).

So far we have been studying only the case of utility for binary gain functions. In the future we aim at lifting this limitation, i.e. we would like to consider also other kinds of gain. Furthermore, we intend to study how the utility decreases when we use a tight-constraints mechanism outside the class of ϵ-regular priors. In particular, we aim at identifying a class of priors, larger than the ϵ-regular ones, for which the tight-constraints mechanism is close to be optimal.

The definition of tight-constrains mechanism is related to the connectivity condition of the column graphs introduced by Kifer and Lin [22, 23]. They show that this property implies maximality w.r.t. the postprocessing preorder. As pointed out by an anonymous reviewer, we can probably exploit this result to strengthen our results in Section 5, in particular Theorem 3.

The negative result of [1] is stated in terms of the graph induced by $\sim_f$: a universally optimal mechanism can exist only if such graph is a line. This is the case of counting queries, but not only: any composition of a counting query with a function that preserves the graph structure would induce the same kind of graph, and it's for this reason that the authors of [1] write "*essentially* counting queries". As pointed out by an anonymous reviewer, we can use the techniques of our paper to prove that a universally optimal mechanism exists if and only if the query is derivable from a counting query by a bijection, thus making the result of [1] more precise, and extending the result of [6].

Acknowledgement We would like to thank the anonymous reviewers for their invaluable recommendations for improving the paper, and their suggestion for future work.

References

1. Brenner, H., Nissim, K.: Impossibility of differentially private universally optimal mechanisms. In: Proc. of FOCS, IEEE (2010) 71–80
2. Dwork, C.: Differential privacy. In: Proc. of ICALP. Volume 4052 of LNCS., Springer (2006) 1–12
3. Dwork, C., Lei, J.: Differential privacy and robust statistics. In: Proc. of STOC, ACM (2009) 371–380
4. Dwork, C.: Differential privacy in new settings. In: Proc. of SODA, SIAM (2010) 174–183
5. Dwork, C.: A firm foundation for private data analysis. Communications of the ACM **54**(1) (2011) 86–96
6. Ghosh, A., Roughgarden, T., Sundararajan, M.: Universally utility-maximizing privacy mechanisms. In: Proc. of STOC, ACM (2009) 351–360
7. Gupte, M., Sundararajan, M.: Universally optimal privacy mechanisms for minimax agents. In: Proc. of PODS, ACM (2010) 135–146
8. Alvim, M.S., Andrés, M.E., Chatzikokolakis, K., Palamidessi, C.: On the relation between Differential Privacy and Quantitative Information Flow. In: Proc. of ICALP. Volume 6756 of LNCS., Springer (2011) 60–76
9. Köpf, B., Basin, D.A.: An information-theoretic model for adaptive side-channel attacks. In: Proc. of CCS, ACM (2007) 286–296
10. Clark, D., Hunt, S., Malacaria, P.: Quantitative information flow, relations and polymorphic types. J. of Logic and Computation **18**(2) (2005) 181–199
11. Boreale, M.: Quantifying information leakage in process calculi. In: Proc. of ICALP. Volume 4052 of LNCS., Springer (2006) 119–131
12. Malacaria, P.: Assessing security threats of looping constructs. In: Proc. of POPL, ACM (2007) 225–235
13. Chatzikokolakis, K., Palamidessi, C., Panangaden, P.: Anonymity protocols as noisy channels. Inf. and Comp. **206**(2–4) (2008) 378–401
14. Smith, G.: On the foundations of quantitative information flow. In: Proc. of FOSSACS. Volume 5504 of LNCS., Springer (2009) 288–302
15. Braun, C., Chatzikokolakis, K., Palamidessi, C.: Quantitative notions of leakage for one-try attacks. In: Proc. of MFPS. Volume 249 of ENTCS., Elsevier (2009) 75–91
16. Köpf, B., Smith, G.: Vulnerability bounds and leakage resilience of blinded cryptography under timing attacks. In: Proc. of CSF, IEEE (2010) 44–56
17. Andrés, M.E., Palamidessi, C., van Rossum, P., Smith, G.: Computing the leakage of information-hiding systems. In: Proc. of TACAS. Volume 6015 of LNCS., Springer (2010) 373–389
18. Barthe, G., Köpf, B.: Information-theoretic bounds for differentially private mechanisms. In: Proc. of CSF, IEEE (2011) 191–204
19. Clarkson, M.R., Schneider, F.B.: Quantification of integrity (2011) Tech. Rep.. `http://hdl.handle.net/1813/22012`.
20. Alvim, M.S., Andrés, M.E., Chatzikokolakis, K., Degano, P., Palamidessi, C.: Differential Privacy: on the trade-off between Utility and Information Leakage. In: Postproceedings of the 8th Int. Worshop on Formal Aspects in Security and Trust (FAST). Volume 7140 of LNCS., Springer (2011) 39–54
21. Rényi, A.: On Measures of Entropy and Information. In: Proc. of the 4th Berkeley Symposium on Mathematics, Statistics, and Probability. (1961) 547–561
22. Kifer, D., Lin, B.R.: Towards an axiomatization of statistical privacy and utility. In: Proc. of PODS, ACM (2010) 147–158
23. Kifer, D., Lin, B.R.: An axiomatic view of statistical privacy and utility. Journal of Privacy and Confidentiality **4**(1) (2012) 5–49

Appendix

A Proofs

Proof of Lemma 1:

Proof. Assuming $\mathcal{K}$ is oblivious, then by Definition 1, the ϵ- differential privacy of $\mathcal{K}$ is written as follows.

$$\frac{\sum_{r\in\mathcal{R}_f} P(r \mid v)\cdot P(S \mid r)}{\sum_{r\in\mathcal{R}_f} P(r \mid v')\cdot P(S \mid r)} \leq e^{\epsilon} \quad \forall S \subseteq \mathcal{O}, \forall v, v' \in \mathcal{V} \text{ such that } v \sim v'. \tag{12}$$

Since the query f is deterministic, $P(r \mid v) = 1$ if $r = f(v)$, and is 0 otherwise. Therefore, Condition (12) is written as follows.

$$\frac{P(S \mid f(v))}{P(S \mid f(v'))} \leq e^{\epsilon} \quad \forall S \subseteq \mathcal{O}, \forall v, v' \in \mathcal{V} \text{ such that } v \sim v'. \tag{13}$$

Now we express the above condition in terms of adjacent query results instead of adjacent databases. For any pair of query results i, h such that $i \sim_f h$, there exists (by Def. 3) a pair of databases v, v' such that $f(v) = i$, $f(v') = h$, and $v \sim v'$. Applying Condition (13) to v, v', we get $P(S \mid i)/P(S \mid h) \leq e^{\epsilon}$. Repeating the same argument for all pairs of adjacent query results we get

$$\frac{P(S \mid i)}{P(S \mid h)} \leq e^{\epsilon} \quad \forall S \subseteq \mathcal{O}, \forall i, h \in \mathcal{R}_f \text{ such that } i \sim_f h. \tag{14}$$

We also imply Condition (13) from (14) as follows. For any pair of adjacent databases v, v', if $f(v) \neq f(v')$ then $f(v) \sim_f f(v')$ (because $v \sim v'$), and hence applying Condition (14) with $i = f(v), h = f(v')$ yields that $P(S \mid f(v))/P(S \mid f(v')) \leq e^{\epsilon}$. If otherwise $f(v) = f(v')$ then this ratio is 1 which is strictly less than e^{ϵ}. Repeating the same argument for all pairs of adjacent databases we get Condition (13). It holds therefore that (13) is equivalent to (14). It remains to show that (14) is equivalent to

$$\frac{P(o \mid i)}{P(o \mid h)} \leq e^{\epsilon} \quad \forall o \in \mathcal{O}, \forall i, h \in \mathcal{R}_f \text{ such that } i \sim_f h. \tag{15}$$

For all $o \in \mathcal{O}$, applying (14) to the subsets $S = \{o\}$, we easily get (15). Now we consider the other direction of implication. Note that it holds for any subset $S \subseteq \mathcal{O}$ and query result i that $P(S \mid i) = \sum_{o\in S} P(o \mid i)$. If (15) holds. Then it holds for any subset S and any adjacent query results i, h that $P(S \mid i) \leq e^{\epsilon} \sum_{o\in S} P(o \mid h) = e^{\epsilon} P(S \mid h)$, and hence (14) is implied by quantifying over all possible subsets and adjacent query results. □

Proof of Proposition 1:

Proof. Using the characterisation of ϵ-differential privacy given by Lemma 1, it is enough to prove the following equivalence for all outputs $o \in \mathcal{O}$.

$$x_{io} \leq e^{\epsilon}\cdot x_{ho} \quad \forall i, h \in \mathcal{R}_f, i \sim_f h \quad \Leftrightarrow \quad x_{io} \leq e^{\epsilon\, d(i,h)}\cdot x_{ho} \quad \forall i, h \in \mathcal{R}_f.$$

The direction '⇐' is proved by restricting the right statement to pairs i, h having distance $d(i,h) = 1$. The other implication '⇒' is proved by induction on the distance between graph nodes: For all i, h where $d(i,h) = 1$, it holds that

$$x_{io} \leq e^{\epsilon\, d(i,h)} \cdot x_{ho} \quad \forall o. \tag{16}$$

Now we set our hypothesis that Inequality (16) holds for all i, h where $d(i,h) = d$, and then prove that the hypothesis holds for distance $d+1$. For any node u at distance $d+1$ from i, there is an adjacent node h (to u) such that $d(i,h) = d$. Then we have

$$x_{io} \leq e^{\epsilon\, d} \cdot x_{ho} \quad \text{and} \quad x_{ho} \leq e^{\epsilon}\, x_{uo}.$$

Thus, we obtain

$$x_{io} \leq e^{\epsilon\,(d+1)} \cdot x_{uo}$$

That is, the hypothesis (16) holds for all pairs i, u having distance $d(i,u) = d+1$. □

Proof of Proposition 2:

Proof. By Definition 5, the ratio π_i/π_j is given by

$$\pi_i/\pi_j = \frac{\sum_k y_k \phi_{ki}}{\sum_k y_k \phi_{kj}} \tag{17}$$

where

$$\phi_{kj} = \left(e^{-\epsilon}\right)^{d(k,j)} \geq \left(e^{-\epsilon}\right)^{d(k,i)+d(i,j)} = \left(e^{-\epsilon}\right)^{d(i,j)} \cdot \phi_{ki}$$

The above inequality is implied by the triangle inequality, $d(k,j) \leq d(k,i) + d(i,j)$ and the fact that $e^{-\epsilon} < 1$. Since $y_k \geq 0$ for all k, it holds that

$$\sum_k y_k \phi_{kj} \geq \left(e^{-\epsilon}\right)^{d(i,j)} \cdot \sum_k y_k \phi_{ki}$$

Substituting the above inequality in Eq. (17) completes the proof. □

Proof of Proposition 3:

Proof. By Proposition 2, it holds for any pair of entries π_i, π_j that

$$\pi_j \leq e^{\epsilon\, d(i,j)} \cdot \pi_i \quad \text{and} \quad e^{-\epsilon\, d(i,j)} \cdot \pi_i \leq \pi_j.$$

Summing the above inequalities over j, we get

$$\sum_j \pi_j \leq \pi_i \cdot \sum_j e^{\epsilon\, d(i,j)} \quad \text{and} \quad \pi_i \sum_j e^{-\epsilon\, d(i,j)} \leq \sum_j \pi_j.$$

Since $\sum_j \pi_j = 1$, the above inequalities imply the upper and lower bounds for π_i. □

Proof of Proposition 4:

Proof. By Definition 5, a prior $\boldsymbol{\pi}$ is ϵ-differentially informative if and only if there exists vector $\boldsymbol{y}$ such that $\boldsymbol{\pi} = \boldsymbol{y}\,\Phi$ and $y_i \geq 0$ for all $i \in \mathcal{R}_f$; that is if and only if there are reals $y_i \geq 0$ for all $i \in \mathcal{R}_f$, such that $\boldsymbol{\pi}$ can be written as a linear combination Φ's rows as follows.

$$\boldsymbol{\pi} = \sum_{i \in \mathcal{R}_f} y_i\, \Phi_i,$$

where Φ_i is the row of Φ corresponding to the query result i. By Definition 6, observe that each row Φ_i is equal to $\left(\sum_{k \in \mathcal{R}_f} \phi_{ik}\right) \boldsymbol{c}^i$. Now substitute this expression in the above equation for $\boldsymbol{\pi}$, and let $\gamma_i = y_i \left(\sum_{k \in \mathcal{R}_f} \phi_{ik}\right)$. Note that $\gamma_i \geq 0$ if and only if $y_i \geq 0$ for all $i \in \mathcal{R}_f$. Thus we conclude that the condition given by the proposition is equivalent to the condition given by Def. 5. □

Proof of Theorem 1:

Proof. Since $\boldsymbol{\pi}$ is ϵ-regular, then it holds $\boldsymbol{\pi} = \boldsymbol{y}\,\Phi$ for a vector $\boldsymbol{y}$ where $y_i \geq 0$ for all $i \in \mathcal{R}_f$. Applying Lemma 2 and noting that $s_i \geq 0$ for all $i \in \mathcal{R}_f$, we observe that $\boldsymbol{y}\,\boldsymbol{s} \geq 0$ and hence the utility is upper-bounded by $\boldsymbol{y}\,\mathbf{1} = \sum_{i \in \mathcal{R}_f} y_i$. Note also that this bound is met if and only if all entries of the slack vector $\boldsymbol{s}$ in Eq. (7) are 0, because $y_i \geq 0$ for all entries i. By Eq. (3), the condition $\boldsymbol{s} = \mathbf{0}$ is equivalent to $\Phi \cdot \mathrm{diag}(X \cdot R) = \mathbf{1}$. □

Proof of Proposition 6:

Proof. By Proposition 5, the leakage $\mathcal{L}(X, \boldsymbol{\pi})$ is monotonically increasing with the utility $\mathcal{U}(X, \boldsymbol{\pi}, \hat{R})$. By Theorem 1, this utility is upper-bounded by $\sum_{i \in \mathcal{R}_f} y_i$ substituting this upper bound in Proposition 5 yields the inequality (9) where the equality holds if and only if it also holds in Theorem 1 for X and $\hat{R}$. That is if and only if $\Phi\,\mathrm{diag}(X\,\hat{R}) = \mathbf{1}$. This condition is equivalent to the condition of equality in Proposition 6, because if a remap R satisfies this latter condition then it must be optimal as the utility with R (by Theorem 1) is globally maximum, that is no other remap can achieve higher utility. □

Proof of Proposition 7:

Proof. For the noise matrix $\hat{X}$ of a tight-constraints mechanism, we want to show (according to Proposition 1) that for every pair of query results i, h and every output o, it holds that

$$\hat{x}_{io} \leq e^{\epsilon\, d(i,h)} \cdot \hat{x}_{ho}. \tag{18}$$

By Definition 7 it holds for every pair of nodes i, h and every output o, that

$$\hat{x}_{ho} = e^{-\epsilon\, d(h,o)} \cdot \hat{x}_{oo} \quad \text{and} \quad \hat{x}_{io} = e^{-\epsilon\, d(i,o)} \cdot \hat{x}_{oo}. \tag{19}$$

If $\hat{x}_{oo} = 0$ then $\hat{x}_{ho} = \hat{x}_{io} = 0$. In this case, Condition (18) is satisfied. On the other hand, if $\hat{x}_{oo} \neq 0$, then both $\hat{x}_{ho}$ and $\hat{x}_{io}$ are non-zero, and it follows from Equations (19) that, for every inputs i and h, and every output o,

$$\hat{x}_{ho} / \hat{x}_{io} = e^{-\epsilon\,(d(h,o) - d(i,o))}.$$

By the triangle inequality, it holds that $d(h,o) - d(i,o) \leq d(i,h)$. Knowing also that $e^{-\epsilon} < 1$, it follows from the above inequality that

$$\hat{x}_{ho} / \hat{x}_{io} \geq e^{-\epsilon\, d(i,h)}.$$

The above inequality implies Condition (18) of differential privacy. □

Proof of Lemma 3:

Proof. If a mechanism $\mathcal{K}$ is a tight-constraints mechanism, then it holds for its noise matrix X, by Def. 7 that $x_{ik} = e^{-\epsilon\, d(i,k)}\, x_{kk}$ for all $i, k \in \mathcal{R}_f$. It also holds that $\sum_{k\in\mathcal{R}_f} x_{ik} = 1$ for all $i \in \mathcal{R}_f$. Combining these equations yields

$$\sum_{k\in\mathcal{R}_f} e^{-\epsilon\, d(i,k)}\, x_{kk} = 1, \qquad \forall i \in \mathcal{R}_f.$$

Using the privacy-constraints matrix Φ, the above equations can be written in the matrix form (11). Now let X be the noise matrix of any ϵ-differentially private mechanism $\mathcal{K}$. We prove that if X satisfies Equation (11) then $\mathcal{K}$ must be a tight-constraints mechanism. Note that if X satisfies Equation (11), then there must be a tight-constraints mechanism $\hat{X}$ having the same diagonal as X. Suppose for a contradiction that X deviates from $\hat{X}$ in the values of non-diagonal entries. Deviating the mechanism $\hat{X}$, which satisfies Eqs. (10), to another mechanism X while keeping the same diagonal requires increasing at least one non-diagonal entry x_{ik} to preserve the differential privacy condition $x_{ik} \geq e^{-\epsilon\, d(i,k)}\, x_{kk}$. This increment of x_{ik} has to be deducted from one or more entries in the same row i. Let x_{ij} be any of these decremented entries. To preserve the privacy condition $x_{ij} \geq e^{-\epsilon\, d(i,j)} x_{jj}$ we have to decrease the diagonal entry x_{jj}. The change of x_{jj} contradicts that the diagonals of X and $\hat{X}$ are the same. □

Proof of Lemma 4:

Proof. By Lemma 3, if there is at least a tight-constraints mechanism, then Eq. (11) must hold for this mechanism's noise matrix $\hat{X}$. Taking the transpose of both sides in this equation, and noting that $\Phi^t = \Phi$ (because Φ is symmetric), then we imply that

$$(\mathrm{diag}(\hat{X}))^t \cdot \Phi = \mathbf{1}^t.$$

Scaling the above equation by $1/|\mathcal{R}_f|$ yields the row vector $\boldsymbol{u}$, the uniform prior, on the right hand side. Thus if a tight-constraints mechanism, with noise matrix $\hat{X}$, exists then

$$(1/|\mathcal{R}_f|)\,(\mathrm{diag}(\hat{X}))^t \cdot \Phi = \boldsymbol{u}.$$

which means (By Def. 5) that $\boldsymbol{u}$ is ϵ-regular, because the row vector $(\mathrm{diag}(\hat{X}))^t$ has only non-negative entries. For the opposite implication, assume that $\boldsymbol{u}$ is ϵ-regular. Then by definition there is a row vector $\boldsymbol{y}$ with non-negative entries such that $\boldsymbol{y}\,\Phi = \boldsymbol{u}$. Taking the transpose of both sides, and multiplying by $|\mathcal{R}_f|$, yields that Eq. (11) is satisfied for the noise matrix X, whose diagonal is given by $\mathrm{diag}(X) = |\mathcal{R}_f| \cdot \boldsymbol{y}^t$ (non-negative). Thus there exists a tight-constraints mechanism whose noise matrix is X. □

B On the invertibility of the privacy-constraints matrix

In this section we show that the matrix Φ introduced in Definition 4 can be both invertible or not invertible.

Assume that the set of query results $\mathcal{R}_f$, and its adjacency relation, are given by the graph represented in Fig. 7, which is obtained by the Hamming graph 2^3 by adding an

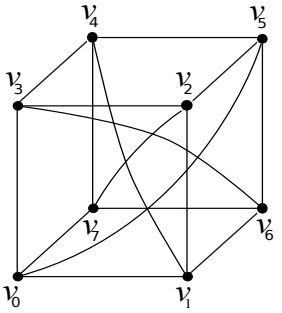


Fig. 7. The set of query results $\mathcal{R}_f$ and its adjacency relation

arc between each pair of nodes at distance 3 (thus in the resulting graph the maximal distance is 2).

Consider the matrix Φ defined by $\phi_{ih} = \alpha^{d(v_i,v_h)}$, where $\alpha = e^{-\epsilon}$. It is easy to see that if $\alpha = 1/3$, then the matrix is not invertible. In fact, if we denote by c_i the row corresponding to the node v_i, we have

$$c_0 + c_2 + c_4 + c_6 = c_1 + c_3 + c_5 + c_7$$

This can be easily proved by observing that for each position of the vectorial sum one side of the equation is $1 + 3\,\alpha^2$ while the other is $4\,\alpha$.

On the other hand, there are values of α for which Φ is invertible. For instance for $\alpha \leq 1/7$ it is possible to show that the columns are linearly independent. Intuitively, this is because the elements which are not in the diagonal are too small to sum up to the diagonal. Note also that as α approaches 0 the matrix Φ approaches the identity matrix.

Broadening the scope of Differential Privacy Using Metrics*

Konstantinos Chatzikokolakis[1,2], Miguel E. Andrés[2], Nicolás E. Bordenabe[3,2], and Catuscia Palamidessi[3,2]

[1] CNRS
[2] LIX, Ecole Polytechnique
[3] INRIA

Abstract. Differential Privacy is one of the most prominent frameworks used to deal with disclosure prevention in statistical databases. It provides a formal privacy guarantee, ensuring that sensitive information relative to individuals cannot be easily inferred by disclosing answers to aggregate queries. If two databases are adjacent, i.e. differ only for an individual, then the query should not allow to tell them apart by more than a certain factor. This induces a bound also on the distinguishability of two generic databases, which is determined by their distance on the Hamming graph of the adjacency relation.
In this paper we explore the implications of differential privacy when the indistinguishability requirement depends on an arbitrary notion of distance. We show that we can naturally express, in this way, (protection against) privacy threats that cannot be represented with the standard notion, leading to new applications of the differential privacy framework. We give intuitive characterizations of these threats in terms of Bayesian adversaries, which generalize two interpretations of (standard) differential privacy from the literature. We revisit the well-known results stating that universally optimal mechanisms exist only for counting queries: We show that, in our extended setting, universally optimal mechanisms exist for other queries too, notably sum, average, and percentile queries. We explore various applications of the generalized definition, for statistical databases as well as for other areas, such that geolocation and smart metering.

1 Introduction

Differential privacy [1,2] is a formal definition of privacy which originated from the area of statistical databases, and it is now applied in many other domains, ranging from programming languages [3] to social networks [4] and geolocation [5].

Statistical databases are queried by analysts to obtain aggregate information about individuals. It is important to protect the privacy of the participants in the database, in the sense that it should not be possible to infer the value of an individual from the aggregate information. This can be achieved by adding random noise to the answer.

* This work is partially funded by the Inria large scale initiative CAPPRIS, the EU FP7 grant no. 295261 (MEALS), and the project ANR-12-IS02-001 PACE. Nicolás E. Bordenabe was partially funded by the French Defense procurement agency (DGA) with a PhD grant.

Because of the focus on the single individual as the unit of protection, differential privacy relies in a crucial way on the notion of two databases being *adjacent*, i.e. differing only for an individual. A mechanism K is ϵ-differentially private if for any two adjacent databases x, x', and any property Z, the probability distributions $K(x), K(x')$ differ on Z at most by e^{ϵ}, namely, $K(x)(Z) \leq e^{\epsilon} K(x')(Z)$. For two non-adjacent databases, there is no requirement other than the one induced by the transitive application of the property. Note that the set of all possible databases, together with the adjacency relation, forms a *Hamming graph*, and the graph distance $d_h(x, x')$ between x and x' is exactly the number of individuals in which x and x' differ. Then, for any databases x, x', it is easy to see (by transitivity on a path from x to x') that $K(x)(Z) \leq e^{\epsilon\, d_h(x,x')} K(x')(Z)$. We can view $\epsilon d_h(x, x')$ as the distinguishability level between two generic databases x, x': the smaller $\epsilon d_h(x, x')$ is, the more similar the probability distributions $K(x)$, $K(x')$ are required to be.

When the sensitive information to be protected is other than the value of a single individual, it is common to consider different notions of adjacency. For example, in cases of cohesive groups with highly correlated values, we could consider adjacent two databases differing in any number of individuals of the same group. Similarly, when dealing with friendship graphs in social networks, adjacency could be defined as differing in a single edge.

We argue that in some situations the distinguishability level between x and x' should depend not only on the number of different values between x and x', but also on the values themselves. We might require, for instance, databases in which the value of an individual is only slightly modified to be highly indistinguishable, thus protecting the *accuracy* by which an analyst can infer an individual's value.

More generally, we might want to apply differential privacy in scenarios when x, x' are not databases at all, but belong to an *arbitrary domain of secrets* $\mathcal{X}$. In such a scenario, there might be no natural notion of adjacency, but it is still reasonable to define a distinguishability level between secrets, and employ the same principle of differential privacy – i.e. the smaller the distinguishability level between x, x' is, the more similar the probability distributions $K(x)$, $K(x')$ are required to be – to obtain a meaningful notion of privacy. For instance, when dealing with geographic locations (aka, geolocation), it might be acceptable to disclose the fact that an individual is in Paris rather than in New York. However, disclosing the *precise* location of the individual within Paris is likely to be undesired (because, for instance, the individual is currently in Moulin Rouge rather than in his office in Place d'Italie). Thus it would be useful to have a distinguishability level that depends on the geographical distance.

In this paper we assume that we have a numeric function $\boldsymbol{\varepsilon}(x, x')$, giving the distinguishability level between x, x', which depends on the application at hand and the privacy guarantees we wish to express. The corresponding notion of privacy is the requirement that for an arbitrary pair x, x' we have

$$K(x)(Z) \leq e^{\boldsymbol{\varepsilon}(x,x')} K(x')(Z)$$

Note that standard ϵ-differential privacy is a particular case of this notion, that we obtain by setting $\boldsymbol{\varepsilon}(x, x') = \epsilon\, d_h(x, x')$.

Since $\boldsymbol{\varepsilon}$ models distinguishability, there are some properties that it is expected to satisfy. First, it should be the case that any element is indistinguishable from itself, i.e.

$\varepsilon(x, x) = 0$. Second, the distinguishability level of x and x' should be the same as that of x' and x, i.e. $\varepsilon(x, x') = \varepsilon(x', x)$ (symmetry). Finally, if x_1 and x_2 are hardly distinguishable from x_3, then they should be also hardly distinguishable from each other. In other words, $\varepsilon(x_1, x_2)$ should be bounded by a function of $\varepsilon(x_1, x_3)$, $\varepsilon(x_2, x_3)$. In this paper we assume the triangle inequality, namely $\varepsilon(x_1, x_2) \leq \varepsilon(x_1, x_3) + \varepsilon(x_3, x_2)$, which means that ε is a metric. We believe that more relaxed notions of distinguishability could potentially be useful, we leave the investigation of this possibility as future work.[4] In the rest of this paper we use d (for "distance") instead of ε, and we call the corresponding privacy notion "d-privacy".

Similarly to the standard definition, d-privacy does not explicitly talk about the adversary's gain of knowledge. In order to better understand a privacy property, however, it is useful to provide interpretations that directly reason about the capabilities of the adversary. Two such interpretations exist for differential privacy: the first states that, regardless of side knowledge, the adversary's gain of knowledge by observing the reported answer is the same whether or not the individual's data were included in the database [1,6]. The second states that, an informed adversary who already knows all values except individual's i, gains no extra knowledge from the reported answer, regardless of side knowledge about i's value [2].[5]

In the case of d-privacy, we provide two results that generalize the above interpretations, showing the privacy guarantees provided by a certain metric d. The first uses the concept of a *hiding* function $\phi : \mathcal{X} \to \mathcal{X}$. The idea is that ϕ can be applied to a secret x before the mechanism K, so that the latter has only access to a hidden version $\phi(x)$, instead of the real secret x. Then d-privacy implies that the adversary's conclusions (captured by his posterior distribution) are similar (up to a factor depending on ϕ) regardless of whether ϕ is applied to the secret or not. Moreover, we show that certain classes of hiding functions are "canonical", in the sense that if the property holds for those functions, it must hold in general.

The above characterization compares two posterior distributions and does not imply that the adversary learns no information, but that he learns the same regardless of whether the secret has been hidden or not. We then give a second characterization, comparing the adversary's conclusions (a posterior distribution) to his initial knowledge (a prior distribution). Since some information is allowed to be revealed, we cannot expect the two to be similar. Still, if we restrict to a neighborhood N of secrets that are close to each other, we can show that d-privacy implies that an informed adversary, knowing that the secret belongs to N, can gain little more information about the exact secret, regardless of his prior knowledge within N. Similarly to the previous characterization, we also show that certain classes of neighborhoods are canonical.

We give examples of privacy problems in various contexts, and show how to define appropriate metrics. In the context of statistical databases, we consider metrics that

[4] Several of our results do not actually depend on ε being a metric.

[5] The knowledge increase of a non-informed adversary is not bounded by e^{ϵ}. Recalling the well-known example from [1], consider the side information that Terry Gross is two inches shorter than the average Lithuanian woman. Then obtaining the average height (even a noisy one) gives little additional information about Terry Gross to an informed adversary, but substantial information to a non-informed one.

depend not only on the number of different values, but also on the values themselves. First, a stronger variant of differential privacy is given in which databases differing in a single individual are hardly distinguishable, but the distinguishability level becomes even lower when the difference in the values is small. Moreover, this metric can be relaxed to obtain a privacy notion that focuses on protecting the accuracy of a value. This can be useful, for instance, in case an individual does not mind disclosing his age group, but wants to protect his exact birthday date (such precise information could in principle allow to identify the individual with little margin of error).

Departing from statistical databases, we consider smart meters, and the problem for privacy that can derive from accurate measurement of energy consumption at high frequency. Further, we consider the problem of hiding the exact position in location-based services. In all these examples, besides the proper metric notion, we construct also the canonical adversary which provides the operational interpretation.

Next, we turn our attention to the notion of utility, namely the accuracy of the reported answer, and in particular the Bayesian notion of utility [7,8], which takes into account the prior knowledge of the user. In general mechanisms may provide different degrees of utility for the same level of privacy, and obviously it is desirable to identify the optimal ones. Of particular interest are the *universally optimal* mechanisms, which provide optimal utility for all users (i.e., all priors). There are two well known results concerning universal optimality: the first [7] establishes that for counting queries the geometric and the truncated geometric mechanisms are universally optimal. The second [8] says that for any other kind of query no universally optimal mechanism exists.

We revisit these results in our framework and show that in contrast to the standard case, d-privacy allows to construct (for certain metrics) universally optimal mechanisms for many other kinds of queries. More precisely, we show that universally optimal mechanisms exist in the cases of (i) the sum, average and percentile queries for the Manhattan metric, and (ii) the average and percentile queries for the Maximum metric.

We also study the additional noise required to achieve privacy for databases queries, when we use a finer metric than the Hamming distance. Surprisingly, it turns out that in the case (i) above, the sensitivity of the queries remains the same as in the standard case. This means that, a standard ϵ-differentially private mechanism already incorporates "for free" the additional protection w.r.t. proximity of values.

Related Work. Several works in the differential privacy literature consider adjacency relations different than the standard one, effectively using a metric tailored to that application. Examples include group privacy [1] and edge privacy for graphs [9].

The generalization of differential privacy to arbitrary metrics was considered also in [10,3]. In those works, however, the purpose of extending the definition was to obtain compositional methods for proving differential privacy in programming languages, while in our work we focus on the implications of such extension for the theory of differential privacy. Namely, we aim at obtaining new meaningful definitions of privacy for various contexts through the use of different metrics (cf. the examples of the smart meters and of geolocation), and at investigating the existence of optimal mechanisms.

Another work closely related to ours is [11] in which an extended definition of differential privacy is used to capture the notion of fairness in classification. A metric d is used to model the fact that certain individuals are required to be classified similarly,

and a mechanism satisfying d-privacy is considered fair, since it produces similar results for similar individuals. We view fairness as one of the many interesting notions that can be obtained through the use of metrics in various contexts, thus it encourages our goal of studying d-privacy. With respect to the actual metrics used in this paper, the difference is that we consider metrics that depend on the individuals' values, while [11] considers metrics between individuals.

Contribution. The main contributions of this paper are summarized below:

- We study d-privacy – an extension of differential privacy to arbitrary domains endowed with a metric d – in the general case, independently from any specific metric.
- We give two operational characterizations of d-privacy that directly constraint the capabilities of the adversary.
- We show examples of applications of d-privacy to privacy scenarios both in databases and in other contexts.
- We show that several queries (including the sum, average and percentile) admit universally optimal mechanisms for certain metrics. This contrasts sharply with standard differential privacy, where such mechanisms exist only for counting queries.

Plan of the Paper. In the next section we recall some preliminary notions about mechanisms, metrics, and differential privacy. Section 3 introduces the notion of d-privacy and presents two characterization results. In Section 4 we give a sufficient and necessary condition for the privacy of an oblivious mechanism, we discuss Laplace mechanisms, and we give sufficient conditions for a mechanism to be optimal. In Sections 5 and 6 we give several examples of applications of our notions, in statistical databases with an enriched notion of privacy, and in other domains, respectively. We also show how to construct universally optimal mechanisms for some of those examples in the cases of sum, average, and percentile queries. Section 7 concludes.

All proofs can be found in the appendix.

2 Preliminaries

Mechanisms. Given two sets $\mathcal{X}$ and $\mathcal{Z}$, let $\mathcal{F}_\mathcal{Z}$ be a σ-algebra over $\mathcal{Z}$ and let $\mathcal{P}(\mathcal{Z})$ be the set of probability measures over $\mathcal{Z}$. A *mechanism* from $\mathcal{X}$ to $\mathcal{Z}$ is a (probabilistic) function $K : \mathcal{X} \to \mathcal{P}(\mathcal{Z})$. A mechanism K can be described in terms of probability density functions (pdf's), that is by a function $D : \mathcal{X} \to \mathcal{D}(\mathcal{Z})$ (where $\mathcal{D}(\mathcal{Z})$ denotes the space of the pdf's over $\mathcal{Z}$), such that $D(x)$ is the pdf of $K(x)$.

The composition $H \circ f$ of a deterministic function $f : \mathcal{X} \to \mathcal{Y}$ (called a *query*) and a mechanism $H : \mathcal{Y} \to \mathcal{P}(\mathcal{Z})$ is the mechanism $K : \mathcal{X} \to \mathcal{P}(\mathcal{Z})$ defined as $K(x) = (H \circ f)(x) = H(f(x))$. Mechanisms of this form are called *oblivious*.

Let π be a discrete probability measure on $\mathcal{X}$, called a *prior*.[6] Starting from π and using Bayes' rule, each observation $Z \in \mathcal{Z}$ of a mechanism $K : \mathcal{X} \to \mathcal{P}(\mathcal{Z})$ induces a *posterior* measure $\sigma = \mathbf{Bayes}(\pi, K, Z)$ on $\mathcal{X}$, defined as $\sigma(x) = \frac{K(x)(Z)\pi(x)}{\sum_{x' \in \mathcal{X}} K(x')(Z)\pi(x')}$.

[6] We restrict to discrete priors for simplicity; all results could be carried to the continuous case.

Metrics. A metric on a set $\mathcal{X}$ is a function $d_{\mathcal{X}} : \mathcal{X}^2 \to [0, \infty]$ such that $d_{\mathcal{X}}(x, y) = 0$ iff $x = y$, $d_{\mathcal{X}}(x, y) = d_{\mathcal{X}}(y, x)$, and $d_{\mathcal{X}}(x, z) \leq d_{\mathcal{X}}(x, y) + d_{\mathcal{X}}(y, x)$ for all $x, y, z \in \mathcal{X}$. The *diameter* of $A \subseteq \mathcal{X}$ is defined as $d_{\mathcal{X}}(A) = \sup_{x,x' \in A} d_{\mathcal{X}}(x, x')$.

A sequence $x_1, \ldots, x_n$ is called a *chain* from x_1 to x_n and denoted by $\tilde{x}$. The length $d_{\mathcal{X}}(\tilde{x})$ of a chain is defined as $d_{\mathcal{X}}(\tilde{x}) = \sum_{i=1}^{n-1} d_{\mathcal{X}}(x_i, x_{i+1})$. If $d_{\mathcal{X}}(\tilde{x}) = d_{\mathcal{X}}(x_1, x_n)$ then $\tilde{x}$ is called *tight*.

Of particular interest are metrics *induced by a graph* $(\mathcal{X}, \sim_{\mathcal{X}})$, where $\sim_{\mathcal{X}}$ is the graph's adjacency relation. In the induced metric, $d_{\mathcal{X}}(x, x')$ is the length of the shortest path from x to x' (or infinite if no path exists). Of great interest are also the Manhattan (or L_1), the Euclidean (or L_2) and the Maximum (or L_∞) metrics, denoted by d_1, d_2, d_∞ respectively. The numerical distance on the reals (which coincides with all d_1, d_2, d_∞) will be denoted by $d_{\mathbb{R}}$ for clarity. Finally, of great interest is the metric $d_{\mathcal{P}}$ on $\mathcal{P}(\mathcal{Z})$ defined as $d_{\mathcal{P}}(\mu_1, \mu_2) = \sup_{Z \in \mathcal{F}_{\mathcal{Z}}} |\ln \frac{\mu_1(Z)}{\mu_2(Z)}|$ with the convention that $|\ln \frac{\mu_1(Z)}{\mu_2(Z)}| = 0$ if both $\mu_1(Z), \mu_2(Z)$ are zero and ∞ if only one of them is zero.

Differential Privacy. We fix a finite domain of values $\mathcal{V}$, called the *universe*. A database $x \in \mathcal{V}^n$ consists of n records from $\mathcal{V}$ - each corresponding to an individual - that is x is a tuple $\langle x[1], \ldots, x[n] \rangle, x[i] \in \mathcal{V}$, where $x[i]$ is the value of the i-th individual in the database. We denote by $x[{}^{v}/_{i}]$ the database obtained from x by substituting the value v for individual i. The case when individuals are allowed to be absent from the database can be modeled by the universe $\mathcal{V}_\varnothing = \mathcal{V} \cup \{\varnothing\}$ where the null value $\varnothing$ denotes absence.

A crucial notion for differential privacy is that of *adjacency*: two databases x, x' are adjacent, written $x \sim_h x'$, if they differ in exactly one element. Let d_h be the distance induced by $\sim_h$ (i.e., $d_h(x, x')$ is the number of elements in which x, x' differ). The graph $(\mathcal{V}^n, \sim_h)$ is known as *Hamming graph*, and d_h as Hamming distance.

Let $\mathcal{Z}$ be a set of query outcomes; a mechanism $K : \mathcal{V}^n \to \mathcal{P}(\mathcal{Z})$ satisfies ϵ-differential privacy if adjacent databases produce answers with probabilities that differ at most by a factor e^ϵ:

$$K(x)(Z) \leq e^\epsilon \, K(x')(Z) \quad \forall x \sim_h x' \in \mathcal{V}^n, Z \in \mathcal{F}_{\mathcal{Z}} \tag{1}$$

Following [3], the definition can be expressed as $d_{\mathcal{P}}(K(x), K(x')) \leq \epsilon$ for all $x \sim_h x'$. Moreover, as explained in the introduction, we can rewrite it in terms of the Hamming distance: $d_{\mathcal{P}}(K(x), K(x')) \leq \epsilon d_h(x, x')$ for all $x, x' \in \mathcal{V}^n$.

A desirable feature of this definition is that it solely depends on the mechanism itself, without explicitly talking about the adversary's side knowledge, or the information that he learns from the reported answer. However, in order to get a better understanding of a privacy definition, it is useful to give an "operational" (or "semantic") interpretation that directly restricts the abilities of the adversary. To this end, we capture the adversary's side knowledge by a prior distribution π on $\mathcal{V}^n$, and his conclusions after observing Z by the posterior distribution $\sigma = \mathbf{Bayes}(\pi, K, Z)$.

There are two operational interpretations commonly given to differential privacy. The first can be informally stated as: "regardless of side knowledge, by observing the reported answer an adversary obtains the same information whether or not the individual's data were included in the database". This can be formalized as follows: consider a *hiding* function $\phi_{i,v} : \mathcal{V}^n \to \mathcal{V}^n$ replacing i's value by a fixed value v, i.e.

$\phi_{i,v}(x) = x[^v/_i]$, and let $\Phi_h = \{\phi_{i,v} \mid i \in 1..n, v \in \mathcal{V}\}$ be the set of all such functions. The mechanism $K \circ \phi_{i,v}$ behaves as K after removing i's value; hence we require the posterior distributions induced by $K, K \circ \phi_{i,v}$ to be similar. The resulting notion (called "semantic privacy" in [6])[7] can be shown to be implied by differential privacy.

Theorem 1 ([6]). *If a mechanism $K : \mathcal{V}^n \to \mathcal{P}(\mathcal{Z})$ satisfies ϵ-differential privacy then for all priors π on $\mathcal{V}^n$, all $\phi \in \Phi_h$, and all $Z \in \mathcal{F}_\mathcal{Z}$:*

$$d_\mathcal{P}(\sigma_1, \sigma_2) \leq 2\epsilon \qquad \textit{where} \;\; \sigma_1 = \mathbf{Bayes}(\pi, K, Z) \;\; \textit{and} \;\; \sigma_2 = \mathbf{Bayes}(\pi, K \circ \phi, Z)$$

Note that the above interpretation compares two *posterior* measures. This requirement does not imply that the adversary learns no information, but that he learns the same regardless of the presence of the individual's data. Both σ_1, σ_2 can be very different than the prior π, as the well-known example of Terry Gross [1] demonstrates.

A different interpretation can be obtained by comparing the posterior σ to the prior distribution π. Of course, we cannot expect those to be similar, since some information is allowed to be disclosed. Still, we can require the distributions to be similar when restricted to the value of a single individual, by assuming an informed adversary who knows all other values in the database. Let $N_i(x) = \{x[^v/_i] \mid v \in \mathcal{V}\}$ denote the set of databases obtained from x by modifying i's value, and let $\mathcal{N}_h = \{N_i(x) \mid x \in \mathcal{V}^n, i \in 1..n\}$. Knowing that the database belongs to a set $N \in \mathcal{N}_h$ means that we know all values except one. We denote by $\pi_{|N}$ the distribution obtained from π by restricting to N, i.e. $\pi_{|N}(x) = \pi(x|N)$. Requiring $\pi_{|N}, \sigma_{|N}$ to be similar brings us the definition of "semantic security" from [2], which is a full characterization of differential privacy.

Theorem 2 ([2]). *A mechanism $K : \mathcal{V}^n \to \mathcal{P}(\mathcal{Z})$ satisfies ϵ-differential privacy iff for all priors π on $\mathcal{V}^n$, all $N \in \mathcal{N}_h$, and all $Z \in \mathcal{F}_\mathcal{Z}$:*

$$d_\mathcal{P}(\pi_{|N}, \sigma_{|N}) \leq \epsilon \qquad \textit{where} \qquad \sigma = \mathbf{Bayes}(\pi, K, Z)$$

Note that if the adversary does not know $N \in \mathcal{N}_h$, then his knowledge can (and will in most cases) be increased. Note also that the above result does not imply that K allows the adversary to learn $N_i(x)$! In fact, this is clearly forbidden since it would violate the same condition for $N_j(x), j \neq i$, i.e. it would violate the other individuals' privacy.

3 Generalized Privacy

As discussed in the introduction, differential privacy can be generalized to the case of an arbitrary set of secrets $\mathcal{X}$, equipped with a metric $d_\mathcal{X}$.

Definition 1. *A mechanism $K : \mathcal{X} \to \mathcal{P}(\mathcal{Z})$ satisfies $d_\mathcal{X}$-privacy, iff $\forall x, x' \in \mathcal{X}$: $d_\mathcal{P}(K(x), K(x')) \leq d_\mathcal{X}(x, x')$, or equivalently:*

$$K(x)(Z) \leq e^{d_\mathcal{X}(x,x')} \, K(x')(Z) \quad \forall Z \in \mathcal{F}_\mathcal{Z}$$

[7] The only difference between the semantic privacy of [6] and our formulation is that an "additive" metric between distributions is used instead of the "multiplicative" $d_\mathcal{P}$.

Intuitively, the definition requires that secrets close to each other wrt $d_{\mathcal{X}}$, meaning hardly distinguishable, should produce outcomes with similar probability. This is the same core idea as in differential privacy, which can be retrieved as $\mathcal{X} = \mathcal{V}^n, d_{\mathcal{X}} = \epsilon d_h$.

Note that Definition 1 contains no ϵ; the distinguishability level is directly given by the metric. In practice, the desired metric can be obtained from a standard one by scaling by a proper factor ϵ (recall that a scaled metric is also a metric). For instance, in the case of standard differential privacy, the Hamming distance between adjacent databases is 1, and we want their distinguishability level to be ϵ, hence we use the scaled version ϵd_h.

Note also that an *extended* metric (allowing $d_{\mathcal{X}}(x, x') = \infty$) can be useful in cases when we allow two secrets to be completely distinguished. The understanding of Definition 1 is that the requirement is always satisfied for those secrets. Similarly, *pseudo*-metrics (allowing $d_{\mathcal{X}}(x, x') = 0$ for $x \neq x'$) could be useful when we want some secrets to be completely indistinguishable (forcing $K(x)$ and $K(x')$ to be identical). To simplify the presentation, the results of this paper assume an extended metric (but not pseudo). An approximate version of $d_{\mathcal{X}}$-privacy can be defined, similarly to (α, δ) differential privacy [13]. We leave the study of such notion as future work.

Different metrics $d_{\mathcal{X}}, {d_{\mathcal{X}}}'$ on the same set $\mathcal{X}$ clearly give rise to different privacy notions. The "strength" of each notion depends on the distinguishability level assigned to each pair of secrets; $d_{\mathcal{X}}$-privacy and ${d_{\mathcal{X}}}'$-privacy are in general incomparable. However, lower distinguishability level implies stronger privacy.

Proposition 1. *If $d_{\mathcal{X}} \leq {d_{\mathcal{X}}}'$ (point-wise) then $d_{\mathcal{X}}$-privacy implies ${d_{\mathcal{X}}}'$-privacy.*

For example, some works consider an adjacency relation $\sim_r$ slightly different than $\sim_h$, defined as $x \sim_r x'$ iff $x' = x[^{\varnothing}/_i]$ (or vice versa), i.e. x' can be obtained from x by removing one individual. This relation gives rise to a metric d_r for which it holds that: $\frac{1}{2} d_r \leq d_h \leq d_r$. From Proposition 1, the two models are essentially equivalent; one can obtain ϵd_r-privacy from ϵd_h-privacy by doubling ϵ and vice versa.

Characterization 1. Similarly to standard differential privacy, $d_{\mathcal{X}}$-privacy does not explicitly talk about the adversary's gain of knowledge. To better understand the privacy guarantees provided by a certain metric $d_{\mathcal{X}}$, it is useful to directly reason about the capabilities of the adversary. Two such characterizations are given, generalizing the two interpretations of standard differential privacy (Theorems 1,2).

The first characterization uses the concept of a *hiding* function $\phi : \mathcal{X} \to \mathcal{X}$. The idea is that ϕ can be applied to x before the mechanism K, so that the latter has only access to a hidden version $\phi(x)$, instead of the real secret x. Let $d_{\mathcal{X}}(\phi) = \sup_{x \in \mathcal{X}} d_{\mathcal{X}}(x, \phi(x))$ be the maximum distance between a secret and its hidden version. We can show that $d_{\mathcal{X}}$-privacy implies that the adversary's conclusions (captured by his posterior measure) are the same (up to $2d_{\mathcal{X}}(\phi)$) regardless of whether ϕ is applied or not. Moreover, we show that certain classes of hiding functions are "canonical", in the sense that if the property holds for those, it must hold in general. We start be defining this class.

Definition 2. *Let Φ be a set of functions from $\mathcal{X}$ to $\mathcal{X}$, called* hiding *functions. A chain $\tilde{x}$ is called a* maximal Φ-chain *iff for every step i there exists $\phi \in \Phi$ s.t. $\phi(x_i) = x_{i+1}, \phi(x_{i+1}) = x_i$ and $d_{\mathcal{X}}(x_i, x_{i+1}) = d_{\mathcal{X}}(\phi)$. Then Φ is called* maximally tight *wrt $d_{\mathcal{X}}$ iff $\forall x, x' \in \mathcal{X}$ there exists a tight maximal Φ-chain from x to x'.*

Note that the above property requires hiding functions that *swap* the secrets x_i, x_{i+1}. This is not satisfied by the hiding functions $\phi_{i,v}$ introduced in the previous section, but will be satisfied by more general functions used later in the paper.

Theorem 3. *Let Φ be a set of hiding functions. If K satisfies $d_{\mathcal{X}}$-privacy then for all $\phi \in \Phi$, all priors π on $\mathcal{X}$, and all $Z \in \mathcal{F}_{\mathcal{Z}}$:*

$$d_{\mathcal{P}}(\sigma_1, \sigma_2) \leq 2\, d_{\mathcal{X}}(\phi) \quad \textit{where} \;\; \sigma_1 = \mathbf{Bayes}(\pi, K, Z) \textit{ and } \sigma_2 = \mathbf{Bayes}(\pi, K \circ \phi, Z)$$

If Φ is maximally tight then the converse also holds.

The above characterization compares two posterior distributions; hence, it does not impose that the adversary gains no information, but that this information is the same regardless of whether ϕ has been applied to the secret or not.

Characterization 2. A different approach is to compare the adversary's prior and posterior distributions, measuring how much he learned about the secret. Since we allow some information to be revealed, we cannot expect these distributions to be similar. Still, if we restrict to a neighborhood N of secrets that are close to each other, we can show that $d_{\mathcal{X}}$-privacy implies that an informed adversary, knowing that the secret belongs to N, can gain little more information about the exact secret regardless of his side knowledge about N. Moreover, similarly to the previous characterization, we show that certain classes of neighborhoods are "canonical".

Definition 3. *Let $\mathcal{N} \subseteq 2^{\mathcal{X}}$. The elements of $\mathcal{N}$ are called* neighborhoods*. A chain $\tilde{x}$ is called a* maximal $\mathcal{N}$-chain *iff for every step i there exists $N \in \mathcal{N}$ such that $\{x_i, x_{i+1}\} \subseteq N$ and $d_{\mathcal{X}}(x_i, x_{i+1}) = d_{\mathcal{X}}(N)$. Then $\mathcal{N}$ is called* maximally tight *wrt $d_{\mathcal{X}}$ iff $\forall x, x' \in \mathcal{X}$ there exists a tight maximal $\mathcal{N}$-chain from x to x'.*

Theorem 4. *Let $\mathcal{N} \subseteq 2^{\mathcal{X}}$. If K satisfies $d_{\mathcal{X}}$-privacy then for all $N \in \mathcal{N}$, all priors π on $\mathcal{X}$, and all $Z \in \mathcal{F}_{\mathcal{Z}}$:*

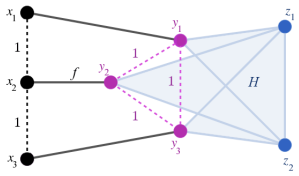
$$d_{\mathcal{P}}(\pi_{|N}, \sigma_{|N}) \leq d_{\mathcal{X}}(N) \qquad \textit{where} \qquad \sigma = \mathbf{Bayes}(\pi, K, Z)$$

If $\mathcal{N}$ is maximally tight then the converse also holds.

Using meaningful (and maximally tight) sets $\Phi, \mathcal{N}$, and applying the above characterizations, we can get an intuitive understanding of the privacy guarantees offered by $d_{\mathcal{X}}$-privacy. For example, in the case of databases, it can be shown that $\mathcal{N}_h$ is maximally tight wrt the d_h metric, hence the characterization of Theorem 2 can be obtained as a special case of Theorem 4. Theorem 1 can also be obtained from Theorem 3 (even though Φ_h is not maximally tight) since it only states an implication in one direction.

4 Answering Queries

To obtain the answer to a query $f : \mathcal{X} \to \mathcal{Y}$ in a private way, we can compose it with a mechanism $H : \mathcal{Y} \to \mathcal{P}(\mathcal{Z})$, thus obtaining an oblivious mechanism $H \circ f$:



	z_1	z_2
y_1	3/4	1/4
y_2	1/2	1/2
y_3	1/4	3/4

Fig. 1. Counterexample to the converse of Fact 5. The table represents the distribution H. We note that $H \circ f$ satisfies $(\ln 2)$-privacy, and that f is 1-sensitive. However $H(y_1)(z_1) = 3/4 \not\leq 2\, H(y_3)(z_1) = 2\, 1/4$, hence H does not satisfy $(\ln 2)$-privacy.

$\mathcal{X} \to \mathcal{P}(\mathcal{Z})$. In this section, we first state the standard compositionality result about the privacy of $H \circ f$, relying on the notion of Δ-*sensitivity* (aka Lipschitz continuity), naturally extended to the case of $d_{\mathcal{X}}$-privacy. Then, we introduce the concept of *uniform* sensitivity, and we use it to obtain the converse of the aforementioned compositionality result, which in turn allows to give optimality results later in the paper.

Definition 4. *f is Δ-sensitive wrt $d_{\mathcal{X}}, d_{\mathcal{Y}}$ iff $d_{\mathcal{Y}}(f(x), f(x')) \leq \Delta\, d_{\mathcal{X}}(x, x')$ for all $x, x' \in \mathcal{X}$. The smallest such Δ (if exists) is called the* sensitivity *of f wrt $d_{\mathcal{X}}, d_{\mathcal{Y}}$.*

Fact 5. *Assume that f is Δ-sensitive wrt $d_{\mathcal{X}}, d_{\mathcal{Y}}$ and H satisfies $d_{\mathcal{Y}}$-privacy. Then $H \circ f$ satisfies $\Delta d_{\mathcal{X}}$-privacy.*

Note that it is common to define a family of mechanisms $H_\epsilon, \epsilon > 0$, instead of a single one, where each H_ϵ satisfies privacy for a scaled version $\epsilon d_{\mathcal{Y}}$ of a metric of interest $d_{\mathcal{Y}}$. Given such a family and a query f, we can define a family of oblivious mechanisms $K_\epsilon = H_{\epsilon/\Delta} \circ f, \epsilon > 0$, each satisfying $\epsilon d_{\mathcal{X}}$-privacy (from Fact 5).

The converse of the above result does not hold in general, see Fig. 1 for a counterexample. However, it does hold if we replace the notion of sensitivity by the stronger notion of *uniform sensitivity*.

Definition 5. *Two elements $y, y' \in \mathcal{Y}$ are called Δ-expansive iff $d_{\mathcal{Y}}(y, y') = \Delta d_{\mathcal{X}}(x, x')$ for some $x \in f^{-1}(y), x' \in f^{-1}(y')$. A chain $\tilde{y}$ is Δ-expansive iff all steps y_i, y_{i+1} are Δ-expansive. Finally, f is uniformly Δ-sensitive iff it is Δ-sensitive and for all $y, y' \in \mathcal{Y}$ there exists a tight and Δ-expansive chain from y to y'.*

Theorem 6. *Assume that f is uniformly Δ-sensitive wrt $d_{\mathcal{X}}$, $d_{\mathcal{Y}}$. Then H satisfies $d_{\mathcal{Y}}$-privacy if and only if $H \circ f$ satisfies $\Delta d_{\mathcal{X}}$-privacy.*

4.1 Laplace Mechanisms

Adding Laplace noise is the most widely used technique for achieving differentia privacy. The mechanism can be naturally adapted to any metric, using a variant of the exponential mechanism [14], by providing a properly constructed scaling function. Note that in the framework of d-privacy, we can express the privacy of the mechanism itself, on its own domain, without the need to consider a query or a notion of sensitivity.

(i)	$\mathcal{Y} \subset \mathbb{R}, \quad \mathcal{Z} = \mathbb{R}$	$d_{\mathcal{Y}} = \epsilon d_{\mathbb{R}}$	$\lambda_\epsilon(z) = \frac{\epsilon}{2}$
(ii)	$\mathcal{Y} \subset \mathbb{R}^2, \ \mathcal{Z} = \mathbb{R}^2$	$d_{\mathcal{Y}} = \epsilon d_2$	$\lambda_\epsilon(z) = \frac{\epsilon^2}{2\pi}$
(iii)	$\mathcal{Y} \subset \mathbb{R}^2, \ \mathcal{Z} = \mathbb{R}^2$	$d_{\mathcal{Y}} = \epsilon d_1$	$\lambda_\epsilon(z) = \frac{\epsilon^2}{4}$
(iv)	$\mathcal{Y} = \mathcal{Z} = q[0..k]$	$d_{\mathcal{Y}} = \epsilon d_{\mathbb{R}}$	$\lambda_\epsilon(z) = \begin{cases} \frac{e^{q\epsilon}}{e^{q\epsilon}+1} & z \in \{0, qk\} \\ \frac{e^{q\epsilon}-1}{e^{q\epsilon}+1} & 0 < z < qk \end{cases}$

Fig. 2. Instantiations of the Laplace mechanism

Definition 6. *Let $\mathcal{Y}, \mathcal{Z}$ be two sets, and let $d_{\mathcal{Y}}$ be a metric on $\mathcal{Y} \cup \mathcal{Z}$. Let $\lambda : \mathcal{Z} \to [0, \infty)$ be a scaling function such that $D(y)(z) = \lambda(z)\, e^{-d_{\mathcal{Y}}(y,z)}$ is a pdf for all $y \in \mathcal{Y}$ (i.e. $\int_{\mathcal{Z}} D(y)(z) d\nu(z) = 1$). Then the mechanism $L : \mathcal{Y} \to \mathcal{P}(\mathcal{Z})$, described by the pdf D, is called a* Laplace mechanism *from $(\mathcal{Y}, d_{\mathcal{Y}})$ to $\mathcal{Z}$.*

Fact 7 ([14]). *Any Laplace mechanism from $(\mathcal{Y}, d_{\mathcal{Y}})$ to $\mathcal{Z}$ satisfies $d_{\mathcal{Y}}$-privacy.*

Figure 4.1 provides instantiations of the general definition for various choices of $\mathcal{Y}, \mathcal{Z}$ and $d_{\mathcal{Y}}$ used in the paper, by properly adjusting $\lambda(z)$. The basic case (i) is that of the one-dimensional continuous Laplace mechanism. Similarly, we can define a two-dimensional continuous Laplace mechanism (used in Section 6.2), measuring the distance between points by either the Euclidean (ii) or the Manhattan (iii) metric. In the discrete setting, we obtain the Truncated Geometric mechanism TG_ϵ [7], given by (iv), using a quantized set of reals as input. We denote by $q[0..k]$ the set $\{qi \mid i \in 0..k\}$, i.e. the set of $k+1$ quantized reals with step size $q > 0$.

4.2 Mechanisms of Optimal Utility

Answering a query privately is useless if the consumer gets no information about the real answer, thus it is crucial to analyze the mechanism's utility. We consider consumers applying Bayesian inference to map the mechanism's output to a guess that minimizes their expected loss. A consumer is characterized by a prior π on the set of secrets, and a loss function l (assumed to be monotone wrt a metric of reference, which is always $d_{\mathbb{R}}$ for the needs of this paper). The utility $\mathcal{U}(H, \pi, l)$ of a mechanism H for such a consumer is given by the expected loss (under an optimal remap strategy). This is the Bayesian notion of utility [7], but our results can be extended to risk-averse consumers.

A natural question to ask, then, is whether, for a *given query f*, there exists a mechanism that universally (i.e. for all priors and loss functions) provides optimal utility. Let $\mathcal{H}_f(d_{\mathcal{X}})$ be the set of all mechanisms $H : \mathcal{Y} \to \mathcal{Z}$ (for any $\mathcal{Z}$) such that $H \circ f$ satisfies $d_{\mathcal{X}}$-privacy. All mechanisms in $\mathcal{H}_f(d_{\mathcal{X}})$ can be used to answer f privately, hence we are interested in the one that maximizes utility.

Definition 7. *A mechanism $H \in \mathcal{H}_f(d_{\mathcal{X}})$ is f-$d_{\mathcal{X}}$-optimal iff $\mathcal{U}(H, \pi, l) \geq \mathcal{U}(H', \pi, l)$ for all $H' \in \mathcal{H}_f(d_{\mathcal{X}})$, all priors π and all loss functions l.*

The existence of (universally) optimal mechanisms is far from trivial. For standard differential privacy, a well-known result from [7] states that such a mechanism does exist for *counting* queries, i.e. those of the form "how many users satisfy property P".

Theorem 8 ([7]). *Let $\mathcal{Y} = [0..k]$ and let $f : \mathcal{V}^n \to \mathcal{Y}$ be a counting query. Then the TG_ϵ mechanism with input $\mathcal{Y}$ is f-ϵd_h-optimal for all $\epsilon > 0$.*

On the other hand, a well-known impossibility result [8] states that counting queries are essentially the only ones for which an optimal mechanism exists. This result is based on the concept of the *induced graph* $\sim_f$ of a query $f : \mathcal{V}^n \to \mathcal{Y}$, defined as: $y \sim_f y'$ iff $\exists x \sim_h x'$ s.t. $f(x) = y, f(x') = y'$.

Theorem 9 ([8]). *Let $f : \mathcal{V}^n \to \mathcal{Y}$ be a query such that $\sim_f$ is not a path graph. Then no f-ϵd_h-optimal mechanism exists for any $\epsilon < \ln 2$.*

Thus, most interesting queries, e.g. the sum and average, have no optimal mechanisms.

However, the above negative result and the concept of the induced graph are tied to the Hamming metric d_h. This raises the question of whether this special status of counting queries holds for any metric $d_\mathcal{X}$. To answer this question, we give a sufficient condition for showing the optimality of TG_ϵ for an arbitrary query f and metric $d_\mathcal{X}$, based on the concept of uniform sensitivity.

Theorem 10. *Let $\mathcal{Y} = q[0..k]$ and assume that $f : \mathcal{X} \to \mathcal{Y}$ is uniformly Δ-sensitive wrt $d_\mathcal{X}, d_\mathbb{R}$. Then the TG_ϵ mechanism with input $\mathcal{Y}$ is f-$\Delta d_\mathcal{X}$-optimal.*

In the following sections we show that this condition is indeed satisfied by several important queries, including the sum and average, for various metrics of interest.

5 Privacy in Statistical Databases

In this section, we investigate privacy notions in the context of statistical databases, other than the standard differential privacy. In contrast to the Hamming distance, which can be defined independently from the structure of the universe $\mathcal{V}$, we are interested in metrics that depend on the actual values and the distance between them. To this end, we assume that the universe is equipped with a metric $d_\mathcal{V}$, measuring how far apart two values are. When the universe is numeric (i.e. $\mathcal{V} \subset \mathbb{R}$) then $d_\mathcal{V} = d_\mathbb{R}$ is the natural choice. In the case of null values, we can extend a metric $d_\mathcal{V}$ from $\mathcal{V}$ to $\mathcal{V}_\varnothing$ by considering $\varnothing$ to be maximally distant from all other values, that is taking $d_\mathcal{V}(\varnothing, v) = d_\mathcal{V}(\mathcal{V}), v \in \mathcal{V}$. Note that this construction preserves the maximum distance between values, i.e. $d_\mathcal{V}(\mathcal{V}_\varnothing) = d_\mathcal{V}(\mathcal{V})$.

The first metric we consider, the normalized Manhattan metric, allows to strengthen differential privacy, obtaining a notion that not only protects the value of an individual, but also offers higher protection to small modifications of a value. Then we relax this metric, to obtain a weaker notion, that only protects the "accuracy" of an individual's value, but offers higher utility.

5.1 The Normalized Manhattan Metric

Differential privacy provides indistinguishability between databases differing in a single individual, but the level of distinguishability is independent from the actual value in those databases. Consider for example a database with salary information, and two adjacent databases $x \sim_i x'$ ($\sim_i$ denoting that they differ only in the value of the i-th individual) with $x[i] = v, x'[i] = v'$. A differentially private mechanism offers distinguishability level $\varepsilon(x, x') = \epsilon$, independently from v, v'. This means that when $v = 0, v' = 1$M, the indistinguishability level between x, x' will be the same as in the case $v = 20$K, $v' = 20.001$K.

One might expect, however, to have better protection in the second case, since the change in the individual's data is insignificant. Being insensitive to such small changes seems a reasonable privacy requirement since many queries (e.g. sum, average, etc) are themselves insensitive to small perturbations. The equal treatment of values is particularly problematic when we are obliged to use a "weak" ϵ, due to a high sensitivity. In this case, all values are only guaranteed to be weakly protected, while we could expect that at least close values would still enjoy high protection.

The normalized Manhattan metric $\widetilde{d}_1$ expresses exactly this idea. Databases differing in a single value have distance at most 1, but the distance can be substantially smaller for small modifications of values, offering higher protection in those cases. The Manhattan metric d_1 on $\mathcal{V}^n$ and its normalized version $\widetilde{d}_1$ are defined as:[8] $d_1(x, x') = \sum_{i=1}^{n} d_{\mathcal{V}}(x[i], x'[i])$ and $\widetilde{d}_1(x, x') = \frac{d_1(x,x')}{d_{\mathcal{V}}(\mathcal{V})}$. Similarly to differential privacy, we use a scaled version $\epsilon\widetilde{d}_1$ of the metric, to properly adjust the distinguishability level.

Concerning the operational characterizations of Section 3, the hiding functions and neighborhoods suitable for this metric are:

$$\phi_{i,w} = x[^{w(x[i])}/_i] \text{ for } w : \mathcal{V} \to \mathcal{V} \qquad N_{i,V}(x) = \{x[^v/_i] \mid v \in V\}$$
$$\Phi_1 = \{\phi_{i,w} \mid i \in 1..n, w : \mathcal{V} \to \mathcal{V}\} \qquad \mathcal{N}_1 = \{N_{i,V}(x) \mid x \in \mathcal{V}^n, i \in 1..n, V \subseteq \mathcal{V}\}$$

A hiding function $\phi_{i,w}$ replaces the value of individual i by applying an arbitrary substitution of values w (instead of replacing with a fixed value as $\phi_{i,v}$ does). Moreover, for the adversary, knowing $N_{i,V}(x)$ means that he knows the values of all individuals in the database but i, and moreover he knows that the value of i lies within V. Note that $\Phi_h \subset \Phi_1$ and $\mathcal{N}_h \subset \mathcal{N}_1$. We show that $\Phi_1, \mathcal{N}_1$ are "canonical".

Proposition 2. *$\Phi_1, \mathcal{N}_1$ are maximally tight wrt both $d_1, \widetilde{d}_1$.*

From Theorem 3, we conclude that $\epsilon\widetilde{d}_1$-privacy is equivalent to requiring that the adversary's posterior distributions with or without hiding i's value should be at most $2\epsilon\widetilde{d}_1(\phi_{i,w})$ distant. Since $\widetilde{d}_1(\phi_{i,w}) \leq 1$, hiding the individual's value in any way has small effect on the adversary's conclusions. But if i's value is replaced by one close to it, $\widetilde{d}_1(\phi_{i,w})$ can be much lower than 1, meaning that the effect on the adversary's conclusions is even smaller.

[8] Note that in the differential privacy literature, the d_1 distance is often used on *histograms*. This metric is closely related to the standard d_h distance on $\mathcal{V}^n$ (it depends only on the record counts), and different than d_1 on $\mathcal{V}^n$ which depends on the actual values.

Then, from Theorem 4 we conclude that $\epsilon\widetilde{d}_1$-privacy is equivalent to requiring that, for an informed adversary knowing the value of all individuals but i, and moreover knowing that i's value lies in V, his conclusions differ from his initial knowledge by at most $\epsilon\frac{d_{\mathcal{V}}(V)}{d_{\mathcal{V}}(\mathcal{V})}$. This difference is at most ϵ, but can be much smaller if values in V are close to each other, meaning that for an adversary who knows i's value with high accuracy, the gain is even smaller.

Intuitively, $\epsilon\widetilde{d}_1$-privacy offers a stronger notion of privacy than ϵd_h-privacy:

Proposition 3. *$\widetilde{d}_1 \leq d_h$, thus $\epsilon\widetilde{d}_1$-privacy implies ϵd_h-privacy.*

Since distances in $\widetilde{d}_1$ can be smaller than those in d_h, the sensitivity of a query wrt $\widetilde{d}_1$ is in general greater than the sensitivity wrt d_h, which means that to achieve $\epsilon\widetilde{d}_1$-privacy we need to apply more noise. However, for a general class of queries, it turns out that the two sensitivities coincide.

Definition 8. *A query f belongs to the family $\mathcal{C}$ iff $d_{\mathbb{R}}(f(x), f(x')) \leq d_{\mathcal{V}}(x[i], x'[i])$ for all $i \in 1..n$, $x \sim_i x' \in \mathcal{V}^n$, and moreover $\exists x \sim_i x' \in \mathcal{V}^n$ such that $d_{\mathbb{R}}(f(x), f(x')) = d_{\mathcal{V}}(\mathcal{V})$.*

Proposition 4. *Let $f \in \mathcal{C}$. The sensitivity of f wrt both $d_h, d_{\mathbb{R}}$ and $\widetilde{d}_1, d_{\mathbb{R}}$ is $d_{\mathcal{V}}(\mathcal{V})$.*

Intuitively, the class $\mathcal{C}$ contains queries for which the sensitivity is obtained for values that are maximally distant. For those queries, using the Truncated Geometric mechanism we can achieve a notion of privacy stronger than differential privacy *using the same amount of noise*!

Results About Some Common Queries. We now focus to some commonly used queries, namely the sum, average and p-percentile queries. Note that other commonly used queries such as the max, min and median queries are specific cases of the p-percentile query. In the following, we assume that the universe is $\mathcal{V} = q[0..k]_{\varnothing}$ with metric $d_{\mathbb{R}}$, and take $\mathcal{X} = \mathcal{V}^n \setminus \{\langle\varnothing, \ldots, \varnothing\rangle\}$, that is we exclude the empty database so that the queries can be always defined.

For these queries we obtain two results: first, we show that they belong to the $\mathcal{C}$ family, which means that we can achieve $\epsilon\widetilde{d}_1$-privacy via the TG_ϵ mechanism, using the same amount of noise that we would need for standard differential privacy.

Proposition 5. *The* $\mathrm{sum}, \mathrm{avg}, p\text{-perc}$ *queries belong to $\mathcal{C}$.*

More interestingly, we can show that the Truncated Geometric mechanism is in fact universally optimal wrt $\widetilde{d}_1$ for such queries.

Theorem 11. *The* $\mathrm{sum}, \mathrm{avg}$ *and* $p\text{-perc}$ *queries are all uniformly qk-sensitive wrt $\widetilde{d}_1, d_{\mathbb{R}}$.*

Corollary. *$TG_{\epsilon/qk}$ is f-$\epsilon\widetilde{d}_1$-optimal for $f \in \{\mathrm{sum}, \mathrm{avg}, p\text{-perc}\}$, $\epsilon > 0$.*

5.2 The Manhattan Metric

In the previous section, we used the normalized Manhattan metric $\epsilon\widetilde{d_1}$, obtaining a strong privacy notion that protects an individual's value, while offering even stronger protection for small changes in an individual's value. This however, requires at least as much noise as standard differential privacy.

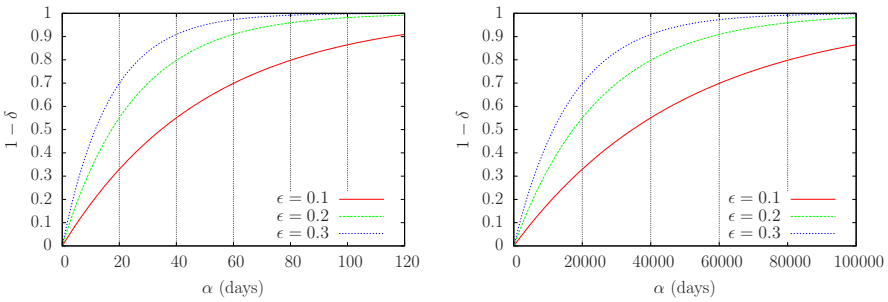
On the other hand, there are applications in which a complete protection of an individual's value is not required. This happens, for instance, in situations when the actual value is not sensitive, but knowing it with high accuracy might allow an adversary to identify the individual. Consider for example a database with the individuals' birthday, or the registration date and time to some social network. This information, by itself, might not be considered private, however knowing such information with minute-accuracy could easily allow to identify an individual. In such situations we might wish to protect only the accuracy of the value, thus achieving privacy with less noise and offering more accurate results.

This can be achieved by the Manhattan metric ϵd_1 (without normalization). This metric might assign a level of distinguishability higher than ϵ for adjacent databases, thus the privacy guarantees could be weaker than those of ϵ-differential privacy. However, adjacent databases with small changes in value will be highly protected, thus an adversary cannot infer an individual's value with accuracy.

Similarly to the previous section, we can obtain characterizations of ϵd_1-privacy using the same hiding functions Φ_1 and neighborhoods $\mathcal{N}_1$. The only difference is that $\epsilon d_1(\phi_{i,w})$ and $\epsilon d_1(N_{i,V})$ can be now higher than ϵ, offering weaker protection. However, when the adversary already knows i's value with high accuracy, meaning that values in V are close to each other, it is guaranteed that his knowledge will increase by a small factor (possibly even smaller than ϵ), ensuring that he cannot infer the value with even higher accuracy.

Note that the sensitivity of a query can be substantially lower wrt d_1 than wrt d_h. For example, the sum query is 1-sensitive wrt d_1 but qr-sensitive wrt d_h. This means that the noise we need to add could be substantially lower, offering better utility at the expense of lower privacy, but still sufficient for a given application.

Example 1. Consider a database containing the registration date on some social network, expressed as the number of days since Jan 1, 2000. We want to privately release the earliest registration date among individuals satisfying some criteria. A registration date itself is not considered sensitive, however from the result of the query it should be impossible to infer whether a particular individual belongs to that set. Since values can range between 0 and approximately 5.000, the sensitivity of the min query wrt d_h is 5.000, while wrt d_1 it is only 1. By using ϵd_h we protect (up to the intended level ϵ) an individual's registration date within the whole range of values, while by using $\frac{\epsilon}{5}d_1$ we provide the intended protection only within a radius of 5 days. More precisely: in the first case two adjacent databases will always have distinguishability level ϵ, while in the second case such level of protection is guaranteed only if the individual's registration date differs by at most 5 days in the two databases (if they differ more the distinguishability level will increase proportionally). The second case, of course, offers less privacy, but, depending on the application, confusion within 5 days can be enough



(a) $\frac{\epsilon}{5}d_1$-privacy (b) ϵd_h-privacy

Fig. 3. Utility for various values of ϵ

to prevent an individual from being identified. On the other hand, the trade-off with utility can be much more favorable in the second case: In Figure 1 we show the utility of a Laplace mechanism for both metrics, in terms of (α, δ)-usefulness (meaning that the mechanism reports a result within distance α from the real value with probability at least $1 - \delta$).[9] Clearly, $\frac{\epsilon}{5}d_1$-privacy gives acceptable utility while ϵd_h-privacy renders the result almost useless.

Finally, the optimality result from the previous section also holds for d_1.

Theorem 12. *The* $\mathrm{sum}, \mathrm{avg}$ *and* p-perc *queries are all uniformly* 1*-sensitive wrt* $d_1, d_{\mathbb{R}}$.

Corollary. TG_ϵ *is* f-ϵd_1*-optimal for* $f \in \{\mathrm{sum}, \mathrm{avg}, p\text{-}\mathrm{perc}\}$, $\epsilon > 0$.

6 Privacy in Other Contexts

6.1 Smart Meters

A smart meter is a device that records the consumption of electrical energy at potentially very short time intervals, and transmits the information to the utility provider, thus offering him the capability to monitor consumption accurately and almost in real-time.

The Problem. Although smart meters can help improving energy management, they create serious privacy threats: By analyzing accurate consumption data, thanks to appliance signature libraries it is possible to identify which electric devices are being used [15]. It has even been shown that, depending on the granularity of measurement and the resolution of data, it is possible to deduce what TV channels, and which movies are being watched [16].

Several papers addressed the privacy problems of smart metering in the recent past. The solution proposed in [17] is based on the use of techniques of (standard) differential privacy in order to send sanitized sums of the readings over some period of time (e.g. an hour, a day, a month) to the service provider. Since this solution is tailored to the use of smart metering for billing purposes, the noise added is assumed to be positive.

[9] Using Bayesian utility leads to similar results.

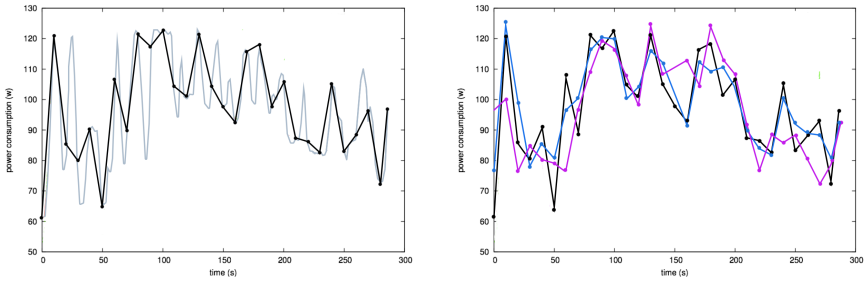
The Model. For the sake of generality, we assume here that the noise could be of any kind (not necessarily positive). We can regard the readings over the period $[1..n]$ as a tuple $x \in \mathcal{V}^n$, so that $x[i]$ represents the reading at the time i. Since [17] uses the standard differential privacy framework, the distinguishability metric on these tuples is assumed to be the Hamming distance, and therefore the privacy mechanism is tuned to protect the value of $x[i]$, regardless of whether the variation of this value is small or large. However, the solution proposed in [17] is general and can be adapted to a different distinguishability metric.

We argue that for the case of smart meters, the problem that derives from the extreme accuracy of the readings can be addressed with limited noise by adopting a metric that is sensitive also to the distance between values, and not only to the change of the value for a reading $x[i]$. The reason is the same as illustrated in previous section: if we want to protect small variations in the reading of $x[i]$, it is not a good idea to tune the sensitivity on the difference between the extremes values, because we would end up introducing a lot of noise. In fact, the experiments in [16] are performed on actual smart meters that are in the process of being deployed. These meters send readings to the service provider every 2 seconds. The solution proposed in [17] offers good privacy guarantees by completely protecting each measurement. However, such a definition is too strong if reporting values at short intervals is a requirement. With standard differential privacy, we cannot hope to fully protect each measurement without introducing too much noise. On the other hand, using a more relaxed metric, we can at least provide a meaningful privacy guarantee by protecting the accuracy of the values. Some privacy will still be lost, but the attacks described above where the individual's behaviour is completely disclosed, will be prevented.

The Manhattan distance d_1 on $\mathcal{V}^n$, however, is not suitable to model the privacy problem we have here: in fact d_1 is suitable to protect an individual $x[i]$ and its value, while here we want to protect *all the values at the same time*. This is because the adversary, i.e., the service provider, already knows *an approximation of all values*. Note the difference from the case of Section 5: there, the canonical adversary knows all exact values except $x[i]$, and for $x[i]$ he only knows an approximate value. (In the case of standard differential privacy, the canonical adversary knows all values except $x[i]$, and for $x[i]$ he does not even know an approximate value.)

The suitable distance, in this case, is the maximum distance between components, d_∞. In fact, we should consider x, x' "indistinguishable enough" (i.e. $d(x, x') \leq \delta$, for a certain δ) if and only if for each component i, $x[i], x'[i]$ are "indistinguishable enough" (i.e. $d(x[i], x'[i]) \leq \delta$, for the same δ). It is easy to see that the only distance that satisfies this property is $d(x, x') = d_\infty(x, x') = \max_i d_\nu(x[i], x'[i])$.

Example 2. We illustrate the application our method to distort the digital signature of a tv program. The grey line in Fig. 4(a) represents the energy consumption of the first 5 minutes of Star Trek 11 [15]. The black line is (the approximation of) the signature produced by a smart meter that reports the true readings every 10 seconds (the samples are represented by the dots). The blue and the magenta dots in 4(b) are obtained by adding laplacian noise to the true readings, with ϵ values .1 and .5 respectively. As we can see, especially in the case of $\epsilon = .5$, the signature is not recognizable.



(a) (b)

Fig. 4. Digital signature of a tv program (a) and its noisy reporting (b).

Concerning the characterization results, we use hiding functions substituting the value of all readings. Moreover, we use neighborhoods modelling an adversary that knows all readings with some accuracy, i.e. knows that each reading i lies within V_i.

$$\begin{aligned}\Phi_\infty &= \{\phi_{1,w_1} \circ \ldots \circ \phi_{n,w_n} \mid w_i : \mathcal{V} \to \mathcal{V}\ \forall i \in 1..n\}\\ N_{\{V_i\}} &= \{\langle v_1, \ldots, v_n\rangle \mid v_i \in V_i, i \in 1..n\}\\ \mathcal{N}_\infty &= \{N_{\{V_i\}} \mid V_i \subseteq \mathcal{V}, i \in 1..n\}\end{aligned}$$

We can show that $\Phi_\infty, \mathcal{N}_\infty$ are maximally tight.

Finally, we show that TG_ϵ is universally optimal for avg and p-perc.

Theorem 13. *The* avg *and* p-perc *queries are both uniformly* 1*-sensitive wrt* $d_\infty, d_\mathbb{R}$.

Corollary. *TG_ϵ is f-ϵd_∞-optimal for $f \in \{\text{avg}, p\text{-perc}\}$, $\epsilon > 0$.*

6.2 Geolocation

In this subsection we briefly describe an application of our framework to privacy-aware location-based systems. We refer to [18] for more details.

Privacy notions have been already studied in previous works. Some of these works [19,20,21] propose the use of the *expectation of distance error* of the attacker as the way to quantify the privacy offered by a mechanism. Others works [22,23,24] rely on the well-known concept of k-anonymity. The notion of *relevance* is also used to measure location privacy in [25]. A strong advantage of the use of d-privacy as privacy notion is that it abstracts from the side-knowledge of the attacker.

The Problem. In several situations it is desirable to know the location of an individual or a group of individuals in order to provide a service. For instance: In census-based statistics, to determine the population density in certain areas, in transportation industry, to estimate the average number of people who need to travel between two given stations, and in smartphone applications, to obtain points of interest nearby such as restaurants.

Due to privacy concerns, an individual may refuse to disclose his exact location to the service provider. Nevertheless, he may be willing to reveal approximate location information.It is worth noting that for several location-based systems it is usually enough to obtain an approximate location to be able to provide an accurate service. Note however, that in order to guarantee a non-negligible level of privacy, the random location cannot be generated naively. Therefore, if we want to develop a method to randomize location coordinates, we have to understand what kind of privacy the user expects to have, and how much information he is willing to reveal.

The Model. In this scenario, the privacy level depends on the accuracy with which an attacker can guess an individual's location from the reported one. We will therefore aim for a distance-dependent notion of privacy, requiring points that are close in distance to each other to be *indistinguishable* from the attacker's point of view. Our method will still allow the service provider to distinguish between points that are far from each other.

We consider the problem of geolocation on the Euclidean plane, which is a good approximation of the Earth surface when the area is not "too large". In this scenario, possible locations of an individual will be modeled with a set $\mathcal{X} \subseteq \mathbb{R}^2$, and possible reported values will be represented by a set $\mathcal{Z} \subseteq \mathbb{R}^2$. The metric $d_{\mathcal{X}}$ used in this context will be the Euclidean distance d_2.

Concerning the characterizations of Section 3, any function $\phi : \mathbb{R}^2 \to \mathbb{R}^2$ can be used as a hiding function. Moreover, a neighborhood can be any region $N \subseteq \mathbb{R}^2$, modelling an informed adversary who knows that the user is located within N. Hence we take $\Phi_2 = \mathbb{R}^2 \to \mathbb{R}^2$ and $\mathcal{N}_2 = 2^{\mathbb{R}^2}$, both of which are maximally tight wrt d_2.

In order to obtain a mechanism which satisfies ϵd_2-privacy, we can use the *Laplace* mechanism L_ϵ on $\mathbb{R}^2$ mentioned in Section 4.1, that is, the one described by the pdf $D_\epsilon(x)(z) = \frac{\epsilon^2}{2\pi} e^{-\epsilon d_2(x,z)} \quad x, z \in \mathbb{R}^2$. The results in Section 4.1 ensure that such mechanism satisfies ϵd_2-privacy.

7 Conclusion

Starting from the observation that differential privacy requires that the distinguishability of two databases depends on their Hamming distance, we have explored the consequences of extending this principle to arbitrary metrics. In this way we have obtained a rich framework suitable to model a large variety of privacy problems, and in domains other than statistical databases. Furthermore, even in statistical databases applications, whenever the privacy concern is related to disclosing small variations in the values of the individuals (rather than large ones), then our framework allows a more precise calibration of the noise necessary for achieving the intended level of privacy, and this results, in general, in a better utility than the one achievable under the constraint of standard differential privacy. We have investigated the trade-off between privacy and utility in this extended setting, and it turns out changing the metric has considerable implications on the existence of universally optimal mechanisms. In particular, for the Manhattan distance, the normalized Manhattan distance, and the max distance it is possible to define universally optimal mechanisms for several common queries like the

sum, the average, and the percentile. This contrast sharply with the case of standard differential privacy, where universally optimal mechanisms exist only for counting queries. Finally, we have shown the applicability of our framework to various privacy problems in different domains, including smart meters and geolocation.

References

1. Dwork, C.: Differential privacy. In: Proc. of ICALP. Volume 4052 of LNCS., Springer (2006) 1–12
2. Dwork, C., Mcsherry, F., Nissim, K., Smith, A.: Calibrating noise to sensitivity in private data analysis. In: Proc. of TCC. Volume 3876 of LNCS., Springer (2006) 265–284
3. Reed, J., Pierce, B.C.: Distance makes the types grow stronger: a calculus for differential privacy. In: Proc. of ICFP, ACM (2010) 157–168
4. Narayanan, A., Shmatikov, V.: De-anonymizing social networks. In: Proc. of S&P, IEEE (2009) 173–187
5. Machanavajjhala, A., Kifer, D., Abowd, J.M., Gehrke, J., Vilhuber, L.: Privacy: Theory meets practice on the map. In: Proc. of ICDE, IEEE (2008) 277–286
6. Ganta, S.R., Kasiviswanathan, S.P., Smith, A.: Composition attacks and auxiliary information in data privacy. In: Proc. of KDD, ACM (2008) 265–273
7. Ghosh, A., Roughgarden, T., Sundararajan, M.: Universally utility-maximizing privacy mechanisms. In: Proc. of STOC, ACM (2009) 351–360
8. Brenner, H., Nissim, K.: Impossibility of differentially private universally optimal mechanisms. In: Proc. of FOCS, IEEE (2010) 71–80
9. Nissim, K., Raskhodnikova, S., Smith, A.: Smooth sensitivity and sampling in private data analysis. In: Proc. of STOC, ACM (2007) 75–84
10. Barthe, G., Köpf, B., Olmedo, F., Béguelin, S.Z.: Probabilistic relational reasoning for differential privacy. In: Proc. of POPL, ACM (2012)
11. Dwork, C., Hardt, M., Pitassi, T., Reingold, O., Zemel, R.S.: Fairness through awareness. In: Proc. of ITCS, ACM (2012) 214–226
12. Chatzikokolakis, K., Andrés, Miguel, E., Bordenabe, Nicolás, E., Palamidessi, C.: Broadening the scope of Differential Privacy using metrics. Tech. rep., INRIA (2012) To appear in PETS 2013. Available at: `http://hal.inria.fr/hal-00767210`.
13. Dwork, C., Kenthapadi, K., McSherry, F., Mironov, I., Naor, M.: Our data, ourselves: Privacy via distributed noise generation. In: Proc. of EUROCRYPT. Volume 4004 of LNCS., Springer (2006) 486–503
14. McSherry, F., Talwar, K.: Mechanism design via differential privacy. In: Proc. of FOCS, IEEE (2007) 94–103
15. Lam, H., Fung, G., Lee, W.: A novel method to construct taxonomy electrical appliances based on load signatures. IEEE Trans. on Consumer Electronics **53**(4) (2007) 653—660
16. Greveler, U., Justus, B., Loehr, D.: Multimedia content identification through smart meter power use profiles. In: CPDP. (2012)
17. Danezis, G., Kohlweiss, M., Rial, A.: Differentially private billing with rebates. IACR Cryptology ePrint Archive **2011** (2011) 134
18. Andrés, M., Bordenabe, N., Chatzikokolakis, K., Palamidessi, C.: Geo-indistinguishability: Differential privacy for location-based systems. CoRR **abs/1212.1984** (2012)
19. Shokri, R., Theodorakopoulos, G., Boudec, J.Y.L., Hubaux, J.P.: Quantifying location privacy. In: Proc. of S&P, IEEE (2011) 247–262
20. Shokri, R., Theodorakopoulos, G., Troncoso, C., Hubaux, J.P., Boudec, J.Y.L.: Protecting location privacy: optimal strategy against localization attacks. In: Proc. of CCS, ACM (2012) 617–627

21. Hoh, B., Gruteser, M.: Protecting location privacy through path confusion. In: SecureComm, IEEE (2005) 194–205
22. Kido, H., Yanagisawa, Y., Satoh, T.: Protection of location privacy using dummies for location-based services. In: Proc. of ICDE Workshops. (2005) 1248
23. Shankar, P., Ganapathy, V., Iftode, L.: Privately querying location-based services with sybilquery. In: Proc. of UbiComp, ACM (2009) 31–40
24. Duckham, M., Kulik, L.: A formal model of obfuscation and negotiation for location privacy. In: Proc. of PERVASIVE. Volume 3468 of LNCS., Springer (2005) 152–170
25. Ardagna, C.A., Cremonini, M., Damiani, E., di Vimercati, S.D.C., Samarati, P.: Location privacy protection through obfuscation-based techniques. In: Proc. of DAS. Volume 4602 of LNCS., Springer (2007) 47–60

A Proofs of Section 3

Proposition 1. *If $d_{\mathcal{X}} \leq d_{\mathcal{X}}{}'$ (point-wise) then $d_{\mathcal{X}}$-privacy implies $d_{\mathcal{X}}{}'$-privacy.*

Proof. Trivial, since $d_{\mathcal{P}}(K(x), K(x')) \leq d_{\mathcal{X}}(x, x') \leq d_{\mathcal{X}}{}'(x, x')$. □

The following simple lemma states that bounding $d_{\mathcal{P}}$ is equivalent to the usual formulation of bounding the ratio between probabilities.

Lemma 1. *Let μ_1, μ_2 be probability measures on $\mathcal{Z}$. Then*

$$d_{\mathcal{P}}(\mu_1, \mu_2) \leq b \quad \textit{iff} \quad \forall Z \in \mathcal{F}_{\mathcal{Z}} : e^{-b}\mu_2(Z) \leq \mu_1(Z) \leq e^b \mu_2(Z)$$

Proof. iff) We have $|\ln \frac{\mu_1(Z)}{\mu_2(Z)}| \leq d_{\mathcal{P}}(\mu_1, \mu_2) \leq b$, hence $-b \leq \ln \frac{\mu_1(Z)}{\mu_2(Z)} \leq b$, which implies $e^{-b} \leq \frac{\mu_1(Z)}{\mu_2(Z)} \leq e^b$. if) We have that $|\ln \frac{\mu_1(Z)}{\mu_2(Z)}|$ is bounded from above by b, but $d_{\mathcal{P}}(\mu_1, \mu_2)$ is the least such bound hence $d_{\mathcal{P}}(\mu_1, \mu_2) \leq b$. □

The following Lemma shows the usefulness of *tight* chains.

Lemma 2. *Let $x_1, \ldots, x_n$ be a tight chain. If K satisfies $d_{\mathcal{X}}$-privacy on all adjacent elements of the chain, then it also satisfies it for x_1, x_n. That is*

$$d_{\mathcal{P}}(K(x_i), K(x_{i+1})) \leq d_{\mathcal{X}}(x_i, x_{i+1}) \quad \forall 1 \leq i < n$$

implies $d_{\mathcal{P}}(K(x_1), K(x_n)) \leq d_{\mathcal{X}}(x_1, x_n)$.

Proof. Using the fact that $d_{\mathcal{P}}$ is itself a metric, we have

$$\begin{aligned} d_{\mathcal{P}}(K(x_1), K(x_n)) &\leq \textstyle\sum_{i=1}^{n-1} d_{\mathcal{P}}(K(x_i), K(x_{i+1})) && \text{triangle ineq. for } d_{\mathcal{P}} \\ &\leq \textstyle\sum_{i=1}^{n-1} d_{\mathcal{X}}(x_i, x_{i+1}) && \text{hypothesis} \\ &= d_{\mathcal{X}}(x_1, x_n) && \text{tightness} \end{aligned}$$

□

Theorem 3. *Let Φ be a set of hiding functions. If K satisfies $d_{\mathcal{X}}$-privacy then for all $\phi \in \Phi$, all priors π on $\mathcal{X}$, and all $Z \in \mathcal{F}_{\mathcal{Z}}$:*

$$d_{\mathcal{P}}(\sigma_1, \sigma_2) \leq 2\, d_{\mathcal{X}}(\phi) \quad \textit{where} \;\; \sigma_1 = \mathbf{Bayes}(\pi, K, Z) \textit{ and } \sigma_2 = \mathbf{Bayes}(\pi, K \circ \phi, Z)$$

If Φ is maximally tight then the converse also holds.

Proof. Assume that K satisfies $d_{\mathcal{X}}$-privacy and let π be a prior, $\phi \in \Phi$ and $Z \in \mathcal{F}_{\mathcal{Z}}$. We need to show that

$$\forall x \in \mathcal{X} : e^{-2d_{\mathcal{X}}(\phi)}\sigma_1(x) \leq \sigma_2(x) \leq e^{2d_{\mathcal{X}}(\phi)}\sigma_1(x)$$

(then conclude by applying Lemma 1). Let $x \in \mathcal{X}$, we have:

$$\begin{aligned}
&\sigma_2(x) \\
&= \frac{(K \circ \phi)(x)(Z)\pi(x)}{\sum_{x' \in \mathcal{X}} (K \circ \phi)(x')(Z)\pi(x')} && \text{def. of } \mathbf{Bayes} \\
&= \frac{K(\phi(x))(Z)\pi(x)}{\sum_{x' \in \mathcal{X}} K(\phi(x'))(Z)\pi(x')} \\
&\leq \frac{e^{d_{\mathcal{X}}(x, \phi(x))} K(x)(Z)\pi(x)}{\sum_{x' \in \mathcal{X}} e^{-d_{\mathcal{X}}(x', \phi(x'))} K(x')(Z)\pi(x')} && d_{\mathcal{X}}\text{-privacy} \\
&\leq \frac{e^{d_{\mathcal{X}}(\phi)} K(x)\pi(x)(Z)}{e^{-d_{\mathcal{X}}(\phi)} \sum_{x' \in \mathcal{X}} K(x')(Z)\pi(x')} && d_{\mathcal{X}}(x, \phi(x)) \leq d_{\mathcal{X}}(\phi) \\
&\leq e^{2d_{\mathcal{X}}(\phi)}\sigma_1(x) && \text{def. of } \mathbf{Bayes}
\end{aligned}$$

and symmetrically for $\sigma_2(x) \geq e^{-2d_{\mathcal{X}}(\phi)}\sigma_1(x)$.

For the opposite direction, assume that Φ is maximally tight (Def 2), that $d_{\mathcal{P}}(\sigma_1, \sigma_2) \leq 2d_{\mathcal{X}}(\phi)$ holds for all π, ϕ, Z, but $d_{\mathcal{X}}$-privacy is violated for some $x, x' \in \mathcal{X}$. From Def 2, there exist a tight maximal Φ-chain $\tilde{x}$ from x to x'. Then from Lemma 2, we get that $d_{\mathcal{X}}$-privacy is also violated for some adjacent x_i, x_{i+1} in the chain, that is:

$$K(x_i)(Z) > e^{d_{\mathcal{X}}(x_i, x_{i+1})} K(x_{i+1})(Z) \qquad \text{for some } Z \tag{2}$$

We fix Z to the one above. Since $\tilde{x}$ is a maximal Φ-chain, there exists $\phi \in \Phi$ such that $\phi(x_i) = x_{i+1}, \phi(x_{i+1}) = x_i$ and $d_{\mathcal{X}}(x_i, x_{i+1}) = d_{\mathcal{X}}(\phi)$. Fixing this ϕ, we define a function $f : \mathcal{P}(\mathcal{X}) \to \mathbb{R}$ as follows:

$$f(\pi) = \frac{\sum_{x' \in \mathcal{X}} K(x')(Z)\pi(x')}{\sum_{x' \in \mathcal{X}} K(\phi(x'))(Z)\pi(x')}$$

Let $\delta(x)$ denote the Dirac measure assigning probability 1 to x, from (2) we have that

$$\begin{aligned}
f(\delta(x_i)) &= \frac{K(x_i)(Z)}{K(x_{i+1})(Z)} > e^{-d_{\mathcal{X}}(x_i, x_{i+1})} \\
f(\delta(x_{i+1})) &= \frac{K(x_{i+1})(Z)}{K(x_i)(Z)} < e^{-d_{\mathcal{X}}(x_i, x_{i+1})}
\end{aligned}$$

From the continuity of f on the line between $\delta(x_i)$ and $\delta(x_{i+1})$, there exists a prior $\pi = t\delta(x_i)+(1-t)\delta(x_{i+1}), t \in (0,1)$, such that $f(\pi) = e^{-d_{\mathcal{X}}(x_i,x_{i+1})}$. Note that since π is distinct from $\delta(x_i), \delta(x_{i+1})$, it holds that $\pi(x_i) > 0, \pi(x_{i+1}) > 0$. By applying the hypothesis for this π, we get

$$\begin{aligned}
d_{\mathcal{P}}(\sigma_1,\sigma_2) &\leq 2d_{\mathcal{X}}(\phi) && \Rightarrow \\
\sigma_1(x_i) &\leq e^{2d_{\mathcal{X}}(\phi)}\sigma_2(x_i) && \text{(Lemma 1)} \Rightarrow \\
\frac{K(x_i)(Z)\pi(x_i)}{\sum_{x'\in\mathcal{X}} K(x')(Z)\pi(x')} &\leq e^{2d_{\mathcal{X}}(\phi)}\frac{K(\phi(x_i))(Z)\pi(x_i)}{\sum_{x'\in\mathcal{X}} K(\phi(x'))(Z)\pi(x')} && \text{(Def. of } \sigma_1,\sigma_2) \Rightarrow \\
K(x_i)(Z) &\leq e^{2d_{\mathcal{X}}(\phi)} f(\pi)K(\phi(x_i))(Z) && (\pi(x_i) > 0) \Rightarrow \\
K(x_i)(Z) &\leq e^{d_{\mathcal{X}}(x_i,x_{i+1})}K(x_{i+1})(Z)
\end{aligned}$$

which contradicts (2). □

Theorem 4. *Let $\mathcal{N} \subseteq 2^{\mathcal{X}}$. If K satisfies $d_{\mathcal{X}}$-privacy then for all $N \in \mathcal{N}$, all priors π on $\mathcal{X}$, and all $Z \in \mathcal{F}_{\mathcal{Z}}$:*

$$d_{\mathcal{P}}(\pi_{|N},\sigma_{|N}) \leq d_{\mathcal{X}}(N) \qquad \textit{where} \qquad \sigma = \mathbf{Bayes}(\pi,K,Z)$$

If $\mathcal{N}$ is maximally tight then the converse also holds.

Proof. Assume that K satisfies $d_{\mathcal{X}}$-privacy. We fix some $N \in \mathcal{N}, \pi \in \mathcal{P}(\mathcal{X}), Z \in \mathcal{F}_{\mathcal{Z}}$ and let $\sigma = \mathbf{Bayes}(\pi,K,Z)$. Note that $\pi_{|N},\sigma_{|N}$ are distributions on N. From Lemma 1 we need to show that

$$e^{-d_{\mathcal{X}}(N)}\pi_{|N}(x) \leq \sigma_{|N}(x) \leq e^{d_{\mathcal{X}}(N)}\pi_{|N}(x) \qquad \forall x \in N$$

Fixing some $x \in N$, we have:

$$\begin{aligned}
\sigma_{|N}(x) &= \sigma(x|N) && \text{def. of } \sigma_{|N} \\
&= \frac{\sigma(x)}{\sum_{x'\in N}\sigma(x')} \\
&= \frac{\pi(x)K(x)(Z)}{\sum_{x'\in N}\pi(x')K(x')(Z)} && \text{def. of } \mathbf{Bayes} \\
&\leq \frac{\pi(x)K(x)(Z)}{\sum_{x'\in N}\pi(x')e^{-d_{\mathcal{X}}(x,x')}K(x)(Z)} && d_{\mathcal{X}}\text{-privacy} \\
&\leq e^{d_{\mathcal{X}}(N)}\frac{\pi(x)}{\sum_{x'\in N}\pi(x')} && d_{\mathcal{X}}(x,x') \leq d_{\mathcal{X}}(N) \\
&= e^{d_{\mathcal{X}}(N)}\pi_{|N}(x)
\end{aligned}$$

and symmetrically for $\sigma_{|N}(x) \geq e^{-d_{\mathcal{X}}(N)}\pi_{|N}(x)$.

For the opposite direction, assume that $\mathcal{N}$ is maximally tight (Def 3) but $d_{\mathcal{X}}$-privacy is violated for some $x, x' \in \mathcal{X}$. From Def 3, there exist a tight $\mathcal{N}$-chain $\tilde{x}$ from x to x'.

Then from Lemma 2, we get that $d_{\mathcal{X}}$-privacy is also violated for some adjacent x_i, x_{i+1} in the chain, that is:

$$K(x_i)(Z) > e^{d_{\mathcal{X}}(x_i,x_{i+1})}K(x_{i+1})(Z) \qquad \text{for some } Z \tag{3}$$

Since $\tilde{x}$ is an $\mathcal{N}$-chain, there exist $N \in \mathcal{N}$ such that $\{x_i, x_{i+1}\} \subseteq N$ and $d_{\mathcal{X}}(x_i, x_{i+1}) = d_{\mathcal{X}}(N)$. We define a prior distribution $\pi_t(x)$ as

$$\pi_t(x) = \begin{cases} t & x = x_i \\ 1-t & x = x_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

Using that prior for $t > 0$, we fix some $Z \in \mathcal{F}_{\mathcal{Z}}$ and let $\sigma_t = \mathbf{Bayes}(\pi_t, K, Z)$. We have

$$\begin{aligned}
\sigma_{t|N}(x_i) &\leq e^{d_{\mathcal{X}}(N)}\pi_{t|N}(x_i) && \text{(hypoth., Lemma 1)} \Rightarrow \\
\sigma_t(x_i) &\leq e^{d_{\mathcal{X}}(N)}\pi_t(x_i) && (\pi_t(N) = \sigma_t(N) = 1) \Rightarrow \\
\frac{\pi_t(x_i)K(x_i)(Z)}{\sum_{x' \in \mathcal{X}} \pi_t(x')K(x')(Z)} &\leq e^{d_{\mathcal{X}}(N)}\pi_t(x_i) && \text{(def. of } \mathbf{Bayes}) \Rightarrow \\
\frac{tK(x_i)(Z)}{tK(x_i)(Z) + (1-t)K(x_{i+1})(Z)} &\leq e^{d_{\mathcal{X}}(N)}t && \text{(def. of } \pi_t) \Rightarrow \\
\frac{K(x_i)(Z)}{tK(x_i)(Z) + (1-t)K(x_{i+1})(Z)} &\leq e^{d_{\mathcal{X}}(N)} && (t > 0) \Rightarrow \\
\frac{K(x_i)(Z)}{tK(x_i)(Z) + (1-t)K(x_{i+1})(Z)} &\leq e^{d_{\mathcal{X}}(x_i,x_{i+1})} && (d_{\mathcal{X}}(x_i, x_{i+1}) = d_{\mathcal{X}}(N))
\end{aligned}$$

The above inequality holds for all $t > 0$. Finally, taking the $\lim_{t\to 0}$ on both sides we get

$$K(x_i)(Z) \leq e^{d_{\mathcal{X}}(x_i,x_{i+1})}K(x_{i+1})(Z)$$

which is a contradiction of (3). □

B Proofs of Section 4

Fact 5. *Assume that f is Δ-sensitive wrt $d_{\mathcal{X}}, d_{\mathcal{Y}}$ and H satisfies $d_{\mathcal{Y}}$-privacy. Then $H \circ f$ satisfies $\Delta d_{\mathcal{X}}$-privacy.*

Proof. Assume that H satisfies $d_{\mathcal{Y}}$-privacy and let $x, x' \in \mathcal{X}$. We have:

$$\begin{aligned}
d_{\mathcal{P}}((H \circ f)(x), (H \circ f)(x')) &= d_{\mathcal{P}}(H(f(x)), H(f(x'))) \\
&\leq d_{\mathcal{Y}}(f(x), f(x')) && d_{\mathcal{Y}}\text{-privacy} \\
&\leq \Delta d_{\mathcal{X}}(x, x') && \Delta\text{-sensitivity}
\end{aligned}$$

□

Theorem 6. *Assume that f is uniformly Δ-sensitive wrt $d_{\mathcal{X}}$, $d_{\mathcal{Y}}$. Then H satisfies $d_{\mathcal{Y}}$-privacy if and only if $H \circ f$ satisfies $\Delta d_{\mathcal{X}}$-privacy.*

Proof. The only if part is Fact 5. For the if part, fix some $y, y' \in \mathcal{Y}$ and let $y_1, \ldots, y_n$ be the tight Δ-expansive chain from y to y' guaranteed to exist by the definition of uniform Δ-sensitivity. Then, for all $1 \leq i < n$, since y_i, y_{i+1} are Δ-expansive, there exist

$$x \in f^{-1}(y_i), x' \in f^{-1}(y_{i+1}) \quad \text{such that} \quad d_{\mathcal{Y}}(f(x), f(x')) = \Delta\, d_{\mathcal{X}}(x, x')$$

Hence

$$\begin{aligned} d_{\mathcal{P}}(H(y_i), H(y_{i+1})) &= d_{\mathcal{P}}(H(f(x)), H(f(x'))) \\ &\leq \Delta d_{\mathcal{X}}(x, x') && \Delta d_{\mathcal{X}}\text{-privacy of } H \circ f \\ &= d_{\mathcal{Y}}(y_i, y_{i+1}) \end{aligned}$$

So H satisfies $d_{\mathcal{Y}}$-privacy for all adjacent elements in the chain, hence from Lemma 2 it also satisfies it for y, y'. □

The following result is standard, we provide the proof for completeness.

Fact 7 ([14]). *Any Laplace mechanism from $(\mathcal{Y}, d_{\mathcal{Y}})$ to $\mathcal{Z}$ satisfies $d_{\mathcal{Y}}$-privacy.*

Proof. For the pdf describing the mechanism we have:

$$\begin{aligned} D(y)(z) &= \lambda(z)\, e^{-d_{\mathcal{Y}}(y,z)} \\ &\leq \lambda(z)\, e^{-(d_{\mathcal{Y}}(y',z) - d_{\mathcal{Y}}(y,y'))} && \text{triangle ineq.} \\ &= e^{d_{\mathcal{Y}}(y,y')} \lambda(z)\, e^{-\epsilon d_{\mathcal{Y}}(y',z)} \\ &= e^{d_{\mathcal{Y}}(y,y')} D(y')(z) \end{aligned}$$

for all $y, y' \in \mathcal{Y}, z \in \mathcal{Z}$. The above inequality can be directly extended from the pdf to the measures, thus we conclude that the mechanism satisfies $d_{\mathcal{Y}}$-privacy. □

B.1 Proofs of Section 4.2

To prove our sufficient condition for optimality (Theorem 10), we introduce the concept of a mechanism being optimal not wrt a specific query, but wrt *the metric of its input domain*. Let $\mathcal{H}(d_{\mathcal{Y}})$ be the set of all mechanisms $H : \mathcal{Y} \to \mathcal{Z}$ (for any $\mathcal{Z}$) satisfying $d_{\mathcal{Y}}$-privacy.

Definition 9. *A mechanism $H \in \mathcal{H}(d_{\mathcal{Y}})$ is $d_{\mathcal{Y}}$-optimal iff $\mathcal{U}(H, \pi, l) \geq \mathcal{U}(H', \pi, l)$ for all $H' \in \mathcal{H}(d_{\mathcal{Y}})$, all priors π and all loss functions l.*

Notice the difference between $d_{\mathcal{Y}}$-optimal and f-$d_{\mathcal{X}}$-optimal; the latter refers to a specific query. The two notions can be related in the case of uniformly sensitive queries by the following result:

Proposition 6. *Assume that f is uniformly Δ-sensitive wrt $d_{\mathcal{X}}, d_{\mathcal{Y}}$. Then H is f-$\Delta d_{\mathcal{X}}$-optimal iff it is $d_{\mathcal{Y}}$-optimal.*

Proof. From uniform Δ-sensitivity and Theorem 6, we get that $\mathcal{H}(d_{\mathcal{Y}}) = \mathcal{H}_f(\Delta d_{\mathcal{X}})$. Then the result follows directly from the definition of optimality (Definitions 7,9). □

The importance of the induced graph $\sim_f$ in optimality results comes from the fact that f is always uniformly sensitive wrt the metric induced by $\sim_f$.

Proposition 7. *Let f be a query with induced graph $\sim_f$, and let d_f be the metric induced by $\sim_f$. Then f is uniformly 1-sensitive wrt d_h, d_f.*

Proof. Let $x, x' \in \mathcal{X}$ and let $n = d_h(x, x')$. We first need to show that f is 1-sensitive wrt d_h, d_f, that is $d_f(f(x), f(x')) \leq n$. Since d_h is induced by $\sim_h$, there exists a $\sim_h$-path $x_1 \dots, x_n$ such that $x = x_1, x' = x_n$. By definition of $\sim_f$ we have that $f(x_i) \sim_f f(x_{i+1})$, thus $f(x_1), \dots, f(x_n)$ is a $\sim_f$-path of length n from $f(x)$ to $f(x')$. Since $d_f(f(x), f(x'))$ is the length of the shortest such path, we have that $d_f(f(x), f(x')) \leq n$.

For the "uniformly" part, let $y, y' \in \mathcal{Y}$ and $n = d_f(y, y')$. We need to show that there exists a tight and 1-expansive chain from y to y'.

Since d_f is induced by $\sim_f$, there exist a $\sim_f$-path $\tilde{y} = y_1, \dots, y_n$ such that $y = y_1, y' = y_n$ This implies that $d_f(y_i, y_{i+1}) = 1$ and thus $d_f(\tilde{y}) = n = d_f(y, y')$ so the chain is tight.

Moreover, from the definition of $\sim_f$ we have that there exist $x \sim_h x'$ such that $f(x) = y_i, f(x') = y_{i+1}$, so $d_h(x, x') = 1$ which means that y_i, y_{i+1} are 1-expansive, and this happens for all $1 \leq i < n$ so the chain is 1-expansive. □

We can now show the optimality of TG_ϵ with input $q[0..k]$ wrt the $\epsilon d_{\mathbb{R}}$ metric, independently from any query.

Proposition 8. *Let $\mathcal{Y}_q = q[0..k]$. The TG_ϵ mechanism with input $\mathcal{Y}_q$ is $\epsilon d_{\mathbb{R}}$-optimal for all $\epsilon > 0$.*

Proof. Fix $\mathcal{Y} = 0..k$ and $\mathcal{Y}_q = q[0..k]$ for some $k \in \mathbb{N}, q > 0$, and let $TG_\epsilon(\mathcal{Y}), TG_\epsilon(\mathcal{Y}_q)$ denote the Truncated Geometric mechanisms with input $\mathcal{Y}, \mathcal{Y}_q$ respectively.

From Theorem 8 we known that $TG_\epsilon(\mathcal{Y})$ is f-ϵd_h-optimal when f is a counting query. For counting queries, d_f (the metric that corresponds to their induced graph) and $d_{\mathbb{R}}$ coincide, thus from Prop 7 we get that f is uniformly 1-sensitive wrt $d_h, d_{\mathbb{R}}$. Then from Prop 6 we have that that $TG_\epsilon(\mathcal{Y})$ is $\epsilon d_{\mathbb{R}}$-optimal. This mechanism has pdf

$$D_\epsilon(y)(z) = \lambda_\epsilon(z)\, e^{-\epsilon\, d_{\mathbb{R}}(y,z)} \quad \lambda_\epsilon(z) = \begin{cases} \frac{e^\epsilon}{e^\epsilon+1} & z \in \{0, k\} \\ \frac{e^\epsilon-1}{e^\epsilon+1} & 0 < z < k \end{cases}$$

We now show that $TG_\epsilon(\mathcal{Y}_q)$ is also $\epsilon d_{\mathbb{R}}$-optimal. The metric spaces $(\mathcal{Y}, \epsilon d_{\mathbb{R}})$ and $(\mathcal{Y}_q, \epsilon \frac{1}{q} d_{\mathbb{R}})$ are isometric. So we can obtain a mechanism D'_ϵ with input $\mathcal{Y}_q$ by replacing i with qi and $\epsilon d_{\mathbb{R}}$ with $\epsilon \frac{1}{q} d_{\mathbb{R}}$). The pdf of this mechanism is:

$$D'_\epsilon(y)(z) = \lambda_\epsilon(z)\, e^{-\epsilon\, \frac{1}{q} d_{\mathbb{R}}(y,z)} \quad \lambda_\epsilon(z) = \begin{cases} \frac{e^\epsilon}{e^\epsilon+1} & z \in \{0, qk\} \\ \frac{e^\epsilon-1}{e^\epsilon+1} & 0 < z < k \end{cases}$$

Due to the isometry, D_ϵ satisfies $\epsilon d_{\mathbb{R}}$-privacy iff D'_ϵ satisfies $\epsilon \frac{1}{q} d_{\mathbb{R}}$-privacy, thus (from the optimality of D_ϵ) it follows that D'_ϵ is $\epsilon \frac{1}{q} d_{\mathbb{R}}$-optimal for all $\epsilon > 0$.

Finally we define:

$$D''_\epsilon = D'_{q\epsilon}$$

From the optimality of D'_ϵ we get that D''_ϵ is $(q\epsilon)\frac{1}{q} d_{\mathbb{R}}$-optimal, i.e. it is $\epsilon d_{\mathbb{R}}$-optimal.

This concludes the proof, since D''_ϵ is exactly the pdf of $TG_\epsilon(\mathcal{Y}_q)$. □

The results above bring us directly to our sufficient condition.

Theorem 10. *Let $\mathcal{Y} = q[0..k]$ and assume that $f : \mathcal{X} \to \mathcal{Y}$ is uniformly Δ-sensitive wrt $d_{\mathcal{X}}, d_{\mathbb{R}}$. Then the TG_ϵ mechanism with input $\mathcal{Y}$ is f-$\Delta d_{\mathcal{X}}$-optimal.*

Proof. Direct corollary of Prop 8 and Prop 6. □

C Proofs of Section 5

Proposition 2. *$\Phi_1, \mathcal{N}_1$ are maximally tight wrt both $d_1, \widetilde{d}_1$.*

Proof. We first consider d_1. Let $x, x' \in \mathcal{V}^n$, we show that there exist a tight chain from x to x' that is both a maximal Φ_1-chain and a maximal $\mathcal{N}_1$-chain. We recursively create a chain $x_1, \ldots, x_{n+1}$ from x to x' by modifying one element at a time:

$$x_1 = x$$
$$x_{i+1} = x_i[^{x'[i]}/_i] \quad i \in 1..n$$

It is easy to see that $d_1(x, x') = \sum_{i=1}^{n} d_1(x_i, x_{i+1})$ so the chain is tight wrt d_1.

