Partition Refinement for Bisimilarity in CCP

Andres Aristizabal [*]
CNRS/DGA
andresaristi@lix.polytechnique.fr

Filippo Bonchi [†]
CNRS
filippo.bonchi@ens-lyon.fr

Luis Fernando Pino [*]
INRIA/DGA
luis.pino@lix.polytechnique.fr

Frank D. Valencia [*]
CNRS
frank.valencia@lix.polytechnique.fr

ABSTRACT

Saraswat's concurrent constraint programming (ccp) is a mature formalism for modeling processes (or programs) that interact by telling and asking constraints in a global medium, called the store. Bisimilarity is a standard behavioural equivalence in concurrency theory, but a well-behaved notion of bisimilarity for ccp has been proposed only recently. When the state space of a system is finite, the ordinary notion of bisimilarity can be computed via the well-known *partition refinement algorithm*, but unfortunately, this algorithm does not work for ccp bisimilarity.

In this paper, we propose a variation of the partition refinement algorithm for verifying ccp bisimilarity. To the best of our knowledge this is the first work providing for the automatic verification of program equivalence for ccp.

Keywords

Concurrent Constraint Programming, Bisimilarity, Partition Refinement.

1. INTRODUCTION

Bisimilarity is the main representative of the so called behavioral equivalences, i.e., equivalence relations that determine when two processes (e.g., the specification and the implementation) behave in the same way. Many efficient algorithms and tools for bisimilarity checking have been developed [15, 8, 9]. Among these, the *partition refinement algorithm* [10] is the best known: first it generates the state space of a labeled transition system (LTS), i.e., the set of states reachable through the transitions; then, it creates a partition equating all states and afterwards, iteratively, refines these partitions by splitting non equivalent states. At the end, the resulting partition equates all and only bisimilar states.

Concurrent Constraint Programming (ccp) [13] is a formalism that combines the traditional algebraic and operational view of process calculi with a declarative one based upon first-order logic. In ccp, processes (agents or programs) interact by *adding* (or *telling*) and *asking* information (namely, constraints) in a medium (*store*).

Problem. The ccp formalism has been widely investigated and tested in terms of theoretical studies and the implementation of several ccp programming languages. From the *applied computing point of view*, however, ccp lacks algorithms and tools to automatically verify program equivalence. In this paper, we will give the first step towards automatic verification of ccp program equivalences by providing an algorithm to automatically verify a ccp process (or program) equivalence from the literature. Namely, *saturated barbed bisimilarity*.

Saturated barbed bisimilarity ($\dot{\sim}_{sb}$) for ccp was introduced in [1]. Two configurations are equivalent according to $\dot{\sim}_{sb}$ if (i) they have the same store, (ii) their transitions go into equivalent states and (iii) they are still equivalent when adding an arbitrary constraint to the store. In [1], a weak variant of $\dot{\sim}_{sb}$ is shown to be *fully abstract* w.r.t. the standard observational equivalence of [14].

Unfortunately, the standard partition refinement algorithm does not work for $\dot{\sim}_{sb}$ because condition (iii) requires to check all possible constraints that might be added to the store. In this paper we introduce a modified partition refinement algorithm for $\dot{\sim}_{sb}$.

We closely follow the approach in [5] that studies the notion of saturated bisimilarity from a more general perspective and proposes an abstract checking procedure.

We first define a *derivation relation* $\vdash_D$ amongst the transitions of ccp processes: $\gamma \xrightarrow{\alpha_1} \gamma_1 \vdash_D \gamma \xrightarrow{\alpha_2} \gamma_2$ which intuitively means that the latter transition is a logical consequence of the former.

Then we introduce the notion of *redundant transition*. Intuitively, a transition $\gamma \xrightarrow{\alpha_2} \gamma_2$ is redundant if there exists another transition $\gamma \xrightarrow{\alpha_1} \gamma_1$ that logically implies it, that is $\gamma \xrightarrow{\alpha_1} \gamma_1 \vdash_D \gamma \xrightarrow{\alpha_2} \gamma_2'$ and $\gamma_2 \dot{\sim}_{sb} \gamma_2'$. Now, if we consider the LTS 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 the standard partition refinement algorithm. But how can we decide which transitions are redundant, if redundancy itself depends on $\dot{\sim}_{sb}$?

Our solution 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

[*]Comète, LIX, Laboratoire de l'École Polytechnique associé à l'INRIA
[†]ENS Lyon, Université de Lyon, LIP (UMR 5668 CNRS ENS Lyon UCBL INRIA), 46 Allée d'Italie, 69364 Lyon, France

SAC'12 March 26-30, 2012, Riva del Garda, Italy.

transitions are updated according to the new computed partition.

A distinctive aspect of our algorithm is that in the initial partition, we insert not only the reachable states, but also extra ones which are needed to check for redundancy. We prove that these additional states are finitely many and thus the termination of the algorithm is guaranteed whenever the original LTS is finite (as it is the case of the standard partition refinement). Unfortunately, the number of these states might be exponential wrt the size of the original LTS, consequently the worst-case running time is exponential.

Contributions. We provide an algorithm that allows us to verify saturated barbed bisimilarity for ccp. To the best of our knowledge, this is the first algorithm for the automatic verification of a ccp program equivalence. This is done in Sections 3 and 4 by building upon the results of [5]. In Section 4.1 and 4.2, we also show the termination and the complexity of the algorithm. We have implemented the algorithm in c++ and the code is available at http://www.lix.polytechnique.fr/~andresaristi/strong/.

2. BACKGROUND

We now introduce the original standard partition refinement [10] and concurrent constraint programming (ccp).

Partition Refinement

In this section we recall the partition refinement algorithm introduced in [10] for checking bisimilarity over the states of a *labeled transition system* (LTS). Recall that an LTS can be intuitively seen as a graph where nodes represent states (of computation) and arcs represent transitions between states. A transition $P \xrightarrow{a} Q$ between P and Q labelled with a can be typically thought of as an evolution from P to Q provided that a condition a is met.

