# A Category of Explicit Fusions

Filippo Bonchi<sup>1</sup>, Maria Grazia Buscemi<sup>2</sup>, Vincenzo Ciancia<sup>1</sup>, and Fabio Gadducci<sup>1</sup>

 <sup>1</sup> Dipartimento di Informatica, University of Pisa, Italy {fibonchi,ciancia, gadducci}@di.unipi.it
 <sup>2</sup> IMT Lucca Institute for Advanced Studies, Italy

m.buscemi@imtlucca.it

Abstract. Name passing calculi are nowadays an established field on its own. Besides their practical relevance, they offered an intriguing challenge, since the standard operational, denotational and logical methods often proved inadequate to reason about these formalisms. A domain which has been successfully employed for languages with asymmetric communication, like the  $\pi$ -calculus, are presheaf categories based on (injective) relabelings, such as  $Set^{\mathbb{I}}$ . Calculi with symmetric binding, in the spirit of the fusion calculus, give rise to new research problems. In this work we examine the calculus of explicit fusions, and propose to model its syntax and semantics using the presheaf category  $Set^{\mathbb{E}}$ , where  $\mathbb{E}$  is the category of equivalence relations and equivalence preserving morphisms.

### 1 Introduction

Among the many research themes Ugo Montanari considered in his carrier, he was always concerned with the semantics of interactive systems. In our work we consider a few topics, related to such a large area, that always interested Ugo, namely final semantics, coalgebras, names and name fusions, and constraints.

**Denotational Semantics** *via* **Final Object.** In [35] Scott and Strachey introduced denotational semantics as a way of formalizing the meaning of programming languages: to each expression of the language a *denotation* is assigned, i.e., an object in a mathematical domain. In their original proposal, each program denotes a continuous function on a partially ordered set, mapping each input of the program into the corresponding output. Despite its expressiveness, the approach is less adequate in modelling the semantics of *interactive systems*: indeed, in this case the non-deterministic behaviour of a program is more important than the function it computes: thus, these systems can not be simply denoted as if they were input-output functions.

An important tenet of denotational semantics is that it should be *compositional*, i.e, the denotation of a program expression has to be constructed by the denotation of its sub-expressions. This property allows one to reason inductively on the program structure, providing a general methodology for proving properties of programs.

P. Degano et al. (Eds.): Montanari Festschrift, LNCS 5065, pp. 544–562, 2008.

A lot of effort has been spent to give compositional denotational semantics to concurrent programming languages. Usually, this has been accomplished by restricting the focus to simple computational models exhibiting fundamental aspects of concurrent computations. Amongst the various proposals, process calculi are one of the most successful: a set of basic operators defines the syntax of the calculus (more formally, the set of operators is a signature and the expressions of the language are the elements of the *initial algebra* associated to such a signature) and for each operator there is a set of (SOS [32]) rules describing the behaviour of the composite expression in terms of the behaviours of its sub-expressions. The resulting operational semantics is usually a *labeled transition system* (LTS) where labels represent interactions amongst the various sub-expressions (representing components) of an expression (a system).

Moreover, Universal Coalgebra [33] provides a good categorical framework for the denotation of process calculi. Given a behavioural endo-functor B, one can define the category of B-coalgebras and B-cohomomorphisms. By choosing a certain endo-functor on *Set* (i.e., the category of sets and functions), we get the category of all labeled transition systems and "zig-zag" morphisms (i.e., morphisms that both respect and reflect transitions). This category has a *final object* F, i.e., from every LTS there is a unique morphism to F.

In this setting, one can easily define the denotation of an LTS as its image through this unique morphism. This idea, that nowadays is called *final semantics*, was originally proposed for abstract data types by Giarratana, Gimona and Montanari in [20] and it is still central in Ugo's work (see e.g. [11,2]).

Compositional Denotational Semantics via Initial and Final Object. This kind of representation is not completely satisfactory, because the intrinsic algebraic structure of states is lost, and compositionality of denotational semantics is not reflected in this model. In [37], Turi and Plotkin provide a solution by means of *bialgebras*. These are pairs composed of a  $\Sigma$ -algebra and a *B*-coalgebra for  $\Sigma$  and *B* two endo-functors on *Set* related by a distributive law  $\lambda$ . Roughly, they have shown that providing a set of SOS rules (in some well-behaved format) corresponds to defining  $\lambda$ ; the syntax of the formalism is the initial algebra for  $\Sigma$  and the semantics domain is the final coalgebra for *B*. This uniquely induces a (bialgebraic) morphism (representing the denotational semantics) that maps each element of the initial  $\Sigma$ -algebra (i.e., each term of the syntax) into the final *B*-coalgebra (representing the denotation of the terms). Since morphisms also respect the operations of  $\Sigma$ , the denotational semantics is also compositional.