Fix any $i \in 1..n$ and let $w : \mathcal{V} \to \mathcal{V}$ defined as:

$$w(v) = \begin{cases} x'[i] & \text{if } v = x[i] \\ x[i] & \text{if } v = x'[i] \\ v & \text{otherwise} \end{cases}$$

For the hiding function $\phi_{i,w} \in \Phi_1$ it holds that

$$\phi_{i,w}(x_i) = x_{i+1} \qquad \phi_{i,w}(x_{i+1}) = x_i \qquad d_1(x_i, x_{i+1}) = d_1(\phi_{i,w})$$

hence the chain is a maximal Φ_1-chain.

Moreover, let $V = \{x[i], x'[i]\}$. For the neighborhood $N_{i,V}(x_i) \in \mathcal{N}_1$ it holds that

$$\{x_i, x_{i+1}\} \subseteq N_{i,V}(x_i) \qquad d_1(x_i, x_{i+1}) = d_1(N_{i,V}(x_i))$$

so the chain is a maximal $\mathcal{N}_1$-chain.

The case of $\widetilde{d}_1$ is similar, since it is a scaled version of d_1. □

Proposition 3. *$\widetilde{d}_1 \leq d_h$, thus $\epsilon\widetilde{d}_1$-privacy implies ϵd_h-privacy.*

Proof. Fix $x, x' \in \mathcal{V}^n$ and let $I = \{i \in 1..n \mid x[i] \neq x[i']\}$. Then

$$\widetilde{d}_1(x, x') = \frac{\sum_{i \in I} d_{\mathcal{V}}(x[i], x[i'])}{d_{\mathcal{V}}(\mathcal{V})} \leq \frac{\sum_{i \in I} d_{\mathcal{V}}(\mathcal{V})}{d_{\mathcal{V}}(\mathcal{V})} = |I| = d_h(x, x')$$

□

We continue by introducing a relaxed version of the concept of maximal $\mathcal{N}$-tightness (Def 3), called $\mathcal{N}$-tightness, by simply dropping the requirement $d_{\mathcal{X}}(x_i, x_{i+1}) = d_{\mathcal{X}}(N)$ from Def 3.

Definition 10. *Let $\mathcal{N} \subseteq 2^{\mathcal{X}}$. A chain $\tilde{x}$ is called an* $\mathcal{N}$-chain *iff for every step i there exists $N \in \mathcal{N}$ such that $\{x_i, x_{i+1}\} \subseteq N$. Then $\mathcal{N}$ is called* tight *wrt $d_{\mathcal{X}}$ iff $\forall x, x' \in \mathcal{X}$ there exists a tight $\mathcal{N}$-chain from x to x'.*

We can now show the (maximal wrt d_h, simple wrt $d_1, \widetilde{d}_1$) tightness of $\mathcal{N}_h$, which will be useful later on.

Proposition 9. *$\mathcal{N}_h$ is maximally tight wrt d_h and tight wrt both $d_1, \widetilde{d}_1$.*

Proof. Let $x, x' \in \mathcal{V}^n$. We need to show that there exists a tight (also maximal in the case of d_h) $\mathcal{N}_h$-chain from x to x'. We recursively create a chain $x_1, \ldots, x_{n+1}$ from x to x' by modifying one element at a time:

$$x_1 = x$$
$$x_{i+1} = x_i[{}^{x'[i]}/_i] \quad i \in 1..n$$

It is easy to see that $d(x, x') = \sum_{i=1}^{n} d(x_i, x_{i+1})$, for all $d \in \{d_h, d_1, \widetilde{d}_1\}$, so the chain is tight wrt all three metrics. Moreover, we have that $\{x_i, x_{i+1}\} \subseteq N_i(x_i)$ so the chain is an $\mathcal{N}_h$-chain wrt all metrics.

For d_h, it also holds that $d_h(x_i, x_{i+1}) = d_h(N_i(x_i)) = 1$, so the chain is a maximal $\mathcal{N}_h$-chain wrt d_h. □

We continue with a lemma that facilitates proofs of Δ-sensitivity by reducing the pairs of secrets x, x' that one needs to check.

Lemma 3. *Let $\mathcal{N} \subseteq 2^{\mathcal{X}}$ be tight (Def 10) and assume:*

$$d_{\mathcal{Y}}(f(x), f(x')) \leq \Delta d_{\mathcal{X}}(x, x') \qquad \forall N \in \mathcal{N}, x, x' \in N$$

Then f is Δ-sensitive wrt $d_{\mathcal{X}}, d_{\mathcal{Y}}$.

Proof. Fix $x, x' \in \mathcal{X}$, we need to show that $d_{\mathcal{Y}}(f(x), f(x')) \leq \Delta d_{\mathcal{X}}(x, x')$. Since $\mathcal{N}$ is tight there exist a tight $\mathcal{N}$-chain $x = x_1, \ldots, x_n = x'$, such that each step x_i, x_{i+1}

belongs to some set $N \in \mathcal{N}$. We have:

$$\begin{aligned}
& d_{\mathcal{Y}}(f(x), f(x')) \\
& \leq \sum_{i=1}^{n-1} d_{\mathcal{Y}}(f(x_i), f(x_{i+1})) && \text{triangle ineq.} \\
& \leq \Delta \sum_{i=1}^{n-1} d_{\mathcal{X}}(x_i, x_{i+1}) && \text{hypoth., } x_i, x_{i+1} \in N \\
& = \Delta d_{\mathcal{X}}(x, x') && \text{tightness of chain}
\end{aligned}$$

□

Proposition 4. *Let $f \in \mathcal{C}$. The sensitivity of f wrt both $d_h, d_{\mathbb{R}}$ and $\widetilde{d}_1, d_{\mathbb{R}}$ is $d_{\mathcal{V}}(\mathcal{V})$.*

Proof. Let $f \in \mathcal{C}$. We first show that f is $d_{\mathcal{V}}(\mathcal{V})$-sensitive wrt both $d_h, d_{\mathbb{R}}$ and $\widetilde{d}_1, d_{\mathbb{R}}$. From Prop 9 together with Theorem 3, we only need to show the sensitivity for databases x, x' from some set $N_i(x) \in \mathcal{N}_h$, i.e. $x \sim_i x'$.

For d_h, we have $d_h(x, x') = 1$, thus

$$d_{\mathcal{Y}}(f(x), f(x')) \leq d_{\mathcal{V}}(x[i], x'[i]) \leq d_{\mathcal{V}}(\mathcal{V}) d_h(x, x')$$

For $\widetilde{d}_1$ we have:

$$d_{\mathcal{Y}}(f(x), f(x')) \leq d_{\mathcal{V}}(x[i], x'[i]) = d_{\mathcal{V}}(\mathcal{V}) \widetilde{d}_1(x, x')$$

Then, for any $\Delta < d_{\mathcal{V}}(\mathcal{V})$, f is not Δ-sensitive for neither metric, since from Def 8 there exists $x \sim_i x'$ such that

$$d_{\mathcal{Y}}(f(x), f(x')) = d_{\mathcal{V}}(\mathcal{V}) > \Delta d_h(x, x')$$

and similarly for $\widetilde{d}_1$. □

Proposition 5. *The* $\mathrm{sum}, \mathrm{avg}, p\text{-perc}$ *queries belong to $\mathcal{C}$.*

Proof. The universe is assumed to be $\mathcal{V} = q[0..k]_{\varnothing}$ for some $k \in \mathbb{N}, q > 0$. Let $x \sim_i x' \in \mathcal{V}^n$. We first show that $d_{\mathbb{R}}(f(x), f(x')) \leq d_{\mathcal{V}}(x[i], x'[i])$.

For sum, it is easy to see that

$$|\mathrm{sum}(x), \mathrm{sum}(x')| = \begin{cases} d_{\mathcal{V}}(x[i], x'[i]) & x[i] \neq \varnothing, x'[i] \neq \varnothing \\ x'[i] & x[i] = \varnothing \\ x[i] & x'[i] = \varnothing \end{cases}$$

Note that $x'[i] \leq d_{\mathcal{V}}(x[i], x'[i]) = qr$ in the case $x[i] = \varnothing$ (and similarly for $x'[i] = \varnothing$).[10]

[10] It is crucial here that $\mathcal{V}$ contains 0, so that $v \leq d_{\mathcal{V}}(\mathcal{V})$ for all non-null v. If $0 \notin \mathcal{V}$, we can achieve a similar result for sum by adapting the way $d_{\mathcal{V}}$ treats $\varnothing$.

Consider now $f \in \{\mathrm{avg}, p\text{-perc}\}$. From Theorem 13 we know that both queries are 1-sensitive wrt $d_\infty, d_\mathbb{R}$. And since $d_\infty(x, x') = d_\mathcal{V}(x[i], x'[i])$ we have:

$$d_\mathbb{R}(f(x), f(x')) \leq d_\infty(x, x') = d_\mathcal{V}(x[i], x'[i])$$

Finally, we need to show that there exist $x \sim_i x' \in \mathcal{V}^n$ such that $d_\mathbb{R}(f(x), f(x')) = d_\mathcal{V}(\mathcal{V}) = qk$. We construct $x = \langle 0, \varnothing, \ldots, \varnothing\rangle$, $x' = \langle qk, \varnothing, \ldots, \varnothing\rangle$. These databases satisfy $d_\mathbb{R}(f(x), f(x')) = qk$ for all queries. □

Theorem 11. *The* $\mathrm{sum}, \mathrm{avg}$ *and* p-perc *queries are all uniformly* qk*-sensitive wrt* $\widetilde{d}_1, d_\mathbb{R}$.

Proof. First we have to show that the queries are qk-sensitive wrt $\widetilde{d}_1, d_\mathbb{R}$. This comes from Prop 4 and 5, since all queries belong to the family $\mathcal{C}$.

We now show the uniform sensitivity of sum. Let $y, y' \in q[0..nk]$ and assume that $y \geq y'$. It is easy to see that we can construct databases x, x' such that $\mathrm{sum}(x) = y, \mathrm{sum}(x') = y'$ and $x[i] \geq x'[i]$ for all $i \in 1..n$. For x, x' we have

$$\begin{aligned} d_1(x, x') &= \textstyle\sum_i |x[i] - x'[i]| \\ &= |\textstyle\sum_i x[i]| - |\sum_i x'[i]| && x[i] \geq x'[i] \\ &= d_\mathbb{R}(\mathrm{sum}(x), \mathrm{sum}(x')) \end{aligned}$$

Thus $d_\mathbb{R}(y, y') = qk\ \widetilde{d}_1(x, x')$, which means that the chain y, y' is qk-expansive wrt $\widetilde{d}_1$.

Finally, for $f \in \{\mathrm{avg}, p\text{-perc}\}$ let $y, y' \in q[0..k]$. We construct two databases x, x' with a single present individual as follows:

$$x = \langle y, \varnothing, \ldots, \varnothing\rangle \qquad x' = \langle y', \varnothing, \ldots, \varnothing\rangle$$

It is easy to see that $f(x) = y, f(x') = y'$ and $d_1(x, x') = d_\mathbb{R}(y, y')$. Thus $d_\mathbb{R}(y, y') = qk\ \widetilde{d}_1(x, x')$ which means that the chain y, y' is qk-expansive wrt $\widetilde{d}_1$. □

Theorem 12. *The* $\mathrm{sum}, \mathrm{avg}$ *and* p-perc *queries are all uniformly* 1*-sensitive wrt* $d_1, d_\mathbb{R}$.

Proof. Direct consequence of Theorem 11, since $d_1 = qk\ \widetilde{d}_1$. □

D Proofs of Section 6

Proposition 10. $\Phi_\infty, \mathcal{N}_\infty$ *are maximally tight wrt* d_∞.

Proof. Let $x, x' \in \mathcal{V}^n$, and consider the trivial 1-step chain x, x'. This chain is trivially tight, we need to show that it is both a maximal Φ_∞-chain and a maximal $\mathcal{N}_\infty$-chain.

For each $i \in 1..n$ we define a function $w_i : \mathcal{V} \to \mathcal{V}$ as:

$$w_i(v) = \begin{cases} x'[i] & \text{if } v = x[i] \\ x[i] & \text{if } v = x'[i] \\ v & \text{otherwise} \end{cases}$$

For the hiding function $\phi = \phi_{1,w_1} \circ \ldots \circ \phi_{n,w_n}$ we have that $\phi \in \Phi_\infty$ and moreover:

$$\phi(x) = x' \qquad \phi(x') = x \qquad d_\infty(x, x') = d_\infty(\phi)$$

hence the chain is a maximal Φ_∞-chain.

Moreover, let $V_i = \{x[i], x'[i]\}, i \in 1..n$. For the neighborhood $N_{\{V_i\}} \in \mathcal{N}_\infty$ it holds that

$$\{x, x'\} \subseteq N_{\{V_i\}} \qquad d_\infty(x, x') = d_\infty(N_{\{V_i\}})$$

so the chain is a maximal $\mathcal{N}_\infty$-chain. □

Theorem 13. *The* avg *and* p-perc *queries are both uniformly* 1*-sensitive wrt* $d_\infty, d_\mathbb{R}$.

Proof. The universe is assumed to be $\mathcal{V} = q[0..k]_\varnothing$ for some $k \in \mathbb{N}, q > 0$. The p-percentile query ($0 \leq p < 100$) is defined as $p\text{-perc}(x) = \text{sort}(x)[l]$ for $l = \lfloor \frac{p}{100} m + 1 \rfloor$, where m is the number of non-null values in x and sort returns a sorted version of x (after removing the null values). We also define $I_\varnothing(x) = \{i \in 1..n \mid x[i] = \varnothing\}$.

We fist show that both queries are 1-sensitive wrt $d_\infty, d_\mathbb{R}$. Let $x, x' \in \mathcal{V}^n$. If $I_\varnothing(x) \neq I_\varnothing(x')$ then x, x' are maximally distant, i.e. $d_\infty(x, x') = d_\infty(\mathcal{V}^n) = qk$. Then for both queries it trivially holds that $d_\mathbb{R}(f(x), f(x')) \leq qk = d_\infty(x, x')$ since their range is $q[0..k]$.

It remains to show 1-sensitivity for the case $I_\varnothing(x) = I_\varnothing(x') = I$. For the average query we have

$$\begin{aligned}
&d_\mathbb{R}(\text{avg}(x), \text{avg}(x')) \\
&= \frac{1}{|I|} |(\textstyle\sum_{i \in I} x[i]) - (\sum_{i \in I} x'[i])| \\
&= \frac{1}{|I|} |\textstyle\sum_{i \in I} (x[i] - x'[i])| \\
&\leq \frac{1}{|I|} \textstyle\sum_{i \in I} |x[i] - x'[i]| && \text{subadditivity of } |\cdot| \\
&\leq \frac{1}{|I|} \textstyle\sum_{i \in I} d_\infty(x, x') && |x[i] - x'[i]| \leq d_\infty(x, x') \\
&= d_\infty(x, x')
\end{aligned}$$

For the p-perc query it holds that $p\text{-perc}(x) = \text{sort}(x)[l]$ and $p\text{-perc}(x') = \text{sort}(x')[l]$ for the same l (since $I_\varnothing(x) = I_\varnothing(x')$). Let $h, h' \in 1..n$ such that $x[h] = \text{sort}(x)[l]$ and $x'[h'] = \text{sort}(x')[l]$.

Assume that $x[h] \leq x[h']$ (the case $x[h] \geq x[h']$ is symmetric). By the definition of sort, there are at least l elements $j \in 1..n$ such that $x[j] \leq x[h]$ (including h itself). Moreover, there are at most $l - 1$ elements $j \in 1..n$ such that $x'[j] < x'[h']$. Hence, there exists at least one $j \in 1..n$ such that

$$x[j] \leq x[h] \qquad \text{and} \qquad x'[j] \geq x'[h']$$

It also holds that $|x[i] - x'[i]| \leq d_\infty(x, x')$, i.e.

$$x[i] - d_\infty(x, x') \leq x'[i] \leq x[i] + d_\infty(x, x') \quad \forall i \in 1..n \tag{4}$$

From $x[h] \leq x[h']$ and (4) we get

$$x[h] - d_\infty(x, x') \leq x'[h']$$

Moreover, it holds that

$$x'[h'] \leq x'[j] \leq x[j] + d_\infty(x, x') \leq x[h] + d_\infty(x, x')$$

thus

$$d_{\mathbb{R}}(p\text{-perc}(x), p\text{-perc}(x')) = |x[h] - x'[h']| \leq d_\infty(x, x')$$

For the "uniformly" part, let $y, y' \in q[0..k]$; we construct $x = \langle y, \varnothing, \ldots, \varnothing \rangle, x' = \langle y', \varnothing, \ldots, \varnothing \rangle$, for which it holds that $f(x) = y, f(x') = y'$ (for both queries) and $d_\infty(x, x') = d_{\mathbb{R}}(y, y')$.

Note that for p-perc we can construct x, x' without $\varnothing$ values with the same property. However, for the avg query this is not possible; its uniform optimality depends on the fact that $\varnothing$ values are allowed. If $\varnothing \notin \mathcal{V}$ then avg is essentially equivalent to sum, which is not uniformly optimal wrt $d_\infty, d_{\mathbb{R}}$. □

Geo-Indistinguishability: Differential Privacy for Location-Based Systems*

M. Andrés
LIX, École Polytechnique
gueles@gmail.com

N. Bordenabe
INRIA
LIX, École Polytechnique
nbordenabe@lix.polytechnique.fr

K. Chatzikokolakis
CNRS
LIX, École Polytechnique
kostas@chatzi.org

C. Palamidessi
INRIA
LIX, École Polytechnique
catuscia@lix.polytechnique.fr

ABSTRACT

The growing popularity of location-based systems, allowing unknown/untrusted servers to easily collect huge amounts of information regarding users' location, has recently started raising serious privacy concerns. In this paper we study geo-indistinguishability, a formal notion of privacy for location-based systems that protects the user's exact location, while allowing approximate information – typically needed to obtain a certain desired service – to be released.

Our privacy definition formalizes the intuitive notion of protecting the user's location within a radius r with a level of privacy that depends on r, and corresponds to a generalized version of the well-known concept of *differential privacy*. Furthermore, we present a perturbation technique for achieving geo-indistinguishability by adding controlled random noise to the user's location. We demonstrate the applicability of our technique on a LBS application. Finally, we compare our mechanism with other ones in the literature. It turns our that our mechanism offers the best privacy guarantees, for the same utility, among all those which do not depend on the prior.

1. INTRODUCTION

In recent years, the increasing availability of location information about individuals has led to a growing use of systems that record and process location data, generally referred to as "location-based systems". Such systems include (a) Location Based Services (LBSs), in which a user obtains, typically in real-time, a service related to his current location, and (b) location-data mining algorithms, used to determine points of interest and traffic patterns.

The use LBSs, in particular, has been has significantly increased by the growing popularity of mobile devices equipped with GPS chips, in combination with the increasing availability of wireless data connections. A resent study in the US shows that 46% of the adult population of the country owns a smartphone and, furthermore, that 74% of those owners use LBSs [1]. Examples of LBSs include mapping applications (eg, Google Maps), Points of Interest (POI) retrieval (eg, AroundMe), coupon/discount providers (eg, GroupOn), GPS navigation (eg, TomTom), and location-aware social networks (eg, Foursquare).

While location-based systems have demonstrated to provide enormous benefits to individuals and society, the growing exposure of users' location information raises important privacy issues. First of all, location information itself may be considered as sensitive. Furthermore, it can be easily linked to a variety of other information that an individual usually wishes to protect: by collecting and processing accurate location data on a regular basis, it is possible to infer an individual's home or work location, sexual preferences, political views, religious inclinations, etc. In its extreme form, monitoring and control of an individual's location has been even described as a form of slavery [8].

Several notions of privacy for location-based systems have been proposed in the literature. In Section 2 we give an overview of such notions, and we discuss their shortcomings in relation to our motivating LBS application. Aiming at addressing these shortcomings, we propose a *formal privacy definition* for LSB's, as well as a randomized technique that allows a user to disclose *enough location information* to obtain the desired service, while satisfying the aforementioned privacy notion. Our proposal is based on a generalization of *differential privacy* [10] developed in [5]. Like differential privacy, our notion and technique abstract from the side information of the adversary, such as any prior probabilistic knowledge about the user's actual location.

As a running example, we consider a user located in Paris who wishes to query an LBS provider for nearby restaurants in a private way, i.e., by disclosing some approximate information z instead of his exact location x. A crucial question is: what kind of privacy guarantee can the user expect in this scenario? To formalize this notion, we consider the privacy *within a radius*. We say that the user enjoys ℓ-*privacy within* r if, any two locations at distance at most r produce observations with "similar" distributions, where the "level of similarity" depends on ℓ. The idea is that ℓ represents the user's *level* of privacy for that radius: the smaller ℓ is, the higher is the privacy.

In order to allow the LSB to provide a useful service, we require that the (inverse of the) level of privacy ℓ depend on the radius r. In particular, we require that it is proportional to r, which brings us to our definition of *geo-indistinguishability*:

> A mechanism satisfies ϵ-geo-indistinguishability iff for any radius $r > 0$, the user enjoys ϵr-privacy within r.

This definition implies that the user is protected within any radius r, but with a level $\ell = \epsilon r$ that increases with the distance. Within a short radius, for instance $r = 1$ km, ℓ is small, guaranteeing that the provider cannot infer the user's location within, say, the 7th arrondissement of Paris. Farther away from the user, for instance for

*This work is partially funded by the Inria large scale initiative CAPPRIS, the EU FP7 grant no. 295261 (MEALS), and the project ANR-12-IS02-001 PACE. Nicolás E. Bordenabe was partially funded by the French Defense procurement agency (DGA) with a PhD grant.

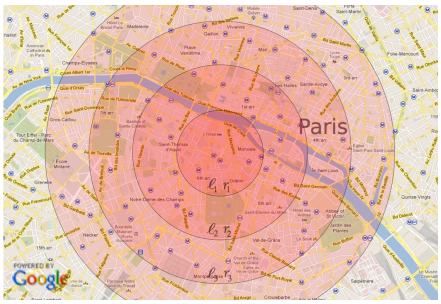


Figure 1: Geo-indistinguishability: privacy varying with r

$r = 10.000$ km, ℓ becomes large, allowing the LBS provider to infer that with high probability the user is located in Paris instead of, say, London. Figure 1 illustrates the idea of privacy levels decreasing with the radius.

We develop a mechanism to achieve geo-indistinguishability by perturbating the user's location x. The inspiration comes from one of the most popular approaches for differential privacy, namely the Laplacian noise. We adopt a specific planar version of the Laplace distribution, allowing to draw points in a *geo-indistinguishable* way. Moreover, via a transformation to polar coordinates, we are able to draw in an efficiently. However, as standard (digital) applications require a finite representation of locations, it is necessary to add a discretization step. Such operation jeopardizes the privacy guarantees, for reasons similar to the rounding effects of finite-precision operations [25]. We show how to preserve geo-indistinguishability, at the price of a degradation of the privacy level, and how to adjust the privacy parameters in order to obtain a desired level of privacy.

Finally, we compare our mechanism with other ones in the literature, using the privacy metric proposed in [32]. It turns our that our mechanism offers the best privacy guarantees, for the same utility, among all those which do not depend on the prior knowledge of the adversary. The advantages of the independence from the prior are obvious: first, the mechanism is designed once and for all (i.e. it does not need to be recomputed every time the adversary changes, it works also in simultaneous presence of different adversaries, etc.). Second, and even more important, it is applicable also when we do not know the prior.

Road Map. In Section 2 we discuss notions of location privacy from the literature and point out their weaknesses and strengths. In Section 3 we formalize the notion of geo-indistinguishability in three equivalent ways. We then proceed to describe a mechanism that provides geo-indistinguishability in Section 4. In Sections 5 we demonstrate the applicability of our approach by a case study related to LBSs. In Section 6 we compare the privacy guarantees of our methods with those of two other methods from the literature. Section 7 discusses related work and Section 8 concludes.

2. EXISTING NOTIONS OF PRIVACY

In this section, we examine various notions of location privacy from the literature, as well as techniques to achieve them. We consider the motivating example from the introduction, of a user in Paris wishing to find nearby restaurants with good reviews. To achieve this goal, he uses a handheld device (eg. a smartphone) to query a public LBS provider. However, the user expects his location to be kept private: informally speaking, the information sent to the provider should not allow him to accurately infer the user's location. Our goal is to provide a *formal* notion of privacy that adequately captures the user's expected privacy. From the point of view of the employed mechanism, we require a technique that can be performed in real-time by a handheld device, without the need of any trusted anonymization party.

Expected Distance Error.

Expectation of distance error [31, 32, 19] is a natural way to quantify the privacy offered by a location-obfuscation mechanism. Intuitively, it reflects the degree of accuracy by which an adversary can guess the real location of the user by observing the obfuscated location, and using the side-information available to him.

There are several works relying on this notion. In [19], a perturbation mechanism is used to confuse the attacker by crossing paths of individual users, rendering the task of tracking individual paths challenging. In [32], an optimal location-obfuscation mechanism (i.e., achieving maximum level of privacy for the user) is obtained by solving a linear program in which the contraints are determined by the quality of service and by the user's profile.

It is worth noting that this privacy notion and the obfuscation mechanisms based on it are explicitly defined in terms of the adversary's side information. In contrast, our notion of geo-indistinguishability abstracts from the attacker's prior knowledge, and is therefore suitable for scenarios where the prior is unknown, or the same mechanism must be used for multiple users. A detailed comparison with the mechanism of [32] is provided in Section 6.

k-anonymity.

The notion of k-anonymity is the most widely used definition of privacy for location-based systems in the literature. Many systems in this category [17, 15, 26] aim at protecting the user's *identity*, requiring that the attacker cannot infer which user is executing the query, among a set of k different users. Such systems are outside the scope of our problem, since we are interested in protecting the user's *location*.

On the other hand, k-anonymity has also been used to protect the user's location (sometimes called l-diversity in this context), requiring that it is indistinguishable among a set of k points (often required to share some semantic property). One way to achieve this is through the use of *dummy locations* [21, 29]. This technique involves generating $k-1$ properly selected dummy points, and performing k queries to the service provider, using the real and dummy locations. Another method for achieving k-anonymity is through *cloaking* [3, 9, 34]. This involves creating a cloaking region that includes k points sharing some property of interest, and then querying the service provider for this cloaking region.

Even when side knowledge does not explicitly appear in the definition of k-anonymity, a system cannot be proved to satisfy this notion unless assumptions are made about the attacker's side information. For example, dummy locations are only useful if they look equally likely to be the real location from the point of view of the attacker. Any side information that allows to rule out any of those points, as having low probability of being the real location, would immediately violate the definition.

Counter-measures are often employed to avoid this issue: for instance, [21] takes into account concepts such as ubiquity, congestion and uniformity for generating dummy points, in an effort to make them look realistic. Similarly, [34] takes into account the user's side information to construct a cloaking region. Such counter-measures have their own drawbacks: first, they complicate the employed techniques, also requiring additional data to be taken into account (for instance, precise information about the environment or the location of nearby users), making their application in real-time by a handheld device challenging. Moreover, the attacker's actual side information might simply be inconsistent with

the assumptions being made.

As a result, notions that abstract from the attacker's side information, such as differential privacy, have been growing in popularity in recent years, compared to k-anonymity-based approaches.

Differential Privacy.

Differential Privacy [10] is a notion of privacy from the area of statistical databases. Its goal is to protect an individual's data while publishing aggregate information about the database. Differential privacy requires that modifying a single user's data should have a negligible effect on the query outcome. More precisely, it requires that the probability that a query returns a value v when applied to a database D, compared to the probability to report the same value when applied to an *adjacent* database D' – meaning that D, D' differ in the value of a single individual – should be within a bound of e^{ϵ}. A typical way to achieve this notion is to add controlled random noise to the query output, for example drawn from a Laplace distribution. An advantage of this notion is that a mechanism can be shown to be differentially private independently from any side information that the attacker might possess.

Differential privacy has also been used in the context of location privacy. In [24], it is shown that a synthetic data generation technique can be used to publish statistical information about commuting patterns in a differentially private way. In [18], a quadtree spatial decomposition technique is used to ensure differential privacy in a database with location pattern mining capabilities.

As shown in the aforementioned works, differential privacy can be successfully applied in cases where aggregate information about several users is published. On the other hand, the nature of this notion makes it poorly suitable for applications in which a single individual is involved, such as our motivating scenario. The secret in this case is the location of a single user. Thus, differential privacy would require that any change in that location should have negligible effect on the published output, making it impossible to communicate any useful information to the service provider.

Other location-privacy metrics.

[7] proposes a location cloaking mechanism, and focuses on the evaluation of Location-based Range Queries. The degree of privacy is measured by the size of the cloak (also called *uncertainty region*), and by the coverage of sensitive regions, which is the ratio between the area of the cloak and the area of the regions inside the cloak that the user considers to be sensitive. In order to deal with the side-information that the attacker may have, ad-hoc solutions are proposed, like patching cloaks to enlarge the uncertainty region or delaying requests. Both solutions may cause a degradation in the quality of service.

In [2], the real location of the user is assumed to have some level of inaccuracy, due to the specific sensing technology or to the environmental conditions. Different obfuscation techniques are then used to increase this inaccuracy in order to achieve a certain level of privacy. This level of privacy is defined as the ratio between the accuracy before and after the application of the obfuscation techniques.

Similar to the case of k-anonymity, both privacy metrics mentioned above make implicit assumptions about the adversary's side information. This may imply a violation of the privacy definition in a scenario where the adversary has some knowledge about the user's real location.

Transformation-based approaches.

A number of approaches for location privacy are radically different from the ones mentioned so far. Instead of cloaking the user's location, they aim at making it completely invisible to the service provider. This is achieved by transforming all data to a different space, usually employing cryptographic techniques, so that they can be mapped back to spatial information only by the user [20, 16]. The data stored in the provider, as well as the location send by the user are encrypted. Then, using techniques from Private Information Retrieval, the provider can return information about the encrypted location, without ever discovering which actual location it corresponds to.

A drawback of these techniques is that they are computationally demanding, making it difficult to implement them in a handheld device. Moreover, they require the provider's data to be encrypted, making it impossible to use providers, such as Google Maps, which have access to the real data.

3. GEO-INDISTINGUISHABILITY

In this section we formalize our notion of geo-indistinguishability. As already discussed in the introduction, the main idea behind this notion is that, for any radius $r > 0$, the user enjoys ϵr-privacy within r, i.e. the level of privacy is proportional to the radius. Note that the parameter ϵ corresponds to the level of privacy at one unit of distance. For the user, a simple way to specify his privacy requirements is by a tuple (ℓ, r), where r is the radius he is mostly concerned with and ℓ is the privacy level he wishes *for that radius*. In this case, it is sufficient to require ϵ-geo-indistinguishability for $\epsilon = \ell/r$; this will ensure a level of privacy ℓ within r, and a proportionally selected level for all other radii.

So far we kept the discussion on an informal level by avoiding to explicitly define what ℓ-privacy within r means. In the remaining of this section we give a formal definition, as well as two characterizations which clarify the privacy guarantees provided by geo-indistinguishability.

Probabilistic model.

We first introduce a simple model used in the rest of the paper. We start with a set $\mathcal{X}$ of *points of interest*, typically the user's possible locations. Moreover, let $\mathcal{Z}$ be a set of possible *reported values*, which in general can be arbitrary, allowing to report obfuscated locations, cloaking regions, sets of locations, etc. However, to simplify the discussion, we sometimes consider $\mathcal{Z}$ to also contain spatial points, assuming an operational scenario of a user located at $x \in \mathcal{X}$ and communicating to the attacker a randomly selected location $z \in \mathcal{Z}$ (e.g. an obfuscated point).

Probabilities come into place in two ways. First, the attacker might have side information about the user's location, knowing, for example, that he is likely to be visiting the Eiffel Tower, while unlikely to be swimming in the Seine river. The attacker's side information can be modeled by a *prior* distribution π on $\mathcal{X}$, where $\pi(x)$ is the probability assigned to the location x.

Second, the selection of a reported value in $\mathcal{Z}$ is itself probabilistic; for instance, z can be obtained by adding random noise to the actual location x (a technique used in Section 4). A *mechanism* K is a probabilistic function for selecting a reported value; i.e. K is a function assigning to each location $x \in \mathcal{X}$ a probability distribution on $\mathcal{Z}$, where $K(x)(Z)$ is the probability that the reported point belongs to the set $Z \subseteq \mathcal{Z}$, when the user's location is x.[1] Starting from π and using Bayes' rule, each observation $Z \subseteq \mathcal{Z}$ of a mechanism K induces a *posterior* distribution $\sigma = \mathbf{Bayes}(\pi, K, Z)$ on $\mathcal{X}$, defined as $\sigma(x) = \frac{K(x)(Z)\pi(x)}{\sum_{x'} K(x')(Z)\pi(x')}$.

[1]For simplicity we assume distributions on $\mathcal{X}$ to be discrete, but allow those on $\mathcal{Z}$ to be continuous (c.f. Section 4). All sets to which probability is assigned are implicitly assumed to be measurable.

We define the *multiplicative distance* between two distributions σ_1, σ_2 on some set $\mathcal{S}$ as $d_{\mathcal{P}}(\sigma_1, \sigma_2) = \sup_{S \subseteq \mathcal{S}} |\ln \frac{\sigma_1(S)}{\sigma_2(S)}|$, with the convention that $|\ln \frac{\sigma_1(S)}{\sigma_2(S)}| = 0$ if both $\sigma_1(S), \sigma_2(S)$ are zero and ∞ if only one of them is zero.

3.1 Definition

We are now ready to state our definition of geo-indistinguishability. Intuitively, a privacy requirement is a constraint on the distributions $K(x), K(x')$ produced by two different points x, x'. Let $d(\cdot,\cdot)$ denote the Euclidean metric. Enjoying ℓ-privacy within r means that for any x, x' s.t. $d(x, x') \leq r$, the distance $d_{\mathcal{P}}(K(x), K(x'))$ between the corresponding distributions should be at most l. Then, requiring ϵr-privacy for all radii r, forces the two distributions to be similar for locations close to each other, while relaxing the constraint for those far away from each other, allowing a service provider to distinguish points in Paris from those in London.

DEFINITION 3.1 (GEO-INDISTINGUISHABILITY). *A mechanism K satisfies ϵ-geo-indistinguishability iff for all x, x':*

$$d_{\mathcal{P}}(K(x), K(x')) \leq \epsilon d(x, x')$$

Equivalently, the definition can be formulated as $K(x)(Z) \leq e^{\epsilon d(x,x')} K(x')(Z)$ for all $x, x' \in \mathcal{X}, Z \subseteq \mathcal{Z}$. Note that for all points x' within a radius r from x, the definition forces the corresponding distributions to be at most ϵr distant.

The above definition is very similar to the one of differential privacy, which requires $d_{\mathcal{P}}(K(x), K(x')) \leq \epsilon d_h(x, x')$, where d_h is the Hamming distance between databases x, x', i.e. the number of individuals in which they differ. In fact, geo-indistinguishability is an instance of a generalized variant of differential privacy, using an arbitrary metric between secrets. This generalized formulation has been known for some time: for instance, [27] uses it to perform a compositional analysis of standard differential privacy for functional programs, while [12] uses metrics between individuals to define "fairness" in classification. On the other hand, the usefulness of using different metrics to achieve different privacy goals and the semantics of the privacy definition obtained by different metrics have only recently started to be studied [5]. This paper focuses on location-based systems and is, to our knowledge, the first work considering privacy under the Euclidean metric, which is a natural choice for spatial data.

Note that in our scenario, using the Hamming metric of standard differential privacy – which aims at completely protecting the value of an individual – would be too strong, since the only information is the location of a single individual. Nevertheless, we are not interested in completely hiding the user's location, since some approximate information needs to be revealed in order to obtain the required service. Hence, using a privacy level that depends on the Euclidean distance between locations is a natural choice.

Finally, note that, since ϵ corresponds to the privacy level for one unit of distance, it is affected by the unit in which distances are measured. For instance, assume that $\epsilon = 0.1$ and distances are measured in meters. The level of privacy for points one kilometer away is 1000ϵ, hence changing the unit to kilometers requires to set $\epsilon = 100$ in order for the definition to remain unaffected. In other words, if r is a physical quantity expressed in some unit of measurement, then ϵ has to be expressed in the inverse unit. In this paper we omit the unit since the choice is orthogonal to our goals.

3.2 Characterizations

In this section we state two characterizations of geo-indistinguishability, obtained from the corresponding results of [5] (for general metrics), which provide intuitive interpretations of the privacy guarantees offered by geo-indistinguishability.

Adversary's conclusions under hiding.

The first characterization uses the concept of a *hiding function* $\phi : \mathcal{X} \to \mathcal{X}$. The idea is that ϕ can be applied to the user's actual location before the mechanism K, so that the latter has only access to a hidden version $\phi(x)$, instead of the real location x. A mechanism K with hiding applied is simply the composition $K \circ \phi$. Intuitively, a location remains private if, regardless of his side knowledge (captured by his prior distribution), an adversary draws the same conclusions (captured by his posterior distribution), regardless of whether hiding has been applied or not. However, if ϕ replaces locations in Paris with those in London, then clearly the adversary's conclusions will be greatly affected. Hence, we require that the effect on the conclusions depends on the maximum distance $d(\phi) = \sup_{x \in \mathcal{X}} d(x, \phi(x))$ between the real and hidden location.

THEOREM 3.1. *A mechanism K satisfies ϵ-geo-indistinguishability iff for all $\phi : \mathcal{X} \to \mathcal{X}$, all priors π on $\mathcal{X}$, and all $Z \subseteq \mathcal{Z}$:*

$$d_{\mathcal{P}}(\sigma_1, \sigma_2) \leq 2\epsilon d(\phi) \quad \textit{where} \quad \sigma_1 = \mathbf{Bayes}(\pi, K, Z) \quad \sigma_2 = \mathbf{Bayes}(\pi, K \circ \phi, Z)$$

Note that this is a natural adaptation of a well-known interpretation of standard differential privacy, stating that the attacker's conclusions are similar, regardless of his side knowledge, and regardless of whether an individual's real value has been used in the query or not. This corresponds to a hiding function ϕ removing the value of an individual.

Note also that the above characterization compares two *posterior* distributions. Both σ_1, σ_2 can be substantially different than the initial knowledge π, which means that an adversary does learn some information about the user's location.

Knowledge of an informed attacker.

A different approach is to measure how much the adversary learns about the user's location, by comparing his prior and posterior distributions. However, since some information is allowed to be revealed by design, these distributions can be far apart. Still, we can consider an *informed* adversary who already knows that the user is located within a set $N \subseteq \mathcal{X}$. Let $d(N) = \sup_{x,x' \in N} d(x, x')$ be the maximum distance between points in x. Intuitively, the user's location remains private if, regardless of his prior knowledge within N, the knowledge obtained by such an informed adversary should be limited by a factor depending on $d(N)$. This means that if $d(N)$ is small, i.e. the adversary already knows the location with some accuracy, then the information that he obtains is also small, meaning that he cannot improve his accuracy. Denoting by $\pi_{|N}$ the distribution obtained from π by restricting to N (i.e. $\pi_{|N}(x) = \pi(x|N)$), we obtain the following characterization:

THEOREM 3.2. *A mechanism K satisfies ϵ-geo-indistinguishability iff for all $N \subseteq \mathcal{X}$, all priors π on $\mathcal{X}$, and all $Z \subseteq \mathcal{Z}$:*

$$d_{\mathcal{P}}(\pi_{|N}, \sigma_{|N}) \leq \epsilon d(N) \quad \textit{where} \quad \sigma = \mathbf{Bayes}(\pi, K, Z)$$

Note that this is a natural adaptation of a well-known interpretation of standard differential privacy, stating that in informed adversary who already knows all values except individual's i, gains no extra knowledge from the reported answer, regardless of side knowledge about i's value [13].

Abstracting from side information.

A major difference of geo-indistinguishability, compared to similar approaches from the literature, is that it abstracts from the side information available to the adversary, i.e. from the prior distribution. This is a subtle issue, and often a source of confusion, thus we would like to clarify what "abstracting from the prior" means. The goal of a privacy definition is to restrict the information *leakage* caused by the observation. Note that the lack of leakage does not mean that the user's location cannot be inferred (it could be inferred by the prior alone), but instead that the adversary's knowledge does not increase *due to the observation*.

However, in the context of LBSs, no privacy definition can ensure a small leakage under any prior, and at the same time allow reasonable utility. Consider, for instance, an attacker who knows that the user is located at some airport, but not which one. The attacker's prior knowledge is very limited, still any useful LBS query should reveal at least the user's city, from which the exact location (i.e. the city's airport) can be inferred. Clearly, due to the side information, the leakage caused by the observation is high.

So, since we cannot eliminate leakage under any prior, how can we give a reasonable privacy definition without restricting to a particular one? First, we give a formulation (Def 3.1) which does not involve the prior at all, allowing to verify it without knowing the prior. At the same time, we give two characterizations which explicitly quantify over all priors, shedding light on how the prior affects the privacy guarantees.

Finally, we should point out that differential privacy abstracts from the prior in exactly the same way. Contrary to what is sometimes believed, the user's value is *not protected* under any prior information. Recalling the well-known example from [10], if the adversary knows that Terry Gross is two inches shorter than the average Lithuanian woman, then he can accurately infer the height, even if the average is release in a differentially private way (in fact no useful mechanism can prevent this leakage). Differential privacy does ensure that her risk is the same whether she participates in the database or not, but this might me misleading: it does not imply the lack of leakage, only that it will happen anyway, whether she participates or not!

3.3 Protecting location sets

So far, we have assumed that the user has a single location that he wishes to communicate to a service provider in a private way (typically his current location). In practice, however, it is common for a user to have multiple points of interest, for instance a set of past locations or a set of locations he frequently visits. In this case, the user might wish to communicate to the provider some information that depends on all points; this could be either the whole set of points itself, or some aggregate information, for instance their centroid. As in the case of a single location, privacy is still a requirement; the provider is allowed to obtain only approximate information about the locations, their exact value should be kept private. In this section, we discuss how ϵ-geo-indistinguishability extends to the case where the secret is a tuple of points $\mathbf{x} = (x_1, \ldots, x_n)$.

Similarly to the case of a single point, the notion of distance is crucial for our definition. We define the distance between two tuples of points $\mathbf{x} = (x_1, \ldots, x_n), \mathbf{x}' = (x'_1, \ldots, x'_n)$ as:

$$d_\infty(\mathbf{x}, \mathbf{x}') = \max_i d(x_i, x'_i)$$

Intuitively, the choice of metric follows the idea of reasoning within a radius r: when $d_\infty(\mathbf{x}, \mathbf{x}') \leq r$, it means that all x_i, x'_i are within distance r from each other. All definitions and results of this section can be then directly applied to the case of multiple points, by using d_∞ as the underlying metric. Enjoying ℓ-privacy within a radius r means that two tuples at most r away from each other, should produce distributions at most ϵr apart.

Reporting the whole set.

A natural question then to ask is how we can obfuscate a tuple of points, by independently applying an existing mechanism K_0 to each individual point, and report the obfuscated tuple. Starting from a tuple $\mathbf{x} = (x_1, \ldots, x_n)$, we independently apply K_0 to each x_i obtaining a reported point z_i, and then report the tuple $\mathbf{z} = (z_1, \ldots, z_n)$. Thus, the probability that the combined mechanism K reports $\mathbf{z}$, starting from $\mathbf{x}$, is the product of the probabilities to obtain each point z_i, starting from the corresponding point x_i, i.e. $K(\mathbf{x})(\mathbf{z}) = \prod_i K_0(x_i)(z_i)$.

The next question is what level of privacy does K satisfy. For simplicity, consider a tuple of only two points (x_1, x_2), and assume that K_0 satisfies ϵ-geo-indistinguishability. At first look, one might expect the combined mechanism K to also satisfy ϵ-geo-indistinguishability, however this is not the case. The problem is that the two points might be *correlated*, thus an observation about x_1 will reveal information about x_2 and vice versa. Consider, for instance, the extreme case in which $x_1 = x_2$. Having two observations about the same point reduces the level of privacy, thus we cannot expect the combined mechanism to provide the same level of privacy.

Still, if K_0 satisfies ϵ-geo-indistinguishability, then K can be shown to satisfy $n\epsilon$-geo-indistinguishability, i.e. a level of privacy that scales linearly with n. Due to this scalability issue, the technique of independently applying a mechanism to each point is only useful when the number of points is small. Still, this is sufficient for some applications, such as the case study of Section 5. Note, however, that this technique is by no means the best we can hope for: similarly to standard differential privacy [4, 28], better results could be achieved by adding noise to the whole tuple $\mathbf{x}$, instead of each individual point. We believe that using such techniques we can achieve geo-indistinguishability for a large number of locations with reasonable noise, leading to practical mechanisms for highly mobile applications. We have already started exploring this direction of future work.

Reporting an aggregate location.

Another interesting case is when we need to report some aggregate information obtained by $\mathbf{x}$, for instance the centroid of the tuple. In general we might need to report the result of a query $f : \mathcal{X}^n \to \mathcal{X}$. Similarly to the case of standard differential privacy, we can compute the real answer $f(\mathbf{x})$ and the add noise by applying a mechanism K to it. If f is Δ-sensitive wrt d, d_∞, meaning that $d(f(\mathbf{x}), f(\mathbf{x}')) \leq \Delta d_\infty(\mathbf{x}, \mathbf{x}')$ for all $\mathbf{x}, \mathbf{x}'$, and K satisfies geo-indistinguishability, then the composed mechanism $K \circ f$ can be shown to satisfy $\Delta\epsilon$-geo-indistinguishability.

Note that when dealing with aggregate data, standard differential privacy becomes a viable option. However, one needs to also examine the loss of utility caused by the added noise. This highly depends on the application: differential privacy is suitable for publishing aggregate queries with *low sensitivity*, meaning that changes in a single individual have a relatively small effect on the outcome. On the other hand, location information often has high sensitivity. A trivial example is the case where we want to publish the complete tuple of points. But sensitivity can be high even for aggregate information: consider the case of publishing the centroid of 5 users located anywhere in the world. Modifying a single user can hugely affect their centroid, thus achieving differential privacy would require so much noise that the result would be useless. For geo-indistinguishability, on the other hand, one needs to consider the distance between points when computing the sensitivity. In the case of the centroid, a small (in terms of distance) change in the

tuple has a small effect on the result, thus geo-indistinguishability can be achieved with much less noise.

4. A MECHANISM

In this section we present a method to generate a noise satisfying geo-indistinguishability. We model the location domain as a discrete[2] Cartesian plane with the standard notion of Euclidean distance. This model can be considered a good approximation of the Earth surface when the area of interest is not too large.

(a) First, we define a geo-indistinguishable, continuous mechanism for the ideal case of the continuous plane.

(b) Then, we discretized the mechanism by remapping each point generated according to (a) to the closest point in the discrete domain.

(c) Finally, we truncate the mechanism, so to report only points within the limits of the area of interest.

4.1 A mechanism for the continuous plane

Following the above plan, we start by defining a geo-indistinguishable mechanism on the continuous plane. The idea is that whenever the actual location is $x_0 \in \mathbb{R}^2$, we report, instead, a point $x \in \mathbb{R}^2$ generated randomly according to the noise function. The latter needs to be such that the probabilities of reporting a point in a certain (infinitesimal) area around x, when the actual locations are x_0 and x'_0 respectively, differs at most by a multiplicative factor $e^{-\epsilon\, d(x_0,x'_0)}$.

We can achieve this property by requiring that the probability of generating a point in the area around x decreases exponentially with the distance from the actual location x_0. In a linear space this is exactly the behavior of the Laplace distribution, whose probability density function (pdf) is $\epsilon/2\, e^{-\epsilon\,|x-\mu|}$. This distribution has been used in the literature to add noise to query results on statistical databases, with μ set to be the actual answer, and it can be shown to satisfy ϵ-differential privacy [11].

There are two possible definitions of Laplace distribution on higher dimensions (multivariate Laplacians). The first one, investigated in [23], and used also in [13], is obtained from the standard Laplacian by replacing $|x-\mu|$ with $d(x,\mu)$. The second way consists in generating each Cartesian coordinate independently, according to a linear Laplacian. For reasons that will become clear in the next paragraph, we adopt the first approach.

The probability density function.

Given the parameter $\epsilon \in \mathbb{R}^+$, and the actual location $x_0 \in \mathbb{R}^2$, the pdf of our noise mechanism, on any other point $x \in \mathbb{R}^2$, is:

$$D_\epsilon(x_0)(x) = \frac{\epsilon^2}{2\,\pi}\, e^{-\epsilon\, d(x_0,x)} \qquad (1)$$

where $\epsilon^2/2\,\pi$ is a normalization factor. We call this function *planar Laplacian centered in* x_0. The corresponding distribution is illustrated in Figure 2. It is possible to show that (i) the projection of a planar Laplacian on any vertical plane passing by the center gives a (scaled) linear Laplacian, and (ii) the corresponding mechanism satisfies ϵ-geo-indistinguishability. These two properties would not be satisfied by the second approach to the multivariate Laplacian.

[2]For applications with digital interface the domain of interest is discrete, since the representation of the coordinates of the points is necessarily finite.

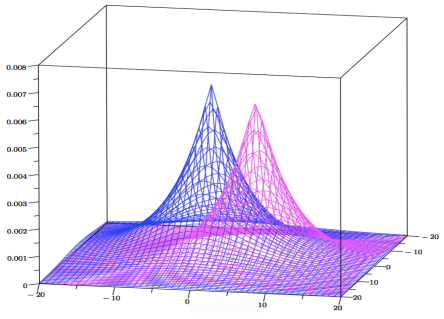


Figure 2: The pdf's of two planar Laplacians, centered in $(-2,-4)$ and in $(5,3)$ respectively, with $\epsilon = 1/5$.

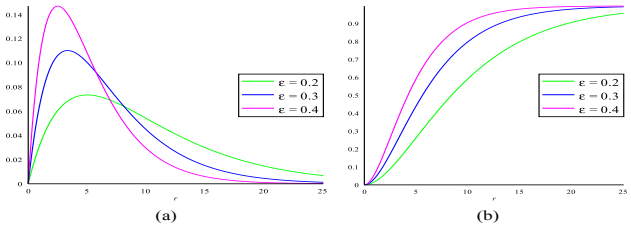


Figure 3: Gamma distribution: pdf and cdf for various ϵ.

Drawing a random point.

We illustrate now how to draw a random point from the pdf defined in (1). First of all, we note that the pdf of the planar Laplacian depends only on the distance from x_0. It will be convenient, therefore, to switch to a system of polar coordinates with origin in x_0. A point x will be represented as a point (r,θ), where r is the distance of x from x_0, and θ is the angle that the line $x\,x_0$ forms with respect to the horizontal axis of the Cartesian system. Following the standard transformation formula, the pdf of the *polar Laplacian* centered in the origin (x_0) is:

$$D_\epsilon(r,\theta) = \frac{\epsilon^2}{2\,\pi}\, r\, e^{-\epsilon\, r} \qquad (2)$$

We note now that the polar Laplacian defined above enjoys a property that is very convenient for drawing in an efficient way: *the two random variables that represent the radius and the angle are independent*. Namely, the pdf can be expressed as the product of the two marginals. In fact, let us denote these two random variables by R (the radius) and Θ (the angle). The two marginals are:

$$D_{\epsilon,R}(r) \;=\; \int_0^{2\pi} D_\epsilon(r,\theta)\, d\theta \;=\; \epsilon^2\, r\, e^{-\epsilon\, r}$$

$$D_{\epsilon,\Theta}(\theta) \;=\; \int_0^{\infty} D_\epsilon(r,\theta)\, dr \;=\; \tfrac{1}{2\,\pi}$$

Hence we have $D_\epsilon(r,\theta) = D_{\epsilon,R}(r)\; D_{\epsilon,\Theta}(\theta)$.

Note that $D_{\epsilon,R}(r)$ corresponds to the pdf of the *gamma distribution* with shape 2 and scale $1/\epsilon$. Figure 3 shows the graph of this function for various values of ϵ.

Thanks to the fact that R and Θ are independent, in order to draw a point (r,θ) from $D_\epsilon(r,\theta)$ it is sufficient to draw separately r and θ from $D_{\epsilon,R}(r)$ and $D_{\epsilon,\Theta}(\theta)$ respectively.

Since $D_{\epsilon,\Theta}(\theta)$ is constant, drawing θ is easy: it is sufficient to generate θ as a random number in the interval $[0, 2\pi)$ with uniform distribution.

We now show how to draw r. Following standard lines, we con-

Drawing a point (r, θ) from the polar Laplacian
1. draw θ uniformly in $[0, 2\pi)$
2. draw z uniformly in $[0, 1)$ and set $r = C_\epsilon^{-1}(z)$

Figure 4: Method to generate Laplacian noise.

sider the cumulative distribution function (cdf) $C_\epsilon(r)$:

$$C_\epsilon(r) = \int_0^r D_{\epsilon,R}(\rho)d\rho = 1 - (1 + \epsilon\, r)\, e^{-\epsilon\, r}$$

Intuitively, $C_\epsilon(r)$ (Fig 3(b)) represents the probability that the radius of the random point falls between 0 and r. Finally, we generate a random number z with uniform probability in the interval $[0, 1)$, and we set $r = C_\epsilon^{-1}(z)$. Note that

$$C_\epsilon^{-1}(z) = -\tfrac{1}{\epsilon}\big(W_{-1}(\tfrac{z-1}{e}) + 1\big)$$

where W_{-1} is the Lambert W function (the -1 branch), which can be computed efficiently.

4.2 Discretization

We discuss now how to approximate the Laplace mechanism on a grid $\mathcal{G}$ of discrete Cartesian coordinates. Let us recall the points (a) and (b) of the plan, in light of the development so far: Given the actual location x_0, report the point x in $\mathcal{G}$ obtained as follows:

(a) first, draw a point (r, θ) following the method in Figure 4,

(b) then, remap (r, θ) to the closest point x on $\mathcal{G}$.

We will denote by $K_\epsilon : \mathcal{G} \to \mathcal{P}(\mathcal{G})$ the above mechanism. In summary, $K_\epsilon(x_0)(x)$ represents the probability of reporting the point x when the actual point is x_0.

It is not obvious that the discretization preserves geo-indistinguishability, due to the following problem: In principle, each point x in $\mathcal{G}$ should gather the probability of the set of points for which x is the closest point in $\mathcal{G}$, namely

$$R(x) = \{y \in \mathbb{R}^2 \,|\, \forall x' \in \mathcal{G}.\ d(y, x') \leq d(y, x')\}$$

However, due to the finite precision of the machine, the noise generated according to (a) is already discretized in accordance with the polar system. Let $\mathcal{W}$ denote the discrete set of points actually generated in (a). Each of those points (r, θ) is drawn with the probability of the area between r, $r + \delta_r$, θ and $\theta + \delta_\theta$, where δ_r and δ_θ denote the precision of the machine in representing the radius and the angle respectively. Hence, step (b) generates a point x in $\mathcal{G}$ with the probability of the set $R_\mathcal{W}(x) = R(x) \cap \mathcal{W}$. This introduces some irregularity in the mechanism, because the region associated to $R_\mathcal{W}(x)$ has a different shape and area depending on the position of x relatively to x_0. The situation is illustrated in Figure 5 with $R_0 = R_\mathcal{W}(x_0)$ and $R_1 = R_\mathcal{W}(x_1)$.

Geo-indistinguishability of the discretized mechanism.

We now analyze the privacy guarantees provided by our discretized mechanism. We show that the discretization preserves geo-indistinguishability, at the price of a degradation of the privacy parameter ϵ.

For the sake of generality we do not require the step units along the two dimensions of $\mathcal{G}$ to be equal. We will call them *grid units*, and will denote by u and v the smaller and the larger unit, respectively. We recall that δ_θ and δ_r denote the precision of the machine in representing θ and r, respectively. We assume that $\delta_r \leq r_{\max}\delta_\theta$.

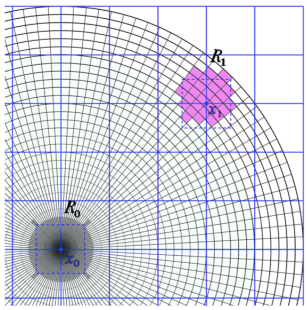


Figure 5: Remapping the points in polar coordinates to points in the grid.

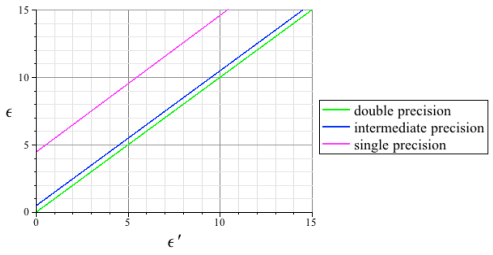


Figure 6: The relation between ϵ and ϵ' for various precisions, and $r_{\max} = 10^2$ Km.

The following theorem states the geo-indistinguishability guarantees provided by our mechanism: $K_{\epsilon'}$ satisfies ϵ-geo-indistinguishability, within a range $r_{\max}$, provided that ϵ' is chosen in a suitable way that depends on ϵ, on the length of the step units of $\mathcal{G}$, and on the precision of the machine.

THEOREM 4.1. *Assume $r_{\max} < u/\delta_\theta$, and let $q = u/r_{\max}\delta_\theta$. Let $\epsilon, \epsilon' \in \mathbb{R}^+$ such that*

$$\epsilon' + \frac{1}{u} \ln \frac{q + 2\, e^{\epsilon' u}}{q - 2\, e^{\epsilon' u}} \leq \epsilon$$

Then $K_{\epsilon'}$ provides ϵ-geo-indistinguishability within the range of $r_{\max}$. Namely, if $d(x_0, x), d(x_0', x) \leq r_{\max}$ then:

$$K_{\epsilon'}(x_0)(x) \leq e^{\epsilon\, d(x_0, x_0')} K_{\epsilon'}(x_0')(x).$$

The difference between ϵ' and ϵ represents the extra noise needed to compensate the effect of discretization. Figure 6 shows that the needed extra noise can vary a lot depending on the precision of the machine, and that for double precision it is rather minor.

Note that in Theorem 4.1 the restriction about $r_{\max}$ is crucial. Namely, ϵ-geo-indistinguishability does not hold for arbitrary distances for any finite ϵ. Intuitively, this is because the step units of $\mathcal{W}$ (see Figure 5) become larger with the distance r from x_0. The step units of $\mathcal{G}$, on the other hand, remain the same. When the steps in $\mathcal{W}$ become larger than those of $\mathcal{G}$, some x's have an empty $R_\mathcal{W}(x)$. Therefore when x is far away from x_0 its probability may or may not be 0, depending on the position of x_0 in $\mathcal{G}$, which means that geo-indistinguishability cannot be satisfied.

4.3 Truncation

In practical applications we are typically interested in locations within a certain region. The Laplacian mechanisms described in previous sections, however, has the potential to generate points everywhere in the plane. If the user knows that the actual location

Sanitizing Algorithm for a Location – *NoisyPt*

Input: x // point to sanitize
ϵ // privacy parameter
u, v, δ_θ, δ_r // precision parameters – Section 4.2
$\mathcal{A}$ // region of acceptable locations – Section 4.2
Output: Sanitized version x' of input x

1. $q = {}^{u}/_{\mathcal{A}\delta_\theta}$;
2. $\epsilon' = safe\_\epsilon(\epsilon, u, v, q)$; // Theorem 4.2
3. Draw angle $\theta \sim Uniform(2\pi)$; // Figure 4
4. Draw radius $r \sim gamma(2, {}^{1}/_{\epsilon'})$; // Figure 4
5. $x' = Pt(x, \rho, \theta)$; // sanitized location
6. **if** $x' \notin \mathcal{A}$ **then** $x' = closestPt(\mathcal{A}, x, \rho, \theta)$; //truncation
7. **return** x';

Figure 7: Our sanitizing algorithm for a location.

is situated within a certain region, it is desirable that the reported location lies within the same region as well. To this purpose we propose a variant of the discrete Laplacian described in previous section, which generates points only within a specified region.

We assume that the specified region $\mathcal{A}$ of acceptable report points is a circle centered in o, and diameter $diam(\mathcal{A})$. *This region $\mathcal{A}$ is fixed, i.e. it does not depend on the actual location x_0.* Our truncated mechanism $K^T_{\epsilon'} : \mathcal{A} \to \mathcal{P}(\mathcal{A} \cap \mathcal{G})$ works like the discretized Laplacian of previous section, with the difference that, whenever the point generated in step (a) lies outside $\mathcal{A}$, we remap it to the closest point in $\mathcal{A} \cap \mathcal{G}$ (which necessarily will be on the perimeter of $\mathcal{A}$, modulo discretization).

We are now going to show that this new method satisfies geo-indistinguishability on all $\mathcal{A}$, provided that $r_{\max}$ is not smaller than $diam(\mathcal{A})$.

THEOREM 4.2. *Let $r_{\max}$, ϵ and ϵ' satisfy the premise of Theorem 4.1. If $r_{\max} \geq diam(\mathcal{A})$, then $K^T_{\epsilon'}$ provides ϵ-geo-indistinguishability within $\mathcal{A}$.*

In the following we generally assume $\mathcal{A} = r_{\max}$.

5. ENHANCING LBSS WITH PRIVACY

In this section we present a case study of our privacy mechanism in the context of LBSs. We assume a simple client-server architecture where users communicate via a trusted mobile application (the client – typically installed in a smart-phone) with an unknown/untrusted LBS provider (the server – typically running on the cloud). Hence, in contrast to other solutions proposed in the literature, our approach does not rely on trusted third-party servers.

In the following we distinguish between *mildly-location-sensitive* and *highly-location-sensitive* LBS applications.

The former category corresponds to LBS applications offering a service that does not heavily rely on the precision of the location information provided by the user. Examples of such applications are weather forecast applications and LBS applications for retrieval of certain kind of POI (like gas stations). Enhancing this kind of LBSs with geo-indistinguishability is relatively straightforward. It requires to implement the location perturbation mechanism presented in Section 4 on the client party of the LBS application and then report the sanitized location (instead of the real location) to the LBS server party. Figure 7 delineates a location sanitizing algorithm based on the techniques described in Section 4.2.

Our running example lies within the second category: For the user sitting at Café Les Deux Magots, information about restaurants nearby Champs Élysées is considerably less valuable than information about restaurants around his location. Enhancing highly-

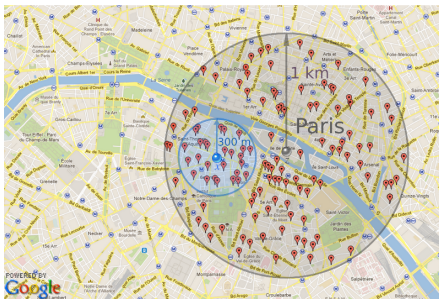


Figure 8: Retrieval information situation for private LBS

location-sensitive LBSs with privacy guarantees is considerably more challenging. Our approach is the following:

1. The algorithm illustrated in Figure 7 should be implemented on the client application in order to report to the LBS server party the user's approximate location z rather than his real location x.

2. Due to the fact that the information retrieved from the server is about POI nearby z, the area of POI information retrieval should be increased. In this way, if the user wishes to obtain information about POI within, say, 300 meters of x, the client application should request information about POI within, say, 1 km of z. Figure 8 illustrates the situation. We will refer to the blue circle as *area of interest* and to the grey circle as *area of retrieval*.

3. Finally, the client application should filter the retrieved POI information (depicted by the pins within the area of retrieval in Figure 8) in order to provide to the user with the desired information (depicted by pins within the user's area of interest in Figure 8).

Clearly, for our approach it is crucial that the area of interest is fully contained in the area of retrieval. However, the latter depends on a randomly generated location, hence such condition cannot be guaranteed. The client application could dynamically adjust the area of retrieval in order to ensure that it always contains the area of interest. However, this approach would jeopardize the privacy guarantees: on the one hand, the size of the area of retrieval would leak information about the user's real location and, on the other hand, the LBS provider would know with certainty that the user is located within the retrieval area. In order to provide geo-indistinguishability, the area of retrieval should be defined *independently* from the randomly generated location.

Our approach consists on *statically* defining the area of retrieval as a function of the security parameters (we use here ℓ and r, where $\ell = \epsilon\, r$) and of the area of interest. Our goal is to define an area of retrieval as small as possible (in order to avoid unnecessary bandwidth usage) in a way that the area of interest is contained in it with probability as high as possible. To this purpose, we adapt the notion of (α, δ)-usefulness [4] to our setting.

5.1 Usefulness of our mechanisms

A location perturbation mechanism $\mathcal{K}$ is (α, δ)-*useful* if for every location x, with probability at least $1 - \delta$, the reported location $z = \mathcal{K}(x)$ satisfies $d(x, z) \leq \alpha$. In the case of our mechanism, δ can be computed using the cdf of the Gamma distribution from which the radius is drawn. Figure 9 illustrates how our mechanism behaves with respect to (α,δ)-usefulness when providing ϵ-

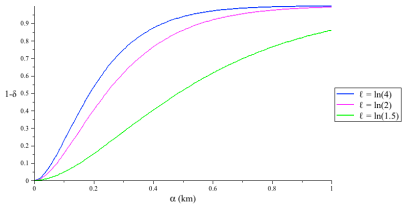


Figure 9: (α, δ)-usefulness for $r = 0.2$ and various values of ℓ.

geo-indistinguishability for $r = 0.2$ (as in our running example) and several values of ℓ.

It follows from the information in Figure 9, that a mechanism providing the privacy guarantees specified in our running example (ϵ-geo-indistinguishability, with $\ell = \ln(4)$ and $r = 0.2$) generates an approximate location z falling within 1 km of the user's location x with probability 0.99, falling within 690 meters with probability 0.95, falling within 560 meters with probability 0.9, and falling within 390 meters with probability 0.75.

We now have all the necessary ingredients to define an area of retrieval containing the area of interest with a given probability. Note that an area of retrieval with radius, say, r_A contains the area of interest with radius say, r_I, with probability at least $1 - \delta$ if the mechanism used to generate the reported location is (α,δ)-useful, for an $\alpha \leq r_A - r_I$.

Therefore, by setting r_A to 1 km in our running example and since our mechanism is (0.69, 0.05)-useful, it is guaranteed that the retrieval area contains the area of interest with probability at least 0.95.

5.2 Further challenges: using a LBS multiple times

As discussed in Section 3.3, geo-indistinguishability can be naturally extended to multiple locations. In short, the idea of being *ℓ-private within r* remains the same but for all locations simultaneously. In this way the locations, say, x_1, x_2 of a user employing the LBS twice remain indistinguishable from all pair of locations at (point-wise) distance at most r (ie, from all pairs x_1', x_2' such that $d(x_1, x_1') \leq r$ and $d(x_2, x_2') \leq r$).

A simple way of obtaining geo-indistinguishability guarantees when performing multiple queries is to employ our technique for protecting single locations to *independently* generate approximate locations for each of the user's locations. In this way, a user performing n queries via a mechanism providing ϵ-geo-indistinguishability enjoys $n\epsilon$-geo-indistinguishability (see Section 3.3).

This solution might be satisfactory when the number of queries to perform remains fairly low, but in other cases impractical, due to the privacy degradation. It is worth noting that the canonical technique for achieving standard differential privacy (based on adding noise according to the Laplace distribution) suffers of the same privacy degradation problem (ϵ increases linearly on the number of queries). Several articles in the literature focus on this problem (see [28] for instance). We believe that the principles and techniques used to deal with this problem for standard differential privacy could be adapted to our scenario (either directly or motivationally).

6. COMPARISON WITH OTHER METHODS

In this section we compare the privacy of our mechanism with that of others proposed in the literature. Of course it is not interesting to make a comparison in terms of geo-indistinguishability, since other mechanisms usually do not satisfy this property. We will consider, instead, the rather natural Bayesian notion of privacy proposed in [32]. In order to make a fair comparison, we will tune the parameters so to obtain the same quality of service, measured according to [32], and also in terms of the notion of usefulness [4] used in the previous section.

We consider the obfuscation mechanism over "regions" presented in [32], which gives optimal privacy for a given quality of service, and a given prior of the adversary modeling his side knowledge. The authors of [32] have actually provided a tool, which produces the optimal mechanism. This tool needs the map to be divided into a finite number of regions. We will experiment with a map divided in 81 square regions of 100m of side length, forming a 9×9 grid.

Note that the construction of the mechanism is done assuming a specific prior, and that in presence of a different adversary the optimality is not guaranteed. This dependency on the prior is a key difference with respect to our approach, which abstracts from the adversary's side information.

The other method that we will compare to ours is a simple cloaking one, in which the area of interest is divided in *zones*, and we report the zone in which the real location is situated. This simple cloaking satisfies k-anonymity where k is the number of locations within each zone. We will consider, for simplicity, a division shaped as a (larger) grid. Figure 10 illustrates the setting: the small squares with black borders are the regions considered for the application of the tool in [32]. The larger squares with blue borders are the zones considered in this second method. For instance, any point situated in one of the regions 1, 2, 3, 10, 11, 12, 19, 20 or 21, would be reported as zone 1.

Privacy Metric.

As already stated, we will use the privacy metric proposed in [32], which is called *LP* (Location Privacy) and defined as the *expected estimation error* of an *optimal adversary*. Namely:

$$LP = \sum_{r,r',\hat{r} \in R} \pi(r) K(r)(r') h(\hat{r}|r') d(\hat{r}, r)$$

where R is the set of regions, π is the prior distribution of the user over the regions, K is the mechanism, i.e. $K(r)(r')$ gives the probability that the real region r is reported as r', h is the optimal remap for the adversary, representing the probability that the reported region is remapped into $\hat{r}$, and d is the distance between regions. Calculating LP is relatively easy, which makes it a suitable option for the comparison. However, we need to assume that that the map is divided into a finite number of regions, hence also for our mechanism and the cloaking mechanism we consider the area of interest as divided into a 9×9 grid of 100m of side length.

Setting and results.

Our mechanism is already tuned to be mapped into a discrete grid (see Section 4). Let us make more precise the way we proceed in order to compute *LP*:

1. When obfuscating a region r, we considered the location to be the center of that region.
2. We then apply our usual mechanism to this location, and get an obfuscated location as result.
3. Finally, we remap this obfuscated location to the closest region r' of the grid, and report r' as the obfuscated region.

For the cloaking mechanism, we consider the map to be divided into a grid of 3×3 zones, where each zone contains 9 regions (cfr. Figure 10). When obfuscating a region r, the mechanism reports

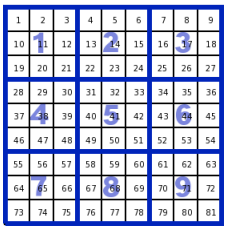


Figure 10: The division of the map into regions and zones.

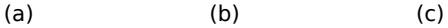


Figure 11: Priors considered for the experiments

the region at the center of its corresponding zone. For instance, for any of the regions 1, 2, 3, 10, 11, 12, 19, 20 or 21, the reported region would be 11.

In order to perform a fair comparison, it is important that the parameters of each mechanism are set in such a way that the *Service Quality Loss* (*SQL*) is the same for each of them. The *SQL* is defined as the *expected distance* between the reported point and the real one [32]. It is worth noting that for the optimal mechanism in [32] *SQL* and LP coincide (when the mechanism is used in presence of the same adversary for which it has been designed), i.e. the adversary does not need to make any remapping. It turns out that this is the case also for our mechanism and for the cloaking mechanism, when the adversary's prior is the uniform one. For our experiments, we fix the value of *SQL* to the one of the cloaking mechanism, i.e. 107.03m. We find that in order to obtain such *SQL* for our mechanism we need to set $\epsilon = 0.0162$ (the difference with ϵ' in this case is negligible).

Figure 11 illustrates the priors that we consider: in each case, the probability distribution is accumulated in the regions in the purple area, and distributed uniformly over them. Note that it is not interesting to consider the uniform distribution over the whole map, since, as explained before, on that prior all the mechanisms under consideration give the same result.

Figure 12 illustrates the results we obtain in terms of *LP*, where (a), (b) and (c) refer to the priors in Figure 11. The optimal mechanism is considered in two instances: the one designed exactly for the prior for which it is used (optimal rp – rp stands for real prior), and the one designed for the uniform distribution on all the map (optimal unif. – which is not necessarily optimal for the priors considered here). As we can see, our method offers the best *LP* among the mechanisms which do not depend on the prior, or are designed with a fixed prior (optimal unif.). When the prior has a more circular symmetry the performance approaches the one of optimal rp (the optimal mechanism).

Finally, we compare our mechanism to the cloaking mechanism using the notion of usefulness of [4], as it seems a very natural notion for the case of LBS applications. We cannot compare the optimal method of [32] because its construction is tied to the *SQL* (and there is no 1-1 correspondence between usefulness and *SQL*).

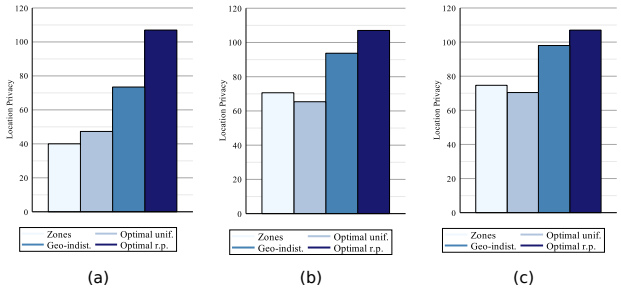


Figure 12: Location Privacy for *SQL* $= 107.03$m.

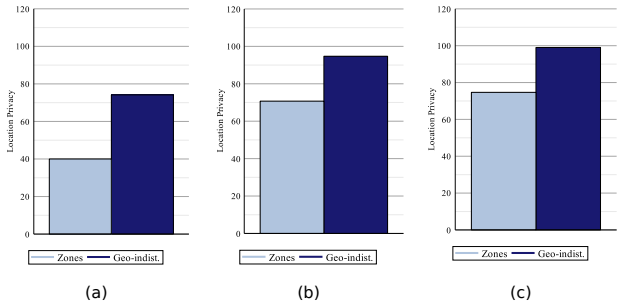


Figure 13: Location Privacy for $\alpha = (\sqrt{2} \cdot 150)$m and $\delta = 0.01$.

We fix r_I and r_A to be 200m and $(\sqrt{2} \cdot 150 + 200)$m respectively, and $\delta = 0.01$. It turns out that with such parameters ϵ must be 0.016. Figure 13 illustrates the values of *SQL* for the two mechanisms, with the same three priors as before. As we can see, our mechanism outperforms the cloaking mechanism in all the three cases.

7. RELATED WORK

Much of the related work has been already discussed in Section 2, here we only mention the works that were not reported there. There are excellent works and surveys [33, 22, 30]) that summarize the different threats, methods, and guarantees in the context of location privacy.

LISA [6] provides location privacy by preventing an attacker from relating any particular point of interest (POI) to the user's location. That way, the attacker cannot infer which POI the user will visit next. The privacy metric used in this work is m-*unobservability*. The method achieves m-unobservability if, with high probability, the attacker cannot relate the estimated location to at least m different POIs in the proximity.

SpaceTwist [35] reports a fake location (called the "anchor") and queries the geolocation system server incrementally for the nearest neighbors of this fake location until the k-nearest neighbors of the real location are obtained.

In a recent paper [25] it has been shown that, due to finite precision and rounding effects of floating-point operations, the standard implementations of the Laplacian mechanism result in an irregular distribution which causes the loss of the property of differential privacy. In [14] the study has been extended to the planar Laplacian, and to any kind of finite-precision semantics. The same paper proposes a solutions for the truncated version of the planar laplacian, based on a snapping meccanism, which maintains the level of privacy at the cost of introducing an additional amount of noise.

8. CONCLUSION AND FUTURE WORK

In this paper we have presented a framework for achieving privacy in location-based applications, taking into account the desired level of protection as well as the side-information that the attacker might have. The core of our proposal is a new notion of privacy, that we call geo-indistinguishability, and a method, based on a bivariate version of the Laplace function, to perturbate the actual location. We have put a strong emphasis in the formal treatment of the privacy guarantees, both in giving a rigorous definition of geo-indistinguishability, and in providing a mathematical proof that our method satisfies such property. We also have shown how geo-indistinguishability relates to the popular notion of differential privacy. Finally, we have illustrated the applicability of our method on a POI-retrieval service, and we have compared it with other mechanisms in the literature, showing that it outperforms those which do not depend on the prior.

In the future we aim at extending our method to cope with more complex applications, possibly involving the sanitization of several (potentially related) locations. One important aspect to consider when generating noise on several data is the fact that their correlation may degrade the level of protection. We aim at devising techniques to control the possible loss of privacy and to allow the composability of our method.

9. REFERENCES

[1] Pew Internet & American Life Project. `www.pewinternet.org`.

[2] C. A. Ardagna, M. Cremonini, E. Damiani, S. D. C. di Vimercati, and P. Samarati. Location privacy protection through obfuscation-based techniques. In *Proc. of DAS*, volume 4602 of *LNCS*, pages 47–60. Springer, 2007.

[3] B. Bamba, L. Liu, P. Pesti, and T. Wang. Supporting anonymous location queries in mobile environments with privacygrid. In *Proc. of WWW*, pages 237–246. ACM, 2008.

[4] A. Blum, K. Ligett, and A. Roth. A learning theory approach to non-interactive database privacy. In *Proc. of STOC*, pages 609–618. ACM, 2008.

[5] K. Chatzikokolakis, E. Andrés, Miguel, E. Bordenabe, Nicolás, and C. Palamidessi. Broadening the scope of Differential Privacy using metrics. In *Proc. of PETS*. IEEE, 2013. To appear. Tech. Rep. available at: `http://hal.inria.fr/hal-00767210`.

[6] Z. Chen. *Energy-efficient Information Collection and Dissemination in Wireless Sensor Networks.* PhD thesis, University of Michigan, 2009.

[7] R. Cheng, Y. Zhang, E. Bertino, and S. Prabhakar. Preserving user location privacy in mobile data management infrastructures. In *Privacy Enhancing Technologies, 6th Int. Workshop, PET 2006, Cambridge, UK, June 28-30, 2006, Revised Selected Papers*, volume 4258 of *LNCS*, pages 393–412. Springer, 2006.

[8] J. E. Dobson and P. F. Fisher. Geoslavery. *Technology and Society Magazine, IEEE*, 22(1):47–52, 2003.

[9] M. Duckham and L. Kulik. A formal model of obfuscation and negotiation for location privacy. In *Proc. of PERVASIVE*, volume 3468 of *LNCS*, pages 152–170. Springer, 2005.

[10] C. Dwork. Differential privacy. In *Proc. of ICALP*, volume 4052 of *LNCS*, pages 1–12. Springer, 2006.

[11] C. Dwork. A firm foundation for private data analysis. *Communications of the ACM*, 54(1):86–96, 2011.

[12] C. Dwork, M. Hardt, T. Pitassi, O. Reingold, and R. S. Zemel. Fairness through awareness. In *Proc. of ITCS*, pages 214–226. ACM, 2012.

[13] C. Dwork, F. Mcsherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Proc. of TCC*, volume 3876 of *LNCS*, pages 265–284. Springer, 2006.

[14] I. Gazeau, D. Miller, and C. Palamidessi. Preserving differential privacy under finite-precision semantics. In *Proc. of QAPL*, 2013. To appear.

[15] B. Gedik and L. Liu. Location privacy in mobile systems: A personalized anonymization model. In *Proc. of ICDCS*, pages 620–629. IEEE, 2005.

[16] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan. Private queries in location based services: anonymizers are not necessary. In *Proc. of SIGMOD*, pages 121–132. ACM, 2008.

[17] M. Gruteser and D. Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proc. of MobiSys*. USENIX, 2003.

[18] S.-S. Ho and S. Ruan. Differential privacy for location pattern mining. In *Proc. of SPRINGL*, pages 17–24. ACM, 2011.

[19] B. Hoh and M. Gruteser. Protecting location privacy through path confusion. In *SecureComm*, pages 194–205. IEEE, 2005.

[20] A. Khoshgozaran and C. Shahabi. Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy. In *Proc. of SSTD*, volume 4605 of *LNCS*, pages 239–257. Springer, 2007.

[21] H. Kido, Y. Yanagisawa, and T. Satoh. Protection of location privacy using dummies for location-based services. In *Proc. of ICDE Workshops*, page 1248, 2005.

[22] J. Krumm. A survey of computational location privacy. *Personal and Ubiquitous Computing*, 13(6):391–399, 2009.

[23] K. Lange and J. S. Sinsheimer. Normal/independent distributions and their applications in robust regression. *J. of Comp. and Graphical Statistics*, 2(2):175–198, 1993.

[24] A. Machanavajjhala, D. Kifer, J. M. Abowd, J. Gehrke, and L. Vilhuber. Privacy: Theory meets practice on the map. In *Proc. of ICDE*, pages 277–286. IEEE, 2008.

[25] I. Mironov. On significance of the least significant bits for differential privacy. In *Proc. of CCS*, pages 650–661. ACM, 2012.

[26] M. F. Mokbel, C.-Y. Chow, and W. G. Aref. The new casper: Query processing for location services without compromising privacy. In *Proc. of VLDB*, pages 763–774. ACM, 2006.

[27] J. Reed and B. C. Pierce. Distance makes the types grow stronger: a calculus for differential privacy. In *Proc. of ICFP*, pages 157–168. ACM, 2010.

[28] A. Roth and T. Roughgarden. Interactive privacy via the median mechanism. In *Proc. of STOC*, pages 765–774, 2010.

[29] P. Shankar, V. Ganapathy, and L. Iftode. Privately querying location-based services with sybilquery. In *Proc. of UbiComp*, pages 31–40. ACM, 2009.

[30] K. G. Shin, X. Ju, Z. Chen, and X. Hu. Privacy protection for users of location-based services. *IEEE Wireless Commun*, 19(2):30–39, 2012.

[31] R. Shokri, G. Theodorakopoulos, J.-Y. L. Boudec, and J.-P. Hubaux. Quantifying location privacy. In *Proc. of S&P*, pages 247–262. IEEE, 2011.

[32] R. Shokri, G. Theodorakopoulos, C. Troncoso, J.-P. Hubaux, and J.-Y. L. Boudec. Protecting location privacy: optimal strategy against localization attacks. In *Proc. of CCS*, pages

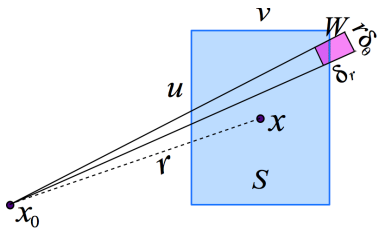


Figure 14: Bounding the probability of x in the discrete Laplacian.

617–627. ACM, 2012.

[33] M. Terrovitis. Privacy preservation in the dissemination of location data. *SIGKDD Explorations*, 13(1):6–18, 2011.

[34] M. Xue, P. Kalnis, and H. Pung. Location diversity: Enhanced privacy protection in location based services. In *Proc. of LoCA*, volume 5561 of *LNCS*, pages 70–87. Springer, 2009.

[35] M. L. Yiu, C. S. Jensen, X. Huang, and H. Lu. Spacetwist: Managing the trade-offs among location privacy, query performance, and query accuracy in mobile services. In *Proc. of ICDE*, pages 366–375. IEEE, 2008.

APPENDIX

In this appendix we provide the technical details that have been omitted from the main body of the paper.

THEOREM 4.1. *Assume* $r_{\max} < {}^{u}/_{\delta_\theta}$, *and let* $q = {}^{u}/_{r_{\max}\delta_\theta}$. *Let* $\epsilon, \epsilon' \in \mathbb{R}^+$ *such that*

$$\epsilon' + \frac{1}{u} \ln \frac{q + 2\, e^{\epsilon' u}}{q - 2\, e^{\epsilon' u}} \leq \epsilon$$

Then $K_{\epsilon'}$ *provides* ϵ*-geo-indistinguishability within the range of* $r_{\max}$. *Namely, if* $d(x_0, x), d(x_0', x) \leq r_{\max}$ *then:*

$$K_{\epsilon'}(x_0)(x) \leq e^{\epsilon\, d(x_0, x_0')} K_{\epsilon'}(x_0')(x).$$

PROOF. The case in which $x_0 = x_0'$ is trivial. We consider therefore only the case in which $x_0 \neq x_0'$. Note that in this case $d(x_0, x_0') \geq u$. We proceed by determining an upper bound on $K_{\epsilon'}(x_0)(x)$ and a lover bound on $K_{\epsilon'}(x_0')(x)$ for generic x_0, x_0' and x such that $d(x_0, x), d(x_0', x) \leq r_{\max}$. Let S be the set of points for which x is the closest point in $\mathcal{G}$, namely:

$$S = R(x) = \{y \in \mathbb{R}^2 \,|\, \forall x' \in \mathcal{G}.\ d(y, x') \leq d(y, x')\}$$

Ideally, the points remapped in x would be exactly those in S. However, due to the finite precision of the machine, the points actually remapped in x are those of $R_{\mathcal{W}}(x)$ (see Section 4.2). Hence the probability of x is that of S plus or minus the small rectangles[3] W of size $\delta_r \times r\, \delta_\theta$ at the border of S, where $r = d(x_0, x)$, see Figure 14. Let us denote by S_W the total area of these small rectangles W on one of the sides of S. Since $d(x_0, x) \leq r_{\max} < {}^{u}/_{\delta_\theta}$, and $\delta_r < r_{\max}\delta_\theta$, we have that S_W is less than ${}^{1}/_{q}$ of the area of S, where $q = {}^{u}/_{r_{\max}\delta_\theta}$. The probability density on this area differs at most by a factor $e^{\epsilon' u}$ from that of the other points in S. Finally, note that on two sides of S the rectangles W contribute positively

[3] W is actually a fragment of a circular crown, but since δ_θ is very small, it approximates a rectangle. Also, the side of W is not exactly $r\, \delta_\theta$, it is a number in the interval $[(r - {}^{u}/_{\sqrt{2}})\, \delta_\theta, (r + {}^{u}/_{\sqrt{2}})\, \delta_\theta]$. However ${}^{u}/_{\sqrt{2}}\, \delta_\theta$ is very small with respect to the other quantities involved, hence we consider negligible this difference.

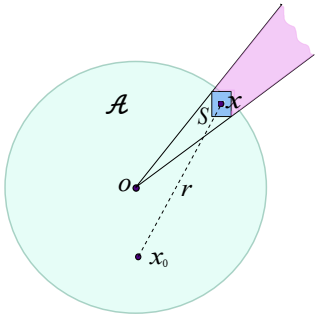


Figure 15: Probability of x in the truncated discrete laplacian.

to $K_{\epsilon'}(x_0)(x)$, while on two sides they contribute negatively. Summarizing, we have:

$$K_{\epsilon'}(x_0)(x) \ \leq\ (1 + \frac{2\, e^{\epsilon' u}}{q}) \int_S D_{\epsilon'}(x_0)(x_1) ds \qquad (3)$$

and

$$(1 - \frac{2\, e^{\epsilon' u}}{q}) \int_S D_{\epsilon'}(x_0')(x_1) ds \ \leq\ K_{\epsilon'}(x_0')(x) \qquad (4)$$

Observe now that

$$\frac{D_{\epsilon'}(x_0)(x_1)}{D_{\epsilon'}(x_0')(x_1)} = e^{-\epsilon'(d(x_0, x_1) - d(x_0', x_1))}$$

By triangular inequality we obtain

$$D_{\epsilon'}(x_0)(x_1) \leq e^{\epsilon'\, d(x_0, x_0')} D_{\epsilon'}(x_0')(x_1)$$

from which we derive

$$\int_S D_{\epsilon'}(x_0)(x_1) ds \leq e^{\epsilon'\, d(x_0, x_0')} \int_S D_{\epsilon'}(x_0')(x_1) ds \qquad (5)$$

from which, using (3), (5), and (4), we obtain

$$K_{\epsilon'}(x_0)(x) \leq e^{\epsilon'\, d(x_0, x_0')}\, K_{\epsilon'}(x_0')(x)\, \frac{q + 2\, e^{\epsilon' u}}{q - 2\, e^{\epsilon' u}} \qquad (6)$$

Assume now that

$$\epsilon' + \frac{1}{u} \ln \frac{q + 2\, e^{\epsilon' u}}{q - 2\, e^{\epsilon' u}} \leq \epsilon$$

Since we are assuming $d(x_0, x_0') \geq u$, we derive:

$$e^{\epsilon'\, d(x_0, x_0')}\, \frac{q + 2\, e^{\epsilon' u}}{q - 2\, e^{\epsilon' u}} \leq e^{\epsilon\, d(x_0, x_0')} \qquad (7)$$

Finally, from (6) and (7), we conclude. □

THEOREM 4.2. *Let* $r_{\max}$, ϵ *and* ϵ' *satisfy the premise of Theorem 4.1. If* $r_{\max} \geq diam(\mathcal{A})$, *then* $K_{\epsilon'}^{T}$ *provides* ϵ*-geo-indistinguishability within* $\mathcal{A}$.

PROOF. The proof proceeds like the one for Theorem 4.1, except when $R(x)$ is on the border of $\mathcal{A}$. In this latter case, the probability on x is given not only by the probability on $R(x)$ (plus or minus the small rectangles W – see the proof of Theorem 4.1), but also by the probability of the part C of the cone determined by o, $R(x)$, and lying outside $\mathcal{A}$ (see Figure 15). Following a similar reasoning as in the proof of Theorem 4.1 we get

$$K_{\epsilon'}^{T}(x_0)(x) \ \leq\ (1 + \frac{2\, e^{\epsilon' u}}{q}) \int_{S \cup C} D_{\epsilon'}(x_0)(x_1) ds$$

and

$$(1 - \frac{2\, e^{\epsilon' u}}{q}) \int_{S \cup C} D_{\epsilon'}(x'_0)(x_1) ds \;\leq\; K^T_{\epsilon'}(x'_0)(x)$$

The rest follows as in the proof of Theorem 4.1. $\square$

Checking Equality and Regularity for Normed BPA with Silent Moves*

Yuxi Fu

BASICS, Department of Computer Science, Shanghai Jiao Tong University
MOE-MS Key Laboratory for Intelligent Computing and Intelligent Systems

Abstract. The decidability of weak bisimilarity on normed BPA is a long standing open problem. It is proved in this paper that branching bisimilarity, a standard refinement of weak bisimilarity, is decidable for normed BPA and that the associated regularity problem is also decidable.

1 Introduction

In [BBK87] Baeten, Bergstra and Klop proved a surprising result that strong bisimilarity between context free grammars without empty production is decidable. The decidability is in sharp contrast to the well known fact that language equivalence between these grammars is undecidable. After [BBK87] decidability and complexity issues of equivalence checking of infinite systems *à la* process algebra have been intensively investigated. As regards BPA, Hüttel and Stirling [HS91] improved Baeten, Bergstra and Klop's proof by a more straightforward one using tableau system. Hüttel [Hüt92] then repeated the tableau construction for branching bisimilarity on totally normed BPA processes. Later Hirshfeld [Hir96] applied the tableau method to the weak bisimilarity on the totally normed BPA. An affirmative answer to the decidability of the strong bisimilarity on general BPA is given by Christensen, Hüttel and Stirling by applying the technique of bisimulation base [CHS92].

The complexity aspect of BPA has also been investigated over the years. Balcazar, Gabarro and Santha [BGS92] pointed out that strong bisimilarity is P-hard. Huynh and Tian [HT94] showed that the problem is in Σ_2^p, the second level of the polynomial hierarchy. Hirshfeld, Jerrum and Moller [HJM96] completed the picture by offering a remarkable polynomial algorithm for the strong bisimilarity of normed BPA. For the general BPA, Burkart, Caucal and Steffen [BCS95] showed that the strong bisimilarity problem is elementary. They claimed that their algorithm can be optimized to get a 2-EXPTIME upper bound. A further elaboration of the 2-EXPTIME upper bound is given in [Jan12] with the introduction of infinite regular words. The current known best lower bound of the problem, EXPTIME, is obtained by Kiefer [Kie13], improving both the PSPACE lower bound result and its proof of Srba [Srb02]. Much less is known about the weak bisimilarity on BPA. Stříbrná's PSPACE lower bound [Stř98] is subsumed

* F.V. Fomin et al. (Eds.): ICALP 2013, Part II, LNCS 7966, pp. 244-255, 2013.

by both the result of Srba [Srb02] and that of Mayr [May03], all of which are subsumed by Kiefer's recent result. A slight modification of Mayr's proof shows that the EXPTIME lower bound holds for the branching bisimilarity as well.

It is generally believed that weak bisimilarity, as well as branching bisimilarity, on BPA is decidable. There has been however a lack of technique to resolve the difficulties caused by silent transitions. This paper aims to advance our understanding of the decidability problems of BPA in the presence of silent transitions. The main contributions of the paper are as follows:

- We introduce branching norm, which is the least number of nontrivial actions a process has to do to become an empty process. With the help of this concept one can carry out a much finer analysis on silent actions than one would have using weak norm. Branching norm turns out to be crucial in our approach.
- We reveal that in normed BPA the length of a state preserving silent transition sequence can be effectively bounded. As a consequence we show that branching bisimilarity on normed BPA processes can be approximated by a sequence of finite branching bisimulations.
- We establish the decidability of branching bisimilarity on normed BPA by constructing a sound and complete tableau system for the equivalence.
- We demonstrate how to derive the decidability of the associated regularity problem from the decidability of the branching bisimilarity of normed BPA.

The result of this paper is significantly stronger than previous decidability results on the branching bisimilarity of totally normed BPA [Hüt92,CHT95]. It is easy to derive effective size bound for totally normed BPA since a totally normed BPA process with k variable occurrences has a norm at least k. For the same reason right cancellation property holds. Hence the decidability. The totality condition makes the branching bisimilarity a lot more like strong bisimilarity.

2 Branching Bisimilarity for BPA

A *basic process algebra* (BPA for short) Γ is a triple $(\mathcal{V}, \mathcal{A}, \Delta)$ where $\mathcal{V} = \{X_1, \ldots . X_n\}$ is a finite set of *variables*, $\mathcal{A} = \{a_1, \ldots . a_m\} \cup \{\tau\}$ is a finite set of *actions* ranged over by ℓ, and Δ is a finite set of *transition rules*. The special symbol τ denotes a *silent* action. A *BPA process* defined in Γ is an element of the set $\mathcal{V}^*$ of finite string of element of $\mathcal{V}$. The set $\mathcal{V}$ will be ranged over by capital letters and $\mathcal{V}^*$ by lower case Greek letters. The empty string is denoted by ϵ. We will use $=$ for the grammar equality on $\mathcal{V}^*$. A transition rule is of the form $X \stackrel{\ell}{\longrightarrow} \alpha$, where ℓ ranges over $\mathcal{A}$. The transitional semantics is closed under *composition* in the sense that $X\gamma \stackrel{\ell}{\longrightarrow} \alpha\gamma$ for all γ whenever $X \stackrel{\ell}{\longrightarrow} \alpha$. We shall assume that every variable of a BPA is defined by at least one transition rule and every action in $\mathcal{A}$ appears in some transition rule. Accordingly we sometimes refer to a BPA by its set of transition rules. We write $\longrightarrow$ for $\stackrel{\tau}{\longrightarrow}$ and $\Longrightarrow$ for the reflexive transitive closure of $\stackrel{\tau}{\longrightarrow}$. The set $\mathcal{A}^*$ will be ranged over by ℓ^*. If $\ell^* = \ell_1 \ldots \ell_k$ for some $k \geq 0$, then $\alpha \stackrel{\ell^*}{\longrightarrow} \alpha'$ stands for $\alpha \stackrel{\ell_1}{\longrightarrow} \alpha_1 \ldots \stackrel{\ell_{k-1}}{\longrightarrow} \alpha_{k-1} \stackrel{\ell_k}{\longrightarrow} \alpha'$ for some $\alpha_1, \ldots, \alpha_{k-1}$. We say that α' is a *descendant* of α if $\alpha \stackrel{\ell^*}{\longrightarrow} \alpha'$ for some ℓ^*.

A BPA process α is *normed* if there are some actions $\ell_1, \ldots \ell_j$ such that $\alpha \xrightarrow{\ell_1} \ldots \xrightarrow{\ell_j} \epsilon$. A process is *unnormed* if it is not normed. The *norm* of a BPA process α, denoted by $\|\alpha\|$, is the least k such that $\alpha \xrightarrow{\ell_1} \ldots \xrightarrow{\ell_k} \epsilon$ for some $\ell_1, \ldots \ell_k$. A *normed BPA*, or *nBPA*, is one in which every variable is normed.

For each given BPA Δ, we introduce the following notations:

- m_Δ is the number of transition rules; and n_Δ is the number of variables.
- r_Δ is $\max\left\{|\gamma| \;\middle|\; X \xrightarrow{\lambda} \gamma \in \Delta\right\}$, where $|\gamma|$ denotes the length of γ.
- $\|\Delta\|$ is $\max\{\|X_i\| \mid 1 \le i \le n_\Delta \text{ and } X_i \text{ is normed}\}$.

Each of m_Δ, n_Δ, r_Δ and $\|\Delta\|$ can be effectively calculated from Δ.

2.1 Branching Bisimilarity

The idea of the branching bisimilarity of van Glabbeek and Weijland [vGW89] is that not all silent actions can be ignored. What can be ignored are those that do not change system states irreversibly. For BPA we need to impose additional condition to guarantee congruence. In what follows $x\mathcal{R}y$ stands for $(x, y) \in \mathcal{R}$.

Definition 1. *A symmetric relation $\mathcal{R}$ on BPA processes is a* branching bisimulation *if the following statements are valid whenever $\alpha\mathcal{R}\beta$:*

1. *If $\beta\mathcal{R}\alpha \xrightarrow{\ell} \alpha'$ then one of the following statements is valid:*
   (i) $\ell = \tau$ and $\alpha'\mathcal{R}\beta$.
   (ii) $\beta \Longrightarrow \beta''\mathcal{R}\alpha$ for some β'' such that $\beta'' \xrightarrow{\ell} \beta'\mathcal{R}\alpha'$ for some β'.
2. *If $\alpha = \epsilon$ then $\beta \Longrightarrow \epsilon$.*

The branching bisimilarity $\simeq$ *is the largest branching bisimulation.*

The branching bisimilarity $\simeq$ satisfies the standard property of observational equivalence stated in the next lemma [vGW89].

Lemma 1. *Suppose $\alpha_0 \xrightarrow{\tau} \alpha_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} \alpha_k \simeq \alpha_0$. Then $\alpha_0 \simeq \alpha_1 \simeq \ldots \simeq \alpha_k$.*

Using Lemma 1 it is easy to show that $\simeq$ is a congruence and that whenever $\beta \simeq \alpha \xrightarrow{\ell} \alpha'$ is simulated by $\beta \xrightarrow{\tau} \beta_1 \xrightarrow{\tau} \beta_2 \ldots \xrightarrow{\tau} \beta_k \xrightarrow{\ell} \beta'$ such that $\beta_k \simeq \alpha$ and $\beta' \simeq \alpha'$ then $\beta \simeq \beta_1 \simeq \ldots \simeq \beta_k$.

Having defined an equality for BPA, we can formally draw a line between the silent actions that change the capacity of systems and those that do not. We say that a silent action $\alpha \xrightarrow{\tau} \alpha'$ is *state preserving* if $\alpha \simeq \alpha'$; it is a *change-of-state* if $\alpha \not\simeq \alpha'$. We will write $\alpha \to \alpha'$ if $\alpha \xrightarrow{\tau} \alpha'$ is state preserving and $\alpha \xrightarrow{\iota} \alpha'$ if it is a change-of-state. The reflexive and transitive closure of $\to$ is denoted by $\to^*$. Since both external actions and change-of-state silent actions must be explicitly bisimulated, we let $\jmath$ range over the set $(\mathcal{A} \setminus \{\tau\}) \cup \{\iota\}$. So $\alpha \xrightarrow{\jmath} \alpha'$ means either $\alpha \xrightarrow{a} \alpha'$ for some $a \neq \tau$ or $\alpha \xrightarrow{\iota} \alpha'$.

Let's see an example.

Example 1. The BPA Γ_1 is defined by the following transition rules:

$$A \xrightarrow{a} A,\ A \xrightarrow{\tau} \epsilon,\ B \xrightarrow{b} B,\ B \xrightarrow{\tau} \epsilon,\ C \xrightarrow{a} C,\ C \xrightarrow{b} C,\ C \xrightarrow{\tau} \epsilon.$$

Clearly $AC \simeq BC$, although $A \not\simeq B$. In this example all variables are normed.

2.2 Bisimulation Base

An *axiom system* $\mathcal{B}$ is a finite set of equalities on nBPA processes. An element $\alpha = \beta$ of $\mathcal{B}$ is called an *axiom*. Write $\mathcal{B} \vdash \alpha = \beta$ if the equality $\alpha = \beta$ can be derived from the axioms of $\mathcal{B}$ by repetitive use of any of the three equivalence rules and two congruence rules. For our purpose the most useful axiom systems are those that generate branching bisimulations. These are bisimulation bases originally due to Caucal. The following definition is Hüttel's adaptation to the branching scenario [Hüt92].

Definition 2. *A finite axiom system $\mathcal{B}$ is a* bisimulation base *if the following bisimulation base property hold for every axiom (α_0, β_0) of $\mathcal{B}$:*

1. *If $\beta_0 \longrightarrow \beta_1 \longrightarrow \ldots \longrightarrow \beta_n \xrightarrow{\ell} \beta'$ then there are $\alpha_1, \ldots, \alpha_n, \alpha'$ such that $\mathcal{B} \vdash \beta_1 = \alpha_1$, $\ldots$, $\mathcal{B} \vdash \beta_n = \alpha_n$, $\mathcal{B} \vdash \beta' = \alpha'$ and the following hold:*
   (i) *For each i with $0 \leq i < n$, either $\alpha_i = \alpha_{i+1}$, or $\alpha_i \longrightarrow \alpha_{i+1}$, or there are $\alpha_i^1, \ldots, \alpha_i^{k_i}$ such that $\alpha_i \longrightarrow \alpha_i^1 \longrightarrow \ldots \longrightarrow \alpha_i^{k_i} \longrightarrow \alpha_{i+1}$ and $\mathcal{B} \vdash \beta_i = \alpha_i^1$, $\ldots$, $\mathcal{B} \vdash \beta_i = \alpha_i^{k_i}$.*
   (ii) *Either $\ell = \tau$ and $\alpha_n = \alpha'$, or $\alpha_n \xrightarrow{\ell} \alpha'$, or there are $\alpha_n^1, \ldots, \alpha_n^{k_n}$ such that $\alpha_n \longrightarrow \alpha_n^1 \longrightarrow \ldots \longrightarrow \alpha_n^{k_n} \xrightarrow{\ell} \alpha'$ and $\mathcal{B} \vdash \beta_n = \alpha_n^1$, $\ldots$, $\mathcal{B} \vdash \beta_n = \alpha_n^{k_n}$.*
2. *If $\beta_0 = \epsilon$ then either $\alpha_0 = \epsilon$ or $\alpha_0 \longrightarrow \alpha_1 \longrightarrow \ldots \longrightarrow \alpha_k \longrightarrow \epsilon$ for some $\alpha_1, \ldots, \alpha_k$ with $k \geq 0$ such that $\mathcal{B} \vdash \alpha_1 = \epsilon$, $\ldots$, $\mathcal{B} \vdash \alpha_k = \epsilon$.*
3. *The conditions symmetric to 1 and 2.*

The next lemma justifies the above definition [Hüt92].

Lemma 2. *If $\mathcal{B}$ is a bisimulation base then $\mathcal{B}^{\vdash} = \{(\alpha, \beta) \mid \mathcal{B} \vdash \alpha = \beta\} \subseteq \simeq$.*

Proof. If $\mathcal{B} \vdash \alpha = \beta$, then an inductive argument shows that there exist $\gamma_1\delta_1\lambda_1$, $\gamma_2\delta_2\lambda_2, \gamma_3\delta_3\lambda_3, \ldots, \gamma_{k-1}\delta_{k-1}\lambda_{k-1}, \gamma_k\delta_k\lambda_k$ and $\delta_1', \ldots, \delta_k'$ for $k \geq 1$ such that $\alpha = \gamma_1\delta_1\lambda_1, \gamma_k\delta_k'\lambda_k = \beta$ and $\gamma_1\delta_1\lambda_1 \; \mathcal{B} \; \gamma_1\delta_1'\lambda_1 = \gamma_2\delta_2\lambda_2 \; \mathcal{B} \; \gamma_2\delta_2'\lambda_2 = \ldots \gamma_{k-1}\delta_{k-1}'\lambda_{k-1} = \gamma_k\delta_k\lambda_k \; \mathcal{B} \; \gamma_k\delta_k'\lambda_k$. The transitive closure makes it easy to see that $\mathcal{B}^{\vdash}$ satisfies the bisimulation base property. Consequently it is a branching bisimulation. □

3 Approximation of Branching Bisimilarity

To look at the algebraic property of the branching bisimilarity $\simeq$ more closely, we introduce a notion of normedness appropriate for the equivalence.

Definition 3. *The* branching norm *of an nBPA process α is the least number k such that $\exists \jmath_1 \ldots \jmath_k . \exists \alpha_1 \ldots \alpha_k . \alpha \rightarrow^* \xrightarrow{\jmath_1} \alpha_1 \rightarrow^* \xrightarrow{\jmath_2} \ldots \alpha_{k-1} \rightarrow^* \xrightarrow{\jmath_k} \alpha_k \rightarrow^* \epsilon$. The branching norm of α is denoted by $\|\alpha\|_b$.*

For example the branching norm of B defined by $\{B \xrightarrow{a} B, B \xrightarrow{\tau} \epsilon\}$ is 1. It is easy to prove that if $\alpha \simeq \beta$ then $\|\alpha\|_b = \|\beta\|_b$ and that if $\|\alpha\|_b = 0$ then $\alpha \simeq \epsilon$. It follows that $\|\alpha'\|_b = \|\alpha\|_b$ whenever $\alpha \rightarrow^* \alpha'$. Also notice that $\|\alpha\|_b \leq \|\alpha\|$.

An important property of branching norm is stated next.

Lemma 3. *Suppose α is normed. Then $\alpha \simeq \delta\alpha$ if and only if $\|\alpha\|_b = \|\delta\alpha\|_b$.*

Proof. If $\|\alpha\|_b = \|\delta\alpha\|_b$ then every silent action sequence from $\delta\alpha$ to α must contain only state preserving silent transitions according to Lemma 1. Moreover there must exist such a silent action path for otherwise $\|\alpha\|_b < \|\delta\alpha\|_b$. □

It does not follow from $\alpha \simeq \delta\alpha$ that $\delta \simeq \epsilon$. A counter example is given by the BPA defined in Example 1. One has $AC \simeq C \simeq BC$. But clearly $\epsilon \not\simeq A \not\simeq B \not\simeq \epsilon$. To deal with situations like this we need the notion of relative norm.

Definition 4. *The* relative norm $\|\alpha\|_b^\sigma$ *of α with respect to σ is the least k such that $\alpha\sigma \rightarrow^* \xrightarrow{\jmath_1} \alpha_1\sigma \ldots \alpha_{k-1}\sigma \rightarrow^* \xrightarrow{\jmath_k} \alpha_k\sigma \rightarrow^* \sigma$ for some $\jmath_1, \ldots, \jmath_k, \alpha_1, \ldots, \alpha_k$.*

Obviously $0 \leq \|\alpha\|_b^\sigma \leq \|\alpha\|_b$. Returning to the BPA Γ_1 defined in Example 1, we see that $\|A\|_b^B = 1$ and $\|A\|_b^C = 0$. Using the notion of relative norm we may introduce the following terminologies:

- A transition $X\sigma \xrightarrow{\ell} \eta\sigma$ is *norm consistent* if either $\|\eta\|_b^\sigma = \|X\|_b^\sigma$ and $\ell = \tau$ or $\|\eta\|_b^\sigma = \|X\|_b^\sigma - 1$ and $\ell \neq \tau \vee \ell = \iota$.
- If $X\sigma \longrightarrow \eta\sigma$ is norm consistent with $\|X\|_b^\sigma > 0$, then it is *norm splitting* if at least two variables in η have (smaller) nonzero relative norms in $\eta\sigma$.

For an nBPA Δ no silent transition sequence contains more than $\|\Delta\|_b$ norm splitting transitions, where $\|\Delta\|_b$ is $\max\{\|X_i\|_b \mid 1 \leq i \leq n_\Delta \text{ and } X_i \text{ is normed}\}$.

The crucial property about relative norm is described in the following lemma.

Lemma 4. *Let $\alpha, \beta, \delta, \gamma$ be normed with $\|\alpha\|_b^\gamma = \|\beta\|_b^\delta$. If $\alpha\gamma \simeq \beta\delta$ then $\gamma \simeq \delta$.*

Proof. Suppose $\|\alpha\|_b^\gamma = \|\beta\|_b^\delta$. Now $\|\alpha\|_b^\gamma + \|\gamma\|_b = \|\alpha\gamma\|_b = \|\beta\delta\|_b = \|\beta\|_b^\delta + \|\delta\|_b$. Therefore $\|\gamma\|_b = \|\delta\|_b$. A norm consistent action sequence $\alpha\gamma \rightarrow^* \xrightarrow{\jmath_1} \ldots \rightarrow^* \xrightarrow{\jmath_k} \rightarrow^* \gamma$ must be matched up by $\beta\delta \rightarrow^* \xrightarrow{\jmath_1} \ldots \rightarrow^* \xrightarrow{\jmath_k} \beta'\delta$ for some β'. Clearly $\|\beta'\delta\|_b = \|\gamma\|_b = \|\delta\|_b$. It follows from Lemma 3 that $\delta \simeq \beta'\delta \simeq \gamma$. □

Lemma 4 describes a weak form of left cancelation property. The general left cancelation property fails. Fortunately there is a nice property of nBPA that allows us to control the size of common suffix of a pair of bisimilar processes.

Definition 5. *A process α is* irredundant over γ *if $\|\alpha\|_b^\gamma > 0$. It is* redundant over γ *if $\|\alpha\|_b^\gamma = 0$. A process α is* head irredundant *if either $\alpha = \epsilon$ or $\alpha = X\alpha'$ for some X, α' such that $\alpha \not\simeq \alpha'$. It is head redundant otherwise. We write $Hirred(\alpha)$ to indicate that α is head irredundant. A process α is* completely irredundant *if every suffix of α is head irredundant. We write $Cirred(\alpha)$ to mean that α is completely irredundant.*

If α is normed, then α is irredundant over γ if and only if $\alpha\gamma \not\simeq \gamma$. In nBPA a redundant process consists solely of redundant variables.

Lemma 5. *Suppose $X_1, \ldots, X_k, \sigma$ are normed. Then $X_1 \ldots X_k$ is redundant over σ if and only if X_i is redundant over σ for every $X_i \in \{X_1, \ldots, X_k\}$.*

Proof. Suppose $X_1, \ldots, X_k, \sigma$ are normed and $X_1 \ldots X_k$ is redundant over σ. Then $X_1 \ldots X_k\sigma \Longrightarrow X_2 \ldots X_k\sigma \Longrightarrow \ldots \Longrightarrow X_k\sigma \Longrightarrow \sigma \simeq X_1 \ldots X_k\sigma$. It follows from Lemma 1 that $X_1 \ldots X_k\sigma \simeq X_2 \ldots X_k\sigma \simeq \ldots \simeq X_k\sigma \simeq \sigma$. We are done by using the congruence property. □

For each σ, let the *redundant set* $\mathcal{R}_\sigma$ of σ be $\{X \mid X\sigma \simeq \sigma\}$. Let $\mathcal{V}(\alpha)$ be the set of variables appearing in α. We have two useful corollaries.

Corollary 1. *Suppose α, σ are normed. Then $\alpha\sigma \simeq \sigma$ if and only if $\mathcal{V}(\alpha) \subseteq \mathcal{R}_\sigma$.*

Corollary 2. *Suppose $\alpha, \beta, \sigma_0, \sigma_1$ are defined in an nBPA and $\mathcal{R}_{\sigma_0} = \mathcal{R}_{\sigma_1}$. Then $\alpha\sigma_0 \simeq \beta\sigma_0$ if and only if $\alpha\sigma_1 \simeq \beta\sigma_1$.*

Proof. Suppose $\mathcal{R}_{\sigma_0} = \mathcal{R}_{\sigma_1}$. Let $\mathcal{S}$ be $\{(\alpha\sigma_0, \beta\sigma_0) \mid \alpha\sigma_1 \simeq \beta\sigma_1\}$. It is not difficult to see that $\mathcal{S} \cup \simeq$ is a branching bisimulation. □

We now take a look at the state preserving transitions of nBPA processes. We are particularly interested in knowing if the quotient set $\{\theta \mid \alpha \to^* V\theta\}/\simeq$ of the equivalence classes is finite for every nBPA process α and every variable V. It turns out that all such sets are finite with effective size bound.

Lemma 6. *For each nBPA process $\alpha = X\omega$, there is an effective bound H_α, uniformly computable from α, satisfying the following: If $\alpha \to^* V\theta$ then $\alpha \to^* V\eta$ for some η such that $\theta \simeq \eta$ and the length of $\alpha \to^* V\eta$ is no more than H_α.*

Proof. The basic idea is to show that in an effectively bounded number of steps α can reach, via norm consistent and norm splitting silent transitions, terms $V\theta$ with all possible variable V and all possible relative norm of V. We then apply Lemma 4. The bound H_α is computed from $|\alpha|$ and the transition system. □

Under the assumption $\gamma \not\simeq \beta\gamma$ we can repeat the proof of Lemma 6 for $\beta\gamma$ in a way that γ is not affected. Hence the next corollary.

Corollary 3. *Suppose $\alpha, \beta\gamma$ are nBPA processes and $\gamma \not\simeq \beta\gamma$. If $\beta\gamma \simeq \alpha \xrightarrow{\jmath} \alpha'$, then there is a transition sequence $\beta\gamma \to^* \beta''\gamma \xrightarrow{\jmath} \beta'\gamma$ with its length bounded by H_β such that $\beta''\gamma \simeq \alpha$ and $\beta'\gamma \simeq \alpha'$.*

We are now in a position to prove the following.

Proposition 1. *The relation $\not\simeq$ on nBPA processes is semi-decidable.*

Proof. We define $\simeq_k$, the branching bisimilarity up to depth k, by exploiting Corollary 3. The inductive definition is as follows:

- $\alpha \simeq_0 \beta$ for all α, β.
- $\alpha \simeq_{i+1} \beta$ if the following condition and its symmetric version hold: If $\alpha \simeq_i \beta \xrightarrow{\ell} \beta'$ then one of the following statements is valid:
  (i) $\ell = \tau$ and $\alpha \simeq_i \beta'$.
  (ii) $\alpha \Longrightarrow \alpha'' \simeq_i \beta$ for some α'' such that $\alpha'' \xrightarrow{\ell} \alpha' \simeq_i \beta'$ for some α' and the length of $\alpha \Longrightarrow \alpha''$ is bounded by H_α.

Each $\simeq_k$ is decidable. Using Corollary 3 one easily sees that $\simeq \subseteq \bigcap_{k\in\omega} \simeq_k$. The proof of the converse inclusion is standard. □

4 Equality Checking

A straightforward approach to proving an equality between two processes is to construct a finite bisimulation tree for the equality. A tree of this kind has been called a tableau system [HS91,Hüt92]. To apply this approach we need to make sure that the following properties are satisfied: (i) Every tableau for an equality $\alpha = \beta$ is finite. (ii) The set of tableaux for an equality $\alpha = \beta$ is finite. We can achieve (i) by using Corollary 2 and Corollary 3. This is because if σ is long enough then according to Corollary 2 it can be decomposed into some $\sigma_0\sigma_1\sigma_2$ such that $\mathcal{R}_{\sigma_1\sigma_2} = \mathcal{R}_{\sigma_2}$. Then $\lambda\sigma_0\sigma_1\sigma_2 \simeq \gamma\sigma_0\sigma_1\sigma_2$ can be simplified to $\lambda\sigma_0\sigma_2 \simeq \gamma\sigma_0\sigma_2$. The equivalence provides a method to control the size of labels of a tableau. Now (ii) is a consequence of (i), Corollary 3 and König lemma.