Let us now introduce some notation. Given a set S, a *partition* of S is a set of *blocks*, i.e., subsets of S, that are all disjoint and whose union is S. We write $\{B_1\} \dots \{B_n\}$ to denote a partition consisting of blocks $B_1, \dots, B_n$. A partition represents an equivalence relation where equivalent elements belong to the same block. We write $P \mathcal{P} Q$ to mean that P and Q are equivalent in the partition $\mathcal{P}$.

The *partition refinement algorithm* (see Alg. 1) checks the bisimilarity of a set of initial states IS as follows. First, it computes $IS^\star$, that is the set of all states that are reachable from IS. Then it creates the partition $\mathcal{P}^0$ where all the elements of $IS^\star$ belong to the same block (i.e., they are all equivalent). After the initialization, it iteratively refines the partitions by employing the function $\mathbf{F}$, defined as follows: for all partitions $\mathcal{P}$, $P\, \mathbf{F}(\mathcal{P})\, Q$ iff

- if $P \xrightarrow{a} P'$ then exists Q' s.t. $Q \xrightarrow{a} Q'$ and $P' \,\mathcal{P} Q'$.

The algorithm terminates whenever two consecutive partitions are equivalent. In such partition two states belong to the same block iff they are bisimilar.

Note that any iteration splits blocks and never fuses them. For this reason if $IS^\star$ is finite, the algorithm terminates in at most $|IS^\star|$ iterations.

Proposition 1. If $IS^\star$ is finite, then the algorithm terminates and the resulting partition equates all and only the bisimilar states.

Algorithm 1 `Partition-Refinement(`IS`)`

Initialization

1. $IS^\star$ is the set of all processes reachable from IS,
2. $\mathcal{P}^0 := \{IS^\star\}$,

Iteration $\mathcal{P}^{n+1} := \mathbf{F}(\mathcal{P}^n)$,
Termination If $\mathcal{P}^n = \mathcal{P}^{n+1}$ then return $\mathcal{P}^n$.

CCP

We now recall the concurrent constraint programming process calculus (ccp) [13, 14]. In particular its notion of barbed saturated bisimilarity ($\dot{\sim}_{sb}$) [1].

Constraint Systems. The ccp model is parametric in a *constraint system* specifying the structure and interdependencies of the information that processes can ask and tell. Following [14, 7], we regard a constraint system as a complete algebraic lattice structure.

Definition 1. A *constraint system* $\mathbf{C}$ is a complete algebraic lattice $(Con, Con_0, \sqsubseteq, \sqcup, true, false)$ where Con (the set of constraints) is a partially ordered set w.r.t. $\sqsubseteq$, Con_0 is the subset of *finite* elements of Con, $\sqcup$ is the lub operation, and $true$, $false$ are the least and greatest elements of Con, respectively.

To capture local variables [14] introduces cylindric constraint systems. A *cylindric constraint system* over an infinite set of variables V is a constraint system equipped with an operation $\exists_x$ for each $x \in V$. Broadly speaking $\exists_x$ has the properties of the existential quantification of x–e.g., $\exists_x c \sqsubseteq c, \exists_x \exists_y c = \exists_y \exists_x c$ and $\exists_x (c \sqcup \exists_x d) = \exists_x c \sqcup \exists_x d$. For the sake of space, we do not formally introduce this notion as it is not crucial to our work–see [14].

Given a partial order $(C, \sqsubseteq)$, we say that c is strictly smaller than d ($c \sqsubset d$) if $c \sqsubseteq d$ and $c \neq d$. We say that $(C, \sqsubseteq)$ is *well-founded* if there exists no infinite descending chains $\cdots \sqsubset c_n \sqsubset \cdots \sqsubset c_1 \sqsubset c_0$. For a set $A \subseteq C$, we say that an element $m \in A$ is *minimal* in A if for all $a \in A$, $a \not\sqsubset m$. We shall use $min(A)$ to denote the set of all minimal elements of A. Well-founded order and minimal elements are related by the following result.

Lemma 1. Let $(C, \sqsubseteq)$ be a well-founded order and $A \subseteq C$. If $a \in A$, then $\exists m \in min(A)$ s.t., $m \sqsubseteq a$.

Remark 1. We shall assume that the constraint system is well-founded and, for practical reasons, that its $\sqsubseteq$ 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, $\mathcal{P}(A)$ 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 $c \cap d$. By abusing the notation, we will often use a formula like $x < n$ to denote the corresponding constraint, i.e., the set of all assignments that map x in a number smaller than n.

Syntax. Let us presuppose a cylindric constraint system $\mathbf{C} = (Con, Con_0, \sqsubseteq, \sqcup, true, false)$ over a set of variables Var. The ccp processes are given by the following syntax,

$$P, Q ::= \mathbf{0} \mid \mathbf{tell}(c) \mid \mathbf{ask}(c) \rightarrow P \mid P \parallel Q \mid P + Q \mid \exists_x^c P \mid p(\vec{z})$$

where $c \in Con_0$, $x \in Var$, $\vec{z} \in Var^*$.

Intuitively, $\mathbf{0}$ 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; $\exists_x^c$ is a *hiding operator*, namely it indicates that in $\exists_x^c P$ the variable x is *local* to P and c is some local information (*local store*)

$$\text{R1}\quad \langle \mathbf{tell}(c), d\rangle \longrightarrow \langle \mathbf{0}, d \sqcup c\rangle \qquad \text{R2}\quad \frac{c \sqsubseteq d}{\langle \mathbf{ask}\ (c)\ \rightarrow\ P, d\rangle \longrightarrow \langle P, d\rangle} \qquad \text{R5}\quad \frac{\langle P, e \sqcup \exists_x d\rangle \longrightarrow \langle P', e' \sqcup \exists_x d\rangle}{\langle \exists_x^e P, d\rangle \longrightarrow \langle \exists_x^{e'} P', d \sqcup \exists_x e'\rangle}$$

$$\text{R3}\quad \frac{\langle P, d\rangle \longrightarrow \langle P', d'\rangle}{\langle P \parallel Q, d\rangle \longrightarrow \langle P' \parallel Q, d'\rangle} \qquad \text{R4}\quad \frac{\langle P, d\rangle \longrightarrow \langle P', d'\rangle}{\langle P\ +\ Q, d\rangle \longrightarrow \langle P', d'\rangle} \qquad \text{R6}\quad \frac{\langle P[\vec{z}/\vec{x}], d\rangle \longrightarrow \gamma'}{\langle p(\vec{z}), d\rangle \longrightarrow \gamma'} \text{ for } p(\vec{x}) \stackrel{\text{def}}{=} P$$

Table 1: Reduction semantics for ccp (the symmetric rules for R3 and R4 are omitted)

possibly containing x. A process $p(\vec{z})$ is said to be a *procedure call* with identifier p and actual parameters $\vec{z}$. We presuppose that for each procedure call $p(z_1 \dots z_m)$ there exists a unique *procedure definition* possibly *recursive*, of the form $p(x_1 \dots x_m) \stackrel{\text{def}}{=} P$ where $fv(P) \subseteq \{x_1, \dots, x_m\}$.