This approach is general enough to allow one, by simply modifying either the base category or the associated endo-functors, also to handle sophisticated process calculi having complex variable binding (like the value passing CCS [25] and the  $\pi$ -calculus [26]). More precisely, in order to represent both syntax and semantics, one has to consider proper endo-functors not on *Set*, but on categories *Set*<sup>C</sup> of *covariant presheaves* over some C. These are functors from a generic category C of interfaces and contexts to *Set*. Intuitively, a presheaf maps each object *i* of C to the set of states having *i* as interface, and each arrow  $c: i \to j$ to a function turning states with interface *i* into states with interface *j*. As an example, abstract syntax with variable binding has been tackled as a category of endo-functors  $\Sigma$  over  $Set^{\mathbb{F}}$  [14,21], for  $\mathbb{F}$  the category of finite cardinals (i.e., sets of variables) and all functions (i.e., variable substitutions).

Models for Names. The index category C can be made up of a single object. This is the case of presheaves in the category  $Set^{\mathbb{G}}$  with  $\mathbb{G}$  the group of permutations over natural numbers. This category is essentially the same as the *nominal sets* of Pitts and Gabbay [16], used to model abstract syntax with variable binding, and of *permutation algebras*, that have been exploited to give a final semantics of the  $\pi$ -calculus [28,6]. This corresponds to an *untyped* view of interfaces, where the actual element of interest is the action of a presheaf on the *arrows* of the one-object category.

Having more objects in the index category corresponds instead to a *typed* framework. One of the most widely investigated cases is  $\mathbb{I}$ , that is, the full subcategory of  $\mathbb{F}$  containing only injective morphisms. Both the early and late semantics of the  $\pi$ -calculus [26] can be characterized by proper endo-functors on  $Set^{\mathbb{I}}$  [36,13,15]. Intuitively, any object *i* of  $\mathbb{I}$  is mapped to a set of processes whose free names belong to *i*, and an arrow is mapped into an injective renaming. In this perspective, it is easy to understand the roots of a problem which is common to the typed and untyped case: interfaces of  $\pi$ -processes can always be enlarged (an operation that corresponds to allocate new names) but never contracted (so that two names can never be coalesced).

A solution to this problem was proposed by Montanari and Pistore by introducing *history dependent automata* [27,31], where at any step of the execution a set of names can be junked away, obtaining an operational model where finite state verification of name passing calculi can be carried out in interesting cases. A coalgebraic formulation of HD-automata was given in [10,11], by employing the category of *named sets*.

An equivalence result between the categories of named sets, permutation algebras,  $Set^{\mathbb{G}}$  and the full subcategory of  $Set^{\mathbb{I}}$  of pullback preserving functors (also known as *Schanuel topos*) has been given by the fourth author, Marino Miculan and Ugo in [17]. An extension of this result to the associated categories of coalgebras has recently been proposed in [9], by the third author and Ugo. There, a garbage-collecting functor for name abstraction is defined on named sets, allowing the coalgebraic framework of HD-automata to be generalised to calculi different than the  $\pi$ -calculus.

In the previous proposals, arrows of the index category are just injections of names, and thus names can not be fused. In order to tackle non injective substitutions of names, one can consider different index categories. The open semantics of the  $\pi$ -calculus [34], for example, can be defined using an endofunctor on  $Set^{\mathbb{D}}$ , for  $\mathbb{D}$  the category of irreflexive relations and relation-preserving morphisms [19,24]. Roughly, every object is a set of names equipped with an irreflexive relation such that two related names are considered distinguished. Morphisms are functions between sets of names that preserve the relation and thus they can not coalesce those names that are considered distinct. **Explicit Fusions.** In this work we introduce the category  $\mathbb{E}$ , of equivalence relations, in order to properly model *explicit fusions* [39], i.e., processes that equate two names, allowing all the processes running in parallel with them to use one name in place of the other. Our interest for the explicit fusion calculus comes from the fact that it lies half-way between the fusion calculus [30] (to which Ugo has devoted much attention in the last years [4,22,18,12,8]) and the CC-Pi [7]. Indeed, explicit fusions are just instances of the more general concept of *named c-semiring* introduced in [7] by the second author and Ugo.