The building blocks for tableaux are matches. Suppose $\alpha_0\alpha \not\equiv \alpha$ and $\beta_0\beta \not\equiv \beta$. A *match* for the equality $\alpha_0\alpha = \beta_0\beta$ *over* (α, β) is a finite symmetric relation $\{\gamma_i\alpha = \lambda_i\beta\}_{i=1}^k$ containing only those equalities accounted for in the following condition: For each transition $\alpha_0\alpha \xrightarrow{\ell} \alpha'\alpha$, one of the following holds:

- $\ell = \tau$ and $\alpha'\alpha = \beta_0\beta \in \{\gamma_i\alpha = \lambda_i\beta\}_{i=1}^k$;
- there is a sequence $\beta_0\beta \xrightarrow{\tau} \beta_1\beta \xrightarrow{\tau} \ldots \xrightarrow{\tau} \beta_n\beta \xrightarrow{\ell} \beta'\beta$, for $n < H_{\beta_0}$, such that $\{\alpha_0\alpha = \beta_1\beta, \ldots, \alpha_0\alpha = \beta_n\beta, \alpha'\alpha = \beta'\beta\} \subseteq \{\gamma_i\alpha = \lambda_i\beta\}_{i=1}^k$.

If $\alpha_0\sigma \not\equiv \sigma \not\equiv \beta_0\sigma$, a match for $\alpha_0\sigma = \beta_0\sigma$ over (σ, σ) is said to be a match for $\alpha_0\sigma = \beta_0\sigma$ over σ. The computable bound H_{β_0}, given by Corollary 3, guarantees that the number of matches for $\alpha_0\alpha = \beta_0\beta$ is effectively bounded.

Suppose α_0, β_0 are nBPA processes. A *tableau* for $\alpha_0 = \beta_0$ is a tree with each of its nodes labeled by an equality between nBPA processes. The root is labeled by $\alpha_0 = \beta_0$. We shall distinguish between *global tableau* and *local tableau*. The global tableau is the overall tableau whose root is labeled by the goal $\alpha_0 = \beta_0$. It is constructed from the rules given in Fig. 1. Decmp rule decomposes a goal into several subgoals. We shall find it useful to use SDecmp, which is a stronger version of Decmp. The side condition of SDecmp ensures that it is unnecessary to apply it consecutively. When applying Decmp rule we assume that an equality $\gamma\sigma = \sigma$, respectively $\sigma = \gamma\sigma$, is always decomposed in the following manner

$$\frac{\gamma\sigma = \sigma}{\sigma = \sigma \quad \{V\sigma = \sigma\}_{V \in \mathcal{V}(\gamma)}} \text{ respectively } \frac{\sigma = \gamma\sigma}{\sigma = \sigma \quad \{V\sigma = \sigma\}_{V \in \mathcal{V}(\gamma)}}.$$

Accordingly $\gamma = \epsilon$, respectively $\epsilon = \gamma$, is decomposed in the following fashion

$$\frac{\gamma = \epsilon}{\epsilon = \epsilon \quad \{V = \epsilon\}_{V \in \mathcal{V}(\gamma)}} \text{ respectively } \frac{\epsilon = \gamma}{\epsilon = \epsilon \quad \{V = \epsilon\}_{V \in \mathcal{V}(\gamma)}}.$$

SubstL and SubstR allow one to create common suffix for the two processes in an equality. ContrL and ContrR are used to remove a redundant variable inside a process. In the side conditions of these two rules, α_0, β_0 are the processes appearing in the root of the global tableau. ContrC deletes redundant variables from the common suffix of a node label whenever the size of the common suffix

Decmp	$\dfrac{\gamma\alpha = \lambda\beta}{\alpha = \beta \quad \{U\alpha = \alpha\}_{U\in\mathcal{V}(\gamma)} \quad \{V\beta = \beta\}_{V\in\mathcal{V}(\lambda)}}$	$\lvert\gamma\rvert + \lvert\lambda\rvert > 0$, $\forall U \in \mathcal{V}(\gamma).U \Longrightarrow \epsilon$, $\forall V \in \mathcal{V}(\lambda).V \Longrightarrow \epsilon$.
SDecmp	$\dfrac{\gamma\alpha = \lambda\beta}{\alpha = \beta \quad \{U\alpha = \alpha\}_{U\in\mathcal{V}(\gamma)} \quad \{V\beta = \beta\}_{V\in\mathcal{V}(\lambda)}}$	$\lvert\gamma\rvert + \lvert\lambda\rvert > 0$, $Hirred(\alpha)$, $Hirred(\beta)$, $\forall U \in \mathcal{V}(\gamma).U \Longrightarrow \epsilon$, $\forall V \in \mathcal{V}(\lambda).V \Longrightarrow \epsilon$.
Match	$\dfrac{\gamma\alpha = \lambda\beta}{\alpha_1\alpha = \beta_1\beta \ \ldots \ \alpha_k\alpha = \beta_k\beta}$	$\gamma\alpha \not\equiv \alpha$, $\lambda\beta \not\equiv \beta$, and $\{\alpha_i\alpha = \beta_i\beta\}_{i=1}^{k}$ is a match for $\gamma\alpha = \lambda\beta$ over (α, β).
SubstL	$\dfrac{\gamma\alpha = \lambda\beta}{\gamma\delta\beta = \lambda\beta}$	$\alpha = \delta\beta$ is the residual.
SubstR	$\dfrac{\gamma\alpha = \lambda\beta}{\gamma\alpha = \lambda\delta\alpha}$	$\delta\alpha = \beta$ is the residual.
ContrL	$\dfrac{\gamma Z\delta = \lambda}{\gamma\delta = \lambda \quad Z\delta = \delta}$	$Hirred(\delta)$, $Z \Longrightarrow \epsilon$ and $\lvert\gamma Z\delta\rvert > \max\{\lvert\alpha_0\rvert, \lvert\beta_0\rvert\}\lVert\Delta\rVert$.
ContrR	$\dfrac{\gamma = \lambda Z\delta}{\gamma = \lambda\delta \quad Z\delta = \delta}$	$Hirred(\delta)$, $Z \Longrightarrow \epsilon$ and $\lvert\lambda Z\delta\rvert > \max\{\lvert\alpha_0\rvert, \lvert\beta_0\rvert\}\lVert\Delta\rVert$.
ContrC	$\dfrac{\gamma\sigma'\sigma_0\sigma_1 = \lambda\sigma'\sigma_0\sigma_1}{\gamma\sigma'\sigma_1 = \lambda\sigma'\sigma_1 \quad \{V\sigma_1 = \sigma_1\}_{V\in\mathcal{V}(\sigma_0)}}$	$\lvert\sigma'\sigma_0\sigma_1\rvert > 2^{n\Delta}$, $\lvert\sigma_0\rvert > 0$, $Hirred(\sigma_1)$, $\forall V \in \mathcal{V}(\sigma_0).V \Longrightarrow \epsilon$.

Fig. 1. Rules for Global Tableaux

is over limit. Notice that all the side conditions on the rules are semi-decidable due to the semi-decidability of $\not\equiv$. So we can effectively enumerate tableaux.

In what follows a node $Z\eta = W\kappa$ to which Match rule is applied with the condition $Z\eta \not\equiv \eta \wedge W\kappa \not\equiv \kappa$ is called an *M-node*. A node of the form $Z\sigma = \sigma$ with σ being head irredundant is called a *V-node*. We now describe how a global tableau for $\alpha_0 = \beta_0$ is constructed. Assuming $\alpha_0 = \gamma X\alpha_1$ and $\beta_0 = \lambda Y\beta_1$ such that $X\alpha_1 \not\equiv \alpha_1$ and $Y\beta_1 \not\equiv \beta_1$, we apply the following instance of SDecmp rule:

$$\frac{\gamma X\alpha_1 = \lambda Y\beta_1}{X\alpha_1 = Y\beta_1 \quad \{UX\alpha_1 = X\alpha_1\}_{U\in\mathcal{V}(\gamma)} \quad \{VY\beta_1 = Y\beta_1\}_{V\in\mathcal{V}(\lambda)}}.$$

By definition $X\alpha_1 = Y\beta_1$ is an M-node and $\{UX\alpha_1 = X\alpha_1\}_{U\in\mathcal{V}(\gamma)} \cup \{VY\beta_1 = Y\beta_1\}_{V\in\mathcal{V}(\lambda)}$ is a set of V-nodes. These nodes are the roots of new subtableaux. Starting from $X\alpha_1 = Y\beta_1$ we apply Match rule under the condition that neither α_1 nor β_1 is affected. The application of Match rule is repeated to grow the subtableau rooted at $X\alpha_1 = Y\beta_1$. The construction of the tree is done in a breadth first fashion. So the tree grows level by level. At some stage we apply Decmp rule to all the current leaves. This particular application of Decmp must meet the following conditions: (i) Both α_1 and β_1 must be kept intact in all the current leaves; (ii) Either α_1 or β_1 is exposed in at least one current leaf. Choose a leaf labeled by either $\alpha_1 = \delta_1\beta_1$ for some δ_1 or by $\delta_1'\alpha_1 = \beta_1$ for some δ_1' and call it the *residual node* or *R-node*. Suppose the residual node is $\alpha_1 = \delta_1\beta_1$. All the other current leaves, the *non-residual nodes*, must be labeled by an equality of the form $\gamma_1\alpha_1 = \lambda_1\beta_1$. A non-residual node with label $\gamma_1\alpha_1 = \lambda_1\beta_1$ is then

Localization	$\dfrac{\gamma\sigma'\sigma_0\sigma_1 = \lambda\sigma'\sigma_0\sigma_1}{\begin{array}{l}\gamma\sigma'\sigma_1 = \lambda\sigma'\sigma_1 \\ \{X_i\sigma_1 = \sigma_1\}_{i\in I} \\ \{X_i\sigma_0\sigma_1 = \sigma_0\sigma_1\}_{i\in I}\end{array}}$	$\begin{array}{l}\lvert\gamma\rvert > 0 \text{ and } \lvert\lambda\rvert > 0;\ \lvert\sigma'\sigma_0\sigma_1\rvert > 2^{n_\Delta}, \\ 2^{n_\Delta} \geq \lvert\sigma_1\rvert > 0 \text{ and } \lvert\sigma_0\rvert > 0; \\ Cirred(\sigma'\sigma_0\sigma_1) \text{ and } Cirred(\sigma'\sigma_1); \\ \gamma\sigma'\sigma_0\sigma_1 \not\equiv \sigma'\sigma_0\sigma_1,\ \gamma\sigma'\sigma_1 \not\equiv \sigma'\sigma_1; \\ \lambda\sigma'\sigma_0\sigma_1 \not\equiv \sigma'\sigma_0\sigma_1,\ \lambda\sigma'\sigma_1 \not\equiv \sigma'\sigma_1; \\ I \cap J = \emptyset,\ I \cup J = \{1, \ldots, n_\Delta\}; \\ \forall j \in J.\ X_j\sigma_0\sigma_1 \not\equiv \sigma_0\sigma_1 \text{ and } X_j\sigma_1 \not\equiv \sigma_1; \\ X_i \Longrightarrow \epsilon \text{ for all } i \in I.\end{array}$

Fig. 2. Rule for Local Tableaux

attached with a single child labeled by $\gamma_1\delta_1\beta_1 = \lambda_1\beta_1$. This is an application of SubstL rule. Now we can recursively apply the global tableau construction to $\gamma_1\delta_1\beta_1 = \lambda_1\beta_1$ to produce a new subtableau. The treatment of a V-node child, say $UX\alpha_1 = X\alpha_1$, is similar. We keep applying Match rule over α_1 as long as the side condition is met. At certain stage we apply Decmp rule to all the leaves. The application should meet the following conditions: (i) No occurrence of α_1 is affected; (ii) There is an application of Decmp that takes the following shape

$$\frac{\gamma_1\alpha_1 = \lambda_1\alpha_1}{\alpha_1 = \alpha_1 \qquad \{V\alpha_1 = \alpha_1\}_{V\in\mathcal{V}(\gamma_1)} \qquad \{V\alpha_1 = \alpha_1\}_{V\in\mathcal{V}(\lambda_1)}}.$$

We then recursively apply the tableau construction to create new subtableaux.

In the above construction the R-node $\alpha_1 = \delta_1\beta_1$ can be the root of a new subtableau, which might contain another R-node. In fact a chain of R-nodes is possible. ContrL/ContrR is used to control the size of R-nodes.

After an application of SubstL/SubstR rule we may get a *C-node* $\alpha'\sigma'\sigma_0\sigma_1 = \beta'\sigma'\sigma_0\sigma_1$ if ContrC rule is applicable. Once a C-node appears, we immediately apply ContrC rule to reduce the size of its common suffix. Intuitively we should apply ContrC rule sufficiently often so that the common suffix becomes completely irredundant. Eventually either the length of the common suffix has become no more than 2^{n_Δ}, in which case we continue to build up the global tableau, or Localization rule as defined in Fig. 2 is applicable, in which case we get an *L-node*. The soundness of Localization rule is guaranteed by Corollary 2.

Suppose Localization rule is applied to an L-node $\alpha'\sigma'\sigma_0\sigma_1 = \beta'\sigma'\sigma_0\sigma_1$:

$$\frac{\alpha'\sigma'\sigma_0\sigma_1 = \beta'\sigma'\sigma_0\sigma_1}{\{X_i\sigma_1 = \sigma_1\}_{i\in I} \qquad \alpha'\sigma'\sigma_1 = \beta'\sigma'\sigma_1 \qquad \{X_i\sigma_0\sigma_1 = \sigma_0\sigma_1\}_{i\in I}}.$$

The node $\alpha'\sigma'\sigma_1 = \beta'\sigma'\sigma_1$ is a new L-node. We call $\{X_i \mid i \in I\}$ the *R-set* of the new L-node. If the size of the common suffix of $\alpha'\sigma'\sigma_1 = \beta'\sigma'\sigma_1$ is still larger than 2^{n_Δ}, we continue to apply Localization rule. Otherwise we get an *L-root*, which is the root of a local tableau. Now suppose $\alpha'\sigma'\sigma_1 = \beta'\sigma'\sigma_1$ is an L-root. The construction of the local tableau should stick to two principles described as follows: (I) *Locality*. No application of Decmp, SDecmp, SubstL, SubsR and ContrC should ever affect $\sigma'\sigma_1$ or any suffix of $\sigma'\sigma_1$. Notice that applications of

SubstL or SubstR can never affect $\sigma'\sigma_1$ or any suffix of $\sigma'\sigma_1$. (II) *Consistency.* Suppose $\gamma\alpha = \lambda\beta$ is a node to which Match rule is applied using a match over (α, β). Then either $\sigma'\sigma_1$ is a suffix of both α and β, or $\alpha = \beta = \sigma''\sigma_1$ for some σ'' satisfying the following: (i) σ'' is a proper suffix of σ'; (ii) $\gamma = UZ$ and $\lambda = Z$ such that $Z\sigma''$ is a suffix of σ'; and (iii) the match is over $\sigma''\sigma_1$. The locality and consistency conditions basically say that choices made in the construction of the local tableau should not contradict to the fact that $\sigma'\sigma_1$ is completely irredundant.

The construction of a path in a tableau ends with a leaf. A *successful leaf* is either a node labeled by $\varsigma = \varsigma$ for some ς, or a node labeled by $\epsilon = V$ $(V = \epsilon)$ with $V \simeq \epsilon$, or a node that has the same label as one of its ancestors. An *unsuccessful leaf* is produced if the node is either labeled by $\epsilon = V$ $(V = \epsilon)$ with $V \not\simeq \epsilon$, or labeled by some $\varsigma = \varsigma'$ with distinct ς, ς' such that no rule is applicable to $\varsigma = \varsigma'$. A local tableau has additionally two new kind of successful/unsuccessful leaves: (i) An L-root is a successful leaf if it shares the same label with one of its ancestors that is also an L-root. (ii) Suppose $\alpha'\sigma'\sigma_0\sigma_1 = \beta'\sigma'\sigma_0\sigma_1$ is an L-node and its child $\alpha'\sigma'\sigma_1 = \beta'\sigma'\sigma_1$ is an L-root. In the local tableau rooted at $\alpha'\sigma'\sigma_1 = \beta'\sigma'\sigma_1$, a node of the form $Z\sigma_1 = \sigma_1$ is deemed as a leaf. It is a successful leaf if Z is in the R-set of the L-root; it is an unsuccessful leaf otherwise.

Tableau constructions always terminate. In fact we have the following.

Lemma 7. *The size of every tableau for an equality is effectively bounded. The number of tableaux for an equality is effectively bounded.*

A tableau is *successful* if all of its leaves are successful. Successful tableaux generate bisimulation bases.

Proposition 2. *Suppose $X\alpha, Y\beta$ are nBPA processes. Then $X\alpha \simeq Y\beta$ if and only if there is a successful tableau for $X\alpha = Y\beta$.*

Proof. If $X\alpha \simeq Y\beta$ we can easily construct a tableau using the bisimulation property, Corollary 2 and Corollary 3. Conversely suppose there is a successful tableau $\mathfrak{T}$ for $X\alpha = Y\beta$. Let $\mathcal{A} = \mathcal{A}_b \cup \mathcal{A}_z \cup \mathcal{A}_l$. The set $\mathcal{A}_b$ of basic axioms is given by $\{\gamma = \lambda \mid \gamma = \lambda \text{ is a label of a node in } \mathfrak{T}\}$. The set $\mathcal{A}_z$ is defined by

$$\mathcal{A}_z = \left\{ V\sigma = \theta\sigma, \theta\sigma = \sigma \;\middle|\; \begin{array}{l} V\sigma = \sigma \text{ is in } \mathcal{A}_b, \text{ and } V \stackrel{\tau}{\Longrightarrow} \theta \stackrel{\tau}{\Longrightarrow} \epsilon \\ \text{is a chosen shortest path from } V \text{ to } \epsilon. \end{array} \right\}.$$

Suppose $\gamma\sigma'\sigma_1 = \lambda\sigma'\sigma_1$ is an L-root and $\gamma\sigma'\sigma_0\sigma_1 = \lambda\sigma'\sigma_0\sigma_1$ is its parent. A node $\eta\sigma'\sigma_1 = \kappa\sigma'\sigma_1$ in the local tableau rooted at $\gamma\sigma'\sigma_1 = \lambda\sigma'\sigma_1$ must be lifted to $\eta\sigma'\sigma_0\sigma_1 = \kappa\sigma'\sigma_0\sigma_1$ in order to show that $\gamma\sigma'\sigma_0\sigma_1 = \lambda\sigma'\sigma_0\sigma_1$ satisfies the bisimulation base property. Since local tableaux may be nested, the node might have several lifted versions. The set $\mathcal{A}_l$ is defined to be the collection of all such lifted pairs. We can prove by induction on the nodes of the tableau, starting with the leaves, that $\mathcal{A}$ is a bisimulation base. Hence $X\alpha \simeq Y\beta$ by Lemma 2. □

Our main result follows from Proposition 1, Lemma 7 and Proposition 2.

Theorem 1. *The branching bisimilarity on nBPA processes is decidable.*

5 Regularity Checking

Regularity problem asks if a process is bisimilar to a finite state process. For strong regularity problem of nBPA, Kučera [Kuč96] showed that it is decidable in polynomial time. Srba [Srb02] observed that it is actually NL-complete. The decidability of strong regularity problem for the general BPA was proved by Burkart, Caucal and Steffen [BCS95,BCS96]. It was shown to be PSPACE-hard by Srba [Srb02]. The decidability of almost all weak regularity problems of process rewriting systems [May00] are unknown. The only exception is Jancar and Esparza's undecidability result of weak regularity problem of Petri Net and its extension [JE96]. Srba [Srb03] proved that weak regularity is both NP-hard and co-NP-hard for nBPA. Using a result by Srba [Srb03], Mayr proved that weak regularity problem of nBPA is EXPTIME-hard [May03].

The present paper improves our understanding of the issue by the following.

Theorem 2. *The regularity problem of $\simeq$ on nBPA is decidable.*

Proof. One proves by a combinatorial argument that, in the transition tree of an infinite state BPA process, (i) a path $V_0\sigma_0 \xrightarrow{\ell_1^*} V_1\sigma_1 \xrightarrow{\ell_2^*} V_2\sigma_2 \ldots \xrightarrow{\ell_m^*} V_m\sigma_m$ exists such that (ii) $|\sigma_0| < |\sigma_1| < |\sigma_2| < \ldots < |\sigma_m|$ and (iii) $\|V_0\sigma_0\|_b < \|V_1\sigma_1\|_b < \|V_2\sigma_2\|_b < \ldots < \|V_m\sigma_m\|_b$. We can choose m large enough such that $0 \leq i < j \leq m$ for some i, j satisfying $V_i = V_j$ and $\mathcal{R}_{\sigma_i} = \mathcal{R}_{\sigma_j}$. Let $\sigma_j = \sigma\sigma_i$ for some σ. Clearly $\|\sigma_i\|_b < \|\sigma_j\|_b$. Using Corollary 2 one can prove by induction that $\sigma^i\sigma_i \not\simeq \sigma^j\sigma_i$ whenever $i \neq j$. It is semi-decidable to find (i) with properties (ii,iii). The converse implication is proved by a tree construction using Theorem 1. □

6 Remark

For parallel processes (BPP/PN) with silent actions, the only known decidability result on equivalence checking is due to Czerwiński, Hofman and Lasota [CHL11]. This paper provides the analogous decidability result for the sequential processes (BPA/PDA) with silent actions. For further research one could try to apply the technique developed in this paper to general BPA and normed PDA.

Acknowledgement. I am indebted to He, Huang, Long, Shen, Tao, Yang, Yin and the anonymous referees. The support from NSFC (60873034, 61033002, ANR 61261130589) and STCSM (11XD1402800) is gratefully acknowledged.

References

[BBK87] J. Baeten, J. Bergstra, and J. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. In *PARLE'87*, pages 94–113. Lecture Notes in Computer Science 259, 1987.

[BCS95] O. Burkart, D. Caucal, and B. Steffen. An elementary bisimulation decision procedure for arbitrary context free processes. In *MFCS'95*, pages 423–433. Lecture Notes in Computer Science 969, Springer, 1995.

[BCS96] O. Burkart, D. Caucal, and B. Steffen. Bisimulation collapse and the process taxonomy. In *CONCUR'96*, pages 247–262. Lecture Notes in Computer Science 1119, Springer, 1996.

[BGS92] J. Balcazar, J. Gabarro, and M. Santha. Deciding bisimilarity is p-complete. *Formal Aspects of Computing*, 4:638–648, 1992.

[CHL11] W. Czerwiński, P. Hofman, and S. Lasota. Decidability of branching bisimulation on normed commutative context-free processes. In *CONCUR'11*, pages 528–542. Lecture Notes in Computer Science 6901, Springer, 2011.

[CHS92] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. In *CONCUR'92*, pages 138–147. Lecture Notes in Computer Science 630, Springer, 1992.

[CHT95] D. Caucal, D. Huynh, and L. Tian. Deciding branching bisimilarity of normed context-free processes is in σ_2^p. *Information and Computation*, 118:306–315, 1995.

[Hir96] Y. Hirshfeld. Bisimulation trees and the decidability of weak bisimulations. *Electronic Notes in Theoretical Computer Science*, 5:2–13, 1996.

[HJM96] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimilarity of normed context free processes. *Theoretical Computer Science*, 158(1-2):143–159, 1996.

[HS91] H. Hüttel and C. Stirling. Actions speak louder than words: Proving bisimilarity for context-free processes. In *LICS'91*, pages 376–386, 1991.

[HT94] T. Huynh and L. Tian. Deciding bisimilarity of normed context free processes is in σ_2^p. *Theoretical Computer Science*, 123:83–197, 1994.

[Hüt92] H. Hüttel. Silence is golden: Branching bisimilarity is decidable for context free processes. In *CAV'91*, pages 2–12. Lecture Notes in Computer Science 575, Springer, 1992.

[Jan12] P. Jančar. Bisimilarity on basic process algebra is in 2-exptime. 2012.

[JE96] P. Jančar and J. Esparza. Deciding finiteness of petri nets up to bisimulation. In *ICALP'96*, pages 478–489. Lecture Notes in Computer Science 1099, Springer, 1996.

[Kie13] S. Kiefer. Bpa bisimilarity is exptime-hard. *Information Processing Letters*, 113:101–106, 2013.

[Kuč96] A. Kučera. Regularity is decidable for normed bpa and normed bpp processes in polynomial time. In *SOFSEM'96*, pages 377–384. Lecture Notes in Computer Science 1175, Springer, 1996.

[May00] R. Mayr. Process rewrite systems. *Information and Computation*, 156:264–286, 2000.

[May03] R. Mayr. Weak bisimilarity and regularity of bpa is exptime-hard. In *EXPRESS'03*, 2003.

[Srb02] J. Srba. Strong bisimilarity and regularity of basic process algebra is pspace-hard. In *ICALP'02*, pages 716–727. Lecture Notes in Computer Science 2380, Springer, 2002.

[Srb03] J. Srba. Complexity of weak bisimilarity and regularity for BPA and BPP. *Mathematical Structures in Computer Science*, 13:567–587, 2003.

[Stř98] J. Stříbrná. Hardness results for weak bisimilarity of simple process algebras. *Electronic Notes in Theoretical Computer Science*, 18:179–190, 1998.

[vGW89] R. van Glabbeek and W. Weijland. Branching time and abstraction in bisimulation semantics. In *Information Processing'89*, pages 613–618. North-Holland, 1989.

Well-Structured Pushdown Systems, Part I: Decidable classes for Coverability*

Xiaojuan Cai[1] and Mizuhito Ogawa[2]

[1] BASICS Lab, Shanghai Jiao Tong University, China
cxj@sjtu.edu.cn
[2] Japan Advanced Institute of Science and Technology, Japan
mizuhito@jaist.ac.jp

Abstract. *Pushdown systems* (PDSs) nicely model single-thread recursive programs, and *well-structured transition systems* (WSTS), such as *vector addition systems*, are useful to represent non-recursive multi-thread programs. Our goal is to investigate *well-structured pushdown systems* (WSPDSs), pushdown systems with well-quasi-ordered control states and stack alphabet, to combine these ideas.
This paper focuses on decidable classes of coverability and extends P-automata techniques for configuration reachability of PDSs to those for coverability of WSPDSs, in forward and backward ways. A $Post^*$-automata (resp. Pre^*-automata) construction is combined with Karp-Miller acceleration (resp. ideal representation) to characterize the set of successors (resp. predecessors) of given configurations. We show decidability results of coverability, which include *recursive vector addition system with states* [1], *multi-set pushdown systems* [2, 3], and a WSPDS with finite control states and well-quasi-ordered stack alphabet.

1 Introduction

There are two directions of infinite (discrete) state systems. A *pushdown system* (PDS) consists of finite control states and finite stack alphabet, where a stack stores the context. It nicely models single-thread recursive programs. *Well-structured transition systems* (WSTS) [4, 5] consists of a well-quasi-ordered set of states, in which *Vector addition system* (VAS, or Petri Net) is a typical example. It often works for modeling dynamic thread creation of multi-thread program [6]. Our naive motivation comes from what happens when we combine them as a general framework for modeling recursive multi-thread programs.

Ramalingam [7] showed that a 3-thread recursive program with synchronization mechanism can solve *Post-correspondence-problem*. This is a natural result since a 2-stack PDS is Turing complete. Roughly speaking, there are two sources to be Turing complete in a 2-stack PDS. i) the depth of both stacks is unbounded. ii) the interleaving between two stacks can be arbitrarily many. By restricting i),

* This work is supported by the NSFC-JSPS joint project (61011140074) and NSFC projects (61003013,61100052)

Qadeer an Rehof proposed context-bounded pushdown model [8], in which the number of context switching is bounded. The idea is after a bounded number of context switching, only one stack can work, so that it is simulated by a single stack. Atig, *et.al.* further extended with dynamic thread creation [6].

By restricting ii), Sen *et. al.* [2] proposed Multi-set pushdown systems (Multi-set PDSs) to model multi-thread asynchronous programs, and Bouajjani and Emmi [1] proposed a Recursive Vector Addition System with States (RVASS) to model multi-thread programs with fork/join synchronizations. They showed decidability of the coverability and the state reachability, respectively. Note that the coverability lies between the configuration reachability and the state reachability. They are single stack PDSs with infinite control states and stack alphabet, which are beyond ordinary PDSs with finite control states and stack alphabet.

The *configuration reachability*, i.e., to determine whether a target configuration is reachable from an initial configuration, is decidable for ordinary PDSs. In implementation, P-automata construction is a popular technique, which can be tracked back to Büchi's seminal work [9], and has been clarified in [10–12]. There are two kinds of P-automata constructions. A $Post^*$ automaton computes the set of successor configurations from an initial configuration, and a Pre^* automaton computes the set of predecessor configurations from a target configuration.

Different from PDSs, a popular property of WSTSs is *coverability*, which is reachability from an initial configuration to a certain configuration that covers the target configuration. There are also forward and backward proof techniques. For instance, in case of VASs, Karp-Miller acceleration [13] is a typical instance of the former, which was generalized in [14, 15]. For the latter, an ideal (i.e., an upward closed set) representation is a typical technique [4, 5]. Note that the reachability is hard for WSTSs. For instance, the reachability of VASs stays decidable, but its proof requires deep insight on Presburger arithmetic [16, 17].

Our ultimate goal is to investigate *well-structured pushdown systems* (WSPDSs), pushdown systems with well-quasi-ordered control states and stack alphabet, to combine PDSs and WSTSs. This paper focuses on decidable classes of coverability and extends P-automata techniques for configuration reachability of PDSs to those for coverability of WSPDSs, in forward and backward ways. $Post^*$-automata (resp. Pre^*-automata) construction is combined with Karp-Miller acceleration (resp. ideal representation) to characterize the set of successors (resp. predecessors) of given configurations. We show decidability results of coverability, which include RVASSs [1], Multi-set PDSs [2, 3], and a WSPDS with finite control states and WQO stack alphabet. The first one extends the decidability of the state reachability of RVASSs [1] to that of the coverability.

Related Work

Combining PDSs and VASs is not new. Process rewrite system (PRS) [18] is a pioneer work on such combination. A PRS is a(n AC) ground term rewriting system, consisting of the sequential composition ".", the parallel composition "|| ", and finitely many constants, which can be regarded as a PDS with finite control states and vector stack alphabet. The decidability of the reachability

between ground terms was shown based on the reachability of a VAS. However, a PRS is rather weak to model multi-thread programs, since it cannot describe vector additions between adjacent stack frames during push/pop operations.

An RVASS [1], in which we are inspired, allows vector additions during pop rules. The state reachability was shown by reduction of an RVASS into a Branching VASS [19]. Our WSPDS framework extends the decidability result to the coverability. A more general framework is a WQO automaton [20], which is a WSTS with auxiliary storage (e.g., stacks and queues). Although in general undecidable, its coverability becomes decidable under the compatibility of *rank* functions with a WQO. An Multi-set PDS [3, 2] is such a instance.

To sum up, our contribution is a simplified framework, which has more focus on well-quasi-ordered stack alphabet, and a unified proof methodology based on extensions of P-automata techniques.

2 Preliminaries

2.1 Well-structured transition system

A *quasi-order* $(D, \leq)$ is a reflexive transitive binary relation on D. An upward closure of $X \subseteq D$, denoted by $X^{\uparrow}$, includes all elements in D larger than elements in X, i.e., $X^{\uparrow} = \{d \in D \mid \exists x \in X.x \leq d\}$). A subset I is an *ideal* if $I = I^{\uparrow}$. Similarly, a downward closure of $X \subseteq D$ is denoted by $X^{\downarrow} = \{d \in D \mid \exists x \in X.x \geq d\}$. We denote the set of all ideals by $\mathcal{I}(D)$. A quasi-order $(D, \leq)$ is a *well-quasi-order* (WQO) if, for each infinite sequence $a_1, a_2, a_3, \cdots$ in D, there exist i, j with $i < j$ and $a_i \leq a_j$.

Definition 1. *A* well-structured transition system *(WSTS) is a triplet* $M = \langle (P, \preceq), \Delta \rangle$ *where* $(P, \preceq)$ *is a WQO, and* $\Delta \subseteq P \times P$ *is the set of transitions. We write* $p \to q$ *if* $(p, q) \in \Delta$.

M is monotonic *if, for each* $p_1, q_1, p_2 \in P$, $p_1 \to q_1 \wedge p_1 \preceq p_2$ *implies* $\exists q_2 \,.\, p_2 \to q_2 \ \wedge q_1 \preceq q_2$.

Given two states $p, q \in P$, the *coverability* problem is to determine whether there exists some $q' \succeq q$ and $p \to^* q'$.

Vector addition systems (VAS) (equivalently, Petri net) are WSTSs, with vectors as states and additions as transition rules. The reachability problem of VAS is decidable [16, 17]. It is elegant, but too difficult to implement. The coverability also attracts attentions and is implemented, such as in **Pep**. *Karp-Miller acceleration* is an efficient technique for the coverability. If there is a descendant vector (wrt transitions) strictly larger than one of its ancestors on some coordinates, values at these coordinates are accelerated to ω.

There is an alternative backward method to decide coverability for a WSTS, beyond VASs. Starting from an ideal $\{q\}^{\uparrow}$, where q is the target state to be covered, its predecessors are repeatedly computed. Note that, for a monotonic WSTS and an ideal $I(\subseteq P)$, the predecessor set $pre(I) = \{p \in P \mid \exists q \in I.p \to q\}$ is also an ideal. Its termination is obtained by the following lemma.

Lemma 1. *[5] $(D, \leq)$ is a WQO, if, and only if, any infinite sequence $I_0 \subseteq I_1 \subseteq I_2 \subseteq \cdots$ in $\mathcal{I}(D)$ eventually stabilize.*

From now on, we denote $\mathbb{N}$ (resp. $\mathbb{Z}$) for the set of natural numbers (resp. integers), and $\mathbb{N}^k$ (resp. $\mathbb{Z}^k$) is the set of k-dimensional vectors over $\mathbb{N}$ (resp. $\mathbb{Z}$). As notational convention, $\boldsymbol{n}, \boldsymbol{m}$ are for vectors in $\mathbb{N}^k$, $\boldsymbol{z}, \boldsymbol{z}'$ are for vectors in $\mathbb{Z}^k$, $\widetilde{\boldsymbol{n}}, \widetilde{\boldsymbol{m}}$ are for sequences of vectors.

2.2 Pushdown system

We define a pushdown system (PDS) with extra rules, *simple-push* and *nonstandard-pop*. These extra rules do not appear in the standard definition, but they can be encoded into standard rules. For example, a non-standard pop rule $(p, \alpha\beta \to q, \gamma)$ can be split into $(p, \alpha \to p_\alpha, \epsilon)$ and $(p_\alpha, \beta \to q, \gamma)$ by adding an intermediate state p_α. However, later we will consider a PDS with infinite stack alphabet, and this encoding may change the context. For instance, when a PDS has finite states and infinite stack alphabet, the encoding of nonstandard pop rules make a PDS with both infinite states and stack alphabet.

Definition 2. *A* pushdown system (PDS) *is a triplet $\langle P, \Gamma, \Delta \rangle$ where*

- *P is a finite set of states,*
- *Γ is finite stack alphabet, and*
- *$\Delta \subseteq P \times \Gamma^{\leq 2} \times P \times \Gamma^{\leq 2}$ is a finite set of transitions, where $(p, v, q, w) \in \Delta$ is denoted by $(p, v \to q, w)$.*

We use $\alpha, \beta, \gamma, \cdots$ to range over Γ, and $w, v, \cdots$ over words in Γ^*. A *configuration* $\langle p, w \rangle$ is a pair of a state p and a stack content (word) w. As convention, we denote configurations by $c_1, c_2, \cdots$. One step transition $\hookrightarrow$ between configurations is defined as follows. $\hookrightarrow^*$ is the reflexive transitive closure of $\hookrightarrow$.

$$\text{inter}\frac{(p, \gamma \to p', \gamma') \in \Delta}{\langle p, \gamma w \rangle \hookrightarrow \langle p', \gamma' w \rangle} \quad \text{push}\frac{(p, \gamma \to p', \alpha\beta) \in \Delta}{\langle p, \gamma w \rangle \hookrightarrow \langle p', \alpha\beta w \rangle} \quad \text{pop}\frac{(p, \gamma \to p', \epsilon) \in \Delta}{\langle p, \gamma w \rangle \hookrightarrow \langle p', w \rangle}$$

$$\text{simple-push}\frac{(p, \epsilon \to p', \alpha) \in \Delta}{\langle p, w \rangle \hookrightarrow \langle p', \alpha w \rangle} \quad \text{nonstandard-pop}\frac{(p, \alpha\beta \to p', \gamma) \in \Delta}{\langle p, \alpha\beta w \rangle \hookrightarrow \langle p', \gamma w \rangle}$$

A PDS enjoys decidable *reachability*, i.e., given configurations $\langle p, w \rangle$, $\langle q, v \rangle$ with $p, q \in P$ and $w, v \in \Gamma^*$, decide whether $\langle p, w \rangle \hookrightarrow^* \langle q, v \rangle$.

3 WSPDS and P-automata technique

3.1 P-automaton

P-automaton is an automaton which exactly accepts the reachable configurations of some PDS. Distinguished by the forward and backward of transitions, P-automata are classified into $Post^*$-automata and Pre^*-automata.

Nested Timed Automata

Guoqiang Li[1], Xiaojuan Cai[1], Mizuhito Ogawa[2], and Shoji Yuen[3]

[1] School of Software, Shanghai Jiao Tong University, China
{li.g, cxj}@sjtu.edu.cn
[2] Japan Advanced Institute of Science and Technology, Japan
mizuhito@jaist.ac.jp
[3] Graduate School of Information Science, Nagoya University, Japan
yuen@is.nagoya-u.ac.jp

Abstract. This paper proposes a new timed model named *nested timed automata (NeTAs)*. A NeTA is a pushdown system whose stack symbols are *timed automata (TAs)*. It either behaves as the top TA in the stack, or switches from one TA to another by pushing, popping, or changing the top TA of the stack. Different from existing component-based context-switch models such as *recursive timed automata* and *timed recursive state machines*, when time passage happens, all clocks of TAs in the stack elapse uniformly. We show that the safety property of NeTAs is decidable by encoding NeTAs to the *dense timed pushdown automata*. NeTAs provide a natural way to analyze the recursive behaviors of component-based timed systems with structure retained. We illustrate this advantage by the deadline analysis of nested interrupts.

1 Introduction

Due to the rapid development of large and complex timed systems, requirements to model and analyze complex real-time frameworks with recursive context switches have been stresses. Difficulty comes from two dimensions of infinity, a stack with unbounded number of symbols, and clocks recording dense time.

Timed automata (TAs) [1] are a finite automaton with a finite set of *clocks* that grow uniformly. A typical timed model with context switches is *timed pushdown automata (TPDAs)* [2], equipped with an unbounded stack, where clocks are not updated in the stack. This limitation is found unnatural in analyzing the timed behavior of programs since clock values should be updated in suspension. Recently, a new timed pushdown model, *dense timed pushdown automata (DTPDAs)* [3] has been proposed, where each symbol in the stack is equipped with local clocks named "ages", and all ages in the stack are updated uniformly for time passage. Reachability problem of DTPDAs is in EXPTIME [3].

This paper proposes a new timed model named *nested timed automata (NeTAs)*. A NeTA is a pushdown system whose stack symbols are TAs. It either behaves as the top TA in the stack, or switches from one TA to another following three kinds of transitions: pushing a new TA, popping the current TA when terminates, or replacing the top TA of the stack. This hierarchical design captures the dynamic feature of functionally independent component-based structure of timed systems. The existing models, such as *recursive timed automata* [4], and

timed recursive state machines [5] do not update clocks in the stack when time passage happens, while in NeTAs, all clocks elapse uniformly. When a TA is pushed into the stack, a set of fresh local clocks is introduced to the system. In this respect, NeTAs may have to handle an unbounded number of local clocks. NeTAs are shown to be encoded to DTPDAs preserving the state reachability. All transitions of NeTAs are simulated by DTPDAs, and vice versa. We illustrate that NeTAs are adopted to analyze the timely deadline of nested interrupts.

The rest of the paper is organized as follows. Section 2 gives an introduction of TAs and DTPDAs. Section 3 gives the formal definition and semantics of NeTAs. Section 4 presents an encoding method from NeTAs to DTPDAs, and proves its correctness. Section 5 illustrates the usages of NeTAs by an application example. Section 6 gives the related work and Section 7 concludes the paper.

Due to the lack of space, we omit proofs of theorems, detailed explanations and nations, which can be found in its extended version [6].

2 Preliminaries

Let $\mathbb{R}^{\geq 0}$ and $\mathbb{N}$ denote the sets of non-negative real numbers and natural numbers, respectively. We define $\mathbb{N}^{\omega} := \mathbb{N} \cup \{\omega\}$, where ω is the first limit ordinal. Let $\mathcal{I}$ denote the set of *intervals*. An interval is a set of numbers, written as (a, b), $[a, b]$, $[a, b)$ or $(a, b]$, where $a \in \mathbb{N}$ and $b \in \mathbb{N}^{\omega}$. For a number $r \in \mathbb{R}^{\geq 0}$ and an interval $I \in \mathcal{I}$, we use $r \in I$ to denote that r belongs to I.

Let $X = \{x_1, \ldots, x_n\}$ be a finite set of *clocks*. A *clock valuation* $\nu : X \to \mathbb{R}^{\geq 0}$, assigns a value to each clock $x \in X$. $\nu_{\mathbf{0}}$ represents all clocks in X assigned to zero. Given a clock valuation ν and a time $t \in \mathbb{R}^{\geq 0}$, $(\nu + t)(x) = \nu(x) + t$, for $x \in X$. A clock assignment function $\nu[y \leftarrow b]$ is defined by $\nu[y \leftarrow b](x) = b$ if $x = y$, and $\nu(x)$ otherwise.

2.1 Timed Automata

A timed automaton is an automaton augmented with a finite set of clocks [1, 7]. Time can elapse in a location, while switches are instantaneous.

Since we focus on the *safety properties* (i.e., *emptiness* problem of a TA, or *reachability* problem of a timed transition system), we omit input symbols for all concerned automata, following the formalization in [3].

We adopt the TA definition style from that in [3], which looks different from the one in [1, 7]. The main difference is that we do not adopt *invariant*, a time constraint assigned to each control location. The reason lies that invariants cause time lock problems. When context switches back, it may occur that the system can neither stay in the current control location since the invariant is violated nor transit to other control location since all constraints on transitions are violated. Nondeterministic clock updates are also taken from [9] with interval tests as diagonal free time constraints where decidability results are not affected.

Definition 1 (Timed Automata). *A timed automaton is a tuple* $\mathcal{A} = (Q, q_0, F, X, \Delta) \in \mathscr{A}$, *where*

- Q *is a finite set of control locations, with the initial location* $q_0 \in Q$,

- $F \subseteq Q$ *is the set of final locations,*
- X *is a finite set of clocks,*
- $\Delta \subseteq Q \times \mathcal{O} \times Q$*, where* $\mathcal{O}$ *is a set of* operations*. A transition* $\delta \in \Delta$ *is a triplet* (q_1, ϕ, q_2)*, written as* $q_1 \xrightarrow{\phi} q_2$*, in which* ϕ *is either of*
  Local ϵ*, an* empty *operation,*
  Test $x \in I?$ *where* $x \in X$ *is a clock and* $I \in \mathcal{I}$ *is an interval, and*
  Assignment $x \leftarrow I$ *where* $x \in X$ *and* $I \in \mathcal{I}$*.*

Given a TA $\mathcal{A} \in \mathscr{A}$, we use $Q(\mathcal{A})$, $q_0(\mathcal{A})$, $F(\mathcal{A})$, $X(\mathcal{A})$ and $\Delta(\mathcal{A})$ to represent its set of control locations, initial location, set of final locations, set of clocks and set of transitions, respectively. We will use similar notations for other models.

Definition 2 (Semantics of TAs). *Given a TA* $\mathcal{A} = (Q, q_0, F, X, \Delta)$*, a configuration is a pair* (q, ν) *of a control location* $q \in Q$*, and a clock valuation* ν *on* X*. The transition relation of the TA is represented as follows,*

- Progress transition*:* $(q, \nu) \xrightarrow{t}_{\mathscr{A}} (q, \nu + t)$*, where* $t \in \mathbb{R}^{\geq 0}$*.*
- Discrete transition*:* $(q_1, \nu_1) \xrightarrow{\phi}_{\mathscr{A}} (q_2, \nu_2)$*, if* $q_1 \xrightarrow{\phi} q_2 \in \Delta$*, and one of the following holds,*
  - **Local** $\phi = \epsilon$*, then* $\nu_1 = \nu_2$*.*
  - **Test** $\phi = x \in I?$*,* $\nu_1 = \nu_2$ *and* $\nu_2(x) \in I$ *holds.*
  - **Assignment** $\phi = x \leftarrow I$*,* $\nu_2 = \nu_1[x \leftarrow r]$ *where* $r \in I$*.*

The initial configuration is $(q_0, \nu_\mathbf{0})$*. The transition relation is* $\rightarrow$ *and we define* $\rightarrow = \xrightarrow{t}_{\mathscr{A}} \cup \xrightarrow{\phi}_{\mathscr{A}}$*, and define* $\rightarrow^*$ *to be the reflexive and transitive closure of* $\rightarrow$*.*

Remark 1 (Sound Simulation). The TA definition in Definition 1 follows the style in [3]. In [1], a TA allows a logical connection of several constraint tests, e.g. $x \leq 15 \wedge y > 20$, and several resets operations of different clocks during one discrete transition. Only one test or assignment (a generalization of the reset) is allowed during one discrete transition in the definition. Since a discrete transition is followed by a progress transition where time elapses, the main ambiguity of two definitions is whether a conjunction of two tests can be checked one by one, between which the time elapses. It is shown that TA with our definition soundly simulates the timed traces in the original definition, as follows,

For $\geq$ or $>$, $c \xrightarrow{x \in [a, +\infty)?}_{\mathscr{A}} c' \xrightarrow{t}_{\mathscr{A}} c'$ is safely converted to $c \xrightarrow{t}_{\mathscr{A}} c \xrightarrow{[a, +\infty)?}_{\mathscr{A}} c'$, for some configurations c and c'.

For $\leq$ or $<$, $c \xrightarrow{t}_{\mathscr{A}} c \xrightarrow{x \in [0, a]?}_{\mathscr{A}} p'$ is safely converted to $c \xrightarrow{x \in [0, a]?}_{\mathscr{A}} c' \xrightarrow{t}_{\mathscr{A}} c'$, for some configurations c and c'.

For test transitions, a general simulation strategy is, firstly, checking the $\geq$, and $>$ one by one, then check $\leq$ and $<$ later. If there exists a "$=$" constraint, decomposed it into $\geq \wedge \leq$. For example, a transition $p \xrightarrow{x \leq 15 \wedge y > 20} q$ in the original definition is simulated by two transitions $p \xrightarrow{y \in (20, +\infty)?} p' \xrightarrow{x \in [0, 15]?} q$ under the new definition, where p' is a fresh control location.

For reset transitions, a group of clocks are reset simultaneously can be simulated by resetting clocks one by one, with a zero test of the first reset clock on the tail. For example, a transition $p \xrightarrow{\{x, y\}} q$, resetting x and y simultaneously, in

the original definition is simulated by $p \xrightarrow{x\leftarrow[0,0]} p' \xrightarrow{y\leftarrow[0,0]} p'' \xrightarrow{x\in[0,0]?} q$, where p', p'' are fresh control locations.

If a transition contains both test and reset operations, we firstly simulate test operation, then reset operation, following the above rules.

2.2 Dense Timed Pushdown Automata

Dense Timed Pushdown Automata (DTPDAs) [3] extend TPDAs with time update in the stack. Each symbol in the stack is equipped with a local clock named *age*, and all ages in the stack elapse uniformly.

Definition 3 (Dense Timed Pushdown Automata). *A dense timed pushdown automaton is a tuple $\mathcal{D} = \langle S, s_0, \Gamma, C, \Delta\rangle \in \mathscr{D}$, where*

- *S is a finite set of states with the initial state $s_0 \in S$,*
- *Γ is a finite stack alphabet,*
- *C is a finite set of clocks, and*
- *$\Delta \subseteq S \times \mathcal{O} \times S$ is a finite set of transitions.*

A transition $\delta \in \Delta$ is a triplet (s_1, ϕ, s_2), written as $s_1 \xrightarrow{\phi} s_2$, in which ϕ is either of

- **Local** *ϵ, an empty operation,*
- **Test** *$x \in I?$, where $x \in X$ is a clock and $I \in \mathcal{I}$ is an interval,*
- **Assignment** *$x \leftarrow I$ where $x \in C$ and $I \in \mathcal{I}$,*
- **Push** *$push(\gamma, I)$, where $\gamma \in \Gamma$ is a stack symbol and $I \in \mathcal{I}$. It pushes γ to the top of the stack, with the age in the interval I.*
- **Pop** *$pop(\gamma, I)$, where $\gamma \in \Gamma$ and $I \in \mathcal{I}$. It pops the top-most stack symbol provided that this symbol is γ, and its age belongs to I.*
- **Push**$_A$ *$push(\gamma, x)$, where $\gamma \in \Gamma$ is a stack symbol and $x \in C$, and*
- **Pop**$_A$ *$pop(\gamma, x)$, where $\gamma \in \Gamma$ is a stack symbol and $x \in C$.*

Definition 4 (Semantics of DTPDAs). *For a DTPDA $\langle S, s_0, \Gamma, C, \Delta\rangle$, a configuration is a triplet (s, w, ν) with $s \in S$, $w \in (\Gamma \times \mathbb{R}^{\geq 0})^*$, and a clock valuation ν on X. Time passage of the stack $w + t = (\gamma_1, t_1 + t).\cdots.(\gamma_n, t_n + t)$ for $w = (\gamma_1, t_1).\cdots.(\gamma_n, t_n)$.*

The transition relation of the DTPDA is defined as follows:

- Progress transition*: $(s, w, \nu) \xrightarrow{t}_{\mathscr{D}} (s, w + t, \nu + t)$, where $t \in \mathbb{R}^{\geq 0}$.*
- Discrete transition*: $(s_1, w_1, \nu_1) \xrightarrow{\phi}_{\mathscr{D}} (s_2, w_2, \nu_2)$, if $s_1 \xrightarrow{\phi} s_2$, and one of the following holds,*
  - **Local** *$\phi = \epsilon$, then $w_1 = w_2$, and $\nu_1 = \nu_2$.*
  - **Test** *$\phi = x \in I?$, then $w_1 = w_2$, $\nu_1 = \nu_2$ and $\nu_2(x) \in I$ holds.*
  - **Assignment** *$\phi = x \leftarrow I$, then $w_1 = w_2$, $\nu_2 = \nu_1[x \leftarrow r]$ where $r \in I$.*
  - **Push** *$\phi = push(\gamma, I)$, then $\nu_1 = \nu_2$, and $w_2 = (\gamma, r).w_1$ for some $r \in I$.*
  - **Pop** *$\phi = pop(\gamma, I)$, then $\nu_1 = \nu_2$, and $w_1 = (\gamma, r).w_2$ for some $r \in I$.*
  - **Push**$_A$ *$\phi = push(\gamma, x)$, then $\nu_1 = \nu_2$, and $w_2 = (\gamma, \nu_1(x))w_1$.*
  - **Pop**$_A$ *$\phi = pop(\gamma, x)$, then $\nu_2 = \nu_1[x \leftarrow t]$, and $w_1 = (\gamma, t)w_2$.*

The initial configuration $\kappa_0 = (q_0, \epsilon, \nu_\mathbf{0})$. We use $\hookrightarrow$ to range over these transitions, and $\hookrightarrow^*$ is the transitive closure of $\hookrightarrow$, conventionally.

Example 1. Fig. 1 shows transitions between configurations of a DTPDA consisting of a singleton state set $S = \{\bullet\}$ (omitted in the figure), clocks $C = \{x_1, x_2, x_3\}$, and stack symbols $\Gamma = \{a, b, d\}$. From κ_1 to κ_2, a discrete transition $push(d, x_3)$ pushes $(d, 2.3)$ into the stack. A time transition from κ_2 to κ_3 elapses 2.6 time units, and each value grows older for 2.6. From κ_3 to κ_4, the value of x_2 is reset to 3.8, which lies in the interval $(2, 5]$, and the last transition pops (d, x_1) and resets x_1 to 4.9.

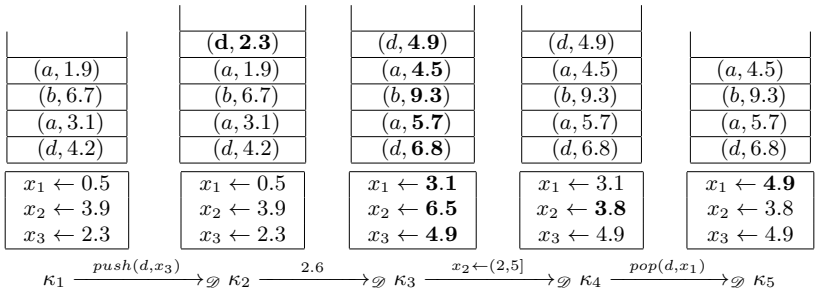


Fig. 1. An Example of DTPDA

Remark 2. Definition 3 extends the definition in [3] by adding $\mathbf{Push}_A$ and $\mathbf{Pop}_A$, which stores and recovers from clocks to ages and vice versa. This extension does not destroy decidability of state reachability of DTPDAs [8], since its symbolic encoding is easily modified to accept $\mathbf{Push}_A$ and $\mathbf{Pop}_A$. For instance, $\mathbf{Push}_A$ is encoded similar to **Push**, except for the definition on *Reset* [3]. $Reset(R)[a \leftarrow I]$ symbolically explores all possibility of the fraction of an instance in I. Instead, $Reset(R)[a \leftarrow x]$ will assign the same integer and fraction parts to x, which means an age is simply placed at the same position to x in the region.

3 Nested Timed Automata

Nested Timed Automata (NeTAs) aim to give an operation strategy to a group of TAs, in which a TA is able to preempt the other ones. All clocks in a NeTA are local clocks, with the scope of their respective TAs. These clocks in the stack elapse simultaneously. An unbounded number of clocks may be involved in one NeTA, due to recursive preemption loops.

Definition 5 (Nested Timed automata). *A nested timed automaton is a triplet $\mathcal{N} = (T, \mathcal{A}_0, \Delta) \in \mathscr{N}$, where*

- *T is a finite set of timed automata, with the initial timed automaton $\mathcal{A}_0 \in T$.*

– $\Delta \subseteq T \times \mathcal{P} \times (T \cup \{\varepsilon\})$, *where* $\mathcal{P} = \{push, pop, internal\}$. *A rule* $(\mathcal{A}_i, \Phi, \mathcal{A}_j) \in \Delta$ *is written as* $\mathcal{A}_i \xrightarrow{\Phi} \mathcal{A}_j$, *where*
Push $\mathcal{A}_i \xrightarrow{push} \mathcal{A}_j$,
Pop $\mathcal{A}_i \xrightarrow{pop} \varepsilon$, *and*
Internal $\mathcal{A}_i \xrightarrow{internal} \mathcal{A}_j$.

The initial state of NeTAs is the initial location in $\mathcal{A}_0$, s.t. $q_0(\mathcal{A}_0)$. We also assume that $X(\mathcal{A}_i) \cap X(\mathcal{A}_j) = \emptyset$, and $Q(\mathcal{A}_i) \cap Q(\mathcal{A}_j) = \emptyset$ for $\mathcal{A}_i, \mathcal{A}_j \in T$ and $i \neq j$.

The operational semantics of NeTAs is informally summarized as follows. It starts with a stack with the only symbol of the initial TA. The system has the following four behaviors: when there exists time passage, all clocks in the stack elapse simultaneously; it is able to behave like the top TA in the stack; when there exists a push transition from the top TA of the stack to the other TA, a new instance of the latter TA is pushed into the stack at any time and executed, while the suspended location of the former TA is recorded in the stack; when the top TA in the stack reaches the final location and a pop transition happens, it will be popped from the stack, and the system will run the next TA beginning with the suspended location; if an internal transition from the top TA to the other TA occurs, the top TA in the stack will be changed to a new instance of the latter TA when it reaches some final location.

Definition 6 (Semantics of NeTAs). *Given a NeTA* $(T, \mathcal{A}_0, \Delta)$, *a configuration is a stack, and the stack alphabet is a tuple* $\langle \mathcal{A}, q, \nu \rangle$, *where* $\mathcal{A} \in T$ *is a timed automaton,* q *is the current running control location where* $q \in Q(\mathcal{A})$, *and* ν *is the clock valuation of* $X(\mathcal{A})$. *For a stack content* $c = \langle \mathcal{A}_1, q_1, \nu_1 \rangle \langle \mathcal{A}_2, q_2, \nu_2 \rangle \ldots \langle \mathcal{A}_n, q_n, \nu_n \rangle$, *let* $c + t$ *be* $\langle \mathcal{A}_1, q_1, \nu_1 + t \rangle \langle \mathcal{A}_2, q_2, \nu_2 + t \rangle \ldots \langle \mathcal{A}_n, q_n, \nu_n + t \rangle$.
The transition of NeTAs is represented as follows:

– *Progress transitions:* $c \xrightarrow{t}_{\mathcal{N}} c + t$.
– *Discrete transitions:* $c \xrightarrow{\phi}_{\mathcal{N}} c'$ *is defined as a union of the following four kinds of transition relations,*
  - **Intra-action** $\langle \mathcal{A}, q, \nu \rangle c \xrightarrow{\phi}_{\mathcal{N}} \langle \mathcal{A}, q', \nu' \rangle c$, *if* $q \xrightarrow{\phi} q' \in \Delta(\mathcal{A})$, *and one of the following holds,*
    * **Local** $\phi = \epsilon$, *then* $\nu = \nu'$.
    * **Test** $\phi = x \in I?$, $\nu = \nu'$ *and* $\nu'(x) \in I$ *holds.*
    * **Assignment** $\phi = x \leftarrow I$, $\nu' = \nu[x \leftarrow r]$ *where* $r \in I$.
  - **Push** $\langle \mathcal{A}, q, \nu \rangle c \xrightarrow{push}_{\mathcal{N}} \langle \mathcal{A}', q_0(\mathcal{A}'), \nu'_{\mathbf{0}} \rangle \langle \mathcal{A}, q, \nu \rangle c$, *if* $\mathcal{A} \xrightarrow{push} \mathcal{A}'$, *and* $q \in Q(\mathcal{A})$.
  - **Pop** $\langle \mathcal{A}, q, \nu \rangle c \xrightarrow{pop}_{\mathcal{N}} c$, *if* $\mathcal{A} \xrightarrow{pop} \varepsilon$, *and* $q \in F(\mathcal{A})$.
  - **Inter-action** $\langle \mathcal{A}, q, \nu \rangle c \xrightarrow{internal}_{\mathcal{N}} \langle \mathcal{A}', q_0(\mathcal{A}'), \nu'_{\mathbf{0}} \rangle c$, *if* $\mathcal{A} \xrightarrow{internal} \mathcal{A}'$, *and* $q \in F(\mathcal{A})$.

The initial configuration $c_0 = \langle \mathcal{A}_0, q_0(\mathcal{A}_0), \nu_{\mathbf{0}} \rangle$. *We use* $\longrightarrow$ *to range over these transitions and* $\longrightarrow^*$ *is the transitive closure of* $\longrightarrow$, *conventionally.*

In followings, we focus on the state reachability that is regarded as the most important property in modelling software behavior.

Definition 7 (Safety Property). *A safety property of NeTAs is defined as the state reachability problem: Given a NeTA $\mathcal{N} = (T, \mathcal{A}_0, \Delta)$, and a control location $p_f \in Q(\mathcal{A})$ for some $\mathcal{A} \in T$, decide whether there exists a configuration c of $\mathcal{N}$ and a clock valuation ν, such that $c_0 \longrightarrow^* \langle \mathcal{A}, p_f, \nu \rangle c$.*

Example 2. We take a simple example to show the usage of NeTAs. Assume that two processes access a shared buffer. One is to read from the buffer periodically each 4 time units. It accomplishes after it reads one or more data. The other is to write to the buffer periodically. The execution time is between 3 and 5 time units. It will return after writes one or more data. The writing process may overtake the reading process which initially starts running. The NeTA is shown in Fig. 2, with three TAs. $\mathcal{A}_0$ is an empty TA for the idle state. $\mathcal{A}_1$ and $\mathcal{A}_2$ are for reading and writing processes, respectively. We have three transition rules: $\mathcal{A}_0 \xrightarrow{internal} \mathcal{A}_1$, $\mathcal{A}_1 \xrightarrow{push} \mathcal{A}_2$, and $\mathcal{A}_2 \xrightarrow{pop} \varepsilon$. The pop transition is not explicitly represented in the figure. We use dash-line frames to represent the border of TAs in the NeTA, double-line arrows to indicate the initial location/TA, and double-line circles to represent the final locations of TAs.

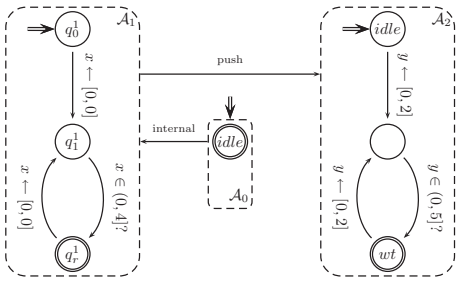


Fig. 2. An Example of NeTA

Remark 3 (Composition with timed automata). A NeTA is composed with a TA by synchronization with shared actions in Σ, where we are allowed to add input symbols as actions on transitions of NeTA. A composed TA presents behavioral properties independent of recursive context switches such as the environment. Although this extension does not increase the expressiveness of NeTAs, it is very useful to model and analyze the behavioral properties in the component-based manner [10, 11]. A formal definition of the parallel composition, between a NeTA $\mathcal{N}$ and a TA $\mathcal{A}$, written as $\mathcal{N} \| \mathcal{A}$, is formally defined in [6].

4 Decidability of Safety Property

In this section we prove the *safety property* problem of NeTAs is decidable by encoding it into DTPDAs, of which state reachability is decidable.

4.1 Encoding NeTA to DTPDA

The idea to encode a NeTA to a DTPDA is straightforward, dealing with multiple clocks at push and pop operations. We adopt extra fresh locations and transitions to check whether a group of push/pop actions happens instantly.

Given a NeTA $\mathcal{N} = (T, \mathcal{A}_0, \Delta)$, we define $\mathcal{E}(\mathcal{N}) = \langle S, s_0, \Gamma, C, \nabla \rangle$ as the target of DTPDA encoding of $\mathcal{N}$. Each element is described as,

The set of states $S = S_{\mathcal{N}} \cup S_{inter}$, where $S_{\mathcal{N}} = \bigcup_{\mathcal{A}_i \in T} Q(\mathcal{A}_i)$ is the set of all locations of TAs in $T(\mathcal{N})$. S_{inter} is the set of *intermediate* states during the simulation of push, pop, and internal rules of NeTAs. We assume that $S_{\mathcal{N}}$ and S_{inter} are disjoint.

Let $n = |T|$ and $m_i = |X(\mathcal{A}_i)|$ for each $\mathcal{A}_i \in T$. S_{inter} is

$$S_{inter} = (\bigcup_{\mathcal{A}_i \in T} S^i_{reset}) \cup (\bigcup_{\mathcal{A}_i \xrightarrow{push} \mathcal{A}_j \in \Delta} S^{i,j}_{push}) \cup \{o\}$$

- For every $\mathcal{A}_i \in T$, we define $S^i_{reset} = (\bigcup_{k \in \{1 \cdots m_i+1\}} r^i_k) \cup t^i$. $r^i_k \in S^i_{reset}$ is the start state of a transition to initialize the k-th clock of $\mathcal{A}_i$ to 0. t^i is the start state of a testing transition to make sure that no time is elapsed during the sequence of initializing transitions.
- For every push rule $\mathcal{A}_i \xrightarrow{push} \mathcal{A}_j$, we define $S^{i,j}_{push} = \bigcup_{k \in \{1 \cdots m_i+1\}} p^{i,j}_k$. $p^{i,j}_k$ is the start state of a push transition that push the pair of the k-th clock of $\mathcal{A}_i$ and its value. After all clock values are stored in the stack, the last destination is the initial state $q_0(\mathcal{A}_j)$ of $\mathcal{A}_j$.
- o is a special state for repeat popping.

The initial state $s_0 = q_0(\mathcal{A}_0)$ is the initial location of the initial TA of $\mathcal{N}$.

The set of clocks $C = \{d\} \cup \bigcup_{\mathcal{A} \in T} X(\mathcal{A})$ consists of all clocks of TA in $T(\mathcal{N})$ and the special dummy clock d only to fulfill the field of push and pop rules, like $push(q, d)$ and $pop(q, d)$. (The value of d does not matter.)

The stack alphabet $\Gamma = C \cup S_{\mathcal{N}}$.

The set of transitions ∇ is the union of $\bigcup_{\mathcal{A}_i \in T} \Delta(\mathcal{A}_i)$ (as **Local** transitions of $\mathcal{E}(\mathcal{N})$) and the set of transitions described in Fig. 3. For indexes, we assume $0 \le i, j \le n-1$ and $1 \le k \le m_i$ (where i is specified in a context).

Local	$p^{i,j}_{m_i+1} \xrightarrow{\epsilon} r^j_1,\ r^i_{m_i+1} \xrightarrow{\epsilon} t^i,\ q_i \xrightarrow{\epsilon} r^j_1,\ q_i \xrightarrow{\epsilon} o \quad$ for $q_i \in F(\mathcal{A}_i)$.
Test	$t^i \xrightarrow{x^i_1 \in [0,0]?} q_0(\mathcal{A}_i)$.
Assignment	$r^i_k \xrightarrow{x^i_k \leftarrow [0,0]} r^i_{k+1}$.
Push	$q_i \xrightarrow{push(q_i,d)} p^{i,j}_1,\ p^{i,j}_k \xrightarrow{push(x^i_k, x^i_k)} p^{i,j}_{k+1}\ if\ k \le m_i, \quad$ for $q_i \in Q(\mathcal{A}_i)$.
Pop	$o \xrightarrow{pop(x,x)} o,\ o \xrightarrow{pop(q,d)} q \quad$ forall $x \in X(\mathcal{A}_i).\ q \in Q(\mathcal{A}_i)$.

Fig. 3. Transition Rules ∇ of $\mathcal{E}(\mathscr{C})$

Definition 8. *For a NeTA $\mathcal{N} = (T, \mathcal{A}_0, \Delta)$, its encoding into a DTPDA $\mathcal{E}(\mathcal{N})$ is as follows.*

$\mathcal{A}_i \xrightarrow{push} \mathcal{A}_j$	$q_i \xrightarrow{push(q_i,d)} p_1^{i,j} \xrightarrow{push(x_1^i,x_1^i)} \cdots p_{m_i}^{i,j} \xrightarrow{push(x_{m_i}^i,x_{m_i}^i)} p_{m_i+1}^{i,j} \xrightarrow{\epsilon} r_1^j \xrightarrow{x_1^j \leftarrow [0,0]} r_2^j \cdots r_{m_j+1}^j \xrightarrow{\epsilon} t^j \xrightarrow{x_1^j \in [0,0]?} q_0(\mathcal{A}_j)$
$\mathcal{A}_i \xrightarrow{pop} \epsilon$	$q_i \xrightarrow{\epsilon} o \xrightarrow{pop(x_{m_i+1}^i,x_{m_i+1}^i)} \cdots \xrightarrow{pop(x_1^i,x_1^i)} o \xrightarrow{pop(q,d)} q$
$\mathcal{A}_i \xrightarrow{internal} \mathcal{A}_j$	$q_i \xrightarrow{\epsilon} r_1^j \xrightarrow{x_1^j \leftarrow [0,0]} r_2^j \cdots r_{m_j+1}^j \xrightarrow{\epsilon} t^j \xrightarrow{x_1^j \in [0,0]?} q_0(\mathcal{A}_j)$

For a *push* transition $\mathcal{A}_i \xrightarrow{push} \mathcal{A}_j$, $\mathcal{E}(\mathcal{N})$ simulates, by storing current state of $\mathcal{A}_i$ into the stack, pushing each clock with its current value as an age, and switching to the initial configuration of $\mathcal{A}_j$ (which consists of initializing each clock $x \in X(\mathcal{A}_j)$, testing that no timed transitions interleaved, and move to the initial state $q_0(\mathcal{A}_j)$).

For a *pop* transition $\mathcal{A}_i \xrightarrow{pop} \epsilon$, $\mathcal{A}_i$ has finished its run at a final state and restores the previous context. $\mathcal{E}(\mathcal{N})$ simulates, first popping and setting each clock (of $\mathcal{E}(\mathcal{N})$), and set a state to q being stored in the stack.

Note that clocks of $\mathcal{E}(\mathcal{N})$ are used for currently running TA (at the top of the stack), and ages are used to store values of clocks of suspended TAs.

Example 3. A NeTA is shown in Fig. 4, which includes two TAs $\mathcal{A}_1$ and $\mathcal{A}_2$. $p_1, p_2 \in Q(\mathcal{A}_1)$, $x_1, x_2 \in X(\mathcal{A}_1)$, $q_1, q_2 \in Q(\mathcal{A}_2)$, and $y_1, y_2 \in X(\mathcal{A}_2)$, respectively. A push transition from $\mathcal{A}_1$ to $\mathcal{A}_2$ occurs at the location p_2, and the value of x_1 and x_2 are 2.9 and 3.3, respectively. After pushing, y_1 and y_2 are reset to zero, and the system begins with q_1. The encoding DTPDA is shown in Fig. 5. p_2 is firstly pushed into the stack, and afterwards, x_1 and x_2 in $\mathcal{A}_1$ is pushed into the stack one by one, with the initial value of the age as their respective value. Then after y_1 and y_2 in $\mathcal{A}_2$ are reset to 0 through the states r_1^2, r_2^2, and r_3^2, the system moves to q_1 provided the value of y_1 is kept as 0.

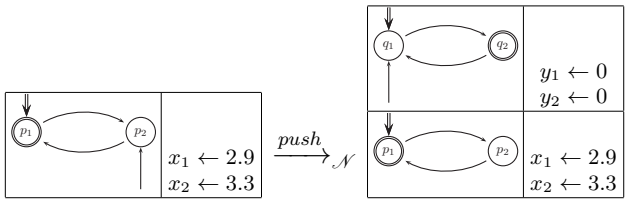


Fig. 4. A Push Transition on a Nested Timed Automaton

Example 4. The NeTA in Fig. 2 is encoded into a DTPDA in Fig. 6.

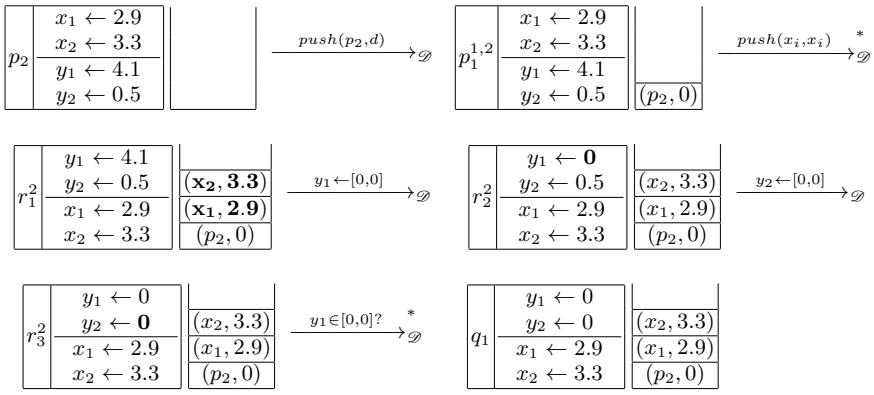


Fig. 5. Encoding the Push Transition in DTPDA

- The larger circles are the original states from the NeTA, while the smaller ones are intermediate states.
- Since $\mathcal{A}_0 \xrightarrow{internal} \mathcal{A}_1$, before q_0^0 connects to q_0^1, all clocks in $\mathcal{A}_1$ are reset to zero and kept uniformly. q_0^0 firstly is connected to r_1^1. r_1^1 and r_2^1 reset clocks and t^1 tests the uniformity of clocks.
- Since $\mathcal{A}_1 \xrightarrow{push} \mathcal{A}_2$, each state in $\mathcal{A}_1$ connects to $p_1^{1,2}$ by a transition to push itself. $p_1^{1,2}$ and $p_2^{1,2}$ push each clock in $\mathcal{A}_1$. Before connecting to $q_0^2 \in \mathcal{A}_2$, all clocks in $\mathcal{A}_2$ are similarly reset and tested, through r_1^2, r_2^2 and t^2.
- Since $\mathcal{A}_2 \xrightarrow{pop} \varepsilon$, after some final state of $\mathcal{A}_2$ is reached, each clock in the stack should be popped, through an extra state o. After that, o will connect each state in $\mathcal{A}_1$, by which the respective suspended state is popped.

4.2 Correctness of the Encoding

To reduce state reachability problem of NeTAs to that of DTPDAs, we show that transitions are preserved and reflected by the encoding.

Definition 9 (Encoded Configuration). *For a NeTA $\mathcal{N} = (T, \mathcal{A}_0, \Delta)$, its DTPDA encoding $\mathcal{E}(\mathcal{N}) = \langle S, s_0, \Gamma, C, \nabla \rangle$. and a NeTA configuration*

$$c = \langle \mathcal{A}_1, q_1, \nu_1 \rangle \langle \mathcal{A}_2, q_2, \nu_2 \rangle \dots \langle \mathcal{A}_n, q_n, \nu_n \rangle$$

let $c_{hd} = \langle \mathcal{A}_1, q_1, \nu_1 \rangle$ and $c_{tl} = \langle \mathcal{A}_n, q_n, \nu_n \rangle$. A clock valuation of c, $\overline{\nu}(c) : C \to \mathbb{R}^{\geq 0}$ is defined as $\overline{\nu}(c)(x) = \nu_1(x)$ if $x \in X(\mathcal{A}_1)$, and any*, otherwise.* [1] *Let $\overline{w}(c) = w_1 \cdots w_n$, where $w_i = (x^i_{m_i}, \nu_i(x^i_{m_i})) \cdots (x^i_1, \nu_i(x^i_1))(q_i, 0)$.*

[1] any means any value, since except for a clock in the top stack frame of a nested timed automaton, its value does not matter.

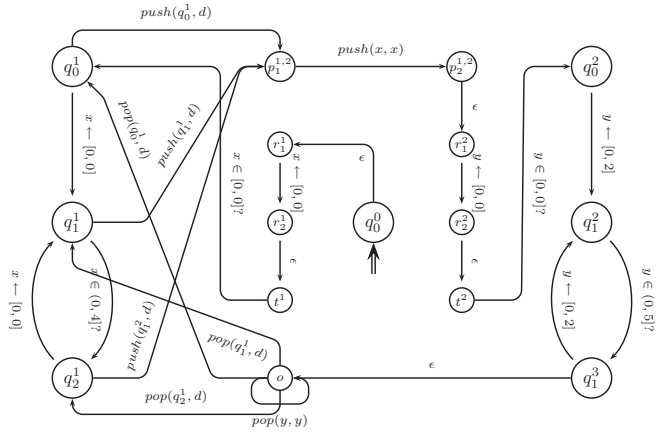


Fig. 6. Encoding the NeTA to DTPDA

We denote a configuration $(q_1, \overline{w}(c_{tl}), \overline{\nu}(c))$ *of* $\mathcal{E}(\mathcal{N})$ *by* $[\![c]\!]$. *A configuration* κ *of DTPDA with some* c *and* $\kappa = [\![c]\!]$ *is called an* encoded configuration.

Lemma 1. *Given a NeTA* $\mathcal{N}$, *its encoding* $\mathcal{E}(\mathcal{N})$, *and configurations* c, c' *of* $\mathcal{N}$.

- **(Preservation)** *if* $c \longrightarrow c'$, *then* $[\![c]\!] \hookrightarrow^* [\![c']\!]$;
- **(Reflection)** *if* $[\![c]\!] \hookrightarrow^* \kappa$,
  1. *there exists* c' *such that* $\kappa = [\![c']\!]$ *and* $c \longrightarrow^* c'$, *or*
  2. κ *is not an encoded configuration, and there exists* c' *such that* $\kappa \hookrightarrow^* [\![c']\!]$ *by discrete transitions (of* $\mathcal{E}(\mathcal{N})$*) and* $c \longrightarrow^* c'$.

From Lemma 1, we have the decidability of the safety property of NeTAs.

Theorem 1. *The state reachability problem of NeTAs is decidable.*

Remark 4 (Global clocks). We can assign a disjoint finite set of global clocks to a NeTA. These global clocks are tested and reassigned during push, pop and internal transitions, to control time conditions for push, pop and internal actions. Global clocks do not affect the safety property of a NeTA, since during the encoding to DTPDA, we just include these clocks to the set of clocks in DTPDA, keeping the copies of global clocks for all stack elements.

Fact 1 *A parallel composition of a NeTA and a TA can be encoded into a NeTA with global clocks by forgetting the synchronizing actions.*

Remark 5 (Encode DTPDAs into NeTAs). We can also encode a DTPDA into a NeTA with global clocks by regarding each state of the DTPDA as a TA with only one (local) clock. These TAs and their respective clocks can thus be used to represent pairs of stack symbols and ages.

5 Deadline Analysis for Nested Interrupts

Timely interrupt handling is part of correctness for real-timed, interrupt-driven system. It is vital for a *deadline analysis* [12, 13] in such systems to check that all specified deadlines for interrupt handling will be met. Such analysis is a daunting task because of large number of different possible interrupt arrival scenarios. An *interrupt signal* from outside transfers the control to an *interrupt handler* deferring the interrupted execution. When there are more than one interrupts, an *interrupt controller* provides priorities for them according to urgency of each interrupt. An interrupt handler is suspended by another handler with higher priority. After the high priority handler is done, the previous handler is resumed. In the followings NeTA combined with TA is shown to be useful for deadline analysis with such nested interrupt handling.

The time and frequency of interrupt signals can be represented by a TA, with input actions as events that trigger interrupt handlers. For instance, Fig. 7 gives an example of a TA that trigger three interrupt handlers, by $coming_P$, $coming_Q$, and $coming_R$, respectively.

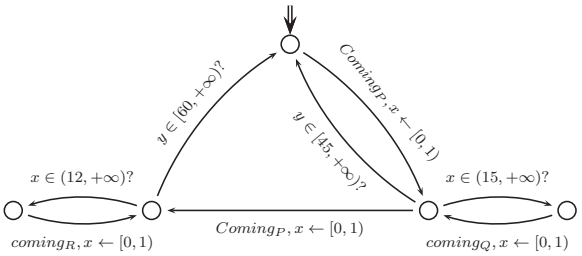


Fig. 7. A Timed Automata as an Environment

Assume a finite set of interrupt handler specifications $\mathscr{H}$. Each handler is specified by $P(\mathcal{A}, D)$, where $\mathcal{A}$ is a TA to describe its behavior, and D is its *relative deadline*. A system should guarantee that each executed handler p of $P(\mathcal{A}, D)$ is executed as $\mathcal{A}$ and reached to some final location of $\mathcal{A}$ within D time units. If the handler misses the deadline, it raises an error.

An interrupt handler with relative deadline D is transformed from $\mathcal{A}$ to another TA with error location. $\texttt{Guarded} : \mathscr{H} \to \mathscr{A}$ is defined by $\texttt{Guarded}(P(\mathcal{A}, D)) = (Q^G, q_0^G, F^G, X^G, \Delta^G)$. Each element is shown as follows,

- $Q^G = Q(\mathcal{A}) \cup Q_\Delta \cup \{q_{err}\}$, where $Q_\Delta = \{q_\delta \mid \text{for each } \delta \in \Delta(\mathcal{A})\}$.
- $q_0^G = q_0(\mathcal{A})$, and $F^G = F(\mathcal{A})$.
- $X^G = X(\mathcal{A}) \cup \{x_{sch}\}$.
- $\Delta^G = \Delta_{sch} \cup \Delta_{err}$, where
  - $\Delta_{sch} = \{q \xrightarrow{o} q_\delta, q_\delta \xrightarrow{x_{sch} \in [0,D]?} q' \mid \delta = (q, o, q') \in \Delta(\mathcal{A})\}$.
  - $\Delta_{err} = \{q \xrightarrow{x_{sch} \in (D, +\infty)?} q_{err} \mid q \in Q(\mathcal{A}) \cup Q_\Delta\}$.

Given a finite set of handler specifications $\mathscr{H}$, a priority policy is described by a relation $\prec$ on $\mathscr{H}$. For instance, the most common policy is *fixed priority strategy (FPS)*, where $\prec$ is a strict partial order (irreflexive, asymmetric and transitive). An interrupt controller $\texttt{Sch}(\mathscr{H}, \prec)$ with $\prec$ as a FPS is defined by a nested timed automaton $(T, \mathcal{A}_0, \Delta)$ over a set of input symbols Σ where,

- $\Sigma = \{\texttt{Coming}_P \mid \text{for each } P \in \mathscr{H}\}$.
- $T = \{\texttt{Guarded}(P) \mid \text{for each } P \in \mathscr{H}\} \cup \{\mathcal{A}_{idle}\}$, where $\mathcal{A}_{idle}$ is a singleton timed automaton without any transitions.
- $\mathcal{A}_0 = \mathcal{A}_{idle}$.
- Δ is defined by $\Delta_{idle} \cup \Delta_{push} \cup \Delta_{pop} \cup \Delta_{internal}$, where
  - $\Delta_{idle} = \{\mathcal{A}_{idle} \xrightarrow{\texttt{Coming}_P, push} \mathcal{A} \mid \forall P \in \mathscr{H}, \mathcal{A} = \texttt{guarded}(P)\}$.
  - $\Delta_{push} = \{\mathcal{A} \xrightarrow{\texttt{Coming}_{P'}, push} \mathcal{A}' \mid \forall P, P' \in \mathscr{H}, P \prec P' \wedge \mathcal{A} = \texttt{guarded}(P) \wedge \mathcal{A}' = \texttt{guarded}(P')\}$.
  - $\Delta_{pop} = \{\mathcal{A} \xrightarrow{pop} \varepsilon \mid \forall P \in \mathscr{H}, \mathcal{A} = \texttt{guarded}(P)\}$.
  - $\Delta_{internal} = \{\mathcal{A} \xrightarrow{\texttt{Coming}_{P'}, internal} \mathcal{A}' \mid \forall P, P' \in \mathscr{H}, P \not\prec P' \wedge P \not\succ P' \wedge \mathcal{A} = \texttt{guarded}(P) \wedge \mathcal{A}' = \texttt{guarded}(P')\}$.

After performing parallel composition with a TA as an environment, we are allowed to check the deadline of each interrupt handler P_i through the reachability problem on the error location in $\texttt{Guarded}(P_i)$, considering the fact that a finite number of interrupt handlers are effectively invoked.

6 Related Work

After TAs [1] had been proposed, lots of researches were intended timed context switches. TPDAs were firstly proposed in [2], which enjoys decidability of reachability problem. Dang proved in [14] the decidability of binary reachability of TPDAs. All clocks in TPDAs were treated globally, which were not effected when the context switches.

Our model relied heavily on a recent significant result, named *dense timed pushdown automata (DTPDAs)* [3]. The difference between DTPDAs and NeTAs was the hierarchical feature. In NeTAs, a finite set of local clocks were pushed into the stack at the same time. When a pop action happens, the values of clocks belonging to popped TA were popped simultaneously and reused. This feature eased much for modelling the behavior of time-aware software. In DTPDAs, local clocks must be dealt within some proper bookkeeping process, which was not essential part of the analysis. In [15], a discrete version of DTPDAs, named *discrete timed pushdown automata* was introduced, where time was incremented in discrete steps and thus the ages of clocks and stack symbols are in the natural numbers. This made the reachability problem much simpler, and easier for efficient implementation.

Based on *recursive state machines* [16], two similar timed extensions, *timed recursive state machines (TRSMs)* [5] and *recursive timed automata (RTAs)* [4], were given independently. The main differences from NeTAs were, the two models had no stack time-update during progress transitions, and the number of clocks was essentially finite. The *hierarchical timed automata (HTAs)* [17] kept the

similar structure of clocks, where only a bounded number of levels were treated, while NeTAs treated an unbounded number of levels.

The class of *extended timed pushdown automata (ETPDAs)* was proposed in [5]. An ETPDA was a pushdown automaton enriched with a set of clocks, with an additional stack used to store/restore clock valuations. Two stacks were independently. ETPDAs have the same expressiveness with TRTMs via weak timed bisimulation. The reachability problem of ETPDAs was undecidable, while the decidability held with a syntactic restriction on the stack.

Controller automata (CAs) [18, 11], was proposed to analyze interrupts. In a CA, a TA was assigned to each state. A TA at a state may be preempted by another state by a labeled transition. The number of clocks of CAs were finite, and thus when existing preemption loop, only the newest timed context were kept. Given a strict partial order over states, an ordered controller automaton was able to be faithfully encoded into a TA, and thus safety property of the restrictive version was preserved.

The *updatable timed automata (UTAs)* [9] proposed the possibility of updating the clocks in a more elaborate way, where the value of a clock could be reassigned to a basic arithmetic computation result of values of other clocks. UTAs raised up another way to accumulate time when timed context switches, and thus *updatable timed pushdown automata (UTPDAs)* could be an interesting extension.

7 Conclusion

This paper proposed a timed model called *nested timed automata (NeTAs)*. A NeTA was a pushdown system with a finite set of TAs as stack symbols. All clocks in the stack elapse uniformly, capable to model the timed behavior of the suspended components in the stack. The safety property of NeTAs was shown to be decidable by encoding NeTAs to DTPDAs. As an example of its application, behavior of multi-level interrupt handling is concisely modelled and its deadline analysis is encoded as a safety property.

We are planning to develop a tool based on NeTAs. Instead of general NeTAs, we will restrict a class such that a pop action occurs only with an integer-valued age. We expect this subclass of NeTAs can be encoded into updatable TPDAs (without local age), which would be more efficiently implemented.

Acknowledgements This work is supported by the NSFC-JSPS bilateral joint research project (61011140074), NSFC(61100052, 61003013, 61033002, 6126113 0589), and JSPS KAKENHI Grant-in-Aid for Scientific Research(B) (23300008 and 25280023).

References

1. Alur, R., Dill, D.L.: A Theory of Timed Automata. Theoretical Computer Science **126** (1994) 183–235

2. Bouajjani, A., Echahed, R., Robbana, R.: On the Automatic Verification of Systems with Continuous Variables and Unbounded Discrete Data Structures. In: Proceedings of the International Conference on Hybrid Systems: Computation and Control. LNCS 999, Springer-Verlag (1994) 64–85
3. Abdulla, P.A., Atig, M.F., Stenman, J.: Dense-Timed Pushdown Automata. In: Proceedings of the LICS'12, IEEE Computer Society (2012) 35–44
4. Trivedi, A., Wojtczak, D.: Recursive Timed Automata. In: Proceedings of the ATVA'10. LNCS 6252, Springer-Verlag (2010) 306–324
5. Benerecetti, M., Minopoli, S., Peron, A.: Analysis of Timed Recursive State Machines. In: Proceedings of the TIME'10, IEEE Computer Society (2010) 61–68
6. Li, G., Cai, X., Ogawa, M., Yuen, S.: Nested Timed Automata. Technical Report IS-RR-2013-004, JAIST (2013)
7. Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic Model Checking for Real-Time Systems. Information and Computation **111** (1994) 193–244
8. Ogawa, M., Cai, X.: On Reachability of Dense Timed Pushdown Automata. Technical Report IS-RR-2013-005, JAIST (2013)
9. Bouyer, P., Dufourd, C., Fleury, E., Petit, A.: Updatable Timed Automata. Theoretical Computer Science **321** (2004) 291–345
10. Bengtsson, J., Yi, W.: Timed Automata: Semantics, Algorithms and Tools. In: Lecture Notes on Concurrency and Petri Nets. LNCS 3098, Springer-Verlag (2004) 87–124
11. Li, G., Yuen, S., Adachi, M.: Environmental Simulation of Real-Time Systems with Nested Interrupts. In: Proceedings of the TASE'09, IEEE Computer Society (2009) 21–28
12. Brylow, D., Palsberg, J.: Deadline Analysis of Interrupt-Driven Software. IEEE Transactions on Software Engineering (TSE) **30** (2004) 634–655
13. Fersman, E., Krcal, P., Pettersson, P., Yi, W.: Task Automata: Schedulability, Decidability and Undecidability. Information and Computation **205** (2007) 1149–1172
14. Dang, Z.: Pushdown Timed Automata: a Binary Reachability Characterization and Safety Verification. Theoretical Computer Science **302** (2003) 93–121
15. Abdulla, P.A., Atig, M.F., Stenman, J.: The Minimal Cost Reachability Problem in Priced Timed Pushdown Systems. In: Proceedings of the LATA'12. LNCS 7183, Springer-Verlag (2012) 58–69
16. Alur, R., Benedikt, M., Etessami, K., Godefroid, P., Reps, T.W., Yannakakis, M.: Analysis of Recursive State Machines. ACM Transactions on Programming Languages and Systems (TOPLAS) **27** (2005) 786–818
17. David, A., Möller, M.O.: From HUPPAAL to UPPAAL - A Translation from Hierarchical Timed Automata to Flat Timed Automata. Technical Report RS-01-11, BRICS (2001)
18. Li, G., Cai, X., Yuen, S.: Modeling and Analysis of Real-Time Systems with Mutex Components. International Journal of Foundations of Computer Science (IJFCS) **23** (2012) 831–851

Well-Structured Pushdown Systems, Part 2: On Reachability of Dense Timed Pushdown Automata.

Mizuhito Ogawa
School of Information Science
Japan Advanced Institute of Science and Technology

Xiaojuan Cai
BASICS Lab, Shanghai Jiotong University

August 19, 2013

IS-RR-2013-005

Well-structured pushdown system, Part 2: On Reachability of Dense Timed Pushdown Automata ⋆

Mizuhito Ogawa[1] and Xiaojuan Cai[2]

[1] Japan Advanced Institute of Science and Technology
mizuhito@jaist.ac.jp
[2] Shanghai Jiao Tong University, China
cxj@sjtu.edu.cn

Abstract. This paper investigates a general framework of a pushdown system with well-quasi-ordered states and stack alphabet to show decidability of reachability. As an instance, an alternative proof of the decidability of the (configuration) reachability for dense-timed pushdown system (in *P.A. Abdulla, M.F. Atig, F. Stenman, Dense-Timed Pushdown Automata, IEEE LICS 2012*) is presented.

1 Introduction

Infinite state transition systems appear in many places still keeping certain decidable properties, e.g., pushdown systems (PDS), timed automata [3], and vector addition systems (VAS, or Petri nets). Well-structured transition systems (WSTSs) [2, 11] are one of successful general frameworks to reason about decidability (except for PDSs). The coverability of VASs, the reachability of communicating finite state machines with lossy channels [11], and the inclusion problem between timed automata with single clocks [15] are beginning of a long list.

A natural extension of WSTS is to associate a stack. It is tempting to apply Higman's lemma on stacks. However this fails immediately, since the monotonicity of transitions with respect to the embedding on stacks hardly holds.

This paper investigates a general framework for PDSs with well-quasi-ordered states and stack alphabet, *well-structured pushdown systems*. Well-quasi-orderings (WQOs) over states and stack alphabet are extended to configurations by the element-wise comparison. Note that this extension will not preserve WQO (nor well founded). By combining classical Pre^*-automaton technique [5, 12, 10], we reduce the argument on stacks to that on stack symbols, and similar to WSTS, finite convergence of antichain techniques during Pre^*-automata saturation is guaranteed by a WQO.

When the set P of states is finite, we have decidability of coverability [6]. When P is infinite (but equipped with WQO), we can state decidability of quasi-coverability only. To compensate, we introduce a well-formed projection $\Downarrow_{\mathcal{Y}}$,

⋆ JAIST Research Report IS-RR-2013-005, August 19th 2013

which extracts a core shape from the stack related to pushdown transitions. If we find $\Downarrow_\Upsilon$ such that, for configurations c, c' with $c \hookrightarrow c'$,

- **compatibility**: $\Downarrow_\Upsilon (c) \hookrightarrow \Downarrow_\Upsilon (c')$, and
- **stability**: $c \in \Upsilon$ if, and only if, $c' \in \Upsilon$, where $\Upsilon = \{c \mid c = \Downarrow_\Upsilon (c)\}$,

the quasi-coverability leads the configuration reachability. The compatibility strengthens the quasi-coverability to the coverability, and the stability boosts the coverability to the configuration reachability.

As an instance, we encode a dense-timed pushdown automaton (DTPDA) [1] into a snapshot PDS, inspired by the digitization techniques in [15]. A snapshot PDS has the set of snapshot words as stack alphabet. A snapshot word is essentially a region construction of the dimension equal to its size. Since a snapshot PDS contains non-standard pop rules (i.e., $(p, \gamma\gamma') \to (q, \gamma'')$), by associating a top stack symbol to a state, it is encoded as a PDS with WQO states and stack alphabet. Our general framework shows an alternative decidability proof of the reachability of a DTPDA, which has shown in [1].[3]

Our contribution is not on logically difficult proofs, but clarifying the proof structure behind theorems. Different from [1], our encoding idea into a snapshot PDS is inspired by [15].

Related Work

There are lots of works with context-sensitive infinite state systems. A process rewrite systems combines a PDS and a Petri net, in which vector additions/subtractions between adjacent stack frames during push/pop operations are prohibited [14]. With this restrictions, its reachability becomes decidable. A WQO automaton [7], is a WSTS with auxiliary storage (e.g., stacks and queues). It proves that the coverability is decidable under compatibility of *rank* functions with a WQO, of which an Multiset PDS is an instance. A timed pushdown automaton is a timed extension of a pushdown automaton. It has only global clocks, and the region construction [3] encodes it to a standard PDS [4, 8, 9]. DTPDA [1] firstly introduces local ages, which are stored with stack symbols when pushed, and never reset. DTPDA utilizes them to check whether an age in a stack frame satisfies constraints when pop occurs.

A WSPDS is firstly introduced in [6]. It focuses on WSPDSs with finite control states (and well-quasi-ordered stack alphabet), whereas the paper explores WSPDSs with well-quasi-ordered control states at the cost of weakening the target decidable property from the coverability to the quasi-coverability. The well-formed projection (Section 4), if exists, strengthens it again to the configuration reachability.

[3] In [1], only the state reachability is mentioned, but the proof is applied also for the configuration reachability.

Paper construction

The rest of the paper is constructed as follows. Section 2 briefly reviews a DTPDA [1]. Section 3 introduces P-automaton techniques for reachability of a pushdown system (PDS), which are extended to the coverability and the quasi-coverability. Note that we discuss on their correctness (at the limit), without assuming finite convergence. Section 4 proposes a *well-formed projection*. If we can find it, the quasi-coverability is lifted up to the configuration reachability. Section 5 introduces a Well-Structured Pushdown System (WSPDS) [6] and shows that the backward saturation of P-automaton (with upward ideals, in Section 3.3) finitely converges. Section 6 proposes *snapshot words*, which summarize and discretize the stack content as a (top) stack symbol. Section 7 presents the decidability of the reachability of a DTPDA, by encoding it into a WSPDS and finding a well-formed projection $\Downarrow_\Upsilon$ for snapshot words. Finally, Section 8 concludes the paper.

2 Dense-Timed Pushdown Automata

Dense-timed pushdown automaton (DTPDA) extends timed pushdown automaton (TPDA) with *local ages* [1]. A local age in each context is set when a push transition occurs, and restricts a pop transition only when the value of a local age meets the condition. The values of local ages proceed synchronously to global clocks, and they are never reset. Following [1], we omit input alphabet, since our focus is on reachability (regardless of an input word).

As notational convention, Section 2 and 7.2 use I for an interval (obeying to [1]), whereas Section 5 used I for an ideal.

Definition 1. *A DTPDA is a tuple $\langle S, s_{init}, \Gamma, \mathcal{C}, \Delta\rangle$, where*

- *S is a finite set of states with the initial state $s_{init} \in S$,*
- *Γ is a finite stack alphabet,*
- *$\mathcal{C}$ is a finite set of clocks, and*
- *Δ is a finite set of transitions.*

A transition $t \in \Delta$ is a triplet (s, op, s') in which $s, s' \in S$ and op is either of

- **Local** *nop, a state transition in S,*
- **Assignment** *$x \leftarrow I$, assign a clock $x \in \mathcal{C}$ to an arbitrary value in I,*
- **Test** *$x \in I?$, test whether the value of a clock $x \in \mathcal{C}$ is in I,*
- **Push** *$push(\gamma, I)$, push γ on a stack associated with a local age of an arbitrary value in I, and*
- **Pop** *$pop(\gamma, I)$, pop γ on a stack if the associated age a is in I.*

where I is an interval bounded by natural numbers (i.e., $[l,h], (l,h], [l,h), (l,h)$ for $l, h \in \mathbb{N} \cup \{\omega\}$ with $l \leq h$).

If each I in **Push** and **Pop** rules is $[0, \infty)$ (i.e., no conditions on local ages), we say simply a Timed Pushdown Automaton.

Definition 2. *For a DTPDA* $\langle S, s_{init}, \Gamma, \mathcal{C}, \Delta \rangle$*, a configuration is a triplet* (s, ν, w) *with* $s \in S$*, a clock valuation* $\nu : \mathcal{C} \to \mathbb{R}_{\geq 0}$*, and* $w \in (\Gamma \times \mathbb{R}_{\geq 0})^*$*. We refer* s *in a configuration* $c = (s, \nu, w)$ *by state*(c)*. For* $t \in \mathbb{R}_{\geq 0}$*, we denote*

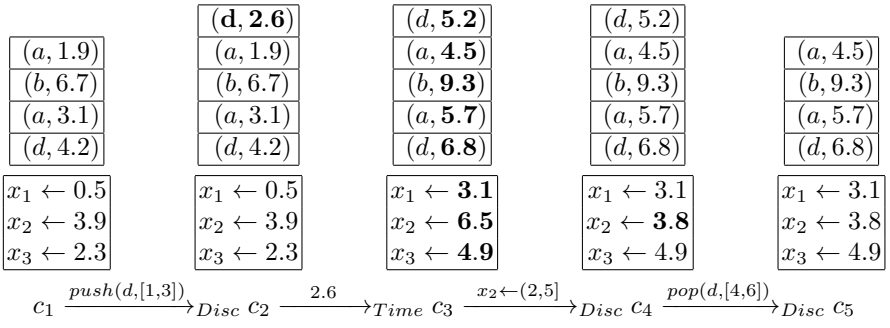
- $\nu_{\mathbf{0}}(x) = 0$ *for* $x \in \mathcal{C}$*,*
- $\nu_{x \leftarrow t}(x) = t$ *and* $\nu_{x \leftarrow t}(y) = \nu(y)$ *if* $y \neq x$*,*
- $(\nu + t)(x) = \nu(x) + t$ *for* $x \in \mathcal{C}$*, and*
- $w + t = (\gamma_1, t_1 + t).\cdots.(\gamma_k, t_k + t)$ *for* $w = (\gamma_1, t_1).\cdots.(\gamma_k, t_k)$*.*

There are two types of transitions, timed $\xrightarrow{t}_{Time}$ *and* discrete *transitions* $\xrightarrow{op}_{Disc}$*. Semantics of a timed transition is* $(s, \nu, w) \xrightarrow{t}_{Time} (s, \nu + t, w + t)$*, and a discrete transitions* (s, op, s') *is either*

- **Local** $(s, \nu, w) \xrightarrow{nop}_{Disc} (s', \nu, w)$*,*
- **Assignment** $(s, \nu, w) \xrightarrow{x \leftarrow I}_{Disc} (s', \nu_{x \leftarrow t}, w)$ *for* $t \in I$*,*
- **Test** $(s, \nu, w) \xrightarrow{x \in I?}_{Disc} (s', \nu, w)$ *if* $\nu(x) \in I$*,*
- **Push** $(s, \nu, w) \xrightarrow{push(\gamma, I)}_{Disc} (s', \nu, (\gamma, t).w)$ *for* $t \in I$*, and*
- **Pop** $(s, \nu, (\gamma, t).w) \xrightarrow{pop(\gamma, I)}_{Disc} (s', \nu, w)$ *if* $t \in I$*.*

We assume that the initial configuration is $(s_{init}, \nu_{\mathbf{0}}, \epsilon)$*.*

Example 1. The figure shows transitions between configurations in which $S = \{\bullet\}$ (omitted), $\mathcal{C} = \{x_1, x_2, x_3\}$, and $\Gamma = \{a, b, d\}$. From c_1 to c_2, a discrete transition $push(d, [1, 3])$ pushes $(d, 2.6)$ into the stack. At the timed transition from c_2 to c_3, 2.6 time units have elapsed, and each value grows older by 2.6. From c_3 to c_4, the value of x_2 is assigned to 3.8, which lies in the interval $(2, 5]$, and the last transition pops $(d, 5.2)$ after testing that its local age lies in $[4, 6]$.



3 P-automaton

A textbook standard technique to decide the emptiness of a pushdown automaton is, first converting it to context free grammar (with cubic explosion), and then applying CYK algorithm, which is a well-known dynamic programming technique. A practical alternative (with the same complexity) is a P-automaton [12, 10]. Starting from a regular set C of initial configurations (resp.

target configurations) $Post^*$ (resp. Pre^*) saturation procedure is applied on an initial P-automaton (which accepts C) until convergence. The resulting P-automaton accepts the set of all successors (resp. predecessors) of C. In literature, this technique is applied only for PDSs with finite control states and stack alphabet. We confirm that it works for PDSs without finite restriction (ignoring finite convergence), and extend it to the coverability and the quasi-coverability.

3.1 P-automaton for reachability of pushdown system

In the standard definition, a pushdown system (PDS) has a finite set of states and finite stack alphabet. We will consider a PDS with an infinite set of states and infinite stack alphabet. For (possibly infinitely many) individual transition rules, we introduce a partial function ψ to describe a pattern of transitions. We denote the set of partial functions from X to Y by $\mathcal{P}Fun(X, Y)$.

Definition 3. *A* pushdown system *(PDS) $\mathcal{M} = \langle P, \Gamma, \Delta \rangle$ consists of a finite set $\Delta \subseteq \mathcal{P}Fun(P \times \Gamma, P \times \Gamma^2) \cup \mathcal{P}Fun(P \times \Gamma, P \times \Gamma) \cup \mathcal{P}Fun(P \times \Gamma, P)$ of transition rules. We say that $\psi \in \Delta$ is a push, internal, and pop rule if $\psi \in \mathcal{P}Fun(P \times \Gamma, P \times \Gamma^2)$, $\psi \in \mathcal{P}Fun(P \times \Gamma, P \times \Gamma)$, and $\psi \in \mathcal{P}Fun(P \times \Gamma, P)$, respectively. A* configuration *$\langle p, w \rangle$ consists of $p \in P$ and $w \in \Gamma^*$. For a transition rule $\psi \in \Delta$, a transition is $\langle p, \gamma w \rangle \hookrightarrow \langle p', vw \rangle$ for $(p', v) = \psi(p, \gamma)$*

Remark 1. Often in multi-thread program modelings and in snapshot PDSs (Section 7.2) for discretizing DTPDAs, PDSs are defined with finite control states, but with non-standard pop rules, like $\langle p, \gamma_1\gamma_2 \rangle \hookrightarrow \langle q, \gamma \rangle \in \mathcal{P}Fun(P \times \Gamma^2, P \times \Gamma)$ with $|P| < \infty$. This can be encoded into PDSs in Definition 3 by associating a top stack symbol to a state, like $\langle (p, \gamma_1), \gamma_2 \rangle \hookrightarrow \langle (q, \gamma), \epsilon \rangle \in \mathcal{P}Fun(P' \times \Gamma, P')$ with $P' = P \times \Gamma$, at the cost that the set P' of control states becomes infinite.

We use $c_1, c_2, \cdots$ to range over configurations. $\hookrightarrow^*$ is the reflexive transitive closure of $\hookrightarrow$. There are two kinds of reachability problems.

- **Configuration reachability** : Given configurations $\langle p, w \rangle$, $\langle q, v \rangle$ with $p, q \in P$ and $w, v \in \Gamma^*$, decide whether $\langle p, w \rangle \hookrightarrow^* \langle q, v \rangle$.
- **State reachability** : Given a configuration $\langle p, w \rangle$ and a state q with $p, q \in P$ and $w \in \Gamma^*$, decide whether there exists $v \in \Gamma^*$ with $\langle p, w \rangle \hookrightarrow^* \langle q, v \rangle$.

Given a set of configurations C, we write $pre^*(C)$ (resp. $post^*(C)$) for the set $\{c' \mid c' \hookrightarrow^* c \wedge c \in C\}$ (resp. $\{c' \mid c \hookrightarrow^* c' \wedge c \in C\}$). The reachability problem from $\langle p, w \rangle$ to $\langle q, v \rangle$ is reduced to whether $c \in pre^*(\{c'\})$ (or $c' \in post^*(\{c\})$).

Definition 4. *A Pre^*-automaton $\mathcal{A}$ is a quadruplet (S, Γ, ∇, F) with $F \subseteq S$ and $\nabla \subseteq S \times \Gamma \times S$. A Pre^*-automaton is* initial *if each state in $S \cap P$ has no incoming transitions and S is finite. $\mathcal{A}$ accepts a configuration $\langle p, w \rangle$ with $p \in P$ and $w \in \Gamma^*$, if w is accepted starting from p (as an initial state).*

The set of configurations accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$. When $(p, \gamma, q) \in \nabla$, we denote $p \overset{\gamma}{\mapsto} q$. For $w = \gamma_1 \dots \gamma_k \in \Gamma^*$, $p \overset{\gamma_1}{\mapsto} \cdot \dots \overset{\gamma_k}{\mapsto} q$ is denoted by $p \overset{w}{\mapsto}{}^* q \in \nabla^*$. If $k = 0$ (i.e., $p \overset{\epsilon}{\mapsto} q$), we assume $p = q$.

Starting from an initial Pre^*-automaton $\mathcal{A}_0$ that accepts C (i.e., $C = L(A_0)$), the repeated (possibly infinite) applications of saturation rules

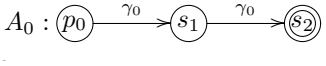
$$\frac{(S, \Gamma, \nabla, F)}{(S \cup \{p'\}, \Gamma, \nabla \cup \{p' \overset{\gamma}{\mapsto} q\}, F)} \quad \text{if } p \overset{w}{\mapsto}{}^* q \in \nabla^* \text{ and } \psi(p', \gamma) = (p, w) \text{ for } \psi \in \Delta$$

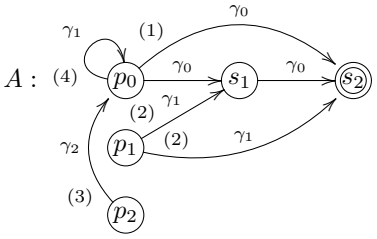
converges to $Pre^*(\mathcal{A}_0)$. Note that saturation rules never eliminate transitions, but monotonically enlarge. When $(p, \gamma, q) \in \nabla$, we denote $p \overset{\gamma}{\mapsto} q$.

Theorem 1. *[12, 10] (Theorem 1 in [6]) For a PDS, $pre^*(C) = L(Pre^*(A_0))$. where $C = L(A_0)$.*

Example 2. Let $\langle \{p_i\}, \{\gamma_i\}, \Delta \rangle$ be a pushdown system with $i = 0, 1, 2$ and Δ given below. The saturation $\mathcal{A}$ of Pre^*-automata started from $\mathcal{A}_0$ accepting $C = \{\langle p_0, \gamma_0\gamma_0 \rangle\}$. $L(\mathcal{A})$ coincides $pre^*(C)$.

(1). $\langle p_0, \gamma_0 \rangle \to \langle p_1, \gamma_1\gamma_0 \rangle$
(2). $\langle p_1, \gamma_1 \rangle \to \langle p_2, \gamma_2\gamma_0 \rangle$
(3). $\langle p_2, \gamma_2 \rangle \to \langle p_0, \gamma_1 \rangle$
(4). $\langle p_0, \gamma_1 \rangle \hookrightarrow \langle p_0, \epsilon \rangle$





Remark 2. Since the saturation procedure monotonically extends Pre^*-automaton, even if a PDS has an infinite set of states / stack alphabet and the initial Pre^*-automaton $\mathcal{A}_0$ has infinite states, it converges (after infinite many saturation steps), and $pre^*(C) = L(Pre^*(A_0))$ holds.

3.2 P-automata for coverability of OPDS

A quasi-ordering (QO) is a reflexive transitive binary relation. We denote the upward (resp. downward) closure of X by $X^\uparrow$ (resp. $X^\downarrow$), i.e., $X^\uparrow = \{y \mid \exists x \in X. x \leq y\}$ (resp. $X^\downarrow = \{y \mid \exists x \in X. y \leq x\}$).

For a PDS $\mathcal{M} = \langle P, \Gamma, \Delta \rangle$, we introduce QOs $(P, \preceq)$ and $(\Gamma, \leq)$ on P and Γ, respectively. We call $\mathcal{M} = \langle (P, \preceq), (\Gamma, \leq), \Delta \rangle$ an ordered PDS (OPDS).

Definition 5. *For $w_1 = \alpha_1\alpha_2 \cdots \alpha_n, w_2 = \beta_1\beta_2 \cdots \beta_m \in \Gamma^*$, let*

- **Element-wise comparison** $w_1 \lessdot w_2$ *if $m = n$ and $\forall i \in [1..n]. \alpha_i \leq \beta_i$.*
- **Embedding** $w_1 \preccurlyeq w_2$ *if there is an order-preserving injection f from $[0..n]$ to $[0..m]$ with $\alpha_i \leq \beta_{f(i)}$ for each $i \in [0..n]$.*

We extend $\lessdot$ on configurations such that $(p, w) \lessdot (q, v)$ if $p \preceq q$ and $w \lessdot v$.

A partial function $\psi \in \mathcal{P}Fun(X, Y)$ is *monotonic* if $\gamma \leq \gamma'$ and $\gamma \in dom(\psi)$ imply $\psi(\gamma) \lessdot \psi(\gamma')$ and $\gamma' \in dom(\psi)$ for each $\gamma, \gamma' \in \Gamma$. We say that an OPDS $\mathcal{M} = \langle (P, \preceq), (\Gamma, \leq), \Delta \rangle$ is *monotonic* if ψ is monotonic for each $\psi \in \Delta$.

– **Coverability** : Given configurations (p, w), (q, v) with $p, q \in P$ and $w, v \in \Gamma^*$, decide whether there exists $v' \in \Gamma^*$ with $v \lessdot v'$ and $(p, w) \hookrightarrow^* (q, v')$.

Coverability is reduced to whether $(p, w) \in pre^*(\{(q, v)\}^\uparrow)$. For coverability, we restrict saturation rules of Pre^*-automata.

$$\frac{(S, \Gamma, \nabla, F)}{(S \cup \{p'\}, \Gamma, \nabla \oplus \{p' \overset{\gamma}{\mapsto} q\}, F)} \quad \begin{array}{l}\text{if } p \overset{w}{\mapsto}{}^* q \in \nabla^* \text{ and} \\ \psi(p', \gamma) \in \{(p, w)\}^\uparrow \text{ for } \psi \in \Delta\end{array}$$

where $\nabla \oplus \{p' \overset{\gamma}{\mapsto} q\}$ is

$$\begin{cases} \nabla & \text{if there exists } \{p'' \overset{\gamma'}{\mapsto} q\} \in \nabla \text{ with } p'' \preceq p' \text{ and } \gamma' \leq \gamma \\ \nabla \cup \{p' \overset{\gamma}{\mapsto} q\} & \text{otherwise.} \end{cases}$$

Theorem 2. *(Theorem 3 in [6]) For a monotonic OPDS, $pre^*(C^\uparrow) = L(Pre^*(A_0))^\uparrow$. where $C^\uparrow = L(A_0)$.*

3.3 P-automata for quasi-coverability of OPDS

– **Quasi-coverability**. Given configurations $\langle p, w\rangle$, $\langle q, v\rangle$, decide whether there exist $\langle p', w'\rangle$ and $\langle q', v'\rangle$ such that $\langle p, w\rangle \lessdot \langle p', w'\rangle$, $\langle q, v\rangle \lessdot \langle q', v'\rangle$, and $\langle p', w'\rangle \hookrightarrow^* \langle q', v'\rangle$.

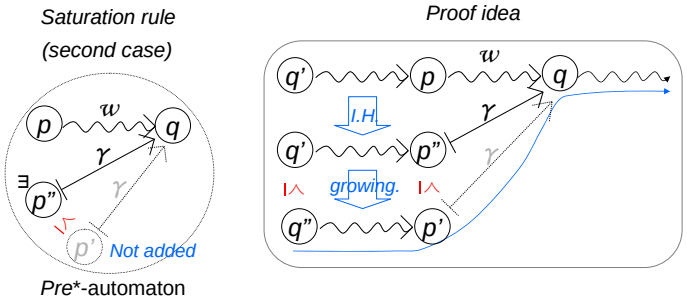
Quasi-coverability is reduced to whether $\langle p, w\rangle \in pre^*(\{(q, v)\}^\uparrow)^\downarrow$. For quasi-coverability, we further restrict saturation rules of Pre^*-automata.

$$\frac{(S, \Gamma, \nabla, F)}{(S \cup \{p'\}, \Gamma, \nabla \oplus \{p' \overset{\gamma}{\mapsto} q\}, F)} \quad \begin{array}{l}\text{if } p \overset{w}{\mapsto}{}^* q \in \nabla^* \text{ and} \\ \psi(p', \gamma) \in \{(p, w)\}^\uparrow \text{ for } \psi \in \Delta\end{array}$$

where $\nabla \oplus \{p' \overset{\gamma}{\mapsto} q\}$ is

$$\begin{cases} \nabla & \text{if there exists } \{p'' \overset{\gamma'}{\mapsto} q\} \in \nabla \text{ with } p'' \preceq p' \text{ and } \gamma' \leq \gamma \\ \nabla \cup \{p'' \overset{\gamma}{\mapsto} q\} & \text{if there exists } p'' \in S \cap P \text{ with } p'' \preceq p' \\ \nabla \cup \{p' \overset{\gamma}{\mapsto} q\} & \text{otherwise.} \end{cases}$$

The second condition (illustrated in the figure below) suppresses adding new states in Pre^*-automata, and the first condition gives a termination condition for adding new edges.



Definition 6. *An OPDS* $\mathcal{M} = \langle (P, \preceq), (\Gamma, \leq), \Delta \rangle$ *is* growing *if, for each* $\psi(p, \gamma) = (q, w)$ *with* $\psi \in \Delta$ *and* $(q', w') \geqslant (q, w)$*, there exists* (p', γ') *with* $(p', \gamma') \geqslant (p, \gamma)$ *such that* $\psi(p', \gamma') \geqslant (q', w')$*.*

Lemma 1 is obtained by induction on steps of Pre^*-automata saturation, of which the proof idea is illustrated in the figure above.

Lemma 1. *Assume* $p \overset{w}{\longmapsto}^* s$ *in* $Pre^*(\mathcal{A}_0)$*. For each* $(p', w') \geqslant (p, w)$*,*

- *If* $s \in P$*, there exist* $(p'', w'') \geqslant (p', w')$ *and* $q' \succeq s$ *with* $\langle p'', w'' \rangle \hookrightarrow^* \langle q', \epsilon \rangle$*.*
- *If* $s \in S \setminus P$*, there exist* $(p'', w'') \geqslant (p', w')$*,* $q \overset{v}{\longmapsto}^* s$ *in* $\mathcal{A}_0$ *with* $q \in P$*, and* $\langle q', v' \rangle \geqslant \langle q, v \rangle$ *such that* $\langle p'', w'' \rangle \hookrightarrow^* \langle q', v' \rangle$*.*

For simplicity, we say "c_0 covers c_1" to mean that there exists $c_1' \geqslant c_1$ with $c_0 \hookrightarrow^* c_1'$. The next **Claim** is easily proved by induction on the steps of $\hookrightarrow$.