Reduction Semantics. The operational semantics is given by transitions between configurations. 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 $\gamma, \gamma', \dots$ to denote the set of configurations. The operational model of ccp is given by the transition relation $\longrightarrow \subseteq Conf \times Conf$ defined in Tab. 1. Except for R5, these standard rules are self-explanatory. We include R5 for completeness of the presentation but it is not necessary to understand our work in the next section. For the sake of space we refer the interested reader to [1] for a detailed explanation of the rules.

Barbed Semantics. The authors in [1] introduced a barbed semantics for ccp. Barbed equivalences have been introduced in [11] for CCS, and become the standard behavioural equivalences for 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. A configuration $\gamma = \langle P, d\rangle$ is said to *satisfy* the barb c ($\gamma \downarrow_c$) iff $c \sqsubseteq d$.

Definition 2. A *barbed bisimulation* is a symmetric relation $\mathcal{R}$ on configurations s.t. whenever $(\gamma_1, \gamma_2) \in \mathcal{R}$:

(i) if $\gamma_1 \downarrow_c$ then $\gamma_2 \downarrow_c$,

(ii) if $\gamma_1 \longrightarrow \gamma_1'$ then there exists γ_2' s.t. $\gamma_2 \longrightarrow \gamma_2'$ and $(\gamma_1', \gamma_2') \in \mathcal{R}$.

γ_1 and γ_2 are *barbed bisimilar* ($\gamma_1 \dot{\sim}_b \gamma_2$), if there exists a barbed bisimulation $\mathcal{R}$ s.t. $(\gamma_1, \gamma_2) \in \mathcal{R}$.

One can verify that $\dot{\sim}_b$ is an equivalence. However, it is not a *congruence*; i.e., it is not preserved under arbitrary contexts (the interested reader can check Ex. 7 in [1]). An elegant solution to modify bisimilarity for obtaining a congruence consists in *saturated bisimilarity* [4, 3] (pioneered by [12]). The basic idea is simple: saturated bisimulations are closed w.r.t. 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. A *saturated barbed bisimulation* is a symmetric relation $\mathcal{R}$ on configurations s.t. whenever $(\gamma_1, \gamma_2) \in \mathcal{R}$ with $\gamma_1 = \langle P, d\rangle$ and $\gamma_2 = \langle Q, e\rangle$:

(i) if $\gamma_1 \downarrow_c$ then $\gamma_2 \downarrow_c$,

(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 $a \in Con_0$, $(\langle P, d \sqcup a\rangle, \langle Q, e \sqcup a\rangle) \in \mathcal{R}$.

γ_1 and γ_2 are *saturated barbed bisimilar* ($\gamma_1 \dot{\sim}_{sb} \gamma_2$) if there exists a saturated barbed bisimulation $\mathcal{R}$ s.t. $(\gamma_1, \gamma_2) \in \mathcal{R}$.

Example 2. Take $T = \mathbf{tell}(true)$, $P = \mathbf{ask}\ (x < 7)\ \rightarrow\ T$ and $Q = \mathbf{ask}\ (x < 5)\ \rightarrow\ T$. You can see that $\langle P, true\rangle \not\dot{\sim}_{sb} \langle Q, true\rangle$, since $\langle P, x < 7\rangle \longrightarrow$, while $\langle Q, x < 7\rangle \not\longrightarrow$. Consider now the configuration $\langle P + Q, true\rangle$ 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 behaviour of Q is somehow absorbed by the behaviour of P.

Example 3. Since $\dot{\sim}_{sb}$ is upward closed, $\langle P+Q, z < 5\rangle \dot{\sim}_{sb} \langle P, z < 5\rangle$ follows immediately by the previous example. Now take $R = \mathbf{ask}\ (z < 5)\ \rightarrow\ (P + Q)$ and $S = \mathbf{ask}\ (z < 7)\ \rightarrow\ P$. By analogous arguments of the previous example, one can show that $\langle R + S, true\rangle \dot{\sim}_{sb} \langle S, true\rangle$.

Example 4. Take $T' = \mathbf{tell}(y = 1)$, $Q' = \mathbf{ask}\ (x < 5)\ \rightarrow\ T'$ and $R' = \mathbf{ask}\ (z < 5)\ \rightarrow\ P + Q'$. Observe that $\langle P + Q', z < 5\rangle \not\dot{\sim}_{sb} \langle P, z < 5\rangle$ and that $\langle R' + S, true\rangle \not\dot{\sim}_{sb} \langle S, true\rangle$, since $\langle P + Q', x < 5\rangle$ and $\langle R' + S, true\rangle$ can reach a store containing the constraint $y = 1$.

In [1], a *weak variant* of $\dot{\sim}_{sb}$ is introduced and it is shown that it is *fully abstract* w.r.t. the standard observational equivalence of [14]. In this paper, we will show an algorithm for checking $\dot{\sim}_{sb}$ and we leave, as future work, to extend it for the weak semantics.

Nevertheless, the equivalence $\dot{\sim}_{sb}$ would seem hard to (automatically) check because of the upward-closure (namely, the quantification over all possible $a \in Con_0$ in condition (iii)) of Def. 3. The work in [1] deals with this issue by refining the notion of transition by adding to it a *label* that carries additional information about the constraints that cause the reduction.