Each object of  $\mathbb{E}$  is an equivalence relation over a set of names. Analogously to  $\mathbb{I}$ , morphisms preserve names (i.e., names can not be junked away), but equivalence classes can be enlarged, thus obtaining semantical fusion of names without loosing any syntactic name. We prove that  $\mathbb{E}$  is suitable for providing both syntax and semantics to the *calculus of explicit fusions* [39] by showing both a syntactic endo-functor  $\Sigma$  and a behavioural endo-functor B on  $Set^{\mathbb{E}}$ .

Unfortunately, it seems hard to recover the Turi-Plotkin framework, since the operational behaviour of explicit fusion calculus is not compositional with respect to the semantical name fusions provided by the arrows of  $\mathbb{E}$ . This might suggest that  $\mathbb{E}$  is not adequate for our purposes, but the main result states that *inside-outside bisimulations* [38] (i.e., the standard bisimulation of explicit fusion) are in one to one correspondence with coalgebraic bisimulations for the endo-functor B. Moreover, considering a proper saturated semantics [29,1] in  $Set^{\mathbb{E}}$  might bring to a compositional operational semantics, and thus to a bialgebraic representation.

Thus the paper presents no conclusive result for the denotational semantics of explicit fusion, but just a solid base for further studies. Moving beyond explicit fusion,  $\mathbb{E}$  is the first step toward a deeper understanding of named c-semirings and then for a Turi-Plotkin characterization of CC-Pi [7]. Moreover, by slightly modifying  $\mathbb{E}$ , thanks to its definition as a comma category, one can obtain a category of equivalence relations and distinctions that is suitable for open  $\pi$ -calculus [34] and D-fusion [4]. Finally, these formalisms share a symbolic semantics that can not be naively tackled through coalgebras. Giving a presheaf semantics for them will hopefully allow these symbolic semantics to be characterized through normalized coalgebras as shown by the first author and Ugo in [2,3].

Synopsis. In § 2 we give a brief overview of the explicit fusion calculus. In § 3 we introduce the category  $\mathbb{E}$  of equivalence relations. In § 4 we show how to use  $\mathbb{E}$  to characterise an abstract syntax of explicit fusion calculus as an initial algebra. In § 5 we define a behavioural endo-functor B on  $Set^{\mathbb{E}}$  and we state a correspondence between inside-outside bisimulations and B-bisimulations. In § 6 we draw some conclusions and provide directions for future work.

## 2 Background on the Explicit Fusion Calculus

The explicit fusion calculus is a variant of the  $\pi$ -calculus that aims at guaranteeing asynchronous broadcasting of name equivalences to the environment. In order to ease its introduction, this section presents the explicit fusion calculus in the standard  $\pi$ -calculus fashion rather than in the "commitment" style of [39].

Assume a set of names  $\mathcal{N}$ , ranged over by  $x, y, \ldots$  The *(explicit) fusion* processes, ranged over by  $P, Q, \ldots$  are defined by the syntax in Fig. 1(a). The  $\tau$  prefix stands for a silent action, while the complementary output  $\overline{x}\langle y \rangle$  and input  $x\langle y \rangle$  prefixes are used for communications. Unlike  $\pi$ -calculus, the input prefix is not a binder, hence input and output operations are fully symmetric. As usual, **0** stands for the inert process,  $P \mid P$  for the parallel composition, !P for the replication operator and (x)P for the process that makes the name x local in P. An *explicit fusion* x = y is a process that exists concurrently with the rest of the system and that enables to use the names x and yinterchangeably.