Claim For a monotonic and growing OPDS, if $\langle p, w \rangle \hookrightarrow^* \langle q, v \rangle$, then for any $(q', v') \geqslant (q, v)$, there exists $(p', w') \geqslant (p, w)$ such that $\langle p', w' \rangle$ covers $\langle q', v' \rangle$.

Proof. By induction on steps of the Pre^* saturation procedure $\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \cdots$. For $\mathcal{A}_0$, the statements hold immediately. Assume the statements hold for $\mathcal{A}_i$, and $\mathcal{A}_{i+1}$ is constructed by adding new transition $p_0 \overset{\gamma_0}{\mapsto} q_0$.

$$\frac{(S, \Gamma, \nabla, F)}{(S \cup \{p_0\}, \Gamma, \nabla \oplus \{p_0 \overset{\gamma_0}{\mapsto} q_0\}, F)} \quad \begin{array}{l} \text{if } p_1 \overset{w_1}{\mapsto}^* q_0 \in \nabla^* \text{ and} \\ \psi(p_0, \gamma_0) \in \{(p_1, w_1)\}^\uparrow \text{ for } \psi \in \Delta \end{array}$$

We give a proof only for the first statement. The second statement is similarly proved. According to the definition of $\oplus$, there are three cases:

- There exists $\{p_0' \overset{\gamma_0'}{\mapsto} q_0\} \in \nabla$ with $p_0' \preceq p_0$ and $\gamma_0' \leq \gamma_0$. Nothing added.
- There exists p_0' in $S \cap P$ and $p_0' \preceq p_0$. Then, $p_0' \overset{\gamma_0}{\mapsto} q_0$ is added.
- Otherwise. $p_0 \overset{\gamma_0}{\mapsto} q_0$ is added.

The second case is the most complex, and we focus on it. Assume that a path $p \overset{w}{\longmapsto}^* q$ contains $p_0' \overset{\gamma_0}{\mapsto} q_0$ k-times. We apply (nested) induction on k, and we focus on its leftmost occurrence. Let $w = w_l \gamma_0 w_r$ and $p \overset{w_l}{\longmapsto}^* p_0' \overset{\gamma_0}{\mapsto} q_0 \overset{w_r}{\longmapsto}^* q$. For each $p' \succeq p, w_l' \geqslant w_l, w_r' \geqslant w_r$ and $\gamma_0' \geq \gamma_0$:

1. By induction hypothesis on $p \overset{w_l}{\longmapsto}^* p_0'$, there exists $(p'', w_l'') \geqslant (p', w_l')$ such that $\langle p'', w_l'' \rangle$ covers $\langle p_0', \epsilon \rangle$.
2. By the definition of saturation rules, there exist $p_1' \succeq p_1$ and $w_1' \geqslant w_1$ such that $\langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_1', w_1' \rangle$.
3. By induction hypothesis on $p_1 \overset{w_1 w_r}{\longmapsto}^* q$, there exist $p_1'' \succeq p_1'$ and $w_1'' w_r'' \geqslant w_1' w_r'$ such that $\langle p_1'', w_1'' w_r'' \rangle$ covers $\langle q, \epsilon \rangle$.
4. By the growing property, there exist $p_0'' \succeq p_0 \succeq p_0'$ and $\gamma_0'' \geq \gamma_0'$ such that $\langle p_0'', \gamma_0'' \rangle$ covers $\langle p_1'', w_1'' \rangle$.

By **Claim** and 1., there exists $(p''', w_l''') \geqslant (p'', w_l'') \geqslant (p', w_l')$ such that $\langle p''', w_l'''\rangle$ covers $\langle p_0'', \epsilon\rangle$. Put all these together, for each $(p', w_l'\gamma_0'w_r') \geqslant (p, w_l\gamma_0 w_r)$, there exists $(p''', w_l'''\gamma_0''w_r'') \geqslant (p', w_l'\gamma_0'w_r')$. Therefore, each of $\langle p''', w_l'''\gamma_0''w_r''\rangle$, $\langle p_0'', \gamma_0''w_r''\rangle$, $\langle p_1'', w_1''w_r''\rangle$, and $\langle q, \epsilon\rangle$ covers the next. □

From Lemma 1, Theorem 3 is immediate.

Theorem 3. *For a monotonic and growing OPDS,* $pre^*(C^\uparrow)^\downarrow = (L(Pre^*(A_0))^\uparrow)^\downarrow$. *where* $C^\uparrow = L(A_0)$.

4 Well-formed projection and well-formed constraint

Definition 7. *For an OPDS* M, *a pair* $(\Upsilon, \Downarrow_\Upsilon)$ *of a set* $\Upsilon \subseteq P \times \Gamma^*$ *and a projection function* $\Downarrow_\Upsilon\colon P \times \Gamma^* \to (P \times \Gamma^*) \cup \{\#\}$ *is a* well-formed projection *if, for configurations* c, c' *with* $c \hookrightarrow c'$,

- $c \in \Upsilon$ *if, and only if* $c' \in \Upsilon$,
- $\Downarrow_\Upsilon (c) \hookrightarrow \Downarrow_\Upsilon (c')$,
- $\Downarrow_\Upsilon (c) \lessdot c$, *and*
- $c_1 \lessdot c_2$ *implies either* $\Downarrow_\Upsilon (c_1) = \Downarrow_\Upsilon (c_2)$ *or* $\Downarrow_\Upsilon (c_1) = \#$,

where $\#$ *is added to* $P \times \Gamma^*$ *as the least element (wrt* $\lessdot$*) and* $\Upsilon = \{c \in P \times \Gamma^* \mid c = \Downarrow_\Upsilon (c)\}$. Υ *is called a* well-formed constraint. *(*$\#$ *represents failures of* $\Downarrow_\Upsilon$*.)*

Lemma 2. *For a monotonic OPDS* M *with a well-formed projection* $\Downarrow_\Upsilon$, *assume* $C \subseteq \Upsilon$. *Then,* $pre^*(C) = pre^*(C^\uparrow)^\downarrow \cap \Upsilon$.

Proof. From $C \subseteq \Upsilon$, $pre^*(C) \subseteq pre^*(C^\uparrow)^\downarrow \cap \Upsilon$ is obvious, For the opposite direction, we first show $\Downarrow_\Upsilon (pre^*(C^\uparrow)) \subseteq pre^*(C)$. Since $c \in pre^*(C^\uparrow)$ is equivalent to $\exists c' \in C^\uparrow . c \hookrightarrow^* c'$, we have $\Downarrow_\Upsilon (c) \hookrightarrow^* \Downarrow_\Upsilon (c') \in C$. Since $C \subseteq \Upsilon$ implies $\Downarrow_\Upsilon (c') \in C$, $\Downarrow_\Upsilon (c) \in pre^*(C)$ is obtained. For $pre^*(C) \supseteq pre^*(C^\uparrow)^\downarrow \cap \Upsilon$,

$$pre^*(C^\uparrow)^\downarrow \cap \Upsilon = \Downarrow_\Upsilon (pre^*(C^\uparrow)^\downarrow \cap \Upsilon) \subseteq \Downarrow_\Upsilon (pre^*(C^\uparrow)^\downarrow) = \Downarrow_\Upsilon (pre^*(C^\uparrow)) \cup \{\#\}.$$

From $\Downarrow_\Upsilon (pre^*(C^\uparrow)) \subseteq pre^*(C)$, $\Downarrow_\Upsilon (pre^*(C^\uparrow)) \cup \{\#\} \subseteq pre^*(C) \cup \{\#\}$. Thus, $pre^*(C^\uparrow)^\downarrow \cap \Upsilon \subseteq (pre^*(C) \cup \{\#\}) \cap \Upsilon = pre^*(C)$. ■

From Theorem 3 and Lemma 2, Theorem 4 is immediate, which strengthens the quasi-coverability to the configuration reachability, and the decidability is reduced to finite convergence of $L(Pre^*(\mathcal{A}_0))$.

Theorem 4. *Let* C *be a regular set of configurations with a P-automaton* $\mathcal{A}_0$) *with* $C^\uparrow = L(A_0)$. *For a monotonic and growing OPDS and a well-formed constraint* Υ, $pre^*(C) = L(Pre^*(A_0))^\downarrow \cap \Upsilon$.

Example 3. In Example 4, let Υ be

$$\left\{ \begin{array}{l} \langle p_0, (n,n)\cdots(0,0)\rangle, \langle p_2, (n,n)\cdots(0,0)\rangle \\ \langle p_1, (n,n-2)(n-1,n-1)\cdots(0,0)\rangle, \end{array} \mid n \geq m \geq 0 \right\}$$

It is easy to see that Υ is compatible. Since both $\langle p_0, (0,0)\rangle$ and $\langle p_2, (0,0)\rangle$ are in Υ and $\{\langle p, (0,0)\rangle\}^\uparrow \cap \Upsilon = \{\langle p, (0,0)\rangle\}$, we conclude that $\langle p_0, (0,0)\rangle \hookrightarrow^* \langle p_2, (0,0)\rangle$ by Theorem 4.

5 Finite convergence of Pre^*-automata

Definition 8. *A QO $\leq$ is a* well-quasi-ordering *(WQO) if, for each infinite sequence $a_1, a_2, \cdots$, there exist i, j with $i < j$ and $a_i \leq a_j$.*

A QO $\leq$ is a WQO, if, and only if each upward closed set $X^{\uparrow}$ has finite basis (i.e., minimal elements). Note that $\lll$ may be no longer a WQO (nor well founded), while the embedding $(\Gamma^*, \preccurlyeq)$ stays a WQO by *Higman's lemma*.

Lemma 3. *Let $(D, \leq)$ and $(D', \leq')$ be WQOs.*

- **(Dickson's lemma)** *$(D \times D', \leq \times \leq')$ is a WQO.*
- **(Higman's lemma)** *$(D^*, \preccurlyeq)$ is a WQO, where $\preccurlyeq$ is the embedding.*

For a monotonic OPDS, if $(P, \preceq), (\Gamma, \leq)$ are WQOs, we call it *a Well-Structured PDS* (WSPDS). For a WSPDS $((P, \preceq), (\Gamma, \leq), \Delta)$, $\psi^{-1}(\{(p, w)\}^{\uparrow})$ is upward-closed and has finite basis (i.e., finitely many minimal elements). In the Pre^* saturation rule of Section 3.3, its side condition contains $\psi(p', \gamma) \in \{(p, w)\}^{\uparrow}$ for $\psi \in \Delta$, which allows arbitrary choices of (p', γ). For a WSPDS, we focus only on finite basis of upward-closed sets $(p', \gamma) \in Min(\psi^{-1}(\{(p, w)\}^{\uparrow}))$.

We assume that such finite basis are computable for each $\psi \in \Delta$, and the initial Pre^*-automaton has finitely many states S_0.

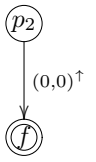
Theorem 5. *For a WSPDS $((P, \preceq), (\Gamma, \leq), \Delta)$, if (i) $(P, \preceq), (\Gamma, \leq)$ are computable WQOs, and (ii) a finite basis of $\psi^{-1}(\{(p, w)\}^{\uparrow})$ is computable for each $\psi \in \Delta$ and $\langle p, w \rangle \in P \times \Gamma^{\leq 2}$, $Pre^*(\mathcal{A}_0)$ finitely converges.*

Proof. (*Sketch*) Starting from a WQO over S such that $\preceq$ over $S_0 \cap P$ and $=$ on $S_0 \setminus P$, the set S of states of the Pre^*-automaton make a bad sequence, since saturation rules in Section 3.3 do not add larger states. For each pair (p, q) of states, they also do not add larger stack symbols as labels of Pre^* automaton transitions $p \overset{\gamma}{\mapsto} q$. Thus, during the saturation procedure, a sequence of added edges $p_1 \overset{\gamma_1}{\mapsto} q_1, p_2 \overset{\gamma_2}{\mapsto} q_2, \cdots$ is bad. Thus, it finitely terminates. Since Δ has finitely many transition rules (i.e., partial functions), dependency during generation of Pre^* automaton transitions is finitely branching. Thus, by König's lemma, $Pre^*(\mathcal{A}_0)$ finitely converges. □

Example 4. Let $M = \langle \{p_i\}, \mathbb{N}^2, \Delta \rangle$ be a monotonic OPDS with vectors in $\mathbb{N}^2$ as a stack alphabet and Δ consists of four rules given in the figure. The figure illustrates a Pre^*-automaton construction starting from initial $\mathcal{A}_0$ that accepts $C = \langle p_2, (0, 0)^{\uparrow} \rangle$. For $v \in \mathbb{N}^2$, we abbreviate $\{v\}^{\uparrow}$ by $v^{\uparrow}$. Note that $\mathbb{N}^2$ is WQO by the element-wise comparison. $\mathcal{A}$ is the saturation of the Pre^*-automaton.

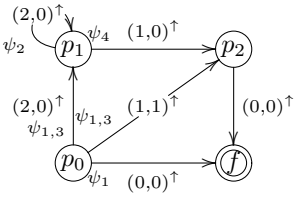
For instance, when $m = 2$, $p_0 \overset{(2,2)^{\uparrow}}{\mapsto} p_1$ in $\mathcal{A}$ is generated from $p_1 \overset{(2,0)^{\uparrow}}{\mapsto} p_1$ by ψ_3. By repeating application of ψ_1 twice to $p_0 \overset{(2,2)^{\uparrow}}{\mapsto} p_1 \overset{(2,0)^{\uparrow}}{\mapsto} p_1$, we obtain $p_0 \overset{(2,0)^{\uparrow}}{\mapsto} p_1$. Then, applying ψ_1 to $p_0 \overset{(2,0)^{\uparrow}}{\mapsto} p_1 \overset{(1,0)^{\uparrow}}{\mapsto} p_2$, we obtain $p_0 \overset{(1,0)^{\uparrow}}{\mapsto} p_2$. $p_0 \overset{(1,2)^{\uparrow}}{\mapsto} p_2$ is also generated from $p_1 \overset{(1,0)^{\uparrow}}{\mapsto} p_2$ by ψ_3 (since $\psi_3^{-1}(\{(1, 0)\}^{\uparrow}) = \{(1, 2)\}^{\uparrow}$), but it will not affect.

$\mathcal{A}_0$:

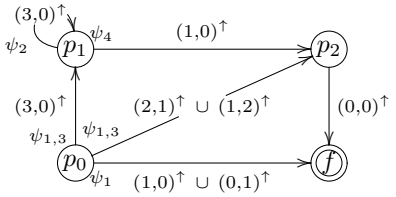


$$
\begin{array}{ll}
\psi_1 : \langle p_0, v\rangle \to \langle p_0, (v+(1,1))v\rangle & \\
\psi_2 : \langle p_1, v\rangle \to \langle p_1, \epsilon\rangle & \text{if } v \geq (m,0) \\
\psi_3 : \langle p_0, v\rangle \to \langle p_1, v-(0,2)\rangle & \text{if } v \geq (0,2) \\
\psi_4 : \langle p_1, v\rangle \to \langle p_2, \epsilon\rangle & \text{if } v \geq (1,0)
\end{array}
$$

$\mathcal{A}$: $(m=2)$



$\mathcal{A}'$: $(m=3)$



By Theorem 2, we obtain

$$
\begin{aligned}
pre^*(C) = \{&\langle p_2, (0,0)^\uparrow\rangle, \langle p_1, ((2,0)^\uparrow)^*(1,0)^\uparrow(0,0)^\uparrow\rangle, \\
&\langle p_0, (0,0)^\uparrow\rangle, \langle p_0, (1,1)^\uparrow(0,0)^\uparrow\rangle, \langle p_0, ((2,0)^\uparrow)^+(1,0)^\uparrow(0,0)^\uparrow\rangle\}
\end{aligned}
$$

Thus, $\langle p_0, (0,0)\rangle$ covers $\langle p_2, (0,0)\rangle$. Actually,

$$
\begin{aligned}
\langle p_0, (0,0))\rangle &\hookrightarrow \langle p_0, (1,1)(0,0)\rangle \hookrightarrow \langle p_0, (2,2)(1,1)(0,0)\rangle \hookrightarrow \langle p_1, (2,0)(1,1)(0,0)\rangle \\
&\hookrightarrow \langle p_1, (1,1)(0,0)\rangle \hookrightarrow \langle p_2, (0,0)\rangle
\end{aligned}
$$

Note that if we change the condition of ψ_2 from $v \geq (2,0)$ to $v \geq (3,0)$, the saturated Pre^*-automaton becomes $\mathcal{A}'$, and $\langle p_0, (0,0)\rangle$ no more covers $\langle p_2, (0,0)\rangle$, though $\langle p_0, (0,0)\rangle$ is reachable to p_2. Actually,

$$
\begin{aligned}
\langle p_0, (0,0))\rangle &\hookrightarrow \langle p_0, (1,1)(0,0)\rangle \hookrightarrow \langle p_0, (2,2)(1,1)(0,0)\rangle \hookrightarrow \langle p_0, (3,3)(2,2)(1,1)(0,0)\rangle \\
&\hookrightarrow \langle p_1, (3,1)(2,2)(1,1)(0,0)\rangle \hookrightarrow \langle p_1, (2,2)(1,1)(0,0)\rangle \hookrightarrow \langle p_2, (1,1)(0,0)\rangle
\end{aligned}
$$

To detect the state reachability, instead of $\mathcal{A}_0$, we can start with an initial automaton $\mathcal{A}'_0$ that accepts $p_2 \times \Gamma^* = \{\langle p_2, ((0,0)^\uparrow)^*\}$.

6 Snapshot Word

In a DTPDA, the stack content is a sequence of pairs of stack symbols and local ages. When a DTPDA is encoded into a discrete WSPDS, it can operate only the top stack symbol. Such a target WSPDS is a *snapshot PDS* (Section 7.2), of which stack symbols are snapshot words. A *snapshot word* summarizes the ordering of fractions of all local ages and global clocks in the stack, after applying the digitization technique in [15], whereas the encoding in [1] summarizes global clocks and an age in the top stack frame and copies of global clocks in the next stack frame. Then, a snapshot PDS handles all timed behavior at the top stack symbol, and left untouched inside the stack. When a pop occurs, time progress recorded at the top stack symbol is propagated to the next stack symbol after finding a permutation (of time progress) by matching via markings ρ_1 and ρ_2.

6.1 Snapshot word

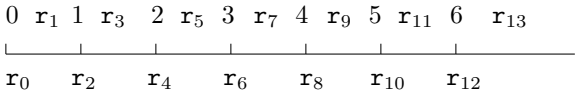
As notational convention, let $\mathcal{MP}(D)$ be the set of finite multisets over D. We regard a finite set as a multiset in which the multiplicity of each element is 1. For a finite word $w = a_1 a_2 \cdots a_k$, we denote $w(j) = a_j$

Let $\langle S, s_{init}, \Gamma, \mathcal{C}, \Delta \rangle$ be a DTPDA, and let n be the largest integer (except for ∞) that appears in Δ. For $v \in \mathbb{R}_{\geq 0}$, $proj(v) = \mathtt{r}_i$ if $v \in \mathtt{r}_i \in Intv(n)$ and

$$Intv(n) = \begin{cases} \mathtt{r}_{2i} & = [i,i] \quad \text{if } 0 \leq i \leq n \\ \mathtt{r}_{2i+1} & = (i, i+1) \text{ if } 0 \leq i < n \\ \mathtt{r}_{2n+1} & = (n, \infty) \end{cases}$$

Definition 9. *Let $frac(x,t) = t - floor(t)$ for $(x,t) \in (\mathcal{C} \cup \Gamma) \times \mathbb{R}_{\geq 0}$. A* digitization $\mathtt{digi} : \mathcal{MP}((\mathcal{C} \cup \Gamma) \times \mathbb{R}_{\geq 0}) \to (\mathcal{MP}((\mathcal{C} \cup \Gamma) \times Intv(n)))^*$ *is as follows. For $\mathcal{X} \in \mathcal{MP}((\mathcal{C} \cup \Gamma) \times \mathbb{R}_{\geq 0})$, let $X_1, \cdots, X_k$ be multisets that collect $(x, proj(t))$'s in $\mathcal{X}$ having the same $frac(x,t)$. We assume that X_i's are sorted by the increasing order of $frac(x,t)$ (i.e., , $frac(x,t) < frac(x',t')$ for $(x, proj(t)) \in X_i$ and $(x', proj(t')) \in X_{i+1}$). Then, $\mathtt{digi}(\mathcal{X})$ is a word $X_1 \cdots X_k$.*

Example 5. In Example 1, $n = 6$ and we have 13 intervals illustrated below.



From the configuration c_1 in Example 1, the clock information is extracted from the stack content of c_1 as a multiset

$$\mathcal{X} = \{(a, 1.9), (b, 6.7), (a, 3.1), (d, 4.2), (x_1, 0.5), (x_2, 3.9), (x_3, 2.3)\}$$

and $\mathtt{digi}(\mathcal{X}) = \{(a, \mathtt{r}_7)\}\{(d, \mathtt{r}_9)\}\{(x_3, \mathtt{r}_5)\}\{(x_1, \mathtt{r}_1)\}\{(b, \mathtt{r}_{13})\}\{(x_2, \mathtt{r}_7), (a, \mathtt{r}_3)\}$. For instance, The value of the clock x_2 and the age of the top stack frame $(a, 1.9)$ have the same fraction 0.9, thus they are packed into the same multiset $\{(x_2, \mathtt{r}_7), (a, \mathtt{r}_3)\}$, and placed at the last since their fraction is the largest.

Definition 10. *A word $\bar{\gamma} \in (\mathcal{MP}((\mathcal{C} \cup \Gamma) \times Intv(n)))^*$ is a* snapshot word *if it has two pointers ρ_1, ρ_2 such that $\rho_1(\bar{\gamma}), \rho_2(\bar{\gamma})$ point to different elements of $\Gamma \times Intv(n)$ appearing in $\bar{\gamma}$. We denote the set of snapshot word by $sw(\mathcal{C}, \Gamma, n)$, and $\bar{\gamma}|_\Gamma$ is obtained by removing all elements in $\mathcal{C} \times Intv(n)$ from $\bar{\gamma}$.*

Example 6. From $\mathtt{digi}(\mathcal{X})$ in Example 5, by adding ρ_1 and ρ_2 (marked with *overline* and *underline*), which point to $(a, \mathtt{r}_3)$ and $(b, \mathtt{r}_{13})$, respectively, we have

$$\{(a, \mathtt{r}_7)\}\{(d, \mathtt{r}_9)\}\{(x_3, \mathtt{r}_5)\}\{(x_1, \mathtt{r}_1)\}\{\underline{(b, \mathtt{r}_{13})}\}\{(x_2, \mathtt{r}_7), \overline{(a, \mathtt{r}_3)}\}$$

and $\mathtt{digi}(\mathcal{X})|_\Gamma = \{(a, \mathtt{r}_7)\}\{(d, \mathtt{r}_9)\}\{\underline{(b, \mathtt{r}_{13})}\}\{\overline{(a, \mathtt{r}_3)}\}$.

Definition 11. *For snapshot words $\bar{\gamma} = X_1 \cdots X_m$ and $\bar{\gamma}' = Y_1 \cdots Y_n$ with $X_i, Y_j \in \mathcal{MP}((\mathcal{C} \cup \Gamma) \times Intv(n))$, we define the embedding $\bar{\gamma} \sqsubseteq \bar{\gamma}'$, if there exists a monotonic injection $f : [1..m] \to [1..n]$ such that*

– $X_k \subseteq Y_{f(k)}$ *for each* $k \in [1..m]$,
– $\rho_i(\bar{\gamma}) \in X_j$ *implies* $\rho_i(\bar{\gamma}') \in Y_{f(j)}$ *for* $i = 1, 2$ *and* $j \in [1..m]$, *and*
– $\rho_i(\bar{\gamma}) = \rho_i(\bar{\gamma}')$ *for* $i = 1, 2$.

Since Γ and $\mathcal{C}$ are finite, $\sqsubseteq$ is a WQO over $sw(\mathcal{C}, \Gamma, n)$ by Higman's lemma.

Definition 12. *Let* $c = (s, \nu, w)$ *be a configuration of a DTPDA with* $s \in S$, $w \in (\Gamma \times \mathbb{R}_{\geq 0})^*$, *and* $\nu : \mathcal{C} \to \mathbb{R}_{\geq 0}$, *and let* $\mathtt{mp}(w, \nu) = w \cup \{(x, \nu(x)) \mid x \in C\}$ *by regarding* w *as a multiset (i.e., ignore the ordering).* $snap(c)$ *is a snapshot word obtained by adding* ρ_1, ρ_2 *to* $\mathtt{digi}(\mathtt{mp}(w, \nu))$ *as:*

$$\begin{cases} \rho_1,\ \rho_2 \text{ are left undefined} & \text{if } w = \epsilon \\ \rho_1(snap(c)) = (\gamma, proj(t)),\ \rho_2 \text{ is left undefined} & \text{if } w = (\gamma, t) \\ \rho_1(snap(c)) = (\gamma, proj(t)),\ \rho_2(snap(c)) = \rho_1(snap((s, \nu, w'))) & \text{if } w = (\gamma, t)w' \end{cases}$$

Example 7. For c_2 in Example 1, $snap(c_1)$ is $\mathtt{digi}(\mathcal{X})$ (with ρ_1 and ρ_2) in Example 6. ρ_1 and ρ_2 point to the top and second stack frames $(a, 1.9), (b, 6.7)$.

Definition 13. *For a configuration* $c = (s, \nu, w)$ *of a DTPDA, a snapshot configuration* $Snap(c) = (s, \tilde{w})$ *with stack alphabet* $sw(\mathcal{C}, \Gamma, n)^*$ *is with*

$$\tilde{w} = snap(s, \nu, w[m])\ snap(s, \nu, w[m-1])\ \cdots\ snap(s, \nu, w[1])\ snap(s, \nu, \epsilon)$$

where $w = (a_m, t_m) \cdots (a_1, t_1) \in (\Gamma \times \mathbb{R}_{\geq 0})^*$ *and* $w[i] = (a_i, t_i) \cdots (a_1, t_1)$.

Example 8. For c_1 in Example 1 (with $\nu(x_1) = 0.5, \nu(x_2) = 3.9, \nu(x_3) = 2.3$), $Snap(c_1)$ is shown below. The top snapshot word in the stack summarizes a current time sequence of values of all clocks and ages.

$(a, 1.9)$		$\{(a, \mathtt{r}_7)\}\{(d, \mathtt{r}_9)\}\{(x_3, \mathtt{r}_5)\}\{(x_1, \mathtt{r}_1)\}\{\underline{(b, \mathtt{r}_{13})}\}\{(x_2, \mathtt{r}_7), \overline{(a, \mathtt{r}_3)}\}$
$(b, 6.7)$		$\{\underline{(a, \mathtt{r}_7)}\}\{(d, \mathtt{r}_9)\}\{(x_3, \mathtt{r}_5)\}\{(x_1, \mathtt{r}_1)\}\{\overline{(b, \mathtt{r}_{13})}\}\{(x_2, \mathtt{r}_7)\}$
$(a, 3.1)$	$\Rightarrow$	$\{\overline{(a, \mathtt{r}_7)}\}\{\underline{(d, \mathtt{r}_9)}\}\{(x_3, \mathtt{r}_5)\}\{(x_1, \mathtt{r}_1)\}\{(x_2, \mathtt{r}_7)\}$
$(d, 4.2)$		$\{\overline{(d, \mathtt{r}_9)}\}\{(x_3, \mathtt{r}_5)\}\{(x_1, \mathtt{r}_1)\}\{(x_2, \mathtt{r}_7)\}$
$\perp$		$\{(x_3, \mathtt{r}_5)\}\{(x_1, \mathtt{r}_1)\}\{(x_2, \mathtt{r}_7)\}$

Stack of c_1 Stack of $Snap(c_1)$

6.2 Operations on snapshot words

Definition 14. *Let* $\bar{\gamma} = X_1 \cdots X_m \in (\mathcal{MP}((\mathcal{C} \cup \Gamma) \times Intv(n)))^*$ *be a snapshot word and let* $\gamma \in \Gamma \cup \mathcal{C}$. *We define operations as follows.*

– **Insert** $\bar{\gamma}' = insert(\bar{\gamma}, (\delta, \mathtt{r}_k))$ *is obtained from* $\bar{\gamma}$ *by inserting* $(\delta, \mathtt{r}_k)$

$$\begin{cases} \text{either into } X_j, \text{ or between } X_j \text{ and } X_{j+1} \text{ for some } j \in [0..m] & \text{if } k \text{ is odd} \\ \text{into } X_1, \text{ if each } \mathtt{r}_i \text{ in } X_1 \text{ has an even index; before } X_1, \text{ o.w.} & \text{if } k \text{ is even} \end{cases}$$

and setting $\rho_1(\bar{\gamma}') = (\delta, \mathtt{r}_k)$ *and* $\rho_2(\bar{\gamma}') = \rho_1(\bar{\gamma})$.

- **Delete**$_\Gamma$ $\bar{\gamma}' = delete_\Gamma(\bar{\gamma})$ *is obtained from* $\bar{\gamma}$ *by deleting* $\rho_1(\bar{\gamma})$ *and setting* $\rho_1(\bar{\gamma}') = \rho_2(\bar{\gamma})$ *and* $\rho_2(\bar{\gamma}')$ *left* undefined.
- **Delete**$_\mathcal{C}$ *For* $x \in \mathcal{C}$, $delete_\mathcal{C}(\bar{\gamma}, x)$ *is obtained from* $\bar{\gamma}$ *by deleting* $(x, \mathtt{r})$ *(and* ρ_1, ρ_2 *are kept unchanged).*
- **Assignment** *For* $x \in \mathcal{C}$, $\mathtt{r} \in Intv(n)$, $assign(\bar{\gamma}, x, \mathtt{r}) = insert(delete_C(\bar{\gamma}, x), (x, \mathtt{r}))$.
- **Permutation** *Let* $i \in [1..m]$ *and* $0 \leq k \leq n$. *Permutation* $\sigma(\bar{\gamma})$ *is either* $\dot{\sigma}_{i,k}(\bar{\gamma})$ *or* $\ddot{\sigma}_{i,k}(\bar{\gamma})$, *defined by*

$$\begin{cases} \dot{\sigma}_{i,k}(\bar{\gamma}) = (X_i \dot{+} 2k+2)(X_{i+1} \dot{+} 2k+2)\cdots(X_m \dot{+} 2k+2)(X_1 \dot{+} 2k)\cdots(X_{i-1} \dot{+} 2k) \\ \ddot{\sigma}_{i,k}(\bar{\gamma}) = (X_i \ddot{+} 2k+2)(X_{i+1} \dot{+} 2k+2)\cdots(X_m \dot{+} 2k+2)(X_1 \dot{+} 2k)\cdots(X_{i-1} \dot{+} 2k) \end{cases}$$

  where, for $y \in \mathcal{C} \cup \Gamma$, $X_i \dot{+} j$ *updates each* $(y, \mathtt{r}_l) \in X_i$ *with* $(y, \mathtt{r}_{min(l+j,2n+1)})$ *if* l *is odd, and* $(y, \mathtt{r}_{min(l+j+1,2n+1)})$ *if* l *is even.* $X_i \ddot{+} j$ *updates each* $(y, \mathtt{r}_l) \in X_i$ *with* $(y, \mathtt{r}_{min(l+j,2n+1)})$ *if* $i = 1$ *and* l *is even; with* $(y, \mathtt{r}_{min(l+j-1,2n+1)})$, *otherwise.*
- **Propagate** $propagate(\bar{\gamma}, \bar{\gamma}')$ *is obtained from* $delete_\Gamma(\bar{\gamma})$ *by assigning* $\sigma(\rho_2(\bar{\gamma}'))$ *to* $\rho_2(delete_\Gamma(\bar{\gamma}))$ *for a permutation* σ *with* $\bar{\gamma}|_\Gamma = \sigma(\bar{\gamma}')|_\Gamma$.

Example 9. Consider $snap(c_i)$ in Example 7 for c_1 in Example 1.

$$\{(a,\mathtt{r}_7)\}\{(d,\mathtt{r}_9)\}\{(x_3,\mathtt{r}_5)\}\{(x_1,\mathtt{r}_1)\}\{\underline{(b,\mathtt{r}_{13})}\}\{(x_2,\mathtt{r}_7),\overline{(a,\mathtt{r}_3)}\}$$

- $insert(snap(c_1),(d,\mathtt{r}_5))$ has lots of choices, e.g.,
  $\{(a,\mathtt{r}_7)\}\{(d,\mathtt{r}_9)\}\{(x_3,\mathtt{r}_5)\}\{(x_1,\mathtt{r}_1),\overline{(d,\mathtt{r}_5)}\}\{(b,\mathtt{r}_{13})\}\{(x_2,\mathtt{r}_7),\underline{(a,\mathtt{r}_3)}\}$,
  $\{(a,\mathtt{r}_7)\}\{(d,\mathtt{r}_9)\}\{(x_3,\mathtt{r}_5)\}\{(x_1,\mathtt{r}_1)\},\{\overline{(d,\mathtt{r}_5)}\},\{(b,\mathtt{r}_{13})\}\{(x_2,\mathtt{r}_7),\underline{(a,\mathtt{r}_3)}\},\cdots$
  The transition from c_1 to c_2 in Example 1 is simulated by pushing the second one (say, $\bar{\gamma}_2$) to $Snap(c_1)$ in Example 8.
- For $c_2 \xrightarrow{2.6}_{Time} c_3$, the permutation $\dot{\sigma}_{4,2}(\bar{\gamma}_2)$ results in $\bar{\gamma}_3$ below.
  $\{(x_1,\mathtt{r}_7)\},\{\overline{(d,\mathtt{r}_{11})}\},\{(b,\mathtt{r}_{19})\}\{(x_2,\mathtt{r}_{13}),\underline{(a,\mathtt{r}_9)}\}\{(a,\mathtt{r}_{11})\}\{(d,\mathtt{r}_{13})\}\{(x_3,\mathtt{r}_9)\}$.
  If a timed transition is $c_2 \xrightarrow{2.5}_{Time} c_3$ (in time elapses 2.5 such that the fraction of $\nu(x_1)$ becomes 0), $\ddot{\sigma}_{4,2}(\bar{\gamma}_2)$ simulates it as
  $\{(x_1,\mathtt{r}_6)\},\{\overline{(d,\mathtt{r}_{11})}\},\{(b,\mathtt{r}_{19})\}\{(x_2,\mathtt{r}_{13}),\underline{(a,\mathtt{r}_9)}\}\{(a,\mathtt{r}_{11})\}\{(d,\mathtt{r}_{13})\}\{(x_3,\mathtt{r}_9)\}$.

Propagate is used with $delete_\Gamma$ to simulate a pop transition. Since time progress is recorded only at the top stack frame (including updates on clock values), after $delete_\Gamma$ is applied to the top stack frame, the second stack frame is replaced with the top. Lacking information is a pointer ρ_2, which is recovered from the second stack frame. This will be illustrated in Example 11.

7 Decidability of reachability of DTPDA

7.1 Well-formed projection on snapshot configurations

Let $\langle s, \bar{\gamma}_k \cdots \bar{\gamma}_2 \bar{\gamma}_1 \rangle$ be a snapshot configuration for $s \in S$ and $\bar{\gamma}_i \in (\mathcal{MP}((\mathcal{C} \cup \Gamma) \times Intv(n)))^*$ (regarding $\bar{\gamma}_k$ as a top stack symbol). A marking completion marks elements in $\Gamma \times Intv(n)$ that relate to pushdown transitions.

Definition 15. *For $\bar{\gamma}_k \cdots \bar{\gamma}_2\bar{\gamma}_1$ with $\bar{\gamma}_i \in (\mathcal{MP}((\mathcal{C} \cup \Gamma) \times Intv(n)))^*$, the* marking completion $\mathtt{comp}$ *inductively marks elements in $\bar{\gamma}_i|_\Gamma$ for each i.*

$$\begin{cases} \mathtt{comp}(\bar{\gamma}_1) & = \textit{add marking on } \rho_1(\bar{\gamma}_1) \\ \mathtt{comp}(\bar{\gamma}_k \cdots \bar{\gamma}_2\bar{\gamma}_1) & = \bar{\gamma}'_k \cdots \bar{\gamma}'_2\bar{\gamma}'_1 \end{cases}$$

where $\bar{\gamma}'_{k-1} \cdots \bar{\gamma}'_2\bar{\gamma}'_1 = \mathtt{comp}(\bar{\gamma}_{k-1} \cdots \bar{\gamma}_2\bar{\gamma}_1)$ and $\bar{\gamma}'_k$ is obtained from $\bar{\gamma}_k$ by marking

- *$\rho_1(\bar{\gamma}_k)$, and*
- *each element in $delete_\gamma(\bar{\gamma}_k)|_\Gamma$ corresponding to a marked element in $\bar{\gamma}'_{k-1}|_\Gamma$ by a permutation σ satisfying $\sigma(\bar{\gamma}_{k-1})|_\Gamma = delete_\gamma(\bar{\gamma}_k)|_\Gamma$.*

If such σ does not exist, $\mathtt{comp}(\bar{\gamma}_k \cdots \bar{\gamma}_2\bar{\gamma}_1) = \#$.

We define a *well-formed projection* $\Downarrow_\Upsilon (s, \bar{\gamma}_k \cdots \bar{\gamma}_2\bar{\gamma}_1)$ by removing all unmarked elements of $\Gamma \times Intv(n)$ in each $\bar{\gamma}_i$ in $(s, \mathtt{comp}(\bar{\gamma}_k \cdots \bar{\gamma}_2\bar{\gamma}_1))$, and left s as is. A snapshot configuration $(s, \bar{\gamma}_k \cdots \bar{\gamma}_2\bar{\gamma}_1)$ is *well-formed* if $\Downarrow_\Upsilon (s, \bar{\gamma}_k \cdots \bar{\gamma}_2\bar{\gamma}_1) = (s, \bar{\gamma}_k \cdots \bar{\gamma}_2\bar{\gamma}_1)$ (ignoring markings), and Υ is the set of well-formed snapshot configurations.

Example 10. In Example 8, $\bar{\gamma}_5$ is well-formed (i.e., $(a, \mathtt{r}_7), (d, \mathtt{r}_9), (b, \mathtt{r}_{13}), (b, \mathtt{r}_{13})$ are all marked). For instance, a marking on $(a, \mathtt{r}_7)$ succeeds the pointer ρ_1 of $\bar{\gamma}_3$.

7.2 Snapshot PDS

Definition 16. *Let $\langle S, s_{init}, \Gamma, \mathcal{C}, \Delta\rangle$ be a DTPDA and let n be the largest integer in Δ. A* snapshot PDS *is a PDS $\mathcal{S} = \langle S, sw(\mathcal{C}, \Gamma, n), \Delta\rangle$. We assume that its initial configuration is $\langle s_{init}, \{(x, \mathtt{r}_0) \mid x \in \mathcal{C}\}\rangle$.*

Transition rule to simulate timed transitions $\langle s, \bar{\gamma}\rangle \xrightarrow{t}_{\mathcal{S}} \langle s, \sigma(\bar{\gamma})\rangle$, *where σ is either $\dot{\sigma}_{i,m}$ or $\ddot{\sigma}_{i,m}$ with $m = floor(t)$ and $1 \le i \le length(\bar{\gamma})$*

Transition rules to simulate discrete transitions (s, op, s')

- **Local** $\langle s,\ \epsilon\rangle \xrightarrow{nop}_{\mathcal{S}} \langle s',\ \epsilon\rangle$,
- **Assignment** $\langle s,\ \bar{\gamma}\rangle \xrightarrow{x \leftarrow I}_{\mathcal{S}} \langle s',\ assign(\bar{\gamma}, x, \mathtt{r})\rangle$ *for* $\mathtt{r} \subseteq I$,
- **Test** $\langle s,\ \bar{\gamma}\rangle \xrightarrow{x \in I?}_{\mathcal{S}} \langle s',\ \bar{\gamma}\rangle$ *if* $\mathtt{r} \subseteq I$ *for* $(x, \mathtt{r})$ *in* $\bar{\gamma}$.
- **Push** $\langle s,\ \bar{\gamma}\rangle \xrightarrow{push(\gamma', I)}_{\mathcal{S}} \langle s',\ insert(\bar{\gamma}, (\gamma', \mathtt{r}))\ \bar{\gamma}\rangle$ *for* $\mathtt{r} \subseteq I$, *and*
- **Pop** $\langle s,\ \bar{\gamma}\ \bar{\gamma}'\rangle \xrightarrow{pop(\gamma', I)}_{\mathcal{S}} \langle s',\ propagate(delete_\Gamma(\bar{\gamma}), \bar{\gamma}')\rangle$.

By induction on the number of steps of transitions, complete and sound simulation between a DTPDA and a snapshot PDS is observed. Note that the initial clock valuation of a DTPDA to be set $\boldsymbol{\nu_0}$ is essential.

Lemma 4. *Let us denote c_0 and c (resp. $\langle s_{init}, \bar{\gamma}_\mathbf{0}\rangle$ and $\langle s, \tilde{w}\rangle$) for the initial configuration and a configuration of a DTPDA $\mathcal{T}$ (resp. its snapshot PDS $\mathcal{S}$).*

1. *If $c_0 \hookrightarrow^* c$ then there exists $\langle s, \tilde{w}\rangle$ such that $\langle s_{init}, \bar{\gamma}_\mathbf{0}\rangle \underset{\Upsilon\ \mathcal{S}}{\hookrightarrow^*} \langle s, \tilde{w}\rangle$, $s = state(c)$, and $\tilde{w}$ is well-formed.*

2. *If* $\langle s_{init}, \bar{\gamma}_{\mathbf{0}} \rangle \overset{*}{\underset{\Upsilon}{\hookrightarrow}}_{\mathcal{S}} \langle s, \tilde{w} \rangle$ *and* $\tilde{w}$ *is well-formed. there exists* c *such that* $c_0 \hookrightarrow^* c$, $s = state(c)$, *and* $Snap(c) \hookleftarrow \tilde{w}$.

Example 11. We show how a snapshot PDS simulates a DTPDA in Example 1, as continuation to Example 9 (which shows transitions from c_1 to c_3).

- $c_3 \xrightarrow{x_2 \leftarrow (2,5]}_{Disc} c_4$ is simulated by $assign(delete_{\mathcal{C}}(snap(c_3), x_2), x_2, \mathtt{r}_7)$ at the top stack frame, since $\nu(x_2) = 3.8 \in \mathtt{r}_7$. There are several choices of $assign(delete_{\mathcal{C}}(snap(c_3), x_2), x_2, \mathtt{r}_7)$. Among them, $\{(x_1, \mathtt{r}_7)\}, \{\overline{(d, \mathtt{r}_{11})}\}, \{(b, \mathtt{r}_{19})\}\{\underline{(a, \mathtt{r}_9)}\}\{(a, \mathtt{r}_{11})\}\{(x_2, \mathtt{r}_7), (d, \mathtt{r}_{13})\}\{(x_3, \mathtt{r}_9)\}$. corresponds to 3.8. A different value, e.g., $\nu(x_2) = 3.3$, corresponds to $\{(x_1, \mathtt{r}_7)\}, \{\overline{(d, \mathtt{r}_{11})}\}, \{(x_2, \mathtt{r}_7), (b, \mathtt{r}_{19})\}\{\underline{(a, \mathtt{r}_9)}\}\{(a, \mathtt{r}_{11})\}\{(d, \mathtt{r}_{13})\}\{(x_3, \mathtt{r}_9)\}$.
- $c_4 \xrightarrow{pop(d,[4,6])}_{Disc} c_5$ is simulated by $propagate(delete_{\Gamma}(snap(c_4)), snap(c_1))$. Note that a snapshot PDS does not change anything except for the top stack frame. Thus, the second stack frame is kept unchanged from $snap(c_1)$. First, $delete_{\Gamma}$ removes the element pointed by ρ_1, which results in
  $\{(x_1, \mathtt{r}_7)\}, \{(b, \mathtt{r}_{19})\}\{\underline{(a, \mathtt{r}_9)}\}\{(a, \mathtt{r}_{11})\}\{(x_2, \mathtt{r}_7), (d, \mathtt{r}_{13})\}\{(x_3, \mathtt{r}_9)\}$.
  $snap(c_1) = \{(a, \mathtt{r}_7)\}\{(d, \mathtt{r}_9)\}\{(x_3, \mathtt{r}_5)\}\{(x_1, \mathtt{r}_1)\}\{\underline{(b, \mathtt{r}_{13})}\}\{(x_2, \mathtt{r}_7), \overline{(a, \mathtt{r}_3)}\}$ and, by pattern matching between ρ_2 in the former and ρ_1 in the latter, $\dot{\sigma}_{4,2}$ (which is used in the timed transition from c_2 to c_3 in Example 9) is found. Then ρ_1 is updated with the current ρ_2 and ρ_2 is recovered by σ as
  $\{(x_1, \mathtt{r}_7)\}, \{\underline{(b, \mathtt{r}_{19})}\}\{\overline{(a, \mathtt{r}_9)}\}\{(a, \mathtt{r}_{11})\}\{(x_2, \mathtt{r}_7), (d, \mathtt{r}_{13})\}\{(x_3, \mathtt{r}_9)\}$.

It is not difficult to see that $\Downarrow_{\Upsilon}$ satisfies Definition 7. A snapshot PDS has finite states and WQO stack alphabet. By applying the encoding in Remark 1, we obtain our main result from Theorem 3, 5, Lemma 2, and 4.

Corollary 1. *The (configuration) reachability of a DTPDA is decidable.*

7.3 Comparison among discretizations

In [13], we apply slight extensions of a DTPDA to make it able to set the value of an age to that of a clock when a push occurs, and set the value of a clock to that of an age when a pop occurs. They are easily encoded into snapshot words.

- **Push-set** $push(\gamma, x)$, push γ on a stack associated with a local age of the value of a clock $x \in \mathcal{C}$, and
- **Pop-set** $pop(\gamma, x)$, pop γ on a stack and set the value of a clock $x \in \mathcal{C}$ to the value of the associated age a.

When we consider extensions of DTPDA [1] with such operations, we see the difference between the original discretization [1] and ours as a snapshot PDS. Note that our snapshot word encoding summarizes the ordering of fractions of all local ages and global clocks in the stack, whereas the encoding in [1] summarizes boundedly many information, i.e., global clocks and an age in the top stack frame and copies of global clocks in the next stack frame.

Example 12. The encoding in [1] does not contain $x^\bullet$ for $x \in \mathcal{C}$, which represents the position of the value of a clock x in the previous stack frame. Our encoding of a DTPDA as a snapshot word PDS is quite equivalent to an extension of that in [1] with $x^\bullet$ for $c \in \mathcal{C}$. With and without $x^\bullet$ are different when we consider an extension of DTPDA, e.g., that with

- **Compare** $compare_\sim(x)$ for a clock $x \in \mathcal{C}$ and $\sim \in \{\geq, >, \leq, <, =\}$, which compares values between an age in the top stack frame and a clock x by $\sim$.

  $compare_\sim(x)$ is a quite strong operator. It enables us to define

- **Push-set** $push(\gamma, x)$, push γ on a stack associated with a local age of the value of a clock $c \in \mathcal{C}$, and
- **Push-set**$^+$ $push^+(\gamma, x)$, push γ on a stack associated with a local age whose value is between the value of c and its ceiling value.
- **Push-set**$^-$ $push^+(\gamma, x)$, push γ on a stack associated with a local age whose value is between the value of c and its floor value.

Similar for **Pop-set**. With a fresh clock y prepared as a stop watch, we can encode these operations with $compare_\sim(x)$ as follows.

- **Push-set** $push(\gamma, x)$ is encoded as
  $y \leftarrow [0,0]; push(\gamma, [0, \infty)); compare_=(x); y \in [0,0]?;$.
- **Push-set**$^+$ $push^+(\gamma, x)$ is encoded as
  $y \leftarrow [0,0]; x \in (j, j+1)?; push(\gamma, (j, j+1)); compare_>(x); y \in [0,0]?;$.
- **Push-set**$^-$ $push^+(\gamma, x)$ is encoded as
  $y \leftarrow [0,0]; x \in (j, j+1)?; push(\gamma, (j, j+1)); compare_<(x); y \in [0,0]?;$.

Note that these operations enables us to prepare an operation that compares values of ages in different stack frames. For instance, a sequence

$$push^+(\gamma, x); (\mathbf{push})^*; push^-(\gamma', x); (x \leftarrow [0, \infty)); pop(\gamma', y); (\mathbf{pop})^*; compare_<(y);$$

compares ages in the different stack frames containing γ and γ'.

Note that the original encoding in [1] cannot decide $compare_<(y)$. With $x^\bullet$, it can correctly decide that $compare_<(y)$ interrupts transitions.

8 Conclusion

This paper investigated a general framework of pushdown systems with well-quasi-ordered control states and stack alphabet, *well-structured pushdown systems*, to show decidability of the reachability. This extends the decidability results on a pushdown system with finite control states and well-quasi-ordered stack alphabet [6]. The ideas behind are,

- combining WSTS [2, 11] and classical Pre^*-automaton technique [5, 12, 10], which enables us to reduce arguments on stacks to on stack symbols, and
- introduction of a well-formed projection $\Downarrow_\Upsilon$, which extracts the shape of reachable configurations.

As an instance, an alternative decidability proof of the reachability for dense-timed pushdown system [1] was shown. Note that the original encoding [1] cannot handle the extension with $compare_{\sim}(x)$ (which compares values of a local age in the top stack frame and a clock x). The encoding is inspired by the digitization techniques in [15].

Acknowledgements

The authors would like to thank Shoji Yuen, Yasuhiko Minamide, Tachio Terauchi, and Guoqiang Li for valuable comments and discussions. This work is supported by the NSFC-JSPS bilateral joint research project (61011140074), NSFC projects (61003013,61100052,61033002), NSFC-ANR joint project (61261130589), and JSPS KAKENHI Grant-in-Aid for Scientific Research(B) (23300008).

References

1. P.A. Abdulla, M.F. Atig, and F. Stenman. Dense-Timed Pushdown Automata. *IEEE LICS 2012*, pages 35–44, 2012.
2. P.A. Abdulla, K. Cerans, C. Jonsson, and T. Yih-Kuen. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160(1–2):109–127, 2000.
3. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
4. A. Bouajjani, R. Echahed, and R. Robbana On the Automatic Verification of Systems with Continuous Variables and Unbounded Discrete Data Structures. *Hybrid Systems II*, *LNCS 999*, pages 64–85, 1995.
5. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. *CONCUR 1997*, *LNCS 1243*, pages 135–150, 1997.
6. X. Cai and M. Ogawa. Well-Structured Pushdown Systems. CONCUR 2013, LNCS 8052 (2013), 121–136. Long version: JAIST Research Report IS-RR-2013-001.
7. R. Chadha and M. Viswanathan. Decidability results for well-structured transition systems with auxiliary storage. *CONCUR 2007*, *LNCS 4703*, pages 136–150, 2007.
8. Z. Dang. Pushdown timed automata:a binary reachability characterization and safety verification. *Theoretical Computer Science*, 302:93–121, 2003.
9. M. Emmi and R. Majumdar. Decision Problems for the Verification of Real-Time Software. *HSCC'06*, *LNCS 3927*, pages 200–211, 2006.
10. J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. *CAV 2000*, *LNCS 1855*, pages 232–247, 2000.
11. A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.
12. A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems (extended abstract). *INFINITY'97*, *ENTCS 9*. 1997.
13. G. Li, X. Cai, M. Ogawa, and S. Yuen. Nested Timed Automata. *FORMATS 2013*, *LNCS 8503*, pages 168–182, 2013.
14. R. Mayr. Process rewrite systems. *Information and Computation*, 156:264–286, 1999.
15. J. Ouaknine and J. Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. *IEEE LICS 2004*, pages 54–63, 2004.

Kleene Algebra with Tests and Coq Tools for While Programs

Damien Pous

CNRS – LIP, ENS Lyon, UMR 5668

Abstract We present a Coq library about Kleene algebra with tests, including a proof of their completeness over the appropriate notion of languages, a decision procedure for their equational theory, and tools for exploiting hypotheses of a certain kind in such a theory.
Kleene algebra with tests make it possible to represent if-then-else statements and while loops in most imperative programming languages. They were actually introduced by Kozen as an alternative to propositional Hoare logic.
We show how to exploit the corresponding Coq tools in the context of program verification by proving equivalences of while programs, correctness of some standard compiler optimisations, Hoare rules for partial correctness, and a particularly challenging equivalence of flowchart schemes.

Introduction

Kleene algebra with tests (KAT) have been introduced by Kozen [19], as an equational system for program verification. A Kleene algebra with tests is a Kleene algebra (KA) with an embedded Boolean algebra of tests. The Kleene algebra component deals with the control-flow graph of the programs—sequential composition, iteration, and branching—while the Boolean algebra component deals with the conditions appearing in if-then-else statements, while loops, or pre- and post-assertions.

This formalism is both concise and expressive, which allowed Kozen and others to give detailed paper proofs about various problems in program verification (see, e.g., [3, 19, 21, 23]). More importantly, the equational theory of KAT is decidable and complete over relational models [24], and hypotheses of a certain kind can moreover be eliminated [11, 15]. This suggests that a proof using KAT should not be done manually, but with the help of a computer. The goal of the present work is to give this possibility, inside the Coq proof assistant.

The underlying decision procedure cannot be formulated, a priori, as a simple rewriting system: it involves automata algorithms, it cannot be defined in Ltac, at the meta-level, and it does not produce a certificate which could easily be checked in Coq, a posteriori. This leaves us with only one possibility: defining a reflexive tactic [1, 8, 14]. Doing so is quite challenging: we basically have to prove completeness of KAT axioms w.r.t. the model of guarded string languages (the

natural generalisation of languages for KA, to KAT), and to provide a provably correct algorithm for language equivalence of KAT expressions.

The completeness theorem is far from trivial; we actually have to formalise a lot of preliminary material: finite sums, finite sets, unique decomposition of Boolean expressions into sums of atoms, regular expression derivatives, expansion theorem for regular expressions, matrices, automata... As a consequence, we only give here a high-level overview of the involved mathematics, leaving aside standard definitions, technical details, or secondary formalisation tricks. The interested reader can consult the library, which is documented [30].

Outline. We first present KAT and its models (§1). We then sketch the completeness proof (§2), the decision procedure (§3), and the method used to eliminate hypotheses (§4). We finally illustrate the benefits of our tactics on several case-studies (§5), before discussing related works (§6), and concluding (§7).

1 Kleene Algebra with Tests

A Kleene algebra with tests consists of:

- a Kleene algebra $\langle X, \cdot, +, ^\star, 1, 0\rangle$ [18], i.e., an idempotent semiring with a unary operation, called "Kleene star", satisfying an axiom: $1 + x \cdot x^\star \leq x^\star$ and two inference rules: $y \cdot x \leq x$ entails $y^\star \cdot x \leq x$ and the symmetric one. (The preorder $(\leq)$ being defined by $x \leq y \triangleq x + y = y$.)
- a Boolean algebra $\langle B, \wedge, \vee, \neg, \top, \bot\rangle$;
- a homomorphism from $\langle B, \wedge, \vee, \top, \bot\rangle$ to $\langle X, \cdot, +, 1, 0\rangle$, that is, a function $[\cdot] : B \to X$ such that $[a \wedge b] = [a] \cdot [b]$, $[a \vee b] = [a] + [b]$, $[\top] = 1$, and $[\bot] = 0$.

The elements of the set B are called "tests"; we denote them by a, b. The elements of X, called "Kleene elements", are denoted by x, y, z. We usually omit the operator "$\cdot$" from expressions, writing xy for $x \cdot y$. The following (in)equations illustrate the kind of laws that hold in all Kleene algebra with tests:

$$[a \vee \neg a] = 1 \qquad [a \wedge (\neg a \vee b)] = [a][b] = [\neg(\neg a \vee \neg b)]$$

$$x^\star x^\star = x^\star \qquad (x + y)^\star = x^\star (y x^\star)^\star \qquad (x + xxy)^\star \leq (x + xy)^\star$$

$$[a]([\neg a]x)^\star = [a] \qquad [a]([a]x[\neg a] + [\neg a]y[a])^\star [a] \leq (xy)^\star$$

The laws from the first line come from the Boolean algebra structure, while the ones from the second line come from the Kleene algebra structure. The two laws from the last line are more interesting: their proof must mix both Boolean algebra and Kleene algebra reasoning. They are left to the reader as a non-trivial exercice; the tools we present in this paper allow one to prove them automatically.

1.1 The model of binary relations

Binary relations form a Kleene algebra with tests; this is the main model we are interested in, in practice. The Kleene elements are the binary relations over a

given set S, the tests are the predicates over this set, and the star of a relation is its reflexive transitive closure:

$$\begin{array}{rlr} X = \mathcal{P}(S \times S) & & B = \mathcal{P}(S) \\ x \cdot y = \{(p,q) \mid \exists r, (p,r) \in x \wedge (r,q) \in y\} & & a \wedge b = a \cap b \\ x + y = x \cup y & & a \vee b = a \cup b \\ x^\star = \{(p_0,p_n) \mid \exists p_1 \ldots p_{n-1}, \forall i < n, (p_i,p_{i+1}) \in x\} & & \neg a = S \setminus a \\ 1 = \{(p,p) \mid p \in S\} & & \top = S \\ 0 = \emptyset & [a] = \{(p,p) \mid p \in a\} & \bot = \emptyset \end{array}$$

The laws of a Kleene algebra are easily proved for these operations; note however that one needs either to restrict to decidable predicates (i.e., to take `S → bool` or {`p`: `S → Prop` | `forall p, S p ∨ ¬S p`} for B), or to assume the law of excluded middle: B must be a Boolean algebra, so that negation has to be an involution. This choice for B is left to the user of the library.

This relational model is typically used to interpret imperative programs: such programs are state transformers, i.e., binary relations between states, and the conditions appearing in these programs are just predicates on states. These conditions are usually decidable, so that the above constraint is actually natural.

The equational theory of Kleene algebra with tests is complete over the relational model [24]: any equation $x = y$ that holds universally in this model can be proved from the axioms of KAT. We do not need to formalise this theorem, but it is quite informative in practice: by contrapositive, if an equation cannot be proved from KAT, then it cannot be universally true on binary relations, meaning that proving its validity for a particular instantiation of the variables necessarily requires one to exploit additional properties of this particular instance.

1.2 Other models

We describe two other models in the sequel: the syntactic model (§1.3) and the model of guarded string languages (§1.4); these models have to be formalised to build the reflexive tactic we aim at.

There are other important models of KAT. First of all, any Kleene algebra can be extended into a Kleene algebra with tests by embedding the two-element Boolean lattice. We also have traces models (where one keeps track of the whole execution traces of the programs rather than just their starting and ending points), matrices over a Kleene algebra with tests, but also models inherited from semirings like min-plus and max-plus algebra. The latter models have a degenerate Kleene star operation; they become useful when one constructs matrices over them, for instance to study shortest path algorithms.

Also note that like for Kleene algebra [9, 20, 29], KAT admits a natural "typed" generalisation, allowing for instance to encompass heterogeneous binary relations and rectangular matrices. Our Coq library is actually based on this generalisation, and this deeply impacts the whole infrastructure; we however omit the corresponding details and technicalities here, for the sake of clarity.

1.3 KAT expressions

Let p, q range over a set Σ of *letters* (or *actions*), and let $a_1, \ldots, a_n$ be the elements of a finite set Θ of *primitive tests*. *Boolean expressions* and *KAT expressions* are defined by the following syntax:

$$a, b ::= a_i \in \Theta \mid a \wedge a \mid a \vee a \mid \neg a \mid \top \mid \bot \qquad \text{(Boolean expressions)}$$
$$x, y ::= p \in \Sigma \mid [a] \mid x \cdot y \mid x + y \mid x^\star \mid 1 \mid 0 \ . \qquad \text{(KAT expressions)}$$

Given a Kleene algebra with tests $\mathcal{K} = \langle X, B, [\cdot] \rangle$, any pair of maps $\theta : \Theta \to B$ and $\sigma : \Sigma \to X$ gives rise to a KAT homomorphism allowing to interpret expressions in $\mathcal{K}$. Given two such expressions x and y, the equation $x = y$ is a *KAT theorem*, written $\text{KAT} \vdash x = y$, when the equation holds in any Kleene algebra with tests, under any interpretation. One checks easily that KAT expressions quotiented by the latter relation form a Kleene algebra with tests; this is the free Kleene algebra with tests over Σ and Θ. (We actually use this impredicative encoding of KAT derivability in the Coq library.)

1.4 Guarded strings languages

Guarded string languages are the natural generalisation of string languages for Kleene algebra with tests. We briefly define them.

An *atom* is a function from elementary tests (Θ) to Booleans; it indicates which of these tests are satisfied. We let α, β range over atoms, the set of which is denoted by At. (Technically, we represent elementary tests as finite ordinals of a given size n ($\Theta = \texttt{ord}\ n$), and we encode atoms as ordinals ($\texttt{At} = \texttt{ord}\ 2^n$). This allows us to avoid functional extensionality problems.) We let u, v range over *guarded strings*: alternating sequences of atoms and letters, which both start and end with an atom:

$$\alpha_1, p_1, \ldots, \alpha_n, p_n, \alpha_{n+1} \ .$$

The concatenation $u * v$ of two guarded strings u, v is a partial operation: it is defined only if the last atom of u is equal to the first atom of v; it consists in concatenating the two sequences and removing one copy of the shared atom in the middle.

The Kleene algebra with tests of guarded string languages is obtained by considering sets of guarded strings for X and sets of atoms for B:

$$\begin{aligned}
X &= \mathcal{P}\left((At \times \Sigma)^\star \times At\right) & B &= \mathcal{P}\left(At\right) \\
x \cdot y &= \{u * v \mid u \in x \wedge v \in y\} & a \wedge b &= a \cap b \\
x + y &= x \cup y & a \vee b &= a \cup b \\
x^\star &= \{u_1 * \cdots * u_n \mid \exists u_1 \ldots u_n, \forall i \leq n, u_i \in x\} & \neg a &= At \setminus a \\
1 &= \{\alpha \mid \alpha \in At\} & \top &= At \\
0 &= \emptyset \qquad\qquad [a] = \{\alpha \mid \alpha \in a\} & \bot &= \emptyset
\end{aligned}$$

Note that we slightly abuse notation by letting α denote either an atom, or a guarded string reduced to an atom. Also note that the set $B = \mathcal{P}\left(At\right)$ has to be represented by the Coq type $\texttt{At} \to \texttt{bool}$, to get a Boolean algebra on it.

2 Completeness

Let G be the unique homomorphism from KAT expressions to guarded string languages such that

$$G(a_i) = \{\alpha \mid \alpha(a_i) \text{ is true}\} \qquad G(p) = \{\alpha p \beta \mid \alpha, \beta \in At\}$$

Completeness of KAT over guarded string languages can be stated as follows.

Theorem 1. *For all KAT expressions x, y, $G(x) = G(y)$ entails* KAT $\vdash x = y$.

This theorem is central to our development: it allows us to prove (in)equations in arbitrary models of KAT, by resorting to an algorithm deciding guarded string language equivalence (to be described in §3).

We closely follow Kozen and Smith' proof [24]. This proof relies on the completeness of Kleene algebra over languages, which we thus need to prove first.

2.1 Completeness of Kleene algebra axioms

Let R be the Kleene algebra homomorphism from regular expressions to (plain) string languages mapping a letter p to the language consisting of the single-letter word p. KA completeness over languages can be stated as follows [18]:

Theorem 2. *For all regular expressions x, y, $R(x) = R(y)$ entails* KA $\vdash x = y$.

(Like for KAT, the judgement KA $\vdash x = y$ means that $x = y$ holds in any Kleene algebra, under any interpretation.) We already presented a Coq formalisation of this theorem [9], but our development was over-complicated. We re-proved it from scratch here, following a simpler path which we now describe.

The main idea of Kozen's proof consists in replaying automata algorithms algebraically, using matrices to encode automata. The key insight that allowed us to considerably simplify the corresponding formalisation is that the algorithm used for this proof need not be the same as the one to be executed by the reflexive tactic we eventually define. Indeed, we can take the simplest possible algorithm to prove KA completeness, ignoring all complexity aspects, thus allowing us to focus on conciseness and mathematical simplicity. In contrast, the algorithm to be executed by the final reflexive tactic should be relatively efficient, but we do not need to prove it complete, nor to replay its correctness algebraically: we only need to prove its correctness w.r.t. languages, which is much easier.

A preliminary step for the proof consists in proving that matrices over a Kleene algebra form a Kleene algebra. The Kleene star for matrices is non-trivial to define and to prove correct, but this can be done with a reasonable amount of efforts once appropriate lemmas and tools for block matrices have been set up.

A finite automaton can then be represented using three matrices (u, M, v) over regular expressions, where u is a $(1, n)$-matrix, M is a (n, n)-matrix, and v is a $(n, 1)$-matrix, n being the number of states of the automaton. Such a "matricial automaton" can be evaluated into a regular expression by taking the

product $u \cdot M^\star \cdot v$, which is a scalar. The various classes of automata can be recovered by imposing conditions on the coefficients of the three matrices. For instance, a non-deterministic finite automaton (NFA) is such that u and v are 01-vectors and the coefficients of M are sums of letters.

Given a regular expression x, we construct a deterministic finite automaton (DFA) (u, M, v) such that $\mathrm{KA} \vdash x = uM^\star v$, as follows.

1. First construct a NFA with epsilon transitions (u'', M'', v''), such that $\mathrm{KA} \vdash x = u''M''^\star v''$. This is easily done by induction on x, using Thompson construction [31] (which is compositional, unlike the construction we used in [9]).
2. Remove epsilon transitions to obtain a NFA (u', M', v') such that $\mathrm{KA} \vdash u''M''^\star v'' = u'M'^\star v'$. We do it purely algebraically, in one line. In particular the transitive closure of epsilon transitions is computed using Kleene star on matrices. (Unlike in [9] we do not need a dedicated algorithm for this.)
3. Use the powerset construction to convert this NFA into a DFA (u, M, v) such that $\mathrm{KA} \vdash u'M'^\star v' = uM^\star v$. Again, this is done algebraically, and we do not need to perform the standard 'accessible subsets' optimisation.

We can prove that for any DFA (u, M, v), $R(uM^\star v)$ is the language recognised by the DFA. Therefore, to obtain Theorem 2, it suffices to prove that if two DFA (u, M, v) and (s, N, t) recognise the same language, then $\mathrm{KA} \vdash uM^\star v = sN^\star t$. For this last step, it suffices to exhibit a Boolean matrix that relates exactly those states of the two DFA that recognise the same language. We need for that an algorithm to check language equivalence of DFA states; we reduce the problem to DFA emptiness, and we perform a simple reachability analysis.

All in all, the KA completeness proof itself only requires 124 lines of specifications, and 119 lines of proofs (according to `coqwc`).

2.2 Completeness of KAT axioms

To obtain KAT completeness (Theorem 1), Kozen and Smith [24] define a function $\widehat{\cdot}$ on KAT expressions that expands the expressions in such a way that we have $\mathrm{KAT} \vdash x = y$ iff $\mathrm{KA} \vdash \widehat{x} = \widehat{y}$. While this function can be thought as a reduction of KAT to KA, it cannot be used in practice: it produces expressions that are almost systematically exponentially larger than the given ones. It is however sufficient to establish completeness; as explained earlier, we defer actual computations to a completely different algorithm (§3).

More precisely, the function $\widehat{\cdot}$ is defined in such a way that we have:

$$\mathrm{KAT} \vdash \widehat{x} = x \tag{i}$$

$$G(\widehat{x}) = R(\widehat{x}) \tag{ii}$$

We deduce KAT completeness as follows:

$$
\begin{aligned}
& G(x) = G(y) && \\
\Leftrightarrow\ & G(\widehat{x}) = G(\widehat{y}) && (G \text{ is a KAT morphism, and (i)}) \\
\Leftrightarrow\ & R(\widehat{x}) = R(\widehat{y}) && (\text{by (ii)}) \\
\Rightarrow\ & \mathrm{KA} \vdash \widehat{x} = \widehat{y} && (\text{KA completeness}) \\
\Rightarrow\ & \mathrm{KAT} \vdash \widehat{x} = \widehat{y} && (\text{any KAT is a KA}) \\
\Leftrightarrow\ & \mathrm{KAT} \vdash x = y && (\text{by (i)})
\end{aligned}
$$

(Note that the last equation entails the first one, so that all these statements are in fact equivalent.)

The function $\widehat{\cdot}$ is defined recursively over KAT expressions, using an intermediate datastructure: formal sums of *externally guarded terms* (i.e., either an atom, or a product of the form $\alpha x \beta$). The case of a starred expression $x^\star$ is quite involved: $\widehat{x^\star}$ is defined by an internal recursion on the length of the formal sum corresponding to $\widehat{x}$. The proof of the first equation (i) is not too difficult to formalise, using appropriate tools for finite sums (i.e., a simplified form of big operators [7], which we actually use a lot in the whole development). The second one (ii) is more cumbersome, notably because we must deal with the two implicit coercions appearing in its statement: formally, it has to be stated as follows:

$$i(G(\widehat{x})) = R(j(\widehat{x})) \ ,$$

where i takes a guarded string language and returns a finite word language on the alphabet $\Sigma \uplus \Theta \uplus \overline{\Theta}$, and j takes a KAT expression and returns a regular expression over this extended alphabet, by pushing all negations to the leaves.

Apart from the properties of these coercion functions, the proof of (ii) mainly consists in rather technical arguments about regular and guarded string languages concatenation. All in all, once KA completeness has been proved, KAT completeness requires us 278 lines of specifications, and 360 lines of proofs.

3 Decision procedure

To check whether two expressions denote the same language of guarded strings, we use an algorithm based on a notion of *partial derivatives* for KAT expressions. Derivatives were introduced by Brzozowski [10] for regular expressions; they make it possible to define a deterministic automaton where the states of the automaton are the regular expressions themselves.

Derivatives can be extended to KAT expressions in a very natural way [22]: we first define a Boolean function ϵ_α, that indicates whether an expression accepts the single atom α; this function is then used to define the derivation function $\delta_{\alpha,p}$, that intuitively returns what remains of the given expression after reading the atom α and the letter p. These two functions make it possible to give a

$$\delta'_{\alpha,p}(x+y) = \delta'_{\alpha,p}(x) \cup \delta'_{\alpha,p}(y)$$
$$\delta'_{\alpha,p}(xy) = \begin{cases} \delta'_{\alpha,p}(x)y \cup \delta'_{\alpha,p}(y) & \text{if } \epsilon_\alpha(x) \\ \delta'_{\alpha,p}(x)y & \text{otherwise} \end{cases}$$
$$\delta'_{\alpha,p}(x^\star) = \delta'_{\alpha,p}(x)x^\star$$
$$\delta'_{\alpha,p}(q) = \begin{cases} \{1\} & \text{if } p = q \\ \emptyset & \text{otherwise} \end{cases}$$
$$\delta'_{\alpha,p}([a]) = \emptyset$$

Figure1. Partial derivatives for KAT expressions

coalgebraic characterisation of the function G, which underpins the correctness of the algorithm we sketch below:

$$G(x)(\alpha) = \epsilon_\alpha(x) \qquad G(x)(\alpha\, p\, u) = G(\delta_{\alpha,p}(x))(u) \ .$$

Like with standard regular expressions, the set of derivatives of a given KAT expression (i.e., the set of expressions that can be obtained by repeatedly deriving w.r.t. arbitrary atoms and letters) can be infinite. To recover finiteness, we switch to *partial* derivatives [4]. Their generalisation to KAT should be folklore; we define them in Fig. 1. We use the notation Xy to denote the set $\{xy \mid x \in X\}$ when X is a set of expressions and y is an expression. The partial derivation function $\delta'_{\alpha,p}$ returns a (finite) set of expressions rather than a single one; this corresponds to the fact that we build a non-deterministic automaton. Still abusing notations, by letting a set of expressions denote the sum of its elements, we prove that $\mathrm{KAT} \vdash \delta_{\alpha,p}(x) = \delta'_{\alpha,p}(x)$.

Now call *bisimulation* any relation R between sets of expressions such that whenever $X\ R\ Y$, we have

- $\epsilon(X) = \epsilon(Y)$ and
- $\forall \alpha \in At, \forall p \in \Sigma,\ \delta'_{\alpha,p}(X)\ R\ \delta'_{\alpha,p}(Y)$.

We show that if there is a bisimulation R such that $X\ R\ Y$, then $G(X) = G(Y)$ (the converse also holds). This gives us an algorithm to decide language equivalence of two KAT expressions x, y: it suffices to try to construct a bisimulation that relates the singletons $\{x\}$ and $\{y\}$. This algorithm terminates because the set of partial derivatives reachable from a pair of expressions is finite (we do not need to formalise this fact since we just need the correctness of this algorithm).

There is a lot of room for optimisation in our implementation—for instance, we use unordered lists to represent binary relations. An important point in our design is that such optimisations can be introduced and proved correct independently from the completeness proof for KAT, which gives us much more flexibility than in our previous work on Kleene algebra [9].

3.1 Building a reflexive tactic

Using standard methodology [1, 8, 14], we finally pack the previous ingredients into a Coq reflexive tactic called `kat`, allowing us to close automatically any goal which belongs to the equational theory of KAT.

The tactic works on any model of KAT: those already declared in the library (relations, languages, matrices, traces), but also the ones declared by the user. The reification code is written in OCaml; it is quite complicated for at least two reasons: KAT is a two-sorted structure, and we actually deal with "typed" KAT, as explained in §1.2, which requires us to work with a dependently typed syntax.

For the sake of simplicity, the Coq algorithm we implemented for KAT does not produce a counter-example in case of failure. To be able to give such a counter-example to the user, we actually run an OCaml copy of the algorithm first (extracted from Coq, and modified by hand to produce counter-examples). This has two advantages: the tactic is faster in case of failure, and the counter-example—a guarded string—can be pretty-printed in a nicer way.

4 Eliminating hypotheses

The above `kat` tactic works for the equational theory of KAT, i.e., the (in)equations that hold in any model of KAT, under any interpretation. In particular, this tactic does not make use of any hypothesis which is specific to the model or to the interpretation. Some hypotheses can however be exploited [11, 15]: those having one of the following shapes.

(i) $x = 0$;
(ii) $[a]x = x[b]$, $[a]x \leq x[b]$, or $x[b] \leq [a]x$;
(iii) $x \leq [a]x$ or $x \leq x[a]$
(iv) $a = b$ or $a \leq b$;
(v) $[a]p = [a]$ or $p[a] = [a]$, for atomic p ($p \in \Sigma$);

Equations of the first kind (i) are called "Hoare" equations, for reasons to become apparent in §5.2. They can be eliminated using the following implication:

$$\begin{cases} x + uzu = y + uzu \\ z = 0 \end{cases} \quad \text{entails} \quad x = y \ . \tag{$\dagger$}$$

This implication is valid for any term u, and the method is complete [15] when u is taken to be the universal KAT expression, $\Sigma^\star$. Intuitively, for this choice of u, uzu recognizes all guarded strings that contain a guarded string of z as a substring. Therefore, when checking that $x + uzu = y + uzu$ are language equivalent rather than $x = y$, we rule out all counter-examples to $x = y$ that contain a substring belonging to z: such counter-examples are irrelevant since z is known to be empty.

Equations of the shape (iii) and (iv) are actually special cases of those of the shape (ii), which are in turn equivalent to Hoare equations. For instance, we have $[a]x \leq x[b]$ iff $[a]x[\neg b] = 0$. Moreover, two hypotheses of shape (i) can be merged into a single one using the fact that $x = 0 \wedge y = 0$ iff $x + y = 0$. Therefore, we can aggregate all hypotheses of shape (i-iv) into a single one (of shape (i)), and use the above technique just once.

Hypotheses of shape (v) are handled differently, using the following equivalence:

$$[a]p = [a] \quad \text{iff} \quad p = [\neg a]p + [a] \ , \tag{‡}$$

This equivalence allows us to substitute $[\neg a]p + [a]$ for p in the considered goal—whence the need for p to be atomic. Again, the method is complete [15], i.e.,

$$\text{KAT} \vdash ([a]p = [a] \Rightarrow x = y) \quad \text{iff} \quad \text{KAT} \vdash x\theta = y\theta \qquad (\theta = \{p \mapsto [\neg a]p + [a]\})$$

4.1 Automating elimination of hypotheses in Coq

The previous techniques to eliminate some hypotheses in KAT can be easily automated in Coq. We first prove once and for all the appropriate equivalences and implications (the tactic `kat` is useful for that). We then define some tactics in Ltac that collect hypotheses of shape (i-iv), put them into shape (i), and aggregate them into a single one which is finally used to update the goal according to (†). Separately, we define a tactic that rewrites in the goal using all hypotheses of shape (v), through (‡). Finally, we obtain a tactic called `hkat`, that just preprocesses the conclusion of the goal using all hypotheses of shape (i-v) and then calls the `kat` tactic. Note that the completeness of this method [15] is a meta-theorem; we do not need to formalise it.

5 Case studies

We now present some examples of Coq formalisations where one can take advantage of our library.

5.1 Bigstep semantics of 'while' programs

The bigstep semantics of 'while' programs is taught in almost every course on semantics and programming languages. Such programs can be embedded into KAT in a straightforward way [21], thus providing us with proper tools to reason about them. Let us formalise such a language in Coq.

Assume a type `state` of states, a type `loc` of memory locations, and an `update` function allowing to update the value of a memory location. Call *arithmetic expression* any function from states to natural numbers, and *Boolean expression* any function from states to Booleans (we use a partially shallow embedding). The 'while' programming language is defined by the inductive type below:

```
Variable loc, state: Set.
Variable update: loc → nat → state → state.

Definition expr := state → nat.
Definition test := state → bool.
```

```
Inductive prog :=
| skp
| aff (l: loc) (e: expr)
| seq (p q: prog)
| ite (b: test) (p q: prog)
| whl (b: test) (p: prog).
```

The bigstep semantics of such programs is given as a "state transformer", i.e., a binary relation between states. Following standard textbooks, one can define this semantics in Coq using an inductive predicate:

```
Inductive bstep: prog → rel state state :=
| s_skp: ∀ s, bstep skp s s
| s_aff: ∀ l e s, bstep (aff l e) s (update l (e s) s)
| s_seq: ∀ p q s s' s'', bstep p s s' → bstep q s' s'' → bstep (seq p q) s s''
| s_ite_ff: ∀ b p q s s', ¬ b s → bstep q s s' → bstep (ite b p q) s s'
| s_ite_tt: ∀ b p q s s', b s → bstep p s s' → bstep (ite b p q) s s'
| s_whl_ff: ∀ b p s, ¬ b s → bstep (whl b p) s s
| s_whl_tt: ∀ b p s s', b s → bstep (seq p (whl b p)) s s' → bstep (whl b p) s s'.
```

Alternatively, one can define this semantic through the relational model of KAT, by induction over the program structure:

```
Fixpoint bstep (p: prog): rel state state :=
  match p with
    | skp ⇒ 1
    | seq p q ⇒ bstep p·bstep q
    | aff l e ⇒ upd l e
    | ite b p q ⇒ [b]·bstep p + [¬b]·bstep q
    | whl b p ⇒ ([b]· bstep p)^*·[¬b]
  end.
```

(Notations come for free since binary relations are already declared as a model of KAT in our library.) The 'skip' instruction is interpreted as the identity relation; sequential composition is interpreted by relational composition. Assignments are interpreted using an auxiliary function, defined as follows:

```
Definition upd l e: rel state state := fun s s' ⇒ s' = update l (e s) s.
```

For the 'if-then-else' statement, the Boolean expression `b` is a predicate on states, i.e., a test in our relational model of KAT; this test is used to guard both branches of the possible execution paths. Accordingly for the 'while' loop, we iterate the body of the loop guarded by the test, using Kleene star. We make sure one cannot exit the loop before the condition gets false by post-guarding the iteration with the negation of this test.

This alternative definition is easily proved equivalent to the previous one. Its relative conciseness makes it easier to read; more importantly, this definition allows us to exploit all theorems and tactics about KAT, for free. For instance, suppose that one wants to prove some program equivalences. First define program equivalence, through the bigstep semantics:

```
Notation "p ∼ q" := (bstep p == bstep q).
```

(The "==" symbol denotes equality in the considered KAT model; in this case, relational equality.) The following lemmas about unfolding loops and dead code elimination, can be proved automatically.

```
Lemma two_loops b p: whl b (whl b p) ∼ whl b p.
Proof. simpl. kat. Qed.
(* ([b]·(([b]·bstep p)^*·[¬b]))^*·[¬b] == ([b]·bstep p)^*·[¬b] *)

Lemma fold_loop b p: whl b (p ; ite b p skp) ∼ whl b p.
Proof. simpl. kat. Qed.
(* ([b]·(bstep p·([b]·bstep p + [¬b]·1)))^*·[¬b] == ([b]·bstep p)^*·[¬b] *)
```

```
Lemma dead_code a b p q r:  whl (a∨b) p ; ite b q r  ∼  whl (a∨b) p ; r.
Proof. simpl. kat. Qed.
(* ([a∨b]·bstep p)*·[¬(a∨b)]·([b]·bstep q + [¬b]·bstep r)
                                  == ([a∨b]·bstep p)*·[¬(a∨b)]·bstep r *)
```

(The semicolon in program expressions is a notation for sequential composition; the comments below each proof show the intermediate goal where the `bstep` fixpoint has been simplified, thus revealing the underlying KAT equality.)

Of course, the `kat` tactic cannot prove arbitrary program equivalences: the theory of KAT only deals with the control-flow graph of the programs and with the Boolean expressions, not with the concrete meaning of assignments or arithmetic expressions. We can however mix automatic steps with manual ones. Consider for instance the following example, where we prove that an assignment can be delayed. Our tactics cannot solve it automatically since some reasoning about assignments is required; however, by asserting manually a simple fact (in this case, an equation of shape (ii)), the goal becomes provable by the `hkat` tactic.

```
Definition subst l e (b: test): test := fun s ⇒ b (update l (e s) s).
Lemma aff_ite l e b p q: (l←e; ite b p q) ∼ (ite (subst l e b) (l←e; p) (l←e; q)).
Proof.
  simpl. (* upd l e·([b]·bstep p + [¬b]·bstep q) ==
        [subst l e b]·(upd l e·bstep p)·[¬subst l e b]·(upd l e·bstep q) *)
  assert (upd l e·[b] == [subst l e b]·upd l e) by (cbv; firstorder; subst; eauto).
  hkat.
Qed.
```

5.2 Hoare logic for partial correctness

Hoare logic for partial correctness [16] is subsumed by KAT [21]. The key ingredient in Hoare logic is the notion of a "Hoare triple" $\{A\}\,p\,\{B\}$, where p is a program, and A, B are two formulas about the memory manipulated by the program, respectively called pre- and post-conditions. A Hoare triple $\{A\}\,p\,\{B\}$ is *valid* if whenever the program p starts in some state s satisfying A and terminates in a state s', then s' satisfies B. Such a statement can be translated into KAT as a simple equation:

$$[A]p[\neg B] = 0$$

Indeed, $[A]p[\neg B] = 0$ precisely means that there is no execution path along p that starts in A and ends in $\neg B$. Such equations are Hoare equations (they have the shape (i) from §4), so that they can be eliminated automatically. As a consequence, inference rules of Hoare logic can be proved automatically using the `hkat` tactic. For instance, for the 'while' rule, we get the following script:

```
Lemma rule_whl A b p: {A∧b} p {A} → {A} whl b p {A∧¬b}.
Proof. simpl. hkat. Qed.
(* [A∧b]·bstep p·[¬A] == 0 → [A]·(([b]·bstep p)*·[¬b])·[¬(A∧¬b)] == 0 *)
```

5.3 Compiler optimisations

Kozen and Patron [23] use KAT to verify a rather large range of standard compiler optimisations, by equational reasoning. Citing their abstract, they cover "*dead code elimination, common subexpression elimination, copy propagation, loop hoisting, induction variable elimination, instruction scheduling, algebraic simplification, loop unrolling, elimination of redundant instructions, array bounds check elimination, and introduction of sentinels*". They cannot use automation, so that the size of their proofs ranges from a few lines to half a page of KAT computations.

We formalised all those equational proofs using our library. Most of them can actually be solved instantaneously, by a simple call to the `hkat` tactic. For the few remaining ones, we gave three to four line proofs, consisting of first rewriting using hypotheses that cannot be eliminated, and then a call to `hkat`.

The reason why `hkat` performs so well is that most assumptions allowing to optimise the code in these examples are of the shape (i-v). For instance, to state that an instruction p has no effect when $[a]$ is satisfied, we use an assumption $[a]p = [a]$. Similarly, to state that the execution of a program x systematically enforces $[a]$, we use an assumption $x = x[a]$. The assumptions that cannot be eliminated are typically those of the shape $pq = qp$: "the instructions p and q commute"; such assumptions have to be used manually.

5.4 Flowchart schemes

The last example we discuss here is due to Paterson, it consists in proving the equivalence of two flowchart schemes (i.e., goto programs—see Manna's book [26] for a complete description of this model). The two schemes are given in Appendix A; Manna proves their equivalence using several successive graph transormations. His proof is really high-level and informal; it is one page long, plus three additional pages to draw intermediate flowcharts schemes. Angus and Kozen [3] give a rather detailed equational proof in KAT, which is about six pages long. Using the `hkat` tactic together with some ad-hoc rewriting tools, we managed to formalise Angus and Kozen's proof in three rather sparse screens.

Like in Angus and Kozen's proof, we progressively modify the KAT expression corresponding to the first schema, to make it evolve towards the expression corresponding to the second schema. Our mechanised proof thus roughly consists in a sequence of transitivity steps closed by `hkat`, allowing us to perform some rewriting steps manually and to move to the next step. This is illustrated schematically by the code presented in Fig. 2.

Most of our transitivity steps (the y_i's) already appear in Angus and Kozen's proof; we can actually skip a lot of their steps, thanks to `hkat`. Some of these simplifications can be spectacular: for instance, they need one page to justify the passage between their expressions (24) and (27), while a simple call to `hkat` does the job; similarly for the page they need between their steps (38) and (43).

```
Lemma Paterson: x_1 == z.
Proof.
  transitivity y_1. hkat.        (* x_1 == y_1 *)
  a few rewriting steps transforming y_1 into x_2.
  transitivity y_2. hkat.        (* x_2 == y_2 *)
  a few rewriting steps transforming y_2 into x_3.
  (* ... *)
  transitivity y_12. hkat.       (* x_12 == y_12 *)
  a few rewriting steps transforming y_12 into x_13.
  hkat.                          (* x_13 == z *)
Qed.
```

Figure2. Squeleton for the proof of equivalence of Paterson's flowchart schems

6 Related works

Several formalisations of algorithms and results related to regular expressions and languages have been proposed since we released our Coq reflexive decision procedure for Kleene algebra [9]: partial derivatives for regular expressions [2], regular expression equivalence [6, 12, 25, 27], regular expression matching [17]. None of these works contains a formalised proof of completeness for Kleene algebra, so that they cannot be used to obtain a general tactic for KA (note however that Krauss and Nipkow [25] obtain an Isabelle/HOL tactic for binary relations using a nice trick to sidestep the completeness proof—but they cannot deal with other models of KA).

On the more algebraic side, Struth et al. [5, 13] showed how to formalise and use relation algebra and Kleene algebra in Isabelle/HOL; they exploit the automation tools provided by this assistant, but they do not try to define decision procedures specific to Kleene algebra, and they do not prove completeness.

To the best our knowledge, the only formalisation of KAT prior to the present work is due to Pereira and Moreira [28], in Coq. They state all axioms of KAT, derive some simple consequences of these axioms (e.g., Boolean disjunction distribute over conjunction, Kleene star is monotone), and use them to manually prove the inference rules of Hoare logic, as we did automatically in §5.2. They do not provide models, automation tools, or completeness proof.

7 Conclusion

We presented a rather exhaustive Coq formalisation of Kleene algebra with tests: axiomatisation, models, completeness proof, decision procedure, elimination of hypotheses. We then showed several use-cases for the corresponding library: proofs about while programs and Hoare logic, certification of standard compiler optimisations, and equivalence of flowchart schemes.

Most of the theoretical material is due to Kozen et al. [3, 15, 18–24], so that our contribution mostly lies in the Coq mechanisation of these ideas. The completeness proof was particularly challenging to formalise, and lots of aspects of

this work could not be explained in this extended abstract: how to encode the algebraic hierarchy, how to work efficiently with finite sets and finite sums, how to exploit symmetry arguments, reflexive normalisation tactics, tactics about lattices, finite ordinals and encodings of set-theoretic constructs in ordinals...

The Coq library is available online [30]; it is documented and axiom-free; its overall structure is given in Appendix B. This library actually has a larger scope than what we presented here: our long-term goal is to formalise and automate other fragments of relation algebra (residuated structures, Kleene algebra with converse, allegories...), so that the library is designed to allow for such extensions. For instance normalisation tactics and an ad-hoc semi-decision procedures are already defined for algebraic structures beyond Kleene algebra and KAT.

According to `coqwc`, the library consists of 4377 lines of specifications and 3020 lines of proofs, that distribute as follows. Overall, this is slightly less than our previous library for KA [9] (5105+4315 lines), and we do much more: not only we handle KAT, but we also lay the ground for the mechanisation of other fragments of relation algebra, as explained above.

	specifications	proofs	comments
ordinals, comparisons, finite sets...	674	323	225
algebraic hierarchy	490	374	216
models (languages, relations, expressions...)	1279	461	404
linear algebra, matrices	534	418	163
completeness, decisions procedure, tactics	1400	1444	740

The resulting theorems and tactics allowed us to shorten significantly a number of paper proofs—those about Hoare logic, compiler optimisations, and flowchart schemes. Getting a way to guarantee that such proofs are correct is important: although mathematically simple, they tend to be hard to proofread (we invite the skeptical reader to check Angus and Kozen's paper proof of Paterson example [3]). Moreover, automation greatly helps when searching for such proofs: being able to get either a proof or a counter-example for any proposed equation is a big plus: it makes it much easier to progress in the overall proof.

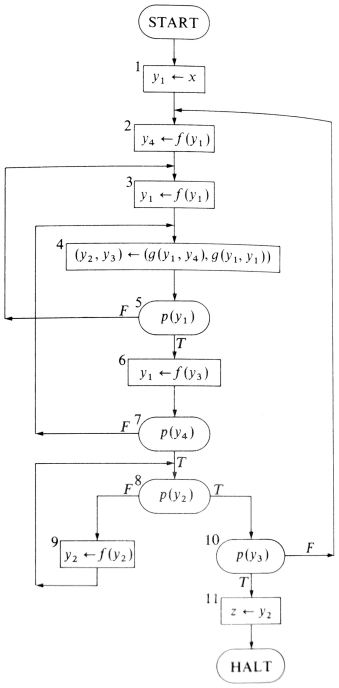
References

1. S. F. Allen, R. L. Constable, D. J. Howe, and W. E. Aitken. The semantics of reflected proof. In *Proc. LICS*, pages 95–105. IEEE Computer Society, 1990.
2. J. B. Almeida, N. Moreira, D. Pereira, and S. M. de Sousa. Partial derivative automata formalized in Coq. In *Proc. CIAA*, volume 6482 of *LNCS*, pages 59–68. Springer, 2010.
3. A. Angus and D. Kozen. Kleene algebra with tests and program schematology. Technical Report TR2001-1844, CS Dpt, Cornell University, July 2001.
4. V. M. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *TCS*, 155(2):291–319, 1996.
5. A. Armstrong and G. Struth. Automated reasoning in higher-order regular algebra. In *Proc. RAMiCS*, volume 7560 of *LNCS*, pages 66–81. Springer, 2012.

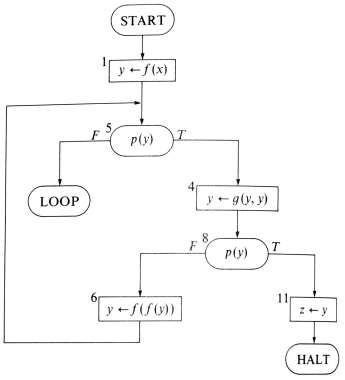
6. A. Asperti. A compact proof of decidability for regular expression equivalence. In *Proc. ITP*, volume 7406 of *LNCS*, pages 283–298. Springer, 2012.
7. Y. Bertot, G. Gonthier, S. O. Biha, and I. Pasca. Canonical big operators. In *TPHOLs*, volume 5170 of *LNCS*, pages 86–101. Springer, 2008.
8. R. Boyer and J. Moore. Metafunctions: proving them correct and using them efficiently as new proof procedures. In *The Correctness Problem in Computer Science*. NY: Academic Press, 1981.
9. T. Braibant and D. Pous. An efficient Coq tactic for deciding Kleene algebras. In *Proc. 1st ITP*, volume 6172 of *LNCS*, pages 163–178. Springer, 2010.
10. J. A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964.
11. E. Cohen. Hypotheses in Kleene algebra. Technical report, Bellcore, Morristown, N.J., 1994.
12. T. Coquand and V. Siles. A decision procedure for regular expression equivalence in type theory. In *Proc. CPP*, volume 7086 of *LNCS*. Springer, 2011.
13. S. Foster and G. Struth. Automated analysis of regular algebra. In *Proc. IJCAR*, volume 7364 of *LNCS*, pages 271–285. Springer, 2012.
14. B. Grégoire and A. Mahboubi. Proving equalities in a commutative ring done right in Coq. In *Proc. TPHOL*, volume 3603 of *LNCS*, pages 98–113. Springer, 2005.
15. C. Hardin and D. Kozen. On the elimination of hypotheses in Kleene algebra with tests. Technical Report TR2002-1879, CS Dpt, Cornell University, October 2002.
16. C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.
17. V. Komendantsky. Reflexive toolbox for regular expression matching: verification of functional programs in Coq+ssreflect. In *Proc. PLPV*, pages 61–70. ACM, 2012.
18. D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Inf. and Comp.*, 110(2):366–390, 1994.
19. D. Kozen. Kleene algebra with tests. *Transactions on Programming Languages and Systems*, 19(3):427–443, May 1997.
20. D. Kozen. Typed Kleene algebra, 1998. TR98-1669, CS Dpt. Cornell University.
21. D. Kozen. On Hoare logic and Kleene algebra with tests. *ACM Trans. Comput. Log.*, 1(1):60–76, 2000.
22. D. Kozen. On the coalgebraic theory of Kleene algebra with tests. Technical Report `http://hdl.handle.net/1813/10173`, CIS, Cornell University, March 2008.
23. D. Kozen and M.-C. Patron. Certification of compiler optimizations using Kleene algebra with tests. In *Proc. CL2000*, volume 1861 of *LNAI*, pages 568–582. Springer, 2000.
24. D. Kozen and F. Smith. Kleene algebra with tests: Completeness and decidability. In *Proc. CSL*, volume 1258 of *LNCS*, pages 244–259. Springer, September 1996.
25. A. Krauss and T. Nipkow. Proof pearl: Regular expression equivalence and relation algebra. *JAR*, 49(1):95–106, 2012.
26. Z. Manna. *Mathematical Theory of Computation*. McGraw-Hill, 1974.
27. N. Moreira, D. Pereira, and S. M. de Sousa. Deciding regular expressions (in-)equivalence in Coq. In *Proc. RAMiCS*, volume 7560 of *LNCS*, pages 98–113. Springer, 2012.
28. D. Pereira and N. Moreira. KAT and PHL in Coq. *Comput. Sci. Inf. Syst.*, 5(2):137–160, 2008.
29. D. Pous. Untyping typed algebraic structures and colouring proof nets of cyclic linear logic. In *Proc. CSL*, volume 6247 of *LNCS*, pages 484–498. Springer, 2010.
30. D. Pous. RelationAlgebra: Coq library containing all material presented in this paper. `http://perso.ens-lyon.fr/damien.pous/ra`, December 2012.
31. K. Thompson. Regular expression search algorithm. *C. ACM*, 11:419–422, 1968.

A Paterson's flowchart schemes

Here are the two flowchart schemes we proved equivalent (§5.4), they appear in [26, pages 254 and 258].



Schema S_{6A}



Schema S_{6E}

Following Angus and Kozen's notations [3], these two schemes can be converted into the following KAT expressions:

$$S_{6A} = x_1 p_{41} p_{11} q_{214} q_{311} \left([\neg a_1] p_{11} q_{214} q_{311}\right)^\star [a_1] p_{13} \\ \Big(([\neg a_4] + [a_4]([\neg a_2] p_{22})^\star [a_2 \wedge \neg a_3] p_{41} p_{11})\, q_{214} q_{311} \left([\neg a_1] p_{11} q_{214} q_{311}\right)^\star [a_1] p_{13}\Big)^\star \\ [a_4] \left([\neg a_2] p_{22}\right)^\star [a_2 \wedge a_3] z_2$$

$$S_{6E} = x_1 [a_1] q_{111} \left([\neg a_1] r_{11} [a_1] q_{111}\right)^\star [a_1] z_1 \ ,$$

where the tests and actions are interpreted as follows:

$$x_i \triangleq y_i \leftarrow x \qquad z_i \triangleq z \leftarrow y_i \qquad a_i \triangleq P(y_i)$$
$$p_{ij} \triangleq y_i \leftarrow f(y_j) \qquad q_{ijk} \triangleq y_i \leftarrow g(y_j, y_k) \qquad r_{ij} \triangleq y_i \leftarrow f(f(y_j))$$

(Note that we actually renamed the local variable y from schema S_{6E} into y_1, for the sake of uniformity.)

B Overall structure of the library

Here is a succinct description of each module from the library:

Utilities
- `common`: basic tactics and definitions used throughout the library
- `comparisons`: types with decidable equality and ternary comparison function
- `positives`: simple facts about binary positive numbers
- `ordinal`: finite ordinals, finite sets of finite ordinals
- `pair`: encoding pairs of ordinals as ordinals
- `powerfix`: simple pseudo-fixpoint iterator
- `lset`: sup-semilattice of finite sets represented as lists

Algebraic hierarchy
- `level`: bitmasks allowing us to refer to an arbitrary point in the hierarchy
- `lattice`: "flat" structures, from preorders to Boolean lattices
- `monoid`: typed structures, from po-monoids to residuated Kleene lattices
- `kat`: Kleene algebra with tests
- `kleene`: Basic facts about Kleene algebra
- `normalisation`: normalisation and semi-decision tactics for relation algebra

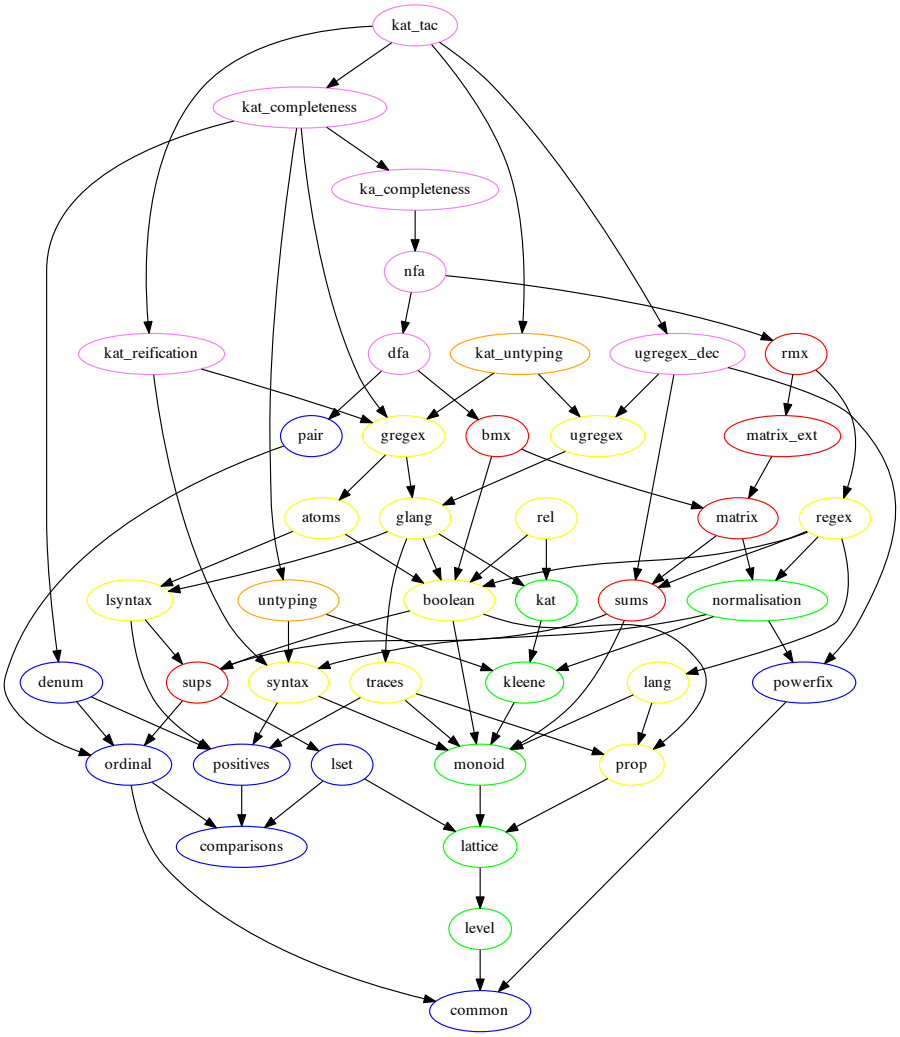
Models
- `prop`: distributive lattice of propositions
- `boolean`: Boolean trivial lattice, extended to a monoid.
- `rel`: heterogeneous binary relations
- `lang`: word languages
- `traces`: trace languages
- `atoms`: atoms of the free Boolean lattice over a finite set
- `glang`: guarded string languages
- `lsyntax`: free lattice (Boolean expressions)
- `syntax`: free relation algebra
- `regex`: regular expressions
- `gregex`: KAT expressions (typed—for KAT completeness)
- `ugregex`: untyped KAT expressions (untyped—for KAT decision procedure)

Untyping theorems
- `untyping`: untyping theorem for structures below KA with converse
- `kat_untyping`: untyping theorem for guarded string languages

Linear algebra
- `sups`: finite suprema/infima (a la bigop, from ssreflect)
- `sums`: finite sums
- `matrix`: matrices over all structures supporting this construction
- `matrix_ext`: additional operations and properties about matrices
- `rmx`: matrices of regular expressions
- `bmx`: matrices of Booleans

Automata, completeness
- `dfa`: deterministic finite state automata, decidability of language inclusion
- `nfa`: matricial non-deterministic finite state automata
- `ugregex_dec`: decision of language equivalence for KAT expressions
- `ka_completeness`: (untyped) completeness of Kleene algebra
- `kat_completeness`: (typed) completeness of Kleene algebra with tests
- `kat_reification`: tools and definitions for KAT reification
- `kat_tac`: decision tactics for KA and KAT, elimination of hypotheses

Here are the dependencies between these modules:



Logical Methods in Computer Science
Vol. 9(1:09)2013, pp. 1–27
www.lmcs-online.org

Submitted Jun. 16, 2011
Published Mar. 4, 2013

GENERALIZING DETERMINIZATION FROM AUTOMATA TO COALGEBRAS

ALEXANDRA SILVA [a], FILIPPO BONCHI [b], MARCELLO BONSANGUE [c], AND JAN RUTTEN [d]

[a] Radboud University Nijmegen and Centrum Wiskunde & Informatica
e-mail address: ams@cwi.nl

[b] ENS Lyon, Université de Lyon, LIP (UMR 5668 CNRS ENS Lyon UCBL INRIA)
e-mail address: filippo.bonchi@ens-lyon.fr

[c] LIACS - Leiden University
e-mail address: marcello@liacs.nl

[d] Centrum Wiskunde & Informatica and Radboud University Nijmegen
e-mail address: janr@cwi.nl

Abstract. The powerset construction is a standard method for converting a nondeterministic automaton into a deterministic one recognizing the same language. In this paper, we lift the powerset construction from automata to the more general framework of coalgebras with structured state spaces. Coalgebra is an abstract framework for the uniform study of different kinds of dynamical systems. An endofunctor F determines both the type of systems (F-coalgebras) and a notion of behavioural equivalence ($\sim_F$) amongst them. Many types of transition systems and their equivalences can be captured by a functor F. For example, for deterministic automata the derived equivalence is language equivalence, while for non-deterministic automata it is ordinary bisimilarity.

We give several examples of applications of our generalized determinization construction, including partial Mealy machines, (structured) Moore automata, Rabin probabilistic automata, and, somewhat surprisingly, even pushdown automata. To further witness the generality of the approach we show how to characterize coalgebraically several equivalences which have been object of interest in the concurrency community, such as failure or ready semantics.

2012 ACM CCS: [**Theory of computation**]: Models of computation—Abstract machines & Formal languages and automata theory—Formalisms—Algebraic language theory & Semantics and reasoning—Program semantics—Categorical semantics.

Key words and phrases: Coalgebras, Powerset Construction, Linear Semantics.

[a] The work of Alexandra Silva is partially funded by the ERDF through the Programme COMPETE and by the Portuguese Foundation for Science and Technology, project ref. `PTDC/EIA-CCO/122240/2010` and `SFRH/BPD/71956/2010`.

[b] The work of Filippo Bonchi is supported by the CNRS PEPS project CoGIP and the project ANR 12IS02001 PACE.

[c,d] The research of Marcello Bonsangue and Jan Rutten has been carried out under the Dutch NWO project *CoRE: Coinductive Calculi for Regular Expressions.*, dossier number 612.063.920.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.2168/LMCS-9(1:09)2013

Introduction

Coalgebra is by now a well established general framework for the study of the behaviour of large classes of dynamical systems, including various kinds of automata (deterministic, probabilistic etc.) and infinite data types (streams, trees and the like). For a functor $F\colon \mathbf{Set} \to \mathbf{Set}$, an F-coalgebra is a pair (X, f), consisting of a set X of states and a function $f\colon X \to F(X)$ defining the observations and transitions of the states. Coalgebras generally come equipped with a standard notion of equivalence called *F-behavioural equivalence* that is fully determined by their (functor) type F. Moreover, for most functors F there exists a *final* coalgebra into which any F-coalgebra is mapped by a unique homomorphism that identifies all F-equivalent states.

Much of the coalgebraic approach can be nicely illustrated with deterministic automata (DA), which are coalgebras of the functor $D(X) = 2 \times X^A$. In a DA, two states are D-equivalent precisely when they accept the same language. The set 2^{A^*} of all formal languages constitutes a final D-coalgebra, into which every DA is mapped by a homomorphism that sends any state to the language it accepts.

It is well-known that *non-deterministic* automata (NDA) often provide more efficient (smaller) representations of formal languages than DA's. Language acceptance of NDA's is typically defined by turning them into DA's via the *powerset construction*. Coalgebraically this works as follows. NDA's are coalgebras of the functor $N(X) = 2 \times \mathcal{P}_\omega(X)^A$, where $\mathcal{P}_\omega$ is the finite powerset. An N-coalgebra $(X, f\colon X \to 2 \times \mathcal{P}_\omega(X)^A)$ is *determinized* by transforming it into a D-coalgebra $(\mathcal{P}_\omega(X), f^\sharp\colon \mathcal{P}_\omega(X) \to 2 \times \mathcal{P}_\omega(X)^A)$ (for details see Section 2). Then, the language accepted by a state s in the NDA (X, f) is defined as the language accepted by the state $\{s\}$ in the DA $(\mathcal{P}_\omega(X), f^\sharp)$.

For a second variation on DA's, we look at *partial automata* (PA): coalgebras of the functor $P(X) = 2 \times (1 + X)^A$, where for certain input letters transitions may be undefined. Again, one is often interested in the DA-behaviour (i.e., language acceptance) of PA's. This can be obtained by turning them into DA's using *totalization*. Coalgebraically, this amounts to the transformation of a P-coalgebra $(X, f\colon X \to 2\times(1+X)^A)$ into a D-coalgebra $(1 + X, f^\sharp\colon 1 + X \to 2 \times (1 + X)^A)$.

Although the two examples above may seem very different, they are both instances of one and the same phenomenon, which it is the goal of the present paper to describe at a general level. Both with NDA's and PA's, two things happen at the same time: (i) more (or, more generally, different types of) transitions are allowed, as a consequence of changing the functor type by replacing X by $\mathcal{P}_\omega(X)$ and $(1+X)$, respectively; and (ii) the behaviour of NDA's and PA's is still given in terms of the behaviour of the original DA's (language acceptance).

For a large family of F-coalgebras, both (i) and (ii) can be captured simultaneously with the help of the categorical notion of *monad*, which generalizes the notion of algebraic theory. The structuring of the state space X can be expressed as a change of functor type from $F(X)$ to $F(T(X))$. In our examples above, both the functors $T_1(X) = \mathcal{P}_\omega(X)$ and $T_2(X) = 1 + X$ are monads, and NDA's and PA's are obtained from DA's by changing the original functor type $D(X)$ into $N(X) = D(T_1(X))$ and $P(X) = D(T_2(X))$. Regarding (ii), one assigns F-semantics to an FT-coalgebra (X, f) by transforming it into an F-coalgebra $(T(X), f^\sharp)$, again using the monad T. In our examples above, the determinization of NDA's and the totalization of PA's consists of the transformation of N- and P-coalgebras (X, f) into D-coalgebras $(T_1(X), f^\sharp)$ and $(T_2(X), f^\sharp)$, respectively.

We shall investigate general conditions on the functor types under which the above constructions can be applied: for one thing, one has to ensure that the FT-coalgebra map $f\colon X \to F(T(X))$ induces a suitable F-coalgebra map $f^\sharp\colon T(X) \to F(T(X))$. Our results will lead to a uniform treatment of all kinds of existing and new variations of automata, that is, FT-coalgebras, by an algebraic structuring of their state space through a monad T. Furthermore, we shall prove a number of general properties that hold in all situations similar to the ones above. For instance, there is the notion of N-behavioural equivalence with which NDA's, being N-coalgebras, come equipped. It coincides with the well-known notion of Park-Milner bisimilarity from process algebra. A general observation is that if two states in an NDA are N-equivalent then they are also D- (that is, language-) equivalent. For PA's, a similar statement holds. One further contribution of this paper is a proof of these statements, once and for all for all FT-coalgebras under consideration.

Coalgebras of type FT were studied in [29, 4, 22]. In [4, 22] the main concern was definitions by coinduction, whereas in [29] a proof principle was also presented. All in all, the present paper can be seen as the understanding of the aforementioned papers from a new perspective, presenting a uniform view on various automata constructions and equivalences.

The structure of the paper is as follows. After preliminaries (Section 1) and the details of the motivating examples above (Section 2), Section 3 presents the general construction as well as many more examples, including the coalgebraic chracterisation of pushdown automata (Section 3.2). In Section 4, a large family of automata (technically: functors) is characterised to which the constructions above can be applied. Section 5 contains the application of the framework in order to recover several interesting equivalences stemming from the world of concurrency, such as failure and ready semantics. Section 6 discusses related work and presents pointers to future work.

This paper is an extended version of [43]. Compared to the conference version, we include the proofs and more examples. More interestingly, the characterisation of pushdown automata coalgebraically (Section 3.2) and the material in Section 5 are original.

1. Background

In this section we introduce the preliminaries on coalgebras and algebras. First, we fix some notation on sets. We will denote sets by capital letters $X, Y, \ldots$ and functions by lower case letters $f, g, \ldots$ Given sets X and Y, $X \times Y$ is the cartesian product of X and Y (with the usual projection maps π_1 and π_2), $X + Y$ is the disjoint union (with injection maps κ_1 and κ_2) and X^Y is the set of functions $f\colon Y \to X$. The collection of finite subsets of X is denoted by $\mathcal{P}_\omega(X)$, while the collection of full-probability distributions with finite support is $\mathcal{D}_\omega(X) = \{f\colon X \to [0,1] \mid f \text{ finite support and } \sum_{x\in X} f(x) = 1\}$. For a set of letters A, A^* denotes the set of all words over A; ϵ the empty word; and $w_1 \cdot w_2$ (and $w_1 w_2$) the concatenation of words $w_1, w_2 \in A^*$.

1.1. **Coalgebras.** A coalgebra is a pair $(X, f\colon X \to F(X))$, where X is a set of states and $F\colon \mathbf{Set} \to \mathbf{Set}$ is a functor. The functor F, together with the function f, determines the *transition structure* (or dynamics) of the F-coalgebra [37].

An *F-homomorphism* from an F-coalgebra (X, f) to an F-coalgebra (Y, g) is a function $h\colon X \to Y$ preserving the transition structure, *i.e.*, $g \circ h = F(h) \circ f$.

An F-coalgebra (Ω, ω) is said to be *final* if for any F-coalgebra (X, f) there exists a unique F-homomorphism $[\![-]\!]_X \colon X \to \Omega$. All the functors considered in examples in this paper have a final coalgebra.

Let (X, f) and (Y, g) be two F-coalgebras. We say that the states $x \in X$ and $y \in Y$ are *behaviourally equivalent*, written $x \sim_F y$, if and only if they are mapped into the same element in the final coalgebra, that is $[\![x]\!]_X = [\![y]\!]_Y$.

For weak pullback preserving functors, behavioural equivalence coincides with the usual notion of bisimilarity [37].

1.2. **Algebras.** Monads can be thought of as a generalization of algebraic theories. A *monad* $\mathbf{T} = (T, \mu, \eta)$ is a triple consisting of an endofunctor T on **Set** and two natural transformations: a *unit* $\eta \colon Id \Rightarrow T$ and a *multiplication* $\mu \colon T^2 \Rightarrow T$. They satisfy the following commutative laws

$$\mu \circ \eta_T = id_T = \mu \circ T\eta \quad \text{and} \quad \mu \circ \mu_T = \mu \circ T\mu.$$

Sometimes it is more convenient to represent a monad $\mathbf{T}$, equivalently, as a *Kleisli triple* $(T, (\_)^\sharp, \eta)$ [31], where T assigns a set $T(X)$ to each set X, the unit η assigns a function $\eta_X \colon X \to T(X)$ to each set X, and the extension operation $(\_)^\sharp$ assigns to each $f \colon X \to T(Y)$ a function $f^\sharp \colon T(X) \to T(Y)$, such that,

$$f^\sharp \circ \eta_X = f \qquad (\eta_X)^\sharp = id_{T(X)} \qquad (g^\sharp \circ f)^\sharp = g^\sharp \circ f^\sharp \,,$$

for $g \colon Y \to T(Z)$. Monads are frequently referred to as *computational types* [32]. We list now a few examples. In what follows, $f \colon X \to T(Y)$ and $c \in T(X)$.

Nondeterminism. $T(X) = \mathcal{P}_\omega(X)$; η_X is the singleton map $x \mapsto \{x\}$; $f^\sharp(c) = \bigcup_{x \in c} f(x)$.

Partiality. $T(X) = 1 + X$ where $1 = \{*\}$ represents a terminating (or diverging) computation; η_X is the injection map $\kappa_2 \colon X \to 1 + X$; $f^\sharp(\kappa_1(*)) = \kappa_1(*)$ and $f^\sharp(\kappa_2(x)) = f(x)$.

Further examples of monads include: exceptions ($T(X) = E + X$), side-effects ($T(X) = (S \times X)^S$), interactive output ($T(X) = \mu v.X + (O \times v) \cong O^* \times X$) and full-probability ($T(X) = \mathcal{D}_\omega(X)$). We will use all these monads in our examples and we will define η_X and $f^\sharp$ for each later in Section 3.1.