Labeled Semantics. As explained in [1], in a transition of the form $\langle P, d\rangle \xrightarrow{\alpha} \langle P', d'\rangle$ the label α represents a *minimal* information (from the environment) that needs to be added to the store d to evolve from $\langle P, d\rangle$ into $\langle P', d'\rangle$, i.e., $\langle P, d \sqcup \alpha\rangle \longrightarrow \langle P', d'\rangle$. The labeled transition relation $\longrightarrow \subseteq Conf \times Con_0 \times Conf$ is defined by the rules in Tab. 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., $\alpha \in \min\{a \in Con_0 \mid c \sqsubseteq d \sqcup a\}$. Note that assuming that $(Con, \sqsubseteq)$ is well-founded (Sec. 2) is necessary to guarantee that α exists whenever $\{a \in Con_0 \mid c \sqsubseteq d \sqcup a\}$ is not empty. The other rules, except LR4, are easily seen to realize the above intuition. An explanation of LR5 is not needed to understand the present work. For the sake of space, we refer the reader to [1] for a more detailed explanation of these labeled rules. Fig. 1 illustrates the LTSs of our running example.

Syntactic Bisimilarity. When defining bisimilarity over a LTS, barbs are not usually needed because they can be somehow inferred

$$\text{LR1}\ \langle \mathbf{tell}(c), d\rangle \xrightarrow{true} \langle \mathbf{0}, d \sqcup c\rangle \quad \text{LR2}\ \frac{\alpha \in \min\{a \in Con_0 \mid c \sqsubseteq d \sqcup a\}}{\langle \mathbf{ask}\ (c)\ \rightarrow\ P, d\rangle \xrightarrow{\alpha} \langle P, d \sqcup \alpha\rangle} \quad \text{LR3}\ \frac{\langle P, d\rangle \xrightarrow{\alpha} \langle P', d'\rangle}{\langle P \parallel Q, d\rangle \xrightarrow{\alpha} \langle P' \parallel Q, d'\rangle} \quad \text{LR4}\ \frac{\langle P, d\rangle \xrightarrow{\alpha} \langle P', d'\rangle}{\langle P + Q, d\rangle \xrightarrow{\alpha} \langle P', d'\rangle}$$

$$\text{LR5}\ \frac{\langle P[z/x], e[z/x] \sqcup d\rangle \xrightarrow{\alpha} \langle P', e' \sqcup d \sqcup \alpha\rangle}{\langle \exists^e_x P, d\rangle \xrightarrow{\alpha} \langle \exists^{e'[x/z]}_x P'[x/z], \exists_x(e'[x/z]) \sqcup d \sqcup \alpha\rangle} \quad x \notin fv(e'), z \notin fv(P) \cup fv(e \sqcup d \sqcup \alpha) \quad \text{LR6}\ \frac{\langle P[\vec{z}/\vec{x}], d\rangle \xrightarrow{\alpha} \gamma'}{\langle p(\vec{z}), d\rangle \xrightarrow{\alpha} \gamma'} \quad \text{for } p(\vec{x}) \stackrel{\text{def}}{=} P$$

Table 2: Labeled semantics for ccp. (the symmetric rules for LR3 and LR4 are omitted)

$T = \mathbf{tell}(true) \quad P = \mathbf{ask}\ (x < 7)\ \rightarrow\ T \quad Q = \mathbf{ask}\ (x < 5)\ \rightarrow\ T \quad R = \mathbf{ask}\ (z < 5)\ \rightarrow\ (P + Q)$
$T' = \mathbf{tell}(y = 1) \quad S = \mathbf{ask}\ (z < 7)\ \rightarrow\ P \quad Q' = \mathbf{ask}\ (x < 5)\ \rightarrow\ T' \quad R' = \mathbf{ask}\ (z < 5)\ \rightarrow\ (P + Q')$

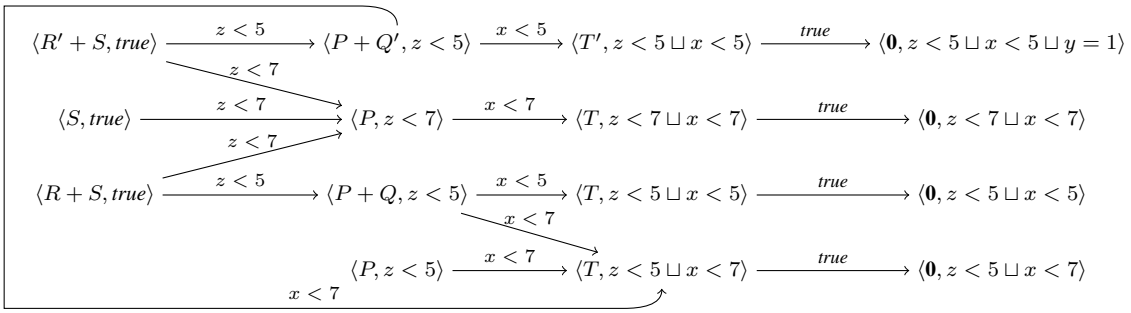

Figure 1: The labeled transition systems of the running example ($IS = \{\langle R' + S, true\rangle, \langle S, true\rangle, \langle R + S, true\rangle\}$).

from the labels of the transitions. For instance, in CCS, $P \downarrow_a$ iff $P \xrightarrow{a}$. However this is not the case of ccp: barbs cannot be removed from the definition of bisimilarity because they cannot be inferred from the transitions.

Taking into account the barbs, the obvious adaptation of labeled bisimilarity for ccp is the following:

Definition 4. [1] A *syntactic bisimulation* is a symmetric relation $\mathcal{R}$ on configurations s.t. whenever $(\gamma_1, \gamma_2) \in \mathcal{R}$:

(i) if $\gamma_1 \downarrow_c$ then $\gamma_2 \downarrow_c$,

(ii) if $\gamma_1 \xrightarrow{\alpha} \gamma_1'$ then $\exists \gamma_2'$ s.t. $\gamma_2 \xrightarrow{\alpha} \gamma_2'$ and $(\gamma_1', \gamma_2') \in \mathcal{R}$.