We now define the *structural* congruences  $\equiv_1$  and  $\equiv_2$  as the least congruences over processes closed with respect to  $\alpha$ -conversion and satisfying the axioms in Fig. 1(b) and Fig. 1(c), respectively. Finally, we define the structural congruence  $\equiv$  as the transitive closure  $(\equiv_1 \cup \equiv_2)^*$ . With respect to the standard structural congruence of fusion processes, here we changed a few axioms. In particular, we weakened the axiom called *reflexivity*, equating x = x and **0**: in our case, we offer a name-preserving version, equating x = x | P with P, whenever P contains x free. Moreover, instead of the *subtraction* axiom, that equates the processes  $(\nu x)(x = y)$  and **0** (thus allowing us to remove names from an equivalence relation via restriction), we prefer to add an explicit  $\alpha$ -conversion law for our processes, which is apparently the reason underlying the introduction of the subtraction axiom in [39]. Summing up, we considered a set of axioms guaranteeing that any parallel composition of fusions is closed with respect to the composition with any fusion, when that fusion occurs in the induced equivalence relation plus three axioms. These axioms (the left-most of Fig. 1(c)) ensure what is called small-step substitution in Fig. 2 of [39]: this is further made explicit by Proposition 1, taking into account the equivalence relation Eq(P) (Fig 1(d)), specifying the name equivalences induced by a process P: it properly generalizes [39, Lemma 5].

**Proposition 1 (decomposition).** Let P, Q be processes such that P = C[Q] for a unary context C[-] that does not bind the names in fn(P) (i.e., such that the placeholder – does not fall in the scope of a restriction operator (x), for any  $x \in fn(P)$ ). Moreover, let  $E_P$  be the process

$$\prod_{\{x,y\in \mathrm{fn}(P)\mid (x,y)\in \mathrm{Eq}(P)\}}x=y$$

(where  $\prod$  denotes the multiple parallel composition). Then,  $P \equiv_2 C[E_P | Q]$ .

The rules of the operational semantics, as depicted in Fig. 1(e), recall the rules of the  $\pi$ -calculus. We assume a set of labels

$$\mathcal{L} = \{\tau, z \langle y \rangle, \overline{z} \langle y \rangle, z(w), \overline{z}(w)\}$$

where z, y are free and w is bound. We let  $\lambda, \lambda_1, \ldots$  range over  $\mathcal{L}$ . By  $Eq(P) \vdash \lambda_1 = \lambda_2$  we mean that P contains enough fusions to interchange  $\lambda_1$  and  $\lambda_2$ . Formally, we have

$$\begin{aligned} & \mathsf{Eq}(P) \vdash \tau = \tau \\ & \mathsf{Eq}(P) \vdash z\langle y \rangle = x\langle w \rangle \quad \text{if } (z, x), (y, w) \in \mathsf{Eq}(P) \\ & \text{(similarly for output)} \\ & \mathsf{Eq}(P) \vdash z(v) = x(v) \quad \text{if } (z, x) \in \mathsf{Eq}(P) \end{aligned}$$

Unlike the  $\pi$ -calculus, the synchronization of two complementary processes  $x\langle y\rangle P$  and  $\overline{x}\langle z\rangle Q$  yields an explicit fusion y = z rather than binding y to z. According to rule (FUS), a process can undergo any transition up to interchanging names that are fused. For instance, a process  $x = y | \overline{x}\langle z \rangle P$  can make both actions  $\overline{x}\langle z \rangle$  and  $\overline{y}\langle z \rangle$ .

In [38] several bisimulations have been proposed for the explicit fusion calculus: they were all proved to coincide, and in fact to be congruences. For convenience, as a reference semantics we consider the *inside-outside* bisimulation that is much like  $\pi$ -calculus open bisimulation.

**Definition 1 (inside-outside bisimulation).** Let R be a symmetric relation on fusion processes. We say that R is an inside-outside bisimulation if whenever P R Q holds then

1.  $\operatorname{Eq}(P) = \operatorname{Eq}(Q);$ 2. If  $P \xrightarrow{\lambda} P'$  with  $\operatorname{bn}(\lambda) \cap \operatorname{fn}(Q) = \emptyset$  then  $Q \xrightarrow{\lambda} Q'$  and P' R Q';3.  $P \mid x = y R Q \mid x = y$ , for all fusions x = y.

We let  $\sim^{io}$  denote the largest such bisimulation, the inside-outside bisimilarity.

According to clause 1, two processes x = y and **0** are not bisimilar, since Eq(x = y) and Eq(0) differ. However, bisimilarity is not name preserving, so that **0** is indeed bisimilar to x = x. Finally, note that clause 3 allows bisimilarity to distinguish the following processes (inspired by an example described by Boreale and Sangiorgi [5] for the  $\pi$ -calculus)

$$P = !\overline{y}\langle \rangle . x \langle \rangle . \tau . z \langle \rangle \mid ! x \langle \rangle . \overline{y} \langle \rangle . \tau . z \langle \rangle \qquad Q = !(w)(\overline{y}\langle \rangle . \overline{w} \langle \rangle \mid x \langle \rangle . w \langle \rangle . z \langle \rangle)$$

When inserted into the context |x = y, Q can perform an action  $\xrightarrow{z\langle\rangle}$  after two steps (synchronizing  $\overline{y}\langle\rangle$  with  $x\langle\rangle$ ), while P at least after three steps.