A $\mathbf{T}$-*algebra* of a monad $\mathbf{T}$ is a pair (X, h) consisting of a set X, called carrier, and a function $h \colon T(X) \to X$ such that $h \circ \mu_X = h \circ Th$ and $h \circ \eta_X = id_X$. A T-homomorphism between two $\mathbf{T}$-algebras (X, h) and (Y, k) is a function $f \colon X \to Y$ such that $f \circ h = k \circ Tf$. $\mathbf{T}$-algebras and their homomorphisms form the so-called *Eilenberg-Moore category* $\mathbf{Set}^{\mathbf{T}}$. There is a forgetful functor $U^{\mathbf{T}} \colon \mathbf{Set}^{\mathbf{T}} \to \mathbf{Set}$ defined by

$$U^{\mathbf{T}}((X, h)) = X \quad \text{and} \quad U^{\mathbf{T}}(f \colon (X, h) \to (Y, k)) = f \colon X \to Y \,.$$

The forgetful functor $U^{\mathbf{T}}$ has left adjoint $X \mapsto (T(X), \mu_X \colon TT(X) \to T(X))$, mapping a set X to its free $\mathbf{T}$-algebra. If $f \colon X \to Y$ with (Y, h) a $\mathbf{T}$-algebra, the unique $\mathbf{T}$-homomorphism $f^\sharp \colon (T(X), \mu_X) \to (Y, h)$ with $f^\sharp \circ \eta_X = f$ is given by

$$f^\sharp \colon T(X) \xrightarrow{Tf} T(Y) \xrightarrow{h} Y \,.$$

The function $f^\sharp \colon (T(X), \mu_X) \to (T(Y), \mu_Y)$ coincides with function extension for a Kleisli triple. For the monad $\mathcal{P}_\omega$ the associated Eilenberg-Moore category is the category of join semi-lattices, whereas for the monad $1 + -$ is the category of pointed sets.

2. Motivating examples

In this section, we introduce two motivating examples. We will present two constructions, the determinization of a non-deterministic automaton and the totalization of a partial automaton, which we will later show to be an instance of the same, more general, construction.

2.1. **Non-deterministic automata.** A deterministic automaton (DA) over the input alphabet A is a pair $(X, \langle o, t\rangle)$, where X is a set of states and $\langle o, t\rangle \colon X \to 2 \times X^A$ is a function with two components: o, the output function, determines if a state x is final ($o(x) = 1$) or not ($o(x) = 0$); and t, the transition function, returns for each input letter a the next state. DA's are coalgebras for the functor $2 \times Id^A$. The final coalgebra of this functor is $(2^{A^*}, \langle \epsilon, (-)_a\rangle)$ where 2^{A^*} is the set of languages over A and $\langle \epsilon, (-)_a\rangle$, given a language L, determines whether or not the empty word is in the language ($\epsilon(L) = 1$ or $\epsilon(L) = 0$, resp.) and, for each input letter a, returns the *derivative* of L: $L_a = \{w \in A^* \mid aw \in L\}$. From any DA, there is a unique map l into 2^{A^*} which assigns to each state its behaviour (that is, the language that the state recognizes).

$$\begin{array}{ccc} X & \overset{l}{- - - \to} & 2^{A^*} \\ {\scriptstyle \langle o,t\rangle}\downarrow & & \downarrow{\scriptstyle \langle \epsilon,(-)_a\rangle} \\ 2 \times X^A & \underset{id\times l^A}{- - - \to} & 2 \times (2^{A^*})^A \end{array}$$

A non-deterministic automaton (NDA) is similar to a DA but the transition function gives a set of next-states for each input letter instead of a single state. Thus, an NDA over the input alphabet A is a pair $(X, \langle o, \delta\rangle)$, where X is a set of states and $\langle o, \delta\rangle \colon X \to 2 \times (\mathcal{P}_\omega(X))^A$ is a pair of functions with o as before and where δ determines for each input letter a a set of possible next states. In order to compute the language recognized by a state x of an NDA $\mathcal{A}$, it is usual to first determinize it, constructing a DA $\mathbf{det}(\mathcal{A})$ where the state space is $\mathcal{P}_\omega(X)$, and then compute the language recognized by the state $\{x\}$ of $\mathbf{det}(\mathcal{A})$. Next, we describe in coalgebraic terms how to construct the automaton $\mathbf{det}(\mathcal{A})$.

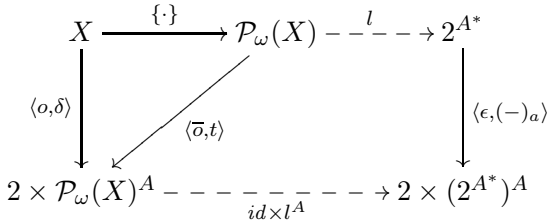
Given an NDA $\mathcal{A} = (X, \langle o, \delta\rangle)$, we construct $\mathbf{det}(\mathcal{A}) = (\mathcal{P}_\omega(X), \langle \overline{o}, t\rangle)$, where, for all $Y \in \mathcal{P}_\omega(X)$, $a \in A$, the functions $\overline{o} \colon \mathcal{P}_\omega(X) \to 2$ and $t \colon \mathcal{P}_\omega(X) \to \mathcal{P}_\omega(X)^A$ are

$$\overline{o}(Y) = \begin{cases} 1 & \exists_{y\in Y} o(y) = 1 \\ 0 & \text{otherwise} \end{cases} \qquad t(Y)(a) = \bigcup_{y\in Y} \delta(y)(a).$$

(Observe that these definitions exploit the join-semilattice structures of 2 and $\mathcal{P}_\omega(X)^A$).

The automaton $\mathbf{det}(\mathcal{A})$ is such that the language $l(\{x\})$ recognized by $\{x\}$ is the same as the one recognized by x in the original NDA $\mathcal{A}$ (more generally, the language recognized by state X of $\mathbf{det}(\mathcal{A})$ is the union of the languages recognized by each state x of $\mathcal{A}$).

We summarize the situation above with the following commuting diagram:



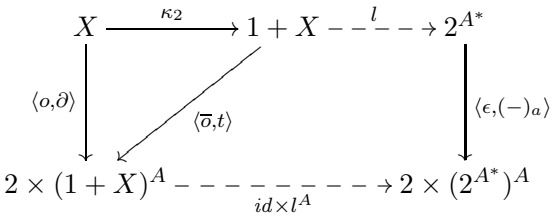
We note that the language semantics of NDA's, presented in the above diagram, can also be obtained as an instance of the abstract definition scheme of λ-coinduction [4, 22].

2.2. **Partial automata.** A partial automaton (PA) over the input alphabet A is a pair $(X, \langle o, \partial\rangle)$ consisting of a set of states X and a pair of functions $\langle o, \partial\rangle \colon X \to 2 \times (1 + X)^A$. Here $o \colon X \to 2$ is the same as with DA. The second function $\partial \colon X \to (1+X)^A$ is a transition function that sends any state $x \in X$ to a function $\partial(x) \colon A \to 1 + X$, which for any input letter $a \in A$ is either undefined (no a-labelled transition takes place) or specifies the next state that is reached. PA's are coalgebras for the functor $2 \times (1 + Id)^A$. Given a PA $\mathcal{A}$, we can construct a total (deterministic) automaton $\mathbf{tot}(\mathcal{A})$ by adding an extra *sink* state to the state space: every undefined a-transition from a state x is then replaced by a a-labelled transition from x to the sink state. More precisely, given a PA $\mathcal{A} = (X, \langle o, \partial\rangle)$, we construct $\mathbf{tot}(\mathcal{A}) = (1 + X, \langle \overline{o}, t\rangle)$, where

$$\begin{array}{ll} \overline{o}(\kappa_1(*)) = 0 & t(\kappa_1(*))(a) = \kappa_1(*) \\ \overline{o}(\kappa_2(x)) = o(x) & t(\kappa_2(x))(a) = \partial(x)(a) \end{array}$$

(Observe that these definitions exploit the pointed-set structures of 2 and $1 + X$).

The language $l(x)$ recognized by a state x will be precisely the language recognized by x in the original partial automaton. Moreover, the new sink state recognizes the empty language. Again we summarize the situation above with the help of following commuting diagram, which illustrates the similarities between both constructions:



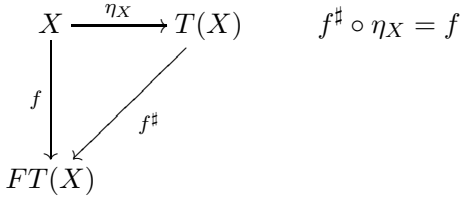
3. Algebraically structured coalgebras

In this section we present a general framework where both motivating examples can be embedded and uniformly studied. We will consider coalgebras for which the functor type FT can be decomposed into a transition type F specifying the relevant dynamics of a system and a monad T providing the state space with an algebraic structure. For simplicity, we fix our base category to be **Set**.

We study coalgebras $f \colon X \to FT(X)$ for a functor F and a monad $\mathbf{T}$ such that $FT(X)$ is a $\mathbf{T}$-algebra, that is $FT(X)$ is the carrier of a $\mathbf{T}$-algebra $(FT(X), h)$. In the motivating

examples, F would be instantiated to $2 \times Id^A$ (in both) and T to $\mathcal{P}_\omega$, for NDAs, and to $1 + -$ for PAs. The condition that $FT(X)$ is a $\mathbf{T}$-algebra would amount to require that $2 \times \mathcal{P}_\omega(X)^A$ is a join-semilattice, for NDAs, and that $2 \times (1 + X)^A$ is a pointed set, for PAs. This is indeed the case, since the set 2 can be regarded both as a join-semilattice ($2 \cong \mathcal{P}_\omega(1)$) or as a pointed set ($2 \cong 1 + 1$) and, moreover, products and exponentials preserve the algebra structure.

The inter-play between the transition type F and the computational type $\mathbf{T}$ (more precisely, the fact that $FT(X)$ is a $\mathbf{T}$-algebra) allows each coalgebra $f\colon X \to FT(X)$ to be extended uniquely to a T-algebra morphism $f^\sharp\colon (T(X), \mu_X) \to (FT(X), h)$ which makes the following diagram commute.



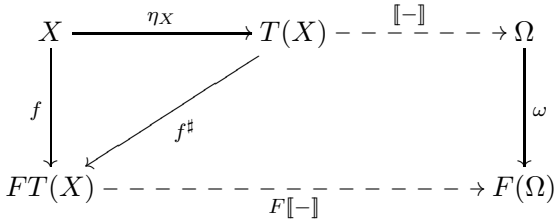
Intuitively, $\eta_X\colon X \to T(X)$ is the inclusion of the state space of the coalgebra $f\colon X \to FT(X)$ into the structured state space $T(X)$, and $f^\sharp\colon T(X) \to FT(X)$ is the extension of the coalgebra f to $T(X)$.

Next, we study the behaviour of a given state or, more generally, we would like to say when two states x_1 and x_2 are equivalent. The obvious choice for an equivalence would be FT-behavioural equivalence. However, this equivalence is not exactly what we are looking for. In the motivating example of non-deterministic automata we wanted two states to be equivalent if they recognize the same language. If we would take the equivalence arising from the functor $2 \times \mathcal{P}_\omega(Id)^A$ we would be distinguishing states that recognize the same language but have difference branching types, as in the following example.

We now define a new equivalence, which *absorbs* the effect of the monad T.

We say that two elements x_1 and x_2 in X are *F-equivalent with respect to a monad* $\mathbf{T}$, written $x_1 \approx_F^T x_2$, if and only if $\eta_X(x_1) \sim_F \eta_X(x_2)$. The equivalence $\sim_F$ is just F-behavioural equivalence for the F-coalgebra $f^\sharp\colon T(X) \to FT(X)$.

If the functor F has a final coalgebra (Ω, ω) , we can capture the semantic equivalence above in the following commuting diagram

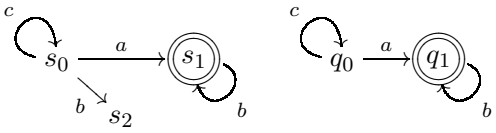


(3.1)

Returning to our first example, two states x_1 and x_2 of an NDA (in which T is instantiated to $\mathcal{P}_\omega$ and F to $2 \times Id^A$) would satisfy $x_1 \approx_F^T x_2$ if and only if they recognize the same language (recall that the final coalgebra of the functor $2 \times Id^A$ is 2^{A^*}).

It is also interesting to remark the difference between the two equivalences in the case of partial automata. The coalgebraic semantics of PAs [39] is given in terms of pairs of prefix-closed languages $\langle V, W\rangle$ where V contains the words that are accepted (that is, are the label of a path leading to a final state) and W contains all words that label any path (that is all that are in V plus the words labeling paths leading to non-final states). We describe V and W in the following two examples, for the states s_0 and q_0:

$W = c^* + c^*b + c^*ab^*$
$V = c^*ab^*$



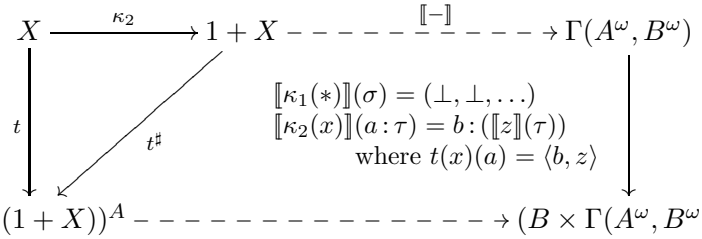
$W = c^* + c^*ab^*$
$V = c^*ab^*$

Thus, the states s_0 and q_0 would be distinguished by FT-equivalence (for $F = 2 \times Id^A$ and $T = 1 + -$) but they are equivalent with respect to the monad $1 + -$, $s_0 \approx_F^T q_0$, since they accept the same language.

We will show in Section 4 that the equivalence $\sim_{FT}$ is always contained in $\approx_F^T$.

3.1. **Examples.** In this section we show more examples of applications of the framework above.

3.1.1. *Partial Mealy machines.* A partial Mealy machine is a set of states X together with a function $t\colon X \to (B \times (1 + X))^A$, where A is a set of inputs and B is a set of output values. We assume that B has a distinguished element $\bot \in B$. For each state x and for each input a the automaton produces an output value and either terminates or continues to a next state. Applying the framework above we will be *totalizing* the automaton, similarly to what happened in the example of partial automata, by adding an extra state to the state space which will act as a sink state. The behaviour of the totalized automaton is given by the set of causal functions from A^ω (infinite sequences of A) to B^ω, which we denote by $\Gamma(A^\omega, B^\omega)$ [38]. A function $f\colon A^\omega \to B^\omega$ is causal if, for $\sigma \in A^\omega$, the n-th value of the output stream $f(\sigma)$ depends only on the first n values of the input stream σ. In the diagram below, we define the final map $[\![-]\!]\colon 1 + X \to \Gamma(A^\omega, B^\omega)$:



Here $* \in 1$, $x \in X$, $a \in A$, $b \in B$, $\sigma \in A^\omega$, $z \in 1 + X$, and $a{:}\tau$ denotes the prefixing of the stream $\tau \in A^\omega$ with the element a.

3.1.2. *Structured Moore automata.* In the following examples we look at the functor

$$F(X) = T(B) \times X^A$$

for arbitrary sets A and B and an arbitrary monad $\mathbf{T} = (T, \eta, (-)^\sharp)$. The coalgebras of F represents Moore automata with outputs in $T(B)$ and inputs in A. Since $T(B)$ is a $\mathbf{T}$-algebra, $T(X)^A$ is a $\mathbf{T}$-algebra and the product of $\mathbf{T}$-algebras is still a $\mathbf{T}$-algebra, then $FT(X)$ is a $\mathbf{T}$-algebra. For this reason, the (pair of) functions $o\colon X \to T(B)$ and $t\colon X \to T(X)^A$ lift to a (pair of) functions

$$o^\sharp\colon T(X) \to T(B) \qquad t^\sharp\colon T(X) \to T(X)^A$$

The final coalgebra of F is $T(B)^{A^*}$. We can characterize the final map $[\![-]\!]\colon T(X) \to T(B)^{A^*}$, for all $m \in T(X)$, $a \in A$ and $w \in A^*$, by

$$\begin{array}{ccc} X & \xrightarrow{\eta_X} & T(X) \xrightarrow{[\![-]\!]} T(B)^{A^*} \\ {\scriptstyle\langle o,t\rangle}\downarrow & \swarrow_{\langle o^\sharp, t^\sharp\rangle} & \begin{array}{l}[\![m]\!](\epsilon) = o^\sharp(m) \\ [\![m]\!](a\cdot w) = [\![t^\sharp(m)(a)]\!](w)\end{array} \quad \downarrow{\scriptstyle\langle \epsilon, (-)_a\rangle} \\ T(B)\times T(X)^A & & \xrightarrow{} T(B)\times (T(B)^{A^*})^A \end{array}$$

Below we shall look at various concrete instances of this scheme, for different choices of the monad T.

Moore automata with exceptions. Let E be an arbitrary set, the elements of which we think of as exceptions. We consider the *exception monad* $T(X) = E + X$ which has the function $\eta(x) = \kappa_2(x)$ as its unit. We define the lifting $f^\sharp\colon T(X) \to T(Y)$, for any function $f\colon X \to T(Y)$, by $f^\sharp = [id, f]$.

An FT-coalgebra $\langle o, t\rangle\colon X \to (E + B) \times (E + X)^A$ will associate with every state x an output value (either in B or an exception in E) and, for each input a, a next state or an exception. The behaviour of a state x, given by $[\![\eta(x)]\!]$, will be a formal power series over A with output values in $E + B$; that is, a function from A^* to $E + B$. The final map is defined as follows, for all $e \in E$, $x \in X$, $a \in A$, and $w \in A^*$:

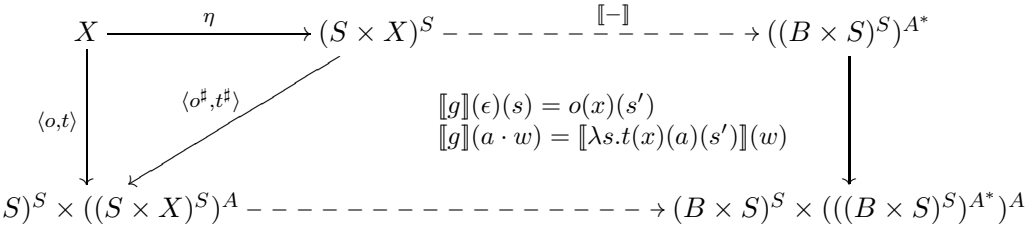
$$\begin{array}{ccc} X & \xrightarrow{\kappa_2} & E + X \xrightarrow{[\![-]\!]} (E + B)^{A^*} \\ {\scriptstyle\langle o,t\rangle}\downarrow & \swarrow_{\langle o^\sharp, t^\sharp\rangle} & \begin{array}{l}[\![\kappa_1(e)]\!](w) = \kappa_1(e) \\ [\![\kappa_2(x)]\!](\epsilon) = o(x) \\ [\![\kappa_2(x)]\!](a\cdot w) = [\![t(x)(a)]\!](w)\end{array} \quad \downarrow \\ (E + B)\times (E + X)^A & & \xrightarrow{} (E + B)\times ((E + B)^{A^*})^A \end{array}$$

Moore automata with side effects. Let S be an arbitrary set of so-called *side-effects*. We consider the monad $T(X) = (S \times X)^S$, with unit η defined, for all $x \in X$ and $s \in S$, by $\eta(x)(s) = \langle s, x\rangle$. We define the lifting $f^\sharp\colon T(X) \to T(Y)$ of a function $f\colon X \to T(Y)$ by $f^\sharp(g)(s) = f(x)(s')$, for any $g \in T(X)$ and $s \in S$, and with $g(s) = \langle s', x\rangle$.

Consider an FT-coalgebra $\langle o, t\rangle\colon X \to (B \times S)^S \times ((S \times X)^S)^A$ and let us explain the intuition behind this type of automaton type. The set $S \times X$ can be interpreted as the configurations of the automaton, where S contains information about the state of the system and X about the control of the system. Using the isomorphism $X \to (S \times B)^S \cong$

$S \times X \to S \times B$, we can think of $o\colon X \to (S \times B)^S$ as a function that for each configuration in $S \times X$ provides an output in B and the new state of the system in S. The transition function $t\colon X \to ((S \times X)^S)^A$ gives a new configuration for each input letter and current configuration, using again the fact that $X \to ((S \times X)^S)^A \cong S \times X \to (S \times X)^A$. In all of this, a concrete instance of the set of side-effects could be, for example, the set $S = V^L$ of functions associating memory locations to values.

The behaviour of a state $x \in X$ will be given by $[\![\eta(x)]\!]$, where the final mapping is as follows. For all $g \in (S \times X)^S$, $s \in S$, $a \in A$ and $w \in A^*$, and with $g(s) = \langle s', x\rangle$, we have



Moore automata with interactive output. Let O be an arbitrary set of *outputs*. Consider the interactive output monad defined by the functor $T(X) = \mu v.X + (O \times v) \cong O^* \times X$ together with the natural transformation $\eta_X = \lambda x \in X.\langle \epsilon, x\rangle$, and for which the lifting $f^\sharp\colon T(X) \to T(Y)$ of a function $f\colon X \to T(Y)$ is given by $f^\sharp(\langle w, x\rangle) = \langle ww', y\rangle$ with $f(x) = \langle w', y\rangle$. We consider FT-coalgebras

$$\langle o, t\rangle\colon X \to (O^* \times B) \times (O^* \times X)^A$$

For $B = 1$, the above coalgebras coincide with *(total) subsequential transducers* [17]: $o\colon X \to O^*$ is the final output function; $t\colon X \to (O^* \times X)^A$ is the pairing of the output function and the next state-function.

The behaviour of a state x will be given by $[\![\eta(x)]\!] = [\![\langle \epsilon, x\rangle]\!]$, where, for every $\langle w, x\rangle \in O^* \times X$, $[\![\langle w, x\rangle]\!]\colon A^* \to O^*$, is given by

$$[\![\langle w, x\rangle]\!](\epsilon) \;=\; w \cdot o(x) \qquad\qquad [\![\langle w, x\rangle]\!](aw_1) \;=\; w \cdot ([\![t(x)(a)]\!](w_1))$$

Probabilistic Moore automata. Consider the monad of probability distributions defined, for any set X, by

$$T(X) = \mathcal{D}_\omega(X)$$

Its unit is given by the Dirac distribution, defined for $x, x' \in X$ by

$$\eta(x)(x') = \begin{cases} 1 & x = x' \\ 0 & \text{otherwise} \end{cases}$$

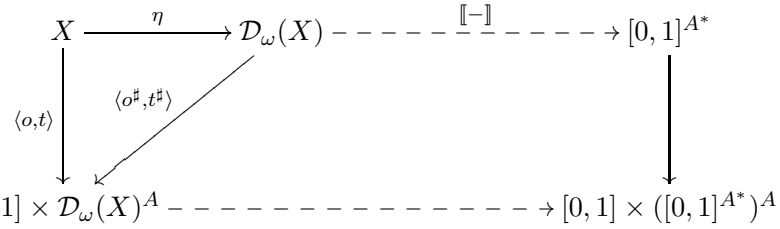
The lifting $f^\sharp\colon T(X) \to T(Y)$ of a function $f\colon X \to T(Y)$ is given, for any distribution $c \in \mathcal{D}_\omega(X)$ and any $y \in Y$, by

$$f^\sharp(c)(y) = \sum_{d \in \mathcal{D}_\omega(Y)} \left(\sum_{x \in f^{-1}(d)} c(x) \right) \times d(y)$$

We will consider FT-coalgebras

$$\langle o, t\rangle\colon X \to \mathcal{D}_\omega(B) \times \mathcal{D}_\omega(X)^A$$

More specifically, we take $B = 2$ which implies $\mathcal{D}_\omega(2) \cong [0,1]$. For this choice of B, the above FT-coalgebras are precisely the *(Rabin) probabilistic automata* [36]. Each state x has an output value in $o(x) \in [0,1]$ and, for each input a, $t(x)(a)$ is a probability distribution of next states. The behaviour of a state x is given by $[\![\eta(x)]\!] \colon A^* \to [0,1]$, defined below. Intuitively, one can think of $[\![\eta(x)]\!]$ as a probabilistic language: each word is associated with a value $p \in [0,1]$. The final mapping



is given, for any $d \in \mathcal{D}_\omega(X)$, $x \in X$, $a \in A$, and $w \in A^*$, by

$$\begin{aligned} [\![d]\!](\epsilon) &= \sum_{b \in [0,1]} \Big(\sum_{o(x)=b} d(x) \Big) \times b \\ [\![d]\!](aw) &= [\![\lambda x'. \sum_{c \in \mathcal{D}_\omega(X)} (\textstyle\sum_{b=t(x)(a)} d(x)) \times c(x')]\!](w) \end{aligned}$$

It is worth noting that this exactly captures the semantics of [36], while the ordinary $\sim_{FT}$ coincides with *probabilistic bisimilarity* of [28]. Moreover $\approx_F^T$ coincides with the trace semantics of probabilistic transition systems defined in [19] (see Section 7.2 of [23]).

3.2. **Pushdown automata, coalgebraically.** Recursive functions in a computer program lead naturally to a stack of recursive function calls during the execution of the program. In this section, we provide a coalgebraic model of automata equipped with a stack memory. A *pushdown machine* is a tuple (Q, A, B, δ), where Q is set of control locations (states), A is a set of input symbols, B is a set of stack symbols, and δ is finite subset of $Q \times A \times B \times Q \times B^*$, called the set of transition rules. Note that we do not insist on the sets Q, A and B to be finite and consider only *realtime* pushdown machines, i.e. without internal transitions (also called ϵ-transitions) [21]. A *configuration* k of a pushdown machine is a pair $\langle q, \beta \rangle$ denoting the current control state $q \in Q$ and the current content of the stack $\beta \in B^*$. In denoting the stack as a string of stack symbols we assume that the topmost symbol is written first. There is a transition $\langle q, b\beta \rangle \xrightarrow{a} \langle q', \alpha\beta \rangle$ if $\langle q', \alpha \rangle \in \delta(q, a, b)$. A convenient notation is to introduce for any string $w \in A^*$ the transition relation on configurations as the least relation such that

(1) $k \xrightarrow{\epsilon} k$
(2) $k \xrightarrow{aw} k'$ if and only if $k \xrightarrow{a} k''$ and $k'' \xrightarrow{w} k'$.

A *pushdown automaton* (PDA) is a pushdown machine together with an initial configuration k_0 and a set K of accepting configurations. The sets of accepting configurations usually considered are (1) the set $F \times B^*$, where $F \subseteq Q$ is called the set of accepting states, or (2) $Q \times \{\epsilon\}$, but also (3) $F \times \{\epsilon\}$ for $F \subseteq Q$, or (4) $Q \times B'B^*$ for B' a subset of B. A word $w \in A^*$ is said to be accepted by a PDA $(Q, A, B, \delta, k_0, K)$ if $k_0 \xrightarrow{w} k$ for some $k \in K$. A PDA with accepting configurations as in (1) is said to be with accepting states, whereas, when they are as in (2) then the PDA is said to be accepting by empty stack. They both

accept exactly proper context free languages (i.e. context free languages without the empty word) [3].

Computations in a pushdown machine are generally non-deterministic and can cause a change in the control state of the automaton as well as in its stack. For this reason we will model the effects of the computations by means of the so-called *non-deterministic side-effect* monad [5]. For a set of states S, let T be the functor $\mathcal{P}_\omega(- \times S)^S$. It is a monad when equipped with the unit $\eta_X \colon X \to T(X)$, defined by $\eta(x)(s) = \{\langle x, s\rangle\}$, and the multiplication $\mu_X \colon T(T(X)) \to T(X)$ given by

$$\mu_X(k)(s) = \bigcup_{\langle c, s'\rangle \in k(s)} c(s')$$

Note that, for a function $f \colon X \to T(Y)$, the extension $f^\sharp \colon T(X) \to T(Y)$ is defined by

$$f^\sharp(c)(s) = \bigcup_{\langle x', s'\rangle \in c(s)} f(x')(s')\,.$$

Examples of algebras for this monad are $T(1) = \mathcal{P}_\omega(S)^S$ and 2^S. The latter can in fact be obtained as a quotient of the former by equating those functions $k_1, k_2 \colon S \to \mathcal{P}_\omega(S)$ such that for all $s \in S$, $k_1(s) = \emptyset$ if and only if $k_2(s) = \emptyset$.

Every pushdown machine (Q, A, B, δ) together with a set of accepting configurations K induces a function $\langle o, t\rangle \colon Q \to FTQ$ where F is the functor $2^{B^*} \times id^A$ and T is the monad defined above specialized for $S = B^*$ (intuitively, side effects in a pushdown machine are changes in its stack). The functions $o \colon Q \to 2^{B^*}$ and $t \colon Q \to \mathcal{P}_\omega(Q \times B^*)^{B^* A}$ are defined as

$$\begin{array}{lcl} o(q)(\beta) & = & 1 \text{ if and only if } \langle q, \beta\rangle \in K \\ t(q)(a)(\epsilon) & = & \emptyset \\ t(q)(a)(b\beta) & = & \{\langle q', \alpha\beta\rangle \mid \langle q', \alpha\rangle \in \delta(q, a, b)\} \end{array}$$

The transition function t describes the steps between PDA configurations and it is specified in terms of the transition instructions δ of the original machine.

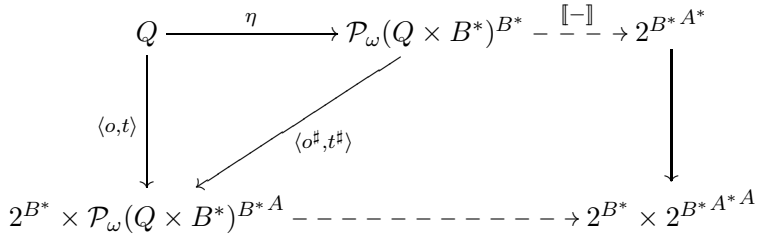
From the above is clear that not every function $\langle o, t\rangle \colon Q \to FTQ$ defines a pushdown machine with accepting configurations, as, for example, $t(q)$ may depend on the whole stack β and not just on the top element b. Therefore we restrict our attention to consider functions $\langle o, t\rangle \colon Q \to FTQ$ such that

(1) $t(q)(a)(\epsilon) = \emptyset$
(2) $t(q)(a)(b\beta) = \{\langle q', \alpha\beta\rangle \mid \langle q', \alpha\rangle \in t(q)(a)(b)\}$,

Every $\langle o, t\rangle$ satisfying (1) and (2) above defines the pushdown machine (Q, A, B, δ) with $\delta(q, a, b) = t(q)(a)(b)$ and with accepting configuration $K = \{\langle q, \beta\rangle | o(q)(\beta) = 1\}$. The first condition is asserting that a machine is in a deadlock configuration when the stack is empty, while the last condition ensures that transition steps depend only on the control state and the top element of the stack. For this reason we will write $q \xrightarrow{a, b|\alpha} q'$ for $\langle q', \alpha\beta\rangle \in t(q)(a)(b)$ indicating that the pushdown machine in the state q by reading an input symbol a and popping b off the stack, can move to a control state q' pushing the string $\alpha \in B^*$ on the current stack (here denoted by β).

Similarly to what we have shown in the examples of structured Moore automata, for every function $\langle o, t\rangle \colon Q \to FTQ$ there is a unique F-coalgebra map $[\![-]\!] \colon T(Q) \to 2^{B^* A^*}$, which is also a T-algebra homomorphism. It is defined for all $c \in \mathcal{P}_\omega(Q \times B^*)^{B^*}$ and $\beta \in B^*$

as

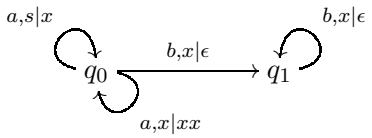


$$[\![\eta(q)]\!](\epsilon) = o(q)$$
$$[\![\eta(q)]\!](aw) = [\![\lambda\beta.t(q)(a)(\beta)]\!](w)$$
$$[\![c]\!](\beta) = \bigcup_{\langle q,\alpha\rangle \in c(\beta)} [\![\eta(q)]\!](\alpha)\,.$$

We then have that a word $w \in A^*$ is *accepted* by the PDA $(Q, A, B, \delta, k_0, K)$ with $k_0 = \langle q, \beta\rangle$ if and only if $[\![\eta(q)]\!](w)(\beta) = 1$.

The above definition implies that for a given word $w \in A^*$ we can decide if it is accepted by $\langle o, t\rangle \colon Q \to FTQ$ from an initial configuration $k_0 = \langle q, \beta\rangle$ in exactly $|w|$ steps (assuming there is a procedure to decide whether $o(q)(\beta) = 1$). As a consequence, we cannot use structured Moore automata to model Turing machines, for which the halting problem is undecidable: in general terms, for Turing machines, we would need internal transitions that do not consume input symbols.

We conclude with an example of our construction using a pushdown machine with control states $Q = \{q_0, q_1\}$, over an input alphabet $A = \{a, b\}$ and using stack symbols $B = \{x, s\}$. The transitions rules δ are given below:



We take $K = \{\langle q_0, \epsilon\rangle, \langle q_1, \epsilon\rangle\}$, meaning that $o(q_0)(\epsilon) = 1$, $o(q_1)(\epsilon) = 1$ and $o(q_i)(\beta) = 0$ in all other cases. By considering $k_0 = \langle q_0, s\rangle$ as initial configuration, we then have

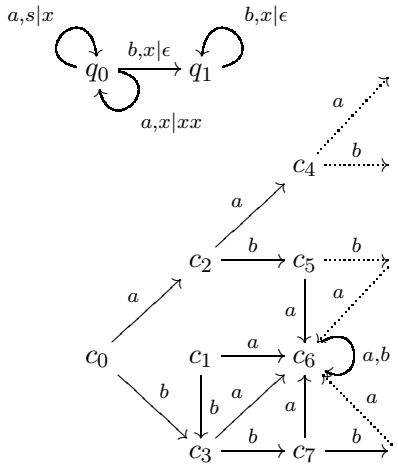
$$[\![\eta(q_0)]\!](\epsilon)(s) = o(q_0)(s) = 0$$

meaning that the empty word is not accepted by the PDA $(Q, A, B, \delta, k_0, K)$. However, the word ab is accepted:

$$\begin{aligned}
[\![\eta(q_0)]\!](ab)(s) &= [\![\lambda\beta.t(q_0)(a)(\beta)]\!](b)(s) \\
&= \bigcup_{\langle p,\beta\rangle \in t(q_0)(a)(s)} [\![\eta(p)]\!](b)(\beta) \\
&= [\![\eta(q_1)]\!](b)(x) \\
&= [\![\lambda\beta.t(q_1)(b)(\beta)]\!](\epsilon)(x) \\
&= \bigcup_{\langle p,\beta\rangle \in t(q_1)(b)(x)} [\![\eta(p)]\!](\epsilon)(\beta) \\
&= [\![\eta(q_1)]\!](\epsilon)(\epsilon) \\
&= o(q_1)(\epsilon) \\
&= 1\,.
\end{aligned}$$

In fact, the language accepted by the above pushdown automaton is $\{a^n b^n \mid n \geq 1\}$. The structured states $c_i \in TQ$, their transitions and their outputs of (part of) the associated Moore automaton are given in Figure 1.

Context-free grammars generating proper languages (i.e. not containing the empty word ϵ) are equivalent to realtime PDA's [11, 13, 42]. Given an input alphabet A, and a set



$$o^\sharp(c_0) = o^\sharp(c_1) = \lambda\beta. \begin{cases} 1 & \text{if } \beta = \epsilon \\ 0 & \text{otherwise} \end{cases}$$

$$o^\sharp(c_2) = o^\sharp(c_4) = o^\sharp(c_6) = \lambda\beta.0$$

$$o^\sharp(c_3) = o^\sharp(c_5) = \lambda\beta. \begin{cases} 1 & \text{if } \beta = x \\ 0 & \text{otherwise} \end{cases}$$

$$o^\sharp(c_7) = \lambda\beta. \begin{cases} 1 & \text{if } \beta = xxs \\ 0 & \text{otherwise} \end{cases}$$

$$c_0 = \eta(q_0) \qquad c_1 = \eta(q_1)$$

$$c_2 = \lambda\beta. \begin{cases} \{\langle q_0, x\beta'\rangle\} & \text{if } \beta = s\beta' \\ \{\langle q_0, xx\beta'\rangle\} & \text{if } \beta = x\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

$$c_3 = \lambda\beta. \begin{cases} \{\langle q_1, \beta'\rangle\} & \text{if } \beta = x\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

$$c_4 = \lambda\beta. \begin{cases} \{\langle q_0, xx\beta'\rangle\} & \text{if } \beta = s\beta' \\ \{\langle q_0, xxx\beta'\rangle\} & \text{if } \beta = x\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

$$c_5 = \lambda\beta. \begin{cases} \{\langle q_1, \beta'\rangle\} & \text{if } \beta = s\beta' \\ \{\langle q_1, x\beta'\rangle\} & \text{if } \beta = x\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

$$c_6 = \lambda\beta.\emptyset$$

$$c_7 = \lambda\beta. \begin{cases} \{\langle q_1, \beta'\rangle\} & \text{if } \beta = xx\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

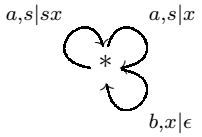
Figure 1: The structured states $c_i \in TQ$, their transitions and their output of (part of) the Moore automaton associated to the PDA $(Q, A, B, \delta, k_0, K)$ where $Q = \{q_0, q_1\}$, $A = \{a, b\}$, $B = \{x, s\}$, δ is depicted on the left top, $k_0 = \langle q_0, s\rangle$ and $K = \{\langle q_0, \epsilon\rangle, \langle q_1, \epsilon\rangle\}$.

of variables B, let $G = (A, B, s, P)$ be a context-free grammar in Greibach normal form [15], i.e. with productions in P of the form $b \to a\alpha$ with $b \in B$, $a \in A$ and $\alpha \in B^*$. We can construct a function $\langle o, t\rangle \colon 1 \to FT1$ (where $1 = \{*\}$) by setting

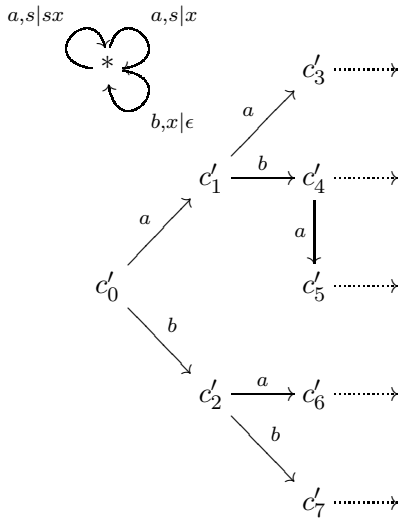
$$o(*)(\beta) = 1 \text{ if and only if } \beta = \epsilon \qquad \text{and} \qquad t(*)(a)(b\beta) = \{\langle *, \alpha\beta\rangle \mid b \to a\alpha \in P\}\,.$$

Clearly this function satisfies conditions (1) and (2) above, and thus, together with the initial configuration $\langle *, s\rangle$ defines a PDA. Furthermore, $[\![\eta(*)]\!](w)(s) = 1$ if and only if there exists a derivation for $w \in A^*$ in the grammar G.

As an example, let us consider the grammar $(\{a, b\}, \{s, x\}, s, P)$ with productions $P = \{s \to asx, s \to ax, x \to b\}$ generating the language $\{a^n b^n \mid n \geq 1\}$. The associated coalgebra $\langle o, t\rangle \colon 1 \to FT1$ is given by



with $o(*)(\beta) = 1$ iff $\beta = \epsilon$



$$c'_0 = \eta(*)$$

$$c'_1 = \lambda\beta.\begin{cases} \{\langle *, sx\beta'\rangle, \langle *, x\beta'\rangle\} & \text{if } \beta = s\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

$$c'_2 = \lambda\beta.\begin{cases} \{\langle *, \beta'\rangle\} & \text{if } \beta = x\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

$$c'_3 = \lambda\beta.\begin{cases} \{\langle *, sxx\beta'\rangle, \langle *, x\beta'\rangle\} & \text{if } \beta = s\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

$$c'_4 = \lambda\beta.\begin{cases} \{\langle *, \beta'\rangle\} & \text{if } \beta = s\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

$$c'_5 = \lambda\beta.\begin{cases} \{\langle *, sx\beta'\rangle, \langle *, x\beta'\rangle\} & \text{if } \beta = ss\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

$$o^\sharp(c'_0) = \lambda\beta.\begin{cases} 1 & \text{if } \beta = \epsilon \\ 0 & \text{otherwise} \end{cases}$$

$$c'_6 = \lambda\beta.\begin{cases} \{\langle *, sx\beta'\rangle, \langle *, x\beta'\rangle\} & \text{if } \beta = xs\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

$$o^\sharp(c'_2) = \lambda\beta.\begin{cases} 1 & \text{if } \beta = x \\ 0 & \text{otherwise} \end{cases}$$

$$c'_7 = \lambda\beta.\begin{cases} \{\langle *, \beta'\rangle\} & \text{if } \beta = xx\beta' \\ \emptyset & \text{otherwise} \end{cases}$$

$$o^\sharp(c'_1) = o^\sharp(c'_3) = o^\sharp(c'_5) = o^\sharp(c'_6) = \lambda\beta.0$$

$$o^\sharp(c'_7) = \lambda\beta.\begin{cases} 1 & \text{if } \beta = xx \\ 0 & \text{otherwise} \end{cases}$$

$$o^\sharp(c'_4) = \lambda\beta.\begin{cases} 1 & \text{if } \beta = s \\ 0 & \text{otherwise} \end{cases}$$

Figure 2: The structured states $c_i \in TQ$, their transitions and their output of (part of) the Moore automaton associated to the PDA $(Q, A, B, \delta, k_0, K)$ where $Q = \{*\}$, $A = \{a, b\}$, $B = \{x, s\}$, δ is depicted on the left top, $k_0 = \langle *, s\rangle$ and $K = \{\langle *, \epsilon\rangle\}$.

Even if the language accepted by the above PDA is the same as the one accepted by the PDA in the previous example (i.e., $[\![\eta(*)]\!](w)(s) = [\![\eta(q_0)]\!](w)(s)$ for all $w \in A^*$), the two associated Moore automaton are not in $\approx_F^T$ (that is $[\![\eta(*)]\!] \neq [\![\eta(q_0)]\!]$). In fact, the Moore automaton associated to the above coalgebra (see below) accepts the string *abab* when starting from the configuration $\langle *, ss\rangle$, while the one in the previous example does not (in symbols, $[\![\eta(*)]\!](abab)(ss) = 1$ while $[\![\eta(q_0)]\!](abab)(ss) = 0$).

The above characterization of context free languages over an alphabet A is different and complementary to the coalgebraic account of context-free languages presented in [44]. The latter, in fact, uses the functor $D(X) = 2 \times X^A$ for deterministic automata (instead of the Moore automata with output in 2^{B^*} above, for B a set of variables), and the idempotent semiring monad $T(X) = \mathcal{P}_\omega((X + A)^*)$ (instead of our side effect monad) to study different

but equivalent ways to present context-free languages: using grammars, behavioural differential equations and generalized regular expressions in which the Kleene star is replaced by a unique fixed point operator.

4. Coalgebras and $\mathbf{T}$-Algebras

In the previous section we presented a framework, parameterized by a functor F and a monad $\mathbf{T}$, in which systems of type FT (that is, FT-coalgebras) can be studied using a novel equivalence $\approx_F^T$ instead of the classical $\sim_{FT}$. The only requirement we imposed was that $FT(X)$ has to be a $\mathbf{T}$-algebra.

In this section, we will present functors F for which the requirement of $FT(X)$ being a $\mathbf{T}$-algebra is guaranteed because they can be *lifted* to a functor F^* on $\mathbf{T}$-algebra. For these functors, the equivalence $\approx_F^T$ coincides with $\sim_{F^*}$. In other words, working on FT-coalgebras in $\mathbf{Set}$ under the novel $\approx_F^T$ equivalence is the same as working on F^*-coalgebras on $\mathbf{T}$-algebras under the ordinary $\sim_{F^*}$ equivalence. Next, we will prove that for this class of functors and an arbitrary monad $\mathbf{T}$ the equivalence $\sim_{FT}$ is contained in $\approx_F^T$. Instantiating this result for our first motivating example of non-deterministic automata will yield the well known fact that bisimilarity implies trace equivalence.

Let $\mathbf{T}$ be a monad. An endofunctor $F^*\colon \mathbf{Set}^{\mathbf{T}} \to \mathbf{Set}^{\mathbf{T}}$ is said to be the $\mathbf{T}$-*algebra lifting* of a functor $F\colon \mathbf{Set} \to \mathbf{Set}$ if the following square commutes[1]:

$$\begin{array}{ccc} \mathbf{Set}^{\mathbf{T}} & \xrightarrow{F^*} & \mathbf{Set}^{\mathbf{T}} \\ {\scriptstyle U^{\mathbf{T}}}\downarrow & & \downarrow{\scriptstyle U^{\mathbf{T}}} \\ \mathbf{Set} & \xrightarrow[F]{} & \mathbf{Set} \end{array}$$

If the functor F has a $\mathbf{T}$-algebra lifting F^* then $FT(X)$ is the carrier of the algebra $F^*(T(X), \mu)$. Functors that have a $\mathbf{T}$-algebra lifting are given, for example, by those endofunctors on $\mathbf{Set}$ constructed inductively by the following grammar

$$F ::= Id \mid B \mid F \times F \mid F^A \mid TG$$

where A is an arbitrary set, B is the constant functor mapping every set X to the carrier of a $\mathbf{T}$-algebra (B, h), and G is an arbitrary functor. Since the forgetful functor $U^{\mathbf{T}}\colon \mathbf{Set}^{\mathbf{T}} \to \mathbf{Set}$ creates and preserves limits, both $F_1 \times F_2$ and F^A have a $\mathbf{T}$-algebra lifting if F, F_1, and F_2 have. Finally, TG has a $\mathbf{T}$-algebra lifting for every endofunctor G given by the assignment $(X, h) \mapsto (TGX, \mu_{GX})$. Note that we do not allow taking coproducts in the above grammar, because coproducts of $\mathbf{T}$-algebras are not preserved in general by the forgetful functor $U^{\mathbf{T}}$. Instead, one could resort to extending the grammar with the carrier of the coproduct taken directly in $\mathbf{Set}^{\mathbf{T}}$. For instance, if $\mathbf{T}$ is the (finite) powerset monad, then we could extend the above grammar with the functor $F_1 \oplus F_2 = F_1 + F_2 + \{\top, \bot\}$.

All the functors of the examples in Sections 2 and 3, as well as those in Section 5, can be generated by the above grammar and, therefore, they have a $\mathbf{T}$-algebra lifting.

Now, let F be a functor with a $\mathbf{T}$-algebra lifting and for which a final coalgebra Ω exists. If Ω can be constructed as the limit of the final sequence (for example assuming the functor

[1]This is equivalent to the existence of a distributive law $\lambda\colon TF \Rightarrow FT$ [24].

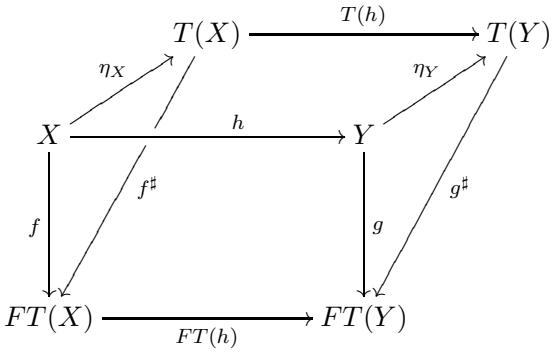
accessible [1]), then, because the forgetful functor $U^{\mathbf{T}}\colon \mathbf{Set}^{\mathbf{T}} \to \mathbf{Set}$ preserves and creates limits, Ω is the carrier of a $\mathbf{T}$-algebra, and it is the final coalgebra of the lifted functor F^*. Further, for any FT-coalgebra $f\colon X \to FT(X)$, the unique F-coalgebra homomorphism $[\![-]\!]$ as in diagram (3.1) is a T-algebra homomorphism between $T(X)$ and Ω. Conversely, the carrier of the final F^*-coalgebra (in $\mathbf{Set}^{\mathbf{T}}$) is the final F-coalgebra (in $\mathbf{Set}$).

Intuitively, the above means that for an accessible functor F with a $\mathbf{T}$-algebra lifting F^*, F^*-equivalence in $\mathbf{Set}^{\mathbf{T}}$ coincides with F-equivalence with respect to $\mathbf{T}$ in $\mathbf{Set}$. The latter equivalence is coarser than the FT-equivalence in $\mathbf{Set}$, as stated in the following theorem.

Theorem 4.1. *Let* $\mathbf{T}$ *be a monad. If* F *is an endofunctor on* $\mathbf{Set}$ *for which a final coalgebra exists and with a* $\mathbf{T}$*-algebra lifting, then* $\sim_{FT}$ *implies* $\approx_F^T$.

Proof. We first show that there exists a functor from the category of FT-coalgebras to the category of F-coalgebras.

This functor maps each FT-coalgebra (X, f) into the F-coalgebra $(T(X), f^\sharp)$ and each FT-homomorphism $h\colon (X, f) \to (Y, g)$ into the F-homomorphism $T(h)\colon (T(X), f^\sharp) \to (T(Y), g^\sharp)$. In order to prove that this is a functor we just have to show that $T(h)$ is an F-homomorphism (i.e., the backward face of the following diagram commutes).



Note that the front face of the above diagram commutes because h is an FT-homomorphism. Also the top face commutes because η is a natural transformation. Thus

$$FT(h) \circ f^\sharp \circ \eta_X = FT(h) \circ f = g \circ h$$

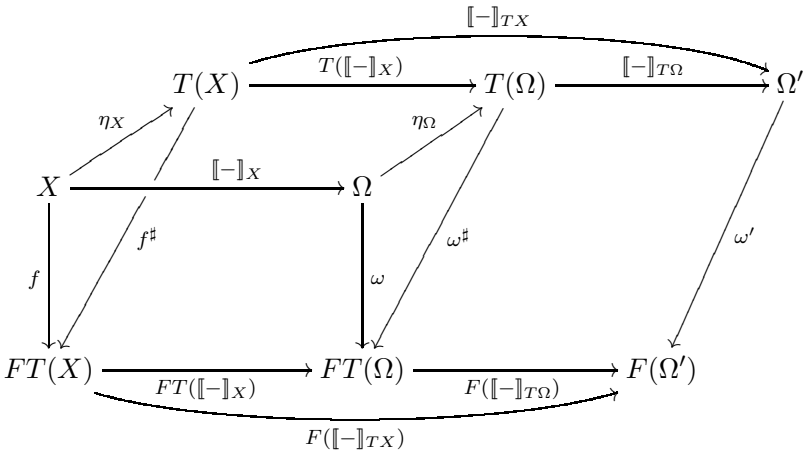
and also

$$g^\sharp \circ T(h) \circ \eta_X = g^\sharp \circ \eta_Y \circ h = g \circ h.$$

Since η is the unit of the adjunction, then there exists a unique $j^\sharp\colon T(X) \to FT(Y)$ in $\mathbf{Set}^{\mathbf{T}}$ such that $g \circ h = j^\sharp \circ \eta_X$. Since both $FT(h) \circ f^\sharp$ and $g^\sharp \circ T(h)$ are (by construction) morphisms in $\mathbf{Set}^{\mathbf{T}}$, then $FT(h) \circ f^\sharp = g^\sharp \circ T(h)$.

Let (X, f) and (Y, g) be two FT-coalgebras and $[\![-]\!]_X$ and $[\![-]\!]_Y$ their morphisms into the final FT-coalgebra (Ω, ω). Let $(T(X), f^\sharp)$, $(T(Y), g^\sharp)$ and $(T(\Omega), \omega^\sharp)$ be the corresponding F-coalgebras and $[\![-]\!]_{TX}$, $[\![-]\!]_{TY}$ and $[\![-]\!]_{T\Omega}$ their morphisms into the final F-coalgebra (Ω', ω').

Since $T([\![-]\!]_X)\colon (T(X), f^\sharp) \to (T(\Omega), \omega^\sharp)$ is an F-homomorphism, then by uniqueness, $[\![-]\!]_{TX} = [\![-]\!]_{T\Omega} \circ T([\![-]\!]_X)$.



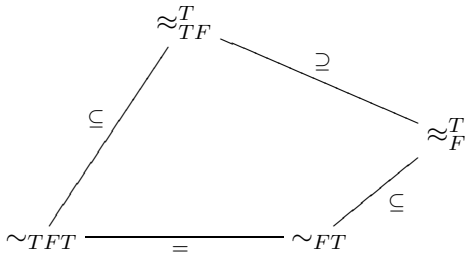
With the same proof, we obtain $[\![-]\!]_{TY} = [\![-]\!]_{T\Omega} \circ T([\![-]\!]_Y)$.

Recall that for all $x \in X$ and $y \in Y$, by definition, $x \sim_{FT} y$ iff $[\![x]\!]_X = [\![y]\!]_Y$ and $x \approx_F^T y$ iff $[\![\eta_X(x)]\!]_{TX} = [\![\eta_Y(y)]\!]_{TY}$.

Suppose that $[\![x]\!]_X = [\![y]\!]_Y$. Then, $T([\![\eta_X(x)]\!]_X) = \eta_\Omega \circ [\![x]\!]_X = \eta_\Omega \circ [\![y]\!]_Y = T([\![\eta_Y(y)]\!]_Y)$ and, finally, $[\![\eta_X(x)]\!]_{TX} = [\![-]\!]_{T\Omega} \circ T([\![\eta_X(x)]\!]_X) = [\![-]\!]_{T\Omega} \circ T([\![\eta_Y(y)]\!]_Y) = [\![\eta_Y(y)]\!]_{TY}$. □

The above theorem instantiates to the well-known facts: for NDA, where $F(X) = 2 \times X^A$ and $T = \mathcal{P}_\omega$, that bisimilarity implies language equivalence; for partial automata, where $F(X) = 2 \times X^A$ and $T = 1 + -$, that equivalence of pairs of languages, consisting of defined paths and accepted words, implies equivalence of accepted words; for probabilistic automata, where $F(X) = [0,1] \times X^A$ and $T = \mathcal{D}_\omega$, that probabilistic bisimilarity implies probabilistic/weighted language equivalence. Note that, in general, the above inclusion is strict.

Remark. Let (X, f) be an FT-coalgebra for a monad $\mathbf{T}$ and a functor F. If $\eta\colon id \Rightarrow T$ is pointwise injective, then $\sim_{FT}$ on the FT-coalgebra (X, f) coincides with $\sim_{TFT}$ on the extended TFT-coalgebra $(X, \eta_{FT(X)} \circ f)$ [37, 4]. If moreover F has a $\mathbf{T}$-algebra lifting then, by the above theorem (on the extended TFT-coalgebra), $\sim_{TFT}$ implies $\approx_{TF}^T$. Combining the two implications, it follows that hat $\sim_{FT}$ on the FT-coalgebra (X, f) implies $\approx_{TF}^T$ on the extended TFT-coalgebra $(X, \eta_{FT(X)} \circ f)$. Finally, under the assumption that F has a $\mathbf{T}$-algebra lifting, we also have that $\approx_F^T$ the FT-coalgebra (X, f) implies $\approx_{TF}^T$ on the extended TFT-coalgebra $(X, \eta_{FT(X)} \circ f)$. This yields the following hierarchy of equivalences.



5. Beyond Bisimilarity and Traces

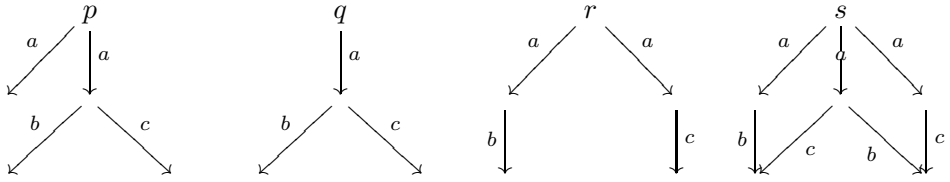
The operational semantics of interactive systems is usually specified by labeled transition systems (LTS's). The denotational semantics is given in terms of behavioural equivalences, which depend the amount of branching structure considered. Bisimilarity (full branching) is sometimes considered too strict, while trace equivalence (no branching) is often considered too coarse. The *linear time / branching time spectrum* [14] shows a taxonomy of many interesting equivalences lying in between bisimilarity and traces.

Labeled transition system are coalgebras for the functor $\mathcal{P}_\omega(Id)^A$ and the coalgebraic equivalence $\sim_{\mathcal{P}_\omega(Id)^A}$ coincides with the standard notion of Park-Milner bisimilarity. In [35], it is shown a coalgebraic characterization of traces semantics (for LTS's) employing Kleisli categories. More recently, [33] have provided a characterization of trace, failure and ready semantics by mean of "behaviour objects". Another coalgebraic approach [26] relies on "test-suite" that, intuitively, are fragments of Hennessy-Milner logic. In this section, we show that (finite) trace equivalence [20], complete trace equivalence [14], failures [9] and ready semantics [34] can be seen as special cases of $\approx_F^T$.

Before introducing these semantics, we fix some notations. A labeled transition system is a pair (X, δ) where X is a set of states and $\delta\colon X \to \mathcal{P}_\omega(X)^A$ is a function assigning to each state $x \in X$ and to each label $a \in A$ a finite set of possible successors states: $x \xrightarrow{a} y$ means that $y \in \delta(x)(a)$. Given a word $w \in A^*$, we write $x \xrightarrow{w} y$ for $x \xrightarrow{a_1} \dots \xrightarrow{a_n} y$ and $w = a_1 \dots a_n$. When $w = \epsilon$, $x \xrightarrow{\epsilon} y$ iff $y = x$. For a function $\varphi \in \mathcal{P}_\omega(X)^A$, $I(\varphi)$ denotes the set of all labels "enabled" by φ, i.e., $\{a \in A \mid \varphi(a) \neq \emptyset\}$, while $\mathit{Fail}(\varphi)$ denotes the set $\{Z \subseteq A \mid Z \cap I(\varphi) = \emptyset\}$.

Let $\langle X, \delta\rangle$ be a LTS and $x \in X$ be a state. A *trace* of x is a word $w \in A^*$ such that $x \xrightarrow{w} y$ for some y. A trace w of x is *complete* if $x \xrightarrow{w} y$ and y stops, i.e., $I(\delta(y)) = \emptyset$. A *failure pair* of x is a pair $\langle w, Z\rangle \in A^* \times \mathcal{P}_\omega(A)$ such that $x \xrightarrow{w} y$ and $Z \in \mathit{Fail}(\delta(y))$. A *ready pair* of x is a pair $\langle w, Z\rangle \in A^* \times \mathcal{P}_\omega(A)$ such that $x \xrightarrow{w} y$ and $Z = I(\delta(y))$. We use $\mathcal{T}(x)$, $\mathcal{CT}(x)$, $\mathcal{F}(x)$ and $\mathcal{R}(x)$ to denote, respectively, the set of all traces, complete traces, failure pairs and ready pairs of x. For $\mathcal{I}$ ranging over $\mathcal{T}, \mathcal{CT}, \mathcal{F}$ and $\mathcal{R}$, two states x and y are $\mathcal{I}$-equivalent iff $\mathcal{I}(x) = \mathcal{I}(y)$.

For an example, consider the following transition systems labeled over $A = \{a, b, c\}$. They are all trace equivalent because their traces are a, ab, ac. The trace a is also complete for p, but not for the others. Only r and s are failure equivalent, since $\langle a, \{bc\}\rangle$ is a failure pair only of p, while $\langle a, \{b\}\rangle$ and $\langle a, \{c\}\rangle$ are failure pairs of p, r and s, but not of q. Finally they are all ready different, since $\langle a, \emptyset\rangle$ is a ready pair only of p, $\langle a, \{b, c\}\rangle$ is a ready pair of q and s but not of r, and $\langle a, \{b\}\rangle$ and $\langle a, \{c\}\rangle$ are ready pairs only of r and s.



We can now show that these equivalences are instances of $\approx_F^T$. We first show ready equivalence in details and then, briefly, the others.

Take $T = \mathcal{P}_\omega$ and $F = \mathcal{P}_\omega(\mathcal{P}_\omega(A)) \times id^A$. For each set X, consider the function $\pi^{\mathcal{R}}_X \colon \mathcal{P}_\omega(X)^A \to FT(X)$ defined for all $\varphi \in \mathcal{P}_\omega(X)^A$ by

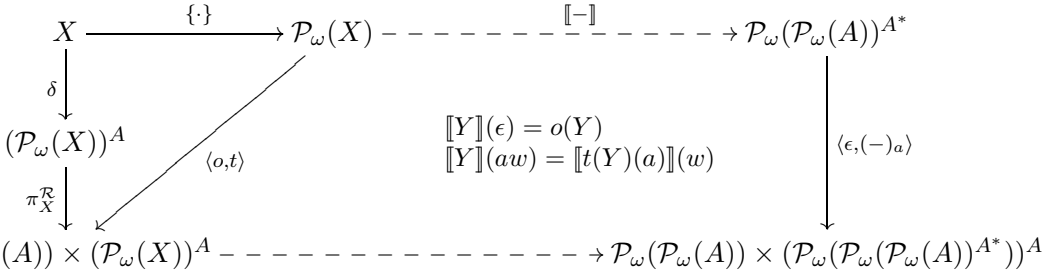
$$\pi^{\mathcal{R}}_X(\varphi) = \langle \{I(\varphi)\}, \varphi \rangle.$$

This function allows to transform each LTS (X, δ) into the FT-coalgebra $(X, \pi^{\mathcal{R}}_X \circ \delta)$. The latter has the same transitions of $\langle X, \delta \rangle$, but each state x is "decorated" with the set $\{I(\varphi)\}$.

Now, by employing the powerset construction, we transform $\langle X, \pi^{\mathcal{R}}_X \circ \delta \rangle$ into the F-coalgebra $(\mathcal{P}_\omega(X), \langle o, t \rangle)$, where, for all $Y \in \mathcal{P}_\omega(X)$, $a \in A$, the functions $o \colon \mathcal{P}_\omega(X) \to \mathcal{P}_\omega(\mathcal{P}_\omega(A))$ and $t \colon \mathcal{P}_\omega(X) \to \mathcal{P}_\omega(X)^A$ are

$$o(Y) = \bigcup_{y \in Y} \{I(\delta(y))\} \qquad t(Y)(a) = \bigcup_{y \in Y} \delta(y)(a).$$

The final F-coalgebra is $(\mathcal{P}_\omega(\mathcal{P}_\omega(A))^{A^*}, \langle \epsilon, (-)_a \rangle)$ where $\langle \epsilon, (-)_a \rangle$ is defined as usual.



Summarizing, the final map $[\![-]\!] \colon \mathcal{P}_\omega(X) \to \mathcal{P}_\omega(\mathcal{P}_\omega(A))^{A^*}$ maps each $\{x\}$ into a function assigning to each word w, the set $\{Z \subseteq A \mid x \overset{w}{\to} y \text{ and } Z = I(\delta(y))\}$. In other terms, $Z \in [\![\{x\}]\!](w)$ iff $\langle w, Z \rangle \in \mathcal{R}(x)$.

For the state s depicted above, $[\![\{s\}]\!](\epsilon) = \{\{a\}\}$, $[\![\{s\}]\!](a) = \{\{b\}, \{b, c\}, \{c\}\}$, $[\![\{s\}]\!](ab) = [\![\{s\}]\!](ac) = \{\emptyset\}$ and for all the other words w, $[\![\{s\}]\!](w) = \emptyset$.

The other semantics can be characterized in the same way, by choosing different functors F and different functions $\pi_X \colon \mathcal{P}_\omega(X)^A \to FT$.

For failure semantics, take the same functor as for the ready semantics, that is $F = \mathcal{P}_\omega(\mathcal{P}_\omega(A)) \times id^A$ and a new function $\pi^{\mathcal{F}}_X \colon \mathcal{P}_\omega(X)^A \to FT(X)$ defined $\forall \varphi \in \mathcal{P}_\omega(X)^A$ by

$$\pi^{\mathcal{F}}_X(\varphi) = \langle \mathit{Fail}(\varphi), \varphi \rangle.$$

The FT-coalgebra $(X, \pi^{\mathcal{F}}_X \circ \delta)$ has the same transitions of the LTS $\langle X, \delta \rangle$, but each state x is "decorated" with the set $\mathit{Fail}(\varphi)$.

For both trace and complete trace equivalence, take $F = 2 \times id^A$ (as for NDA). For trace equivalence, $\pi^{\mathcal{T}}_X \colon \mathcal{P}_\omega(X)^A \to FT(X)$ maps $\varphi \in \mathcal{P}_\omega(X)^A$ into $\langle 1, \varphi \rangle$. Intuitively, $(X, \pi^{\mathcal{T}}_X \circ \delta)$ is an NDA where all the states are accepting. For complete traces, $\pi^{\mathcal{CT}}_X \colon \mathcal{P}_\omega(X)^A \to FT(X)$ maps φ in $\langle 1, \varphi \rangle$ if $I(\varphi) = \emptyset$ (and in $\langle 0, \varphi \rangle$ otherwise).

By taking $T = \mathcal{D}_\omega$ instead of $T = \mathcal{P}_\omega$, we hope to be able to characterize probabilistic trace, complete trace, ready and failure as defined in [25].

6. Discussion

In this paper, we lifted the powerset construction on automata to the more general framework of FT-coalgebras. Our results lead to a uniform treatment of several kinds of existing and new variations of automata (that is, FT-coalgebras) by an algebraic structuring of their state space through a monad T. We showed as examples partial Mealy machines, structured Moore automata, nondeterministic, partial and probabilistic automata. Furthermore, we have presented an interesting coalgebraic characterization of pushdown automata and showed how several behavioural equivalences stemming from concurrency theory can be retrieved from the general framework. It is worth mentioning that the framework instantiates to many other examples, among which are *weighted automata* [41]. These are simply structured Moore automata for $B = 1$ and $\mathbf{T} = \mathbb{S}^-_\omega$ (for a semiring $\mathbb{S}$) [16]. It is easy to see that $\sim_{FT}$ coincides with weighted bisimilarity [10], while $\approx^T_F$ coincides with weighted language equivalence [41].

Some of the aforementioned examples can also be coalgebraically characterized in the framework of [19, 18]. There, instead of considering FT-coalgebras on $\mathbf{Set}$ and F^*-coalgebras on $\mathbf{Set}^{\mathbf{T}}$ (the Eilenberg-Moore category), TG-coalgebras on $\mathbf{Set}$ and $\overline{G}$-coalgebras on $\mathbf{Set}_{\mathbf{T}}$ (the *Kleisli* category) are studied. The main theorem of [19] states that under certain assumptions, the initial G-algebra is the final $\overline{G}$-coalgebra that characterizes (generalized) trace equivalence. The exact relationship between these two approaches has been studied in [23] (and, indirectly, it could be deduced from [6] and [27]). It is worth to remark that many of our examples do not fit the framework in [19]: for instance, the exception, the side effect, the full-probability and the interactive output monads do not fulfill their requirements (the first three do not have a bottom element and the latter is not commutative). Moreover, we also note that the example of partial Mealy machines is not purely trace-like, as all the examples in [19].

The idea of using monads for modeling automata with non-determinism, probabilism or side-effects dates back to the "λ-machines" of [2] that, rather than coalgebras, rely on algebras. More precisely, the dynamic of a λ-machine is a morphism $\delta\colon FX \to TX$, where F is a functor and T is a monad (for instance the transitions of T-structured Moore automata are a function $\delta\colon X \times A \to TX$ mapping a state and an input symbol into an element of TX). Analogously to our approach, each λ-machine induces an "implicit λ-machine" having TX as state space. Many examples of this paper (like Moore automata) can be seen as λ-machines, but those systems that are essentially coalgebraic (like Mealy machines) do not fit the framework in [2].

There are several directions for future research. On the one hand, we will try to exploit *F-bisimulations up to T* [29, 30] as a sound and complete proof technique for $\approx^T_F$. On the other hand, we would like to lift many of those coalgebraic tools that have been developed for "branching equivalences" (such as coalgebraic modal logic [12, 40] and (axiomatization for) regular expressions [8]) to work with the "linear equivalences" induced by $\approx^T_F$.

We have pursued further the applications to decorated traces and the challenging modeling of the full linear-time spectrum in a separate paper [7], work which we also plan to extend to probabilistic traces.

References

[1] J. Adámek. Free algebras and automata realization in the language of categories. *Comment. Math. Univ. Carolinae*, 15:589–602, 1974.

[2] M. Arbib, and E. Manes. Fuzzy machines in a category. *Bull. Austral. Math. Soc.*, 13:169–210, 1975.

[3] J.-M. Autebert, J. Berstel, and L. Boasson. Context-Free Languages and Push-Down Automata. In G. Rozenberg and A. Salomaa (eds.), *Handbook of Formal Languages*, Volume 1, pages 111-174. Springer-Verlag, 1997.

[4] F. Bartels. *On generalized coinduction and probabilistic specification formats*. PhD thesis, Vrije Universiteit Amsterdam, 2004.

[5] N. Benton, J. Hughes, and E. Moggi. Monads and Effects. Course notes for *APPSEM Summer School*, 2000. Available on line at `http://www.disi.unige.it/person/MoggiE/APPSEM00/BHM.ps`.

[6] A. Balan, and A. Kurz. On Coalgebras over Algebras. *Electronic Notes in Theoretical Computer Science*. 264(2): 47-62 (2010)

[7] F. Bonchi, M.M. Bonsangue, G. Caltais, J.J.M.M. Rutten, and A. Silva. Final semantics for decorated traces, In *Proceedings of MFPS*, ENTCS, Elsevier, 2012, to appear.

[8] M.M. Bonsangue, J.J.M.M. Rutten, and A. Silva. An algebra for Kripke polynomial coalgebras. In *Proceedings of 24th Annual IEEE Symposium on Logic In Computer Science (LICS 2009)*, pages 49–58. IEEE Computer Society, 2009.

[9] S.D. Brookes, C.A.R. Hoare and A.W. Roscoe. A Theory of Communicating Sequential Processes. *Journal of the ACM*, 31(3):560–599, ACM 1984.

[10] P. Buchholz. Bisimulation relations for weighted automata. *Theoretical Computer Science*, 393(1-3):109–123, Elsevier, 2008.

[11] N. Chomsky. Context Free Grammars and Pushdown Storage. *Quarterly Progress Report*, volume 65, MIT Research Laboratory in Electronics, Cambridge, MA, 1962.

[12] C. Cîrstea, A. Kurz, D. Pattinson, L. Schröder, and Y. Venema. Modal logics are coalgebraic. *Computer Journal* 54(1):31–41, Oxford University Press, 2011.

[13] R.J. Evey. Application of Pushdown Store Machines. In *Proceedings of the 1963 Fall Joint Computer Conference (AFIPS 1963)*, ACM, 1963.

[14] R.J. van Glabbeek. The Linear Time-Branching Time Spectrum. In E. Best (Ed.), *Proceedings of CONCUR 93*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer, 1990.

[15] S. Greibach. A Note on Pushdown Store Automata and Regular Systems. *Proceedings of the American Mathematical Society*, 18:263–268, American Mathematical Society 1967.

[16] H.P. Gumm and T. Schröder. Monoid-labeled transition systems. *Electronic Notes in Theoretical Computer Science*, 44(1):184–203, Elsevier 2001.

[17] H.H. Hansen. Coalgebraising subsequential transducers. *Electronic Notes in Theoretical Computer Science*, 203(5):109–129, 2008.

[18] I. Hasuo. *Tracing Anonymity with Coalgebras*. PhD thesis, Radboud University Nijmegen, 2008.

[19] I. Hasuo, B. Jacobs, and A. Sokolova. Generic trace semantics via coinduction. *Logical Methods in Computer Science*, 3(4):1–36, 2007.

[20] C. A. R. Hoare. Communicating Sequential Processes. *Communincation of the ACM.*, 21(8):666–677, ACM, 1978.

[21] J. Hopcroft, J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[22] B. Jacobs. Distributive laws for the coinductive solution of recursive equations. *Information and Computation*, 204(4): 561-587, 2006.

[23] B. Jacobs, A. Silva, and A. Sokolova. Trace Semantics via Determinization. To appear in *Proceedings of CMCS 12*, in *Lecture Notes in Computer Science*. Springer, 2012.

[24] P.T. Johnstone. Adjoint lifting theorems for categories of algebras. *Bulletin London Mathematical Society*, 7:294–297, 1975.

[25] C. Jou and S.A. Smolka. Equivalences, Congruences, and Complete Axiomatizations for Probabilistic Processes. In J. Baeten and J.W. Klop (eds), *proceedings of CONCUR '90*, volume 458 of *Lecture Notes in Computer Science*, pages 367–383, Springer, 1990.

[26] B. Klin. A coalgebraic approach to process equivalence and a coinduction principle for traces. *Electronic Notes in Theoretical Computer Science*, 106:201–218, 2004.

[27] C. Kissig, and A. Kurz. Generic Trace Logics. In *arXiv:1103.3239v1 [cs.LO]*, 2011.

[28] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.
[29] M. Lenisa. From Set-theoretic Coinduction to Coalgebraic Coinduction: some results, some problems. *Electronic Notes in Theoretical Computer Science*, 19:2–22, Elsevier, 1999.
[30] M. Lenisa, J. Power and H. Watanabe. Distributivity for endofunctors, pointed and co-pointed endofunctors, monads and comonads. *Electronic Notes in Theoretical Computer Science*, 33:230–260, Elsevier, 2000.
[31] E. Manes. *Algebraic theories. Graduate Texts in Mathematics*, 26, Springer 1976.
[32] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.
[33] L. Monteiro. A Coalgebraic Characterization of Behaviours in the Linear Time - Branching Time Spectrum. In *proceedings of the 19th International Workshop on Recent Trends in Algebraic Development Techniques (WADT 2008)*, volume 5486 of *Lecture Notes in Computer Science*, pages 128–140. Springer, 2009.
[34] E.-R. Olderog and C.A.R. Hoare. Specification-Oriented Semantics for Communicating Processes. *Acta Informaticae*, 21(1):9–66, 1986.
[35] J. Power and D. Turi. A Coalgebraic Foundation for Linear Time Semantics. *Electronic Notes in Theoretical Computer Science*, 160:305–29, 1999.
[36] M.O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963.
[37] J.J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, Elsevier, 2000.
[38] J.J.M.M. Rutten. Algebraic specification and coalgebraic synthesis of mealy automata. *Electronic Notes in Theoretical Computer Science*, 160:305–319, 2006.
[39] J.J.M.M. Rutten. Coalgebra, concurrency, and control. In R. Boel and G. Stremersch (eds.), *proceedings of the 5th Workshop on Discrete Event Systems (WODES 2000)*, pages 31–38, Kluwer, 2000.
[40] L. Schröder. Expressivity of coalgebraic modal logic: The limits and beyond. *Theoretical Computer Science*, 390(2-3):230–247, Elsevier, 2008.
[41] M.P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270, 1961.
[42] M.P. Schützenberger. On Context Free Languages and Pushdown Automata. *Information and Control*, 6:246-264, 1963.
[43] A. Silva, F. Bonchi, M. Bonsangue and J. Rutten. Generalizing the powerset construction, coalgebraically. In proceedings of *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, (FSTTCS 2010)*, volume 8 of *LIPIcs*, pages 272 – 283, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.
[44] J. Winter, M.M. Bonsangue, J.J.M.M. Rutten. Context-Free Languages, Coalgebraically. In A. Corradini, B. Klin, and C. Cirstea, (eds.), *Proceedings of 4th Int. Conference on Algebra and Coalgebra in Computer science (CALCO 2011)*, volume 6859 of *Lecture Notes in Computer Science*, pages 359-376, Springer, 2011.

Efficient Computation of Program Equivalence for Confluent Concurrent Constraint Programming (Technical Report)*

Luis F. Pino
INRIA/DGA and LIX
École Polytechnique
91128 Palaiseau, France
luis.pino@lix.polytechnique.fr

Filippo Bonchi
CNRS and ENS Lyon
Université de Lyon, LIP
69364 Lyon, France
filippo.bonchi@ens-lyon.fr

Frank D. Valencia
CNRS and LIX
École Polytechnique
91128 Palaiseau, France
frank.valencia@lix.polytechnique.fr

ABSTRACT

Concurrent Constraint Programming (ccp) is a well-established *declarative* framework from concurrency theory. Its foundations and principles e.g., semantics, proof systems, axiomatizations, have been thoroughly studied for over the last two decades. In contrast, the development of algorithms and automatic verification procedures for ccp have hitherto been far too little considered. To the best of our knowledge there is only one existing verification algorithm for the standard notion of ccp program (observational) equivalence. In this paper we first show that this verification algorithm has an *exponential-time* complexity even for programs from a representative sub-language of ccp; the *summation-free fragment* (ccp-{+}). We then significantly improve on the complexity of this algorithm by providing two alternative *polynomial-time* decision procedures for ccp-{+} program equivalence. Each of these two procedures has an advantage over the other. One has a better time complexity. The other can be easily adapted for the full language of ccp to produce significant state space reductions. The relevance of both procedures derives from the importance of ccp-{+}. This fragment, which has been the subject of many theoretical studies, has strong ties to first-order logic and an elegant denotational semantics, and it can be used to model real-world situations. Its most distinctive feature is that of *confluence*, a property we exploit to obtain our polynomial procedures.

Categories and Subject Descriptors

D.3.2 [**Language Classifications**]: Constraint and logic languages. Concurrent, distributed, and parallel languages; D.2.4 [**Software / Program Verification**]: Formal methods; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Logic and constraint programming*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*Program analysis*

General Terms

Algorithms, Theory, Verification

Keywords

Concurrent Constraint Programming, Bisimulation, Partition Refinement, Observational Equivalence

*This work has been partially supported by the project ANR 12IS02001 PACE, ANR-09-BLAN-0169-01 PANDA, and by the French Defence procurement agency (DGA) with a PhD grant.

1. INTRODUCTION

Motivation. *Concurrent constraint programming (ccp)* [26] is a well-established *formalism* from concurrency theory that combines the traditional algebraic and operational view of process calculi with a *declarative* one based upon logic. It was designed to give programmers explicit access to the concept of partial information and, as such, has close ties with *logic and constraint programming*.

The ccp framework models systems whose agents (processes or programs) interact by concurrently *posting* (telling) and querying (*asking*) partial information in a shared medium (the store). Ccp is parametric in a *constraint system* indicating interdependencies (entailment) between partial information and providing for the specification of data types and other rich structures. The above features have attracted a renewed attention as witnessed by the works [23, 11, 7, 6, 17] on calculi exhibiting data-types, logic assertions as well as tell and ask operations. A compelling example of the kind of system ccp can model involves users interacting by posting and querying information in a social network [17].

Nevertheless, despite the extensive research on the foundations and principles of ccp, the development of tools and algorithms for the automatic verification of ccp programs has hitherto been far too little considered. As we shall argue below, the only existing algorithm for deciding the standard notion of process equivalence was given [5] and it has an *exponential time* (and space) complexity.

The main goal of this paper is to produce efficient decision procedures for program equivalence for a meaningful fragment of ccp. Namely, the *summation-free fragment of ccp*, henceforth ccp-{+}. The ccp-{+} formalism is perhaps the most representative sublanguage of ccp. It has been the subject of many theoretical studies because of its computational expressivity, strong ties to first-order logic, and elegant denotational semantics based on closure operators [26]. Its most distinctive property is that of *confluence* in the sense the final resulting store is the same regardless of the execution order of the parallel processes. We shall use this property extensively in proving the correctness of our decision procedures.

Approach. To explain our approach we shall briefly recall some ccp equivalences. The standard notion of *observational (program) equivalence* [26], roughly speaking, decrees that two ccp programs are observationally equivalent if each one can be replaced with the other in any ccp context and produce the same final stores. Other alternative notions of program equivalences for ccp such as saturated barbed bisimilarity ($\dot{\sim}_{sb}$) and its weak variant ($\dot{\approx}_{sb}$) were introduced in [3], where it is also shown that $\dot{\approx}_{sb}$ *coincides* with the standard ccp observational equivalence for ccp-{+} programs.

The above-mentioned alternative notions of ccp equivalences are defined in terms of a *labeled transitions system* (LTS) describing the interactive behavior of ccp programs. (Intuitively, a labeled transition $\gamma \xrightarrow{\alpha} \gamma'$ represents the evolution into the program configuration γ' if the information α is added to store of the program configuration γ.) The advantage of using these alternative notions of equivalence instead of using directly the standard notion of observational equivalence for ccp is that there is a substantial amount of work supporting the automatic verification of bisimilarity-based equivalence. In this paper we shall mainly deal with the verification of $\dot{\approx}_{sb}$ for arbitrary ccp-{+} programs since, as mentioned above, $\dot{\approx}_{sb}$ coincides with observational program equivalence [3].

Unfortunately, the standard algorithms for checking bisimilarity (such as [16, 14, 10, 13]) cannot be reused for $\dot{\sim}_{sb}$ and $\dot{\approx}_{sb}$, since in this particular case of the bisimulation game, when the attacker proposes a transition, the defender not necessarily has to answer with a transition with the same label. (This is analogous to what happens in the asynchronous π-calculus [2] where an input transition can be matched also by an internal (tau) transition.)

Partition Refinement for ccp. By building upon [2], we introduced in [4] a variation of the partition refinement algorithm that allows us to decide $\dot{\sim}_{sb}$ in ccp. The variation is based on the observation that some of the transitions are *redundant*, in the sense that they are logical consequences of other transitions. Unfortunately, such notion of redundancy is not syntactical, but semantical, more precisely, it is based on $\dot{\sim}_{sb}$ itself. Now, if we consider the transition system having only non-redundant transitions, the ordinary notion of bisimilarity coincides with $\dot{\sim}_{sb}$. Thus, in principle, we could remove all the redundant transitions and then check bisimilarity with a standard algorithm. But how can we decide which transitions are redundant, if redundancy itself depends on $\dot{\sim}_{sb}$?

The solution in [4] consists in computing $\dot{\sim}_{sb}$ and redundancy *at the same time*. In the first step, the algorithm considers all the states as equivalent and all the transitions (potentially redundant) as redundant. At any iteration, states are discerned according to (the current estimation of) non-redundant transitions and then non-redundant transitions are updated according to the new computed partition.

One peculiarity of the algorithm in [4] is that in the initial partition, we insert not only the reachable states, but also extra ones which are needed to check for redundancy. Unfortunately, the number of these states might be exponentially bigger then the size of the original LTS and therefore worst-case complexity is *exponential*, even as we shall show this paper, for the restricted case of ccp-{+}.

This becomes even more problematic when considering the weak semantics $\dot{\approx}_{sb}$. Usually weak bisimilarity is computed by first closing the transition relation with respect to internal transitions and then by checking strong bisimilarity on the obtained LTS. This approach (which is referred in [1] as saturation) is unsound for ccp. In [5], it is shown that in order to obtain a sound algorithm, one has to close the transition relation, not only w.r.t. the internal transitions, but w.r.t. *all* the transitions. This induces an explosion of the number of transitions which makes the computation of $\dot{\approx}_{sb}$ even more inefficient.

Confluent ccp. In this paper, we shall consider the "summation free" fragment of ccp (ccp-{+}), i.e., the fragment of ccp without non-deterministic choice. Differently from similar fragments of other process calculi (such as the π-calculus or the mobile ambient), ccp-{+} is *confluent* because concurrent constraints programs interact only via reading and telling permanent pieces of information (roughly speaking, resources are not consumed). When considering the weak equivalence $\dot{\approx}_{sb}$, confluency makes possible to characterize redundant transitions syntactically, i.e., without any information about $\dot{\approx}_{sb}$. Therefore for checking $\dot{\approx}_{sb}$ in ccp-{+}, we can first prune redundant transitions and then check the standard bisimilarity with one of the usual algorithms [16, 14, 10, 13]. Since redundancy can be determined statically, the additional states needed by the algorithm in [4] are not necessary anymore: in this way, the worst case complexity from exponential becomes *polynomial*.

Unfortunately, this approach still suffers of the explosion of transitions caused by the "closure" of the transition relation. In order to avoid this problem, we exploit a completely different approach (based on the semantical notion of *compact input-output sets*) that works directly on the original LTS. We shall conclude our paper by also showing how the results obtained for ccp-{+}, can be exploited to optimize the partition refinement for the full language of ccp.

Contributions. The main contribution of this paper is the introduction of two novel decision procedures that can be used to decide the standard notion of program equivalence for ccp-{+} in polynomial time. This represents a significant improvement over the previous algorithm for program equivalence, which, as we show in this paper, has an exponential time complexity even in the restricted case of ccp-{+} programs. Each of these two new procedures has an advantage over the other. One has a better time complexity. The other can be easily adapted for the full language of ccp to produce significant state space reductions.

We wish to conclude this introduction with a quote from [15] that captures the goal of the present paper:

"*The times have gone, where formal methods were primarily a pen-and-pencil activity for mathematicians. Today, only languages properly equipped with software tools will have a chance to be adopted by industry. It is therefore essential for the next generation of languages based on process calculi to be supported by compilers, simulators, verification tools, etc. The research agenda for theoretical concurrency should therefore address the design of efficient algorithms for translating and verifying formal specifications of concurrent systems*" [15].

Structure of the paper. The paper is organized as follows: In Section 2 we recall the basic knowledge concerning the standard partition refinement and the ccp formalism. In Section 3 we present the partition refinement for ccp from [4] and how it can be used to decide observational equivalence following [5]. Our contributions begin in Section 4 where we prove that the partition refinement for ccp from Section 3 is inefficient even for ccp-{+}, then we intro-

duce some particular features of ccp-{+} which are then used to develop a polynomial procedure for checking observational equivalence in ccp-{+}. In Section 5 we introduce our second, more efficient, method for deciding observational equivalence by using the compact input-output sets. In Section 6 we show how the procedure from Section 4 can be adapted to the full ccp language. Finally, in Section 7 we present our conclusions and future work.

2. BACKGROUND

We start this section by recalling the notion of labeled transition system (LTS), partition and the graph induced by an LTS. Then we present the standard partition refinement algorithm, the concurrent constraint programming (ccp) and we show that partition refinement cannot be used for checking equivalence of concurrent constraint processes.

Labeled Transition System. A labeled transition system (LTS) is a triple $(S, L, \rightsquigarrow)$ where S is a set of states, L a set of labels and $\rightsquigarrow \subseteq S \times L \times S$ a transition relation. We shall use $s \overset{a}{\rightsquigarrow} r$ to denote the transition $(s, a, r) \in \rightsquigarrow$. Given a transition $t = (s, a, r)$ we define the source, the target and the label as follows $src(t) = s$, $tar(t) = r$ and $lab(t) = a$. We assume the reader to be familiar with the standard notion of bisimilarity [19].

Partition. Given a set S, a *partition* $\mathcal{P}$ of S is a set of non-empty *blocks*, i.e., subsets of S, that are all disjoint and whose union is S. We write $\{B_1\} \ldots \{B_n\}$ to denote a partition consisting of (non-empty) blocks $B_1, \ldots, B_n$. A partition represents an equivalence relation where equivalent elements belong to the same block. We write $s\mathcal{P}r$ to mean that s and r are equivalent in the partition $\mathcal{P}$.

LTSs and Graphs. Given a LTS $(S, L, \rightsquigarrow)$, we write $LTS_{\rightsquigarrow}$ for the directed graph whose vertices are the states in S and edges are the transitions in $\rightsquigarrow$. Given a set of initial states $IS \subseteq S$, we write $LTS_{\rightsquigarrow}(IS)$ for the subgraph of $LTS_{\rightsquigarrow}$ rechable from IS. Given a graph G we write $\mathtt{V}(G)$ and $\mathtt{E}(G)$ for the set of vertices and edges of G, respectively.

2.1 Partition Refinement

We report the partition refinement algorithm [16] for checking bisimilarity over the states of an LTS $(S, L, \rightsquigarrow)$.

Given a set of initial states $IS \subseteq S$, the partition refinement algorithm (see Algorithm 1) checks bisimilarity on IS as follows. First, it computes $IS^{\star}_{\rightsquigarrow}$, that is the set of all states that are reachable from IS using $\rightsquigarrow$. Then it creates the partition $\mathcal{P}^0$ where all the elements of $IS^{\star}_{\rightsquigarrow}$ belong to the same block (i.e., they are all equivalent). After the initialization, it iteratively refines the partitions by employing the function on partitions $\mathbf{F}_{\rightsquigarrow}(-)$, defined as follows: for a partition $\mathcal{P}$, $s\mathbf{F}_{\rightsquigarrow}(\mathcal{P})r$ iff

$$\text{if } s \overset{a}{\rightsquigarrow} s' \text{ then exists } r' \text{ s.t. } r \overset{a}{\rightsquigarrow} r' \text{ and } s'\mathcal{P}r'. \qquad (1)$$

See Figure 1 for an example.

The algorithm terminates whenever two consecutive partitions are equivalent. In such a partition two states (reachable from IS) belong to the same block iff they are bisimilar (using the standard notion of bisimilarity [19]).

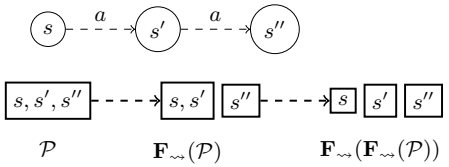


Figure 1: An example of the use of $\mathbf{F}_{\rightsquigarrow}(\mathcal{P})$ from Equation 1

Algorithm 1 $\mathtt{pr}(IS, \rightsquigarrow)$

Initialization

1. $IS^{\star}_{\rightsquigarrow}$ is the set of all states reachable from IS using $\rightsquigarrow$,
2. $\mathcal{P}^0 := IS^{\star}_{\rightsquigarrow}$,

Iteration $\mathcal{P}^{n+1} := \mathbf{F}_{\rightsquigarrow}(\mathcal{P}^n)$ as in Equation 1
Termination If $\mathcal{P}^n = \mathcal{P}^{n+1}$ then return $\mathcal{P}^n$.

2.2 Constraint Systems

The ccp model is parametric in a *constraint system (cs)* specifying the structure and interdependencies of the information that processes can ask or add to a *central shared store*. This information is represented as assertions traditionally referred to as *constraints*. Following [12, 18] we regard a cs as a complete algebraic lattice in which the ordering $\sqsubseteq$ is the reverse of an entailment relation: $c \sqsubseteq d$ means d *entails* c, i.e., d contains "more information" than c. The top element *false* represents inconsistency, the bottom element $true$ is the empty constraint, and the *least upper bound* (lub) $\sqcup$ is the join of information.

Definition 1. (Constraint System) A *constraint system (cs)* is a complete algebraic lattice $\mathbf{C} = (Con, Con_0, \sqsubseteq, \sqcup, true, false)$ where Con, the set of constraints, is a partially ordered set w.r.t. $\sqsubseteq$, Con_0 is the subset of *compact* elements of Con, $\sqcup$ is the lub operation defined on all subsets, and $true$, $false$ are the least and greatest elements of Con, respectively.

Remark 1. We assume that the constraint system is well-founded and that its ordering $\sqsubseteq$ is decidable.

We now define the constraint system we use in our examples.

Example 1. Let Var be a set of variables and ω be the set of natural numbers. A variable assignment is a function $\mu : Var \longrightarrow \omega$. We use $\mathcal{A}$ to denote the set of all assignments, $\mathcal{P}(\mathcal{A})$ to denote the powerset of $\mathcal{A}$, $\emptyset$ the empty set and $\cap$ the intersection of sets. Let us define the following constraint system: The set of constraints is $\mathcal{P}(\mathcal{A})$. We define $c \sqsubseteq d$ iff $c \supseteq d$. The constraint *false* is $\emptyset$, while $true$ is $\mathcal{A}$. Given two constraints c and d, $c \sqcup d$ is the intersection $c \cap d$. We will often use a formula like $x < n$ to denote the corresponding constraint, i.e., the set of all assignments that map x to a number smaller than n.

2.3 Syntax

We now recall the basic ccp process constructions. We are concerned with the verification of finite-state systems, thus we we shall dispense with the recursion operator which is meant for describing

infinite behavior. We shall also omit the local/hiding operator for the simplicity of the presentation (see [3] for further details).

Let $\mathbf{C} = (Con, Con_0, \sqsubseteq, \sqcup, true, false)$ a constraint system. The ccp processes are given by the following syntax:

$$P, Q ::= \mathbf{stop} \mid \mathbf{tell}(c) \mid \mathbf{ask}(c) \rightarrow P \mid P \parallel Q \mid P + Q$$

where $c \in Con_0$. Intuitively, $\mathbf{stop}$ represents termination, $\mathbf{tell}(c)$ adds the constraint (or partial information) c to the store. The addition is performed regardless the generation of inconsistent information. The process $\mathbf{ask}(c) \rightarrow P$ may execute P if c is entailed from the information in the store. The processes $P \parallel Q$ and $P + Q$ stand, respectively, for the *parallel execution* and *non-deterministic choice* of P and Q.

Remark 2. (ccp-{+}). Henceforth, we use ccp-{+} to refer to the fragment of ccp without nondeterministic choice.

2.4 Reduction Semantics

A configuration is a pair $\langle P, d\rangle$ representing a *state* of a system; d is a constraint representing the global store, and P is a process, i.e., a term of the syntax. We use $Conf$ with typical elements $\gamma, \gamma', \ldots$ to denote the set of all configurations. We will use $Conf_{\texttt{ccp}-\{+\}}$ for the ccp-{+} configurations.

The operational semantics of ccp is given by an *unlabeled* transition relation between configurations: a transition $\gamma \longrightarrow \gamma'$ intuitively means that the configuration γ can reduce to γ'. We call these kind of unlabeled transitions *reductions* and we use $\longrightarrow^*$ to denote the reflexive and transitive closure of $\longrightarrow$.

Formally, the reduction semantics of ccp is given by the relation $\longrightarrow$ defined in Table 1. These rules are easily seen to realize the intuitions described in the syntax (Section 2.3).

In [3], the authors introduced a *barbed semantics* for ccp. Barbed equivalences have been introduced in [20] for CCS, and have become a classical way to define the semantics of formalisms equipped with unlabeled reduction semantics. Intuitively, *barbs* are basic observations (predicates) on the states of a system. In the case of ccp, barbs are taken from the underlying set Con_0 of the constraint system.

Definition 2. (Barbs) A configuration $\gamma = \langle P, d\rangle$ is said to satisfy the *barb* c, written $\gamma \downarrow_c$, iff $c \sqsubseteq d$. Similarly, γ satisfies a *weak barb* c, written $\gamma \Downarrow_c$, iff there exist γ' s.t. $\gamma \longrightarrow^* \gamma' \downarrow_c$.

Example 2. Let $\gamma = \langle \mathbf{ask}\ (x > 10) \rightarrow \mathbf{tell}(y < 42), x > 10\rangle$. We have $\gamma \downarrow_{x>5}$ since $(x > 5) \sqsubseteq (x > 10)$ and $\gamma \Downarrow_{y<42}$ since $\gamma \longrightarrow \langle \mathbf{tell}(y < 42), x > 10\rangle \longrightarrow \langle \mathbf{stop}, (x > 10) \sqcup (y < 42)\rangle \downarrow_{y<42}$.

In this context, the equivalence proposed is the *saturated bisimilarity* [9, 8]. Intuitively, in order for two states to be saturated bisimilar, then (i) they should expose the same barbs, (ii) whenever one of them moves then the other should reply and arrive at an equivalent state (i.e. follow the bisimulation game), (iii) they should be equivalent under all the possible contexts of the language. In the case of ccp, it is enough to require that bisimulations are *upward closed* as in condition (iii) below.

Definition 3. (Saturated Barbed Bisimilarity) A *saturated barbed bisimulation* is a symmetric relation $\mathcal{R}$ on configurations s.t. whenever $(\gamma_1, \gamma_2) \in \mathcal{R}$ with $\gamma_1 = \langle P, c\rangle$ and $\gamma_2 = \langle Q, d\rangle$ implies that:

(i) if $\gamma_1 \downarrow_e$ then $\gamma_2 \downarrow_e$,

(ii) if $\gamma_1 \longrightarrow \gamma_1'$ then there exists γ_2' s.t. $\gamma_2 \longrightarrow \gamma_2'$ and $(\gamma_1', \gamma_2') \in \mathcal{R}$,

(iii) for every $a \in Con_0$, $(\langle P, c \sqcup a\rangle, \langle Q, d \sqcup a\rangle) \in \mathcal{R}$.

We say that γ_1 and γ_2 are *saturated barbed bisimilar* ($\gamma_1 \dot{\sim}_{sb} \gamma_2$) if there is a saturated barbed bisimulation $\mathcal{R}$ s.t. $(\gamma_1, \gamma_2) \in \mathcal{R}$.

Weak saturated barbed bisimulations are defined as above by replacing $\downarrow$ by $\Downarrow$ and $\longrightarrow$ by $\longrightarrow^*$. We say that γ_1 and γ_2 are *weak saturated barbed bisimilar* ($\gamma_1 \dot{\approx}_{sb} \gamma_2$) if there exists a weak saturated barbed bisimulation $\mathcal{R}$ s.t. $(\gamma_1, \gamma_2) \in \mathcal{R}$.

Remark 3. It should be noticed that standard notion of observational ccp program equivalence was shown to coincide with $\dot{\approx}_{sb}$ in the case of ccp-{+}[3]. For the sake of space we shall not introduce the standard notion– see [3] for further details.

We now illustrate $\dot{\sim}_{sb}$ and $\dot{\approx}_{sb}$ with the following two examples.

Example 3. Take $T = \mathbf{tell}(true)$, $P = \mathbf{ask}\ (x < 7) \rightarrow T$ and $Q = \mathbf{ask}\ (x < 5) \rightarrow T$. Now, $\langle P, true\rangle \not\dot{\sim}_{sb} \langle Q, true\rangle$, since $\langle P, x < 7\rangle \longrightarrow$, while $\langle Q, x < 7\rangle \not\longrightarrow$. Then consider $\langle P + Q, true\rangle$ and observe that $\langle P+Q, true\rangle \dot{\sim}_{sb} \langle P, true\rangle$. Indeed, for all constraints e, s.t. $x < 7 \sqsubseteq e$, both the configurations evolve into $\langle T, e\rangle$, while for all e s.t. $x < 7 \not\sqsubseteq e$, both configurations cannot proceed. Since $x < 7 \sqsubseteq x < 5$, the behavior of Q is somehow absorbed by the behavior of P.

Example 4. Let $\gamma_1 = \langle \mathbf{tell}(true), true\rangle$ and $\gamma_2 = \langle \mathbf{ask}\ (c) \rightarrow \mathbf{tell}(d), true\rangle$. We can show that $\gamma_1 \dot{\approx}_{sb} \gamma_2$ when $d \sqsubseteq c$. Intuitively, this corresponds to the fact that the implication $c \Rightarrow d$ is equivalent to $true$ when c entails d. The LTSs of γ_1 and γ_2 are the following: $\gamma_1 \longrightarrow \langle \mathbf{stop}, true\rangle$ and $\gamma_2 \xrightarrow{c} \langle \mathbf{tell}(d), c\rangle \longrightarrow \langle \mathbf{stop}, c\rangle$. It is now easy to see that the symmetric closure of the relation $\mathcal{R} = \{(\gamma_2, \gamma_1), (\gamma_2, \langle \mathbf{stop}, true\rangle), (\langle \mathbf{tell}(d), c\rangle, \langle \mathbf{stop}, c\rangle), (\langle \mathbf{stop}, c\rangle, \langle \mathbf{stop}, c\rangle)\}$ is a weak saturated barbed bisimulation as in Definition 3.

2.5 Labeled Semantics

In [3] we have shown that $\dot{\approx}_{sb}$ is *fully abstract* with respect to the standard observational equivalence from [26]. Unfortunately, the quantification over all constraints in condition (iii) of Definition 3 makes hard checking $\dot{\sim}_{sb}$ and $\dot{\approx}_{sb}$, since one should check infinitely many constraints. In order to avoid this problem we have introduced in [3] a labeled transition semantics where labels are constraints.

In a transition of the form $\langle P, d\rangle \xrightarrow{\alpha} \langle P', d'\rangle$ the label $\alpha \in Con_0$ represents a *minimal* information (from the environment) that needs to be added to the store d to reduce from $\langle P, d\rangle$ into $\langle P', d'\rangle$, i.e., $\langle P, d \sqcup \alpha\rangle \longrightarrow \langle P', d'\rangle$. As a consequence, the transitions labeled with the constraint $true$ are in one to one correspondence with the

$$\text{R1}\ \langle \mathbf{tell}(c), d\rangle \longrightarrow \langle \mathbf{stop}, d \sqcup c\rangle \quad \text{R2}\ \frac{c \sqsubseteq d}{\langle \mathbf{ask}\ (c)\ \rightarrow\ P, d\rangle \longrightarrow \langle P, d\rangle} \quad \text{R3}\ \frac{\langle P, d\rangle \longrightarrow \langle P', d'\rangle}{\langle P \parallel Q, d\rangle \longrightarrow \langle P' \parallel Q, d'\rangle} \quad \text{R4}\ \frac{\langle P, d\rangle \longrightarrow \langle P', d'\rangle}{\langle P\ +\ Q, d\rangle \longrightarrow \langle P', d'\rangle}$$

Table 1: Reduction semantics for ccp (the symmetric rules for R3 and R4 are omitted).

reductions defined in the previous section. For this reason, hereafter we will sometimes write $\longrightarrow$ to mean $\xrightarrow{true}$. Before formally introducing the labeled semantics, we fix some notation.

Notation 1. We will use $\rightsquigarrow$ to denote a generic transition relation on the state space $Conf$ and labels Con_0. Also in this case $\rightsquigarrow$ mean $\overset{true}{\rightsquigarrow}$. Given a set of initial configuration IS, $\texttt{Config}_{\rightsquigarrow}(IS)$ denote the sets $\{\gamma' \mid \exists \gamma \in IS \text{ s.t. } \gamma \overset{\alpha_1}{\rightsquigarrow} \ldots \overset{\alpha_n}{\rightsquigarrow} \gamma' \text{ for some } n \geq 0\}$.

The LTS $(Conf, Con_0, \longrightarrow)$ is defined by the rules in Table 2. The rule LR2 , for example, says that $\langle \mathbf{ask}\ (c)\ \rightarrow\ P, d\rangle$ can evolve to $\langle P, d \sqcup \alpha\rangle$ if the environment provides a minimal constraint α that added to the store d entails c, i.e., $\alpha \in \min\{a \in Con_0 \mid c \sqsubseteq d \sqcup a\}$. The other rules are easily seen to realize the intuition given in Section 2.3. Figure 2 illustrates the LTS of our running example.

Given the LTS $(Conf, Con_0, \longrightarrow)$, one would like to exploit it for "efficiently characterizing" $\dot{\sim}_{sb}$ and $\dot{\approx}_{sb}$. One first naive attempt would be to consider the standard notion of (weak) bisimilarity over $\longrightarrow$, but this would distinguish configurations which are in $\dot{\sim}_{sb}$ (and $\dot{\approx}_{sb}$), as illustrated by the following two examples.

Example 5. In Example 3 we saw that $\langle P+Q, true\rangle \dot{\sim}_{sb} \langle P, true\rangle$. However, $\langle P + Q, true\rangle \xrightarrow{x<5} \langle T, x < 5\rangle$, while $\langle P, true\rangle \overset{x<5}{\not\longrightarrow}$.

Example 6. In Example 4, we showed that $\gamma_1 \dot{\approx}_{sb} \gamma_2$. However, $\gamma_2 \xrightarrow{c}$, while $\gamma_1 \overset{c}{\not\longrightarrow}$

The examples above show that the ordinary notion of bisimilarity do not coincide with the intended semantics ($\dot{\sim}_{sb}$ and $\dot{\approx}_{sb}$). As a consequence, the standard partition refinement algorithm (Section 2.1) cannot be used for checking $\dot{\sim}_{sb}$ and $\dot{\approx}_{sb}$. However, one can consider a variation of the bisimulation game, namely *irredundant bisimilarity* [4], which coincide with $\dot{\sim}_{sb}$ and, in the weak case [5], with $\dot{\approx}_{sb}$. This fact allowed us in [4] to define a variation of the partition refinement algorithm which we show in the next section.

First, we recall some results from [4] and [5], which are fundamental for the development of the paper.

Lemma 1. ([3], [5]) (Soundness) If $\langle P, c\rangle \xrightarrow{\alpha} \langle P', c'\rangle$ then $\langle P, c \sqcup \alpha\rangle \longrightarrow \langle P', c'\rangle$. (Completeness) If $\langle P, c \sqcup a\rangle \longrightarrow \langle P', c'\rangle$ then there exists α and b s.t. $\langle P, c\rangle \xrightarrow{\alpha} \langle P', c''\rangle$ where $\alpha \sqcup b = a$ and $c'' \sqcup b = c'$.

The weak labeled transition system $(Conf, Con_0, \Longrightarrow)$ is defined by the rules in Table 3. This LTS is sound and complete, as $\longrightarrow$, and it can be used to decide $\dot{\approx}_{sb}$ as shown in [5].

$$\text{R-Tau}\ \frac{}{\gamma \Longrightarrow \gamma} \quad \text{R-Label}\ \frac{\gamma \xrightarrow{\alpha} \gamma'}{\gamma \overset{\alpha}{\Longrightarrow} \gamma'} \quad \text{R-Add}\ \frac{\gamma \overset{\alpha}{\Longrightarrow} \gamma' \overset{\beta}{\Longrightarrow} \gamma''}{\gamma \overset{\alpha\sqcup\beta}{\Longrightarrow} \gamma''}$$

Table 3: Weak semantics for ccp

Lemma 2. ([5]) (Soundness) If $\langle P, c\rangle \overset{\alpha}{\Longrightarrow} \langle P', c'\rangle$ then $\langle P, c \sqcup \alpha\rangle \Longrightarrow \langle P', c'\rangle$. (Completeness) If $\langle P, c \sqcup a\rangle \Longrightarrow \langle P', c'\rangle$ then there exists α and b s.t. $\langle P, c\rangle \overset{\alpha}{\Longrightarrow} \langle P', c''\rangle$ where $\alpha \sqcup b = a$ and $c'' \sqcup b = c'$.

Note that we close $\longrightarrow$, not only w.r.t $true$ transitions (as similarly done in CCS, where τ intutively corresponds to $true$), but w.r.t. *all* the transitions. This is needed to check $\dot{\approx}_{sb}$, because otherwise the above lemma would not hold.

The following lemma relates the labeled and weak semantics, i.e. $\longrightarrow$ and $\Longrightarrow$. It states that a single transition in $\Longrightarrow$ corresponds to a sequence of reductions ($\longrightarrow^*$).

Lemma 3. ([5]) $\gamma \longrightarrow^* \gamma'$ iff $\gamma \Longrightarrow \gamma'$.

Finally, we introduce some useful notation regarding the transitions whose label is $true$.

Notation 2. When the label of a transition is $true$ we will omit it. Namely, we will use $\gamma \longrightarrow \gamma'$ and $\gamma \Longrightarrow \gamma'$ to denote $\gamma \xrightarrow{true} \gamma'$ and $\gamma \overset{true}{\Longrightarrow} \gamma'$ since they are equivalent by Lemma 1 and 2.

3. PARTITION REFINEMENT FOR ccp

In this section we recall the partition refinement algorithm for ccp and how it can be used to decide observational equivalence.

3.1 Strong equivalence

In [4] we adapted the standard partition refinement procedure to decide strong bisimilarity for ccp ($\dot{\sim}_{sb}$). As we did for the standard partition refinement, we also start with $\texttt{Config}_{\longrightarrow}(IS)$, that is the set of all states that are reachable from the set of initial state IS using $\longrightarrow$. However, in the case of ccp some other states must be added to $IS^{\star}_{\rightsquigarrow}$ in order to verify $\dot{\sim}_{sb}$ as we will explain later on.

Now, since configurations satisfying different barbs are surely different, it can be safely started with a partition that equates all and only those states satisfying the same barbs. Hence, as initial partition of $IS^{\star}_{\rightsquigarrow}$, we take $\mathcal{P}^0 = \{B_1\} \ldots \{B_m\}$, where γ and γ' are in B_i iff they satisfy the same barbs.

When splitting the above-mentioned partitions, unlike for the standard partition refinement, we need to consider a particular kind of

$$\mathsf{LR1}\ \langle \mathbf{tell}(c), d\rangle \xrightarrow{true} \langle \mathbf{stop}, d \sqcup c\rangle \qquad \mathsf{LR2}\ \frac{\alpha \in \min\{a \in Con_0 \mid c \sqsubseteq d \sqcup a\}}{\langle \mathbf{ask}\ (c)\ \rightarrow\ P, d\rangle \xrightarrow{\alpha} \langle P, d \sqcup \alpha\rangle}$$

$$\mathsf{LR3}\ \frac{\langle P, d\rangle \xrightarrow{\alpha} \langle P', d'\rangle}{\langle P \parallel Q, d\rangle \xrightarrow{\alpha} \langle P' \parallel Q, d'\rangle} \qquad \mathsf{LR4}\ \frac{\langle P, d\rangle \xrightarrow{\alpha} \langle P', d'\rangle}{\langle P + Q, d\rangle \xrightarrow{\alpha} \langle P', d'\rangle}$$

Table 2: Labeled semantics for ccp (the symmetric rules for LR3 and LR4 are omitted).

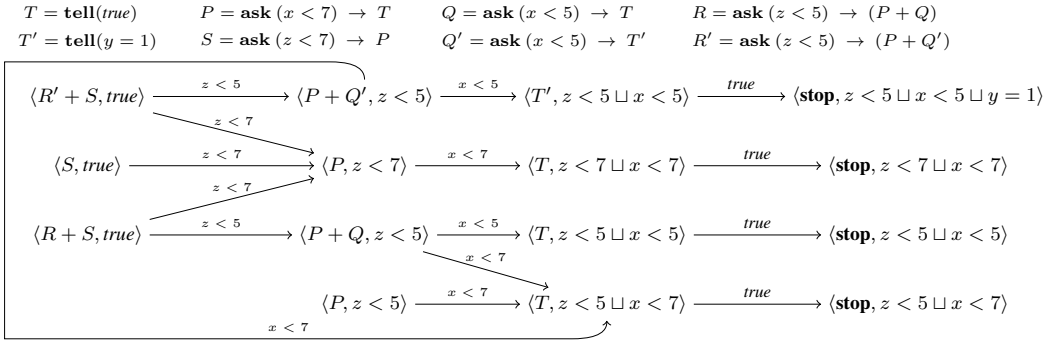


Figure 2: $LTS_{\longrightarrow}(IS)$ **where (**$IS = \{\langle R' + S, true\rangle, \langle S, true\rangle, \langle R + S, true\rangle\}$**).**

transitions, so-called *irredundant transitions*. These are those transitions that are not dominated by others, in a given partition, in the sense defined below.

Definition 4. (Transition Domination) Let t and t' be two transitions of the form $t = (\gamma, \alpha, \langle P', c'\rangle)$ and $t' = (\gamma, \beta, \langle P', c''\rangle)$. We say that t dominates t', written $t \succ_D t'$, iff $\alpha \sqsubset \beta$ and $c'' = c' \sqcup \beta$.

The intuition is that the transition t dominates t' iff t requires less information from the environment than t' does (hence $\alpha \sqsubset \beta$), and they end up in configurations which differ only by the additional information in β not present in α (hence $c'' = c' \sqcup \beta$). To better explain this notion let us give an example.

Example 7. Let $P = (\mathbf{ask}\ (x < 15)\ \rightarrow\ \mathbf{tell}(y > 42)) + (\mathbf{ask}\ (x < 10)\ \rightarrow\ \mathbf{tell}(y > 42))$ and let $\gamma = \langle P, true\rangle$. Consider $t_1 = \gamma \xrightarrow{x<15} \langle \mathbf{tell}(y > 42), x < 15\rangle$ and $t_2 = \gamma \xrightarrow{x<10} \langle \mathbf{tell}(y > 42), x < 10\rangle$, then one can check that $t_1 \succ_D t_2$ since $(x < 15) \sqsubset (x < 10)$ and $(x < 10) = ((x < 15) \sqcup (x < 10))$.

Notice that in the definition above t and t' end up in configurations whose processes are *syntactically identical* (i.e., P'). The following notion parameterizes the notion of dominance w.r.t. a relation on configurations $\mathcal{R}$ (rather than fixing it to the identity on configurations).

Definition 5. (Transition Domination w.r.t. $\mathcal{R}$ and Irredundant Transition w.r.t. $\mathcal{R}$) We say that the transition t dominates a transition t' w.r.t a relation on configurations $\mathcal{R}$, written $t \succ_{\mathcal{R}} t'$, iff there exists t'' such that $t \succ_D t''$, $lab(t'') = lab(t')$ and $tar(t'')\mathcal{R}\ tar(t')$. A transition is said to be *redundant* w.r.t. to $\mathcal{R}$ when it is dominated by another w.r.t. $\mathcal{R}$, otherwise it is said to be *irredundant* w.r.t. to $\mathcal{R}$.

To understand this definition better consider the following example.

Example 8. Let $Q_1 = (\mathbf{ask}\ (b)\ \rightarrow\ (\mathbf{ask}\ (c)\ \rightarrow\ \mathbf{tell}(d)))$, $Q_2 = (\mathbf{ask}\ (a)\ \rightarrow\ \mathbf{stop})$ and $P = Q_1 + Q_2$, where $d \sqsubseteq c$ and $a \sqsubset b$. Now let $\gamma = \langle P, true\rangle$, then consider $t = \gamma \xrightarrow{a} \langle \mathbf{stop}, a\rangle$ and $t' = \gamma \xrightarrow{b} \langle \mathbf{ask}\ (c)\ \rightarrow\ \mathbf{tell}(d), b\rangle$. Let $\mathcal{R} = \dot{\approx}_{sb}$ and take $t'' = (\gamma, b, \langle \mathbf{stop}, b\rangle)$, one can check that $t \succ_{\mathcal{R}} t'$ as in Definition 5. We have that $t \succ_D t''$ follows from $a \sqsubset b$. And we know $tar(t'')\mathcal{R}\ tar(t')$ from Example 4, i.e. $\langle \mathbf{stop}, b\rangle \dot{\approx}_{sb} \langle \mathbf{ask}\ (c)\ \rightarrow\ \mathbf{tell}(d), b\rangle$.

We now explain briefly how to compute $IS^{\star}_{\rightsquigarrow}$ using the Rules in Table 4. Rules $(\mathsf{IS}^{IS}_{\rightsquigarrow})$ and $(\mathsf{RS}^{IS}_{\rightsquigarrow})$ say that all the states generated from the labeled semantics (Table 2) from the set of initial states should be included, i.e., $\mathtt{Config}_{\rightsquigarrow}(IS) \subseteq IS^{\star}_{\rightsquigarrow}$.

The rule $(\mathsf{RD}^{IS}_{\rightsquigarrow})$ adds the additional states needed to check redundancy. Consider the transitions $t_1 = \gamma \overset{\alpha}{\rightsquigarrow} \langle P_1, c_1\rangle$ and $t_2 = \gamma \overset{\beta}{\rightsquigarrow} \langle P_2, c_2\rangle$ with $\alpha \sqsubset \beta$ and $c_2 = c_1 \sqcup \beta$ in Rule $(\mathsf{RD}^{IS}_{\rightsquigarrow})$. Suppose that at some iteration of the partition refinement algorithm the current partition is $\mathcal{P}$ and that $\langle P_2, c_2\rangle \mathcal{P} \langle P_1, c_2\rangle$. Then, according to Definition 5 the transitions t_1 would dominate t_2 w.r.t $\mathcal{P}$. This makes t_2 redundant w.r.t $\mathcal{P}$. Since $\langle P_1, c_2\rangle$ may allow us to witness a potential redundancy of t_2, we include it in $IS^{\star}_{\rightsquigarrow}$ (and thus, from the definition of the initial partition $\mathcal{P}^0$, also in the block of $\mathcal{P}^0$ where $\langle P_2, c_2\rangle$ is). See [4] for further details about the computation of $IS^{\star}_{\rightsquigarrow}$.