γ_1 and γ_2 are syntactically bisimilar, ($\gamma_1 \sim_S \gamma_2$) if there exists a syntactic bisimulation $\mathcal{R}$ s.t. $(\gamma_1, \gamma_2) \in \mathcal{R}$.

Unfortunately as shown in [1] $\sim_S$ is over-discriminating. As an example, consider the configurations $\langle P + Q, z < 5\rangle$ and $\langle P, z < 5\rangle$, whose LTS is shown in Fig. 1. They are not equivalent according to $\sim_S$. Indeed $\langle P + Q, z < 5\rangle \xrightarrow{x<5}$, while $\langle P, z < 5\rangle \not\xrightarrow{x<5}$. However they are equivalent according to $\dot{\sim}_{sb}$ (Ex. 3).

3. IRREDUNDANT BISIMILARITY

Syntactic bisimilarity is over-discriminating because of some *redundant transitions*. For instance, consider the transitions:
(a) $\langle P + Q, z < 5\rangle \xrightarrow{x<7} \langle T, z < 5 \sqcup x < 7\rangle$;
(b) $\langle P + Q, z < 5\rangle \xrightarrow{x<5} \langle T, z < 5 \sqcup x < 5\rangle$.
Transition (a) means that for all constraints e s.t. $x < 7 \sqsubseteq e$, (c)$\langle P + Q, z < 5 \sqcup e\rangle \longrightarrow \langle T, z < 5 \sqcup e\rangle$, while transition (b) means that the reduction (c) is possible for all e s.t. $x < 5 \sqsubseteq e$. Since $x < 7 \sqsubseteq x < 5$, transition (b) is "redundant", in the sense that its meaning is "logically derived" by transition (a).

The following notion captures the above intuition:

Definition 5. We say that $\langle P, c\rangle \xrightarrow{\alpha} \langle P_1, c'\rangle$ derives $\langle P, c\rangle \xrightarrow{\beta} \langle P_1, c''\rangle$, written $\langle P, c\rangle \xrightarrow{\alpha} \langle P_1, c'\rangle \vdash_D \langle P, c\rangle \xrightarrow{\beta} \langle P_1, c''\rangle$, iff there exists e s.t. the following conditions hold:
(i) $\beta = \alpha \sqcup e$ (ii) $c'' = c' \sqcup e$ (iii) $\alpha \neq \beta$

One can verify in the above example that (a) $\vdash_D$ (b). Notice that in order to check if $\langle P + Q, z < 5\rangle \dot{\sim}_{sb} \langle P, z < 5\rangle$, we could first remove the redundant transition (b) and then check $\sim_S$.

More generally, a naive approach to compute $\dot{\sim}_{sb}$ would be to first remove all those transitions that can be derived by others, and then apply the partition refinement algorithm. However, this approach would fail since it would distinguish $\langle R + S, true\rangle$ and $\langle S, true\rangle$ that, instead, are in $\dot{\sim}_{sb}$ (Ex. 3). Indeed, $\langle R + S, true\rangle$ can perform:
(e) $\langle R + S, true\rangle \xrightarrow{z<7} \langle P, z < 7\rangle$,
(f) $\langle R + S, true\rangle \xrightarrow{z<5} \langle P + Q, z < 5\rangle$,
while $\langle S, true\rangle \not\xrightarrow{z<5}$. Note that transition (f) cannot be derived by other transitions, since (e) $\not\vdash_D$ (f). Indeed, P is syntactically different from $P + Q$, even if they have the same behaviour when inserted in the store $z < 5$, i.e., $\langle P, z < 5\rangle \dot{\sim}_{sb} \langle P + Q, z < 5\rangle$ (Ex. 3). The transition (f) is also "redundant", since its behaviour "does not add anything" to the behaviour of (e).

Definition 6. Let $\mathcal{R}$ be a relation and $\gamma \xrightarrow{\alpha} \gamma_1$ and $\gamma \xrightarrow{\beta} \gamma_2$ be two transitions. We say that the former dominates the latter one in $\mathcal{R}$ (written $\gamma \xrightarrow{\alpha} \gamma_1 \succ_{\mathcal{R}} \gamma \xrightarrow{\beta} \gamma_2$) iff
(i) $\gamma \xrightarrow{\alpha} \gamma_1 \vdash_D \gamma \xrightarrow{\beta} \gamma_2'$ (ii) $(\gamma_2', \gamma_2) \in \mathcal{R}$
A transition is redundant w.r.t. $\mathcal{R}$ if it is dominated in $\mathcal{R}$ by another transition. Otherwise, it is irredundant.

Note that the transition $\gamma \xrightarrow{\beta} \gamma_2'$ might not be generated by the rules in Tab. 2, but simply derived by $\gamma \xrightarrow{\alpha} \gamma_1$ through $\vdash_D$. For

$\mathcal{P}^0 = \{\langle R'+S, true\rangle, \langle S, true\rangle, \langle R+S, true\rangle\}, \{\langle P+Q', z<5\rangle, \langle P+Q, z<5\rangle, \langle P, z<5\rangle\}, \{\langle P, z<7\rangle\}, \{\langle T', z<5\sqcup x<5\rangle, \langle T, z<5\sqcup x<5\rangle, \langle \mathbf{0}, z<5\sqcup x<5\rangle\}, \{\langle T, z<7\sqcup x<7\rangle, \langle \mathbf{0}, z<7\sqcup x<7\rangle\}, \{\langle T, z<5\sqcup x<7\rangle, \langle \mathbf{0}, z<5\sqcup x<7\rangle\}, \{\langle \mathbf{0}, z<5\sqcup x<5\sqcup y=1\rangle\}$

$\mathcal{P}^1 = \{\langle R'+S, true\rangle, \langle S, true\rangle, \langle R+S, true\rangle\}, \{\langle P+Q', z<5\rangle, \langle P+Q, z<5\rangle, \langle P, z<5\rangle\}, \{\langle P, z<7\rangle\}, \{\langle T', z<5\sqcup x<5\rangle\}, \{\langle T, z<5\sqcup x<5\rangle\}, \{\langle \mathbf{0}, z<5\sqcup x<5\rangle\}, \{\langle T, z<7\sqcup x<7\rangle\}, \{\langle \mathbf{0}, z<7\sqcup x<7\rangle\}, \{\langle T, z<5\sqcup x<7\rangle\}, \{\langle \mathbf{0}, z<5\sqcup x<7\rangle\}, \{\langle \mathbf{0}, z<5\sqcup x<5\sqcup y=1\rangle\}$