## 3 A Category of Name Equivalences

The paper aims at extending the presheaf approach in order to tackle the calculus of explicit fusions. To this end, we now define a category of equivalence relations  $\mathbb{E}$  that will be used to represent (the sets of) fusion processes as a presheaf

 $\pi ::= \tau \mid \overline{x} \langle y \rangle \mid x \langle y \rangle \qquad P ::= \mathbf{0} \mid x = y \mid \pi.P \mid P \mid P \mid (x)P \mid !P$ (a) syntax

$$\begin{array}{ll} P \mid \mathbf{0} \equiv_{1} P & P \mid Q \equiv_{1} Q \mid P & (P \mid Q) \mid R \equiv_{1} P \mid (Q \mid R) \\ !P \equiv_{1} P \mid !P & (x)(y)P \equiv_{1} (y)(x)P & P \mid (x)Q \equiv_{1} (x)(P \mid Q) & \text{if } x \notin \operatorname{fn}(P) \end{array}$$

(b) structural congruence, I

$$\begin{aligned} x &= y \mid \pi.P \equiv_2 x = y \mid \pi.(x = y \mid P) & x = x \mid P \equiv_2 P \quad if \ x \in \text{fn}(P) \\ x &= y \mid (z)P \equiv_2 x = y \mid (z)(x = y \mid P) & if \ z \notin \{x, y\} & x = y \equiv_2 y = x \\ x &= y \mid !P \equiv_2 ! (x = y \mid P) & x = y \mid y = z \equiv_2 x = z \mid y = z \end{aligned}$$

(c) structural congruence, II

| $\operatorname{Eq}(0) = \operatorname{Eq}(\pi.P) = \operatorname{Id}$                      | the identity relation                   |
|--|---|
| $\operatorname{Eq}(x=y)=\{(x,y),(y,x)\}\cup Id$  | smallest equivalence including $(x, y)$ |
| $\operatorname{Eq}(P   Q) = (\operatorname{Eq}(P) \cup \operatorname{Eq}(Q))^*$            | transitively-closed union               |
| $\operatorname{Eq}((x) P) = \operatorname{Eq}(P) \setminus \{(y, z) \mid x \in \{y, z\}\}$ | removing name from equivalence classes  |
| $\operatorname{Eq}(\operatorname{!} P) = \operatorname{Eq}(P)$                             | removing replication operator           |
|  |   |

(d) equivalence relation Eq(P)

$$\begin{array}{c} \stackrel{(\text{PREF})}{\pi,P \to} P & \stackrel{(\text{COMM})}{P \mid Q \xrightarrow{\tau} P' \quad Q \xrightarrow{\overline{x} \langle w \rangle} Q'} & \stackrel{(\text{PAR})}{P \mid Q \to P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{RES})}{P \mid Q \xrightarrow{\tau} P' \mid Q' \mid y = w} & \stackrel{P \to P' \quad bn(\lambda) \cap fn(Q) = \emptyset}{P \mid Q \xrightarrow{\lambda} P' \quad |Q|} \\ \stackrel{(\text{OPEN-I})}{P \mid Q \to P' \mid Q} & \stackrel{(\text{OPEN-I})}{P \mid Q \to P' \mid Q} \\ \stackrel{(\text{OPEN-I})}{(\text{STRUCT})} & \stackrel{P \to Q' = Q}{P \xrightarrow{\lambda} Q' = Q} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \xrightarrow{\lambda} P' \quad bn(\lambda) \cap fn(Q) \cap fn(Q) = \emptyset} \\ \stackrel{(\text{PAR})}{P \mid Q \mid p$$

(e) operational semantics (omitting rule (OPEN-O) for output)

Fig. 1. Explicit fusion calculus

 $\mathbb{E} \to Set$ . The objects of  $\mathbb{E}$  are basically surjective functions on sets and the arrows of  $\mathbb{E}$  are defined by taking suitable injective functions.