Finally, we shall describe how the refinement is done in the case

$$(\mathsf{IS}^{IS}_{\rightsquigarrow})\ \frac{\gamma \in IS}{\gamma \in IS^{\star}_{\rightsquigarrow}} \qquad (\mathsf{RS}^{IS}_{\rightsquigarrow})\ \frac{\gamma \in IS^{\star}_{\rightsquigarrow} \quad \gamma \overset{\alpha}{\rightsquigarrow} \gamma'}{\gamma' \in IS^{\star}_{\rightsquigarrow}} \qquad (\mathsf{RD}^{IS}_{\rightsquigarrow})\ \frac{\gamma \in IS^{\star}_{\rightsquigarrow} \quad t_1 = \gamma \overset{\alpha}{\rightsquigarrow} \langle P_1, c_1\rangle \quad t_2 = \gamma \overset{\beta}{\rightsquigarrow} \langle P_2, c_2\rangle \quad \alpha \sqsubset \beta \quad c_2 = c_1 \sqcup \beta}{\langle P_1, c_2\rangle \in IS^{\star}_{\rightsquigarrow}}$$

Table 4: Rules for generating the states used in the partition refinement for ccp

ccp. Instead of using the function $\mathbf{F}_{\rightsquigarrow}(\mathcal{P})$ of Algorithm 1, the partitions are refined by employing the function $\mathbf{IR}_{\rightsquigarrow}(\mathcal{P})$ defined as:

Definition 6. (Refinement function for ccp) Given a partition $\mathcal{P}$ we define $\mathbf{IR}_{\rightsquigarrow}(\mathcal{P})$ as follows: $\gamma_1\ \mathbf{IR}_{\rightsquigarrow}(\mathcal{P})\ \gamma_2$ iff

if $\gamma_1 \overset{\alpha}{\rightsquigarrow} \gamma_1'$ is irredundant w.r.t. $\mathcal{P}$

then there exists γ_2' s.t. $\gamma_2 \overset{\alpha}{\rightsquigarrow} \gamma_2'$ and $\gamma_1'\ \mathcal{P}\ \gamma_2'$

See Figure 3 for an example of the use of $\mathbf{IR}_{\rightsquigarrow}(-)$.

Algorithm 2 $\texttt{pr-ccp}(IS, \rightsquigarrow)$

Initialization

1. Compute $IS^{\star}_{\rightsquigarrow}$ with the rules $(\mathsf{IS}^{IS}_{\rightsquigarrow})$, $(\mathsf{RS}^{IS}_{\rightsquigarrow})$, $(\mathsf{RD}^{IS}_{\rightsquigarrow})$ defined in Table 4,
2. $\mathcal{P}^0 = \{B_1\} \ldots \{B_m\}$ is a partition of $IS^{\star}_{\rightsquigarrow}$ where γ and γ' are in B_i iff they satisfy the same barbs $(\downarrow_c)$,

Iteration $\mathcal{P}^{n+1} := \mathbf{IR}_{\rightsquigarrow}(\mathcal{P}^n)$ as in Definition 6
Termination If $\mathcal{P}^n = \mathcal{P}^{n+1}$ then return $\mathcal{P}^n$.

The Algorithm 2 can be used to decide strong saturated bisimilarity $\dot{\sim}_{sb}$ with exponential time. (Recall that $\texttt{Config}_{\longrightarrow}(IS)$ represents the set of states that are reachable from the initial states IS using $\longrightarrow$.) More precisely:

Theorem 1. ([4]) Let γ and γ' be two ccp configurations. Let $IS = \{\gamma, \gamma'\}$ and let $\mathcal{P}$ be the output of $\texttt{pr-ccp}(IS, \longrightarrow)$ in Algorithm 2. Then

- $\gamma\ \mathcal{P}\ \gamma'$ iff $\gamma \dot{\sim}_{sb} \gamma'$.
- $\texttt{pr-ccp}(IS, \longrightarrow)$ may take exponential time in the size of $\texttt{Config}_{\longrightarrow}(IS)$.

The exponential time is due to construction of the set $IS^{\star}_{\longrightarrow}$ (Algorithm 2, step 1) whose size is exponential in $|\texttt{Config}_{\longrightarrow}(IS)|$.

3.2 Weak equivalence

We can also use the above-mentioned algorithm to verify the weak version of saturated bisimilarity ($\dot{\approx}_{sb}$). Recall that in [3] it was shown that in ccp-{+}, $\dot{\approx}_{sb}$ coincides with the standard notion of ccp program (observational) equivalence.

Following [1] the reduction of the problem of deciding $\dot{\approx}_{sb}$ to the problem of deciding $\dot{\sim}_{sb}$ is obtained by adding some additional transitions, so called weak transitions, to the LTS. Given two configurations γ and γ', the first step is to build $G = LTS_{\longrightarrow}(IS)$ where $IS = \{\gamma, \gamma'\}$. Using G we then proceed to compute $G' = LTS_{\Longrightarrow}(IS)$, and finally we run Algorithm 2 adapted to G'. The adaptation consists in using weak barbs $(\Downarrow_c)$ instead of barbs $(\downarrow_c)$ for the initial partition $\mathcal{P}^0$ and using $\Longrightarrow$ as a parameter of Algorithm 2.

Definition 7. (Weak Partition Refinement for ccp) We define the procedure $\texttt{weak-pr-ccp}(IS, \rightsquigarrow)$ by replacing the barbs $(\downarrow_c)$ in step 2 of Algorithm 2 with weak barbs $(\Downarrow_c)$.

Using this algorithm we can decide $\dot{\approx}_{sb}$ also with exponential time. This follows from Theorem 1.

Theorem 2. ([5]) Let γ and γ' be two ccp configurations. Let $IS = \{\gamma, \gamma'\}$ and let $\mathcal{P}$ be the output of $\texttt{weak-pr-ccp}(IS, \Longrightarrow)$ in Definition 7. Then

- $\gamma\ \mathcal{P}\ \gamma'$ iff $\gamma \dot{\approx}_{sb} \gamma'$.
- $\texttt{weak-pr-ccp}(IS, \Longrightarrow)$ may take exponential time in the size of $\texttt{Config}_{\longrightarrow}(IS)$.

As for the strong case, the exponential time is due to construction of the set $IS^{\star}_{\Longrightarrow}$ by $\texttt{weak-pr-ccp}(IS, \Longrightarrow)$, whose size is exponential in $|\texttt{Config}_{\longrightarrow}(IS)|$. In the next section we shall address the issue of avoiding this exponential construction in the context of confluent ccp.

4. USING PARTITION REFINEMENT FOR CHECKING OBSERVATIONAL EQUIVALENCE IN ccp-{+}

In the previous section, we presented a procedure to verify $\dot{\approx}_{sb}$ for ccp and we saw how this method takes exponential time (in the size of the LTS) to check whether two configurations are weakly bisimilar. In this section, we will explore what happens with such procedure when we restrict ourselves to ccp-{+}. We shall see that $\texttt{pr-ccp}(IS, \longrightarrow)$ may also be exponential time for inputs from the ccp-{+} fragment.

Let us consider the following ccp-{+} construction.

Example 9. Let $n > 0$. We define $P^n = P^n_0$ with P^n_i, for $i \in \{0, \ldots, n-1\}$, given by:

$$P^n_i = (\mathbf{ask}\,(a_i) \rightarrow (\mathbf{ask}\,(b_i) \rightarrow P^n_{i+1})) \parallel (\mathbf{ask}\,(b_i) \rightarrow \mathbf{stop})$$

and $P^n_n = \mathbf{tell}(b_n)$. Furthermore, we assume that for all $i \in \{0, \ldots, n-1\}$ we have $a_i \sqsubseteq b_i$ and for all $j \in \{0, \ldots, n-1\}$ if $i \neq j$ then $a_i \not\sqsubseteq a_j$ and $b_i \not\sqsubseteq b_j$. The LTS for $\langle P^n, true\rangle$ is illustrated in Figure 4.

One can verify that by taking $IS = \{\langle P^n, true\rangle\}$ as in the example above, then the size of $IS^{\star}_{\longrightarrow}$ in Algorithm 2 grows exponentially with n, essentially because of the rule $(\mathsf{RD}^{IS}_{\longrightarrow})$.

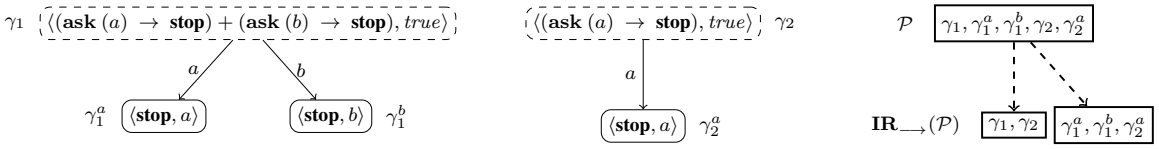


Figure 3: An example of the use of $\mathbf{IR}_{\longrightarrow}(\mathcal{P})$ as in Definition 6. Notice that γ_1 and γ_2 end up in the same block after the refinement since $\gamma_1 \xrightarrow{b} \gamma_1^b$ is a redundant transition w.r.t $\mathcal{P}$ hence it is not required that γ_2 matches it.

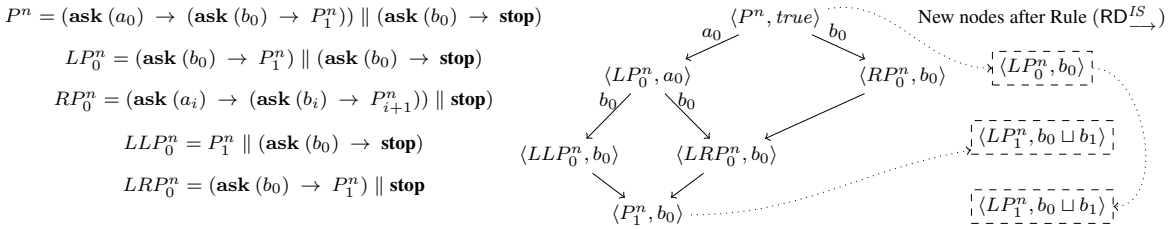


Figure 4: $LTS_{\longrightarrow}(IS)$ where $IS = \{\langle P^n, true\rangle\}$ as in Example 9. The configurations in the right part are generated by $(\mathsf{RD}_{\longrightarrow}^{IS})$ applied to the source nodes of the dotted arrows. Some transitions and stop processes were omitted for clarity.

Proposition 1. Let $\gamma = \langle P^n, true\rangle$ and $IS = \{\gamma\}$, let $\mathcal{P}$ be the output $\texttt{pr-ccp}(IS, \longrightarrow)$ in Algorithm 2, then $\texttt{pr-ccp}(IS, \longrightarrow)$ takes at least exponential time in n.

PROOF. One can check that $|IS^{\star}_{\longrightarrow} IS|$ is given by the following function $f(n) = 2f(n-1)+5$ since $(\mathsf{RD}_{\longrightarrow}^{IS})$creates a new node at each level that contains a new potential redundant transition. Since $f(n) = \Omega(2^n)$ then $\texttt{pr-ccp}(IS, \longrightarrow)$ takes at least exponential time in n. □

The main problem is that the procedure does not distinguish between summation-free processes and the normal ccp processes. Therefore, it is unable to exploit the underlying properties of ccp-{+} and the algorithm will perform (in the worst-case) inherently the same as for the full ccp, as evidenced in the example above.

4.1 Properties of ccp-{+}

In this section we will state some features that (unlike the full ccp) this fragment possess. The first one we want to introduce is confluence. Intuitively, in ccp-{+}, if from a given configuration we have two possible reductions ($\longrightarrow$), then we are guaranteed that they will coincide at some point of the computation. Recall that $Conf_{\texttt{ccp}-\{+\}}$ is the set of all ccp-{+} configurations, i.e. configurations whose process is summation-free.

Proposition 2. Let $\gamma \in Conf_{\texttt{ccp}-\{+\}}$. If $\gamma \longrightarrow^* \gamma_1$ and $\gamma \longrightarrow^* \gamma_2$ then there exists γ' such that $\gamma_1 \longrightarrow^* \gamma'$ and $\gamma_2 \longrightarrow^* \gamma'$.

Before discussing the second property, we need to introduce some notation. We shall call *derivatives* (of γ) the successors reached via (zero or more) reductions ($\longrightarrow^*$) starting from a given configuration γ.

Definition 8. (Derivatives) The derivatives of a configuration γ, written $\mathsf{Deriv}(\gamma)$, are defined as $\mathsf{Deriv}(\gamma) = \{\gamma' \mid \gamma \longrightarrow^* \gamma'\}$.

Using this notation, we can now state another property of ccp-{+}: A configuration is weakly bisimilar to all its derivatives.

Lemma 4. Let $\gamma \in Conf_{\texttt{ccp}-\{+\}}$. For all $\gamma' \in \mathsf{Deriv}(\gamma)$ we have $\gamma \dot{\approx}_{sb} \gamma'$.

PROOF. Let $\mathcal{R} = \{(\gamma_1, \gamma_2) \mid \exists \gamma_3 \text{ s.t. } \gamma_1 \longrightarrow^* \gamma_3 \text{ and } \gamma_2 \longrightarrow^* \gamma_3\}$, we prove that $\mathcal{R}$ is a weak saturated barbed bisimulation. Let (γ_1, γ_2) be any pair of configurations in $\mathcal{R}$.
(i) If $\gamma_1 \Downarrow_e$ then by definition $\gamma_1 \longrightarrow^* \gamma_1' \downarrow_e$. By confluence (Proposition 2) $\gamma_1' \longrightarrow^* \gamma_3$ and thus $\gamma_3 \downarrow_e$ (since constraints can only be added). Since $\gamma_2 \longrightarrow^* \gamma_3 \downarrow_e$ we conclude that $\gamma_2 \Downarrow_e$.
(ii) If $\gamma_1 \longrightarrow^* \gamma_1'$, then by confluence $\gamma_1' \longrightarrow^* \gamma_3$ and therefore $(\gamma_1', \gamma_2) \in \mathcal{R}$.
(iii) Finally, let $\gamma_1 = \langle P_1, c_1\rangle$ and $\gamma_2 = \langle P_2, c_2\rangle$. If $\langle P_1, c_1\rangle \longrightarrow^* \langle P_3, c_3\rangle$ and $\langle P_2, c_2\rangle \longrightarrow^* \langle P_3, c_3\rangle$, then $\langle P_1, c_1 \sqcup e\rangle \longrightarrow^* \langle P_3, c_3 \sqcup e\rangle$ and $\langle P_2, c_2 \sqcup e\rangle \longrightarrow^* \langle P_3, c_3 \sqcup e\rangle$ and thus $(\langle P_1, c_1 \sqcup e\rangle, \langle P_2, c_2 \sqcup e\rangle) \in \mathcal{R}$. □

In the next section we shall take advantage of these properties to check $\dot{\approx}_{sb}$ for ccp-{+} configurations.

4.2 Optimizations to partition refinement for ccp-{+}

We presented how the partition refinement for ccp performs for ccp-{+} as well as some characteristics of the configurations of this fragment. In this section, using such features, we shall show that the complexity of $\texttt{weak-pr-ccp}(IS, \Longrightarrow)$ can be improved, thus we can check $\dot{\approx}_{sb}$ in a more efficient manner.

Due to the nature of ccp-{+}, determining which are the redundant transitions w.r.t. $\dot{\approx}_{sb}$ (Definition 5) becomes an easier task. As we explained in Section 3.1, the purpose of rule $(\mathsf{RD}_{\rightsquigarrow}^{IS})$ from Table 4 is to add some configurations to $IS^{\star}_{\rightsquigarrow}$ that will be used to check redundancy at each iteration of Algorithm 2. In ccp-{+} these additional configurations are not necessary. But before we arrive to this let us introduce some definitions first.

Definition 9. We say that γ goes with α to γ' with a *maximal weak transition*, written $\gamma \stackrel{\alpha}{\Longrightarrow}_{\max} \gamma'$, iff $\gamma \stackrel{\alpha}{\Longrightarrow} \gamma' \not\longrightarrow$.

The definition above reflects the fact that when $\gamma \stackrel{\alpha}{\Longrightarrow}_{\max} \gamma'$ then γ' has no more information to deduce without the aid of the environment, namely no further reduction ($\longrightarrow$) is possible. As $\Longrightarrow$, the maximal weak transition relation $\Longrightarrow_{\max}$ is sound and complete.

Lemma 5. (Soundness) If $\langle P, c\rangle \stackrel{\alpha}{\Longrightarrow}_{\max} \langle P', c'\rangle$ then $\langle P, c \sqcup \alpha\rangle \Longrightarrow_{\max} \langle P', c'\rangle$. (Completeness) If $\langle P, c \sqcup a\rangle \Longrightarrow_{\max} \langle P', c'\rangle$ then there exists α and b s.t. $\langle P, c\rangle \stackrel{\alpha}{\Longrightarrow}_{\max} \langle P', c''\rangle$ where $\alpha \sqcup b = a$ and $c'' \sqcup b = c'$.

PROOF. Follows from the correctness of $\Longrightarrow$ (Lemma 2) and from the fact that $LTS_{\longrightarrow}(\{\langle P, c\rangle\})$ is finite. □

As one would expect, $\Longrightarrow_{\max}$ can also be used to compute $\dot{\approx}_{sb}$ and the complexity of the procedure is equivalent to the case of $\Longrightarrow$ (Theorem 2).

Theorem 3. [5] Let γ and γ' be two ccp configurations. Let $IS = \{\gamma, \gamma'\}$, let $\mathcal{P}$ be the output $\texttt{weak-pr-ccp}(IS, \Longrightarrow_{\max})$ in Definition 7. Then

- $\gamma \,\mathcal{P}\, \gamma'$ iff $\gamma \,\dot{\approx}_{sb}\, \gamma'$.
- $\texttt{weak-pr-ccp}(IS, \Longrightarrow_{\max})$ may take exponential time in the size of $\texttt{Config}_{\longrightarrow}(IS)$.

PROOF. Follows from the correctness of $\Longrightarrow_{\max}$ (Lemma 5), the results in [5] and Theorem 2. □

Nevertheless, in ccp-{+}, the maximal weak transitions $\Longrightarrow_{\max}$ satisfy a particular property that allow us to erase the redundant transitions w.r.t. $\dot{\approx}_{sb}$ before computing $\dot{\approx}_{sb}$ itself.

Proposition 3. Let $\gamma = \langle P, c\rangle \in Conf_{\texttt{ccp}-\{+\}}$. Let $t_1 = \gamma \stackrel{\alpha}{\Longrightarrow}_{\max} \langle P_1, c_1\rangle$ and $t_2 = \gamma \stackrel{\beta}{\Longrightarrow}_{\max} \langle P_2, c_2\rangle$. We have that $\alpha \sqsubset \beta$ and $\langle P_1, c_1 \sqcup \beta\rangle \longrightarrow^* \langle P', c_2\rangle \not\longrightarrow$ iff $t_1 \succ_{\dot{\approx}_{sb}} t_2$.

PROOF. (⇒) By soundness on t_1 we have $\langle P, c \sqcup \alpha\rangle \Longrightarrow_{\max} \langle P_1, c_1\rangle$ then by definition $\langle P, c \sqcup \alpha\rangle \Longrightarrow \langle P_1, c_1\rangle$ now by monotonicity $\langle P, c \sqcup \beta\rangle \Longrightarrow \langle P_1, c_1 \sqcup \beta\rangle$ and then $\langle P, c \sqcup \beta\rangle \longrightarrow^* \langle P_1, c_1 \sqcup \beta\rangle$ then by Lemma 4 $\langle P, c \sqcup \beta\rangle \dot{\approx}_{sb} \langle P_1, c_1 \sqcup \beta\rangle$. Using a similar reasoning on t_2 we can conclude that $\langle P, c \sqcup \beta\rangle \dot{\approx}_{sb} \langle P_2, c_2\rangle$ and by transitivity $\langle P_1, c_1 \sqcup \beta\rangle \dot{\approx}_{sb} \langle P_2, c_2\rangle$. Finally take $t' = (\gamma, \beta, \langle P_1, c_1 \sqcup \beta\rangle)$, hence we can conclude that $t_1 \succ_{\dot{\approx}_{sb}} t_2$ since $t_1 \succ_D t'$ and $\langle P_1, c_1 \sqcup \beta\rangle \dot{\approx}_{sb} \langle P_2, c_2\rangle$.

(⇐) Assume that $t_1 \succ_{\dot{\approx}_{sb}} t_2$ then there exists $t' = (\gamma, \beta, \langle P_1, c'\rangle)$ such that $t_1 \succ_D t'$ and $\langle P_1, c'\rangle \dot{\approx}_{sb} \langle P_2, c_2\rangle$. By $t_1 \succ_D t'$ we know that $\alpha \sqsubset \beta$ and $c' = c_1 \sqcup \beta$. Now since $\langle P_2, c_2\rangle \not\longrightarrow$ by definition of $\Longrightarrow_{\max}$, therefore by condition (i) of $\dot{\approx}_{sb}$ we have $c' \sqsubseteq c_2$. Moreover, $\langle P_1, c'\rangle \longrightarrow^* \langle P', c_3\rangle$ where $c_2 \sqsubseteq c_3$. By contradiction let $c_2 \neq c_3$ then $c_2 \sqsubset c_3$, thus there is e s.t. $\langle P_1, c'\rangle \Downarrow_e$ but since $\langle P_2, c_2\rangle \not\longrightarrow$ then $\langle P_2, c_2\rangle \not\Downarrow_e$ and so $\langle P_1, c'\rangle \not\dot{\approx}_{sb} \langle P_2, c_2\rangle$, an absurd. Thus $c_3 = c_2$ hence $\langle P_1, c'\rangle \longrightarrow^* \langle P', c_2\rangle \not\longrightarrow$. □

Algorithm 3 $\texttt{weak-pr-dccp}(IS)$

Initialization

1. Compute $G = LTS_{\Longrightarrow_{\max}}(IS)$ using the rules $(\mathsf{IS}^{IS}_{\Longrightarrow_{\max}})$ and $(\mathsf{RS}^{IS}_{\Longrightarrow_{\max}})$,
2. $G' = \texttt{remRed}(G)$ where the graph $\texttt{remRed}(G)$ results from removing from G the redundant transitions w.r.t. $\dot{\approx}_{sb}$,
3. $\mathcal{P}^0 = \{B_1\} \ldots \{B_m\}$ is a partition of $\texttt{V}(G')$ where γ and γ' are in B_i iff they satisfy the same weak barbs ($\Downarrow_e$),

Iteration $\mathcal{P}^{n+1} := \mathbf{F}_{\Longrightarrow_{\max}}(\mathcal{P}^n)$ as defined in Equation 1
Termination If $\mathcal{P}^n = \mathcal{P}^{n+1}$ then return $\mathcal{P}^n$.

Using this property we can define a new procedure for deciding $\dot{\approx}_{sb}$ that does not use Rule $(\mathsf{RD}^{IS}_{\rightsquigarrow})$ since redundancy can be checked and erased using Proposition 3 (Algorithm 3, Step 2).

The key idea is that in order to compute $\dot{\approx}_{sb}$, with the redundancy removed, it suffices to refine the partitions using $\mathbf{F}_{\Longrightarrow_{\max}}(\mathcal{P})$ (defined by Equation 1) instead of $\mathbf{IR}_{\Longrightarrow_{\max}}(\mathcal{P})$. The Algorithm 3 can be used to decide $\dot{\approx}_{sb}$ for configurations in $Conf_{\texttt{ccp}-\{+\}}$ with polynomial time.

Theorem 4. Let γ and γ' be two ccp-{+} configurations. Let $IS = \{\gamma, \gamma'\}$, let $\mathcal{P}$ be the output of $\texttt{weak-pr-dccp}(IS)$ in Algorithm 3 and let $N = |\texttt{Config}_{\longrightarrow}(IS)|$. Then

- $\gamma \,\mathcal{P}\, \gamma'$ iff $\gamma \,\dot{\approx}_{sb}\, \gamma'$.
- $\texttt{weak-pr-dccp}(IS)$ takes $O(N^3)$ time and uses $O(N^2)$ space.

PROOF. The first item follows from the Theorem 2 and Proposition 3. As for the second item:
(Step 1) $G = LTS_{\Longrightarrow_{\max}}(IS)$ takes $O(N^2)$ time and space since $\Longrightarrow_{\max}$ will add, at most, a transition from each element in $\texttt{V}(G)$ to every other configuration in $\texttt{V}(G)$ and $|\texttt{V}(G)| = |\texttt{Config}_{\longrightarrow}(IS)| = N$.
(Step 2) Each node in $\texttt{V}(G)$ has at most $N-1$ outgoing transitions, then $G' = \texttt{remRed}(G)$ takes $O((N-1) * (N-1)) = O(N^2)$ per node, thus this step takes $O(N^2 * N) = O(N^3)$ time.
(Step 3) $\mathcal{P}^0$ can be created in $O(N^2)$ by definition of $\Longrightarrow_{\max}$.
(Iteration) Using the procedure from Tarjan et al. [22], this step takes $O(|E| \log |V|)$ time and uses $O(|E|)$ space. Therefore, since $|\texttt{V}(G)| = N$ and $|\texttt{E}(G)| = N^2$, hence we have $O(N^2 \log N)$ and $O(N^2)$ space.
We can conclude that $\texttt{weak-pr-dccp}(IS)$ takes $O(N^3)$ time and uses $O(N^2)$ space. □

Thanks to Proposition 3, by removing redundant transitions, we can solve the problem of checking bisimilarity for ccp-{+} with the standard solutions for checking bisimilarity. In Algorithm 3, we have used the "classical" partition refinement, but different, more effective solutions, are possible. For instance, executing the algorithm in [13] (after having removed all the redundant transitions) would require at most $O(|E| + |V|)$ steps. Note however that, due to the closure needed for weak transitions (Table 3), $|E|$ is usually quadratic w.r.t. the number of states $|V|$. In the following section, we introduce a novel procedure which avoids such expensive closure.

5. USING THE COMPACT INPUT-OUTPUT SETS FOR VERIFYING OBSERVATIONAL EQUIVALENCE IN ccp-{+}

In the previous section we improved the ccp exponential-time decision procedure for $\dot{\approx}_{sb}$ to obtain a polynomial-time procedure for the special case of the summation-free fragment ccp-{+}. (Recall that in $\dot{\approx}_{sb}$, the relation $\dot{\approx}_{sb}$ coincides with the standard notion of observational equivalence.)

In this section, we will present an alternative approach for verifying observational equivalence for ccp-{+}that improves on the time and space complexity of Algorithm 3.

Roughly speaking our approach consists in reducing the problem of whether two given ccp-{+}-configurations γ, γ' are in $\dot{\approx}_{sb}$ to the problem of whether γ and γ' have the same minimal finite representation of the set of weak barbs they satisfy in every possible context.

5.1 Weak bisimilarity and barb equivalence

First we will show that, in ccp-{+}, we can give characterization of $\dot{\approx}_{sb}$ in terms of the simpler notion of weak-barb equivalence defined below. Intuitively, two configurations are saturated weakly bisimilar if and only if for *every* possible augmentation of their stores, the resulting configurations satisfy the same weak barbs. More precisely,

Definition 10. (Barb equivalence) $\langle P, c\rangle$ and $\langle Q, d\rangle$ are (weak) barbed equivalent, written $\langle P, c\rangle \sim_{wb} \langle Q, d\rangle$, iff

$$\forall e, \alpha \in Con_0.\ \langle P, c \sqcup e\rangle \Downarrow_\alpha \Leftrightarrow \langle Q, d \sqcup e\rangle \Downarrow_\alpha$$

The full characterization of $\dot{\approx}_{sb}$ in terms of weak-barbed equivalence is given next. The proof relies on the intrinsic confluent nature of ccp-{+} (Proposition 2).

Theorem 5. $\langle P, c\rangle \dot{\approx}_{sb} \langle Q, d\rangle$ iff $\langle P, c\rangle \sim_{wb} \langle Q, d\rangle$

PROOF. **(⇒)** Assume that $\langle P, c\rangle \dot{\approx}_{sb} \langle Q, d\rangle$ then by condition (i) of $\dot{\approx}_{sb}$ (Definition 3) we have $\forall \alpha \in Con_0.\langle P, c\rangle \Downarrow_\alpha \Leftrightarrow \langle Q, d\rangle \Downarrow_\alpha$, hence in combination with condition (iii) we can conclude $\langle P, c \sqcup e\rangle \Downarrow_\alpha \Leftrightarrow \langle Q, d \sqcup e\rangle \Downarrow_\alpha$.

(⇐) Let $\mathcal{R} = \{(\langle P, c\rangle, \langle Q, d\rangle) \mid \forall e, \alpha \in Con_0.\ \langle P, c \sqcup e\rangle \Downarrow_\alpha \Leftrightarrow \langle Q, d \sqcup e\rangle \Downarrow_\alpha\}$, we prove that $\mathcal{R}$ is a weak saturated barbed bisimulation:
(i) Take $e = true$ then $\forall \alpha \in Con_0.\langle P, c\rangle \Downarrow_\alpha \Leftrightarrow \langle Q, d\rangle \Downarrow_\alpha$.
(ii) Assume that $\langle P, c\rangle \longrightarrow^* \langle P', c'\rangle$, by Lemma 4 $\langle P, c\rangle \dot{\approx}_{sb} \langle P', c'\rangle$ hence by (⇒) we can conclude that $\langle P', c'\rangle \mathcal{R} \langle Q, d\rangle$.
(iii) Assume $\langle P, c\rangle \mathcal{R} \langle Q, d\rangle$ then for all e' we have $\langle P, c \sqcup e'\rangle \mathcal{R} \langle Q, d \sqcup e'\rangle$ just by taking $e = e'$. □

We shall show a compact representation of the set of weak barbs of a configuration under any possible context. First we introduce some convenient notation for this purpose. The set $[\![\langle P, c\rangle]\!]$ will contain pairs of the form (α, e).

Definition 11. (Input-Output set) The *input-output* set of a given configuration $\langle P, c\rangle$ is defined as follows:

$$[\![\langle P, c\rangle]\!] \stackrel{\text{def}}{=} \{(\alpha, e) \mid \langle P, c \sqcup \alpha\rangle \Downarrow_e\}$$

Intuitively, each pair $(\alpha, e) \in [\![\langle P, c\rangle]\!]$ denotes a stimulus-response, or input-output, interaction of $\gamma = \langle P, c\rangle$: If the environment adds α to the store of γ, the resulting configuration $\langle P, c \sqcup \alpha\rangle$ may evolve, without any further interaction with the environment, into a configuration whose store entails e. In other words $\langle P, c \sqcup \alpha\rangle \Downarrow e$. We can think of e as piece of information that $\langle P, c \sqcup \alpha\rangle$ may produce.

The following corollary is an immediate consequence of the definitions.

Corollary 1. $[\![\langle P, c\rangle]\!] = [\![\langle Q, d\rangle]\!]$ iff $\langle P, c\rangle \sim_{wb} \langle Q, d\rangle$

We now introduce the notion of relevant input-output pair.

Definition 12. (Relevant Pair) Let (α, e) and (β, e') be elements from $Con_0 \times Con_0$. We say that (α, e) is more *relevant* than (β, e'), written $(\alpha, e) \succeq (\beta, e')$, iff $\alpha \sqsubseteq \beta$ and $e' \sqsubseteq (e \sqcup \beta)$. Similarly, given $p = (\beta, e')$ s.t. $p \in \mathcal{S}$, we say that the pair p is *irrelevant* in $\mathcal{S}$ if there is a pair $(\alpha, e) \in \mathcal{S}$ more relevant than p, else p is said to be *relevant* in $\mathcal{S}$.

Recall the stimulus-response intuition given above. In other words, the pair (β, e') is *irrelevant* in a given input-output set if there exists another pair (α, e) in the set that represents the need of less stimulus from the environment, hence the condition $\alpha \sqsubseteq \beta$, to produce at least as much information, with the possible exception of information that β may entail but α does not. Hence $e' \sqsubseteq e \sqcup \beta$.

We now list two important properties of $\succeq$ that will be useful later on. The set $[\![\langle P, c\rangle]\!]$ is closed w.r.t. $\succeq$.

Proposition 4. Let $(\alpha, e) \in [\![\langle P, c\rangle]\!]$. If $(\alpha, e) \succeq (\beta, e')$ then $(\beta, e') \in [\![\langle P, c\rangle]\!]$.

PROOF. By definition $\langle P, c \sqcup \alpha\rangle \Downarrow_e$ then by monotonicity $\langle P, c \sqcup \beta\rangle \Downarrow_{e'}$ since $e' \sqsubseteq (e \sqcup \beta)$, therefore $(\beta, e') \in [\![\langle P, c\rangle]\!]$. □

Moreover, the relation $\succeq$ is well-founded. More precisely,

Proposition 5. There is no infinite *strictly* descending chain $p_1 \succ p_2 \succ \ldots$.

PROOF. Follows from the well-foundedness of $\sqsubseteq$ (Remark 1) □

5.2 A canonical representation of ccp-{+} configurations

Clearly $[\![\langle P, c\rangle]\!]$ may be infinite due potential existence of infinitely many arbitrary stimuli (inputs). By using the labeled transition semantics (Table 2) we shall show that we do not need consider arbitrary inputs but only the minimal ones. Recall that in $\gamma \xrightarrow{\alpha} \gamma'$ the label α represents the minimal information needed to evolve from γ to γ'.

Definition 13. The *labeled-based input-output set* of a configuration $\langle P, c\rangle$, denoted as $\mathcal{M}(\langle P, c\rangle)$, is inductively defined as follows:

$\{(true, c)\} \cup \bigcup_{\langle P,c\rangle \xrightarrow{\alpha} \langle P',c'\rangle} (\{(\alpha, c')\} \cup (\alpha \otimes \mathcal{M}(\langle P', c'\rangle)))$

where $\otimes : Con_0 \times 2^{Con_0 \times Con_0} \longrightarrow 2^{Con_0 \times Con_0}$ is defined as $\alpha \otimes \mathcal{S} \stackrel{\text{def}}{=} \{(\alpha \sqcup \beta, e) \mid (\beta, e) \in \mathcal{S})\}$.

Nevertheless, labeled-based input-output sets do not give us a fully-abstract representation of the input-output sets because of the existence of irrelevant pairs. By excluding these pairs we obtain a compact and fully-abstract representation of input-output sets.

Definition 14. (Compact input-output set) The *compact* input-output set of a configuration $\langle P, c\rangle$ is defined as follows:

$$\mathcal{M}^C(\langle P, c\rangle) \stackrel{\text{def}}{=} \{(\alpha, e) \mid (\alpha, e) \in \mathcal{M}(\langle P, c\rangle) \text{ and } (\alpha, e) \text{ is relevant in } \mathcal{M}(\langle P, c\rangle)\}$$

We shall now show the full-abstraction of the compact input-output sets. We need the following lemmata. First, compact sets are closed under weak transitions ($\Longrightarrow$). More precisely:

Proposition 6. If $\langle P, c\rangle \stackrel{\alpha}{\Longrightarrow} \langle P', c'\rangle$ then $(\alpha, c') \in \mathcal{M}(\langle P, c\rangle)$.

PROOF. By induction on the depth of the inference of $\langle P, c\rangle \stackrel{\alpha}{\Longrightarrow} \langle P', c'\rangle$.

- Using Rule R-Tau we have $\langle P, c\rangle \Longrightarrow \langle P, c\rangle$ and $(true, c) \in \mathcal{M}(\langle P, c\rangle)$ by definition.
- Using Rule R-Label we have $\langle P, c\rangle \xrightarrow{\alpha} \langle P', c'\rangle$ and $(\alpha, c') \in \mathcal{M}(\langle P, c\rangle)$ by definition.
- Using Rule R-Add we have $\langle P, c\rangle \stackrel{\alpha''}{\Longrightarrow} \langle P'', c''\rangle \stackrel{\alpha'}{\Longrightarrow} \langle P', c'\rangle$ where $\alpha' \sqcup \alpha'' = \alpha$. Then by induction hypothesis $(\alpha'', c'') \in \mathcal{M}(\langle P, c\rangle)$ and $(\alpha', c') \in \mathcal{M}(\langle P'', c''\rangle)$, hence by definition of $\mathcal{M}(\langle P, c\rangle)$ we have $(\alpha' \sqcup \alpha'', c') \in \mathcal{M}(\langle P, c\rangle)$ so $(\alpha, c') \in \mathcal{M}(\langle P, c\rangle)$.

□

The following proposition states that whenever a pair (α, e) belongs to $\mathcal{M}(\langle P, c\rangle)$, it means that e can be reached from $\langle P, c \sqcup \alpha\rangle$ without aid of the environment.

Proposition 7. If $(\alpha, e) \in \mathcal{M}(\langle P, c\rangle)$ then $\langle P, c \sqcup \alpha\rangle \longrightarrow^* \langle P', e\rangle$

PROOF. By definition of $\mathcal{M}(\langle P, c\rangle)$, since $(\alpha, e) \in \mathcal{M}(\langle P, c\rangle)$ then there exist $\alpha_1, \ldots, \alpha_n$ such that $\alpha = \bigsqcup_{i=1}^n \alpha_n$ and $\langle P, c\rangle \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} \langle P', e\rangle$. Hence by soundness on each transition $\langle P, c \sqcup \bigsqcup_{i=1}^n \alpha_n\rangle = \langle P, c \sqcup \alpha\rangle \longrightarrow^* \langle P', e\rangle$. □

We can now prove our main result, given two configurations $\langle P, c\rangle$ and $\langle Q, d\rangle$, they are observationally equivalent if and only if their compact input-output sets are identical. We split the proof in the following two lemmata.

Lemma 6. If $\mathcal{M}^C(\langle P, c\rangle) = \mathcal{M}^C(\langle Q, d\rangle)$ then $[\![\langle P, c\rangle]\!] = [\![\langle Q, d\rangle]\!]$

PROOF. Let us assume that $(\alpha, \beta) \in [\![\langle P, c\rangle]\!]$ then by definition $\langle P, c \sqcup \alpha\rangle \Downarrow_\beta$ hence there exists P' and β' such that $\langle P, c \sqcup \alpha\rangle \longrightarrow^* \langle P', \beta'\rangle$ and $\beta \sqsubseteq \beta'$. By Lemma 3 we have $\langle P, c \sqcup \alpha\rangle \Longrightarrow \langle P', \beta'\rangle$, then by completeness of $\Longrightarrow$ (Lemma 2) there exist α', b s.t. $\langle P, c\rangle \stackrel{\alpha'}{\Longrightarrow} \langle P', c'\rangle$ where $\alpha' \sqcup b = \alpha$ and $c' \sqcup b = \beta'$ **(1)**. Now by Proposition 6 we know $(\alpha', c') \in \mathcal{M}(\langle P, c\rangle)$, then since $\succeq$ is well-founded (Proposition 5) there is (α'', c'') that is relevant in $\mathcal{M}(\langle P, c\rangle)$ (then it belongs to $\mathcal{M}^C(\langle P, c\rangle)$) such that $(\alpha'', c'') \succeq (\alpha', c')$, namely $\alpha'' \sqsubseteq \alpha'$ (or equivalently $\exists x.(\alpha'' \sqcup x) = \alpha'$ **(2)**) and $c' \sqsubseteq (c'' \sqcup \alpha')$. Given that $(\alpha'', c'') \in \mathcal{M}^C(\langle P, c\rangle)$ then by hypothesis $(\alpha'', c'') \in \mathcal{M}^C(\langle Q, d\rangle)$, this means also that $(\alpha'', c'') \in \mathcal{M}(\langle Q, d\rangle)$ and by Proposition 7 we know that $\langle Q, d \sqcup \alpha''\rangle \longrightarrow^* \langle Q', c''\rangle$. By monotonicity we have the following transition $\langle Q, d \sqcup \alpha'' \sqcup x \sqcup b\rangle \longrightarrow^* \langle Q', c'' \sqcup x \sqcup b\rangle$, now notice that from **(1)** and **(2)** we have $(d \sqcup \alpha'' \sqcup x \sqcup b) = (d \sqcup \alpha' \sqcup b) = (d \sqcup \alpha)$ then $\langle Q, d \sqcup \alpha\rangle \longrightarrow^* \langle Q', c'' \sqcup x \sqcup b\rangle$. Finally, we have to prove that $\beta \sqsubseteq (c'' \sqcup x \sqcup b)$ to conclude that $(\alpha, \beta) \in [\![\langle Q, d\rangle]\!]$, for that purpose, recall that $\beta \sqsubseteq \beta' = (c' \sqcup b) \sqsubseteq (c'' \sqcup \alpha' \sqcup b)$ and since $(c'' \sqcup \alpha'' \sqcup x \sqcup b) = (c'' \sqcup x \sqcup b)$ then $\beta \sqsubseteq (c'' \sqcup x \sqcup b)$ and so $(\alpha, \beta) \in [\![\langle Q, d\rangle]\!]$. □

Lemma 7. If $[\![\langle P, c\rangle]\!] = [\![\langle Q, d\rangle]\!]$ then $\mathcal{M}^C(\langle P, c\rangle) = \mathcal{M}^C(\langle Q, d\rangle)$

PROOF. Assume that $(\alpha, \beta) \in \mathcal{M}^C(\langle P, c\rangle)$ our goal is to prove that $(\alpha, \beta) \in \mathcal{M}^C(\langle Q, d\rangle)$. By definition (α, β) is relevant in $\mathcal{M}(\langle P, c\rangle)$, moreover, by Proposition 7 we have $\langle P, c \sqcup \alpha\rangle \longrightarrow^* \langle P', \beta\rangle$ then by definition $(\alpha, \beta) \in [\![\langle P, c\rangle]\!]$ and by hypothesis $(\alpha, \beta) \in [\![\langle Q, d\rangle]\!]$. This means that $\langle Q, d \sqcup \alpha\rangle \Downarrow_\beta$ then there exists Q', d' s.t. $\langle Q, d \sqcup \alpha\rangle \longrightarrow^* \langle Q', d'\rangle$ where $\beta \sqsubseteq d'$. By Lemma 3 we have $\langle Q, d \sqcup \alpha\rangle \Longrightarrow \langle Q', d'\rangle$, now by completeness of $\Longrightarrow$ (Lemma 2) there exist α', b s.t. $\langle Q, d\rangle \stackrel{\alpha'}{\Longrightarrow} \langle Q', d''\rangle$ where $(\alpha' \sqcup b) = \alpha$ and $(d'' \sqcup b) = d'$. Now let us assume by means of contradiction that $\alpha' \neq \alpha$. By soundness of $\Longrightarrow$ (Lemma 2) we have $\langle Q, d \sqcup \alpha'\rangle \Longrightarrow \langle Q', d''\rangle$ then by Lemma 3 we get $\langle Q, d \sqcup \alpha'\rangle \longrightarrow^* \langle Q', d''\rangle$ hence $(\alpha', d'') \in [\![\langle Q, d\rangle]\!]$. By hypothesis then $(\alpha', d'') \in [\![\langle P, c\rangle]\!]$, now this means that $\langle P, c \sqcup \alpha'\rangle \longrightarrow^* \langle P'', e\rangle$ where $d'' \sqsubseteq e$ (equivalently $\exists z.(d'' \sqcup z) = e$). By Lemma 3 we get that $\langle P, c \sqcup \alpha'\rangle \Longrightarrow \langle P'', e\rangle$ and by completeness there exist x, b' s.t. $\langle P, c\rangle \stackrel{x}{\Longrightarrow} \langle P'', c'\rangle$ where $(x \sqcup b') = \alpha'$ and $c' \sqcup b' = e$. Using Lemma 6 we have that $(x, c') \in \mathcal{M}(\langle P, c\rangle)$, now we will prove that $(x, c') \succeq (\alpha, \beta)$, namely $x \sqsubseteq \alpha$ and $\beta \sqsubseteq (\alpha \sqcup c')$. Recall that $x \sqsubseteq \alpha' \sqsubseteq \alpha$, now for the latter condition $(\alpha \sqcup c') = (\alpha' \sqcup b \sqcup c') = (x \sqcup b' \sqcup b \sqcup c') = (x \sqcup b \sqcup e)$ then since $d'' \sqsubseteq e$ we can check that $\beta \sqsubseteq d' \sqsubseteq (d' \sqcup x) = (d'' \sqcup b \sqcup x) \sqsubseteq (e \sqcup b \sqcup x) = (\alpha \sqcup c')$. Thus, this would mean that (α, β) is irrelevant in $\mathcal{M}(\langle P, c\rangle)$, a contradiction, therefore $\alpha' = \alpha$ and by consequence $d'' = d'$. Therefore, we know that $\langle Q, d\rangle \stackrel{\alpha}{\Longrightarrow} \langle Q', d'\rangle$, now let us assume by contradiction that $d' \neq \beta$ (i.e. $\beta \sqsubset d'$). By soundness and Lemma 3 we have that $\langle Q, d \sqcup \alpha\rangle \longrightarrow^* \langle Q', d'\rangle$, this means that $(\alpha, d') \in [\![\langle Q, d\rangle]\!]$. By hypothesis then $(\alpha, d') \in [\![\langle P, c\rangle]\!]$ so there exist P_1, c_1 s.t. $\langle P, c \sqcup \alpha\rangle \longrightarrow^* \langle P_1, c_1\rangle$ and $d' \sqsubseteq c_1$. By Lemma 3 then $\langle P, c \sqcup \alpha\rangle \Longrightarrow \langle P_1, c_1\rangle$, now by completeness, there exist y, b'' s.t. $\langle P, c\rangle \stackrel{y}{\Longrightarrow} \langle P_1, c_1'\rangle$ where $y \sqcup b'' = \alpha$ and $c_1' \sqcup b'' = c_1$. Using Lemma 6 we get that $(y, c_1') \in \mathcal{M}(\langle P, c\rangle)$. Now let us prove that $(y, c_1') \succeq (\alpha, \beta)$, namely $y \sqsubseteq \alpha$ and $\beta \sqsubseteq (\alpha \sqcup c_1')$. The first condition follows from $y \sqcup b'' = \alpha$ and for the latter condition we proceed as follows $\beta \sqsubset d' \sqsubseteq c_1 \sqsubseteq (c_1 \sqcup y) = (c_1' \sqcup b'' \sqcup y) = (c_1' \sqcup \alpha)$. Again, this

would mean that (α,β) is irrelevant in $\mathcal{M}(\langle P,c\rangle)$, a contradiction, therefore $d'=\beta$. Hence, we know that $\langle Q,d\rangle \stackrel{\alpha}{\Longrightarrow} \langle Q',\beta\rangle$ then by Proposition 6 $(\alpha,\beta)\in\mathcal{M}(\langle Q,d\rangle)$. Finally, let us assume by contradiction that (α,β) is irrelevant in $\mathcal{M}(\langle Q,d\rangle)$. Then there exists $(\alpha_1,\beta_1)\in\mathcal{M}(\langle Q,d\rangle)$ such that $(\alpha_1,\beta_1)\succeq(\alpha,\beta)$, namely $\alpha_1\sqsubseteq\alpha$ (equivalently $\exists z'.\alpha_1\sqcup z'=\alpha$) and $\beta\sqsubseteq\alpha\sqcup\beta_1$. By Proposition 7 we have that $\langle Q,d\sqcup\alpha_1\rangle\longrightarrow^*\langle Q_1,\beta_1\rangle$, then $(\alpha_1,\beta_1)\in[\![\langle Q,d\rangle]\!]$ and by hypothesis $(\alpha_1,\beta_1)\in[\![\langle P,c\rangle]\!]$. This means that $\langle P,c\sqcup\alpha_1\rangle\longrightarrow^*\langle P_2,c_2\rangle$ where $\beta_1\sqsubseteq c_2$, now by Lemma 3 we get $\langle P,c\sqcup\alpha_1\rangle\Longrightarrow\langle P_2,c_2\rangle$. By completeness of $\Longrightarrow$ there exist a,b_1 s.t. $\langle P,c\rangle\stackrel{a}{\Longrightarrow}\langle P_2,c_2'\rangle$ where $(a\sqcup b_1)=\alpha_1$ and $(c_2'\sqcup b_1)=c_2$. Hence, by Proposition 6 we know that $(a,c_2')\in\mathcal{M}(\langle P,c\rangle)$. Now let us prove that $(a,c_2')\succeq(\alpha,\beta)$ namely $a\sqsubseteq\alpha$ and $\beta\sqsubseteq(\alpha\sqcup c_2')$. First $a\sqsubseteq\alpha_1\sqsubseteq\alpha$, for the latter condition we proceed as follows $(\alpha\sqcup c_2')=(\alpha_1\sqcup z'\sqcup c_2')=(a\sqcup b_1\sqcup z'\sqcup c_2')=(c_2\sqcup z')$ and since $\beta_1\sqsubseteq c_2$ and $\alpha\sqsubseteq c_2$ then $\beta\sqsubseteq(\alpha\sqcup\beta_1)\sqsubseteq(c_2\sqcup z')=(\alpha\sqcup c_2')$. Once again, this would mean that (α,β) is irrelevant in $\mathcal{M}(\langle P,c\rangle)$, a contradiction. Finally, we can conclude that (α,β) is relevant in $\mathcal{M}(\langle Q,d\rangle)$ therefore $(\alpha,\beta)\in\mathcal{M}^C(\langle Q,d\rangle)$. □

Using the these lemmata above we conclude the following theorem.

Theorem 6. $[\![\langle P,c\rangle]\!]=[\![\langle Q,d\rangle]\!]$ iff $\mathcal{M}^C(\langle P,c\rangle)=\mathcal{M}^C(\langle Q,d\rangle)$

PROOF. Using Lemma 6 and Lemma 7. □

By combining Theorem 5 and Theorem 6 we get a simple decision procedure for $\dot{\approx}_{sb}$ by reducing weak saturated equivalence between two given configuration to the set equivalence of the corresponding compact input-output representations. The complexity of this procedure is clearly determined by the complexity of constructions of the compact input-output sets.

Theorem 7. Let γ and γ' be two ccp-{+} configurations. Let $IS=\{\gamma,\gamma'\}$ and let $N=|\texttt{Config}_{\longrightarrow}(IS)|$. Then

- $\mathcal{M}^C(\gamma)=\mathcal{M}^C(\gamma')$ iff $\gamma\,\dot{\approx}_{sb}\,\gamma'$.
- Checking whether $\mathcal{M}^C(\gamma)=\mathcal{M}^C(\gamma')$ takes $O(N^2)$ time and uses $O(N)$ space.

PROOF. The first item follows from Follows from Theorem 5 and Theorem 6 and the second item is derived from the construction of $\mathcal{M}^C(\gamma)$ and $\mathcal{M}^C(\gamma')$. □

6. IMPROVING THE PARTITION REFINEMENT FOR CCP

In this section we show that in the general case of ccp systems, the strategy from Section 4.2 can be used for their ccp-{+} components, thus producing a $IS^{\star}_{\rightsquigarrow}$ which may be significant smaller (although the worst case remains exponential).

Given a configuration γ the idea is to detect when an evolution of γ, i.e. a γ' s.t. $\gamma\stackrel{\alpha_1}{\Longrightarrow}\ldots\stackrel{\alpha_k}{\Longrightarrow}\gamma'$, is a ccp-{+} configuration. This way we can avoid adding new configurations with Rule $(\mathsf{RD}^{IS}_{\rightsquigarrow})$ whenever $\gamma'\in Conf_{\texttt{ccp}-\{+\}}$, and redundancy can be then checked using Proposition 3.

Definition 15. (Improved partition refinement for ccp) We define the procedure $\texttt{imp-weak-pr-ccp}(IS,\rightsquigarrow)$ by replacing the rules in Step 1 of $\texttt{weak-pr-ccp}(IS,\rightsquigarrow)$ from Definition 7 with the rules defined in Table 5.

Using this algorithm we can decide $\dot{\approx}_{sb}$ in a more efficient manner, although, in the worst-case scenario, still with exponential time. This follows from Proposition 3 and Theorem 1.

Theorem 8. Let γ and γ' be two ccp configurations. Let $IS=\{\gamma,\gamma'\}$ and let $\mathcal{P}$ be the output of $\texttt{imp-weak-pr-ccp}(IS,\Longrightarrow)$ in Definition 15. Then

- $\gamma\,\mathcal{P}\,\gamma'$ iff $\gamma\,\dot{\approx}_{sb}\,\gamma'$.
- $\texttt{imp-weak-pr-ccp}(IS,\Longrightarrow)$ may take exponential time in the size of $\texttt{Config}_{\longrightarrow}(IS)$.

PROOF. The first item follows Proposition 3 and Theorem 1. The second item follows from [4]. □

It is clear that $\texttt{imp-weak-pr-ccp}(IS,\Longrightarrow)$ performs better than $\texttt{weak-pr-ccp}(IS,\Longrightarrow)$ since the new procedure avoids adding new states whenever they are not necessary to check redundancy w.r.t. $\dot{\approx}_{sb}$. Unfortunately, this improvement does not escape from the worst-case scenario of $\texttt{weak-pr-ccp}(IS,\Longrightarrow)$. Nevertheless, this approach shows the applicability of the strategy developed in Section 4.2.

7. CONCLUSIONS AND FUTURE WORK

In this paper we explored the use of the partition refinement algorithm for ccp from [4] and [5] for checking observational equivalence in the ccp-{+} fragment. We proved that this procedure takes exponential time and space (in the size of the set of reachable configurations) even for the restricted case of ccp-{+}. We then proposed two alternative methods for checking observational equivalence in ccp-{+} by exploiting some of the intrinsic properties of this fragment, in particular confluence. We proved that both procedures take polynomial time (in the size of the set of reachable configurations), thus significantly improving the exponential-time approach from [4, 5], which is, to the best of our knowledge the only algorithm for checking observational equivalence in ccp. Each of the two method has its advantages over the other. On the one hand, the algorithm from Section 4 uses significantly more time and space than the one from Section 5, however it can be easily adapted for verifying observational equivalence for the full ccp as shown in Section 6. On the other hand, the procedure from Section 5 takes less time and uses only linear space nevertheless there is no "trivial" adaptation for the full language since it does not use the partition refinement approach.

Most of the related work was already discussed in the introduction. As we mentioned in Section 4, it remains as a future work to consider more efficient partition refinement algorithms [13] to see whether the algorithm from Section 4 can be further improved. The challenge would be to find a more efficient version of $\Longrightarrow$ that can still be used for deciding $\dot{\approx}_{sb}$ and so it can be adapted to the case of the full ccp. Finally, we plan to investigate how the procedures here defined can be extended to different versions of ccp where the summation operator is not present, for instance timed ccp (tcc) [25], universal temporal ccp (utcc) [21] and epistemic ccp (eccp) [17]).

$$\text{(IS' }\Longrightarrow\text{)}\ \frac{\gamma \in IS}{\gamma \in IS^{\star}_{\rightsquigarrow}}\qquad \text{(RS' }\Longrightarrow\text{)}\ \frac{\gamma \in IS^{\star}_{\rightsquigarrow}\quad \gamma \overset{\alpha}{\rightsquigarrow} \gamma'}{\gamma' \in IS^{\star}_{\rightsquigarrow}}$$

$$\text{(opt-RD }\Longrightarrow\text{)}\ \frac{\gamma \in IS^{\star}_{\rightsquigarrow}\quad \gamma \notin Conf_{\mathtt{ccp}-\{+\}}\quad t_1 = \gamma \overset{\alpha}{\rightsquigarrow} \langle P_1, c_1\rangle\quad t_2 = \gamma \overset{\beta}{\rightsquigarrow} \langle P_2, c_2\rangle\quad \alpha \sqsubset \beta\quad c_2 = c_1 \sqcup \beta}{\langle P_1, c_2\rangle \in IS^{\star}_{\rightsquigarrow}}$$

Table 5: Rules for improved version of the partition refinement for ccp.

8. REFERENCES

[1] L. Aceto, A. Ingolfsdottir, and J. Srba. *Advanced Topics in Bisimulation and Coinduction*, chapter The Algorithmics of Bisimilarity, pages 100–172. Cambridge University Press, 2011.

[2] R. M. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous pi-calculus. In *CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, pages 147–162. Springer, 1996.

[3] A. Aristizabal, F. Bonchi, C. Palamidessi, L. Pino, and F. D. Valencia. Deriving labels and bisimilarity for concurrent constraint programming. In *FOSSACS*, LNCS, pages 138–152. Springer, 2011.

[4] A. Aristizabal, F. Bonchi, L. Pino, and F. D. Valencia. Partition refinement for bisimilarity in ccp. In *SAC*, pages 88–93. ACM, 2012.

[5] A. Aristizábal, F. Bonchi, L. F. Pino, and F. Valencia. Reducing weak to strong bisimilarity in ccp. In *ICE*, pages 2–16, 2012.

[6] M. Bartoletti and R. Zunino. A calculus of contracting processes. In *LICS*, pages 332–341. IEEE Computer Society, 2010.

[7] J. Bengtson, M. Johansson, J. Parrow, and B. Victor. Psi-calculi: Mobile processes, nominal data, and logic. In *LICS*, pages 39–48, 2009.

[8] F. Bonchi, F. Gadducci, and G. V. Monreale. Reactive systems, barbed semantics, and the mobile ambients. In *FOSSACS*, LNCS, pages 272–287, 2009.

[9] F. Bonchi, B. König, and U. Montanari. Saturated semantics for reactive systems. In *LICS*, pages 69–80. IEEE, 2006.

[10] A. Bouali and R. de Simone. Symbolic bisimulation minimisation. In *CAV*, volume 663 of *Lecture Notes in Computer Science*, pages 96–108. Springer, 1992.

[11] M. G. Buscemi and U. Montanari. Open bisimulation for the concurrent constraint pi-calculus. In *ESOP*, pages 254–268, 2008.

[12] F. S. de Boer, A. D. Pierro, and C. Palamidessi. Nondeterminism and infinite computations in constraint programming. *Theor. Comput. Sci.*, 151(1):37–78, 1995.

[13] A. Dovier, C. Piazza, and A. Policriti. An efficient algorithm for computing bisimulation equivalence. *Theor. Comput. Sci.*, 311(1-3):221–256, 2004.

[14] J.-C. Fernandez and L. Mounier. Verifying bisimulations "on the fly". In *FORTE*, pages 95–110. North-Holland, 1990.

[15] H. Garavel. Reflections on the future of concurrency theory in general and process calculi in particular. In *Proc. of LIX Colloquium on Emergent Trends in Concurrency Theory*, Electr. Notes Theor. Comput. Sci. 209, pages 149–164, 2008.

[16] P. C. Kanellakis and S. A. Smolka. Ccs expressions, finite state processes, and three problems of equivalence. In *PODC*, pages 228–240. ACM, 1983.

[17] S. Knight, C. Palamidessi, P. Panangaden, and F. D. Valencia. Spatial and epistemic modalities in constraint-based process calculi. In *CONCUR*, pages 317–332, 2012.

[18] N. P. Mendler, P. Panangaden, P. J. Scott, and R. A. G. Seely. A logical view of concurrent constraint programming. *Nord. J. Comput.*, 2(2):181–220, 1995.

[19] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag New York, Inc., 1980.

[20] R. Milner and D. Sangiorgi. Barbed bisimulation. In *ICALP*, LNCS, pages 685–695. Springer, 1992.

[21] C. Olarte and F. D. Valencia. Universal concurrent constraint programing: symbolic semantics and applications to security. In *SAC*, pages 145–150. ACM, 2008.

[22] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, Dec. 1987.

[23] C. Palamidessi, V. A. Saraswat, F. D. Valencia, and B. Victor. On the expressiveness of linearity vs persistence in the asychronous pi-calculus. In *LICS*, pages 59–68, 2006.

[24] L. Pino, F. Bonchi, and F. Valencia. Efficient computation of program equivalence for confluent concurrent constraint programming (technical report). Technical report, LIX, Ecole Polytechnique, 2013. `http://www.lix.polytechnique.fr/~luis.pino/files/minset-extended.pdf`.

[25] V. A. Saraswat, R. Jagadeesan, and V. Gupta. Foundations of timed concurrent constraint programming. In *LICS*, pages 71–80. IEEE, 1994.

[26] V. A. Saraswat, M. C. Rinard, and P. Panangaden. Semantic foundations of concurrent constraint programming. In *POPL*, pages 333–352. ACM Press, 1991.

Real-Reward Testing for Probabilistic Processes

Yuxin Deng[1*] Rob van Glabbeek[2] Matthew Hennessy[3†] Carroll Morgan[4‡]

[1] Shanghai Jiao Tong University, China

[2] NICTA, Sydney, Australia[§]

[3] Trinity College Dublin, Ireland

[2,4] University of New South Wales, Sydney, Australia

We introduce a notion of real-valued reward testing for probabilistic processes by extending the traditional nonnegative-reward testing with negative rewards. In this richer testing framework, the may- and must preorders turn out to be inverses. We show that for convergent processes with finitely many states and transitions, but not in the presence of divergence, the real-reward must-testing preorder coincides with the nonnegative-reward must-testing preorder. To prove this coincidence we characterise the usual resolution-based testing in terms of the weak transitions of processes, without having to involve policies, adversaries, schedulers, resolutions or similar structures that are external to the process under investigation. This requires establishing the continuity of our function for calculating testing outcomes.

1 Introduction

Extending classical testing semantics [1, 9] to a setting in which probability and nondeterminism co-exist was initiated in [18]. The application of a test to a process yields a set of probabilities for reaching a success state. Traditionally, this set of result probabilities is obtained by *resolving* [7] a system into a non-empty set of deterministic but probabilistic systems, each representing a possible probabilistic run of the original system; concepts such as *policy* [14], *adversary* [15], *scheduler* [16] and *resolution* [7] have been used for this purpose. *Reward testing* was introduced in [10] for concurrency, though earlier pioneered in [11] for sequential programs; here the success states are labelled by nonnegative real numbers—*rewards*—to indicate degrees of success, and reaching a success state accumulates the associated reward. In [17] an infinite set of success actions is used to report success, and the testing outcomes are vectors of probabilities of performing these success actions. Compared to [10] this amounts to distinguishing different qualities of success, rather than different quantities.

In [18] and [17], both tests and testees are nondeterministic probabilistic processes, whereas [10] allows nonprobabilistic tests only, thereby obtaining a less discriminating form of testing. In [7] we strengthened reward testing by also allowing probabilistic tests. Taking reward testing in this form we showed that for finitary processes, i.e. finite-state and finitely branching processes, all three modes of testing lead to the same testing preorders. Thus, vector-based testing is no more powerful than *scalar* testing that employs only one success action, and likewise reward testing is no more powerful than the special case of reward testing in which all rewards are 1. [1]

*Deng was partially supported by the National Natural Science Foundation of China (61173033, 61261130589, 61033002).

†Hennessy was supported by SFI project SFI 06 IN.1 1898.

‡Morgan acknowledges the support of ARC Discovery Grant DP0879529.

§NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

[1] In spite of this there *is* a difference in power between the notions of testing from [18] and [17], but this is an issue that is

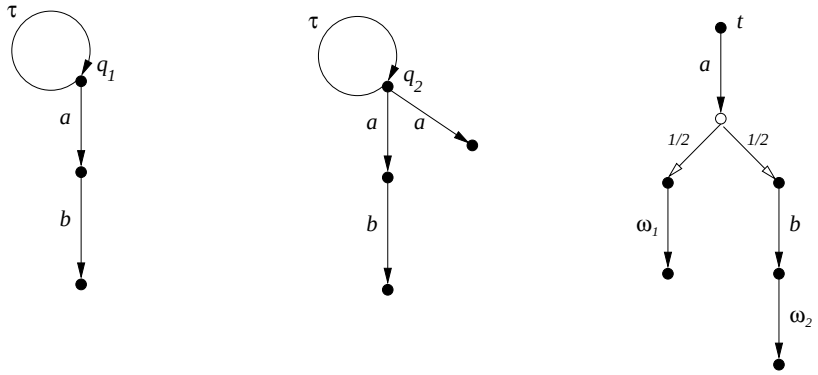


Figure 1: Two processes with divergence and a test

In certain situations it is natural to introduce negative rewards; this is the case, for instance, in the theory of Markov Decision Processes [14]. Intuitively, we could understand negative rewards as costs, while positive rewards are often viewed as benefits or profits. Consider for instance the (nonprobabilistic) processes q_1 and q_2 of Figure 1. Here a represents the action of making an investment. Assuming that the investment is made by bidding for some commodity, the τ-action represents an unsuccessful bid — if this happens one simply tries again. Now b represents the action of reaping the benefits of this investment. Wheres q_1 models a process in which making the investment is always followed by an opportunity to reap the benefits, the process q_2 allows, nondeterministically, for the possibility that the investment is unsuccessful, so that a does not always lead to a state where b is enabled. The test t, which will be explained later, allows us to give a negative reward to action a—its cost—and a positive reward to b.

This leads to the question: *if both negative- and positive rewards are allowed, how would the original reward-testing semantics change?*[2] We refer to the more relaxed form of testing, using positive and negative rewards, as *real-reward testing* and the original one (from [10], but with probabilistic tests as in [7]) as *nonnegative-reward testing*.

The power of real-reward testing is illustrated in Figure 1. The two (nonprobabilistic) processes in the left- and central diagrams are equivalent under (probabilistic) may- as well as must testing; the τ-loops in the initial states cause both processes to fail any nontrivial must test. Yet, if a reward of -1 is associated with performing the action a, and a reward of 2 with the subsequent performance of b, it turns out that in the first process the net reward is either 0, if the process remains stuck in its initial state, or positive, whereas running the second process may yield a loss. See Example 3.8 for details of how these rewards are assigned, and how net rewards are associated with the application of tests such as t. This example shows that for processes that may exhibit divergence, real-reward testing is more discriminating than nonnegative-reward testing, or other forms of probabilistic testing. It also illustrates that the extra power is relevant in applications.

As remarked, in [7] we established that for finitary processes the nonnegative-reward must-testing preorder ($\sqsubseteq_{\mathbf{nr}\text{must}}$) coincides with the probabilistic must-testing preorder ($\sqsubseteq_{\text{pmust}}$), and likewise for the

entirely orthogonal to the distinction between scalar testing, reward testing and vector-based testing. In [17] it is the execution of a success *action* that constitutes success, whereas in [1, 9, 18, 10] it is reaching a success *state* (even though typically success actions are used to identify those states). In [2, Ex 5.3] we showed that state-based testing is (slightly) more powerful than action-based testing. The results presented in [7] about the coincidence of scalar, reward, and vector-based testing preorders pertain to action-based version of each, but in the conclusion it is observed that the same coincidence could be obtained for their state-based versions. In the current paper we stick to state-based testing.

[2] One might suspect no change at all, for any assignment of rewards from the interval $[-1,+1]$ can be converted into a non-negative assignment simply by adding 1 to all of them. But that would not preserve the testing order in the case of zero-outcomes that resulted from a process's failing to reach any success state at all: those zeroes would remain zero.

$$(\sqsubseteq_{\mathbf{rr}\text{may}})^{-1} \overset{\text{Thm. } 3.7}{=} \sqsubseteq_{\mathbf{rr}\text{must}} \overset{\text{Thm. } 6.4}{=} \sqsubseteq_{\mathbf{nr}\text{must}} \overset{[7]}{=} \sqsubseteq_{\text{pmust}} \overset{[3]}{=} \sqsubseteq_{FS}$$

The symbol $=$ between two relations means that they coincide for finitary convergent processes.

Figure 2: The relationship of different testing preorders.

may preorders. The main result of this paper is that restricted to finitary convergent processes, the real-reward must preorder $\sqsubseteq_{\mathbf{rr}\text{must}}$ coincides with the nonnegative-reward must preorder, i.e. for any finitary convergent processes,

$$\Delta \sqsubseteq_{\mathbf{rr}\text{must}} \Gamma \quad \text{iff} \quad \Delta \sqsubseteq_{\mathbf{nr}\text{must}} \Gamma. \tag{1}$$

Here, as we shall see, convergence is the natural generalisation of the standard concept for nonprobabilistic processes to the probabilistic setting; in particular it rules out the processes of Figure 1.

There is also a surprisingly simple proof of the fact that for real-reward testing the may- and must preorders are the inverse of each other, i.e. that for any processes Δ and Γ,

$$\Delta \sqsubseteq_{\mathbf{rr}\text{may}} \Gamma \quad \text{iff} \quad \Gamma \sqsubseteq_{\mathbf{rr}\text{must}} \Delta. \tag{2}$$

This pleasing symmetry does not hold for the more restrictive nonnegative-reward (or scalar) testing. Moreover, the analogy of (1) for the may preorder does not hold, i.e. $\sqsubseteq_{\mathbf{rr}\text{may}}$ does not coincide with $\sqsubseteq_{\mathbf{nr}\text{may}}$ (q.v. the end of Section 8).

Although it is easy to see that in (1) the former implies the latter, to prove the opposite is far from trivial; see more discussion in Section 7. We employ a characterisation of $\sqsubseteq_{\text{pmust}}$ from [2, 3]. *Failure simulation* is a well-known behavioural preorder for nondeterministic processes [8]; in [2] we showed that it could be adapted to characterise the probabilistic must-testing preorder $\sqsubseteq_{\text{pmust}}$, and in [3] this work was generalised from finite to finitary processes. This involved the generalisation of the standard notion of (weak) derivations in state-based systems [13], to probabilistic processes, i.e. probability distributions. By capitalising on this novel notion of derivation between distributions we can show that the failure simulation preorder $\sqsubseteq_{FS}$ is contained in $\sqsubseteq_{\mathbf{rr}\text{must}}$. Convergence is essential here, even though it is not needed to establish that $\sqsubseteq_{FS}$ is contained in $\sqsubseteq_{\mathbf{nr}\text{must}}$. Recall that $\sqsubseteq_{\mathbf{rr}\text{must}}$ is defined using *resolutions*; the key to proving this containment, the heart of the paper, is showing that certain derivations, which we call *extreme derivations*, are essentially the same as *resolutions*. Combining this with the results from [7] and [3] mentioned above leads to our required result that $\sqsubseteq_{\mathbf{nr}\text{must}}$ is included in $\sqsubseteq_{\mathbf{rr}\text{must}}$, as far as finitary convergent processes are concerned. Consequently, in this case, all the relations of Figure 2 collapse into one.

The rest of this paper is organised as follows. We start by recalling notation for probabilistic labelled transition systems. In Section 3 we review the resolution-based testing approach and show that the real-reward may preorder is simply the inverse of the real-reward must preorder. Moreover, using the example of Figure 1, we show that in the presence of divergence the inclusion of $\sqsubseteq_{\mathbf{rr}\text{must}}$ in $\sqsubseteq_{\mathbf{nr}\text{must}}$ is proper. In Section 4 we recall the notions of derivation and the failure simulation preorder. In Section 5 we show that resolutions can be seen as certain kinds of derivations. Then in Section 6 we show for finitary convergent processes that real-reward must testing coincides with nonnegative-reward must testing. We explain in Section 7 why the proof of the coincidence result cannot easily be simplified, and then conclude in Section 8.

Besides the related work already mentioned above, many other studies on probabilistic testing and simulation semantics have appeared in the literature. They are reviewed in [6, 2]. An extended abstract of the current work has appeared as [5]. All the proofs omitted there are now detailed. Section 7 is newly added to explain the subtle difference between $\sqsubseteq_{\mathbf{rr}\text{must}}$ and $\sqsubseteq_{\mathbf{nr}\text{must}}$.

2 Probabilistic Processes

A (discrete) probability *subdistribution* over a set S is a function $\Delta : S \to [0,1]$ with $\sum_{s\in S}\Delta(s) \leq 1$; the *support* of such a Δ is $\lceil\Delta\rceil := \{s \in S \mid \Delta(s) > 0\}$, and its *mass* $|\Delta|$ is $\sum_{s\in\lceil\Delta\rceil}\Delta(s)$. A subdistribution is a (total, or full) *distribution* if $|\Delta| = 1$. The point distribution $\overline{s}$ assigns probability 1 to s and 0 to all other elements of S, so that $\lceil\overline{s}\rceil = \{s\}$. With $\mathscr{D}_{sub}(S)$ we denote the set of subdistributions over S, and with $\mathscr{D}(S)$ its subset of full distributions.

Let $\{\Delta_k \mid k \in K\}$ be a set of subdistributions, possibly infinite. Then $\sum_{k\in K}\Delta_k$ is the real-valued function in $S \to \mathbb{R}$ defined by $(\sum_{k\in K}\Delta_k)(s) := \sum_{k\in K}\Delta_k(s)$. This is a partial operation on subdistributions because for some state s the sum of $\Delta_k(s)$ might exceed 1. If the index set is finite, say $\{1..n\}$, we often write $\Delta_1 + \ldots + \Delta_n$. For p a real number from $[0,1]$ we use $p\cdot\Delta$ to denote the subdistribution given by $(p\cdot\Delta)(s) := p\cdot\Delta(s)$. Finally we use ε to denote the everywhere-zero subdistribution that thus has empty support. These operations on subdistributions do not readily adapt themselves to distributions; yet if $\sum_{k\in K} p_k = 1$ for some $p_k \geq 0$, and the Δ_k are distributions, then so is $\sum_{k\in K} p_k\cdot\Delta_k$.

The expected value $\sum_{s\in S}\Delta(s)\cdot f(s)$ over a subdistribution Δ of a bounded nonnegative function f to the reals or tuples of them is written $\mathrm{Exp}_\Delta(f)$, and the image of a subdistribution Δ through a function $f : S \to T$, for some set T, is written $\mathrm{Img}_f(\Delta)$ — the latter is the subdistribution over T given by $\mathrm{Img}_f(\Delta)(t) := \sum_{f(s)=t}\Delta(s)$ for each $t \in T$.

Definition 2.1 A *probabilistic labelled transition system* (pLTS) is a triple $\langle S, \mathsf{Act}, \to\rangle$, where
(i) S is a set of states,
(ii) Act is a set of visible actions,
(iii) relation $\to$ is a subset of $S \times \mathsf{Act}_\tau \times \mathscr{D}(S)$.
Here Act_τ denotes $\mathsf{Act}\cup\{\tau\}$, where $\tau \notin \mathsf{Act}$ is the invisible- or internal action.

A (nonprobabilistic) labelled transition system (LTS) may be viewed as a degenerate pLTS — one in which only point distributions are used. As with LTSs, we write $s \xrightarrow{\alpha} \Delta$ for $(s,\alpha,\Delta) \in \to$, as well as $s \xrightarrow{\alpha}$ for $\exists\Delta : s \xrightarrow{\alpha} \Delta$ and $s \to$ for $\exists\alpha : s \xrightarrow{\alpha}$, with $s \not\xrightarrow{\alpha}$ and $s \not\to$ representing their negations.

We graphically depict pLTSs as follows. States are represented by nodes of the form $\bullet$ and distributions by nodes of the form $\circ$. For any state s and distribution Δ with $s \xrightarrow{\alpha} \Delta$ we draw an edge from s to Δ, labelled with α. For any distribution Δ and state s in $\lceil\Delta\rceil$, the support of Δ, we draw an edge from Δ to s, labelled with $\Delta(s)$. We leave out point-distributions, diverting an incoming edge to the unique state in its support. See e.g. Figure 4 in the next section for some example pLTSs.

In this paper a *(probabilistic) process* will simply be a distribution over the state set of a pLTS. A pLTS is *deterministic* if for any state s and label α there is at most one distribution Δ with $s \xrightarrow{\alpha} \Delta$. It is *finitely branching* if the set $\{\Delta \mid s \xrightarrow{\alpha} \Delta,\ \alpha \in L\}$ is finite for all states s; if moreover S is finite, then the pLTS is *finitary*. A subdistribution Δ over the state set S of an arbitrary pLTS is *finitary* if restricting S to the states reachable from Δ in the graphical representation of the pLTS yields a finitary sub-pLTS. Similarly, a subdistribution Δ is *finite* if restricting S to the states reachable from Δ yields a finitary sub-pLTS without loops.

3 Testing probabilistic processes

A *test* is a finite distribution over the state set of a pLTS having $\mathsf{Act}_\tau \cup \Omega$ as its set of transition labels, where Ω is a set of fresh *success* actions, not already in Act_τ, introduced specifically to report testing outcomes.[3] For simplicity we may assume a fixed pLTS of processes—our results apply to any choice

[3] For *vector-based* testing we normally take Ω to be countably infinite [17]. This way we have an unbounded supply of success actions for building tests, of course without obligation to use them all. *Scalar* testing is obtained by taking $|\Omega| = 1$.

$$\frac{t \xrightarrow{\alpha}_{\mathbf{T}} \Theta \qquad \alpha \notin \mathsf{Act}}{t\|p \xrightarrow{\alpha} \Theta\|\overline{p}} \qquad \frac{p \xrightarrow{\tau}_{\mathbf{P}} \Delta}{t\|p \xrightarrow{\tau} \overline{t}\|\Delta} \qquad \frac{t \xrightarrow{a}_{\mathbf{T}} \Theta \quad p \xrightarrow{a}_{\mathbf{P}} \Delta \qquad a \in \mathsf{Act}}{t\|p \xrightarrow{\tau} \Theta\|\Delta}$$

Figure 3: Synchronous parallel composition between tests and processes

of such a pLTS—and a fixed pLTS of tests. Since the power of testing depends on the expressivity of the pLTS of tests—in particular certain types of tests are necessary for our results—let us just postulate that this pLTS is sufficiently expressive for our purposes — for example that it can be used to interpret all processes from the language pCSP, as in our previous papers [6, 2, 3]. [4]

Although we use success *actions*, they are used merely to mark certain states as success states, namely the sources of transitions labelled by success actions. For this reason we systematically ignore the distributions that can be reached after a success action. We impose two requirements on all states in a pLTS of tests, namely

(A) if $t \xrightarrow{\omega_1}$ and $t \xrightarrow{\omega_2}$ with $\omega_1, \omega_2 \in \Omega$ then $\omega_1 = \omega_2$. *uniqueness*

(B) if $t \xrightarrow{\omega}$ with $\omega \in \Omega$ and $t \xrightarrow{\alpha} \Delta$ with $\alpha \in \mathsf{Act}_\tau$ then $u \xrightarrow{\omega}$ for all $u \in \lceil \Delta \rceil$. *no ω-disabling*

The first condition says that a success state can have one success identity only, whereas the second condition is a slight weakening of the requirement from [10] that success states must be end states; it allows further progress from an ω-success state, for some $\omega \in \Omega$, but ω must remain enabled. [5]

To apply test Θ to process Δ we form a parallel composition $\Theta\|\Delta$ in which *all* visible actions of Δ must synchronise with Θ. Those synchronisations are immediately renamed into τ so that the resulting composition is a process whose only possible actions are the elements of $\Omega_\tau := \Omega \cup \{\tau\}$. Formally, if $\langle \mathbf{P}, \mathsf{Act}, \rightarrow_{\mathbf{P}} \rangle$ and $\langle \mathbf{T}, \mathsf{Act} \cup \Omega, \rightarrow_{\mathbf{T}} \rangle$ are the pLTSs of processes and tests, then the pLTS of applications of tests to processes is $\langle \mathbf{C}, \Omega, \rightarrow \rangle$, with $\mathbf{C} = \{t\|p \mid t \in \mathbf{T} \wedge p \in \mathbf{P}\}$ and $\rightarrow$ the transition relation generated by the rules in Fig. 3. Here if $\Theta \in \mathscr{D}(\mathbf{T})$ and $\Delta \in \mathscr{D}(\mathbf{P})$, then $\Theta\|\Delta$ is the distribution given by $(\Theta\|\Delta)(t\|p) := \Theta(t) \cdot \Delta(p)$. The resulting pLTS also satisfies (A), (B) above; this would not be the case if we had strengthened (B) to require that success states must be end states.

We will define the result $\mathscr{A}(\Theta, \Delta)$ of applying the test Θ to the process Δ to be a set of testing outcomes, exactly one of which results from each resolution of the choices in $\Theta\|\Delta$. Each *testing outcome* is an Ω-tuple of real numbers in the interval [0,1], i.e. a function $o : \Omega \to [0,1]$, and its ω-component $o(\omega)$, for $\omega \in \Omega$, gives the probability that the resolution in question will reach an *ω-success state*, one in which the success action ω is possible.

Due to the presence of nondeterminism in pLTSs, we need a mechanism to reduce a nondeterministic structure into a set of deterministic structures, each of which determines a single possible outcome. Here we adapt the notion of *resolution*, defined in [7] for probabilistic automata, to pLTSs.

Definition 3.1 [Resolution] A *resolution* of a subdistribution $\Phi \in \mathscr{D}_{sub}(S)$ in a pLTS $\langle S, \Omega, \rightarrow \rangle$ is a triple $\langle R, \Lambda, \rightarrow_R \rangle$ where $\langle R, \Omega, \rightarrow_R \rangle$ is a deterministic pLTS and $\Lambda \in \mathscr{D}_{sub}(R)$, such that there exists a *resolving function* $f : R \to S$ satisfying

(i) $\mathrm{Img}_f(\Lambda) = \Phi$

(ii) if $r \xrightarrow{\alpha}_R \Lambda'$ for $\alpha \in \Omega_\tau$ then $f(r) \xrightarrow{\alpha} \mathrm{Img}_f(\Lambda')$

(iii) if $f(r) \xrightarrow{\alpha}$ for $\alpha \in \Omega_\tau$ then $r \xrightarrow{\alpha}_R$.

[4] In [3] tests are allowed to be finitary, but if two processes are behaviourally different they can be distinguished by some characteristic tests which are always finite. Therefore, the results in [3] still hold if tests are required to be finite, as we do here.

[5] This simplifies our treatment of test but, as can be seen from Appendix A of [7], it is not a heavy restriction.

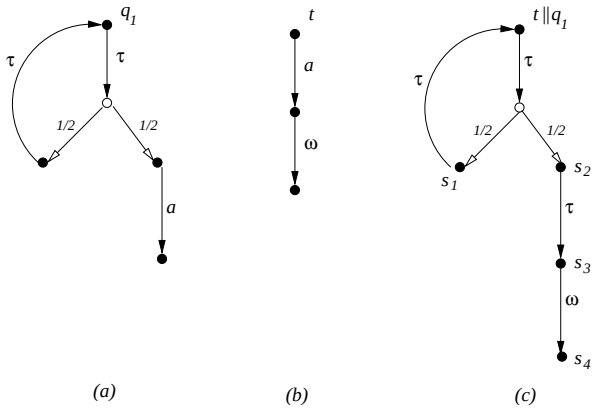


Figure 4: Testing the process $\overline{q_1}$

The reader is referred to Section 2 of [7] for a detailed discussion of the concept of resolution, and the manner in which a resolution represents a run of a process; in particular in a resolution states in S are allowed to be resolved into distributions, and computation steps can be *probabilistically interpolated*. Our resolutions match the results of applying a scheduler as defined in [16].

We now explain how to associate an outcome with a particular resolution, which in turn will associate a set of outcomes with a subdistribution in a pLTS. Given a deterministic pLTS $\langle R, \Omega, \rightarrow_R \rangle$ consider the functional $\mathscr{F} : (R \to [0,1]^\Omega) \to (R \to [0,1]^\Omega)$ defined by

$$\mathscr{F}(g)(r)(\omega) := \begin{cases} 1 & \text{if } r \xrightarrow{\omega} \\ 0 & \text{if } r \not\xrightarrow{\omega} \text{ and } r \not\xrightarrow{\tau} \\ \mathrm{Exp}_\Delta(g)(\omega) & \text{if } r \not\xrightarrow{\omega} \text{ and } r \xrightarrow{\tau} \Delta. \end{cases} \tag{3}$$

We view the unit interval $[0,1]$ ordered in the standard manner as a complete lattice; this induces the structure of a complete lattice on the product $[0,1]^\Omega$ and in turn on the set of functions $R \to [0,1]^\Omega$. The functional $\mathscr{F}$ is easily seen to be monotonic and therefore has a least fixed point, which we denote by $\mathbb{V}_{\langle R,\Omega,\rightarrow_R\rangle}$; this is abbreviated to $\mathbb{V}$ when the deterministic pLTS in question is understood. Intuitively $\mathrm{Exp}_\Lambda(\mathbb{V}_{\langle R,\Omega,\rightarrow_R\rangle})$ is the result of executing the resolution $\langle R, \Lambda, \rightarrow_R \rangle$ starting from the initial distribution Λ, a vector of probabilities. From Definition 3.1 we see that in general a distribution Φ gives rise to a non-empty set of resolutions. Collecting all of the possible results of executing them we get

$$\mathscr{A}(\Phi) = \{\, \mathrm{Exp}_\Lambda(\mathbb{V}_{\langle R,\Omega,\rightarrow_R\rangle}) \mid \langle R, \Lambda, \rightarrow_R \rangle \text{ is a resolution of } \Phi \,\}. \tag{4}$$

This notation is most often used in calculating the results of applying a test to a process. To emphasise this, we will sometimes use the notation $\mathscr{A}(\Theta, \Delta)$ for $\mathscr{A}(\Theta \| \Delta)$.

Example 3.2 Consider the process $\overline{q_1}$ depicted in Figure 4(a). When we apply the test $\overline{t}$ depicted in Figure 4(b) to it we get the process $\overline{t \| q_1}$ depicted in Figure 4(c). This process is already deterministic, hence has essentially only one resolution: itself. Moreover the outcome $\mathrm{Exp}_{\overline{t\|q_1}}(\mathbb{V}) = \mathbb{V}(t \| q_1)$ associated with it is the least solution of the equation $\mathbb{V}(t\|q_1) = \frac{1}{2} \cdot \mathbb{V}(t\|q_1) + \frac{1}{2}\overrightarrow{\omega}$ where $\overrightarrow{\omega} : \Omega \to [0,1]$ is the Ω-tuple with $\overrightarrow{\omega}(\omega) = 1$ and $\overrightarrow{\omega}(\omega') = 0$ for all $\omega' \neq \omega$. In fact this equation has a unique solution in $[0,1]^\Omega$, namely $\overrightarrow{\omega}$. Thus $\mathscr{A}(\overline{t}, \overline{q_1}) = \{\overrightarrow{\omega}\}$. □

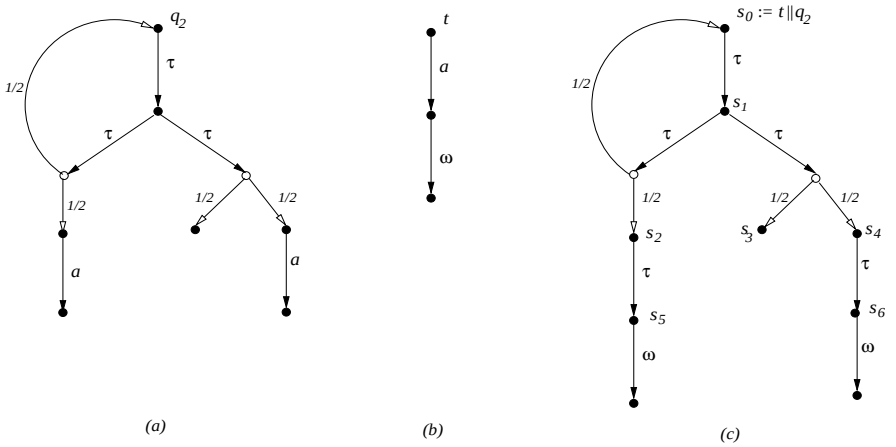


Figure 5: Testing the process $\overline{q_2}$

Example 3.3 Consider the process $\overline{q_2}$ and the application of the test $\overline{t}$ to it, as outlined in Figure 5. For each $k \geq 1$ the process $\overline{t \| q_2}$ has a resolution $\langle R_k, \Lambda, \rightarrow_{R_k} \rangle$ such that $\mathrm{Exp}_\Lambda(\mathbb{V}) = (1-\frac{1}{2^k})\overrightarrow{\omega}$; intuitively it goes around the loop $(k-1)$ times before at last taking the right hand τ action. Thus $\mathscr{A}(\overline{t}, \overline{q_2})$ contains $(1-\frac{1}{2^k})\overrightarrow{\omega}$ for every $k \geq 1$. But it also contains $\overrightarrow{\omega}$, because of the resolution which takes the left hand τ-move every time. Thus $\mathscr{A}(\overline{t}, \overline{q_2})$ includes the set

$$\{(1-\tfrac{1}{2})\overrightarrow{\omega},\ (1-\tfrac{1}{2^2})\overrightarrow{\omega}, \ldots, (1-\tfrac{1}{2^k})\overrightarrow{\omega}, \ldots, \overrightarrow{\omega}\}$$

As resolutions allow any interpolation between the two τ-transitions from state s_1, $\mathscr{A}(\overline{t}, \overline{q_2})$ is actually the convex closure of the above set. □

There are two standard methods for comparing two sets of ordered outcomes:

$O_1 \leq_{\mathrm{Ho}} O_2$ if for every $o_1 \in O_1$ there exists some $o_2 \in O_2$ such that $o_1 \leq o_2$
$O_1 \leq_{\mathrm{Sm}} O_2$ if for every $o_2 \in O_2$ there exists some $o_1 \in O_1$ such that $o_1 \leq o_2$

This gives us our definition of the probabilistic may- and must-testing preorders; they are decorated with $\cdot^\Omega$ for the repertoire Ω of testing actions they employ.

Definition 3.4 [Probabilistic testing preorders]

(i) $\Delta \sqsubseteq^\Omega_{\mathrm{pmay}} \Gamma$ if for every Ω-test Θ, $\mathscr{A}(\Theta, \Delta) \leq_{\mathrm{Ho}} \mathscr{A}(\Theta, \Gamma)$.

(ii) $\Delta \sqsubseteq^\Omega_{\mathrm{pmust}} \Gamma$ if for every Ω-test Θ, $\mathscr{A}(\Theta, \Delta) \leq_{\mathrm{Sm}} \mathscr{A}(\Theta, \Gamma)$.

These preorders are abbreviated to $\Delta \sqsubseteq_{\mathrm{pmay}} \Gamma$ and $\Delta \sqsubseteq_{\mathrm{pmust}} \Gamma$ when $|\Omega| = 1$.

In [7] we established that for finitary processes $\sqsubseteq^\Omega_{\mathrm{pmay}}$ coincides with $\sqsubseteq_{\mathrm{pmay}}$ and $\sqsubseteq^\Omega_{\mathrm{pmust}}$ with $\sqsubseteq_{\mathrm{pmust}}$ for any choice of Ω. We also defined the reward-testing preorders in terms of the mechanism set up so far. The idea is to associate with each success action $\omega \in \Omega$ a reward, which is a nonnegative number in the unit interval $[0,1]$; and then a run of a probabilistic process in parallel with a test yields an expected reward accumulated by those states which can enable success actions. A reward tuple $h \in [0,1]^\Omega$ is used to assign reward $h(\omega)$ to success action ω, for each $\omega \in \Omega$. Due to the presence of nondeterminism, the application of a test Θ to a process Δ produces a set of expected rewards. Two sets of rewards

can be compared by examining their suprema/infima; this gives us two methods of testing called reward may/must testing. In [7] all rewards are required to be nonnegative, so we refer to that approach of testing as *nonnegative-reward testing*. If we also allow negative rewards, which intuitively can be understood as costs, then we obtain an approach of testing called *real-reward testing*. Technically, we simply let reward tuples h range over the set $[-1,1]^\Omega$. If $o \in [0,1]^\Omega$, we use the dot-product $h \cdot o = \sum_{\omega\in\Omega} h(\omega)\cdot o(\omega)$. It can apply to a set $O \subseteq [0,1]^\Omega$ so that $h \cdot O = \{h \cdot o \mid o \in O\}$. Let $A \subseteq [-1,1]$. We use the notation $\bigsqcup A$ for the supremum of set A, and $\bigsqcap A$ for the infimum.

Definition 3.5 [Reward testing preorders]

(i) $\Delta \sqsubseteq^\Omega_{\mathbf{nr}\text{may}} \Gamma$ if for every Ω-test Θ and nonnegative-reward tuple $h \in [0,1]^\Omega$, $\bigsqcup h \cdot \mathscr{A}(\Theta,\Delta) \le \bigsqcup h \cdot \mathscr{A}(\Theta,\Gamma)$.

(ii) $\Delta \sqsubseteq^\Omega_{\mathbf{nr}\text{must}} \Gamma$ if for every Ω-test Θ and nonnegative-reward tuple $h \in [0,1]^\Omega$, $\bigsqcap h \cdot \mathscr{A}(\Theta,\Delta) \le \bigsqcap h \cdot \mathscr{A}(\Theta,\Gamma)$.

(iii) $\Delta \sqsubseteq^\Omega_{\mathbf{rr}\text{may}} \Gamma$ if for every Ω-test Θ and real-reward tuple $h \in [-1,1]^\Omega$, $\bigsqcup h \cdot \mathscr{A}(\Theta,\Delta) \le \bigsqcup h \cdot \mathscr{A}(\Theta,\Gamma)$.

(iv) $\Delta \sqsubseteq^\Omega_{\mathbf{rr}\text{must}} \Gamma$ if for every Ω-test Θ and real-reward tuple $h \in [-1,1]^\Omega$, $\bigsqcap h \cdot \mathscr{A}(\Theta,\Delta) \le \bigsqcap h \cdot \mathscr{A}(\Theta,\Gamma)$.

This time we drop the superscript Ω iff Ω is countably infinite.

It is shown in Corollary 1 of [7] that nonnegative-reward testing is equally powerful as probabilistic testing.

Theorem 3.6 [7] For any finitary processes Δ and Γ,

(i) $\Delta \sqsubseteq_{\mathbf{nr}\text{may}} \Gamma$ if and only if $\Delta \sqsubseteq_{\text{pmay}} \Gamma$.

(ii) $\Delta \sqsubseteq_{\mathbf{nr}\text{must}} \Gamma$ if and only if $\Delta \sqsubseteq_{\text{pmust}} \Gamma$.

In this paper we focus on the real-reward testing preorders $\sqsubseteq_{\mathbf{rr}\text{may}}$ and $\sqsubseteq_{\mathbf{rr}\text{must}}$, by comparing them with the nonnegative reward testing preorders $\sqsubseteq_{\mathbf{nr}\text{may}}$ and $\sqsubseteq_{\mathbf{nr}\text{must}}$. Although these two nonnegative-reward testing preorders are in general incomparable, we have for the real-reward testing preorders:

Theorem 3.7 For any processes Δ and Γ, it holds that $\Delta \sqsubseteq_{\mathbf{rr}\text{may}} \Gamma$ if and only if $\Gamma \sqsubseteq_{\mathbf{rr}\text{must}} \Delta$.

Proof: We first notice that for any nonempty set $A \subseteq [0,1]^\Omega$ and any reward tuple $h \in [-1,1]^\Omega$,

$$\bigsqcup h \cdot A \quad = \quad -(\bigsqcap (-h)\cdot A) \tag{5}$$

where $-h$ is the negation of h, i.e. $(-h)(\omega) = -(h(\omega))$ for any $\omega \in \Omega$. We consider the "if" direction; the "only if" direction is similar. Let Θ be any Ω-test and h be any real reward tuple in $[-1,1]^\Omega$. Clearly, $-h$ is also a real reward tuple. Suppose $\Gamma \sqsubseteq_{\mathbf{rr}\text{must}} \Delta$, then

$$\bigsqcap (-h)\cdot \mathscr{A}(\Theta,\Gamma) \quad \le \quad \bigsqcap (-h)\cdot \mathscr{A}(\Theta,\Delta) \tag{6}$$

Therefore, we can infer that

$$\begin{array}{lcll} \bigsqcup h \cdot \mathscr{A}(\Theta,\Delta) & = & -(\bigsqcap (-h)\cdot \mathscr{A}(\Theta,\Delta)) & \text{by (5)} \\ & \le & -(\bigsqcap (-h)\cdot \mathscr{A}(\Theta,\Gamma)) & \text{by (6)} \\ & = & \bigsqcup h \cdot \mathscr{A}(\Theta,\Gamma) & \text{by (5).} \end{array}$$

□

Our next task is to compare $\sqsubseteq_{\mathbf{rr}\text{must}}$ with $\sqsubseteq_{\mathbf{nr}\text{must}}$. The former is included in the latter, which directly follows from Definition 3.5. Surprisingly, it turns out that for finitary convergent processes the latter is also included in the former, thus establishing that the two preorders are in fact the same. The rest of the paper is devoted to proving this result. However, we first show that this result does not extend to divergent processes.

Example 3.8 Consider the processes $\overline{q_1}$ and $\overline{q_2}$ depicted in Figure 1. Using the characterisations of $\sqsubseteq_{\text{pmay}}$ and $\sqsubseteq_{\text{pmust}}$ in [3], it is easy to see that these processes cannot be distinguished by probabilistic may- and must testing, and hence not by nonnegative-reward testing either. However, let $\overline{t}$ be the test in the right diagram of Figure 1 that first synchronises on the action a, and then with probability $\frac{1}{2}$ reaches a state in which a reward of -2 is allocated, and with the remaining probability $\frac{1}{2}$ synchronises with the action b and reaches a state that yields a reward of 4. Thus the test employs two success actions ω_1 and ω_2, and we use the reward tuple h with $h(\omega_1) = -2$ and $h(\omega_2) = 4$. Then the resolution of $\overline{q_1}$ that does not involve the τ-loop contributes the value $-2 \cdot \frac{1}{2} + 4 \cdot \frac{1}{2} = -1 + 2 = 1$ to the set $h \cdot \mathscr{A}(\overline{t}, \overline{q_1})$, whereas the resolution that only involves the τ-loop contributes the value 0. Due to interpolation, $h \cdot \mathscr{A}(\overline{t}, \overline{q_1})$ is in fact the entire interval $[0,1]$. On the other hand, the resolution corresponding to the a-branch of q_2 contributes the value -1 and $h \cdot \mathscr{A}(\overline{t}, \overline{q_2}) = [-1,1]$. Thus $\bigsqcap h \cdot \mathscr{A}(\overline{t}, \overline{q_1}) = 0 > -1 = \bigsqcap h \cdot \mathscr{A}(\overline{t}, \overline{q_2})$, and hence $\overline{q_1} \not\sqsubseteq_{\mathbf{rr}\text{must}} \overline{q_2}$. □

4 Failure simulations

In this section we explain the characterisation of probabilistic testing from [2, 3]; it depends on a generalisation of failure simulations [8] to the probabilistic setting. The key ingredient is that of weak derivations for distributions. To deal with infinite (but finitary) processes, we need to employ the weak derivations of [3] rather than those of [2].

In a pLTS actions are performed only by states, in that actions are given by relations from states to distributions. But processes in general correspond to distributions over states, so in order to define what it means for a process to perform an action, we need to *lift* these relations so that they also apply to distributions. In fact we will find it convenient to lift them to subdistributions.

Definition 4.1 Let $(S, L, \rightarrow)$ be a pLTS and $\mathscr{R} \subseteq S \times \mathscr{D}_{sub}(S)$ be a relation from states to subdistributions. Then $\overline{\mathscr{R}} \subseteq \mathscr{D}_{sub}(S) \times \mathscr{D}_{sub}(S)$ is the smallest relation that satisfies:

(i) $s \mathscr{R} \Delta$ implies $\overline{s} \overline{\mathscr{R}} \Delta$, and

(ii) (Linearity) $\Gamma_i \overline{\mathscr{R}} \Delta_i$ for $i \in I$ implies $(\sum_{i\in I} p_i \cdot \Gamma_i) \ \overline{\mathscr{R}} \ (\sum_{i\in I} p_i \cdot \Delta_i)$ for any $p_i \in [0,1]$ $(i \in I)$ with $\sum_{i\in I} p_i \leq 1$, where I is a countable set.

An application of this notion is when the relation is $\xrightarrow{\alpha}$ for $\alpha \in \mathsf{Act}_\tau$; in that case we also write $\xrightarrow{\alpha}$ for $\overline{\xrightarrow{\alpha}}$. Thus, as source of a relation $\xrightarrow{\alpha}$ we now also allow distributions, and even subdistributions. A subtlety of this approach is that for any action α, we have $\varepsilon \xrightarrow{\alpha} \varepsilon$ simply by taking $I = \emptyset$ or $\sum_{i\in I} p_i = 0$ in Definition 4.1. That turns out to make ε especially useful for modelling the "chaotic" aspects of divergence in [3], in particular that in the must-case a divergent process can mimic any other.

Definition 4.1 is very similar to our previous definition in [2], although there it applied only to (full) distributions:

Lemma 4.2 $\Gamma \overline{\mathscr{R}} \Delta$ if and only if

(i) $\Gamma = \sum_{i\in I} p_i \cdot \overline{s_i}$, where I is an index set and $\sum_{i\in I} p_i \leq 1$,

(ii) For each $i \in I$ there is a subdistribution Δ_i such that $s_i \mathscr{R} \Delta_i$,

(iii) $\Delta = \sum_{i\in I} p_i \cdot \Delta_i$.

Proof: Straightforward. □

An important point here is that a single state can be split into several pieces: that is, the decomposition of Γ into $\sum_{i\in I} p_i \cdot \overline{s_i}$ is not unique.

Definition 4.3 [Weak derivation] Suppose we have subdistributions $\Delta, \Delta_k^{\rightarrow}, \Delta_k^{\times}$, for $k \geq 0$, with the following properties:

$$\begin{array}{rcl} \Delta & = & \Delta_0^{\rightarrow} + \Delta_0^{\times} \\ \Delta_0^{\rightarrow} & \xrightarrow{\tau} & \Delta_1^{\rightarrow} + \Delta_1^{\times} \\ & \vdots & \\ \Delta_k^{\rightarrow} & \xrightarrow{\tau} & \Delta_{k+1}^{\rightarrow} + \Delta_{k+1}^{\times} \ . \\ & \vdots & \end{array}$$

Then we call $\Delta' := \sum_{k=0}^{\infty} \Delta_k^{\times}$ a *weak derivative* of Δ, and write $\Delta \Longrightarrow \Delta'$ to mean that Δ can make a *weak derivation* to its derivative Δ'.

There is always at least one weak derivative of any subdistribution (the subdistribution itself) and there can be many.

Proposition 4.4 [Transitivity, linearity and decomposition of weak derivations [4]]

(i) If $\Delta \Longrightarrow \Delta'$ and $\Delta' \Longrightarrow \Delta''$ then $\Delta \Longrightarrow \Delta''$.

Let $p_i \in [0,1]$ for $i \in I$ with $\sum_{i\in I} p_i \leq 1$.

(ii) If $\Delta_i \Longrightarrow \Delta_i'$ for all $i \in I$ then $\sum_{i\in I} p_i \cdot \Delta_i \Longrightarrow \sum_{i\in I} p_i \cdot \Delta_i'$.

(iii) If $\sum_{i\in I} p_i \cdot \Delta_i \Longrightarrow \Delta'$ then $\Delta' = \sum_{i\in I} p_i \cdot \Delta_i'$ for subdistributions Δ_i' such that $\Delta_i \Longrightarrow \Delta_i'$ for all $i \in I$.

We now use these weak derivations to define, in the standard manner of [13], weak action relations between derivations; these, together with the refusal relations $\stackrel{A}{\not\longrightarrow}$ for $A \subseteq \mathsf{Act}$ are the key ingredients in the definition of the failure-simulation preorder.

Definition 4.5 Let Δ and its variants $\Delta', \Delta^{\mathrm{pre}}, \Delta^{\mathrm{post}}$ be subdistributions in a pLTS $\langle S, \mathsf{Act}, \rightarrow \rangle$.

- For $a \in \mathsf{Act}$ write $\Delta \stackrel{a}{\Longrightarrow} \Delta'$ whenever $\Delta \Longrightarrow \Delta^{\mathrm{pre}} \xrightarrow{a} \Delta^{\mathrm{post}} \Longrightarrow \Delta'$, for some Δ^{pre} and Δ^{post}. Extend this to Act_τ by allowing as a special case that $\stackrel{\tau}{\Longrightarrow}$ is simply $\Longrightarrow$, i.e. including identity (rather than requiring at least one $\xrightarrow{\tau}$).
- For $A \subseteq \mathsf{Act}$ and $s \in S$ write $s \stackrel{A}{\not\longrightarrow}$ if $s \stackrel{\alpha}{\not\longrightarrow}$ for every $\alpha \in A \cup \{\tau\}$; write $\Delta \stackrel{A}{\not\longrightarrow}$ if $s \stackrel{A}{\not\longrightarrow}$ for every $s \in \lceil \Delta \rceil$.
- More generally write $\Delta \Longrightarrow \stackrel{A}{\not\longrightarrow}$ if $\Delta \Longrightarrow \Delta^{\mathrm{pre}}$ for some Δ^{pre} such that $\Delta^{\mathrm{pre}} \stackrel{A}{\not\longrightarrow}$.

Definition 4.6 [Failure simulation preorder] Define $\lhd_{FS}$ to be the largest relation in $S \times \mathscr{D}_{sub}(S)$ such that if $s \lhd_{FS} \Delta$ then

(i) whenever $\overline{s} \stackrel{\alpha}{\Longrightarrow} \Gamma'$, for $\alpha \in \mathsf{Act}_\tau$, then there is a $\Delta' \in \mathscr{D}_{sub}(S)$ with $\Delta \stackrel{\alpha}{\Longrightarrow} \Delta'$ and $\Gamma' \, \overline{\lhd_{FS}} \, \Delta'$,

(ii) and whenever $\overline{s} \Longrightarrow \stackrel{A}{\not\longrightarrow}$ then $\Delta \Longrightarrow \stackrel{A}{\not\longrightarrow}$.

Any relation $\mathscr{R} \subseteq S \times \mathscr{D}_{sub}(S)$ that satisfies the two clauses above is called a *failure simulation*. The failure simulation preorder $\sqsubseteq_{FS} \subseteq \mathscr{D}_{sub}(S) \times \mathscr{D}_{sub}(S)$ is defined by letting $\Delta \sqsubseteq_{FS} \Gamma$ whenever there is a $\Delta^\natural$ with $\Delta \Longrightarrow \Delta^\natural$ and $\Gamma \, \overline{\lhd_{FS}} \, \Delta^\natural$.

Note that the simulating process, Δ, occurs at the right of $\lhd_{FS}$, but at the left of $\sqsubseteq_{FS}$. The following lemma will bee needed in Section 6.

Lemma 4.7 If $\Gamma \overline{\lhd_{FS}} \Delta$ and $\Gamma \Longrightarrow \Gamma'$ then there is a matching transition $\Delta \Longrightarrow \Delta'$ such that $\Gamma' \overline{\lhd_{FS}} \Delta'$.

Proof: $\Gamma \overline{\lhd_{FS}} \Delta$ implies by Lemma 4.2 that $\Gamma = \sum_{i\in I} p_i \cdot \overline{s_i}$, $s_i \lhd_{FS} \Delta_i$, $\Delta = \sum_{i\in I} p_i \cdot \Delta_i$.

By Proposition 4.4(iii) there are $\Gamma'_i \in \mathscr{D}_{sub}(S)$ for $i \in I$ with $\overline{s_i} \Longrightarrow \Gamma'_i$ and $\Gamma' = \sum_{p_i\in I} p_i \cdot \Gamma'_i$. For each $i \in I$ we infer from $s_i \lhd_{FS} \Delta_i$ and $\overline{s_i} \Longrightarrow \Gamma'_i$ that there is a $\Delta'_i \in \mathscr{D}_{sub}(S)$ with $\Delta_i \Longrightarrow \Delta'_i$ and $\Gamma'_i \overline{\lhd_{FS}} \Delta'$. Let $\Delta' := \sum_{i\in I} p_i \cdot \Delta'_i$. Then Definition 4.1(2) and Proposition 4.4(ii) yield $\Gamma' \overline{\lhd_{FS}} \Delta'$ and $\Delta \Longrightarrow \Delta'$. □

The failure simulation preorder is preserved under parallel composition with a test and it is sound and complete for probabilistic must testing of finitary processes.

Theorem 4.8 [3] For finitary processes Δ and Γ,

(i) If $\Delta \sqsubseteq_{FS} \Gamma$ then for any Ω-test Θ, $\Theta \| \Delta \sqsubseteq_{FS} \Theta \| \Gamma$.

(ii) $\Delta \sqsubseteq_{FS} \Gamma$ if and only if $\Delta \sqsubseteq_{\text{pmust}} \Gamma$.

5 From derivations to resolutions

In this section we explain how resolutions, on which the definitions of the testing preorders in Definitions 3.4 and 3.5 are based, can be seen as certain kinds of derivations.

Definition 5.1 [Extreme derivatives] A state s in a pLTS is called *stable* if $s \stackrel{\tau}{\not\longrightarrow}$, and a subdistribution Δ is called *stable* if every state in its support is stable. We write $\Delta \Longrightarrow\!\!\!\!\!\rightarrow \Delta'$ whenever $\Delta \Longrightarrow \Delta'$ and Δ' is stable, and call Δ' an *extreme* derivative of Δ.

Referring to Definition 4.3, we see this means that in the extreme derivation of Δ' from Δ at every stage a state must move on if it can, so that every stopping component can contain only states which *must* stop: for $s \in \lceil \Delta_k^{\rightarrow} + \Delta_k^{\times} \rceil$ we have $s \in \lceil \Delta_k^{\times} \rceil$ if *and now also* only if $s \stackrel{\tau}{\not\longrightarrow}$.

Lemma 5.2 [Existence and uniqueness of extreme derivatives]

(i) For every subdistribution Δ there exists some (stable) Δ' such that $\Delta \Longrightarrow\!\!\!\!\!\rightarrow \Delta'$.

(ii) In a deterministic pLTS if $\Delta \Longrightarrow\!\!\!\!\!\rightarrow \Delta'$ and $\Delta \Longrightarrow\!\!\!\!\!\rightarrow \Delta''$ then $\Delta' = \Delta''$.

Proof: We construct a derivation as in Definition 4.3 of a stable Δ' by defining the components $\Delta_k, \Delta_k^{\times}$ and $\Delta_k^{\rightarrow}$ using induction on k. Let us assume that the subdistribution Δ_k has been defined; in the base case $k = 0$ this is simply Δ. The decomposition of this Δ_k into the components $\Delta_k^{\times}$ and $\Delta_k^{\rightarrow}$ is carried out by defining the former to be precisely those states which must stop, i.e. those s for which $s \stackrel{\tau}{\not\longrightarrow}$. Formally $\Delta_k^{\times}$ is determined by:

$$\Delta_k^{\times}(s) = \begin{cases} \Delta_k(s) & \text{if } s \stackrel{\tau}{\not\longrightarrow} \\ 0 & \text{otherwise} \end{cases}$$

Then $\Delta_k^{\rightarrow}$ is given by the *remainder* of Δ_k, namely those states which can perform a τ action:

$$\Delta_k^{\rightarrow}(s) = \begin{cases} \Delta_k(s) & \text{if } s \stackrel{\tau}{\longrightarrow} \\ 0 & \text{otherwise} \end{cases}$$

Note that these definitions divide the support of Δ_k into two disjoints sets, namely the support of $\Delta_k^{\times}$ and the support of $\Delta_k^{\rightarrow}$. Moreover by construction we know that $\Delta_k^{\rightarrow} \stackrel{\tau}{\longrightarrow} \Theta$ for some Θ; we let Δ_{k+1} be an arbitrary such Θ.

This completes our construction of an extreme derivative as in Definition 4.3 and so we have established (i).

For (ii) we observe that in a deterministic pLTS the above choice of Δ_{k+1} is unique, so that the whole derivative construction becomes unique. □

*Sub*distributions are essential in the definition of extreme derivations. Consider a state t that has only one transition, a self τ-loop $t \xrightarrow{\tau} \overline{t}$. Then it diverges and it has a unique extreme derivative ε, the empty subdistribution. More generally, suppose a subdistribution Δ diverges, that is there is an infinite sequence of internal transitions $\Delta \xrightarrow{\tau} \Delta_1 \xrightarrow{\tau} \ldots \Delta_k \xrightarrow{\tau} \ldots$. Then one extreme derivative of Δ is ε, but it may have others.

In the extreme derivative $\Delta \Longrightarrow\!\!\!\!\succ \Delta'$, the subdistribution Δ' may be viewed as a final result of an execution starting in Δ and dynamically resolving nondeterministic choices as the execution proceeds. We can tabulate the outcome of this execution in the following manner:

Definition 5.3 [Outcomes] The outcome $\$\Phi \in [0,1]^{\Omega}$ of a stable subdistribution Φ is given by $\$\Phi(\omega) = \sum\{\Phi(s) \mid s \in \lceil\Phi\rceil,\ s \xrightarrow{\omega}\}$. For any distribution Φ we write $\mathscr{V}(\Phi)$ for the set of possible outcomes $\{\$\Phi' \mid \Phi \Longrightarrow\!\!\!\!\succ \Phi'\}$ via extreme derivatives.

Let $p_i \in [0,1]$ for $i \in I$ with $\sum_{i\in I} p_i \leq 1$, and let Δ_i, Φ_i, for $i \in I$, be subdistributions. We use $\sum_{i\in I} p_i \cdot \mathscr{V}(\Delta_i)$ as shorthand for $\{\sum_{i\in I} p_i \cdot \nu_i \mid \nu_i \in \mathscr{V}(\Delta_i)\}$. By construction, $\$\sum_{i\in I} p_i \cdot \Phi_i = \sum_{i\in I} p_i \cdot \$\Phi_i$. Using this, we establish the linearity of $\mathscr{V}$:

Lemma 5.4 Let $p_i \in [0,1]$ for $i \in I$ with $\sum_{i\in I} p_i \leq 1$. Then $\mathscr{V}(\sum_{i\in I} p_i \cdot \Delta_i) = \sum_{i\in I} p_i \cdot \mathscr{V}(\Delta_i)$.

Proof: Suppose $\nu \in \mathscr{V}(\sum_{i\in I} p_i \cdot \Delta_i)$. Then $\nu = \$\Phi$ for some stable Φ with $\sum_{i\in I} p_i \cdot \Delta_i \Longrightarrow \Phi$. By Proposition 4.4(iii) Φ can be written as $\sum_{i\in I} p_i \cdot \Phi_i$ for subdistributions Φ_i such that $\Delta_i \Longrightarrow \Phi_i$ for all $i \in I$; moreover, the Φ_i must be stable. Hence $\nu_i := \$\Phi_i \in \mathscr{V}(\Delta_i)$ and thus $\nu = \sum_{i\in I} p_i \cdot \nu_i \in \sum_{i\in I} p_i \cdot \mathscr{V}(\Delta_i)$.

Conversely, suppose $\nu \in \sum_{i\in I} p_i \cdot \mathscr{V}(\Delta_i)$, i.e., $\nu = \sum_{i\in I} p_i \cdot \nu_i$ with $\nu_i \in \mathscr{V}(\Delta_i)$. Then for all $i \in I$ there are stable subdistributions Φ_i with $\nu_i := \$\Phi_i$ and $\Delta_i \Longrightarrow \Phi_i$. So $\sum_{i\in I} p_i \cdot \Delta_i \Longrightarrow \sum_{i\in I} p_i \cdot \Phi_i$ by Proposition 4.4(ii). Moreover $\sum_{i\in I} p_i \cdot \Phi_i$ is stable and $\nu = \sum_{i\in I} p_i \cdot \nu_i = \$\sum_{i\in I} p_i \cdot \Phi_i \in \mathscr{V}(\sum_{i\in I} p_i \cdot \Delta_i)$. □

The following two examples illustrate that this manner of calculating outcomes often gives the same result as when resolutions are used.

Example 5.5 (Revisiting Example 3.2.) The pLTS in Figure 4(c) is deterministic and therefore from part (ii) of Lemma 5.2 it follows that $\overline{t \| q_1}$ has a unique extreme derivative Λ. Moreover Λ can be calculated to be $\sum_{k\geq 1} \frac{1}{2^k} \cdot \overline{s_3}$, which simplifies to the distribution $\overline{s_3}$. Therefore, since $\$\overline{s_3} = \overrightarrow{\omega}$, it follows that $\mathscr{V}(\overline{t \| q_1}) = \{\overrightarrow{\omega}\}$. This is exactly the same result as obtained in Example 3.2, using resolutions. □

Example 5.6 (Revisiting Example 3.3.) The application of the test $\overline{t}$ to processes $\overline{q_2}$ is outlined in Figure 5(c). Consider any extreme derivative Δ' from $s_0 = \overline{t} \| \overline{q_2}$. Using the notation of Definition 4.3, it is clear that $\Delta_0^{\times}$ and $\Delta_0^{\rightarrow}$ must be ε and $\overline{s_0}$ respectively. Similarly, $\Delta_1^{\times}$ and $\Delta_1^{\rightarrow}$ must be ε and $\overline{s_1}$ respectively. But s_1 is a nondeterministic state, having two possible transitions:

(i) $s_1 \xrightarrow{\tau} \Lambda_0$ where Λ_0 has support $\{s_0, s_2\}$ and assigns each of them the weight $\frac{1}{2}$

(ii) $s_1 \xrightarrow{\tau} \Lambda_1$ where Λ_1 has the support $\{s_3, s_4\}$, again dividing the mass equally among them.