$\mathcal{P}^2 = \{\langle R'+S, true\rangle, \langle S, true\rangle, \langle R+S, true\rangle\}, \{\langle P+Q', z<5\rangle\}, \{\langle P+Q, z<5\rangle, \langle P, z<5\rangle\}, \{\langle P, z<7\rangle\}, \{\langle T', z<5\sqcup x<5\rangle\}, \{\langle T, z<5\sqcup x<5\rangle\}, \{\langle \mathbf{0}, z<5\sqcup x<5\rangle\}, \{\langle T, z<7\sqcup x<7\rangle\}, \{\langle \mathbf{0}, z<7\sqcup x<7\rangle\}, \{\langle T, z<5\sqcup x<7\rangle\}, \{\langle \mathbf{0}, z<5\sqcup x<7\rangle\}, \{\langle \mathbf{0}, z<5\sqcup x<5\sqcup y=1\rangle\}$

$\mathcal{P}^3 = \{\langle R'+S, true\rangle\}, \{\langle S, true\rangle, \langle R+S, true\rangle\}, \{\langle P+Q', z<5\rangle\}, \{\langle P+Q, z<5\rangle, \langle P, z<5\rangle\}, \{\langle P, z<7\rangle\}, \{\langle T', z<5\sqcup x<5\rangle\}, \{\langle T, z<5\sqcup x<5\rangle\}, \{\langle \mathbf{0}, z<5\sqcup x<5\rangle\}, \{\langle T, z<7\sqcup x<7\rangle\}, \{\langle \mathbf{0}, z<7\sqcup x<7\rangle\}, \{\langle T, z<5\sqcup x<7\rangle\}, \{\langle \mathbf{0}, z<5\sqcup x<7\rangle\}, \{\langle \mathbf{0}, z<5\sqcup x<5\sqcup y=1\rangle\}$

$\mathcal{P}^4 = \mathcal{P}^3$

Figure 2: The partitions computed by `CCP-Partition-Refinement(`$\{\langle R'+S, true\rangle, \langle S, true\rangle, \langle R+S, true\rangle\}$`).`

instance, transition (e) dominates (f) in $\dot{\sim}_{sb}$, because (e) $\vdash_D \langle R+S, true\rangle \xrightarrow{z<5} \langle P, z<5\rangle$ and $\langle P, z<5\rangle \dot{\sim}_{sb} \langle P+Q, z<5\rangle$.

Therefore, we could compute $\dot{\sim}_{sb}$, by removing all those transitions that are redundant wrt $\dot{\sim}_{sb}$. This, however, would lead us to a circular situation: How to decide which transitions are redundant when redundancy itself depends on $\dot{\sim}_{sb}$.

Our solution relies on the following definition that allows to compute bisimilarity and redundancy *at the same time*.

Definition 7. An irredundant bisimulation is a symmetric relation $\mathcal{R}$ on configurations s.t. whenever $(\gamma_1, \gamma_2) \in \mathcal{R}$:

(i) if $\gamma_1 \downarrow_c$ then $\gamma_2 \downarrow_c$,

(ii) if $\gamma_1 \xrightarrow{\alpha} \gamma_1'$ is irredundant in $\mathcal{R}$ then $\exists \gamma_2'$ s.t. $\gamma_2 \xrightarrow{\alpha} \gamma_2'$ and $(\gamma_1', \gamma_2') \in \mathcal{R}$.

γ_1 and γ_2 are irredundant bisimilar ($\gamma_1 \sim_I \gamma_2$), if there exists an irredundant bisimulation $\mathcal{R}$ s.t. $(\gamma_1, \gamma_2) \in \mathcal{R}$.

Theorem 1. $\sim_I = \dot{\sim}_{sb}$

PROOF. See [2]. □

4. PARTITION REFINEMENT FOR CCP

Recall that we mentioned in Sec. 2 that checking $\dot{\sim}_{sb}$ seems hard because of the quantification over all possible constraints. However, by using Theo. 1 we shall introduce an algorithm for checking $\dot{\sim}_{sb}$ by employing the notion of irredundant bisimulation.

The first novelty w.r.t. the standard partition refinement (Alg. 1) consists in using *barbs*. Since configurations satisfying different barbs are surely different, we can safely start with a partition that equates all and only those states satisfying the same barbs. Note that two configurations satisfy the same barbs iff they have the same store. Thus, we take as initial partition $\mathcal{P}^0 = \{IS^\star_{d_1}\} \ldots \{IS^\star_{d_n}\}$, where $IS^\star_{d_i}$ is the subset of the configurations of $IS^\star$ with store d_i.

Another difference is that instead of using the function $\mathbf{F}$ of Alg. 1, we refine the partitions by employing the function $\mathbf{IR}$ defined as follows: for all partitions $\mathcal{P}$, $\gamma_1\ \mathbf{IR}(\mathcal{P})\ \gamma_2$ iff

- if $\gamma_1 \xrightarrow{\alpha} \gamma_1'$ is irredundant in $\mathcal{P}$, then there exists γ_2' s.t. $\gamma_2 \xrightarrow{\alpha} \gamma_2'$ and $\gamma_1'\ \mathcal{P}\ \gamma_2'$.

It is now important to observe that in the computation of $\mathbf{IR}(\mathcal{P}^n)$, there might be involved also states that are not reachable from the initial states IS. For instance, consider the LTSs of $\langle S, true\rangle$ and $\langle R+S, true\rangle$ in Fig. 1. The state $\langle P, z<5\rangle$ is not reachable but is needed to check if $\langle R+S, true\rangle \xrightarrow{z<5} \langle P+Q, z<5\rangle$ is redundant (look at the example after Def. 6).