**Definition 2 (The category**  $\mathbb{E}$ ). The category  $\mathbb{E}$  of equivalence relations is the category whose objects are surjective functions  $h: s \to t$  in Set and whose arrows  $\sigma: h \to h'$  are pairs  $\langle \sigma_1, \sigma_2 \rangle$  of functions in Set such that  $\sigma_1: s \to s'$  is injective and the diagram below commutes.



The domain s of the function h specifies a set of names while the codomain represents a set of equivalence classes involving the names of s. Note that every s' has at least the same arity of s. Furthermore, the equivalence classes in t must be preserved in t': in other words, the equivalence classes of the names in s, as represented in t and carried along  $\sigma_1$  to s', may be either left unchanged or further merged in t', but they can not be broken.

Note that all arrows in  $\mathbb{E}$  are monomorphims. We let  $\mathbb{E}(h, h')$  denote the set of all arrows in  $\mathbb{E}$  with source h and target h'. Each object h of  $\mathbb{E}$  defines a functor  $\mathbb{E}(h, \underline{\ }) : \mathbb{E} \to Set$  as follows. Every object h' of  $\mathbb{E}$  is mapped into the set  $\mathbb{E}(h, h')$ . An arrow  $\sigma : h' \to h''$  of  $\mathbb{E}$  is mapped in the function  $\mathbb{E}(h, \sigma) : \mathbb{E}(h, h') \to \mathbb{E}(h, h'')$ , defined by post-composition: for any  $\rho \in \mathbb{E}(h, h')$ ,  $\mathbb{E}(h, \sigma)(\rho) = \rho; \sigma$ .

Let  $\{\star\}$  denote the one element set, graphically represented as  $\bullet$ , and let us denote by 1 and E the following two objects of  $\mathbb{E}$  (corresponding to  $id_{\{\star\}}$  and  $[id_{\{\star\}}, id_{\{\star\}}]$ , respectively, for the uniquely-induced arrow)



Then,  $\mathbb{E}(1, h)$  is the set of all monomorphisms that map  $\star$  to an element of the domain of h: hence, it is isomorphic to it. Similarly,  $\mathbb{E}(E, h)$  abstractly represents the set of explicit fusions x = y (for x, y different names) that hold in h. Hereafter, we denote the set  $\mathbb{E}(1, \_)$  by *Names* and the set  $\mathbb{E}(E, \_)$  by *Fus*.

On the structure of  $\mathbb{E}$  Consider the comma category  $ID_{Set} \downarrow ID_{Set}$ , i.e. the category whose objects are triples  $\langle s, t, h : s \to t \rangle$  and whose arrows are pairs  $\sigma_1 : s \to s', \sigma_2 : t \to t'$  such that the diagram below commutes.



Let  $IN_{\mathbb{I}} : \mathbb{I} \to Set$  be the functor that injects the category  $\mathbb{I}$  into Set. Consider the category  $IN_{\mathbb{I}} \downarrow ID_{Set}$ , where  $\mathbb{I}$  is the category of sets and injective functions: in

the definition above, it simply means that  $\sigma_1$  must always be a monomorphism. Now,  $\mathbb{E}$  is the full subcategory of  $IN_{\mathbb{I}} \downarrow ID_{Set}$ , including only those objects  $\langle s, t, h : s \rightarrow t \rangle$  such that h is surjective.

All limits do exist in  $IN_{\mathbb{I}} \downarrow ID_{Set}$ : they are simply computed point-wise. This is not the case instead in  $\mathbb{E}$ . Let us consider e.g. pullbacks: given any two arrows  $\langle \sigma_1, \sigma_2 \rangle : h \to h^*$ , and  $\langle \sigma'_1, \sigma'_2 \rangle : h' \to h^*$ , their pullback exists only if either  $\sigma_2$ or  $\sigma'_2$  is mono. Note that if both components of an arrow  $\sigma$  are mono, then  $\sigma$  is a *regular* monomorphism in  $IN_{\mathbb{I}} \downarrow ID_{Set}$  (and in  $\mathbb{E}$  as well), meaning that it can be computed as the equalizer of two arrows.

In the following, we sometimes consider functors from  $\mathbb{E}$  to *Set* that are pullback-preserving: this fact which basically implies that all regular monomorphisms in  $\mathbb{E}$  are mapped into injective functions in *Set*.