So there are many possibilities for Δ_2; from Definition 4.3 one sees that in fact Δ_2 can be of the form

$$p \cdot \Lambda_0 + (1-p) \cdot \Lambda_1 \tag{7}$$

for any choice of $p \in [0,1]$.

Let us consider one possibility, an extreme one where p is chosen to be 0; only the transition (ii) above is used. Here $\Delta_2^{\rightarrow}$ is the subdistribution $\frac{1}{2}\overline{s_4}$, and $\Delta_k^{\rightarrow} = \varepsilon$ whenever $k > 2$. A simple calculation shows that in this case the extreme derivative generated is $\Lambda_1^e = \frac{1}{2}\overline{s_3} + \frac{1}{2}\overline{s_6}$ which implies that $\frac{1}{2}\overrightarrow{\omega} \in \mathscr{V}(\overline{t}\|\overline{q_2})$.

Another possibility for Δ_2 is Λ_0, corresponding to $p = 1$ in (7) above. Continuing this derivation leads to Δ_3 being $\frac{1}{2}\cdot\overline{s_1} + \frac{1}{2}\cdot\overline{s_5}$; thus $\Delta_3^{\times} = \frac{1}{2}\cdot\overline{s_5}$ and $\Delta_3^{\rightarrow} = \frac{1}{2}\cdot\overline{s_1}$. Now in the generation of Δ_4 from $\Delta_3^{\rightarrow}$ again we resolve a transition from the nondeterministic state s_1, by choosing some arbitrary $p \in [0,1]$ in (7). Suppose we choose $p = 1$ every time, completely ignoring transition (ii) above. Then the extreme derivative generated is

$$\Lambda_0^e = \sum_{k\geq 1} \frac{1}{2^k}\cdot\overline{s_5}$$

which simplifies to the distribution $\overline{s_5}$. This in turn means that $\overrightarrow{\omega} \in \mathscr{V}(\overline{t}\|\overline{q_2})$.

We have seen two possible derivations of extreme derivatives from $\overline{s_0}$. But there are many others. In general whenever $\Delta_k^{\rightarrow}$ is of the form $q\cdot\overline{s_1}$ we have to resolve the nondeterminism by choosing a $p \in [0,1]$ in (7) above; moreover each such choice is independent. It turns out that every extreme derivative Δ' of $\overline{s_0}$ is of the form $q\cdot\Lambda_0^e + (1-q)\cdot\Lambda_1^e$ for some choice of $q \in [0,1]$, which implies that $\mathscr{V}(\overline{t}\|\overline{q_2})$ is the convex closure of the set $\{\frac{1}{2}\overrightarrow{\omega}, \overrightarrow{\omega}\}$.

Again this is similar to the results obtained using resolutions, in Example 3.3. □

Unfortunately there is not an exact agreement between using resolutions and extreme derivations, as the next example shows.

Example 5.7 Let $\overline{p}$ be a process that first does an a-action, to the point distribution $\overline{q}$, and then diverges, via the τ-loop $q \xrightarrow{\tau} \overline{q}$. Let $\overline{t}$ be the test used in Examples 3.2 and 3.3. It is easy to see that the distribution $\overline{p\|t}$ has a unique resolution, with expected outcome $\overrightarrow{\omega}$; thus $\mathscr{A}(\overline{t},\overline{p}) = \{\overrightarrow{\omega}\}$.

It turns out that $\overline{t}\|\overline{p}$ also has a unique extreme derivative; unfortunately this turns out to be ε. Since $\$\varepsilon = 0$ this means that $\mathscr{V}(\overline{t}\|\overline{p}) = \overrightarrow{0}$; so in this case, which is actually nonprobabilistic, there is a difference between the use of resolutions and extreme derivations. □

To rectify this anomaly, we restrict our attention to a subset of pLTSs.

Definition 5.8 [ω-respecting] A pLTS $\langle S,\Omega,\rightarrow\rangle$ is said to be *ω-respecting* when it satisfies the uniqueness requirement (A) from Page 5, and $s \xrightarrow{\omega}$, for any $\omega \in \Omega$, implies $s \xrightarrow{\tau}\!\!\!\!\!\!/\;$.

It is straightforward to modify the pLTS of applications of tests to processes into one that it is ω-respecting, namely by removing all transitions $s \xrightarrow{\tau} \Delta$ for states s with $s \xrightarrow{\omega}$; we call this *pruning*. We denote the result of pruning the pLTS $\langle S,\Omega,\rightarrow\rangle$ by $\langle S,\Omega,[\rightarrow]\rangle$, and the distribution Φ in this pruned pLTS by $[\Phi]$.

Example 5.9 (Revisiting Example 5.7) Let $\overline{p},\overline{q}$ and $\overline{t}$ be as in Example 5.7. As we have already seen, $\overline{t}\|\overline{p}$ has the unique derivative ε. But by pruning it we obtain a different extreme derivative. If we denote the state reachable from $\overline{t}$ with the outgoing ω-transition, in Figure 5(c), as ω also, then $[\overline{t}\|\overline{p}]$ has the unique extreme derivative $[\omega\|q]$. Since $\$[\omega\|q] = \overrightarrow{\omega}$, we obtain $\mathscr{V}([\overline{t}\|\overline{p}]) = \{\overrightarrow{\omega}\}$; this is exactly the result obtained using resolutions. □

Note that pruning has no effect on Examples 5.5 and 5.6, as the pLTSs concerned are already ω-respecting. It also has no effect on the closure of the failure simulation preorder under parallel composition:

Lemma 5.10 [4] For finitary processes Δ and Γ, if $\Delta \sqsubseteq_{FS} \Gamma$ then for any Ω-test Θ, $[\Theta\|\Delta] \sqsubseteq_{FS} [\Theta\|\Gamma]$.

In the remainder of this section we show that, at least in ω-respecting pLTSs, using resolutions to calculate outcomes, as used in the definition of testing (Definitions 3.4 and 3.5), leads to the same results as using extreme derivations. In the former a set of deterministic structures are associated with a distribution, while in the latter nondeterministic choices are resolved dynamically as the derivation proceeds. We start by showing that resolution-based testing is insensitive to pruning. Let $\mathscr{A}^{\mathrm{p}}(\Phi)$ denote the set of vectors

$$\{\, \mathrm{Exp}_\Lambda(\mathbb{V}_{\langle R,\Omega,\rightarrow_R\rangle}) \mid \langle R,\Lambda,\rightarrow_R\rangle \text{ is a resolution of } [\Phi]\,\}\ .$$

Proposition 5.11 For any distribution Φ in a pLTS $\langle S,\Omega,\rightarrow\rangle$ we have that $\mathscr{A}^{\mathrm{p}}(\Phi) = \mathscr{A}(\Phi)$.

Proof: "$\supseteq$": Let $\langle R,\Lambda,\rightarrow_R\rangle$ be a resolution of Φ. Then, following Definition 3.1, $\langle R,[\Lambda],[\rightarrow_R]\rangle$ is a resolution of $[\Phi]$ and, by (3), $\mathrm{Exp}_{[\Lambda]}(\mathbb{V}_{\langle R,\Omega,[\rightarrow_R]\rangle}) = \mathrm{Exp}_\Lambda(\mathbb{V}_{\langle R,\Omega,\rightarrow_R\rangle})$.

"$\subseteq$": Let $\langle R,\Lambda,\rightarrow_R\rangle$ be a resolution of $[\Phi]$ with resolving function f. We construct a resolution $\langle R',\Lambda,\rightarrow'_R\rangle$ of Φ as a random extension of $\langle R,\Lambda,\rightarrow_R\rangle$. For every pair $(s,\alpha)\in S\times\Omega_\tau$ with $s \xrightarrow{\alpha}$ pick a distribution $\Psi^{(s,\alpha)}\in\mathscr{D}(S)$ such that $s \xrightarrow{\alpha} \Psi^{(s,\alpha)}$. Now define $R' := R \,\dot\cup\, (S\times\mathbb{N})$, where $\dot\cup$ denotes the disjoint union operation, and obtain $\rightarrow'_R$ from $\rightarrow_R$ by adding (A) a transition $(s,k) \xrightarrow{\alpha}{}'_R \Psi^{(s,\alpha)}_{k+1}$ for each $k\in\mathbb{N}$ and each $s\in S$ with $s \xrightarrow{\alpha}$, and (B) a transition $r \xrightarrow{\tau}{}'_R \Psi_0^{(f(r),\tau)}$ for each $r\in R$ with $f(r) \xrightarrow{\tau}$ as well as $f(r) \xrightarrow{\omega}$ for some $\omega\in\Omega$. Here $\Psi^{(s,\alpha)}_{k+1}\in\mathscr{D}(S\times\{k{+}1\})$ is given by $\Psi^{(s,\alpha)}_{k+1}(t,k{+}1) = \Psi^{(s,\alpha)}(t)$ for all $t\in S$. The resolving function f is extended by $f(s,k) := s$. Using Definition 3.1 it follows that $\langle R',\Lambda,\rightarrow'_R\rangle$ is a resolution of Φ and, again by (3), $\mathrm{Exp}_\Lambda(\mathbb{V}_{\langle R',\Omega,\rightarrow'_R\rangle}) = \mathrm{Exp}_\Lambda(\mathbb{V}_{\langle R,\Omega,\rightarrow_R\rangle})$. □

The rest of this section is devoted to showing that $\mathscr{V}([\Phi]) = \mathscr{A}^{\mathrm{p}}(\Phi)$ for any composition $\Phi = \Theta\|\Delta$ of a test Θ and process Δ; this amounts to showing

$$\{\$\Phi' \mid \Phi \Longrightarrow\!\!\succ \Phi'\} = \{\, \mathrm{Exp}_\Lambda(\mathbb{V}_{\langle R,\Omega,\rightarrow\rangle}) \mid \langle R,\Lambda,\rightarrow\rangle \text{ is a resolution of } \Phi\,\}$$

for any distribution Φ in an ω-respecting pLTS $\langle S,\Omega,\rightarrow\rangle$.

Let us see how an extreme derivation can be viewed as a method for dynamically generating a resolution.

Proposition 5.12 [Resolutions from extreme derivatives] Let $\Phi \Longrightarrow\!\!\succ \Phi'$ in a pLTS $\langle S,\Omega,\rightarrow\rangle$. Then there is a resolution $\langle R,\Lambda,\rightarrow_R\rangle$ of Φ, with resolving function f, such that $\Lambda \Longrightarrow\!\!\succ_R \Lambda'$ for some Λ' for which $\Phi' = \mathrm{Img}_f(\Lambda')$.

Proof: Consider an extreme derivation of $\Phi \Longrightarrow\!\!\succ \Phi'$ as given in Definition 4.3 where all $\Phi_k^\times$ must be stable:

$$\Phi = \Phi_0, \qquad \Phi_k = \Phi_k^\times + \Phi_k^\rightarrow, \qquad \Phi_k^\rightarrow \xrightarrow{\tau} \Phi_{k+1}, \qquad \Phi' = \textstyle\sum_{k=0}^\infty \Phi_k^\times .$$

By Lemma 4.2, $\Phi_k^\rightarrow \xrightarrow{\tau} \Phi_{k+1}$ implies that there are states $s_{ik}\in S$ and distributions $\Phi_{i(k+1)}\in\mathscr{D}(S)$, such that

$$\Phi_k^\rightarrow = \textstyle\sum_{i\in I_k} p_{ik}\cdot\overline{s_{ik}}, \qquad s_{ik} \xrightarrow{\tau} \Phi_{i(k+1)} \ \text{ for each } i\in I_k \qquad \text{and} \qquad \Phi_{k+1} = \textstyle\sum_{i\in I_k} p_{ik}\cdot\Phi_{i(k+1)}\ .$$

Let $\Phi_{ik}^\times(s) := \begin{cases}\Phi_{ik}(s) & \text{if } s \stackrel{\tau}{\not\longrightarrow}\\ 0 & \text{if } s \xrightarrow{\tau}\end{cases}$. Since $\Phi_k^\times(s) = \begin{cases}\Phi_k(s) & \text{if } s \stackrel{\tau}{\not\longrightarrow}\\ 0 & \text{if } s \xrightarrow{\tau}\end{cases}$ it follows that $\Phi_{k+1}^\times = \sum_{i\in I_k} p_{ik}\cdot\Phi_{i(k+1)}^\times$.

We will now define the resolution $\langle R,\Lambda,\rightarrow_R\rangle$ and the resolving function f. The set of states R is $(S\times\mathbb{N})\cup\bigcup_{k\in\mathbb{N}}(I_k\times\{k\})$. The resolving function $f: R\rightarrow S$ maps $(s,k)\in S\times\mathbb{N}$ to s and $(i,k)\in I_k\times\{k\}$ to $s_{ik}\in S$. The second component k of a state counts how many transitions have fired already: each transition in $\rightarrow_R$ goes from a state (i,k) or (s,k) to a distribution over $(S\cup I_{k+1})\times\{k{+}1\}$.

Define the subdistributions $\Lambda_k^\times\in\mathscr{D}_{sub}(S\times\{k\})$ and $\Lambda_k^\rightarrow\in\mathscr{D}_{sub}(I_k\times\{k\})$ by $\Lambda_k^\times(s,k) = \Phi_k^\times(s)$ and $\Lambda_k^\rightarrow(i,k) = p_{ik}$. Let $\Lambda_k := \Lambda_k^\times + \Lambda_k^\rightarrow$ and $\Lambda := \Lambda_0$. Furthermore, for all $k > 0$ and $i\in I_{k-1}$, define

$\Lambda_{ik} \in \mathscr{D}_{sub}((S \cup I_k) \times \{k\})$ by

$$\Lambda_{ik}(s,k) = \Phi_{ik}^{\times}(s) \qquad \text{and} \qquad \Lambda_{ik}(j,k) = p_{jk} \cdot \frac{\Phi_{ik}(s_{jk})}{\Phi_k(s_{jk})}$$

for $j \in I_k$. We introduce the transitions $(i,k) \xrightarrow{\tau}_R \Lambda_{i(k+1)}$ for $k \geq 0$ and $i \in I_k$. Moreover, for each state $s \in S$ and label $\alpha \in \mathsf{Act}_\tau$ such that $s \xrightarrow{\alpha}$, pick a transition $s \xrightarrow{\alpha} \Psi$, and add the transition $(s,k) \xrightarrow{\alpha}_R \Psi_{k+1}$ to $\rightarrow_R$, for all $k \in \mathbb{N}$. Here Ψ_{k+1} is the distribution with $\Psi_{k+1}(t,k+1) = \Psi(t)$ for all $t \in S$. Likewise, for each $k \in \mathbb{N}$, $i \in I_k$ and $\omega \in \Omega$ such that $s_{ik} \xrightarrow{\omega}$, pick a transition $s_{ik} \xrightarrow{\omega} \Psi$, and add the transition $(i,k) \xrightarrow{\omega}_R \Psi_{k+1}$ to $\rightarrow_R$. This ends the definition of the resolution $\langle R, \Lambda, \rightarrow_R \rangle$ and the resolving function f. By construction, $\langle R, \Omega, \rightarrow_R \rangle$ is a deterministic pLTS. We now check that f satisfies the requirements for a resolving function of Definition 3.1.

(i) $$\mathrm{Img}_f(\Lambda_k)(s) = \Lambda_k(s,k) + \sum_{s_{ik}=s} \Lambda_k(i,k) = \Lambda_k^{\times}(s,k) + \sum_{s_{ik}=s} p_{ik} = \Phi_k^{\times}(s) + \Phi_k^{\rightarrow}(s) = \Phi_k(s)$$
for all $s \in S$, so $\mathrm{Img}_f(\Lambda_k) = \Phi_k$, and in particular $\mathrm{Img}_f(\Lambda) = \Phi$.

(ii) Let $r \xrightarrow{\alpha}_R \Gamma$ for $\alpha \in \Omega_\tau$. In case $r = (s,k)$ it must be that $\Gamma = \Psi_{k+1}$ and $f(r) = s \xrightarrow{\alpha} \Psi = \mathrm{Img}_f(\Psi_{k+1})$. Likewise, in case $r = (i,k)$ and $\alpha \in \Omega$ it must be that $\Gamma = \Psi_{k+1}$ and $f(r) = s_{ik} \xrightarrow{\alpha} \Psi = \mathrm{Img}_f(\Psi_{k+1})$. The remaining case is $r = (i,k)$, $\alpha = \tau$ and $\Gamma = \Lambda_{i(k+1)}$. Then $f(r) = s_{ik} \xrightarrow{\tau} \Phi_{i(k+1)}$, so it suffices to show that $\mathrm{Img}_f(\Lambda_{ik}) = \Phi_{ik}$ for all $k \in \mathbb{N}$ and $i \in I_k$. For any $s \in S$ we have

$$\mathrm{Img}_f(\Lambda_{ik})(s) = \Lambda_{ik}(s,k) + \sum_{s_{jk}=s} \Lambda_{ik}(j,k) = \Phi_{ik}^{\times}(s) + \sum_{s_{jk}=s} p_{jk} \cdot \frac{\Phi_{ik}(s_{jk})}{\Phi_k(s_{jk})} = \Phi_{ik}^{\times}(s) + \frac{\Phi_{ik}(s)}{\Phi_k(s)} \cdot \sum_{s_{jk}=s} p_{jk}\,.$$

In case $s \xrightarrow{\tau}\!\!\!\!\!/$ we have $s_{jk} = s$ for no $j \in I_k$, so $\mathrm{Img}_f(\Lambda_{ik})(s) = \Phi_{ik}^{\times}(s) = \Phi_{ik}(s)$.
In case $s \xrightarrow{\tau}$ we have $\Phi_{ik}^{\times}(s) = 0$ and $\sum_{s_{jk}=s} p_{jk} = \Phi_k^{\rightarrow}(s) = \Phi_k(s)$, so again $\mathrm{Img}_f(\Lambda_{ik})(s) = \Phi_{ik}(s)$.

(iii) Let $f(r) \xrightarrow{\alpha}$ for $\alpha \in \Omega_\tau$. By construction there is a Ψ_{k+1} such that $r \xrightarrow{\alpha}_R \Psi_{k+1}$.

Hence $\langle R, \Lambda, \rightarrow_R \rangle$ is a resolution of Φ. We have:

$$\sum_{i \in I_k} p_{ik} \cdot \Lambda_{i(k+1)}(s,k+1) = \sum_{i \in I_k} p_{ik} \cdot \Phi_{i(k+1)}^{\times}(s) = \Phi_{k+1}^{\times}(s) = \Lambda_{k+1}^{\times}(s,k+1) = \Lambda_{k+1}(s,k+1)$$

$$\sum_{i \in I_k} p_{ik} \cdot \Lambda_{i(k+1)}(j,k+1) = \sum_{i \in I_k} p_{ik} \cdot p_{j(k+1)} \cdot \frac{\Phi_{i(k+1)}(s_{j(k+1)})}{\Phi_{k+1}(s_{j(k+1)})} = p_{j(k+1)} = \Lambda_{k+1}^{\rightarrow}(j,k+1) = \Lambda_{k+1}(j,k+1).$$

Hence $\Lambda_{k+1} = \sum_{i \in I_k} p_{ik} \cdot \Lambda_{i(k+1)}$. Since also $\Lambda_k^{\rightarrow} = \sum_{i \in I_k} p_{ik} \cdot \overline{(i,k)}$ and $(i,k) \xrightarrow{\tau}_R \Lambda_{i(k+1)}$, Lemma 4.2 yields $\Lambda_k^{\rightarrow} \xrightarrow{\tau}_R \Lambda_{k+1}$. Let $\Lambda' = \sum_{k=0}^{\infty} \Lambda_k^{\times}$. Then, by Definition 4.3, $\Lambda \Longrightarrow\!\!\!\!\!\rightarrow_R \Lambda'$.

By construction $\mathrm{Img}_f(\Lambda_k^{\times}) = \Phi_k^{\times}$ for all $k \in \mathbb{N}$. Hence $\mathrm{Img}_f(\Lambda') = \sum_{k=0}^{\infty} \mathrm{Img}_f(\Lambda_k^{\times}) = \sum_{k=0}^{\infty} \Phi_k^{\times} = \Phi'$. □

The converse is somewhat simpler.

Proposition 5.13 [Extreme derivatives from resolutions] Let $\langle R, \Lambda, \rightarrow_R \rangle$ be a resolution of a subdistribution Φ in a pLTS $\langle S, \Omega, \rightarrow \rangle$ with resolving function f. Then $\Lambda \Longrightarrow\!\!\!\!\!\rightarrow_R \Lambda'$ implies $\Phi \Longrightarrow\!\!\!\!\!\rightarrow \mathrm{Img}_f(\Lambda')$.

Proof: The definition of Img_f implies that $\mathrm{Img}_f(\sum_i p_i \cdot \Psi_i) = \sum_i p_i \cdot \mathrm{Img}_f(\Psi_i)$. Furthermore $\Psi \xrightarrow{\tau} \Psi'$ implies $\mathrm{Img}_f(\Psi) \xrightarrow{\tau} \mathrm{Img}_f(\Psi')$. Namely, by Lemma 4.2, $\Psi \xrightarrow{\tau} \Psi'$ implies

$$\Psi = \textstyle\sum_{i \in I} p_i \cdot \overline{s_i}, \qquad s_i \xrightarrow{\tau} \Psi_i \ \text{ for each } i \in I \qquad \text{and} \qquad \Psi' = \textstyle\sum_{i \in I} p_i \cdot \Psi_i$$

which, using Definition 3.1, entails

$$\mathrm{Img}_f(\Psi) = \sum_{i\in I} p_i \cdot \overline{f(s_i)}, \quad f(s_i) \xrightarrow{\tau} \mathrm{Img}_f(\Psi_i) \text{ for each } i \in I \text{ and } \mathrm{Img}_f(\Psi') = \sum_{i\in I} p_i \cdot \mathrm{Img}_f(\Psi_i).$$

Hence $\mathrm{Img}_f(\Psi) \xrightarrow{\tau} \mathrm{Img}_f(\Psi')$.

Now consider any derivation of $\Lambda \Longrightarrow\!\!\!\!\!\rightarrow_R \Lambda'$ along the lines of Definition 4.3. By systematically applying the function f to the component subdistributions in this derivation we get a derivation $\mathrm{Img}_f(\Lambda) \Longrightarrow \mathrm{Img}_f(\Lambda')$, that is $\Phi \Longrightarrow \mathrm{Img}_f(\Lambda')$. To show that $\mathrm{Img}_f(\Lambda')$ is actually an extreme derivative it suffices to show that s is stable for every $s \in \lceil \mathrm{Img}_f(\Lambda') \rceil$. But if $s \in \lceil \mathrm{Img}_f(\Lambda') \rceil$ then by definition there is some $t \in \lceil \Lambda' \rceil$ such that $s = f(t)$. Since $\Lambda \Longrightarrow\!\!\!\!\!\rightarrow_R \Lambda'$ the state t must be stable. The stability of s now follows from requirement (iii) of Definition 3.1. □

Our next step is to relate the outcomes extracted from extreme derivatives to those extracted from the corresponding resolutions. This requires some analysis of the evaluation function $\mathbb{V}$ applied to ω-respecting deterministic pLTSs. We show that the function $\mathscr{F}$ defined in (3) on Page 6 and its least fixed point $\mathbb{V}$ are continuous with respect to the standard Euclidean metric.

Definition 5.14 [Continuous functions] An ω-chain in a complete lattice L is a sequence of elements $\{c_n \mid n \geq 0\}$ satisfying $c_i \leq c_{i+1}$. Since the lattice is complete, ω-chains have least upper bounds; we denote them by $\bigsqcup_{n\geq 0} c_n$. A function $f : L \to L$ is said to be (ω)-continuous [19] if it preserves the least upper bounds of ω-chains:

$$f(\bigsqcup_{n\geq 0} c_n) = \bigsqcup_{n\geq 0} f(c_n)\ .$$

Lemma 5.15 [Exchange of suprema] Let function $g : \mathbb{N} \times \mathbb{N} \to \mathbb{R}$ be such that it is

(i) monotonic in both of its arguments separately, so that $i \leq i'$ implies $g(i,j) \leq g(i',j)$ for all j, and $j \leq j'$ implies $g(i,j) \leq g(i,j')$ for all i, and

(ii) bounded above, so that there is a $c \in \mathbb{R}_{\geq 0}$ with $g(i,j) \leq c$ for all i, j.

Then

$$\lim_{i\to\infty} \lim_{j\to\infty} g(i,j) \;=\; \lim_{j\to\infty} \lim_{i\to\infty} g(i,j).$$

Proof: Conditions (i) and (ii) guarantee the existence of all the limits. Moreover, for a non-decreasing sequence its limit and supremum agree, and both sides equal the supremum of all $g(i,j)$ for $i, j \in \mathbb{N}$. In fact, $(\mathbb{R}, \leq)$ is a complete partial order (CPO), and it is a basic result of CPOs [19] that

$$\bigsqcup_{i\in\mathbb{N}} (\bigsqcup_{j\in\mathbb{N}} g(i,j)) \;=\; \bigsqcup_{j\in\mathbb{N}} (\bigsqcup_{i\in\mathbb{N}} g(i,j)).$$ □

The following technical proposition states that some real functions satisfy the property of *bounded continuity*, which allows the exchange of limit and sum operations. It plays a crucial role in proving the continuity of $\mathscr{F}$.

Proposition 5.16 [Bounded continuity] Given a function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{R}_{\geq 0}$ which satisfies the following conditions:

C1. f is monotonic in the second parameter, i.e. $j_1 \leq j_2$ implies $f(i,j_1) \leq f(i,j_2)$ for all $i, j_1, j_2 \in \mathbb{N}$;

C2. for any $i \in \mathbb{N}$, the limit $\lim_{j\to\infty} f(i,j)$ exists;

C3. the partial sums $S_n = \sum_{i=0}^{n} \lim_{j\to\infty} f(i,j)$ are bounded, i.e. there exists some $c \in \mathbb{R}_{\geq 0}$ such that $S_n \leq c$ for all $n \geq 0$;

then it holds that

$$\sum_{i=0}^{\infty} \lim_{j\to\infty} f(i,j) \;=\; \lim_{j\to\infty} \sum_{i=0}^{\infty} f(i,j).$$

Proof: Let $g : \mathbb{N}\times\mathbb{N} \to \mathbb{R}_{\geq 0}$ be the function defined by $g(n,j) = \sum_{i=0}^{n} f(i,j)$. It is easy to see that g is monotonic in both arguments. By **C1** and **C2**, we have that $f(i,j) \leq \lim_{j\to\infty} f(i,j)$ for any $i,j \in \mathbb{N}$. So for any $j,n \in \mathbb{N}$ we have that

$$g(n,j) = \sum_{i=0}^{n} f(i,j) \;\leq\; \sum_{i=0}^{n} \lim_{j\to\infty} f(i,j) \;\leq\; c$$

according to **C3**. In other words, g is bounded above. Therefore we can apply Lemma 5.15 and obtain

$$\lim_{n\to\infty}\lim_{j\to\infty} \sum_{i=0}^{n} f(i,j) \;=\; \lim_{j\to\infty}\lim_{n\to\infty} \sum_{i=0}^{n} f(i,j). \tag{8}$$

For any $j\in\mathbb{N}$, the sequence $\{g(n,j)\}_{n\geq 0}$ is nondecreasing and bounded, so its limit $\sum_{i=0}^{\infty} f(i,j)$ exists. That is,

$$\lim_{n\to\infty} \sum_{i=0}^{n} f(i,j) \;=\; \sum_{i=0}^{\infty} f(i,j). \tag{9}$$

In view of **C2**, we have that, for any given $n\in\mathbb{N}$, the limit $\lim_{j\to\infty}\sum_{i=0}^{n} f(i,j)$ exists and

$$\sum_{i=0}^{n} \lim_{j\to\infty} f(i,j) \;=\; \lim_{j\to\infty} \sum_{i=0}^{n} f(i,j). \tag{10}$$

By **C3** the sequence $\{S_n\}_{n\geq 0}$ is bounded. Since it is also nondecreasing, it converges to $\sum_{i=0}^{\infty} \lim_{j\to\infty} f(i,j)$. That is,

$$\lim_{n\to\infty} \sum_{i=0}^{n} \lim_{j\to\infty} f(i,j) \;=\; \sum_{i=0}^{\infty} \lim_{j\to\infty} f(i,j). \tag{11}$$

Hence the left-hand side of the desired equality exists. By combining (8)-(11) we obtain the result that $\sum_{i=0}^{\infty} \lim_{j\to\infty} f(i,j) \;=\; \lim_{j\to\infty} \sum_{i=0}^{\infty} f(i,j)$. □

Lemma 5.17 Let R be a set and $h : R \to [0,1]^{\Omega}$. Furthermore, let $\Delta_0 \leq \Delta_1 \leq \cdots$ be an ω-chain of subdistributions over R — here $\Delta \leq \Delta'$ iff $\Delta(r) \leq \Delta'(r)$ for all $r \in R$. Then $\mathrm{Exp}_{\bigsqcup_{n\geq 0}\Delta_n} h = \bigsqcup_{n\geq 0} \mathrm{Exp}_{\Delta_n} h$.

Proof:
$$\begin{aligned}
\big(\mathrm{Exp}_{\bigsqcup_{n\geq 0}\Delta_n} h\big)(\omega) &= \big(\textstyle\sum_{r\in R}(\bigsqcup_{n\geq 0}\Delta_n)(r)\cdot h(r)\big)(\omega)\\
&= \big(\textstyle\sum_{r\in R}(\bigsqcup_{n\geq 0}\Delta_n(r))\cdot h(r)\big)(\omega)\\
&= \big(\textstyle\sum_{r\in R}\bigsqcup_{n\geq 0}(\Delta_n(r)\cdot h(r))\big)(\omega)\\
&= \textstyle\sum_{r\in R}\bigsqcup_{n\geq 0}(\Delta_n(r)\cdot h(r)(\omega))\\
&= \textstyle\sum_{r\in R}\lim_{n\to\infty}(\Delta_n(r)\cdot h(r)(\omega))\\
&= \textstyle\lim_{n\to\infty}\sum_{r\in R}(\Delta_n(r)\cdot h(r)(\omega)) \quad \text{by Proposition 5.16}\\
&= \textstyle\bigsqcup_{n\geq 0}\sum_{r\in R}(\Delta_n(r)\cdot h(r)(\omega))\\
&= \big(\textstyle\bigsqcup_{n\geq 0}\sum_{r\in R}(\Delta_n(r)\cdot h(r))\big)(\omega)\\
&= \big(\textstyle\bigsqcup_{n\geq 0}\mathrm{Exp}_{\Delta_n} h\big)(\omega).
\end{aligned}$$

In the above reasoning, Proposition 5.16 can be applied because we can define $f : R\times\mathbb{N} \to \mathbb{R}_{\geq 0}$ by letting $f(r,n) = \Delta_n(r)\cdot h(r)(\omega)$ and checking that f satisfies the three conditions in Proposition 5.16. If R is finite, we can extend it to a countable set $R' \supseteq R$ and require $f(r',n) = 0$ for all $r' \in R'\backslash R$ and $n \in \mathbb{N}$.

1. f satisfies condition **C1**. For any $r \in R$ and $j_1, j_2 \in \mathbb{N}$, if $j_1 \leq j_2$ then $\Delta_{j_1} \leq \Delta_{j_2}$. It follows that
$$f(r, j_1) \;=\; \Delta_{j_1}(r) \cdot h(r)(\omega) \;\leq\; \Delta_{j_2}(r) \cdot h(r)(\omega) \;=\; f(r, j_2).$$
2. f satisfies condition **C2**. For any $r \in R$, the sequence $\{\Delta_n(r) \cdot h(r)(\omega)\}_{n \geq 0}$ is nondecreasing and bounded by $h(r)(\omega)$. It follows that the limit $\lim_{n\to\infty} f(r,n)$ exists.
3. f satisfies condition **C3**. For any finite $R'' \subseteq R$, the partial sum $\sum_{r\in R''} \lim_{n\to\infty} f(r,n)$ is bounded because
$$\begin{array}{rcccl} \sum_{r\in R''} \lim_{n\to\infty} f(r,n) & = & \lim_{n\to\infty} \sum_{r\in R''} f(r,n) & = & \lim_{n\to\infty} \sum_{r\in R''} \Delta_n(r) \cdot h(r)(\omega) \\ \leq \lim_{n\to\infty} \sum_{r\in R''} \Delta_n(r) & \leq & \lim_{n\to\infty} \sum_{r\in R} \Delta_n(r) & \leq & \lim_{n\to\infty} 1 = 1. \end{array}$$ □

Lemma 5.18 Consider a deterministic pLTS $\langle R, \Omega, \rightarrow \rangle$. The function $\mathscr{F}$ defined in (3) is continuous.

Proof: Let $f_0 \leq f_1 \leq \ldots$ be an increasing chain in $R \to [0,1]^\Omega$. We need to show that
$$\mathscr{F}(\bigsqcup_{n\geq 0} f_n) \;=\; \bigsqcup_{n\geq 0} \mathscr{F}(f_n) \tag{12}$$

For any $r \in R$, we are in one of the following three cases:

1. $r \xrightarrow{\omega}$ for some $\omega \in \Omega$. We have
$$\begin{array}{rcll} \mathscr{F}(\bigsqcup_{n\geq 0} f_n)(r)(\omega) & = & 1 & \text{by (3)} \\ & = & \bigsqcup_{n\geq 0} 1 & \\ & = & \bigsqcup_{n\geq 0} \mathscr{F}(f_n)(r)(\omega) & \\ & = & (\bigsqcup_{n\geq 0} \mathscr{F}(f_n))(r)(\omega) & \end{array}$$
and
$$\mathscr{F}(\bigsqcup_{n\geq 0} f_n)(r)(\omega') = 0 = (\bigsqcup_{n\geq 0} \mathscr{F}(f_n))(r)(\omega')$$
for all $\omega' \neq \omega$.
2. $r \not\rightarrow$. Similar to the last case. We have
$$\mathscr{F}(\bigsqcup_{n\geq 0} f_n)(r)(\omega) = 0 = (\bigsqcup_{n\geq 0} \mathscr{F}(f_n))(r)(\omega)$$
for all $\omega \in \Omega$.
3. Otherwise, $r \xrightarrow{\tau} \Delta$ for some $\Delta \in \mathscr{D}(R)$. Then we infer that, for any $\omega \in \Omega$,
$$\begin{array}{rcll} \mathscr{F}(\bigsqcup_{n\geq 0} f_n)(r)(\omega) & = & \mathrm{Exp}_\Delta(\bigsqcup_{n\geq 0} f_n)(\omega) & \text{by (3)} \\ & = & \sum_{r\in\lceil\Delta\rceil} \Delta(r) \cdot (\bigsqcup_{n\geq 0} f_n)(r)(\omega) & \\ & = & \sum_{r\in\lceil\Delta\rceil} \Delta(r) \cdot (\bigsqcup_{n\geq 0} f_n(r))(\omega) & \\ & = & \sum_{r\in\lceil\Delta\rceil} \bigsqcup_{n\geq 0} \Delta(r) \cdot f_n(r)(\omega) & \\ & = & \sum_{r\in\lceil\Delta\rceil} \lim_{n\to\infty} \Delta(r) \cdot f_n(r)(\omega) & \\ & = & \lim_{n\to\infty} \sum_{r\in\lceil\Delta\rceil} \Delta(r) \cdot f_n(r)(\omega) & \text{by Proposition 5.16} \\ & = & \bigsqcup_{n\geq 0} \sum_{r\in\lceil\Delta\rceil} \Delta(r) \cdot f_n(r)(\omega) & \\ & = & \bigsqcup_{n\geq 0} \mathrm{Exp}_\Delta(f_n)(\omega) & \\ & = & \bigsqcup_{n\geq 0} \mathscr{F}(f_n)(r)(\omega) & \\ & = & (\bigsqcup_{n\geq 0} \mathscr{F}(f_n))(r)(\omega)\ . & \end{array}$$
In the above reasoning, Proposition 5.16 can be applied because we can define the function $f : R \times \mathbb{N} \to \mathbb{R}_{\geq 0}$ by letting $f(r,n) = \Delta(r) \cdot f_n(r)(\omega)$ and checking that f satisfies the three conditions in Proposition 5.16. If R is finite, we can extend it to a countable set $R' \supseteq R$ and require $f(r',n) = 0$ for all $r' \in R' \backslash R$ and $n \in \mathbb{N}$.

(a) f satisfies condition **C1**. For any $r \in R$ and $j_1, j_2 \in \mathbb{N}$, if $j_1 \leq j_2$ then $f_{j_1} \leq f_{j_2}$. It follows that
$$f(r, j_1) \;=\; \Delta(r) \cdot f_{j_1}(r)(\omega) \;\leq\; \Delta(r) \cdot f_{j_2}(r)(\omega) \;=\; f(r, j_2).$$

(b) f satisfies condition **C2**. For any $r \in R$, the sequence $\{\Delta(r) \cdot f_n(r)(\omega)\}_{n \geq 0}$ is nondecreasing and bounded by $\Delta(r)$. It follows that the limit $\lim_{n\to\infty} f(r,n)$ exists.

(c) f satisfies condition **C3**. For any $R'' \subseteq R$, the partial sum $\sum_{r\in R''} \lim_{n\to\infty} f(r,n)$ is bounded because
$$\sum_{r\in R''} \lim_{n\to\infty} f(r,n) = \sum_{r\in R''} \lim_{n\to\infty} \Delta(r) \cdot f_n(r)(\omega) \leq \sum_{r\in R''} \Delta(r) \leq \sum_{r\in R} \Delta(r) = 1. \qquad \Box$$

The continuity of $\mathscr{F}$ implies that its fixed point $\mathbb{V}$ can be captured by a chain of approximants. The functions $\mathbb{V}^n$, $n \geq 0$ are defined by induction on n:
$$\mathbb{V}^0(r)(\omega) = 0 \qquad \text{for all } r \in R \text{ and } \omega \in \Omega$$
$$\mathbb{V}^{n+1} = \mathscr{F}(\mathbb{V}^n)$$

Now $\mathbb{V} = \bigsqcup_{n\geq 0} \mathbb{V}^n$. This is used in the following result.

Lemma 5.19 Let Λ be a subdistribution in an ω-respecting deterministic pLTS $\langle R, \Omega, \rightarrow_R \rangle$. If $\Lambda \Longrightarrow\!\!\!\!\!\!\rightarrow \Lambda'$ then $\mathrm{Exp}_\Lambda(\mathbb{V}_{\langle R,\Omega,\rightarrow_R\rangle}) = \mathrm{Exp}_{\Lambda'}(\mathbb{V}_{\langle R,\Omega,\rightarrow_R\rangle})$.

Proof: For simplicity let us write $\mathbb{V}(\Delta)$ for $\mathrm{Exp}_\Delta(\mathbb{V}_{\langle R,\Omega,\rightarrow_R\rangle})$ for any Δ. Since the pLTS is ω-respecting we know that $s \xrightarrow{\tau} \Delta$ implies $s \not\xrightarrow{\omega}$ for any ω. Therefore, from the definition of the functional $\mathscr{F}$ we have that $s \xrightarrow{\tau} \Delta$ implies $\mathbb{V}^{n+1}(s) = \mathbb{V}^n(\Delta)$, whence by lifting and linearity we get:
$$\text{if } \Delta \xrightarrow{\tau} \Delta' \text{ then } \mathbb{V}^{n+1}(\Delta) = \mathbb{V}^n(\Delta') \text{ for all } n \geq 0.$$

Now suppose $\Lambda \Longrightarrow\!\!\!\!\!\!\rightarrow \Lambda'$. Then
$$\Lambda = \Lambda_0, \qquad \Lambda_k = \Lambda_k^\times + \Lambda_k^\rightarrow, \qquad \Lambda_k^\rightarrow \xrightarrow{\tau} \Lambda_{k+1}, \qquad \Lambda' = \sum_{k=0}^{\infty} \Lambda_k^\times.$$

Using in the base case that $\mathbb{V}^0(\Delta)(\omega) = 0$ for each Δ, a straightforward induction on n yields, for all $\ell \geq 0$,
$$\mathbb{V}^n(\Lambda_\ell) \;=\; \sum_{k=0}^{n} \mathbb{V}^{n-k}(\Lambda_{\ell+k}^\times)\,. \tag{13}$$

Namely $\mathbb{V}^{n+1}(\Lambda_\ell) = \mathbb{V}^{n+1}(\Lambda_\ell^\times + \Lambda_\ell^\rightarrow) = \mathbb{V}^{n+1}(\Lambda_\ell^\times) + \mathbb{V}^{n+1}(\Lambda_\ell^\rightarrow) = \mathbb{V}^{n+1}(\Lambda_\ell^\times) + \mathbb{V}^n(\Lambda_{\ell+1}) \overset{\text{induction}}{=} \mathbb{V}^{n+1}(\Lambda_\ell^\times) + \sum_{k=0}^n \mathbb{V}^{n-k}(\Lambda_{\ell+1+k}^\times) = \mathbb{V}^{n+1}(\Lambda_\ell^\times) + \sum_{k=1}^{n+1} \mathbb{V}^{n+1-k}(\Lambda_{\ell+k}^\times) = \sum_{k=0}^{n+1} \mathbb{V}^{n+1-k}(\Lambda_{\ell+k}^\times)$.

Since $\Lambda_k^\times$ is stable, we have
$$\mathbb{V}^m(\Lambda_k^\times) = \mathbb{V}(\Lambda_k^\times) \qquad \text{for every } k, m \geq 0. \tag{14}$$

We conclude by reasoning

$\mathbb{V}(\Lambda)$	$= \bigsqcup_{n\geq 0} \mathbb{V}^n(\Lambda)$	by continuity of $\mathscr{F}$
	$= \bigsqcup_{n\geq 0} \sum_{k=0}^n \mathbb{V}^{n-k}(\Lambda_k^\times)$	from (13) above, taking $\ell = 0$
	$= \bigsqcup_{n\geq 0} \sum_{k=0}^n \mathbb{V}(\Lambda_k^\times)$	by (14)
	$= \bigsqcup_{n\geq 0} \mathbb{V}(\sum_{k=0}^n \Lambda_k^\times)$	by linearity of $\mathbb{V}$
	$= \mathbb{V}(\bigsqcup_{n\geq 0} \sum_{k=0}^n \Lambda_k^\times)$	by Lemma 5.17
	$= \mathbb{V}(\sum_{k=0}^\infty \Lambda_k^\times)$	
	$= \mathbb{V}(\Lambda')\,.$	

$\Box$

We are now ready to compare the two methods for calculating the set of outcomes associated with a subdistribution:

- using extreme derivatives and the reward function \$ from Definition 5.3
- using resolutions and the evaluation function $\mathbb{V}$ from page 6.

Theorem 5.20 In an ω-respecting pLTS $\langle S, \Omega, \rightarrow \rangle$, the following statements hold.

(a) If $\Phi \Longrightarrow\!\!\!\!\!\succ \Phi'$ then there is a resolution $\langle R, \Lambda, \rightarrow_R \rangle$ of Φ such that $\mathrm{Exp}_\Lambda(\mathbb{V}_{\langle R,\Omega,\rightarrow_R\rangle}) = \$\Phi'$.

(b) For any resolution $\langle R, \Lambda, \rightarrow_R \rangle$ of Φ, there exists a Φ' such that $\Phi \Longrightarrow\!\!\!\!\!\succ \Phi'$ and $\mathrm{Exp}_\Lambda(\mathbb{V}_{\langle R,\Omega,\rightarrow_R\rangle}) = \$\Phi'$.

Proof: Suppose $\Phi \Longrightarrow\!\!\!\!\!\succ \Phi'$. By Proposition 5.12, there is a resolution $\langle R, \Lambda, \rightarrow_R \rangle$ of Φ with resolving function f and a subdistribution Λ' such that $\Lambda \Longrightarrow\!\!\!\!\!\succ \Lambda'$ and $\Phi' = \mathrm{Img}_f(\Lambda')$. By Lemma 5.19, we have

$$\mathrm{Exp}_\Lambda(\mathbb{V}) = \mathrm{Exp}_{\Lambda'}(\mathbb{V}). \tag{15}$$

Since Λ' is an extreme derivative, all the states s in its support are stable, so $\mathbb{V}(s)(\omega) = 0$ if $s \stackrel{\omega}{\not\rightarrow}$, for all $\omega \in \Omega$. Hence

$$\mathrm{Exp}_{\Lambda'}(\mathbb{V})(\omega) = \sum_{s \in \lceil \Lambda' \rceil} \Lambda'(s) \cdot \mathbb{V}(s)(\omega) = \sum_{s \in \lceil \Lambda' \rceil,\ s \stackrel{\omega}{\longrightarrow}} \Lambda'(s) = \$\Lambda'(\omega)\,. \tag{16}$$

Furthermore, for all $t \in \lceil \Phi' \rceil$, $\Phi'(t) = \mathrm{Img}_f(\Lambda')(t) = \sum_{f(s)=t} \Lambda'(s)$, so, for all $\omega \in \Omega$,

$$\$\Phi'(\omega) = \sum_{t \in \lceil \Phi' \rceil,\ t \stackrel{\omega}{\longrightarrow}} \Phi'(t) = \sum_{t \in \lceil \Phi' \rceil,\ t \stackrel{\omega}{\longrightarrow}} \mathrm{Img}_f(\Lambda')(t) = \sum_{t \in \lceil \Phi' \rceil,\ t \stackrel{\omega}{\longrightarrow}} \sum_{f(s)=t} \Lambda'(s) = \sum_{s \in \lceil \Lambda' \rceil,\ f(s) \stackrel{\omega}{\longrightarrow}} \Lambda'(s) = \$\Lambda'(\omega)\,,$$

where in the last step we the use the property of resolutions that $f(s) \stackrel{\omega}{\longrightarrow}$ iff $s \stackrel{\omega}{\longrightarrow}$. Combining this with (15) and (16) yields that $\mathrm{Exp}_\Lambda(\mathbb{V}) = \$\Phi'$.

To prove part (b), suppose that $\langle R, \Lambda, \rightarrow_R \rangle$ is a resolution of Φ with resolving function f, so that $\Phi = \mathrm{Img}_f(\Lambda)$. We know from Lemma 5.2 that there exists a (unique) subdistribution Λ' such that $\Lambda \Longrightarrow\!\!\!\!\!\succ \Lambda'$. By Proposition 5.13 we have that $\Phi \Longrightarrow\!\!\!\!\!\succ \mathrm{Img}_f(\Lambda')$. The same arguments as in the other direction show that $\mathrm{Exp}_\Lambda(\mathbb{V}) = \$(\mathrm{Img}_f(\Lambda'))$. □

A direct consequence of the above theorem is that $\mathscr{V}(\Phi) = \mathscr{A}(\Phi)$ for any subdistribution Φ in an ω-respecting pLTS $\langle S, \Omega, \rightarrow \rangle$. This implies that $\mathscr{V}([\Phi]) = \mathscr{A}^{\mathrm{p}}(\Phi)$ for any subdistribution Φ in a pLTS $\langle S, \Omega, \rightarrow \rangle$. This, in turn, together with Proposition 5.11, implies the following result.

Corollary 5.21 For any subdistribution Φ in a pLTS $\langle S, \Omega, \rightarrow \rangle$ we have that $\mathscr{V}([\Phi]) = \mathscr{A}(\Phi)$. □

6 Agreement of nonnegative- and real-reward must testing

In this section we prove the agreement of $\sqsubseteq_{\mathbf{nr}\mathrm{must}}$ with $\sqsubseteq_{\mathbf{rr}\mathrm{must}}$ for finitary convergent processes, by using failure simulation [3], recalled in Definition 4.6, as a stepping stone.

Because we prune our pLTSs before extracting values from them, we will be concerned mainly with ω-respecting structures. Moreover, we require the pLTSs to be *convergent* in the sense that there is no wholly divergent state s, i.e. with $s \Longrightarrow \varepsilon$. It follows from Theorem 8 in [3], in combination with Lemma 4.4(iii), that on a finitary convergent pLTS, if $\Delta \Longrightarrow \Delta'$ with Δ a full distribution, then Δ' is a full distribution.

Lemma 6.1 Let Δ and Γ be full distributions in an ω-respecting finitary convergent pLTS $\langle S, \Omega, \rightarrow \rangle$. If distribution Γ is stable and $\Gamma \mathrel{\overline{\lhd_{FS}}} \Delta$, then $\$\Gamma \in \mathscr{V}(\Delta)$.

Proof: We first show that if s is stable and $s \lhd_{FS} \Delta$ with Δ a full distribution, then $\$\overline{s} \in \mathscr{V}(\Delta)$. Since s is stable, we have only two cases:

(i) $s \not\rightarrow$ Here $\$\overline{s} = \overrightarrow{0}$, where $\overrightarrow{0}(\omega) = 0$ for all $\omega \in \Omega$. Since $s \lhd_{FS} \Delta$ we have $\Delta \Longrightarrow \Delta'$ with $\Delta' \not\rightarrow$, whence in fact $\Delta \Longrightarrow\!\!\!\!\succ \Delta'$ and $\$\Delta' = \overrightarrow{0}$. Thus $\$\overline{s} = \overrightarrow{0} \in \mathscr{V}(\Delta)$.

(ii) $s \xrightarrow{\omega} \Gamma'$ for some Γ' Here $\$\overline{s} = \overrightarrow{\omega}$, and since $s \lhd_{FS} \Delta$ we have $\Delta \Longrightarrow \Delta' \xrightarrow{\omega}$. As remarked above, also Δ' is a full distribution. Hence $\$\Delta' = \overrightarrow{\omega}$. Because the pLTS is ω-respecting, in fact $\Delta \Longrightarrow\!\!\!\!\succ \Delta'$ and so again $\$\overline{s} = \overrightarrow{\omega} \in \mathscr{V}(\Delta)$.

Now for the general case we suppose $\Gamma \overline{\lhd_{FS}} \Delta$. By Lemma 4.2 there is an index set I and states s_i, subdistributions Δ_i and probabilities p_i for $i \in I$, with $\sum_{i\in I} p_i \leq 1$, such that

$$\Gamma = \textstyle\sum_{i\in I} p_i \cdot \overline{s_i}, \qquad s_i \lhd_{FS} \Delta_i \text{ for each } i \in I \qquad \text{and} \qquad \Delta = \textstyle\sum_{i\in I} p_i \cdot \Delta_i.$$

Since Δ is full, $\sum_{i\in I} p_i = 1$ and the Δ_i are full distributions. Since Γ is stable, each state s_i is stable. From above we have that $\$\overline{s_i} \in \mathscr{V}(\Delta_i)$ for all $i \in I$, and so $\$\Gamma = \sum_{i\in I} p_i \cdot \$\overline{s_i} \in \sum_{i\in I} p_i \cdot \mathscr{V}(\Delta_i) = \mathscr{V}(\Delta)$, using Lemma 5.4. □

Lemma 6.2 Let Δ and Γ be full distributions in an ω-respecting finitary convergent pLTS $\langle S, \Omega, \rightarrow\rangle$. Then $\Delta \sqsubseteq_{FS} \Gamma$ implies $\mathscr{V}(\Delta) \supseteq \mathscr{V}(\Gamma)$.

Proof: Let $\Gamma, \Delta \in \mathscr{D}(S)$. We first claim that

(i) If $\Delta \Longrightarrow \Delta'$ then $\mathscr{V}(\Delta') \subseteq \mathscr{V}(\Delta)$.

(ii) If $\Gamma \overline{\lhd_{FS}} \Delta$, then we have $\mathscr{V}(\Gamma) \subseteq \mathscr{V}(\Delta)$.

The first claim holds because if $\Delta' \Longrightarrow\!\!\!\!\succ \Delta''$ then $\Delta \Longrightarrow \Delta' \Longrightarrow\!\!\!\!\succ \Delta''$, i.e. every extreme derivative of Δ' is also an extreme derivative of Δ. For the second claim, we assume $\Gamma \overline{\lhd_{FS}} \Delta$. For any $\Gamma \Longrightarrow\!\!\!\!\succ \Gamma'$ Lemma 4.7 gives a matching transition $\Delta \Longrightarrow \Delta'$ such that $\Gamma' \overline{\lhd_{FS}} \Delta'$. By definition Γ' is stable and since $\langle S, \Omega, \rightarrow\rangle$ is finitary and convergent Δ' and Γ' must be full. It follows from Lemma 6.1 and Claim (i) that $\$\Gamma' \in \mathscr{V}(\Delta') \subseteq \mathscr{V}(\Delta)$. Consequently, we obtain $\mathscr{V}(\Gamma) \subseteq \mathscr{V}(\Delta)$.

Now suppose $\Delta \sqsubseteq_{FS} \Gamma$. By definition there exists some Δ' such that $\Delta \Longrightarrow \Delta'$ and $\Gamma \overline{\lhd_{FS}} \Delta'$. By the above two claims we obtain $\mathscr{V}(\Gamma) \subseteq \mathscr{V}(\Delta') \subseteq \mathscr{V}(\Delta)$. □

This lemma shows that the failure-simulation preorder is a very strong relation in the sense that if Δ is related to Γ by the failure-simulation preorder then the set of outcomes generated by Δ includes the set of outcomes given by Γ. It is mainly due to this strong property that we can show that the failure-simulation preorder is sound for the real-reward must-testing preorder. Convergence is a crucial condition in this lemma.

Theorem 6.3 For any finitary convergent processes Δ and Γ, if $\Delta \sqsubseteq_{FS} \Gamma$ then we have that $\Delta \sqsubseteq_{\mathbf{rr}\text{must}} \Gamma$.

Proof: We reason as follows.

	$\Delta \sqsubseteq_{FS} \Gamma$	
implies	$[\Theta \| \Delta] \sqsubseteq_{FS} [\Theta \| \Gamma]$	Lemma 5.10, for any Ω-test Θ
implies	$\mathscr{V}([\Theta \| \Delta]) \supseteq \mathscr{V}([\Theta \| \Gamma])$	$[\cdot]$ is ω-respecting; Lemma 6.2
iff	$\mathscr{A}(\Theta, \Delta) \supseteq \mathscr{A}(\Theta, \Gamma)$	Corollary 5.21
implies	$h \cdot \mathscr{A}(\Theta, \Delta) \supseteq h \cdot \mathscr{A}(\Theta, \Gamma)$ for any $h \in [-1,1]^\Omega$	
implies	$\bigsqcap h \cdot \mathscr{A}(\Theta, \Delta) \leq \bigsqcap h \cdot \mathscr{A}(\Theta, \Gamma)$ for any $h \in [-1,1]^\Omega$	
iff	$\Delta \sqsubseteq^{\Omega}_{\mathbf{rr}\text{must}} \Gamma$.	

Note that in the second line above, both $[\Theta\|\Delta]$ and $[\Theta\|\Gamma]$ are convergent, since for any convergent process Ξ and finite process Θ, by induction on the structure of Θ, it can be shown that the composition $\Theta\|\Xi$ is also convergent. Furthermore, since processes Δ,Γ and tests Θ are defined to be full distributions, also $[\Theta\|\Delta]$ and $[\Theta\|\Gamma]$ are full. □

The proof of the above theorem is subtle. The failure-simulation preorder is defined via weak derivations (cf. Definition 4.6), while the reward must-testing preorder is defined in terms of resolutions (cf. Definition 3.5). Fortunately, we have shown in Corollary 5.21 that we can just as well characterise the reward must-testing preorder in terms of weak derivations. Based on this observation, the proof was carried out by exploiting Lemmas 5.10 and 6.2.

This result does not extend to divergent processes. One witness example is given in Figure 1. A simpler example is as follows. Let Δ be a process that diverges, by performing a τ-loop only, and let Γ be a process that merely performs a single action a. It holds that $\Delta \sqsubseteq_{FS} \Gamma$ because $\Delta \Longrightarrow \varepsilon$ and the empty subdistribution can failure-simulate any processes. However, if we apply the test $\overline{t}$ from Example 3.2 again, and the reward tuple h with $h(\omega) = -1$, then

$$\begin{array}{lllllllll} \bigsqcap h\cdot\mathscr{A}(\overline{t},\Delta) & = & \bigsqcap h\cdot\mathscr{V}([\overline{t}\|\Delta]) & = & \bigsqcap h\cdot\{\$\varepsilon\} & = & \bigsqcap\{0\} & = & 0 \\ \bigsqcap h\cdot\mathscr{A}(\overline{t},\Gamma) & = & \bigsqcap h\cdot\mathscr{V}([\overline{t}\|\Gamma]) & = & \bigsqcap h\cdot\{\overrightarrow{\omega}\} & = & \bigsqcap\{-1\} & = & -1 \end{array}$$

As $\bigsqcap h\cdot\mathscr{A}(\overline{t},\Delta) \not\leq \bigsqcap h\cdot\mathscr{A}(\overline{t},\Gamma)$, we see that $\Delta \not\sqsubseteq_{\mathbf{rr}\text{must}} \Gamma$. Since $\mathscr{V}([\overline{t}\|\Gamma]) = \{\overrightarrow{\omega}\}$ but $\overrightarrow{\omega} \notin \mathscr{V}([\overline{t}\|\Delta])$, this also is a counterexample against an extension of Lemma 6.2 with divergence.

Finally, by combining Theorems 3.6(ii) and 4.8(ii), together with Theorem 6.3, we obtain the main result of the paper which states that, in the absence of divergence, nonnegative-reward must testing is as discriminating as real-reward must testing.

Theorem 6.4 For any finitary convergent processes Δ and Γ, it holds that $\Delta \sqsubseteq_{\mathbf{rr}\text{must}} \Gamma$ if and only if $\Delta \sqsubseteq_{\mathbf{nr}\text{must}} \Gamma$.

Proof: The "only if" direction is obvious (cf. Definition 3.5). For the "if" direction, suppose Δ and Γ are finitary convergent processes. We reason as follows.

	$\Delta \sqsubseteq^{\Omega}_{\mathbf{nr}\text{must}} \Gamma$	
iff	$\Delta \sqsubseteq^{\Omega}_{\text{pmust}} \Gamma$	Theorem 3.6(ii)
iff	$\Delta \sqsubseteq_{FS} \Gamma$	Theorem 4.8(ii)
implies	$\Delta \sqsubseteq^{\Omega}_{\mathbf{rr}\text{must}} \Gamma$.	Theorem 6.3 □

7 Discussion

Below we give a characterisation of $\sqsubseteq_{\mathbf{rr}\text{must}}$ in terms of the set inclusion relation between testing outcome sets. As a similar characterisation for $\sqsubseteq_{\mathbf{nr}\text{must}}$ does in general not hold for finitary (non-convergent) processes, hopefully this gives some indication of the subtle difference between $\sqsubseteq_{\mathbf{rr}\text{must}}$ and $\sqsubseteq_{\mathbf{nr}\text{must}}$, and we see more clearly why our proof of Theorem 6.4 involves the failure simulation preorder.

Theorem 7.1 Let Δ and Γ be any finitary processes. Then $\Delta \sqsubseteq_{\mathbf{rr}\text{must}} \Gamma$ if and only if $\mathscr{A}(\Theta,\Delta) \supseteq \mathscr{A}(\Theta,\Gamma)$ for any Ω-test Θ.

Proof: ($\Leftarrow$) Let Θ be any Ω-test and $h \in [-1,1]^{\Omega}$ be any real-reward tuple. Suppose $\mathscr{A}(\Theta,\Delta) \supseteq \mathscr{A}(\Theta,\Gamma)$. It is obvious that $h\cdot\mathscr{A}(\Theta,\Delta) \supseteq h\cdot\mathscr{A}(\Theta,\Gamma)$, from which it easily follows that

$$\bigsqcap h\cdot\mathscr{A}(\Theta,\Delta) \;\leq\; \bigsqcap h\cdot\mathscr{A}(\Theta,\Gamma).$$

As this holds for an arbitrary real-reward tuple h, we see that $\Delta \sqsubseteq_{\mathbf{rr}\text{must}} \Gamma$.

($\Rightarrow$) Suppose for a contradiction that there is some Ω-test Θ with $\mathscr{A}(\Theta,\Delta) \not\supseteq \mathscr{A}(\Theta,\Gamma)$. Then there exists some outcome $o \in \mathscr{A}(\Theta,\Gamma)$ lying outside $\mathscr{A}(\Theta,\Delta)$, i.e.

$$o \notin \mathscr{A}(\Theta,\Delta). \tag{17}$$

Since Θ is finite, it contains only finitely many elements of Ω, so that we may assume wlog that Ω is finite. Since Δ and Θ are finitary, it is easy to see that the pruned composition $[\Delta\|\Theta]$ is also finitary. By Theorem 1/Corollary 1 in [3], the set $\{\Phi \mid [\Delta\|\Theta] \Longrightarrow \Phi\}$ is convex and compact. With an analogous proof, it can be shown that so is the set $\{\Phi \mid [\Delta\|\Theta] \Longrightarrow\!\!\!\!\succ \Phi\}$. It follows that the set

$$\{\$\Phi \mid [\Delta\|\Theta] \Longrightarrow\!\!\!\!\succ \Phi\}$$

i.e. $\mathscr{V}([\Theta\|\Delta])$, is also convex and compact. By Corollary 5.21 the set $\mathscr{A}(\Theta,\Delta)$ is thus convex and compact. Combining this with (17), and using the Separation Hyperplane Lemma [7, 12], we infer the existence of some hyperplane whose normal is $h \in \mathbb{R}^{\Omega}$ such that $h \cdot o' > h \cdot o$ for all $o' \in \mathscr{A}(\Theta,\Delta)$. By scaling h, we obtain without loss of generality that $h \in [-1,1]^{\Omega}$. It follows that

$$\bigsqcap h \cdot \mathscr{A}(\Theta,\Delta) \;>\; h \cdot o \;\geq\; \bigsqcap h \cdot \mathscr{A}(\Theta,\Gamma)$$

which is a contradiction to the assumption that $\Delta \sqsubseteq_{\mathbf{rr}\text{must}} \Gamma$. □

Note that in the above proof the normal of the separating hyperplane belongs to $[-1,1]^{\Omega}$ rather than $[0,1]^{\Omega}$. So we cannot repeat the above proof for $\sqsubseteq_{\mathbf{nr}\text{must}}$. In general, we do not have that $\Delta \sqsubseteq_{\mathbf{nr}\text{must}} \Gamma$ implies $\mathscr{A}(\Theta,\Delta) \supseteq \mathscr{A}(\Theta,\Gamma)$ for any Ω-test Θ and for arbitrary finitary processes Δ and Γ, that is finitary processes which might not be convergent. However, when we restrict ourselves to finitary convergent processes, this property does indeed hold, as can be seen from the first four lines in the proof of Theorem 6.3. Note that in that proof there is an essential use of the failure simulation preorder; in particular the pleasing property stated in Lemma 6.2. Even for finitary convergent processes we cannot give a direct and simple proof of that property for $\sqsubseteq_{\mathbf{nr}\text{must}}$, analogous to that of Theorem 7.1.

8 Conclusion

We have studied a notion of real-reward testing which extends the traditional nonnegative-reward testing with negative rewards. It turned out that the real-reward may preorder is the inverse of the real-reward must preorder, and vice versa. More interestingly, for finitary convergent processes, the real-reward must testing preorder coincides with the nonnegative-reward testing preorder. In order to prove this result, we have capitalised on a characterisation of nonnegative-reward testing in terms of a derivation based simulation preorder. Relating derivations to resolutions, on which the testing theories are based, involved proving some analytic properties such as the continuity of a function for calculating testing outcomes.

Although for finitary convergent processes real-reward must testing is no more powerful than nonnegative-reward must testing, the same does not hold for may testing. This is immediate from our result that (the inverse of) real-reward may testing is as powerful as real-reward must testing, that is known not to hold for nonnegative-reward may- and must testing. For finitary processes we know from [3] that $\sqsubseteq_{\mathbf{nr}\text{may}}$ and $\sqsubseteq_{\mathbf{nr}\text{must}}$ correspond to the simulation and failure simulation preorder respectively, and without divergence the latter is strictly more discriminating than the former.

References

[1] R. De Nicola & M. Hennessy (1984): *Testing equivalences for processes*. *Theoretical Computer Science* 34, pp. 83–133, doi:10.1016/0304-3975(84)90113-0.

[2] Y. Deng, R.J. van Glabbeek, M. Hennessy & C.C. Morgan (2008): *Characterising testing preorders for finite probabilistic processes*. *Logical Methods in Computer Science* 4(4):4, doi:10.2168/LMCS-4(4:4)2008.

[3] Y. Deng, R.J. van Glabbeek, M. Hennessy & C.C. Morgan (2009): *Testing finitary probabilistic processes*. In: Proc. *CONCUR'09*, LNCS 5710, Springer, pp. 274–288, doi:10.1007/978-3-642-04081-8_19.

[4] Y. Deng, R.J. van Glabbeek, M. Hennessy & C.C. Morgan (2009): *Testing finitary probabilistic processes*. Full version of [3]. Available at `http://www.cse.unsw.edu.au/~rvg/pub/finitary.pdf`.

[5] Y. Deng, R.J. van Glabbeek, M. Hennessy & C.C. Morgan (2011): *Real Reward Testing for Probabilistic Processes*. In: Proc. *QAPL'11*. *EPTCS* 57, pp. 61–73, doi:10.4204/EPTCS.57.5.

[6] Y. Deng, R.J. van Glabbeek, M. Hennessy, C.C. Morgan & C. Zhang (2007): *Remarks on Testing Probabilistic Processes*. *ENTCS* 172, pp. 359–397, doi:10.1016/j.entcs.2007.02.013.

[7] Y. Deng, R.J. van Glabbeek, C.C. Morgan & C. Zhang (2007): *Scalar Outcomes Suffice for Finitary Probabilistic Testing*. In: Proc. *ESOP'07*, LNCS 4421, Springer, pp. 363–368, doi:10.1007/978-3-540-71316-6_25.

[8] R.J. van Glabbeek (1993): *The Linear Time – Branching Time Spectrum II; The semantics of sequential systems with silent moves (extended abstract)*. In: Proc. *CONCUR'93*. LNCS 751, Springer, pp. 66–81, doi:10.1007/3-540-57208-2_6.

[9] M. Hennessy (1988): *An Algebraic Theory of Processes*. MIT Press.

[10] B. Jonsson, C. Ho-Stuart & Wang Yi (1994): *Testing and Refinement for Nondeterministic and Probabilistic Processes*. In: Proc. *FTRTFT'94*, LNCS 863, Springer, pp. 418–430, doi:10.1007/3-540-58468-4_176.

[11] D. Kozen (1985): *A Probabilistic PDL*. *JCSS* 30(2), pp. 162–178, doi:10.1016/0022-0000(85)90012-1.

[12] J. Matousek (2002): *Lectures on Discrete Geometry*. Springer.

[13] R. Milner (1989): *Communication and Concurrency*. Prentice-Hall.

[14] M.L. Puterman (1994): *Markov Decision Processes*. Wiley, doi:10.1002/9780470316887.

[15] J.J.M.M. Rutten, M.Kwiatkowska, G. Norman & D. Parker (2004): *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems*, P. Panangaden and F. van Breugel (eds.). *CRM Monograph Series* 23, American Mathematical Society.

[16] R. Segala (1995): *Modeling and Verification of Randomized Distributed Real-Time Systems*. Ph.D. thesis, MIT.

[17] R. Segala (1996): *Testing Probabilistic Automata*. In: Proceedings *CONCUR'96*, LNCS 1119, Springer, pp. 299–314, doi:10.1007/3-540-61604-7_62.

[18] Wang Yi & K.G. Larsen (1992): *Testing Probabilistic and Nondeterministic Processes*. In: Proc. *PSTV'92*. *IFIP Transactions* C-8, North-Holland, pp. 47–61.

[19] Glynn Winskel (1993): *The Formal Semantics of Programming Languages: An Introduction*. The MIT Press.

Symbolic bisimulation for quantum processes

Yuan Feng[1], Yuxin Deng[2], and Mingsheng Ying[1]

[1] University of Technology, Sydney, Australia, and Tsinghua University, China
[2]Shanghai Jiao Tong University, China

February 22, 2012

Abstract

With the previous notions of bisimulation presented in literature, to check if two quantum processes are bisimilar, we have to instantiate the free quantum variables of them with arbitrary quantum states, and verify the bisimilarity of resultant configurations. This makes checking bisimilarity infeasible from an algorithmic point of view, because quantum states constitute a continuum. In this paper, we introduce a *symbolic* operational semantics for quantum processes directly at the quantum operation level, which allows us to describe the bisimulation between quantum processes without resorting to quantum states. We show that the symbolic bisimulation defined here is equivalent to the open bisimulation for quantum processes in the previous work, when strong bisimulations are considered. An algorithm for checking symbolic ground bisimilarity is presented. We also give a modal logical characterisation for quantum bisimilarity based on an extension of Hennessy-Milner logic to quantum processes.

1 Introduction

An important issue in quantum process algebra is to discover a quantum generalisation of bisimulation preserved by various process constructs, in particular, parallel composition, where one of the major differences between classical and quantum systems, namely quantum entanglement, is present. Jorrand and Lalire [13, 15] defined a branching bisimulation for their *Quantum Process Algebra* (QPAlg), which identifies quantum processes whose associated graphs have the same branching structure. However, their bisimulation cannot always distinguish different quantum operations, as quantum states are only compared when they are input or output. Moreover, the derived bisimilarity is not a congruence; it is not preserved by restriction. Bisimulation defined in [7] indeed distinguishes different quantum operations but it works well only for finite processes. Again, it is not preserved by restriction. In [20], a congruent bisimulation was proposed for a special model where no classical datum is involved. However, as many important quantum communication protocols such as superdense coding and teleportation cannot be described in that model, the scope of its application is very limited.

A general notion of bisimulation for the quantum process algebra qCCS developed by the authors was found in [8], which enjoys the following nice features: (1) it is applicable to general models where both classical and quantum data are involved, and recursion is allowed; (2) it is preserved by all the standard process constructs, including parallel composition; and (3) quantum operations are regarded as invisible, so that they can be combined arbitrarily. Independently, a bisimulation congruence in *Communicating Quantum Processes* (CQP), developed by Gay and Nagarajan [11], was established by Davidson [5]. Later on, motivated by [18], an open bisimulation for quantum processes was defined in [6] that makes it possible to separate ground bisimulation and the closedness

under super-operator applications, thus providing not only a neater and simpler definition, but also a new technique for proving bisimilarity.

The various bisimulations defined in the literature, however, have a common shortcoming: they all resort to the instantiation of quantum variables by quantum states. As a result, to check whether or not two processes are bisimilar, we have to accompany them with an arbitrarily chosen quantum states, and check if the resultant configurations are bisimilar. Note that all quantum states constitute a continuum. The verification of bisimilarity is actually infeasible from an algorithmic point of view. The aim of the present paper is to tackle this problem by the powerful symbolic technique [12, 4]. This paper only considers qCCS, but the ideas and techniques developed here apply to other quantum process algebras.

As a quantum extension of value-passing CCS, qCCS has both (possibly infinite) classical data domain and (doomed-to-be infinite) quantum data domain. The possibly infinite classical data set can be dealt with by symbolic bisimulation [12] for classical process algebras directly. However, in qCCS, we are also faced with the additional difficulty caused by the infinity of all quantum states. The current paper solves this problem by introducing super-operator valued distributions, which allows us to fold the operational semantics of qCCS into a symbolic version and provides us with a notion, also called symbolic bisimulation for simplicity, where to check the bisimilarity of two quantum processes, only a finite number of process-superoperator pairs need to be considered, without appealing to quantum states. To be specific, we propose

- a symbolic operational semantics of qCCS in which quantum processes are described directly by the super-operators they can perform. It also incorporates a symbolic treatment for classical data.
- a notion of symbolic bisimulation, based on the symbolic operational semantics, as well as an efficient algorithm to check its ground version;
- the coincidence of symbolic bisimulation with the open bisimulation defined in [6], when strong bisimulation is considered.
- a modal characterisation of symbolic bisimulation by a quantum logic as an extension of Hennessy-Milner logic.

The remainder of the paper is organised as follows. In Section 2, we review some basic notions from linear algebra and quantum mechanics. The syntax and (ordinary) operational semantics of qCCS are presented in Section 3. We also review the definition of open bisimulation presented in [6]. Section 4 collects some definitions and properties of the semiring of completely positive super-operators. The notion of super-operator valued distributions, which serves as an extension of probabilistic distributions, is also defined. Section 5 is the main part of this paper where we present a symbolic operational semantics of qCCS which describes the execution of quantum processes without resorting to concrete quantum states. Based on it, symbolic bisimulation between quantum processes, which also incorporates a symbolic treatment for classical data, motivated by symbolic bisimulation for classical processes, is presented and shown to be equivalent to the open bisimulation in Section 3. Section 6 is devoted to proposing an algorithm to check symbolic ground bisimulation, which is applicable to reasoning about the correctness of existing quantum communication protocols. In section 7 we propose a modal logic which turns out to be both sound and complete with respect to the symbolic bisimulation. We outline the main results in Section 8 and point out some directions for further study. In particular, we suggest the potential application of our results in model checking quantum communication protocols.

2 Preliminaries

For convenience of the reader, we briefly recall some basic notions from linear algebra and quantum theory which are needed in this paper. For more details, we refer to [16].

2.1 Basic linear algebra

A *Hilbert space* $\mathcal{H}$ is a complete vector space equipped with an inner product

$$\langle\cdot|\cdot\rangle : \mathcal{H} \times \mathcal{H} \to \mathbf{C}$$

such that

(1) $\langle\psi|\psi\rangle \geq 0$ for any $|\psi\rangle \in \mathcal{H}$, with equality if and only if $|\psi\rangle = 0$;

(2) $\langle\phi|\psi\rangle = \langle\psi|\phi\rangle^*$;

(3) $\langle\phi|\sum_i c_i|\psi_i\rangle = \sum_i c_i\langle\phi|\psi_i\rangle$,

where $\mathbf{C}$ is the set of complex numbers, and for each $c \in \mathbf{C}$, c^* stands for the complex conjugate of c. For any vector $|\psi\rangle \in \mathcal{H}$, its length $|||\psi\rangle||$ is defined to be $\sqrt{\langle\psi|\psi\rangle}$, and it is said to be *normalized* if $|||\psi\rangle|| = 1$. Two vectors $|\psi\rangle$ and $|\phi\rangle$ are *orthogonal* if $\langle\psi|\phi\rangle = 0$. An *orthonormal basis* of a Hilbert space $\mathcal{H}$ is a basis $\{|i\rangle\}$ where each $|i\rangle$ is normalized and any pair of them are orthogonal.

Let $\mathcal{L}(\mathcal{H})$ be the set of linear operators on $\mathcal{H}$. For any $A \in \mathcal{L}(\mathcal{H})$, A is *Hermitian* if $A^\dagger = A$ where $A^\dagger$ is the adjoint operator of A such that $\langle\psi|A^\dagger|\phi\rangle = \langle\phi|A|\psi\rangle^*$ for any $|\psi\rangle, |\phi\rangle \in \mathcal{H}$. The fundamental *spectral theorem* states that the set of all normalized eigenvectors of a Hermitian operator in $\mathcal{L}(\mathcal{H})$ constitutes an orthonormal basis for $\mathcal{H}$. That is, there exists a so-called spectral decomposition for each Hermitian A such that

$$A = \sum_i \lambda_i|i\rangle\langle i| = \sum_{\lambda_i \in spec(A)} \lambda_i E_i$$

where the set $\{|i\rangle\}$ constitute an orthonormal basis of $\mathcal{H}$, $spec(A)$ denotes the set of eigenvalues of A, and E_i is the projector to the corresponding eigenspace of λ_i. A linear operator $A \in \mathcal{L}(\mathcal{H})$ is *unitary* if $A^\dagger A = AA^\dagger = I_\mathcal{H}$ where $I_\mathcal{H}$ is the identity operator on $\mathcal{H}$. The *trace* of A is defined as $\mathrm{tr}(A) = \sum_i\langle i|A|i\rangle$ for some given orthonormal basis $\{|i\rangle\}$ of $\mathcal{H}$. It is worth noting that trace function is actually independent of the orthonormal basis selected. It is also easy to check that trace function is linear and $\mathrm{tr}(AB) = \mathrm{tr}(BA)$ for any operators $A, B \in \mathcal{L}(\mathcal{H})$.

Let $\mathcal{H}_1$ and $\mathcal{H}_2$ be two Hilbert spaces. Their *tensor product* $\mathcal{H}_1 \otimes \mathcal{H}_2$ is defined as a vector space consisting of linear combinations of the vectors $|\psi_1\psi_2\rangle = |\psi_1\rangle|\psi_2\rangle = |\psi_1\rangle \otimes |\psi_2\rangle$ with $|\psi_1\rangle \in \mathcal{H}_1$ and $|\psi_2\rangle \in \mathcal{H}_2$. Here the tensor product of two vectors is defined by a new vector such that

$$\left(\sum_i \lambda_i|\psi_i\rangle\right) \otimes \left(\sum_j \mu_j|\phi_j\rangle\right) = \sum_{i,j} \lambda_i\mu_j|\psi_i\rangle \otimes |\phi_j\rangle.$$

Then $\mathcal{H}_1 \otimes \mathcal{H}_2$ is also a Hilbert space where the inner product is defined as the following: for any $|\psi_1\rangle, |\phi_1\rangle \in \mathcal{H}_1$ and $|\psi_2\rangle, |\phi_2\rangle \in \mathcal{H}_2$,

$$\langle\psi_1 \otimes \psi_2|\phi_1 \otimes \phi_2\rangle = \langle\psi_1|\phi_1\rangle_{\mathcal{H}_1}\langle\psi_2|\phi_2\rangle_{\mathcal{H}_2}$$

where $\langle\cdot|\cdot\rangle_{\mathcal{H}_i}$ is the inner product of $\mathcal{H}_i$. For any $A_1 \in \mathcal{L}(\mathcal{H}_1)$ and $A_2 \in \mathcal{L}(\mathcal{H}_2)$, $A_1 \otimes A_2$ is defined as a linear operator in $\mathcal{L}(\mathcal{H}_1 \otimes \mathcal{H}_2)$ such that for each $|\psi_1\rangle \in \mathcal{H}_1$ and $|\psi_2\rangle \in \mathcal{H}_2$,

$$(A_1 \otimes A_2)|\psi_1\psi_2\rangle = A_1|\psi_1\rangle \otimes A_2|\psi_2\rangle.$$

The *partial trace* of $A \in \mathcal{L}(\mathcal{H}_1 \otimes \mathcal{H}_2)$ with respected to $\mathcal{H}_1$ is defined as $\mathrm{tr}_{\mathcal{H}_1}(A) = \sum_i \langle i|A|i\rangle$ where $\{|i\rangle\}$ is an orthonormal basis of $\mathcal{H}_1$. Similarly, we can define the partial trace of A with respected to $\mathcal{H}_2$. Partial trace functions are also independent of the orthonormal basis selected.

Traditionally, a linear operator $\mathcal{E}$ on $\mathcal{L}(\mathcal{H})$ is called a *super-operator* on $\mathcal{H}$. A super-operator is said to be *completely positive* if it maps positive operators in $\mathcal{L}(\mathcal{H})$ to positive operators in $\mathcal{L}(\mathcal{H})$, and for any auxiliary Hilbert space $\mathcal{H}'$, the trivially extended operator $\mathcal{I}_{\mathcal{H}'} \otimes \mathcal{E}$ also maps positive operators in $\mathcal{L}(\mathcal{H}' \otimes \mathcal{H})$ to positive operators in $\mathcal{L}(\mathcal{H}' \otimes \mathcal{H})$. Here $\mathcal{I}_{\mathcal{H}'}$ is the identity operator on $\mathcal{L}(\mathcal{H}')$. The elegant and powerful *Kraus representation theorem* [14] of completely positive super-operators states that a super-operator $\mathcal{E}$ is completely positive if and only if there are some set of operators $\{E_i : i \in I\}$ with appropriate dimension such that

$$\mathcal{E}(A) = \sum_{i \in I} E_i A E_i^\dagger$$

for any $A \in \mathcal{L}(\mathcal{H})$. The operators E_i are called Kraus operators of $\mathcal{E}$. We abuse the notation slightly by denoting $\mathcal{E} = \{E_i : i \in I\}$. A super-operator $\mathcal{E}$ is said to be *trace-nonincreasing* if $\mathrm{tr}(\mathcal{E}(A)) \leq \mathrm{tr}(A)$ for any positive $A \in \mathcal{L}(\mathcal{H})$, and *trace-preserving* if the equality always holds. Equivalently, a super-operator is trace-nonincreasing completely positive (resp. trace-preserving completely positive) if and only if its Kraus operators E_i satisfy $\sum_i E_i^\dagger E_i \leq I$ (resp. $\sum_i E_i^\dagger E_i = I$). In this paper, we will use some well-known (unitary) super-operators listed as follows: the quantum control-not super-operator $\mathcal{CN} = \{C_N\}$ performed on two qubits where

$$C_N = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix},$$

the 1-qubit Hadamard super-operator $\mathcal{H} = \{H\}$, and Pauli super-operators $\sigma^0 = \{I_2\}, \sigma^1 = \{X\}, \sigma^2 = \{Z\}$, and $\sigma^3 = \{Y\}$ where

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \; Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \; Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}.$$

We also use the notations $\mathcal{X}, \mathcal{Z}$, and $\mathcal{Y}$ to denote σ^1, σ^2, and σ^3, respectively.

2.2 Basic quantum mechanics

According to von Neumann's formalism of quantum mechanics [19], an isolated physical system is associated with a Hilbert space which is called the *state space* of the system. A *pure state* of a quantum system is a normalized vector in its state space, and a *mixed state* is represented by a density operator on the state space. Here a density operator ρ on Hilbert space $\mathcal{H}$ is a positive linear operator such that $\mathrm{tr}(\rho) = 1$. Another equivalent representation of density operator is probabilistic ensemble of pure states. In particular, given an ensemble $\{(p_i, |\psi_i\rangle)\}$ where $p_i \geq 0$, $\sum_i p_i = 1$, and $|\psi_i\rangle$ are pure states, then $\rho = \sum_i p_i [|\psi_i\rangle]$ is a density operator. Here $[|\psi_i\rangle]$ denotes the abbreviation of $|\psi_i\rangle\langle\psi_i|$. Conversely, each density operator can be generated by an ensemble of pure states in this way. The set of density operators on $\mathcal{H}$ can be defined as

$$\mathcal{D}(\mathcal{H}) = \{ \rho \in \mathcal{L}(\mathcal{H}) \; : \; \rho \text{ is positive and } \mathrm{tr}(\rho) = 1\}.$$

The state space of a composite system (for example, a quantum system consisting of many qubits) is the tensor product of the state spaces of its components. For a mixed state ρ on $\mathcal{H}_1 \otimes \mathcal{H}_2$, partial traces of ρ have explicit physical meanings: the density operators $\mathrm{tr}_{\mathcal{H}_1}\rho$ and $\mathrm{tr}_{\mathcal{H}_2}\rho$ are exactly the reduced quantum states of ρ on the second and the first component system, respectively. Note that in general, the state of a composite system cannot be decomposed into tensor product of the reduced states on its component systems. A well-known example is the 2-qubit state

$$|\Psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

which appears repeatedly in our examples of this paper. This kind of state is called *entangled state*. To see the strangeness of entanglement, suppose a measurement $M = \lambda_0[|0\rangle] + \lambda_1[|1\rangle]$ is applied on the first qubit of $|\Psi\rangle$ (see the following for the definition of quantum measurements). Then after the measurement, the second qubit will definitely collapse into state $|0\rangle$ or $|1\rangle$ depending on whether the outcome λ_0 or λ_1 is observed. In other words, the measurement on the first qubit changes the state of the second qubit in some way. This is an outstanding feature of quantum mechanics which has no counterpart in classical world, and is the key to many quantum information processing tasks such as teleportation [2] and super-dense coding [3].

The evolution of a closed quantum system is described by a unitary operator on its state space: if the states of the system at times t_1 and t_2 are ρ_1 and ρ_2, respectively, then $\rho_2 = U\rho_1 U^\dagger$ for some unitary operator U which depends only on t_1 and t_2. In contrast, the general dynamics which can occur in a physical system is described by a trace-preserving super-operator on its state space. Note that the unitary transformation $U(\rho) = U\rho U^\dagger$ is a trace-preserving super-operator.

A quantum *measurement* is described by a collection $\{M_m\}$ of measurement operators, where the indices m refer to the measurement outcomes. It is required that the measurement operators satisfy the completeness equation $\sum_m M_m^\dagger M_m = I_{\mathcal{H}}$. If the system is in state ρ, then the probability that measurement result m occurs is given by

$$p(m) = \mathrm{tr}(M_m^\dagger M_m \rho),$$

and the state of the post-measurement system is $M_m \rho M_m^\dagger / p(m)$.

A particular case of measurement is *projective measurement* which is usually represented by a Hermitian operator. Let M be a Hermitian operator and

$$M = \sum_{m \in spec(M)} mE_m \tag{1}$$

its spectral decomposition. Obviously, the projectors $\{E_m : m \in spec(M)\}$ form a quantum measurement. If the state of a quantum system is ρ, then the probability that result m occurs when measuring M on the system is $p(m) = \mathrm{tr}(E_m\rho)$, and the post-measurement state of the system is $E_m\rho E_m / p(m)$. Note that for each outcome m, the map

$$\mathcal{E}_m(\rho) = E_m \rho E_m$$

is again a super-operator by Kraus Theorem; it is not trace-preserving in general.

Let M be a projective measurement with Eq.(1) its spectral decomposition. We call M non-degenerate if for any $m \in spec(M)$, the corresponding projector E_m is 1-dimensional; that is, all eigenvalues of M are non-degenerate. Non-degenerate measurement is obviously a very special case of general quantum measurement. However, when an ancilla system lying at a fixed state is provided, non-degenerate measurements together with unitary operators are sufficient to implement general measurements.

3 qCCS: Syntax and Semantics

In this section, we review the syntax and semantics of a quantum extension of value-passing CCS, called qCCS, introduced in [7, 20, 8], and the definition of open bisimulation between qCCS processes presented in [6].

3.1 Syntax

We assume three types of data in qCCS: Bool for booleans, real numbers Real for classical data, and qubits Qbt for quantum data. Let $cVar$, ranged over by $x, y, \ldots$, be the set of classical variables, and $qVar$, ranged over by $q, r, \ldots$, the set of quantum variables. It is assumed that $cVar$ and $qVar$ are both countably infinite. We assume a set Exp of classical data expressions over `Real`, which includes $cVar$ as a subset and is ranged over by $e, e', \ldots$, and a set of boolean-valued expressions $BExp$, ranged over by $b, b', \ldots$, with the usual set of boolean operators `tt`, `ff`, $\neg$, $\wedge$, $\vee$, and $\rightarrow$. In particular, we let $e \bowtie e'$ be a boolean expression for any $e, e' \in Exp$ and $\bowtie \in \{>, <, \geq, \leq, =\}$. We further assume that only classical variables can occur free in both data expressions and boolean expressions. Let $cChan$ be the set of classical channel names, ranged over by $c, d, \ldots$, and $qChan$ the set of quantum channel names, ranged over by $\mathsf{c}, \mathsf{d}, \ldots$. Let $Chan = cChan \cup qChan$. A relabeling function f is a one to one function from $Chan$ to $Chan$ such that $f(cChan) \subseteq cChan$ and $f(qChan) \subseteq qChan$.

We often abbreviate the indexed set $\{q_1, \ldots, q_n\}$ to $\widetilde{q}$ when $q_1, \ldots, q_n$ are distinct quantum variables and the dimension n is understood. Sometimes we also use $\widetilde{q}$ to denote the string $q_1 \ldots q_n$. We assume a set of process constant schemes, ranged over by $A, B, \ldots$. Assigned to each process constant scheme A there are two non-negative integers $ar_c(A)$ and $ar_q(A)$. If $\widetilde{x}$ is a tuple of classical variables with $|\widetilde{x}| = ar_c(A)$, and $\widetilde{q}$ a tuple of distinct quantum variables with $|\widetilde{q}| = ar_q(A)$, then $A(\widetilde{x}, \widetilde{q})$ is called a process constant. When $ar_c(A) = ar_q(A) = 0$, we also denote by A the (unique) process constant produced by A.

Based on these notations, the syntax of qCCS terms can be given by the Backus-Naur form as

$$\begin{array}{rcl} t & ::= & \mathbf{nil} \mid A(\widetilde{e}, \widetilde{q}) \mid \alpha.t \mid t + t \mid t \| t \mid t \backslash L \mid t[f] \mid \mathbf{if}\ b\ \mathbf{then}\ t \\ \alpha & ::= & \tau \mid c?x \mid c!e \mid \mathsf{c}?q \mid \mathsf{c}!q \mid \mathcal{E}[\widetilde{q}] \mid M[\widetilde{q}; x] \end{array}$$

where $c \in cChan$, $x \in cVar$, $\mathsf{c} \in qChan$, $q \in qVar$, $\widetilde{q} \subseteq qVar$, $e \in Exp$, $\widetilde{e} \subseteq Exp$, τ is the silent action, $A(\widetilde{x}, \widetilde{q})$ is a process constant, f is a relabeling function, $L \subseteq Chan$, $b \in BExp$, and $\mathcal{E}$ and M are respectively a trace-preserving super-operator and a non-degenerate projective measurement applying on the Hilbert space associated with the systems $\widetilde{q}$. In this paper, we assume all super-operators are completely positive.

To exclude quantum processes which are not physically implementable, we also require $q \notin qv(t)$ in $\mathsf{c}!q.t$ and $qv(t) \cap qv(u) = \emptyset$ in $t \| u$, where for a process term t, $qv(t)$ is the set of its free quantum variables inductively defined as follows:

$$\begin{array}{rclcrcl} qv(\mathbf{nil}) & = & \emptyset & & qv(\tau.t) & = & qv(t) \\ qv(c?x.t) & = & qv(t) & & qv(c!e.t) & = & qv(t) \\ qv(\mathsf{c}?q.t) & = & qv(t) - \{q\} & & qv(\mathsf{c}!q.t) & = & qv(t) \cup \{q\} \\ qv(\mathcal{E}[\widetilde{q}].t) & = & qv(t) \cup \widetilde{q} & & qv(M[\widetilde{q}; x].t) & = & qv(t) \cup \widetilde{q} \\ qv(t + u) & = & qv(t) \cup qv(u) & & qv(t \| u) & = & qv(t) \cup qv(u) \\ qv(t[f]) & = & qv(t) & & qv(t \backslash L) & = & qv(t) \\ qv(\mathbf{if}\ b\ \mathbf{then}\ t) & = & qv(t) & & qv(A(\widetilde{e}, \widetilde{q})) & = & \widetilde{q}. \end{array}$$

The notion of free classical variables in quantum processes, denoted by $fv(\cdot)$, can be defined in the usual way with the only modification that the quantum measurement prefix $M[\widetilde{q}; x]$ has binding

power on x. A quantum process term t is closed if it contains no free classical variables, *i.e.*, $fv(t) = \emptyset$. We let $\mathcal{T}$, ranged over by $t, u, \cdots$, be the set of all qCCS terms, and $\mathcal{P}$, ranged over by $P, Q, \cdots$, the set of closed terms. To complete the definition of qCCS syntax, we assume that for each process constant $A(\widetilde{x}, \widetilde{q})$, there is a defining equation

$$A(\widetilde{x}, \widetilde{q}) \stackrel{def}{=} t$$

such that $fv(t) \subseteq \widetilde{x}$ and $qv(P) \subseteq \widetilde{q}$. Throughout the paper we implicitly assume the convention that process terms are identified up to α-conversion.

The process constructs we give here are quite similar to those in classical CCS, and they also have similar intuitive meanings: **nil** stands for a process which does not perform any action; $c?x$ and $c!e$ are respectively classical input and classical output, while $\mathsf{c}?q$ and $\mathsf{c}!q$ are their quantum counterparts. $\mathcal{E}[\widetilde{q}]$ denotes the action of performing the super-operator $\mathcal{E}$ on the qubits $\widetilde{q}$ while $M[\widetilde{q}; x]$ measures the qubits $\widetilde{q}$ according to M and stores the measurement outcome into the classical variable x. $+$ models nondeterministic choice: $t + u$ behaves like either t or u depending on the choice of the environment. $\|$ denotes the usual parallel composition. The operators $\backslash L$ and $[f]$ model restriction and relabeling, respectively: $t\backslash L$ behaves like t as long as any action through the channels in L is forbidden, and $t[f]$ behaves like t where each channel name is replaced by its image under the relabeling function f. Finally, **if** b **then** t is the standard conditional choice where t can be executed only if b is $\mathtt{tt}$.

An evaluation ψ is a function from $cVar$ to Real; it can be extended in an obvious way to functions from Exp to Real and from $BExp$ to $\{\mathtt{tt}, \mathtt{ff}\}$, and finally, from $\mathcal{T}$ to $\mathcal{P}$. For simplicity, we still use ψ to denote these extensions. Let $\psi\{v/x\}$ be the evaluation which differs from ψ only in that it maps x to v.

3.2 Transitional semantics

For each quantum variable $q \in qVar$, we assume a 2-dimensional Hilbert space $\mathcal{H}_q$ to be the state space of the q-system. For any $S \subseteq qVar$, we denote

$$\mathcal{H}_S = \bigotimes_{q \in S} \mathcal{H}_q.$$

In particular, $\mathcal{H} = \mathcal{H}_{qVar}$ is the state space of the whole environment consisting of all the quantum variables. Note that $\mathcal{H}$ is a countably-infinite dimensional Hilbert space.

Suppose P is a closed quantum process. A pair of the form $\langle P, \rho \rangle$ is called a configuration, where $\rho \in \mathcal{D}(\mathcal{H})$ is a density operator on $\mathcal{H}$. The set of configurations is denoted by Con, and ranged over by $\mathcal{C}, \mathcal{D}, \cdots$. Let

$$Act_c \quad = \quad \{\tau\} \cup \{c?v, c!v \mid c \in cChan, v \in \mathsf{Real}\} \cup \{\mathsf{c}?r, \mathsf{c}!r \mid \mathsf{c} \in qChan, r \in qVar\}.$$

For each $\alpha \in Act_c$, we define the bound quantum variables $qbv(\alpha)$ of α as $qbv(\mathsf{c}?r) = \{r\}$ and $qbv(\alpha) = \emptyset$ if α is not a quantum input. The channel names used in action α is denoted by $cn(\alpha)$; that is, $cn(c?v) = cn(c!v) = \{c\}$, $cn(\mathsf{c}?r) = cn(\mathsf{c}!r) = \{\mathsf{c}\}$, and $cn(\tau) = \emptyset$. We also extend the relabelling function to Act_c in an obvious way.

Let $Dist(Con)$, ranged over by $\mu, \nu, \cdots$, be the set of all finite-supported probabilistic distributions over Con. Then the operational semantics of qCCS can be given by the probabilistic labelled transition system (pLTS) $\langle Con, Act_c, \longmapsto \rangle$, where $\longmapsto \subseteq Con \times Act_c \times Dist(Con)$ is the smallest relation satisfying the inference rules depicted in Fig. 1. The symmetric forms for rules Par_c, $C\text{-}Com_c$, $Q\text{-}Com_c$, and Sum_c are omitted.

In these rules, we abuse the notation slightly by writing $\mathcal{C} \stackrel{\alpha}{\longmapsto} \mathcal{D}$ if $\mathcal{C} \stackrel{\alpha}{\longmapsto} \mu$ where μ is the simple distribution such that $\mu(\mathcal{D}) = 1$. We also use the obvious extension of the function $\|$

$$Tau_c \quad \frac{}{\langle \tau.P, \rho\rangle \overset{\tau}{\longmapsto} \langle P, \rho\rangle}$$

$$C\text{-}Inp_c \quad \frac{v \in \mathsf{Real}}{\langle c?x.t, \rho\rangle \overset{c?v}{\longmapsto} \langle t\{v/x\}, \rho\rangle}$$

$$C\text{-}Out_c \quad \frac{v = [\![e]\!]}{\langle c!e.P, \rho\rangle \overset{c!v}{\longmapsto} \langle P, \rho\rangle}$$

$$Q\text{-}Inp_c \quad \frac{r \notin qv(\mathsf{c}?q.P)}{\langle \mathsf{c}?q.P, \rho\rangle \overset{\mathsf{c}?r}{\longmapsto} \langle P\{r/q\}, \rho\rangle}$$

$$Q\text{-}Out_c \quad \frac{}{\langle \mathsf{c}!q.P, \rho\rangle \overset{\mathsf{c}!q}{\longmapsto} \langle P, \rho\rangle}$$

$$Oper_c \quad \frac{}{\langle \mathcal{E}[\widetilde{r}].P, \rho\rangle \overset{\tau}{\longmapsto} \langle P, \mathcal{E}_{\widetilde{r}}(\rho)\rangle}$$

$$Meas_c \quad \frac{M = \sum_{i\in I} \lambda_i E^i,\ p_i = \mathrm{tr}(E^i_{\widetilde{r}}\rho)}{\langle M[\widetilde{r};x].P, \rho\rangle \overset{\tau}{\longmapsto} \sum_{i\in I} p_i\langle P\{\lambda_i/x\}, E^i_{\widetilde{r}}\rho E^i_{\widetilde{r}}/p_i\rangle}$$

$$Par_c \quad \frac{\langle P_1, \rho\rangle \overset{\alpha}{\longmapsto} \mu,\ qbv(\alpha) \cap qv(P_2) = \emptyset}{\langle P_1 \| P_2, \rho\rangle \overset{\alpha}{\longmapsto} \mu \| P_2}$$

$$C\text{-}Com_c \quad \frac{\langle P_1, \rho\rangle \overset{c?v}{\longmapsto} \langle P_1', \rho\rangle,\ \langle P_2, \rho\rangle \overset{c!v}{\longmapsto} \langle P_2', \rho\rangle}{\langle P_1 \| P_2, \rho\rangle \overset{\tau}{\longmapsto} \langle P_1' \| P_2', \rho\rangle}$$

$$Q\text{-}Com_c \quad \frac{\langle P_1, \rho\rangle \overset{\mathsf{c}?r}{\longmapsto} \langle P_1', \rho\rangle,\quad \langle P_2, \rho\rangle \overset{\mathsf{c}!r}{\longmapsto} \langle P_2', \rho\rangle}{\langle P_1 \| P_2, \rho\rangle \overset{\tau}{\longmapsto} \langle P_1' \| P_2', \rho\rangle}$$

$$Sum_c \quad \frac{\langle P, \rho\rangle \overset{\alpha}{\longmapsto} \mu}{\langle P + Q, \rho\rangle \overset{\alpha}{\longmapsto} \mu}$$

$$Rel_c \quad \frac{\langle P, \rho\rangle \overset{\alpha}{\longmapsto} \mu}{\langle P[f], \rho\rangle \overset{f(\alpha)}{\longmapsto} \mu[f]}$$

$$Cho_c \quad \frac{\langle P, \rho\rangle \overset{\alpha}{\longmapsto} \mu,\ [\![b]\!] = \mathtt{tt}}{\langle \mathbf{if}\ b\ \mathbf{then}\ P, \rho\rangle \overset{\alpha}{\longmapsto} \mu}$$

$$Res_c \quad \frac{\langle P, \rho\rangle \overset{\alpha}{\longmapsto} \mu,\ cn(\alpha) \cap L = \emptyset}{\langle P\backslash L, \rho\rangle \overset{\alpha}{\longmapsto} \mu\backslash L}$$

$$Def_c \quad \frac{\langle t\{\widetilde{v}/\widetilde{x}, \widetilde{r}/\widetilde{q}\}, \rho\rangle \overset{\alpha}{\longmapsto} \mu,\ A(\widetilde{x}, \widetilde{q}) \overset{def}{=} t}{\langle A(\widetilde{v}, \widetilde{r}), \rho\rangle \overset{\alpha}{\longmapsto} \mu}$$

Figure 1: Operational semantics of qCCS

on configurations to distributions. To be precise, if $\mu = \sum_{i\in I} p_i\langle P_i, \rho_i\rangle$ then $\mu\|Q$ denotes the distribution $\sum_{i\in I} p_i\langle P_i\|Q, \rho_i\rangle$. Similar extension applies to $\mu[f]$ and $\mu\backslash L$.

3.3 Open bisimulation

In this subsection, we recall the basic definitions and properties of open bisimulation introduced in [6]. Let $\mathcal{R} \subseteq Con \times Con$ be a relation on configurations. We can lift $\mathcal{R}$ to a relation on $Dist(Con)$ by writing $\mu\mathcal{R}\nu$ if

(1) $\mu = \sum_{i\in I} p_i\mathcal{C}_i$,

(2) for each $i \in I$, $\mathcal{C}_i\mathcal{R}\mathcal{D}_i$ for some $\mathcal{D}_i$, and

(3) $\nu = \sum_{i\in I} p_i\mathcal{D}_i$.

Note that here the set of $\mathcal{C}_i, i \in I$, are not necessarily distinct.

Definition 3.1. *A symmetric relation $\mathcal{R} \subseteq Con \times Con$ is called a (strong) open bisimulation if for any $\langle P, \rho\rangle, \langle Q, \sigma\rangle \in Con$, $\langle P, \rho\rangle\mathcal{R}\langle Q, \sigma\rangle$ implies that*

(1) $qv(P) = qv(Q)$, and $\mathrm{tr}_{qv(P)}(\rho) = \mathrm{tr}_{qv(Q)}(\sigma)$,

(2) for any trace-preserving super-operator $\mathcal{E}$ acting on $\mathcal{H}_{\overline{qv(P)}}$, whenever $\langle P, \mathcal{E}(\rho)\rangle \overset{\alpha}{\longmapsto} \mu$, there exists ν such that $\langle Q, \mathcal{E}(\sigma)\rangle \overset{\alpha}{\longmapsto} \nu$ and $\mu\mathcal{R}\nu$.

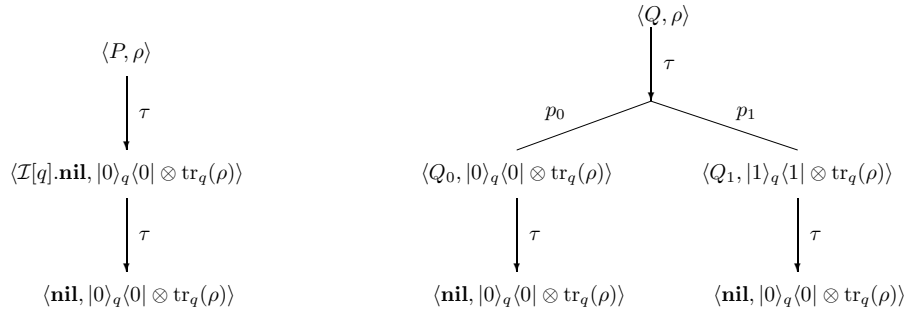


Figure 2: pLTSs for the two ways of setting a quantum system to $|0\rangle$

Definition 3.2. *(1) Two quantum configurations $\langle P, \rho\rangle$ and $\langle Q, \sigma\rangle$ are open bisimilar, denoted by $\langle P, \rho\rangle \dot{\sim} \langle Q, \sigma\rangle$, if there exists an open bisimulation $\mathcal{R}$ such that $\langle P, \rho\rangle \mathcal{R} \langle Q, \sigma\rangle$;*

(2) Two quantum process terms t and u are open bisimilar, denoted by $t \dot{\sim} u$, if for any quantum state $\rho \in \mathcal{D}(\mathcal{H})$ and any evaluation ψ, $\langle t\psi, \rho\rangle \dot{\sim} \langle u\psi, \rho\rangle$.

To illustrate the operational semantics and open bisimulation presented in this section, we give a simple example.

Example 3.3. This example shows two alternative ways of setting a quantum system to the pure state $|0\rangle$. Let $P \stackrel{def}{=} Set^0[q].\mathcal{I}[q].\mathbf{nil}$ and

$$Q \stackrel{def}{=} M_{0,1}[q; x].(\mathbf{if}\ x = 0\ \mathbf{then}\ \mathcal{I}[q].\mathbf{nil}\ +\mathbf{if}\ x = 1\ \mathbf{then}\ \mathcal{X}[q].\mathbf{nil}),$$

where $Set^0 = \{|0\rangle\langle 0|, |0\rangle\langle 1|\}$, $M_{0,1}$ is the 1-qubit measurement according to the computational basis $\{|0\rangle, |1\rangle\}$, $\mathcal{I}$ is the identity super-operator, and $\mathcal{X}$ is the Pauli-X super-operator. For any $\rho \in \mathcal{D}(\mathcal{H})$, the pLTSs rooted by $\langle P, \rho\rangle$ and $\langle Q, \rho\rangle$ respectively are depicted in Fig. 2 where

$$\begin{aligned} Q_0 &= \mathbf{if}\ 0 = 0\ \mathbf{then}\ \mathcal{I}[q].\mathbf{nil}\ +\mathbf{if}\ 0 = 1\ \mathbf{then}\ \mathcal{X}[q].\mathbf{nil}, \\ Q_1 &= \mathbf{if}\ 1 = 0\ \mathbf{then}\ \mathcal{I}[q].\mathbf{nil}\ +\mathbf{if}\ 1 = 1\ \mathbf{then}\ \mathcal{X}[q].\mathbf{nil}, \end{aligned}$$

and $p_i = \mathrm{tr}(|i\rangle_q\langle i|\rho)$. We can show $P \dot{\sim} Q$ by verifying that the relation $\mathcal{R} \cup \mathcal{R}^{-1}$, where

$$\mathcal{R} = \{(\langle P, \rho\rangle, \langle Q, \rho\rangle), (\langle \mathcal{I}[q].\mathbf{nil}, \rho_0\rangle, \langle Q_0, \rho_0\rangle), (\langle \mathcal{I}[q].\mathbf{nil}, \rho_0\rangle, \langle Q_1, \rho_1\rangle), (\langle \mathbf{nil}, \rho_0\rangle, \langle \mathbf{nil}, \rho_0\rangle) : \rho \in \mathcal{D}(\mathcal{H})\}$$

and $\rho_i = |i\rangle_q\langle i| \otimes \mathrm{tr}_q\rho$, is an open bisimulation.

4 Super-operator Valued Distributions

4.1 Semiring of super-operators

We denote by $CP(\mathcal{H})$ the set of super-operators on $\mathcal{H}$, ranged over by $\mathcal{A}, \mathcal{B}, \cdots$. Obviously, both $(CP(\mathcal{H}), 0_{\mathcal{H}}, +)$ and $(CP(\mathcal{H}), \mathcal{I}_{\mathcal{H}}, \circ)$ are monoids, where $\mathcal{I}_{\mathcal{H}}$ and $0_{\mathcal{H}}$ are the identity and null super-operators on $\mathcal{H}$, respectively, and $\circ$ is the composition of super-operators defined by $(\mathcal{A} \circ \mathcal{B})(\rho) = \mathcal{A}(\mathcal{B}(\rho))$ for any $\rho \in \mathcal{D}(\mathcal{H})$. We alway omit the symbol $\circ$ and write $\mathcal{A}\mathcal{B}$ directly for $\mathcal{A} \circ \mathcal{B}$. Furthermore, the operation $\circ$ is (both left and right) distributive with respect to $+$:

$$\mathcal{A}(\mathcal{B}_1 + \mathcal{B}_2) = \mathcal{A}\mathcal{B}_1 + \mathcal{A}\mathcal{B}_2, \quad (\mathcal{B}_1 + \mathcal{B}_2)\mathcal{A} = \mathcal{B}_1\mathcal{A} + \mathcal{B}_2\mathcal{A}.$$

Thus $(CP(\mathcal{H}), +, \circ)$ forms a semiring.

For any $\mathcal{A}, \mathcal{B} \in CP(\mathcal{H})$ and $V \subseteq qVar$, we write $\mathcal{A} \lesssim_V \mathcal{B}$ if for any $\rho \in \mathcal{D}(\mathcal{H})$, $\mathrm{tr}_{\overline{V}}(\mathcal{A}(\rho)) \sqsubseteq \mathrm{tr}_{\overline{V}}(\mathcal{B}(\rho))$, where $\overline{V}$ is the complement set of V in $qVar$, and $\sqsubseteq$ is the Löwner preorder defined on operators such as $A \sqsubseteq B$ if and only if $B - A$ is positive semi-definite. Let $\eqsim_V$ be $\lesssim_V \cap \gtrsim_V$. We usually abbreviate $\lesssim_\emptyset$ and $\eqsim_\emptyset$ to $\lesssim$ and $\eqsim$, respectively. It is easy to check that if $\mathcal{A}$ and $\mathcal{B}$ have Kraus operators $\{A_i : i \in I\}$ and $\{B_j : j \in J\}$ respectively, then $\mathcal{A} \lesssim \mathcal{B}$ if and only if $\sum_{i\in I} A_i^\dagger A_i \sqsubseteq \sum_{j\in J} B_j^\dagger B_j$. The following proposition is direct from definitions:

Proposition 4.1. *Let $\mathcal{A}$ and $\mathcal{B} \in CP(\mathcal{H})$. Then*

(1) $\mathcal{A} \eqsim \mathcal{I}_\mathcal{H}$ if and only if $\mathcal{A}$ is trace-preserving, i.e., $\mathrm{tr}(\mathcal{A}(\rho)) = \mathrm{tr}(\rho)$ for any $\rho \in \mathcal{D}(\mathcal{H})$.

(2) $\mathcal{A} \eqsim 0_\mathcal{H}$ if and only if $\mathcal{A} = 0_\mathcal{H}$.

The next lemma, which is easy from definition, shows that the equivalence relation $\eqsim_V$ is preserved by right application of composition.

Lemma 4.2. *Let $\mathcal{A}, \mathcal{B}, \mathcal{C} \in CP(\mathcal{H})$ and $V \subseteq qVar$. If $\mathcal{A} \eqsim_V \mathcal{B}$, then $\mathcal{AC} \eqsim_V \mathcal{BC}$.*

However, $\eqsim$ is not preserved by composition from the left-hand side. A counter-example is when $\mathcal{A}$ is the X-pauli super-operator, and $\mathcal{C}$ has one single Kraus operator $|0\rangle\langle 0|$. Then $\mathcal{A} \eqsim \mathcal{I}_\mathcal{H}$, but $\mathcal{CA} \neq \mathcal{CI}_\mathcal{H}$ since $\mathrm{tr}(\mathcal{CA}(|0\rangle\langle 0|)) = 0$ while $\mathrm{tr}(\mathcal{CI}_\mathcal{H}(|0\rangle\langle 0|)) = 1$. Nevertheless, we have the following property which is useful for latter discussion.

Lemma 4.3. *Let $\mathcal{A}, \mathcal{B} \in CP(\mathcal{H})$ and $\mathcal{C} \in CP(\mathcal{H}_V)$ where $\emptyset \neq V \subseteq qVar$. If $\mathcal{A} \eqsim_V \mathcal{B}$, then both $\mathcal{AC} \eqsim_V \mathcal{BC}$ and $\mathcal{CA} \eqsim_V \mathcal{CB}$.*

Proof. Easy from the fact that $\mathrm{tr}_{\overline{V}}\mathcal{CA}(\rho) = \mathcal{C}(\mathrm{tr}_{\overline{V}}\mathcal{A}(\rho))$ when $\mathcal{C} \in CP(\mathcal{H}_V)$. □

Let $CP_t(\mathcal{H}) \subseteq CP(\mathcal{H})$ be the set of trace-preserving super-operators, ranged over by $\mathcal{E}, \mathcal{F}, \cdots$. Obviously, $(CP_t(\mathcal{H}), \mathcal{I}_\mathcal{H}, \circ)$ is a sub-monoid of $CP(\mathcal{H})$ while $(CP_t(\mathcal{H}), 0_\mathcal{H}, +)$ is not. It is easy to check that for any $\mathcal{E}, \mathcal{F} \in CP_t(\mathcal{H})$ and $V \subseteq qVar$, $\mathcal{E} \lesssim_V \mathcal{F}$ if and only if $\mathcal{E} \eqsim_V \mathcal{F}$. So for trace-preserving super-operators, we usually use the more symmetric form $\eqsim_V$ instead of $\lesssim_V$.

4.2 Super-operator valued distributions

Let S be a countable set. A super-operator valued distribution, or simply distribution for short, Δ over S is a function from S to $CP(\mathcal{H})$ such that $\sum_{s\in S} \Delta(s) \eqsim \mathcal{I}_\mathcal{H}$. We denote by $\lceil \Delta \rceil$ the support set of Δ, *i.e.*, the set of s such that $\Delta(s) \neq 0_\mathcal{H}$. Let $\mathcal{D}ist_\mathcal{H}(S)$ be the set of finite-support super-operator valued distributions over S; that is,

$$\mathcal{D}ist_\mathcal{H}(S) \quad = \quad \{\Delta : S \to CP(\mathcal{H}) \mid \lceil \Delta \rceil \text{ is finite, and } \sum_{s\in\lceil\Delta\rceil} \Delta(s) \eqsim \mathcal{I}_\mathcal{H}\}.$$

Let Δ, Ξ, etc range over $\mathcal{D}ist_\mathcal{H}(S)$. When Δ is a simple distribution such that $\lceil \Delta \rceil = \{s\}$ for some s and $\Delta(s) = \mathcal{E}$, we abuse the notation slightly to denote Δ by $\mathcal{E} \bullet s$. We further abbreviate $\mathcal{I}_\mathcal{H} \bullet s$ to s. Note that there are infinitely many different simple distributions having the same support $\{s\}$.