For this reason, we have also to change the initialization step of our algorithm, by including in the set $IS^\star$ all the states that are needed to check redundancy. This is done, by using the following closure rules.

$$(\text{IS})\ \frac{\gamma \in IS}{\gamma \in IS^\star} \qquad (\text{RS})\ \frac{\gamma_1 \in IS^\star \quad \gamma_1 \xrightarrow{\alpha} \gamma_2}{\gamma_2 \in IS^\star}$$

$$(\text{RD})\ \frac{\gamma \in IS^\star \quad \gamma \xrightarrow{\alpha_1} \gamma_1 \quad \gamma \xrightarrow{\alpha_2} \gamma_2 \quad \gamma \xrightarrow{\alpha_1} \gamma_1 \vdash_D \gamma \xrightarrow{\alpha_2} \gamma_3}{\gamma_3 \in IS^\star}$$

The rule (RD) adds all the states that are needed to check redundancy. Indeed, if γ can perform both $\xrightarrow{\alpha_1}\ \gamma_1$ and $\xrightarrow{\alpha_2}\ \gamma_2$ s.t. $\gamma \xrightarrow{\alpha_1} \gamma_1 \vdash_D \gamma \xrightarrow{\alpha_2} \gamma_3$, then $\gamma \xrightarrow{\alpha_2} \gamma_2$ would be redundant whenever $\gamma_2 \dot{\sim}_{sb} \gamma_3$.

Algorithm 2 `CCP-Partition-Refinement(`IS`)`

Initialization

1. Compute $IS^\star$ with the rules (IS), (RS) and (RD),
2. $\mathcal{P}^0 := \{IS^\star_{d_1}\} \ldots \{IS^\star_{d_n}\}$,

Iteration $\mathcal{P}^{n+1} := \mathbf{IR}(\mathcal{P}^n)$
Termination If $\mathcal{P}^n = \mathcal{P}^{n+1}$ then return $\mathcal{P}^n$.

Fig. 2 shows the partitions computed by the algorithm with initial states $\langle R'+S, true\rangle$, $\langle S, true\rangle$ and $\langle R+S, true\rangle$. Note that, as expected, in the final partition $\langle R+S, true\rangle$ and $\langle S, true\rangle$ belong to the same block, while $\langle R'+S, true\rangle$ belong to a different one (meaning that the former two are saturated bisimilar, while $\langle R'+S, true\rangle$ is different). In the initial partition all states with the same store are equated. In $\mathcal{P}^1$, the blocks are split by considering the outgoing transitions: all the final states are distinguished (since they cannot perform any transitions) and $\langle T', z<5 \sqcup x<5\rangle$ is distinguished from $\langle T, z<5 \sqcup x<5\rangle$. All the other blocks are not divided, since all the transitions with label $x<5$ are redundant in $\mathcal{P}^0$ (since $\langle P, z<5\rangle \mathcal{P}^0 \langle P+Q', z<5\rangle$, $\langle P, z<5\rangle \mathcal{P}^0 \langle P+Q, z<5\rangle$ and $\langle T', z<5\sqcup x<5\rangle \mathcal{P}^0 \langle T, z<5\sqcup x<5\rangle$). Then, in $\mathcal{P}^2$, $\langle P+Q', z<5\rangle$ is distinguished from $\langle P, z<5\rangle$ since the transition $\langle P+Q', z<5\rangle \xrightarrow{z<5}$ is not redundant anymore in $\mathcal{P}^1$ (since $\langle T', z<5\sqcup x<5\rangle$ and $\langle T, z<5\sqcup x<5\rangle$ belong to different blocks in $\mathcal{P}^1$). Then in $\mathcal{P}^3$, $\langle R'+S, true\rangle$ is distinguished from $\langle S, true\rangle$ since the transition $\langle R'+S, true\rangle \xrightarrow{x<5}$ is not redundant in $\mathcal{P}^2$ (since $\langle P+Q', z<5\rangle\ \not\mathcal{P}^2 \langle P, z<5\rangle$). Finally, the algorithm computes $\mathcal{P}^4$ that is equal to $\mathcal{P}^3$ and return it. It is interesting to observe that the transition $\langle R+S, true\rangle \xrightarrow{x<5}$ is redundant in all the partitions computed by the algorithm (and thus in $\dot{\sim}_{sb}$), while the transition $\langle R'+S, true\rangle \xrightarrow{x<5}$ is considered redundant in $\mathcal{P}^0$ and $\mathcal{P}^1$ and not redundant in $\mathcal{P}^2$ and $\mathcal{P}^3$.

4.1 Termination

Note that any iteration splits blocks and never fuse them. For this reason if $IS^\star$ is finite, the algorithm terminates in at most $|IS^\star|$ iterations. The proof of the next proposition assumes that $\vdash_D$ is decidable. However, as we shall prove in the next section, the decidability of $\vdash_D$ follows from our assumption about the decidability of the ordering relation $\sqsubseteq$ of the underlying constraint system and Theo. 3 in the next section.

Proposition 2. If $IS^\star$ is finite, then the algorithm terminates and the resulting partition coincides with $\dot{\sim}_{sb}$.

PROOF. See [2]. □

We now prove that if the set $\texttt{Config}(IS)$ of all configurations reachable from IS (through the LTS generated by the rules in Tab. 2) is finite, then $IS^\star$ is finite.

This condition can be easily guaranteed by imposing some syntactic restrictions on ccp terms, like for instance, by excluding either the procedure call or the hiding operator.

Theorem 2. If $\texttt{Config}(IS)$ is finite, then $IS^\star$ is finite.

PROOF. See [2]. □

4.2 Complexity of the Implementation

Here we give asymptotic bounds for the execution time of Alg. 2. We assume that the reader is familiar with the $O(.)$ notation for asymptotic upper bounds in analysis of algorithms–see [6].

Our implementation of Alg. 2 is a variant of the original partition refinement algorithm in [10] with two main differences: The computation of $IS^\star$ according to rules (IS), (RS) and (RD) (line 2, Alg. 2) and the decision procedure for $\vdash_D$ (Def. 5) needed in the redundancy checks.