Definition 4.4. *Given $\{\Delta_i : i \in I\} \subseteq \mathcal{D}ist_\mathcal{H}(S)$ and $\{\mathcal{A}_i : i \in I\} \subseteq CP(\mathcal{H})$, $\sum_{i\in I} \mathcal{A}_i \eqsim \mathcal{I}_\mathcal{H}$, we define the combination, denoted by $\sum_{i\in I} \mathcal{A}_i \bullet \Delta_i$, to be a new distribution Δ such that*

(1) $\lceil \Delta \rceil = \bigcup\{\lceil \Delta_i \rceil : i \in I, \mathcal{A}_i \neq 0_\mathcal{H}\}$,

(2) for any $s \in \lceil \Delta \rceil$, $\Delta(s) = \sum_{i\in I} \Delta_i(s)\mathcal{A}_i$.

$$Act_s \quad \frac{\gamma = \tau, c?x, c!e, c?q, c!q}{(\!|\gamma.t, \mathcal{E}|\!) \xrightarrow{\mathtt{tt},\gamma} (\!|t, \mathcal{E}|\!)} \qquad Oper_s \quad \frac{}{(\!|\mathcal{F}[\widetilde{q}].t, \mathcal{E}|\!) \xrightarrow{\mathtt{tt},\tau} \mathcal{F}_{\widetilde{q}} \bullet (\!|t, \mathcal{F}_{\widetilde{q}}\mathcal{E}|\!)}$$

$$Meas_s \quad \frac{M = \sum_{i\in I} \lambda_i |\phi_i\rangle\langle\phi_i|}{(\!|M[\widetilde{q}; x].t, \mathcal{E}|\!) \xrightarrow{\mathtt{tt},\tau} \sum_{i\in I} \mathcal{A}^{\phi_i}_{\widetilde{r}} \bullet (\!|t\{\lambda_i/x\}, Set^{\phi_i}_{\widetilde{r}}\mathcal{E}|\!)} \qquad Par_s \quad \frac{(\!|t, \mathcal{E}|\!) \xrightarrow{b,\gamma} \Delta}{(\!|t\|u, \mathcal{E}|\!) \xrightarrow{b,\gamma} \Delta\|u}, \ \begin{array}{l} bv(\gamma) \cap fv(u) = \emptyset \\ qbv(\gamma) \cap qv(u) = \emptyset \end{array}$$

$$C\text{-}Com_s \quad \frac{(\!|t, \mathcal{E}|\!) \xrightarrow{b_1,c?x} (\!|t', \mathcal{E}|\!), \quad (\!|u, \mathcal{E}|\!) \xrightarrow{b_2,c!e} (\!|u', \mathcal{E}|\!)}{(\!|t\|u, \mathcal{E}|\!) \xrightarrow{b_1\wedge b_2,\tau} (\!|t'\{e/x\}\|u', \mathcal{E}|\!)} \qquad Q\text{-}Com_s \quad \frac{(\!|t, \mathcal{E}|\!) \xrightarrow{b_1,c?q} (\!|t', \mathcal{E}|\!), \quad (\!|u, \mathcal{E}|\!) \xrightarrow{b_2,c!q} (\!|u', \mathcal{E}|\!)}{(\!|t\|u, \mathcal{E}|\!) \xrightarrow{b_1\wedge b_2,\tau} (\!|t'\|u', \mathcal{E}|\!)}$$

$$Sum_s \quad \frac{(\!|t, \mathcal{E}|\!) \xrightarrow{b,\gamma} \Delta}{(\!|t + u, \mathcal{E}|\!) \xrightarrow{b,\gamma} \Delta} \qquad Rel_s \quad \frac{(\!|t, \mathcal{E}|\!) \xrightarrow{b,\gamma} \Delta}{(\!|t[f], \mathcal{E}|\!) \xrightarrow{b,f(\gamma)} \Delta[f]}$$

$$Cho_s \quad \frac{(\!|t, \mathcal{E}|\!) \xrightarrow{b',\gamma} \Delta, \quad bv(\gamma) \cap fv(b) = \emptyset}{(\!|\textbf{if } b \textbf{ then } t, \mathcal{E}|\!) \xrightarrow{b\wedge b',\gamma} \Delta} \qquad Res_s \quad \frac{(\!|t, \mathcal{E}|\!) \xrightarrow{b,\gamma} \Delta, \quad cn(\gamma) \cap L = \emptyset}{(\!|t\backslash L, \mathcal{E}|\!) \xrightarrow{b,\gamma} \Delta\backslash L}$$

$$Def_s \quad \frac{(\!|t\{\widetilde{e}/\widetilde{x}, \widetilde{r}/\widetilde{q}\}, \mathcal{E}|\!) \xrightarrow{b,\gamma} \Delta, \ A(\widetilde{x}, \widetilde{q}) \stackrel{def}{=} t}{(\!|A(\widetilde{e}, \widetilde{r}), \mathcal{E}|\!) \xrightarrow{b,\gamma} \Delta}$$

Figure 3: Symbolic operational semantics of qCCS

Here and in the following of this paper, the index sets I, J, K, etc are all assumed to be finite. By Lemma 4.2, it is easy to check that the above definition is well-defined. Furthermore, since $\eqsim$ is not preserved by left applications of composition, we cannot require $\Delta(s) = \sum_{i\in I} \mathcal{A}_i\Delta_i(s)$ in the second clause, although it seems more natural. As a result, say, $\mathcal{E} \bullet (\mathcal{F} \bullet s) = \mathcal{F}\mathcal{E} \bullet s$ but not $\mathcal{E}\mathcal{F} \bullet s$.

Probability distributions can be regarded as special super-operator valued distributions by requiring that all super-operators appeared in the definitions above have the form $p\mathcal{I}_{\mathcal{H}}$ where $0 \leq p \leq 1$. Since in this case all super-operators commute, we always omit the bullet $\bullet$ in the expressions.

5 Symbolic bisimulation

5.1 Super-operator weighted transition systems

We now extend the ordinary probabilistic labelled transition systems to super-operator weighted ones.

Definition 5.1. *A super-operator weighted labelled transition system, or quantum labelled transition system (qLTS), is a triple $(S, Act, \longrightarrow)$, where*

(1) S is a countable set of states,

(2) Act is a countable set of transition actions,

(3) $\longrightarrow$, called transition relation, is a subset of $S \times Act \times \mathcal{D}ist_{\mathcal{H}}(S)$.

For simplicity, we write $s \xrightarrow{\alpha} \Delta$ instead of $(s, \alpha, \Delta) \in \longrightarrow$. A pLTS may be viewed as a degenerate qLTS in which all super-operator valued distributions are probabilistic ones.

5.2 Symbolic transitional semantics of qCCS

To present the symbolic operational semantics of quantum processes, we need some more notations. Let

$$Act_s \quad = \quad \{\tau\} \cup \{c?x, c!e \mid c \in cChan, x \in cVar, e \in Exp\} \cup \{\mathsf{c}?r, \mathsf{c}!r \mid \mathsf{c} \in qChan, r \in qVar\}$$

and $BAct_s = BExp \times Act_s$. For each $\gamma \in Act_s$, the notion $qbv(\gamma)$ for bound quantum variables, $cn(\gamma)$ for channel names, and $fv(\gamma)$ for free classical variables are similarly defined as for Act_c. We also define $bv(\gamma)$, the set of bound classical variables in γ in an obvious way.

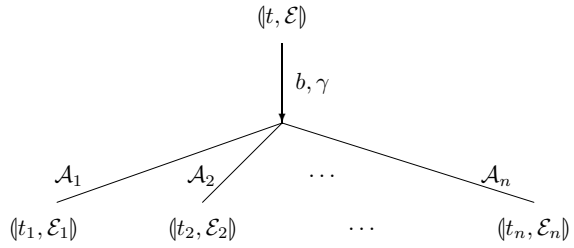
A pair of the form $(\!|t, \mathcal{E}|\!)$, where $t \in \mathcal{T}$ and $\mathcal{E} \in CP_t(\mathcal{H})$, is called a snapshot, and the set of snapshots is denoted by SN. Then the symbolic semantics of qCCS is given by the qLTS $(SN, BAct_s, \longrightarrow)$ on snapshots, where $\longrightarrow \subseteq SN \times BAct_s \times \mathcal{D}ist_{\mathcal{H}}(SN)$ is the smallest relation satisfying the rules defined in Fig. 3. In Rule $Meas_s$, for each $i \in I$, $\mathcal{A}^{\phi_i}_{\widetilde{r}} \in CP(\mathcal{H})$ and $Set^{\phi_i}_{\widetilde{r}} \in CP_t(\mathcal{H})$ are defined respectively as

$$\mathcal{A}^{\phi_i}_{\widetilde{r}} \quad : \quad \rho \mapsto |\phi_i\rangle_{\widetilde{r}}\langle\phi_i|\rho|\phi_i\rangle_{\widetilde{r}}\langle\phi_i| \tag{2}$$

$$Set^{\phi_i}_{\widetilde{r}} \quad : \quad \rho \mapsto \sum_{j\in I} |\phi_i\rangle_{\widetilde{r}}\langle\phi_j|\rho|\phi_j\rangle_{\widetilde{r}}\langle\phi_i|. \tag{3}$$

The symmetric forms for rules Par_s, $C\text{-}Com_s$, $Q\text{-}Com_s$, and Sum_s are omitted. Here again, the functions $\|$, $[f]$, and $\backslash L$ have been extended to super-operator valued distributions by denoting, say, $\Delta \| u$ the super-operator valued distribution $\sum_{i\in I} \mathcal{A}_i \bullet (\!|t_i \| u, \mathcal{E}_i|\!)$, if $\Delta = \sum_{i\in I} \mathcal{A}_i \bullet (\!|t_i, \mathcal{E}_i|\!)$.

The transition graph of a snapshot is depicted as usual where each transition $(\!|t, \mathcal{E}|\!) \xrightarrow{b,\gamma} \sum_{i=1}^{n} \mathcal{A}_i \bullet (\!|t_i, \mathcal{E}_i|\!)$ is depicted as



We sometimes omit the line marked with $\mathcal{I}_{\mathcal{H}}$ for simplicity.

Example 5.2. (Example 3.3 revisited) For the first example, we revisit the two ways of setting a quantum system to pure state $|0\rangle$, presented in Example 3.3. According to the symbolic operational semantics presented in Fig. 3, the qLTSs rooted by $(\!|P, \mathcal{I}_{\mathcal{H}}|\!)$ and $(\!|Q, \mathcal{I}_{\mathcal{H}}|\!)$ respectively can be depicted as in Fig. 4, where $\mathcal{A}_i$ has the single Kraus operator $|i\rangle_q\langle i|$ for $i = 0, 1$.

At the first glance, it is tempting to think that symbolic semantics provides no advantage in describing quantum processes, as the qLTSs in Fig. 4 are almost the same as the pLTSs in Fig. 2 (Indeed, the right-hand side qLTS in the former is even more complicated than the corresponding pLTS in the latter). However, pLTSs in Fig. 2 are depicted for a fixed quantum state ρ; to characterise the behaviours of a quantum process, infinitely many such pLTSs must be given, although typically they share the same structure. On the other hand, the qLTSs in Fig. 4 specify *all* possible behaviours of the processes, by means of the super-operators they can perform.

Example 5.3. This example shows the correctness of super-dense coding protocol. Let $M = \sum_{i=0}^{3} i|\widetilde{i}\rangle\langle\widetilde{i}|$ be a 2-qubit measurement where $\widetilde{i}$ is the binary expansion of i. Let $\mathcal{CN}$ be the controlled-not operation and $\mathcal{H}$ Hadamard operation. Then the quantum processes participating in super-dense

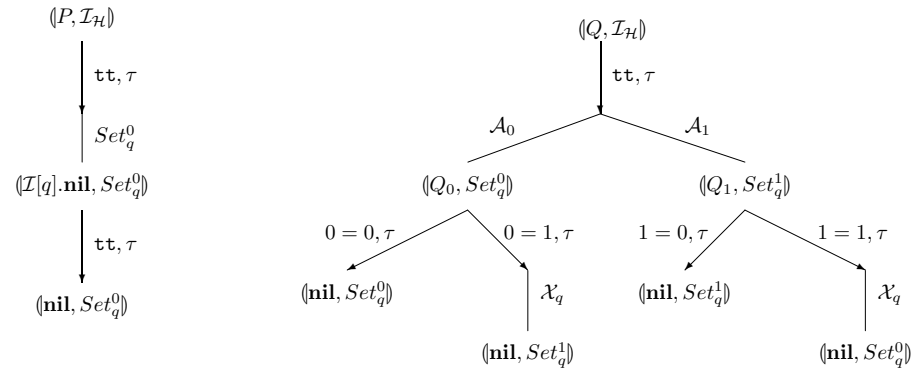


Figure 4: qLTSs for two ways of setting a quantum system to $|0\rangle$

coding protocol can be defined as follows:

$$\begin{aligned}
Alice &\stackrel{def}{=} \mathsf{c}_A?q_1.\sum_{0\le i\le 3}\left(\mathbf{if}\ x=i\ \mathbf{then}\ \sigma^i[q_1].\mathsf{e}!q_1.\mathbf{nil}\right),\\
Bob &\stackrel{def}{=} \mathsf{c}_B?q_2.\mathsf{e}?q_1.\mathcal{CN}[q_1,q_2].\mathcal{H}[q_1].M[q_1,q_2;x].d!x.\mathbf{nil},\\
EPR &\stackrel{def}{=} Set^\Psi[q_1,q_2].\mathsf{c}_B!q_2.\mathsf{c}_A!q_1.\mathbf{nil},\\
Sdc &\stackrel{def}{=} c?x.(EPR\|Alice\|Bob)\backslash\{\mathsf{c}_A,\mathsf{c}_B,\mathsf{e}\}.
\end{aligned}$$

The specification of super-dense coding protocol can be defined as:

$$Sdc_{spec} \stackrel{def}{=} c?x.\tau^7.Set^x[q_1,q_2].d!x.\mathbf{nil}$$

where

$$Set^x[q_1,q_2].d!x.\mathbf{nil} = \sum_{i=0}^{3}(\mathbf{if}\ x=i\ \mathbf{then}\ Set^i[q_1,q_2].d!x.\mathbf{nil}).$$

Here Set^i and Set^Ψ are the 2-qubit super-operators which set the target qubits to $|\widetilde{i}\rangle$ and $|\Psi\rangle = (|00\rangle + |11\rangle)/\sqrt{2}$, respectively. We insert seven τ's in the specification to match the internal actions of Sdc. The qLTSs rooted from $(\!|Sdc_{spec}, \mathcal{I}_\mathcal{H}|\!)$ and $(\!|Sdc, \mathcal{I}_\mathcal{H}|\!)$ respectively are depicted in Fig. 5 where $\widetilde{q} = \{q_1, q_2\}$, $\mathcal{A}_{\widetilde{i}}$ is the super-operator with the single Kraus operator $|\widetilde{i}\rangle\langle\widetilde{i}|$, $L = \{\mathsf{c}_A, \mathsf{c}_B, \mathsf{e}\}$,

$$Sdc^x \quad = \quad \left(\left(\sum_{i=0}^{3}(\mathbf{if}\ x=i\ \mathbf{then}\ \sigma^i[q_1].\mathsf{e}!q_1.\mathbf{nil})\right)\|Bob\right)\backslash\{\mathsf{e}\},$$

and for simplicity, we only draw the transitions along the $x = 0$ branch.

To conclude this subsection, we prove some useful properties of symbolic transitions.

Lemma 5.4. *If $(\!|t, \mathcal{E}|\!) \xrightarrow{b,\gamma} \Delta$, then there exist super-operators $\{\mathcal{B}_i : i \in I\} \subseteq CP(\mathcal{H})$ and $\{\mathcal{F}_i : i \in I\} \subseteq CP_t(\mathcal{H})$, and process terms $\{t_i : i \in I\} \subseteq \mathcal{T}$ such that*

(1) $\sum_{i\in I}\mathcal{B}_i \eqsim \mathcal{I}_\mathcal{H}$,

(2) $\Delta = \sum_{i\in I}\mathcal{B}_i \bullet (\!|t_i, \mathcal{F}_i\mathcal{E}|\!)$,

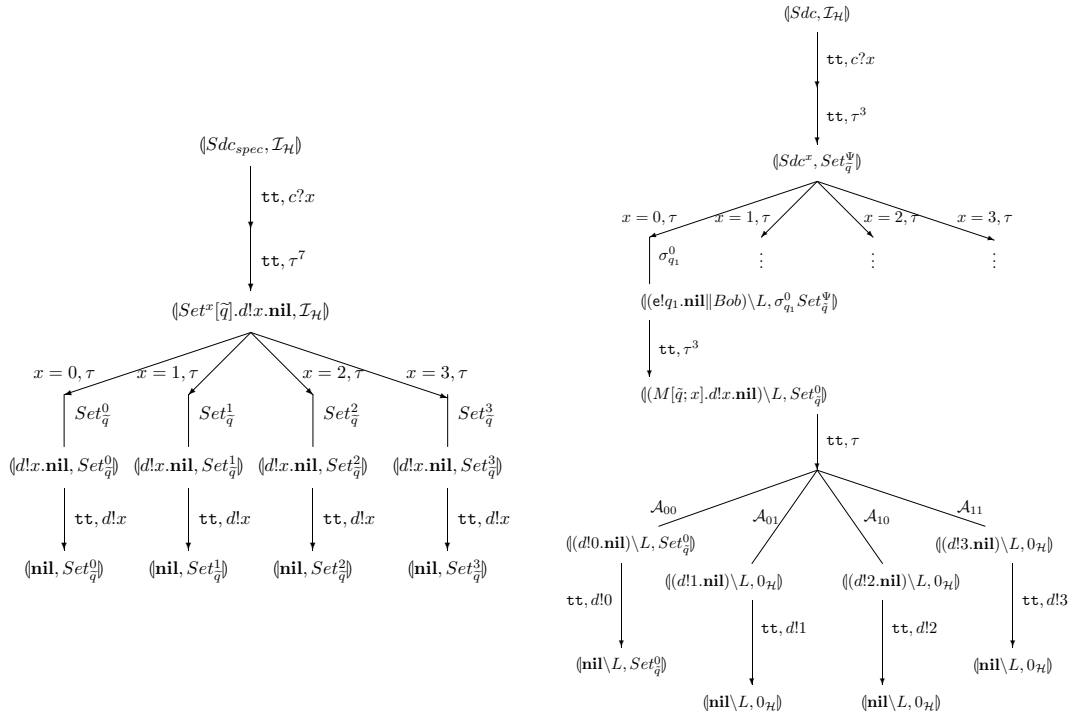


Figure 5: qLTSs for $(\!|Sdc_{spec}, \mathcal{I}_{\mathcal{H}}|\!)$ and $(\!|Sdc, \mathcal{I}_{\mathcal{H}}|\!)$

(3) for any $\mathcal{G} \in CP_t(\mathcal{H})$, $(\!|t, \mathcal{G}|\!) \xrightarrow{b,\gamma} \sum_{i\in I} \mathcal{B}_i \bullet (\!|t_i, \mathcal{F}_i\mathcal{G}|\!)$.

Especially, if $|I| > 1$ *then* $\mathcal{B}_i$ *and* $\mathcal{F}_i$ *take the forms as* $\mathcal{A}^{\phi_i}_{\tilde{r}}$ *and* $Set^{\phi_i}_{\tilde{r}}$ *in Eqs.(2) and (3), respectively.*

Proof. Easy from the definition of inference rules. □

The following lemmas show the relationship between transitions in ordinary semantics and in symbolic semantics. Let ψ be an evaluation, $\alpha \in Act_c$, and $\gamma \in Act_s$. We write $\alpha =_\psi \gamma$ if either $\alpha = c!v$, $\gamma = c!e$, and $\psi(e) = v$, or $\gamma = \alpha$ if neither of them is a classical output.

Lemma 5.5. *Suppose* $\langle t\psi, \rho\rangle \overset{\alpha}{\longmapsto} \mu$. *Then there exist* b, I, ψ', $\{\mathcal{A}_i : i \in I\} \subseteq CP(\mathcal{H})$, $\{\mathcal{E}_i : i \in I\} \subseteq CP_t(\mathcal{H})$, *and* $\{t_i : i \in I\} \subseteq \mathcal{T}$, *such that* $\sum_{i\in I} \mathcal{A}_i \approx \mathcal{I}_{\mathcal{H}}$, *and*

(1) $\psi(b) = \mathtt{tt}$,

(2) $\mu = \sum_{i\in I} \mathrm{tr}(\mathcal{A}_i(\rho))\langle t_i\psi', \mathcal{E}_i(\rho)\rangle$,

(3) for any $\mathcal{E} \in CP_t(\mathcal{H})$, $(\!|t, \mathcal{E}|\!) \xrightarrow{b,\gamma} \sum_{i\in I} \mathcal{A}_i \bullet (\!|t_i, \mathcal{E}_i\mathcal{E}|\!)$, *where*

- *(a) if* $\alpha = c?v$ *then* $\gamma = c?x$ *for some* $x \notin fv(t)$, *and* $\psi' = \psi\{v/x\}$,
- *(b) otherwise,* $\gamma =_\psi \alpha$ *and* $\psi' = \psi$.

Proof. We prove by induction on the depth of the inference by which the action $\langle t\psi, \rho\rangle \overset{\alpha}{\longmapsto} \mu$ is inferred. We argue by cases on the form of t.

(1) $t = c?x.t'$. Then $t\psi = c?x.u$ where u is the process term obtained from t' by instantiating all the free variables in $fv(t') - \{x\}$ according to ψ. By Rule *C-Inp*$_c$ we deduce that $\alpha = c?v$ for some $v \in \mathsf{Real}$ and $\mu = \langle P, \rho\rangle$ where $P = u\{v/x\} = t'\psi\{v/x\}$. By Rule *Act*$_s$, for any $\mathcal{E} \in CP_t(\mathcal{H})$, we have $(\!|t, \mathcal{E}|\!) \xrightarrow{\mathtt{tt}, c?x} (\!|t', \mathcal{E}|\!)$. So we need only to take $b = \mathtt{tt}$, $|I| = 1$, $t_i = t'$, $\mathcal{A}_i = \mathcal{E}_i = \mathcal{I}_{\mathcal{H}}$.

(2) $t = c!e.t'$. Then $t\psi = c!\psi(e).(t'\psi)$, and by Rule *C-Out*$_c$ we deduce that $\alpha = c!\psi(e)$ and $\mu = \langle t'\psi, \rho\rangle$. By Rule *Act*$_s$, for any $\mathcal{E} \in CP_t(\mathcal{H})$, we have $(\!|t, \mathcal{E}|\!) \xrightarrow{\mathtt{tt}, c!e} (\!|t', \mathcal{E}|\!)$. So we need only to take $b = \mathtt{tt}$, $|I| = 1$, $t_i = t'$, $\mathcal{A}_i = \mathcal{E}_i = \mathcal{I}_{\mathcal{H}}$ as well.

(3) $t = \mathtt{c}?q.t'$. Then $t\psi = \mathtt{c}?q.(t'\psi)$, and by Rule *Q-Inp*$_c$ we deduce that $\alpha = \mathtt{c}?r$ for some $r \notin qv(t)$ and $\mu = \langle (t'\psi)\{r/q\}, \rho\rangle$. By Rule *Act*$_s$ and α-conversion, for any $\mathcal{E} \in CP_t(\mathcal{H})$, we have $(\!|t, \mathcal{E}|\!) \xrightarrow{\mathtt{tt}, \mathtt{c}?r} (\!|t'\{r/q\}, \mathcal{E}|\!)$. So we need only to take $b = \mathtt{tt}$, $|I| = 1$, $t_i = t'\{r/q\}$, $\mathcal{A}_i = \mathcal{E}_i = \mathcal{I}_{\mathcal{H}}$.

(4) $t = M[\widetilde{q}; x].t'$. Then $t\psi = M[\widetilde{q}; x].u$ where u is the process term obtained from t' by instantiating all the free variables in $fv(t') - \{x\}$ according to ψ. Let $M = \sum_{i\in I} \lambda_i |\phi_i\rangle\langle\phi_i|$. By Rule *Meas*$_c$ we deduce that $\alpha = \tau$ and $\mu = \sum_{i\in I} \mathrm{tr}(\mathcal{A}_i(\rho))\langle P_i, \mathcal{E}_i(\rho)\rangle$ where $P_i = u\{\lambda_i/x\} = t'\{\lambda_i/x\}\psi$, $\mathcal{A}_i = \{|\phi_i\rangle\langle\phi_i|\}$, and $\mathcal{E}_i = \{|\phi_i\rangle\langle\phi_j| : j \in I\}$. Take $b = \mathtt{tt}$. By Rule *Meas*$_s$, for any $\mathcal{E} \in CP_t(\mathcal{H})$, we have $(\!|t, \mathcal{E}|\!) \xrightarrow{b,\tau} \sum_{i\in I} \mathcal{A}_i \bullet (\!|t'\{\lambda_i/x\}, \mathcal{E}_i\mathcal{E}|\!)$.

(5) $t = t_1 \| t_2$. Then $t\psi = t_1\psi \| t_2\psi$. There are two sub-cases to consider:

  (a) The action is caused by one of the components, say $\langle t_1\psi, \rho\rangle \xrightarrow{\alpha} \mu_1$. Then we have $qbv(\alpha) \cap qv(t_2\psi) = \emptyset$, and $\mu = \mu_1 \| t_2\psi$. By induction, there exist b, I, t_i, $\mathcal{A}_i, \mathcal{E}_i$, $i \in I$, such that $\psi(b) = \mathtt{tt}$, $\mu_1 = \sum_{i\in I} \mathrm{tr}(\mathcal{A}_i(\rho))\langle t_i\psi', \mathcal{E}_i(\rho)\rangle$, and for any $\mathcal{E} \in CP_t(\mathcal{H})$, $(\!|t_1, \mathcal{E}|\!) \xrightarrow{b,\gamma} \sum_{i\in I} \mathcal{A}_i \bullet (\!|t_i, \mathcal{E}_i\mathcal{E}|\!)$. Note that by α-conversion, when $\gamma = c?x$, we can always take x such that $x \notin fv(t_2)$, and consequently, $(t_i \| t_2)\psi' = t_i\psi' \| t_2\psi$. Finally, we have $(\!|t, \mathcal{E}|\!) \xrightarrow{b,\gamma} \sum_{i\in I} \mathcal{A}_i \bullet (\!|t_i \| t_2, \mathcal{E}_i\mathcal{E}|\!)$, using Rule *Par*$_s$.

  (b) The action is caused by a (classical or quantum) communication. Here we only detail the case when $\langle t_1\psi, \rho\rangle \xrightarrow{c?v} \langle P_1, \rho\rangle$, $\langle t_2\psi, \rho\rangle \xrightarrow{c!v} \langle P_2, \rho\rangle$, $\alpha = \tau$, and $\mu = \langle P_1 \| P_2, \rho\rangle$. Then by induction, there exist b_1, b_2, t_1', t_2' such that $\psi(b_1 \wedge b_2) = \mathtt{tt}$, $P_1 = t_1'\psi'$, $P_2 = t_2'\psi$, and for any $\mathcal{E} \in CP_t(\mathcal{H})$, $(\!|t_1, \mathcal{E}|\!) \xrightarrow{b_1, c?x} (\!|t_1', \mathcal{E}|\!)$ and $(\!|t_2, \mathcal{E}|\!) \xrightarrow{b_2, c!e} (\!|t_2', \mathcal{E}|\!)$, where $x \notin fv(t_1)$, $\psi' = \psi\{v/x\}$, and $\psi(e) = v$. Thus

$$(t_1'\{e/x\} \| t_2')\psi = t_1'\{e/x\}\psi \| t_2'\psi = t_1'\psi\{v/x\} \| t_2'\psi = t_1'\psi' \| t_2'\psi = P_1 \| P_2.$$

  Finally, we have $(\!|t, \mathcal{E}|\!) \xrightarrow{b_1 \wedge b_2, \tau} (\!|t_1'\{e/x\} \| t_2', \mathcal{E}|\!)$, using Rule *Q-Com*$_s$.

(6) Other cases. Similar to the cases we discussed above. □

Lemma 5.6. *Suppose* $(\!|t, \mathcal{E}|\!) \xrightarrow{b,\gamma} \Delta$. *Then there exist* I, $\{\mathcal{A}_i : i \in I\} \subseteq CP(\mathcal{H})$, $\{\mathcal{E}_i : i \in I\} \subseteq CP_t(\mathcal{H})$, *and* $\{t_i : i \in I\} \subseteq \mathcal{T}$, *such that* $\sum_{i\in I} \mathcal{A}_i \approx \mathcal{I}_{\mathcal{H}}$, *and*

(1) $\Delta = \sum_{i\in I} \mathcal{A}_i \bullet (\!|t_i, \mathcal{E}_i\mathcal{E}|\!)$,

(2) for any ψ *and* ρ, $\psi(b) = \mathtt{tt}$ *implies* $\langle t\psi, \rho\rangle \xrightarrow{\alpha} \sum_{i\in I} \mathrm{tr}(\mathcal{A}_i(\rho))\langle t_i\psi', \mathcal{E}_i(\rho)\rangle$ *where*

  (a) if $\gamma = c?x$ *then* $\alpha = c?v$ *for some* $v \in \mathsf{Real}$, *and* $\psi' = \psi\{v/x\}$,

  (b) otherwise, $\gamma =_\psi \alpha$ *and* $\psi' = \psi$.

Proof. Similar to Lemma 5.5. □

5.3 Symbolic bisimulation

Let $\mathcal{S} \subseteq SN \times SN$ be an equivalence relation. We lift $\mathcal{S}$ to $\mathcal{D}ist_{\mathcal{H}}(SN) \times \mathcal{D}ist_{\mathcal{H}}(SN)$ by defining $\Delta \mathcal{S} \Xi$ if for any equivalence class $T \in SN/\mathcal{S}$, $\Delta(T) \eqsim \Xi(T)$; that is, $\sum_{(\!|t,\mathcal{E}|\!) \in T} \Delta((\!|t,\mathcal{E}|\!)) \eqsim \sum_{(\!|t,\mathcal{E}|\!) \in T} \Xi((\!|t,\mathcal{E}|\!))$. We write $\gamma =_b \gamma'$ if either $\gamma = c!e$, $\gamma' = c!e'$, and $b \to e = e'$, or $\gamma = \gamma'$ if neither of them is a classical output.

Definition 5.7. *Let $\mathfrak{S} = \{\mathcal{S}^b : b \in BExp\}$ be a family of equivalence relations on SN. $\mathfrak{S}$ is called a symbolic (open) bisimulation if for any $b \in BExp$, $(\!|t,\mathcal{E}|\!)\mathcal{S}^b(\!|u,\mathcal{F}|\!)$ implies that*

(1) $qv(t) = qv(u)$ and $\mathcal{E} \eqsim_{\overline{qv(t)}} \mathcal{F}$, if b is satisfiable;

(2) for any $\mathcal{G} \in CP_t(\mathcal{H}_{\overline{qv(t)}})$, whenever $(\!|t,\mathcal{G}\mathcal{E}|\!) \xrightarrow{b_1,\gamma} \Delta$ with $bv(\gamma) \cap fv(b,t,u) = \emptyset$, then there exists a collection of booleans B such that $b \wedge b_1 \to \bigvee B$ and $\forall\, b' \in B$, $\exists b_2, \gamma'$ with $b' \to b_2$, $\gamma =_{b'} \gamma'$, $(\!|u,\mathcal{G}\mathcal{F}|\!) \xrightarrow{b_2,\gamma'} \Xi$, and $(\mathcal{G}\mathcal{E} \bullet \Delta)\mathcal{S}^{b'}(\mathcal{G}\mathcal{F} \bullet \Xi)$.

Two configurations $(\!|t,\mathcal{E}|\!)$ and $(\!|u,\mathcal{F}|\!)$ are symbolically b-bisimilar, denoted by $(\!|t,\mathcal{E}|\!) \sim^b (\!|u,\mathcal{F}|\!)$, if there exists a symbolic bisimulation $\mathfrak{S} = \{\mathcal{S}^b : b \in BExp\}$ such that $(\!|t,\mathcal{E}|\!)\mathcal{S}^b(\!|u,\mathcal{F}|\!)$. Two quantum process terms t and u are symbolically b-bisimilar, denoted by $t \sim^b u$, if $(\!|t,\mathcal{I}_{\mathcal{H}}|\!) \sim^b (\!|u,\mathcal{I}_{\mathcal{H}}|\!)$. When $b = \mathtt{tt}$, we simply write $t \sim u$.

To show the usage of symbolic bisimulation, we revisit the examples presented in Section 5.2 to show that the proposed protocols indeed achieve the desired goals. Let $\widetilde{A} = \{A_i : i \in I\}$ be a set of disjoint subsets of snapshots. An equivalence relation $\mathcal{S}$ is said to be generated by $\widetilde{A}$ if its equivalence classes on the set of snapshots $\cup_{i\in I} A_i$ are given by the partition $\widetilde{A}$, and it is the identity relation on $SN - \cup_{i\in I} A_i$.

Example 5.8. (Example 5.2 revisited) This example is devoted to showing rigorously that the two ways of setting a quantum system to the pure state $|0\rangle$, presented in Examples 3.3 and 5.2, are indeed bisimilar. Let

$$\begin{aligned} A &= \{(\!|P,\mathcal{I}_{\mathcal{H}}|\!), (\!|Q,\mathcal{I}_{\mathcal{H}}|\!)\}, \\ B &= \{(\!|\mathcal{I}[q].\mathbf{nil}, Set_q^0|\!), (\!|Q_0, Set_q^0|\!), (\!|Q_1, Set_q^1|\!)\} \end{aligned}$$

and $\mathcal{S}'$ be the equivalence relation generated by $\{A, B\}$. It is easy to check that the family $\{\mathcal{S}^b : b \in BExp\}$, where $\mathcal{S}^b = \mathcal{S}'$ for any $b \in BExp$, is a symbolic bisimulation. Thus $P \sim Q$.

Example 5.9. (Superdense coding revisited) This example is devoted to proving rigorously that the protocol presented in Example 5.3 indeed sends two bits of classical information from Alice to Bob by transmitting a qubit. For that purpose, we need to show that $(\!|Sdc_{spec},\mathcal{I}_{\mathcal{H}}|\!) \sim^{\mathtt{tt}} (\!|Sdc,\mathcal{I}_{\mathcal{H}}|\!)$. Indeed, let

$$\begin{aligned} A &= \{(\!|Sdc_{spec},\mathcal{I}_{\mathcal{H}}|\!), (\!|Sdc,\mathcal{I}_{\mathcal{H}}|\!)\}, \\ B^j &= \{(\!|t,\mathcal{E}|\!) : d((\!|t,\mathcal{E}|\!)) = j\}, \\ C_i^k &= \{(\!|t,\mathcal{E}|\!) : (\!|t,\mathcal{E}|\!) \text{ along the branch of } x = i, \text{ and } d((\!|t,\mathcal{E}|\!)) = k\}, \end{aligned}$$

where $d((\!|t,\mathcal{E}|\!))$ is the depth of the node $(\!|t,\mathcal{E}|\!)$ from the root of its corresponding qLTS, $0 < j \leq 4$, $0 \leq i \leq 3$, and $5 \leq k \leq 10$. Let $\mathcal{S}_1^{\mathtt{tt}}$ be the equivalence relation generated by $\{A, B^1, B^2, B^3, B^4\}$, and $\mathcal{S}_1^{x=i}$ generated by $\{C_i^k : 5 \leq k \leq 10\}$. For any $b \in BExp$, let $\mathcal{S}^b$ be $\mathcal{S}_1^{x=i}$ if $b \leftrightarrow x = i$, $\mathcal{S}_1^{\mathtt{tt}}$ if $b \leftrightarrow \mathtt{tt}$, and Id_{SN} otherwise. Then it is easy to check that $\mathfrak{S} = \{\mathcal{S}^b : b \in BExp\}$ is a symbolic bisimulation.

In the following, we denote by $\mathcal{S}^*$ the equivalence closure of a relation $\mathcal{S}$.

Definition 5.10. *A relation family $\mathfrak{S} = \{\mathcal{S}^b : b \in BExp\}$ is called decreasing, if for any $b, b' \in BExp$ with $b \to b'$, we have $\mathcal{S}^{b'} \subseteq \mathcal{S}^b$.*

Lemma 5.11. *Let $\mathfrak{S} = \{\mathcal{S}^b : b \in BExp\}$ be a symbolic bisimulation. Then there exists a decreasing symbolic bisimulation $\mathfrak{U} = \{\mathcal{U}^b : b \in BExp\}$ such that for each $b \in BExp$, $\mathcal{S}^b \subseteq \mathcal{U}^b$.*

Proof. Suppose $\mathfrak{S} = \{\mathcal{S}^b : b \in BExp\}$ is a symbolic bisimulation. For each $b \in BExp$, let

$$\mathcal{U}_1^b = \bigcup\{\mathcal{S}^{b'} : b \to b'\} \text{ and } \mathcal{U}^b = (\mathcal{U}_1^b)^*.$$

Obviously, $\mathfrak{U} = \{\mathcal{U}^b : b \in BExp\}$ is decreasing. We have to show that $\mathfrak{U}$ is a symbolic bisimulation.

Let $b \in BExp$ and $(\!|t, \mathcal{E}|\!)\mathcal{U}^b(\!|u, \mathcal{F}|\!)$. Note that $\mathcal{U}_1^b$ is both reflexive and symmetric. So $\mathcal{U}^b$ is actually the transitive closure of $\mathcal{U}_1^b$, and there exist $n \geq 1$ and a sequence of snapshots $(\!|t_i, \mathcal{E}_i|\!)$, $0 \leq i \leq n$, such that $(\!|t, \mathcal{E}|\!) = (\!|t_0, \mathcal{E}_0|\!)$, $(\!|u, \mathcal{F}|\!) = (\!|t_n, \mathcal{E}_n|\!)$, and for each $0 \leq i \leq n-1$, $(\!|t_i, \mathcal{E}_i|\!)\mathcal{U}_1^b(\!|t_{i+1}, \mathcal{E}_{i+1}|\!)$. For the sake of simplicity, we assume $n = 2$. That is, there exists $(\!|s, \mathcal{G}|\!)$ such that $(\!|t, \mathcal{E}|\!)\mathcal{S}^{b_1}(\!|s, \mathcal{G}|\!)\mathcal{S}^{b_2}(\!|u, \mathcal{F}|\!)$ with $b \to b_1 \wedge b_2$. The general case is more tedious but similar.

First we check that if b is satisfiable, then $qv(t) = qv(s) = qv(u)$ and $\mathcal{E} \approx_{\overline{qv(t)}} \mathcal{G} \approx_{\overline{qv(t)}} \mathcal{F}$. Now for any $\mathcal{G}' \in CP_t(\mathcal{H}_{\overline{qv(t)}})$, suppose $(\!|t, \mathcal{G}'\mathcal{E}|\!) \xrightarrow{b_1', \gamma} \Delta$ with $bv(\gamma) \cap fv(b_1, t, u) = \emptyset$. By α-conversion, we may assume further that $bv(\gamma) \cap fv(s) = \emptyset$. From $(\!|t, \mathcal{E}|\!)\mathcal{S}^{b_1}(\!|s, \mathcal{G}|\!)$, there exists a collection of booleans $\{c_i : 1 \leq i \leq n\}$ such that $b_1 \wedge b_1' \to \bigvee c_i$ and for any i, $\exists c_i', \gamma_i$ with $c_i \to c_i'$, $\gamma =_{c_i} \gamma_i$, $(\!|s, \mathcal{G}'\mathcal{G}|\!) \xrightarrow{c_i', \gamma_i} \Theta$, and $(\mathcal{G}'\mathcal{E} \bullet \Delta)\mathcal{S}^{c_i}(\mathcal{G}'\mathcal{G} \bullet \Theta)$. By α-conversion, we can again assume that for each i, $bv(\gamma_i) \cap fv(b_2, s, u) = \emptyset$. Now by the assumption that $(\!|s, \mathcal{G}|\!)\mathcal{S}^{b_2}(\!|u, \mathcal{F}|\!)$, there exists a collection of booleans $\{d_{ij} : 1 \leq j \leq n_i\}$ such that $b_2 \wedge c_i' \to \bigvee_j d_{ij}$ and for any d_{ij}, $\exists d_{ij}', \gamma_{ij}$ with $d_{ij} \to d_{ij}'$, $\gamma_{ij} =_{d_{ij}} \gamma_i$, $(\!|u, \mathcal{G}'\mathcal{F}|\!) \xrightarrow{d_{ij}', \gamma_{ij}} \Xi$, and $(\mathcal{G}'\mathcal{G} \bullet \Theta)\mathcal{S}^{d_{ij}}(\mathcal{G}'\mathcal{F} \bullet \Xi)$.

Now let

$$B = \{b \wedge c_i \wedge d_{ij} : 1 \leq i \leq n, 1 \leq j \leq n_i\}.$$

From the fact that $b \to b_1 \wedge b_2$, it is easy to check that $b \wedge b_1' \to \bigvee B$. For any $c = b \wedge c_i \wedge d_{ij}$, we take $c' = d_{ij}'$ and $\gamma' = \gamma_{ij}$. Then $c \to c'$, $\gamma' =_c \gamma$, and $(\!|u, \mathcal{G}'\mathcal{F}|\!) \xrightarrow{c', \gamma'} \Xi$. Furthermore, by the fact that $c \to c_i$ and the definition of $\mathcal{U}^c$, we have $(\mathcal{G}'\mathcal{E} \bullet \Delta)\mathcal{U}^c(\mathcal{G}'\mathcal{G} \bullet \Theta)$ indeed. Similarly, $(\mathcal{G}'\mathcal{G} \bullet \Theta)\mathcal{U}^c(\mathcal{G}'\mathcal{F} \bullet \Xi)$. Thus $(\mathcal{G}'\mathcal{E} \bullet \Delta)\mathcal{U}^c(\mathcal{G}'\mathcal{F} \bullet \Xi)$ as required. □

Lemma 5.12. *Let decreasing families $\mathfrak{S}_i = \{\mathcal{S}_i^b : b \in BExp\}$, $i = 1, 2$, be symbolic bisimulations. Then the family $\mathfrak{S} = \{(\mathcal{S}_1^b \mathcal{S}_2^b)^* : b \in BExp\}$ is also a symbolic bisimulation.*

Proof. Let $b \in BExp$ and $(\!|t, \mathcal{E}|\!)(\mathcal{S}_1^b \mathcal{S}_2^b)^*(\!|u, \mathcal{F}|\!)$. Suppose there exist $n \geq 1$ and a sequence of snapshots $(\!|t_i, \mathcal{E}_i|\!)$, $0 \leq i \leq n$, such that $(\!|t, \mathcal{E}|\!) = (\!|t_0, \mathcal{E}_0|\!)$, $(\!|u, \mathcal{F}|\!) = (\!|t_n, \mathcal{E}_n|\!)$, and for each $0 \leq i \leq n-1$, $(\!|t_i, \mathcal{E}_i|\!)\mathcal{S}_1^b\mathcal{S}_2^b(\!|t_{i+1}, \mathcal{E}_{i+1}|\!)$. Again, for the sake of simplicity, we assume $n = 1$. That is, there exists $(\!|s, \mathcal{G}|\!)$ such that $(\!|t, \mathcal{E}|\!)\mathcal{S}_1^b(\!|s, \mathcal{G}|\!)\mathcal{S}_2^b(\!|u, \mathcal{F}|\!)$. The rest of the poof follows almost the same lines of those in Lemma 5.11, by employing the assumption that $\mathfrak{S}_1$ and $\mathfrak{S}_2$ are both decreasing. □

Lemma 5.13. *Let $\mathfrak{S} = \{\mathcal{S}^b : b \in BExp\}$ be a symbolic bisimulation and $c \in BExp$. Then $\mathfrak{S}_c = \{\mathcal{U}^b = \mathcal{S}^{b \vee c} : b \in BExp\}$ is also a symbolic bisimulation.*

Proof. Easy from definition. □

Corollary 5.14. *If $b \to b'$, then $\sim^{b'} \subseteq \sim^b$. That is, the relation family $\{\sim^b : b \in BExp\}$ is decreasing.*

With the lemmas above, we can show that the family $\{\sim^b : b \in BExp\}$ is actually the largest symbolic bisimulation.

Theorem 5.15. *(1) For each $b \in BExp$, $\sim^b$ is an equivalence relation.*

(2) The family $\{\sim^b : b \in BExp\}$ is a symbolic bisimulation.

Proof. (2) is direct from (1). To prove (1), let $b \in BExp$. Obviously, $\sim^b$ is reflexive and symmetric. To show the transitivity of $\sim^b$, let $(\!|t, \mathcal{E}|\!) \sim^b (\!|u, \mathcal{F}|\!)$ and $(\!|u, \mathcal{F}|\!) \sim^b (\!|s, \mathcal{G}|\!)$. Then by definition, there exist symbolic bisimulations $\mathfrak{S}_i = \{\mathcal{S}_i^b : b \in BExp\}$, $i = 1, 2$, such that $(\!|t, \mathcal{E}|\!)\mathcal{S}_1^b(\!|u, \mathcal{F}|\!)$ and $(\!|u, \mathcal{F}|\!)\mathcal{S}_2^b(\!|s, \mathcal{G}|\!)$. By Lemma 5.11, we can assume without loss of generality that both $\mathfrak{S}_1$ and $\mathfrak{S}_2$ are decreasing, thus $\mathfrak{S} = \{(\mathcal{S}_1^b\mathcal{S}_2^b)^* : b \in BExp\}$ is also a symbolic bisimulation, by Lemma 5.12. So $(\!|t, \mathcal{E}|\!) \sim^b (\!|s, \mathcal{G}|\!)$. □

To conclude this subsection, we present a property of symbolic bisimilarity which is useful for the next section.

Theorem 5.16. *Let $(\!|t, \mathcal{E}|\!), (\!|u, \mathcal{F}|\!) \in SN$ and $b \in BExp$. Then $(\!|t, \mathcal{E}|\!) \sim^b (\!|u, \mathcal{F}|\!)$ if and only if*

(1) $qv(t) = qv(u)$ and $\mathcal{E} \approx_{\overline{qv(t)}} \mathcal{F}$, if b is satisfiable;

(2) for any $\mathcal{G} \in CP_t(\mathcal{H}_{\overline{qv(t)}})$, whenever $(\!|t, \mathcal{G}\mathcal{E}|\!) \xrightarrow{b_1,\gamma} \Delta$ with $bv(\gamma) \cap fv(b, t, u) = \emptyset$, then there exist a collection of booleans B such that $b \wedge b_1 \to \bigvee B$ and $\forall\, b' \in B$, $\exists b_2, \gamma'$ with $b' \to b_2$, $\gamma =_{b'} \gamma'$, $(\!|u, \mathcal{G}\mathcal{F}|\!) \xrightarrow{b_2,\gamma'} \Xi$, and $(\mathcal{G}\mathcal{E} \bullet \Delta) \sim^{b'} (\mathcal{G}\mathcal{F} \bullet \Xi)$;

(3) Symmetric condition of (2).

Proof. Routine. □

5.4 Connection of symbolic and open bisimulations

To ease notation, in the rest of the paper we use $\mathfrak{t}, \mathfrak{u}$ to range over SN, and sometimes equate $\mathfrak{t}$ with $(\!|t, \mathcal{E}|\!)$, $\mathfrak{u}$ with $(\!|u, \mathcal{F}|\!)$, Δ with $\sum_{i\in I} \mathcal{A}_i \bullet (\!|t_i, \mathcal{E}_i|\!)$, and Ξ with $\sum_{j\in J} \mathcal{B}_j \bullet (\!|u_j, \mathcal{F}_j|\!)$ without stating them explicitly. We also write

$$(\Delta\psi)(\rho) = \sum_{i\in I} \mathrm{tr}(\mathcal{A}_i(\rho))\langle t_i\psi, \mathcal{E}_i(\rho)\rangle \text{ and } (\Xi\psi)(\rho) = \sum_{j\in J} \mathrm{tr}(\mathcal{B}_j(\rho))\langle u_j\psi, \mathcal{F}_j(\rho)\rangle.$$

In particular, $(\mathfrak{t}\psi)(\rho) = \langle t\psi, \mathcal{E}(\rho)\rangle$ and $(\mathfrak{u}\psi)(\rho) = \langle u\psi, \mathcal{F}(\rho)\rangle$. The basic ideas of the proofs in this subsection are borrowed from [12], with the help of Lemma 5.5 and 5.6.

Let $\mathfrak{S} = \{\mathcal{S}^b : b \in BExp\}$ be a symbolic bisimulation. Define

$$\mathcal{R}_{\mathfrak{S}} = \{((\mathfrak{t}\psi)(\rho), (\mathfrak{u}\psi)(\rho)) : \rho \in \mathcal{D}(\mathcal{H}) \text{ and } \exists b, \psi(b) = \mathtt{tt} \text{ and } \mathfrak{t}\mathcal{S}^b\mathfrak{u}\}.$$

We prove that $\mathcal{R}_{\mathfrak{S}}$ is an open bisimulation. To achieve this, the following lemma is needed.

Lemma 5.17. *Let $\mathfrak{S} = \{\mathcal{S}^b : b \in BExp\}$ be a symbolic bisimulation, $\rho \in \mathcal{D}(\mathcal{H})$, and $\psi(b) = \mathtt{tt}$. Then*

$$\Delta\, \mathcal{S}^b\, \Xi \text{ implies } (\Delta\psi)(\rho)\, \mathcal{R}_{\mathfrak{S}}\, (\Xi\psi)(\rho).$$

Proof. Suppose $\Delta = \sum_{i\in I} \mathcal{A}_i \bullet (\!|t_i, \mathcal{E}_i|\!)$, $\Xi = \sum_{j\in J} \mathcal{B}_j \bullet (\!|u_j, \mathcal{F}_j|\!)$ and $\Delta\, \mathcal{S}^b\, \Xi$. We decompose the set $\lceil\Delta\rceil \cup \lceil\Xi\rceil$ into disjoint subsets $S_1, \cdots, S_n$ such that any two snapshots are in the same S_k if and only if they are related by $\mathcal{S}^b$. For each $1 \le k \le n$, let

$$K_k = \{i \in I : (\!|t_i, \mathcal{E}_i|\!) \in S_k\} \cup \{j \in J : (\!|u_j, \mathcal{F}_j|\!) \in S_k\}.$$

Then

$$\sum_{i\in K_k\cap I} \mathcal{A}_i \approx \sum_{j\in K_k\cap J} \mathcal{B}_j. \tag{4}$$

For any $\rho \in \mathcal{D}(\mathcal{H})$ and ψ such that $\psi(b) = \mathtt{tt}$,

$$\begin{aligned}(\Delta\psi)(\rho) &= \sum_{i\in I} \mathrm{tr}(\mathcal{A}_i(\rho))\langle t_i\psi, \mathcal{E}_i(\rho)\rangle = \sum_{k=1}^{n} \sum_{i\in K_k\cap I} \mathrm{tr}(\mathcal{A}_i(\rho))\langle t_i\psi, \mathcal{E}_i(\rho)\rangle \\ &= \sum_{k=1}^{n} \frac{1}{\sum_{j\in K_k\cap J}\mathrm{tr}(\mathcal{B}_j(\rho))} \sum_{i\in K_k\cap I}\sum_{j\in K_k\cap J} \mathrm{tr}(\mathcal{A}_i(\rho))\mathrm{tr}(\mathcal{B}_j(\rho))\langle t_i\psi, \mathcal{E}_i(\rho)\rangle.\end{aligned}$$

Similarly, we have

$$\begin{aligned}(\Xi\psi)(\rho) &= \sum_{j\in J} \mathrm{tr}(\mathcal{B}_j(\rho))\langle u_j\psi, \mathcal{F}_j(\rho)\rangle = \sum_{k=1}^{n} \sum_{j\in K_k\cap J} \mathrm{tr}(\mathcal{B}_j(\rho))\langle u_j\psi, \mathcal{F}_j(\rho)\rangle \\ &= \sum_{k=1}^{n} \frac{1}{\sum_{i\in K_k\cap I}\mathrm{tr}(\mathcal{A}_i(\rho))} \sum_{i\in K_k\cap I}\sum_{j\in K_k\cap J} \mathrm{tr}(\mathcal{A}_i(\rho))\mathrm{tr}(\mathcal{B}_j(\rho))\langle u_j\psi, \mathcal{F}_j(\rho)\rangle.\end{aligned}$$

Note that by definition, if $\mathfrak{t}\mathcal{S}^b\mathfrak{u}$ then $(\mathfrak{t}\psi)(\rho)\mathcal{R}_{\mathfrak{S}}(\mathfrak{u}\psi)(\rho)$. It follows that for any $1 \le k \le n$, $i \in K_k \cap I$, and $j \in K_k \cap J$, we have $\langle t_i\psi, \mathcal{E}_i(\rho)\rangle \mathcal{R}_{\mathfrak{S}} \langle u_j\psi, \mathcal{F}_j(\rho)\rangle$. Furthermore, by Eq.(4), we know $\sum_{i\in K_k\cap I}\mathrm{tr}(\mathcal{A}_i(\rho)) = \sum_{j\in K_k\cap J}\mathrm{tr}(\mathcal{B}_j(\rho))$. Thus $(\Delta\psi)(\rho)\ \mathcal{R}_{\mathfrak{S}}\ (\Xi\psi)(\rho)$ by definition. □

Lemma 5.18. *Let $\mathfrak{S} = \{\mathcal{S}^b : b \in BExp\}$ be a symbolic bisimulation. Then $\mathcal{R}_{\mathfrak{S}}$ is an open bisimulation.*

Proof. Let $(\mathfrak{t}\psi)(\rho)\mathcal{R}_{\mathfrak{S}}(\mathfrak{u}\psi)(\rho)$. Then there exists b, such that $\psi(b) = \mathtt{tt}$ and $\mathfrak{t}\mathcal{S}^b\mathfrak{u}$. Thus we have

(1) $qv(t\psi) = qv(t) = qv(u) = qv(u\psi)$, and $\mathrm{tr}_{qv(t\psi)}\mathcal{E}(\rho) = \mathrm{tr}_{qv(t\psi)}\mathcal{F}(\rho)$ from $\mathcal{E} \eqsim_{\overline{qv(t)}} \mathcal{F}$.

(2) For any $\mathcal{G} \in CP_t(\mathcal{H}_{\overline{qv(t\psi)}})$, let

$$\langle t\psi, \mathcal{G}\mathcal{E}(\rho)\rangle \overset{\alpha}{\longmapsto} \mu.$$

Then by Lemma 5.5, we have

$$(\!|t, \mathcal{G}\mathcal{E}|\!) \overset{b_1,\gamma}{\longrightarrow} \Delta' = \sum_{i\in I} \mathcal{A}_i \bullet (\!|t_i, \mathcal{E}_i\mathcal{G}\mathcal{E}|\!)$$

such that $\psi(b_1) = \mathtt{tt}$,

$$\mu = \sum_{i\in I} \mathrm{tr}(\mathcal{A}_i\mathcal{G}\mathcal{E}(\rho))\langle t_i\psi', \mathcal{E}_i\mathcal{G}\mathcal{E}(\rho)\rangle.$$

Furthermore, we have $\gamma = c?x$ for some $x \notin fv(t)$ and $\psi' = \psi\{v/x\}$ if $\alpha = c?v$, or $\gamma =_\psi \alpha$ and $\psi' = \psi$ otherwise. Note that if $\gamma = c?x$, we can always take x such that $x \notin fv(t, u, b)$ by α-conversion. Now by the assumption that $\mathfrak{t}\mathcal{S}^b\mathfrak{u}$, there exists a collection of booleans B such that $b \wedge b_1 \to \bigvee B$ and $\forall\, b' \in B$, $\exists b_2, \gamma'$ with $b' \to b_2$, $\gamma =_{b'} \gamma'$,

$$(\!|u, \mathcal{G}\mathcal{F}|\!) \overset{b_2,\gamma'}{\longrightarrow} \Xi' = \sum_{j\in J} \mathcal{B}_j \bullet (\!|u_j, \mathcal{F}_j\mathcal{G}\mathcal{F}|\!),$$

and $(\mathcal{G}\mathcal{E} \bullet \Delta')\mathcal{S}^{b'}(\mathcal{G}\mathcal{F} \bullet \Xi')$. Note that $\psi(b \wedge b_1) = \mathtt{tt}$ and $b \wedge b_1 \to \bigvee B$. We can always find a $b' \in B$ such that $\psi(b') = \mathtt{tt}$, and so $\psi(b_2) = \mathtt{tt}$ as well. Then by Lemma 5.6, we have

$$\langle u\psi, \mathcal{G}\mathcal{F}(\rho)\rangle \overset{\beta}{\longmapsto} \nu = \sum_{j\in J} \mathrm{tr}(\mathcal{B}_j\mathcal{G}\mathcal{F}(\rho))\langle u_j\psi'', \mathcal{F}_j\mathcal{G}\mathcal{F}(\rho)\rangle$$

where $\beta = c?v$ and $\psi'' = \psi\{v/x\}$ if $\gamma' = c?x$, or $\gamma' =_\psi \beta$ and $\psi'' = \psi$ otherwise.

We claim that $\beta = \alpha$, and $\psi'' = \psi'$. There are three cases to consider:

(i) $\alpha = c?v$. Then $\gamma = c?x$ and $\psi' = \psi\{v/x\}$. So $\gamma' = c?x$ by definition, which implies that $\beta = c?v = \alpha$, and $\psi'' = \psi\{v/x\} = \psi'$.

(ii) $\alpha = c!v$. Then $\gamma = c!e$, $\psi(e) = v$, and $\psi' = \psi$. So $\gamma' = c!e'$ with $b' \to e = e'$, which implies that $\beta = c!v'$ where $v' = \psi(e')$, and $\psi'' = \psi = \psi'$. Finally, from $\psi(b') = \mathtt{tt}$ we deduce $v' = v$.

(iii) For other cases, $\beta = \gamma' = \gamma = \alpha$, and $\psi'' = \psi = \psi'$.

Finally, by Lemma 5.17 we deduce $\mu\mathcal{R}_{\mathfrak{S}}\nu$ from the facts that $(\mathcal{G}\mathcal{E} \bullet \Delta')\mathcal{S}^{b'}(\mathcal{G}\mathcal{F} \bullet \Xi')$ and $\psi'(b') = \mathtt{tt}$. □

Corollary 5.19. *Let $b \in BExp$, $t, u \in \mathcal{T}$, and $P, Q \in \mathcal{P}$. Then*

(1) $t \sim^b u$ implies for any evaluation ψ, if $\psi(b) = \mathtt{tt}$ then $t\psi \dot{\sim} u\psi$.

(2) $t \sim u$ implies $t \dot{\sim} u$.

(3) $P \sim^b Q$ implies $P \dot{\sim} Q$, provided that b is satisfiable.

Proof. (2) and (3) are both direct corollaries of (1). To prove (1), let $t \sim^b u$, and $\mathfrak{S} = \{\mathcal{S}^b : b \in BExp\}$ be a symbolic bisimulation such that $(\!|t, \mathcal{I}_{\mathcal{H}}|\!)\mathcal{S}^b(\!|u, \mathcal{I}_{\mathcal{H}}|\!)$. Then by Lemma 5.18, for any evaluation ψ and any ρ, $\psi(b) = \mathtt{tt}$ implies $\langle t\psi, \rho\rangle \dot{\sim} \langle u\psi, \rho\rangle$. Thus $t\psi \dot{\sim} u\psi$ by definition. □

For any $b \in BExp$, define

$$\mathcal{S}^b_{\dot{\sim}} = \{(\mathfrak{t}, \mathfrak{u}) : \forall \psi, \psi(b) = \mathtt{tt} \text{ implies that for any } \rho \in \mathcal{D}(\mathcal{H}), (\mathfrak{t}\psi)(\rho) \dot{\sim} (\mathfrak{u}\psi)(\rho)\}.$$

We prove that $\mathfrak{S}_{\dot{\sim}} = \{\mathcal{S}^b_{\dot{\sim}} : b \in BExp\}$ is a symbolic bisimulation. Firstly, it is easy to check that for each b, $\mathcal{S}^b_{\dot{\sim}}$ is an equivalence relation. Two quantum states $\rho, \sigma \in \mathcal{D}(\mathcal{H})$ are said to be *equal except at* $\widetilde{q}$ if $\mathrm{tr}_{\widetilde{q}}\rho = \mathrm{tr}_{\widetilde{q}}\sigma$. Then we can show the following lemma, which is parallel to Lemma 5.17.

Lemma 5.20. *Let $b \in BExp$. If for any evaluation ψ,*

$$\psi(b) = \mathtt{tt} \text{ implies that } \forall \rho \in \mathcal{D}(\mathcal{H}), (\Delta\psi)(\rho) \dot{\sim} (\Xi\psi)(\rho),$$

then $\Delta\, \mathcal{S}^b_{\dot{\sim}}\, \Xi$.

Proof. Let $\Delta = \sum_{i\in I} \mathcal{A}_i \bullet (\!|t_i, \mathcal{E}_i|\!)$ and $\Xi = \sum_{j\in J} \mathcal{B}_j \bullet (\!|u_j, \mathcal{F}_j|\!)$. We prove this lemma by distinguishing two cases:

(1) Both $|I| > 1$ and $|J| > 1$. Similar to Lemma 5.17, we first decompose the set $\lceil\Delta\rceil \cup \lceil\Xi\rceil$ into disjoint subsets $S_1, \cdots, S_n$ such that any two snapshots are in the same S_k if and only if they are related by $\mathcal{S}^b_{\dot{\sim}}$. For each $1 \le k \le n$, let

$$K_k = \{i \in I : (\!|t_i, \mathcal{E}_i|\!) \in S_k\} \cup \{j \in J : (\!|u_j, \mathcal{F}_j|\!) \in S_k\} \tag{5}$$

and $\mathfrak{K} = \{K_k : 1 \le k \le n\}$. Note that by Lemma 5.4, there are two sets of pairwise orthogonal pure states $\{|\phi_i\rangle : i \in I\}$ and $\{|\phi'_j\rangle : j \in J\}$ in some $\mathcal{H}_{\widetilde{q}}$ such that the Kraus operators of $\mathcal{A}_i$ and $\mathcal{E}_i$ are $\{|\phi_i\rangle\langle\phi_i|\}$ and $\{|\phi_i\rangle\langle\phi_{i'}| : i' \in I\}$, respectively, while the Kraus operators of $\mathcal{B}_j$ and $\mathcal{F}_j$ are $\{|\phi'_j\rangle\langle\phi'_j|\}$ and $\{|\phi'_j\rangle\langle\phi'_{j'}| : j' \in J\}$, respectively. Let $E_k = \sum_{i\in K_k\cap I} |\phi_i\rangle\langle\phi_i|$, and $F_k = \sum_{j\in K_k\cap J} |\phi'_j\rangle\langle\phi'_j|$. Then it suffices to show $E_k = F_k$, $1 \le k \le n$. In the following, we prove $E_1 = F_1$; other cases are similar.

For any ρ and ψ such that $\psi(b) = \mathtt{tt}$, we decompose the set $\lceil(\Delta\psi)(\rho)\rceil \cup \lceil(\Xi\psi)(\rho)\rceil$ into equivalence classes $R_1, \cdots, R_{m^\psi_\rho}$ according to $\dot{\sim}$. For each $1 \le l \le m^\psi_\rho$, let

$$L^{\psi,\rho}_l = \{i \in I : \langle t_i\psi, \mathcal{E}_i(\rho)\rangle \in R_l\} \cup \{j \in J : \langle u_j\psi, \mathcal{F}_j(\rho)\rangle \in R_l\}$$

and $\mathfrak{L}^{\psi,\rho} = \{L_l^{\psi,\rho} : 1 \leq l \leq R_{m_\rho^\psi}\}$. Note that by definition, $\mathfrak{K}$ is a refinement of $\mathfrak{L}^{\psi,\rho}$ for any $\psi(b) = \mathtt{tt}$ and ρ. We assume without loss of generality that $L_1^{\psi,\rho}$ is the partition in $\mathfrak{L}^{\psi,\rho}$ which contains K_1, and $L_1^{\psi,\rho} = K_1 \cup K_1^{\psi,\rho}$ where $K_1^{\psi,\rho} = \bigcup_{k\in I_{\psi,\rho}} K_k$, $I_{\psi,\rho}$ is a subset of $\{2, \cdots, n\}$. As the effects of the super-operators $\mathcal{A}_i$ and $\mathcal{B}_j$ are simply erasing the original information at $\widetilde{q}$ and setting the partial states of $\widetilde{q}$ to be $|\phi_i\rangle$ and $|\phi'_j\rangle$, respectively, we have $\mathfrak{L}^{\psi,\rho} = \mathfrak{L}^{\psi,\sigma}$ (which means $m_\rho^\psi = m_\sigma^\psi$, and $L_l^{\psi,\rho} = L_l^{\psi,\sigma}$ for each l) for all σ which is equal to ρ except at $\widetilde{q}$. Note that $\mathrm{tr}(\mathcal{A}_i(\rho)) = \mathrm{tr}(|\phi_i\rangle_{\widetilde{q}}\langle\phi_i|\rho) = \mathrm{tr}(|\phi_i\rangle_{\widetilde{q}}\langle\phi_i|\rho_{\widetilde{q}})$ where $\rho_{\widetilde{q}} = \mathrm{tr}_{\overline{\widetilde{q}}}\rho$ is the reduced state of ρ at the systems $\widetilde{q}$. Let $E_1^{\psi,\rho} = \sum_{k\in I_{\psi,\rho}} E_k$ and $F_1^{\psi,\rho} = \sum_{k\in I_{\psi,\rho}} F_k$. Then for any $\rho' \in \mathcal{D}(\mathcal{H}_{\widetilde{q}})$,

$$\mathrm{tr}((E_1 + E_1^{\psi,\rho})\rho') = \sum_{i\in L_1^{\psi,\sigma}\cap I} \mathrm{tr}(\mathcal{A}_i(\sigma)) = \sum_{j\in L_1^{\psi,\sigma}\cap J} \mathrm{tr}(\mathcal{B}_j(\sigma)) = \mathrm{tr}((F_1 + F_1^{\psi,\rho})\rho')$$

where $\sigma = \rho' \otimes \mathrm{tr}_{\widetilde{q}}(\rho)$ is equal to ρ except at $\widetilde{q}$, and the second equality is from the assumption that $(\Delta\psi)(\sigma) \dot{\sim} (\Xi\psi)(\sigma)$. This implies $E_1 + E_1^{\psi,\rho} = F_1 + F_1^{\psi,\rho}$.

Let $K = \bigcap_{\rho,\psi(b)=\mathtt{tt}} I_{\psi,\rho}$. We claim that $K = \emptyset$. Otherwise, there exists k such that $k \in I_{\psi,\rho}$ for any $\psi(b) = \mathtt{tt}$ and ρ. Then by the definition of $L_1^{\psi,\rho}$, we have $\langle t_i\psi, \mathcal{E}_i(\rho)\rangle \dot{\sim} \langle t_{i'}\psi, \mathcal{E}_{i'}(\rho)\rangle$ where $i \in K_1$ and $i' \in K_k$. Thus $(\!|t_i, \mathcal{E}_i|\!)\mathcal{S}^b_{\dot{\sim}} (\!|t_{i'}, \mathcal{E}_{i'}|\!)$, contradicting the fact that they belong to different equivalence classes of $\mathcal{S}^b_{\dot{\sim}}$.

Now for any pure state $|\phi\rangle$ such that $E_1|\phi\rangle = |\phi\rangle$, we have $E_1^{\psi,\rho}|\phi\rangle = 0$ for any ρ and $\psi(b) = \mathtt{tt}$, by the orthogonality of E_i's. Thus $F_1^{\psi,\rho}|\phi\rangle = |\phi\rangle - F_1|\phi\rangle$. Note that $F_1^{\psi',\rho'}F_1^{\psi,\rho} = \sum_{k\in I_{\psi,\rho}\cap I_{\psi',\rho'}} F_k = F_1^{\psi,\rho}F_1^{\psi',\rho'}$. We have

$$\sum_{k\in I_{\psi,\rho}\cap I_{\psi',\rho'}} F_k|\phi\rangle = |\phi\rangle - F_1|\phi\rangle,$$

and finally, $\sum_{k\in K} F_k|\phi\rangle = |\phi\rangle - F_1|\phi\rangle$. Then $F_1|\phi\rangle = |\phi\rangle$ from the fact that $K = \emptyset$. Similarly, we can prove that for any $|\phi\rangle$, $F_1|\phi\rangle = |\phi\rangle$ implies $E_1|\phi\rangle = |\phi\rangle$. Thus $E_1 = F_1$.

(2) Either $|I| = 1$ or $|J| = 1$. Let us suppose $|I| = 1$, and $\Delta = (\!|t, \mathcal{E}|\!)$. We need to show that for each $j \in J$, $\mathcal{B}_j \neq 0_{\mathcal{H}}$ implies $(\!|t, \mathcal{E}|\!)\mathcal{S}^b_{\dot{\sim}} (\!|u_j, \mathcal{F}_j|\!)$. This is true because otherwise we can find $\psi(b) = \mathtt{tt}$, $j \in J$, and $\rho \in \mathcal{D}(\mathcal{H})$ such that $\mathrm{tr}(\mathcal{B}_j(\rho)) \neq 0$ but $\langle t\psi, \mathcal{E}(\rho)\rangle \dot{\nsim} \langle u_j\psi, \mathcal{F}_j(\rho)\rangle$. Thus $(\Delta\psi)(\rho) \dot{\nsim} (\Xi\psi)(\rho)$, a contradiction. □

Lemma 5.21. *The family* $\mathfrak{S}_{\dot{\sim}} = \{\mathcal{S}^b_{\dot{\sim}} : b \in BExp\}$ *is a symbolic bisimulation.*

Proof. Let $b \in BExp$ and $\mathfrak{t}\mathcal{S}^b_{\dot{\sim}}\,\mathfrak{u}$. Then for any ψ, $\psi(b) = \mathtt{tt}$ implies that for any $\rho \in \mathcal{D}(\mathcal{H})$, $(\mathfrak{t}\psi)(\rho) \dot{\sim} (\mathfrak{u}\psi)(\rho)$. Thus we have

(1) If b is satisfiable, then $qv(t) = qv(t\psi) = qv(u\psi) = qv(u)$, and $\mathcal{E} \approx_{\overline{qv(t)}} \mathcal{F}$ from the fact that $\mathrm{tr}_{qv(t)}\mathcal{E}(\rho) = \mathrm{tr}_{qv(t)}\mathcal{F}(\rho)$ for any ρ.

(2) For any $\mathcal{G} \in CP_t(\mathcal{H}_{\overline{qv(t)}})$, let

$$(\!|t, \mathcal{G}\mathcal{E}|\!) \xrightarrow{b_1,\gamma} \Delta' = \sum_{i\in I} \mathcal{A}_i \bullet (\!|t_i, \mathcal{E}_i\mathcal{G}\mathcal{E}|\!) \tag{6}$$

with $bv(\gamma) \cap fv(b, t, u) = \emptyset$. We need to construct a set of booleans B such that $b \wedge b_1 \to \bigvee B$, and $\forall\, b' \in B$, $\exists b_2, \gamma'$ with $b' \to b_2$, $\gamma =_{b'} \gamma'$, $(\!|u, \mathcal{G}\mathcal{F}|\!) \xrightarrow{b_2,\gamma'} \Xi'$, and $(\mathcal{G}\mathcal{E} \bullet \Delta')\mathcal{S}^{b'}(\mathcal{G}\mathcal{F} \bullet \Xi')$. Let

$$U = \{\Theta : (\!|u, \mathcal{G}\mathcal{F}|\!) \xrightarrow{b(\Theta),\gamma(\Theta)} \Theta \text{ and } \gamma =_{\mathtt{ff}} \gamma(\Theta)\}.$$

Here similar to [12], to ease the notations we only consider the case where for each Θ, there is at most one symbolic action, denoted by $(b(\Theta), \gamma(\Theta))$, such that $(\!|u, \mathcal{GF}|\!) \overset{b(\Theta),\gamma(\Theta)}{\longrightarrow} \Theta$. For each $\Theta \in U$, let b'_Θ be a boolean expression such that for any ψ,

$$\psi(b'_\Theta) = \mathtt{tt} \text{ if and only if for any } \rho, (\mathcal{GE} \bullet \Delta'\tilde{\psi})(\rho) \dot{\sim} (\mathcal{GF} \bullet \Theta\tilde{\psi})(\rho) \tag{7}$$

where $\tilde{\psi} = \psi\{v/x\}$ for some v if $\gamma = c?x$, and $\tilde{\psi} = \psi$ otherwise.

Let $B = \{b_\Theta : \Theta \in U\}$, where $b_\Theta = b'_\Theta \wedge b''_\Theta \wedge b(\Theta)$ and b''_Θ is a boolean expression defined by

$$b''_\Theta \equiv \begin{cases} e = e' & \text{if } \gamma = c!e \text{ and } \gamma(\Theta) = c!e' \text{ are both classical output,} \\ \mathtt{tt} & \text{otherwise.} \end{cases} \tag{8}$$

Then obviously, $\gamma =_{b_\Theta} \gamma(\Theta)$. We check $b \wedge b_1 \to \bigvee B$. For any evaluation ψ such that $\psi(b \wedge b_1) = \mathtt{tt}$, we have by definition of $\mathcal{S}^b_{\dot{\sim}}$ that $\langle t\psi, \mathcal{E}(\rho)\rangle \dot{\sim} \langle u\psi, \mathcal{F}(\rho)\rangle$ for any ρ. On the other hand, by Lemma 5.6 and Eq.(6), we obtain

$$\langle t\psi, \mathcal{GE}(\rho)\rangle \overset{\alpha}{\longmapsto} \mu = \sum_{i\in I} \mathrm{tr}(\mathcal{A}_i\mathcal{GE}(\rho))\langle t_i\psi', \mathcal{E}_i\mathcal{GE}(\rho)\rangle$$

where $\alpha = c?v$ and $\psi' = \psi\{v/x\}$ if $\gamma = c?x$, and $\alpha =_\psi \gamma$ and $\psi' = \psi$ otherwise. To match this transition, we have

$$\langle u\psi, \mathcal{GF}(\rho)\rangle \overset{\alpha}{\longmapsto} \nu$$

for some ν such that $\mu \dot{\sim} \nu$. Now from Lemma 5.5, there exists $\Xi' \in U$ such that $\psi(b(\Xi')) = \mathtt{tt}$,

$$(\!|u, \mathcal{GF}|\!) \overset{b(\Xi'),\gamma(\Xi')}{\longrightarrow} \Xi' = \sum_{j\in J} \mathcal{B}_j \bullet (\!|u_j, \mathcal{F}_j\mathcal{GF}|\!),$$

$$\nu = \sum_{j\in J} \mathrm{tr}(\mathcal{B}_j\mathcal{GF}(\rho))\langle u_j\psi'', \mathcal{F}_j\mathcal{GF}(\rho)\rangle.$$

Furthermore, we have $\gamma(\Xi') = c?y$ for some $y \notin fv(u)$ and $\psi'' = \psi\{v/y\}$ if $\alpha = c?v$, and $\alpha =_\psi \gamma(\Xi')$ and $\psi'' = \psi$ otherwise.

We claim that $\gamma =_\psi \gamma(\Xi')$, and $\psi'' = \psi'$. There are three cases to consider:

(i) $\gamma = c?x$. Then $\alpha = c?v$ and $\psi' = \psi\{v/x\}$, which implies that $\gamma(\Xi') = c?y$ for some $y \notin fv(u)$. By α-conversion and the fact that $x \notin fv(b, t, u)$, we can also take $y = x$. So $\gamma(\Xi') = \gamma$, and $\psi'' = \psi\{v/x\} = \psi'$.

(ii) For other cases, $\gamma(\Xi') =_\psi \alpha =_\psi \gamma$, and $\psi'' = \psi = \psi'$.

Now we have $\mu = (\mathcal{GE} \bullet \Delta'\psi')(\rho)$ and $\nu = (\mathcal{GF} \bullet \Xi'\psi')(\rho)$. From the arbitrariness of ρ, we know $\psi(b'_{\Xi'}) = \mathtt{tt}$ from Eq.(7). By Eq.(8) and the fact that $\gamma =_\psi \gamma(\Xi')$, we further derive that $\psi(b''_{\Xi'}) = \mathtt{tt}$. Therefore, $\psi(b_{\Xi'}) = \mathtt{tt}$, and so $\psi(\bigvee B) = \mathtt{tt}$.

For any $b_\Theta \in B$, we have $b_\Theta \to b(\Theta)$, $\gamma =_{b(\Theta)} \gamma(\Theta)$, and $(\!|u, \mathcal{GF}|\!) \overset{b(\Theta),\gamma(\Theta)}{\longrightarrow} \Theta$ by definition of B. Finally, for any evaluation ψ, if $\psi(b_\Theta) = \mathtt{tt}$ then $\psi(b'_\Theta) = \mathtt{tt}$, and from Eq.(7) we have $(\mathcal{GE} \bullet \Delta'\tilde{\psi})(\rho) \dot{\sim} (\mathcal{GF} \bullet \Theta\tilde{\psi})(\rho)$ for any $\rho \in \mathcal{D}(\mathcal{H})$. Then $(\mathcal{GE} \bullet \Delta')\mathcal{S}^{b_\Theta}(\mathcal{GF} \bullet \Theta)$ follows by Lemma 5.20. Here we have used that fact that $x \notin fv(b, t, u)$ implies $t\psi\{v/x\} = t\psi$ and $u\psi\{v/x\} = t\psi$. □

Lemma 5.22. *If for any evaluation ψ, $\psi(b) = \mathtt{tt}$ implies $t\psi \dot{\sim} u\psi$, then $t \sim^b u$.*

Proof. For any $\rho \in \mathcal{D}(\mathcal{H})$ and any evaluation ψ such that $\psi(b) = \mathtt{tt}$, we first derive $\langle t\psi, \rho\rangle \dot{\sim} \langle u\psi, \rho\rangle$ from the assumption that $t\psi \dot{\sim} u\psi$. Then by Lemma 5.21, we have $(\!|t, \mathcal{I}_{\mathcal{H}}|\!) \sim^b (\!|u, \mathcal{I}_{\mathcal{H}}|\!)$, and thus $t \sim^b u$ by definition. □

From the above lemmas, we finally reach our main result in this section.

Theorem 5.23. *Let $b \in BExp$, $t, u \in \mathcal{T}$, and $P, Q \in \mathcal{P}$. Then*

(1) $t \sim^b u$ if and only if for any evaluation ψ, $\psi(b) = \mathtt{tt}$ implies $t\psi \dot{\sim} u\psi$.

(2) $t \sim u$ if and only if $t \dot{\sim} u$.

(3) $P \sim^b Q$ if and only if $P \dot{\sim} Q$, provided that b is satisfiable.

6 An algorithm for symbolic ground bisimulation

From Clause (2) of Definition 5.7, to check whether two snapshots are symbolically bisimilar, we are forced to compare their behaviours under any super-operators. This is generally infeasible since all super-operators constitute a continuum, and it seems hopeless to design an algorithm which works for the most general case. In this section, we develop an efficient algorithm for a class of quantum process terms which covers all existing practical quantum communication protocols. To this end, we first define the notion of symbolic ground bisimulation which stems from [18].

Definition 6.1. *A family of equivalence relations $\{\mathcal{S}^b : b \in BExp\}$ is called a symbolic ground bisimulation if for any $b \in BExp$, $(\!|t, \mathcal{E}|\!)\mathcal{S}^b(\!|u, \mathcal{F}|\!)$ implies that*

(1) $qv(t) = qv(u)$, and $\mathcal{E} \simeq_{\overline{qv(t)}} \mathcal{F}$,

(2) whenever $(\!|t, \mathcal{E}|\!) \xrightarrow{b_1,\gamma} \Delta$ with $bv(\gamma) \cap fv(b, t, u) = \emptyset$, then there exists a collection of booleans B such that $b \wedge b_1 \rightarrow \bigvee B$ and $\forall\ b' \in B$, $\exists b_2, \gamma'$ with $b' \rightarrow b_2$, $\gamma =_{b'} \gamma'$, $(\!|u, \mathcal{F}|\!) \xrightarrow{b_2,\gamma'} \Xi$, and $(\mathcal{E} \bullet \Delta)\mathcal{S}^{b'}(\mathcal{F} \bullet \Xi)$.

Given two configurations $(\!|t, \mathcal{E}|\!)$ and $(\!|u, \mathcal{F}|\!)$, we write $(\!|t, \mathcal{E}|\!) \sim_g^b (\!|u, \mathcal{F}|\!)$ if there exists a symbolic ground bisimulation $\{\mathcal{S}^b : b \in BExp\}$ such that $(\!|t, \mathcal{E}|\!)\mathcal{S}^b(\!|u, \mathcal{F}|\!)$.

Definition 6.2. *A relation $\mathcal{S}$ on SN is said to be closed under super-operator application if $(\!|t, \mathcal{E}|\!)\mathcal{S}(\!|u, \mathcal{F}|\!)$ implies $(\!|t, \mathcal{G}\mathcal{E}|\!)\mathcal{S}(\!|u, \mathcal{G}\mathcal{F}|\!)$ for any $\mathcal{G} \in CP_t(\mathcal{H}_{\overline{qv(t)}})$. A family of relations are closed under super-operator application if each individual relation is.*

The following proposition, showing the difference of symbolic bisimulation and symbolic ground bisimulation, is easy from definition.

Proposition 6.3. *$\sim$ is the largest symbolic ground bisimulation that is closed under super-operator application.*

A process term is said to be *free of quantum input* if all of its descendants, including itself, can not perform quantum input actions. Note that all existing quantum communication protocols such as super-dense coding [3], teleportation [2], quantum key-distribution protocols [1], etc, are, or can easily modified to be, free of quantum input. Putting this constraint will not bring too much restriction on the application range of our algorithm.

Lemma 6.4. *Let $(\!|t, \mathcal{E}|\!) \sim_g^b (\!|u, \mathcal{F}|\!)$, and t and u both free of quantum input. Then for any $\mathcal{G} \in CP_t(\mathcal{H}_{\overline{qv(t)}})$, $(\!|t, \mathcal{G}\mathcal{E}|\!) \sim_g^b (\!|u, \mathcal{G}\mathcal{F}|\!)$.*

Proof. We need to show $\mathfrak{S} = \{\mathcal{S}^b : b \in BExp\}$, where

$$\mathcal{S}^b = \{((\!|t, \mathcal{G}\mathcal{E}|\!), (\!|u, \mathcal{G}\mathcal{F}|\!)) : t \text{ and } u \text{ free of quantum input, } \mathcal{G} \in CP_t(\mathcal{H}_{\overline{qv(t)}}), \text{ and } (\!|t, \mathcal{E}|\!) \sim_g^b (\!|u, \mathcal{F}|\!)\},$$

is a symbolic ground bisimulation. This is easy by noting that for any descendant t' of t, $qv(t') \subseteq qv(t)$, and then $\mathcal{G} \in CP_t(\mathcal{H}_{\overline{qv(t')}})$ as well. Consequently, $\mathcal{G}$ commutes with all the super-operators performed by t and its descendants. □

Theorem 6.5. *If t and u are both free of quantum input, then $(\!|t, \mathcal{E}|\!) \sim^b (\!|u, \mathcal{F}|\!)$ if and only if $(\!|t, \mathcal{E}|\!) \sim_g^b (\!|u, \mathcal{F}|\!)$.*

Proof. Easy from Lemma 6.4. □

Algorithm 1 computes the *most general boolean* b such that $\mathfrak{t} \sim_g^b \mathfrak{u}$, for two given snapshots $\mathfrak{t}$ and $\mathfrak{u}$. By the most general boolean $mgb(\mathfrak{t}, \mathfrak{u})$ we mean that $\mathfrak{t} \sim_g^{mgb(\mathfrak{t},\mathfrak{u})} \mathfrak{u}$ and whenever $\mathfrak{t} \sim_g^b \mathfrak{u}$ then $b \to mgb(\mathfrak{t}, \mathfrak{u})$. From Theorem 6.5, this algorithm is applicable to verify the correctness of all existing quantum communication protocols.

The algorithm closely follows that introduced in [12]. The main procedure is **Bisim**$(\mathfrak{t}, \mathfrak{u})$. It starts with the initial snapshot pairs $(\mathfrak{t}, \mathfrak{u})$, trying to find the smallest symbolic bisimulation relation containing the pair by comparing transitions from each pair of snapshots it reaches. The core procedure **Match** has four parameters: $\mathfrak{t}$ and $\mathfrak{u}$ are the current terms under examination; b is a boolean expression representing the constraints accumulated by previous calls; W is a set of snapshot pairs which have been visited. For each possible action enabled by $\mathfrak{t}$ and $\mathfrak{u}$, the procedure **MatchAction** is used to compare possible moves from $\mathfrak{t}$ and $\mathfrak{u}$. Each comparison returns a boolean and a table; the boolean turns out to be $mgb(\mathfrak{t}, \mathfrak{u})$ and the table is used to represent the witnessing bisimulation. We consider a table as a function that maps a pair of snapshots to a boolean. The disjoint union of tables, viewed as sets, is denoted by $\sqcup$.

The main difference from the algorithm of [12] lies in the comparison of τ transitions. We introduce the procedure **MatchDistribution** to approximate $\sim_g^b$ by a relation $\mathcal{R}$. For any two snapshots $\mathfrak{t}_i \in \lceil\Delta\rceil$ and $\mathfrak{u}_j \in \lceil\Theta\rceil$, they are related by $\mathcal{R}$ if $b \to T(\mathfrak{t}_i, \mathfrak{u}_j)$. More precisely, we use the equivalence closure of $\mathcal{R}$ instead in order for it to be used in the procedure **Check**. Moreover, if a snapshot pair $(\mathfrak{t}, \mathfrak{u})$ has been visited before, i.e. $(\mathfrak{t}, \mathfrak{u}) \in W$, then $T(\mathfrak{t}, \mathfrak{u})$ is assumed to be $\mathtt{tt}$ in all future visits. Hence, $\mathcal{R}$ is coarser than $\sim_g^b$ in general. We use **Check**$(\Delta, \Theta, \mathcal{R})$ to computate the constraint so that the super-operator valued distribution Δ is related to Θ by a relation lifted from $\mathcal{R}$. The correctness of the algorithm is stated in the following theorem.

Theorem 6.6. *For two snapshots $\mathfrak{t}$ and $\mathfrak{u}$, the function **Bisim**$(\mathfrak{t}, \mathfrak{u})$ terminates. Moreover, if **Bisim**$(\mathfrak{t}, \mathfrak{u}) = (\theta, T)$ then $T(\mathfrak{t}, \mathfrak{u}) = \theta = mgb(\mathfrak{t}, \mathfrak{u})$.*

Proof. Termination is easy to show. Each time a new snapshot pair is encountered, the procedure **Match** is called and the pair is added to the set W. Since we are considering a finitary transition graph, the number of different pairs is finite. Eventually every possible pair is in W and each call to **Match** immediately terminates.

Correctness of the algorithm is largely similar to that in [12], though we use the additional procedure **MatchDistribution** to compute the constraint that relates two super-operator valued distributions. □

7 Modal characterisation

We now present a modal logic to characterise the behaviour of quantum snapshots and their distributions.

Algorithm 1: Bisim$(\mathfrak{t},\mathfrak{u})$

$\mathbf{Bisim}(\mathfrak{t},\mathfrak{u}) = \mathbf{Match}(\mathfrak{t},\mathfrak{u},\mathtt{tt},\emptyset)$

$\mathbf{Match}(\mathfrak{t},\mathfrak{u},b,W) =$ where $\mathfrak{t} = (\!|t,\mathcal{E}|\!)$ and $\mathfrak{u} = (\!|u,\mathcal{F}|\!)$
if $(\mathfrak{t},\mathfrak{u}) \in W$ **then**
  $(\theta,T) := (\mathtt{tt},\emptyset)$
else
  for $\gamma \in Act(\mathfrak{t},\mathfrak{u})$ **do**
    $(\theta_\gamma,T_\gamma) := \mathbf{MatchAction}(\gamma,\mathfrak{t},\mathfrak{u},b,W)$
  $(\theta,T) := (\bigwedge_\gamma \theta_\gamma,\ \bigsqcup_\gamma(T_\gamma \sqcup \{(\mathfrak{t},\mathfrak{u}) \mapsto (b \wedge \bigwedge_\gamma \theta_\gamma)\}))$
return $(\theta \wedge (qv(t) = qv(u)) \wedge (\mathcal{E} \approx_{\overline{qv(t)}} \mathcal{F}), T)$

$\mathbf{MatchAction}(\gamma,\mathfrak{t},\mathfrak{u},b,W) =$
switch γ **do**
  case $c!$
    for $\mathfrak{t} \xrightarrow{b_i,c!e_i} \mathfrak{t}_i$ *and* $\mathfrak{u} \xrightarrow{b'_j,c!e'_j} \mathfrak{u}_j$ **do**
      $(\theta_{ij},T_{ij}) := \mathbf{Match}(\mathfrak{t}_i,\mathfrak{u}_j,b \wedge b_i \wedge b'_j \wedge e_i = e'_j,\{(\mathfrak{t},\mathfrak{u})\} \cup W)$
    return $(\bigwedge_i(b_i \to \bigvee_j(b'_j \wedge e_i = e'_j \wedge \theta_{ij})) \wedge \bigwedge_j(b'_j \to \bigvee_i(b_i \wedge e_i = e'_j \wedge \theta_{ij})),\ \bigsqcup_{ij} T_{ij})$
  case τ
    for $\mathfrak{t} \xrightarrow{b_i,\tau} \Delta_i$ *and* $\mathfrak{u} \xrightarrow{b'_j,\tau} \Theta_j$ **do**
      $(\theta_{ij},T_{ij}) := \mathbf{MatchDistribution}(\Delta_i,\Theta_j,b \wedge b_i \wedge b'_j,\{(\mathfrak{t},\mathfrak{u})\} \cup W)$
    return $(\bigwedge_i(b_i \to \bigvee_j(b'_j \wedge \theta_{ij})) \wedge \bigwedge_j(b'_j \to \bigvee_i(b_i \wedge \theta_{ij})),\ \bigsqcup_{ij} T_{ij})$
  otherwise
    for $\mathfrak{t} \xrightarrow{b_i,\gamma} \mathfrak{t}_i$ *and* $\mathfrak{u} \xrightarrow{b'_j,\gamma} \mathfrak{u}_j$ **do**
      $(\theta_{ij},T_{ij}) := \mathbf{Match}(\mathfrak{t}_i,\mathfrak{u}_j,b \wedge b_i \wedge b'_j,\{(\mathfrak{t},\mathfrak{u})\} \cup W)$
    return $(\bigwedge_i(b_i \to \bigvee_j(b'_j \wedge \theta_{ij})) \wedge \bigwedge_j(b'_j \to \bigvee_i(b_i \wedge \theta_{ij})),\ \bigsqcup_{ij} T_{ij})$

$\mathbf{MatchDistribution}(\Delta,\Theta,b,W)=$
for $\mathfrak{t}_i \in \lceil\Delta\rceil$ *and* $\mathfrak{u}_j \in \lceil\Theta\rceil$ **do**
  $(\theta_{ij},T_{ij}) := \mathbf{Match}(\mathfrak{t}_i,\mathfrak{u}_j,b,W)$
$\mathcal{R} := \{(\mathfrak{t},\mathfrak{u}) \mid b \to (\bigsqcup_{ij} T_{ij})(\mathfrak{t},\mathfrak{u})\}^*$
return $(\mathbf{Check}(\Delta,\Theta,\mathcal{R}),\ \bigsqcup_{ij} T_{ij})$

$\mathbf{Check}(\Delta,\Theta,\mathcal{R}) = \theta := \mathtt{tt}$
for $S \in \lceil\Delta\rceil \cup \lceil\Theta\rceil/\mathcal{R}$ **do**
  $\theta := \theta \wedge (\Delta(S) \eqsim \Theta(S))$
return θ

Definition 7.1. *The class $\mathcal{L}$ of quantum modal formulae over Act_s, ranged over by ϕ, Φ, etc, is defined by the following grammar:*

$$\begin{aligned} \phi &::= \mathcal{G}_{\widetilde{q}} \mid \neg\phi \mid \bigwedge_{i\in I} \phi_i \mid \mathcal{G}.\phi \mid \langle\gamma\rangle\Phi \\ \Phi &::= Q_{\gtrsim\mathcal{A}}(\phi) \mid \bigwedge_{i\in I} \Phi_i \end{aligned}$$

where $\mathcal{G} \in CP_t(\mathcal{H})$, $\gamma \in Act_s$, and $\mathcal{A} \in CP(\mathcal{H})$. We call ϕ a snapshot formula *and Φ a* distribution formula*.*

The satisfaction relation $\models \subseteq EV \times (SN \cup \mathcal{D}ist_{\mathcal{H}}(SN)) \times \mathcal{L}$ is defined as the minimal relation satisfying

- $\psi,\mathfrak{t} \models \mathcal{G}_{\widetilde{q}}$ if $qv(t) \cap \widetilde{q} = \emptyset$, and $\mathcal{E} \approx_{\widetilde{q}} \mathcal{G}$, where $\mathfrak{t} = (\!|t,\mathcal{E}|\!)$;
- $\psi,\mathfrak{t} \models \neg\phi$ if $\psi,\mathfrak{t} \not\models \phi$;

- $\psi, \mathfrak{t} \models \bigwedge_{i\in I} \phi_i$ if $\psi, \mathfrak{t} \models \phi_i$ for each $i \in I$;
- $\psi, \mathfrak{t} \models \mathcal{G}.\phi$ if $\mathcal{G} \in CP_t(\mathcal{H}_{\overline{qv(t)}})$ and $(\!|t, \mathcal{G}\mathcal{E}|\!) \models \phi$, where $\mathfrak{t} = (\!|t, \mathcal{E}|\!)$;
- $\psi, \mathfrak{t} \models \langle\gamma\rangle\Phi$ if $\mathfrak{t} \xrightarrow{b,\gamma'} \Delta$ for some b, γ', and Δ, such that $\psi(b) = \mathtt{tt}$, $\gamma =_\psi \gamma'$, and $\psi, \Delta \models \Phi$;
- $\psi, \Delta \models Q_{\gtrsim\mathcal{A}}(\phi)$ if
$$\sum_{\mathfrak{t}\in\lceil\Delta\rceil} \{\Delta(\mathfrak{t}) : \psi, \mathfrak{t} \models \phi\} \gtrsim \mathcal{A};$$
- $\psi, \Delta \models \bigwedge_{i\in I} \Phi_i$ if $\psi, \Delta \models \Phi_i$ for each $i \in I$.

Definition 7.2. *Let ψ be an evaluation. We write $\mathfrak{t} =^{\psi}_{\mathcal{L}} \mathfrak{u}$ if for any $\phi \in \mathcal{L}$,*

$$\psi, \mathfrak{t} \models \phi \text{ if and only if } \psi, \mathfrak{u} \models \phi.$$

Similarly, $\Delta =^{\psi}_{\mathcal{L}} \Xi$ if for any $\Phi \in \mathcal{L}$,

$$\psi, \Delta \models \Phi \text{ if and only if } \psi, \Xi \models \Phi.$$

Lemma 7.3. *Let ψ be an evaluation, $\mathfrak{t}, \mathfrak{u} \in SN$, and $\Delta, \Xi \in \mathcal{D}ist_{\mathcal{H}}(SN)$.*

(1) If $\mathfrak{t} \neq^{\psi}_{\mathcal{L}} \mathfrak{u}$, then there exists $\phi \in \mathcal{L}$, such that $\psi, \mathfrak{t} \models \phi$ but $\psi, \mathfrak{u} \not\models \phi$;

(2) If $\Delta \neq^{\psi}_{\mathcal{L}} \Xi$, then there exists $\Phi \in \mathcal{L}$, such that $\psi, \Delta \models \Phi$ but $\psi, \Xi \not\models \Phi$.

Proof. (1) is easy as we have negation operator $\neg$ for state formulae. To prove (2), let $\Delta \neq^{\psi}_{\mathcal{L}} \Xi$, and Φ a distribution formula such that $\psi, \Delta \not\models \Phi$ but $\psi, \Xi \models \Phi$. We construct another distribution formula Φ' satisfying $\psi, \Delta \models \Phi'$ but $\psi, \Xi \not\models \Phi'$ by induction on the structure of Φ.

(i) $\Phi = Q_{\gtrsim\mathcal{A}}(\phi)$. Let

$$S = \{\mathfrak{u} \in SN : \psi, \mathfrak{u} \models \phi\} \qquad \text{and} \qquad \overline{S} = SN - S.$$

Then by definition, $\Xi(S) \gtrsim \mathcal{A}$ but $\Delta(S) \not\gtrsim \mathcal{A}$. Let $\mathcal{B} = \Delta(\overline{S})$ and $\Phi' = Q_{\gtrsim\mathcal{B}}(\neg\phi)$. Then we have trivially $\psi, \Delta \models \Phi'$. Now it suffices to show $\psi, \Xi \not\models \Phi'$. Otherwise, we have $\Xi(\overline{S}) \gtrsim \mathcal{B}$, and then

$$\mathcal{I}_{\mathcal{H}} \approx \Xi(S) + \Xi(\overline{S}) \gtrsim \mathcal{A} + \mathcal{B}.$$

On the other hand, we have

$$\mathcal{I}_{\mathcal{H}} \approx \Delta(S) + \Delta(\overline{S}) = \Delta(S) + \mathcal{B}.$$

Comparing the two formulae above, we conclude that $\Delta(S) \gtrsim \mathcal{A}$, a contradiction.

(ii) $\Phi = \bigwedge_{i\in I} \Phi_i$. Then by definition, $\psi, \Xi \models \Phi_i$ for each $i \in I$ but $\psi, \Delta \not\models \Phi_{i_0}$ for some $i_0 \in I$. By induction we have Φ'_{i_0} such that $\psi, \Delta \models \Phi'_{i_0}$ but $\psi, \Xi \not\models \Phi'_{i_0}$. For any $i \neq i_0$, let $\Phi'_i = \Phi_i$ if $\psi, \Delta \models \Phi_i$, and otherwise it is determined by applying induction on Φ_i. Let $\Phi' = \bigwedge_{i\in I} \Phi'_i$. Then $\psi, \Delta \models \Phi'$ but $\psi, \Xi \not\models \Phi'$. □

With this lemma, we can show that the logic $\mathcal{L}$ exactly characterises the behaviours of quantum snapshots up to symbolic bisimilarity.

Theorem 7.4. *Let $\mathfrak{t}$ and $\mathfrak{u}$ be two snapshots and $b \in BExp$. Then $\mathfrak{t} \sim^b \mathfrak{u}$ if and only if for any evaluation ψ, $\psi(b) = \mathtt{tt}$ implies $\mathfrak{t} =^{\psi}_{\mathcal{L}} \mathfrak{u}$.*

Proof. We first prove the necessity part. For any $\phi, \Phi \in \mathcal{L}$, it suffices to prove the following two properties:

$$\forall\ \mathtt{t}, \mathtt{u}, \psi,\ \text{if } \mathtt{t} \sim^b \mathtt{u} \text{ and } \psi(b) = \mathtt{tt} \text{ then } \psi, \mathtt{t} \models \phi \Leftrightarrow \psi, \mathtt{u} \models \phi,$$
$$\forall\ \Delta, \Xi, \psi,\ \text{if } \Delta \sim^b \Xi \text{ and } \psi(b) = \mathtt{tt} \text{ then } \psi, \Delta \models \Phi \Leftrightarrow \psi, \Xi \models \Phi.$$

We proceed by mutual induction on the structures of ϕ and Φ. Take arbitrarily $\mathtt{t} \sim^b \mathtt{u}$, $\Delta \sim^b \Xi$, and $\psi(b) = \mathtt{tt}$. Let $\mathtt{t} = (\!|t, \mathcal{E}|\!)$, $\mathtt{u} = (\!|u, \mathcal{F}|\!)$, $\psi, \mathtt{t} \models \phi$, and $\psi, \Delta \models \Phi$. There are seven cases to consider:

- $\phi = \mathcal{G}_{\widetilde{q}}$. Then $qv(t) \cap \widetilde{q} = \emptyset$ and $\mathcal{E} \eqsim_{\widetilde{q}} \mathcal{G}$. Since $\mathtt{t} \sim^b \mathtt{u}$ and b is satisfiable, we have $qv(t) = qv(u)$ and $\mathcal{E} \eqsim_{\overline{qv(t)}} \mathcal{F}$. Thus $qv(u) \cap \widetilde{q} = \emptyset$, and $\mathcal{F} \eqsim_{\widetilde{q}} \mathcal{G}$ from the fact that $\widetilde{q} \subseteq \overline{qv(t)}$. Then $\psi, \mathtt{u} \models \mathcal{G}_{\widetilde{q}}$ follows.

- $\phi = \neg\phi'$. Then $\psi, \mathtt{t} \not\models \phi'$. By induction we have $\psi, \mathtt{u} \not\models \phi'$, and $\psi, \mathtt{u} \models \phi$.

- $\phi = \bigwedge_{i \in I} \phi_i$. Then $\psi, \mathtt{t} \models \phi_i$ for each $i \in I$. By induction we have $\psi, \mathtt{u} \models \phi_i$, and $\psi, \mathtt{u} \models \phi$.

- $\phi = \mathcal{G}.\phi'$. Then $\mathcal{G} \in CP_t(\mathcal{H}_{\overline{qv(t)}})$ and $\psi, \mathcal{G}(\mathtt{t}) \models \phi'$. Since $\mathtt{t} \sim^b \mathtt{u}$, we have $\mathcal{G}(\mathtt{t}) \sim^b \mathcal{G}(\mathtt{u})$ by Proposition 6.3, and $qv(t) = qv(u)$. By induction we have $\psi, \mathcal{G}(\mathtt{u}) \models \phi'$, and $\psi, \mathtt{u} \models \phi$.

- $\phi = \langle\gamma\rangle\Phi'$. Then $\mathtt{t} \xrightarrow{b_1, \gamma'} \Delta'$ for some b_1, γ', and Δ' such that $\psi(b_1) = \mathtt{tt}$, $\gamma =_\psi \gamma'$, and $\psi, \Delta' \models \Phi'$. Since $\mathtt{t} \sim^b \mathtt{u}$, there exists a collection of booleans B such that $b \wedge b_1 \rightarrow \bigvee B$ and $\forall\ b' \in B$, $\exists b(b'), \gamma(b')$ with $b' \rightarrow b(b')$, $\gamma' =_{b'} \gamma(b')$, $\mathtt{u} \xrightarrow{b(b'), \gamma(b')} \Xi'$, and $\Delta' \sim^{b'} \Xi'$. Note that $\psi(b \wedge b_1) = \mathtt{tt}$. We can find a $b' \in B$ such that $\psi(b') = \mathtt{tt}$. Thus $\psi(b(b')) = \mathtt{tt}$, and $\gamma =_\psi \gamma(b')$. Furthermore, by induction we have $\psi, \Xi' \models \Phi'$ from $\Delta' \sim^{b'} \Xi'$ and $\psi, \Delta' \models \Phi'$. So $\psi, \mathtt{u} \models \langle\gamma\rangle\Phi'$.

- $\Phi = Q_{\gtrsim_{\mathcal{A}}}(\phi')$. Let $S = \{\mathtt{t} \in SN : \psi, \mathtt{t} \models \phi'\}$. Then by definition, $\Delta(S) \gtrsim \mathcal{A}$. Furthermore, by induction we can see that S is the disjoint union of some equivalence classes $S_1, \cdots, S_k$ of $\sim^b$. Thus
$$\Xi(S) = \Xi(S_1) + \cdots + \Xi(S_k) \eqsim \Delta(S_1) + \cdots + \Delta(S_k) = \Delta(S) \gtrsim \mathcal{A}$$
where the $\eqsim$ equality is derived from the assumption that $\Delta \sim^b \Xi$.

- $\Phi = \bigwedge_{i \in I} \Phi_i$. Then $\psi, \Delta \models \Phi_i$ for each $i \in I$. By induction we have $\psi, \Xi \models \Phi_i$, and $\psi, \Xi \models \Phi$.

By symmetry, we also have $\psi, \mathtt{u} \models \phi$ implies $\psi, \mathtt{t} \models \phi$ and $\psi, \Xi \models \Phi$ implies $\psi, \Delta \models \Phi$. That completes the proof of the necessity part.

We now turn to the sufficiency part. By Lemma 5.21, we need only to prove that $\mathtt{t} =^\psi_{\mathcal{L}} \mathtt{u}$ implies $(\mathtt{t}\psi)(\rho) \dot{\sim} (\mathtt{u}\psi)(\rho)$ for all $\rho \in \mathcal{D}(\mathcal{H})$. Let

$$\mathcal{R} = \{((\mathtt{t}\psi)(\rho), (\mathtt{u}\psi)(\rho)) : \rho \in \mathcal{D}(\mathcal{H}), \psi \in EV, \text{ and } \mathtt{t} =^\psi_{\mathcal{L}} \mathtt{u}\}$$

It suffices to show that $\mathcal{R}$ is an open bisimulation. Suppose $(\mathtt{t}\psi)(\rho)\mathcal{R}(\mathtt{u}\psi)(\rho)$. Then $\mathtt{t} =^\psi_{\mathcal{L}} \mathtt{u}$, and

$$qv(t\psi) = qv(t) = qv(u) = qv(u\psi).$$

We further claim that $\mathrm{tr}_{qv(t)}\mathcal{E}(\rho) = \mathrm{tr}_{qv(t)}\mathcal{F}(\rho)$. Otherwise there exists $\widetilde{q} \subseteq \overline{qv(t)}$ such that $\mathcal{E} \not\eqsim_{\widetilde{q}} \mathcal{F}$. Then $\psi, \mathtt{t} \models \mathcal{E}_{\widetilde{q}}$ while $\psi, \mathtt{u} \not\models \mathcal{E}_{\widetilde{q}}$, a contradiction.

Now let $(\mathtt{t}\psi)(\rho) \xmapsto{\alpha} \mu$. By Lemma 5.5 we have $\mathtt{t} \xrightarrow{b_1, \gamma} \Delta_\mu$ such that $\psi(b_1) = \mathtt{tt}$, $\mu = (\Delta_\mu\psi')(\rho)$, and

(1) if $\alpha = c?v$ then $\gamma = c?x$ for some $x \notin fv(t)$, and $\psi' = \psi\{v/x\}$,

(2) otherwise, $\gamma =_\psi \alpha$ and $\psi' = \psi$.

Let

$$\mathcal{K} = \{\nu \in Dist(Con) : (\mathfrak{u}\psi)(\rho) \stackrel{\alpha}{\longmapsto} \nu \text{ and } \mu\ \mathcal{R}\nu\}.$$

For any $\nu \in \mathcal{K}$, by Lemma 5.5 we have $\mathfrak{u} \stackrel{b(\Xi_\nu),\gamma(\Xi_\nu)}{\longrightarrow} \Xi_\nu$ such that $\psi(b(\Xi_\nu)) = \mathtt{tt}$, $\nu = (\Xi_\nu\psi'')(\rho)$, and

(1) if $\alpha = c?v$ then $\gamma(\Xi_\nu) = c?x$ for some $x \notin fv(u)$, and $\psi'' = \psi\{v/x\}$,

(2) otherwise, $\gamma(\Xi_\nu) =_\psi \alpha$ and $\psi'' = \psi$.

Here again, to ease the notations we only consider the case where for each Ξ, there is at most one pair, denoted $(b(\Xi), \gamma(\Xi))$, such that $\mathfrak{u} \stackrel{b(\Xi),\gamma(\Xi)}{\longrightarrow} \Xi$. Furthermore, by α-conversion, we can always take $\gamma(\Xi_\nu) =_\psi \gamma$ and $\psi'' = \psi'$. For any $\nu \in \mathcal{K}$, we claim $\Delta_\mu \neq^\psi_{\mathcal{L}} \Xi_\nu$. Otherwise, since $\mu = (\Delta_\mu\psi')(\rho)$ and $\nu = (\Xi_\nu\psi')(\rho)$, we have $\mu\mathcal{R}\nu$, a contradiction. Thus, from Lemma 7.3 (2), there exists $\Phi_\nu \in \mathcal{L}$ such that $\psi, \Delta_\mu \models \Phi_\nu$ but $\psi, \Xi_\nu \not\models \Phi_\nu$. Let

$$\Phi_\mu = \bigwedge\{\Phi_\nu : \nu \in \mathcal{K}\} \text{ and } \phi = \langle\gamma\rangle\Phi_\mu.$$

Then $\psi, \Delta_\mu \models \Phi_\mu$ and $\psi, \mathfrak{t} \models \phi$. Since $\mathfrak{t} =^\psi_{\mathcal{L}} \mathfrak{u}$, we have $\psi, \mathfrak{u} \models \phi$ too. That is, there exists Θ such that $\psi(b(\Theta)) = \mathtt{tt}$, $\gamma =_\psi \gamma(\Theta)$, and $\psi, \Theta \models \Phi_\mu$. Now by Lemma 5.6, we have $(\mathfrak{u}\psi)(\rho) \stackrel{\alpha'}{\longmapsto} \omega = (\Theta\psi''')(\rho)$ such that

(1) if $\gamma(\Theta) = c?x$ then $\alpha' = c?v$ for some $v \in \mathsf{Real}$, and $\psi''' = \psi\{v/x\}$,

(2) otherwise, $\alpha' =_\psi \gamma(\Theta)$ and $\psi''' = \psi$.

By transition rule $C\text{-}Inp_c$, we can alway choose $\alpha' = \alpha$, and $\psi''' = \psi'$. We claim that $\omega \notin \mathcal{K}$. Otherwise, if $\omega \in \mathcal{K}$ then $\psi, \Xi_\omega \not\models \Phi_\omega$, and $\psi, \Xi_\omega \not\models \Phi_\mu$ as well. This is a contradiction since by assumption, $\Xi_\omega = \Theta$. So $\omega \notin \mathcal{K}$, and $\mu\mathcal{R}\omega$ as required.

Finally, we prove that $\mathcal{R}$ is closed under super-operator application. To this end, we only need to show that $=^\psi_{\mathcal{L}}$ is ; that is, for any $\mathcal{G} \in CP_t(\mathcal{H}_{\overline{qv(t)}})$, $\mathfrak{t} =^\psi_{\mathcal{L}} \mathfrak{u}$ implies $\mathcal{G}(\mathfrak{t}) =^\psi_{\mathcal{L}} \mathcal{G}(\mathfrak{u})$. Suppose $\mathfrak{t} =^\psi_{\mathcal{L}} \mathfrak{u}$ and let ϕ be a formula such that $\psi, \mathcal{G}(\mathfrak{t}) \models \phi$. Then $\psi, \mathfrak{t} \models \mathcal{G}.\phi$. It follows from $\mathfrak{t} =^\psi_{\mathcal{L}} \mathfrak{u}$ that $qv(t) = qv(u)$ and $\psi, \mathfrak{u} \models \mathcal{G}.\phi$. Therefore, $\psi, \mathcal{G}(\mathfrak{u}) \models \phi$. By symmetry if ϕ is satisfied by $\psi, \mathcal{G}(\mathfrak{u})$ then it is also satisfied by $\psi, \mathcal{G}(\mathfrak{t})$. In other words, we have $\mathcal{G}(\mathfrak{t}) =^\psi_{\mathcal{L}} \mathcal{G}(\mathfrak{u})$. Then $\mathcal{R}$ is an open bisimulation by Proposition 5 of [6]. □

For any $t, u \in \mathcal{T}$ and $b \in BExp$, we write $t =^b_{\mathcal{L}} u$ if for any evaluation ψ, $\psi(b) = \mathtt{tt}$ implies $(\!|t, \mathcal{I}_{\mathcal{H}}|\!) =^\psi_{\mathcal{L}} (\!|u, \mathcal{I}_{\mathcal{H}}|\!)$. Then we have the following theorem:

Theorem 7.5. *For any $t, u \in \mathcal{T}$, $t \sim^b u$ if and only if $t =^b_{\mathcal{L}} u$.*

8 Conclusion and further work

The main contribution of this paper is a notion of symbolic bisimulation for qCCS, a quantum extension of classical value-passing CCS. By giving the operational semantics of qCCS directly by means of the super-operators a process can perform, we are able to assign to each (non-recursively defined) quantum process a *finite* super-operator weighted labelled transition system, comparing to the *infinite* probabilistic labelled transition system in previous literature. We prove that the symbolic bisimulation in this paper coincides with the open bisimulation in [6], thus providing a practical way to decide the latter. We also design an algorithm to check symbolic ground bisimulation, which is applicable to reasoning about the correctness of existing quantum communication protocols. A modal logic characterisation for the symbolic bisimulation is also developed.

A natural extension of the current paper is to study symbolic weak bisimulation where the invisible actions, caused by internal (classical and quantum) communication as well as quantum operations, are abstracted away. To achieve this, we may need to define symbolic weak transitions similar to those proposed in [8] and [6]. Note that one of the distinct features of weak transitions for probabilistic processes is the so-called left decomposibility; that is, if $\mu \Longrightarrow \nu$ and $\mu = \sum_{i\in I} p_i\mu_i$ is a probabilistic decomposition of μ, then ν can be decomposed into $\sum_{i\in I} p_i\nu_i$ accordingly such that $\mu_i \Longrightarrow \nu_i$ for each $i \in I$. This property is useful in proving the transitivity of bisimilarity. However, it is not satisfied by symbolic transitions defined in this paper, since, in general, a super-operator does not have an inverse. Therefore, we will have to explore other ways of defining weak symbolic transitions, which is one of the research directions we are now pursing.

We have presented in this paper, for the first time in literature to the best of our knowledge, the notion of super-operator weighted labelled transition systems, which serves the semantic model for qCCS and plays an important role in describing and reasoning about quantum processes. For the next step, we are going to explore the possibility of model checking quantum communication protocols based on this model. As is well known, one of the main challenges for quantum model checking is that the set of all quantum states, traditionally regarded as the underlying state space of the models to be checked, is a continuum, so that the techniques of classical model checking, which normally works only for finite state space, cannot be applied directly. Gay et al. [9, 10, 17] provided a solution for this problem by restricting the state space to a set of finitely describable states called stabiliser states, and restricting the quantum operations applied on them to the class of Clifford group. By doing this, they were able to obtain an efficient model checker for quantum protocols, employing purely classical algorithms. The limit of their approach is obvious: it can only check the (partial) behaviours of a protocol on stabiliser states, and does not work for general protocols.

Our approach of treating both classical data and quantum operations in a symbolic way provides an efficient and compact way to describe behaviours of a quantum protocol without resorting to the underlying quantum states. In this model, all existing quantum protocols have finite state spaces, and consequently, classical model checking techniques will be easily adapted to verifying quantum protocols.

Acknowledgement

This work was supported by Australian ARC grants DP110103473 and FT100100218.

References

[1] C. H. Bennett and G. Brassard. Quantum cryptography: Public-key distribution and coin tossing. In *Proceedings of the IEEE International Conference on Computer, Systems and Signal Processing*, pages 175–179, 1984.

[2] C. H. Bennett, G. Brassard, C. Crepeau, R. Jozsa, A. Peres, and W. Wootters. Teleporting an unknown quantum state via dual classical and epr channels. *Physical Review Letters*, 70:1895–1899, 1993.

[3] C. H. Bennett and S. J. Wiesner. Communication via one- and two-particle operators on einstein-podolsky-rosen states. *Physical Review Letters*, 69(20):2881–2884, 1992.

[4] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[5] T. A. S. Davidson. *Formal Verification Techniques using Quantum Process Calculus*. PhD thesis, 2011.

[6] Yuxin Deng and Yuan Feng. Open bisimulation for quantum processes. Manuscript. Available at http://arxiv.org/abs/1201.0416.

[7] Y Feng, R Duan, Z Ji, and M Ying. Probabilistic bisimulations for quantum processes. *Information and Computation*, 205(11):1608–1639, 2007.

[8] Y Feng, R Duan, and M Ying. Bisimulations for quantum processes. In Mooly Sagiv, editor, *Proceedings of the 38th ACM Symposium on Principles of Programming Languages (POPL'11)*, pages 523–534, 2011.

[9] S Gay, R Nagarajan, and N Papanikolaou. Probabilistic model-checking of quantum protocols. In *Proceedings of the 2nd International Workshop on Developments in Computational Models*, 2006.

[10] S Gay, R Nagarajan, and N Papanikolaou. Qmc: A model checker for quantum systems. In *CAV 08*, pages 543–547. Springer, 2008.

[11] S. J. Gay and R. Nagarajan. Communicating quantum processes. In J. Palsberg and M. Abadi, editors, *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 145–157, 2005.

[12] M. Hennessy and H. Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138(2):353–389, 1995.

[13] P. Jorrand and M. Lalire. Toward a quantum process algebra. In P. Selinger, editor, *Proceedings of the 2nd International Workshop on Quantum Programming Languages, 2004*, page 111, 2004.

[14] K. Kraus. *States, Effects and Operations: Fundamental Notions of Quantum Theory*. Springer, Berlin, 1983.

[15] Marie Lalire. Relations among quantum processes: Bisimilarity and congruence. *Mathematical Structures in Computer Science*, 16(3):407–428, 2006.

[16] M. Nielsen and I. Chuang. *Quantum computation and quantum information*. Cambridge university press, 2000.

[17] N. K. Papanikolaou. *Model Checking Quantum Protocols*. PhD thesis, 2008.

[18] D. Sangiorgi. A theory of bisimulation for the -calculus. *Acta Informatica*, 33(1):69–97, 1996.

[19] J. von Neumann. *Mathematical Foundations of Quantum Mechanics*. Princeton University Press, Princeton, NJ, 1955.

[20] M Ying, Y Feng, R Duan, and Z Ji. An algebra of quantum processes. *ACM Transactions on Computational Logic (TOCL)*, 10(3):1–36, 2009.