Recall that we assume $\sqsubseteq$ to be decidable. Notice that requirement of having some e that satisfies both conditions (i) and (ii) in Def. 5 suggests that deciding whether two given transitions belong to $\vdash_D$ may be costly. The following theorem, however, provides a simpler characterization of $\vdash_D$ allowing us to reduce the decision problem of $\vdash_D$ to that of $\sqsubseteq$.

Theorem 3. $\langle P, c\rangle \xrightarrow{\alpha} \langle P_1, c'\rangle \vdash_D \langle P, c\rangle \xrightarrow{\beta} \langle P_1, c''\rangle$ iff the following conditions hold: (a) $\alpha \sqsubset \beta$ (b) $c'' = c' \sqcup \beta$

PROOF. See [2]. □

Henceforth we shall assume that given a constraint system $\mathbf{C}$, the function $f_{\mathbf{C}}$ represents the time complexity of deciding (whether two given constraints are in) $\sqsubseteq$. The following is a useful corollary of the above theorem.

Corollary 1. Given two transitions t and t', deciding whether $t \vdash_D t'$ takes $O(f_{\mathbf{C}})$ time.

Remark 2. We introduced $\vdash_D$ as in Def. 5 as natural adaptation of the corresponding notion in [5]. The simpler characterization given by the above theorem is due to particular properties of ccp transitions, in particular monotonicity of the store, and hence it may not hold in a more general scenario.

Complexity. The size of the set IS^* is central to the complexity of Alg. 2 and depends on topology of the underlying transition graph. For tree-like topologies, a typical representation of many transition graphs, one can show by using a simple combinatorial argument that the size of IS^* is quadratic w.r.t. the size of the set of reachable configurations from IS, i.e., $\texttt{Config}(IS)$. For arbitrary graphs, however, the size of IS^* may be exponential in the number of transitions between the states of $\texttt{Config}(IS)$ as shown by the following construction.

Definition 8. Let $P^0 = \mathbf{0}$ and $P^1 = P$. Given an even number n, define $s_n(n,0) = \mathbf{0}$, $s_n(n,1) = \mathbf{ask}\ (true)\ \rightarrow\ s_n(n,0)$ and for each $0 \leq i < n \wedge 0 \leq j \leq 1$ let $s_n(i,j) = (\mathbf{ask}\ (true)\ \rightarrow\ s_n(i, j\oplus 1)^{j\oplus 1} + (\mathbf{ask}\ (b_{i,j})\ \rightarrow\ \mathbf{0}) + (\mathbf{ask}\ (a_i)\ \rightarrow\ s_n(i+1,j))$ where $\oplus$ means addition modulo 2. We also assume that (1) for each $i, j : a_i \sqsubset b_{i,j}$ and (2) for each two different i and $i' : a_i \not\sqsubseteq a_{i'}$, and (3) for each two different (i,j) and (i',j'): $b_{i,j} \not\sqsubseteq b_{i',j'}$.

Let $IS = \{s_n(0,0)\}$. One can verify that the size of IS^* is indeed exponential in the number of transitions between the states of $\texttt{Config}(IS)$.

Since Alg. 2 computes IS^* the above construction shows that on some inputs Alg. 2 take *at least* exponential time. We conclude by stating an upper-bound on the execution time of Alg. 2.

Theorem 4. Let n be the size of the set of states $\texttt{Config}(IS)$ and let m be the number of transitions between those states. Then $n \times 2^{O(m)} \times f_{\mathbf{C}}$ is an upper bound for the running time of Alg. 2.

PROOF. See [2]. □

5. CONCLUDING REMARKS

In this paper we provided an algorithm for verifying (strong) bisimilarity for ccp by building upon the work in [5]. Weak bisimilarity is the variant obtained by ignoring, as much as possible, silent transitions (transitions labelled with $true$ in the ccp case). Neither [5] nor the present work deal with this weak variant. We therefore plan to provide an algorithm for this central equivalence in future work.

6. REFERENCES

[1] A. Aristizabal, F. Bonchi, C. Palamidessi, L. Pino, and F. D. Valencia. Deriving labels and bisimilarity for concurrent constraint programming. In *FOSSACS*, pages 138–152, 2011.

[2] A. Aristizabal, F. Bonchi, L. Pino, and F. Valencia. Partition refinement for bisimilarity in ccp (extended version). Technical report, INRIA-CNRS, 2012. Available at: http://www.lix.polytechnique.fr/~andresaristi/sac2012.pdf.

[3] F. Bonchi, F. Gadducci, and G. V. Monreale. Reactive systems, barbed semantics, and the mobile ambients. In *FOSSACS*, pages 272–287, 2009.

[4] F. Bonchi, B. König, and U. Montanari. Saturated semantics for reactive systems. In *LICS*, pages 69–80, 2006.

[5] F. Bonchi and U. Montanari. Minimization algorithm for symbolic bisimilarity. In *ESOP*, pages 267–284, 2009.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

[7] 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.

[8] J.-C. Fernandez. An implementation of an efficient algorithm for bisimulation equivalence. *Sci. Comput. Program.*, 13(1):219–236, 1989.

[9] G. Ferrari, S. Gnesi, U. Montanari, M. Pistore, and G. Ristori. Verifying mobile processes in the hal environment. In *CAV*, pages 511–515, 1998.

[10] P. C. Kanellakis and S. A. Smolka. Ccs expressions, finite state processes, and three problems of equivalence. In *PODC*, pages 228–240, 1983.

[11] R. Milner and D. Sangiorgi. Barbed bisimulation. In *ICALP*, pages 685–695, 1992.

[12] U. Montanari and V. Sassone. Dynamic congruence vs. progressing bisimulation for ccs. *FI*, 16(1):171–199, 1992.

[13] V. A. Saraswat and M. C. Rinard. Concurrent constraint programming. In *POPL*, pages 232–245, 1990.

[14] V. A. Saraswat, M. C. Rinard, and P. Panangaden. Semantic foundations of concurrent constraint programming. In *POPL*, pages 333–352, 1991.

[15] B. Victor and F. Moller. The mobility workbench - a tool for the pi-calculus. In *CAV*, pages 428–440, 1994.