## 4 Abstract Syntax

In this section we consider the category  $Set^{\mathbb{E}}$  of functors from  $\mathbb{E}$  to Set (called *presheaves* over  $\mathbb{E}^{op}$ ) and natural transformations. This category can be used for both the syntax and the semantics of the explicit fusion calculus.

In the syntax, objects of the so-called *index* category  $\mathbb{E}$  can be viewed as *types* representing the equivalence classes of process names. The presheaf for the syntax gives, for each index  $h : s \twoheadrightarrow t$ , the set of those processes that can be typed with h, i.e., processes whose set of free names coincides with s, and whose equivalence class is the one induced by h.

In Section 4.1 we provide an account of the concrete syntax of the explicit fusion calculus as a presheaf, exploiting the previously defined equivalence relation Eq. Then, in Section 4.3 we explain how this syntax can be described as an initial algebra in  $Set^{\mathbb{E}}$  for a suitable endo-functor, preserving types. For the purpose, in Section 4.2 we introduce a number of basic constructions that are used as a meta-language in  $Set^{\mathbb{E}}$ , and are distinguishing features of this category.

## 4.1 Typing the Concrete Syntax

For the definition of the syntactic presheaf, for each arrow  $h : s \rightarrow t$  we introduce the notation  $Ker_h$ , representing the equivalence relation induced by h (obviously defined), and the process  $P_h$ , containing all the pairs in an equivalence relation h, viewed as fusions (analogous to the process  $E_P$  induced by Eq(P))

$$P_h = \prod_{\{x,y \in s | h(x) = h(y)\}} x = y.$$

Consider the functor  $Syn: \mathbb{E} \to Set$ , defined on objects as

$$Syn (h: s \twoheadrightarrow t) = \{P \mid \mathrm{fn}(P) = s \land \mathrm{Eq}(P) = Ker_h \cup \mathsf{Id}\} / \equiv_2 .$$

The intuition behind this definition is that if an agent P is in  $Syn(h: s \rightarrow t)$ , then its free names are the elements of s, and its equivalence classes Eq(P) are exactly those described by h, modulo adding the pairs (x, x) for  $x \notin fn(P)$ .

The action of the functor on arrows must injectively relabel processes, whilst it syntactically adds as many fusions as needed (actually, all of them) to put a process in the equivalence class chosen as destination

$$Syn (\langle \sigma_1 : s \hookrightarrow s', \sigma_2 : t \to t' \rangle : h \to h')(P) = P\sigma_1 \mid P_{h'}$$

# 4.2 A Basic Metalanguage in $Set^{\mathbb{E}}$

Besides the usual constructors for the polynomial endo-functors (namely constants, finite sums and products),  $Set^{\mathbb{E}}$  actually allows us to define additional constructors that are well-suited for handling fusions.

**Bottom operator**  $F^{\perp}$ . For any  $s \in Set$ , let  $s_{\perp}$  be  $s + \{\star\}$  and for any  $h : s \twoheadrightarrow t$ , let  $h_{\perp} : s_{\perp} \twoheadrightarrow t_{\perp}$  be the arrow  $[h, id_{\{\star\}}]$ . The bottom operator  $F^{\perp}$  is an endo-functor on  $Set^{E}$  defined as  $F^{\perp}(h) = F(h_{\perp})$  and  $F^{\perp}(\langle \sigma_{1}, \sigma_{2} \rangle) = F(\langle (\sigma_{1})_{\perp}, (\sigma_{2})_{\perp} \rangle)$ .

**Box operator**  $F^{\Box}$ . The box operator  $F^{\Box}$  is the endo-functor on  $Set^{\mathbb{E}}$  defined on objects and arrows as below. Let  $h: s \to t$  be an object of  $\mathbb{E}$ . Then

$$F^{\square}(h) = \sum_{p} F(h;p)$$

where  $p: t \rightarrow p(t)$  is a (identity preserving) morphism from t to a partition p(t) of t. Intuitively, p is an epimorphism further merging the names in s. Choosing p(t) as a target of p simply amounts to choose a canonical representative for each isomorphic merging of names.

$$\begin{array}{c|c} s & \stackrel{id_s}{\longrightarrow} s \\ h & \downarrow h; p \\ t & \stackrel{p}{\longrightarrow} p(t) \end{array}$$

Let  $h: s \twoheadrightarrow t$  and  $h': s' \twoheadrightarrow t'$  be objects of  $\mathbb{E}$  and  $\sigma = \langle \sigma_1, \sigma_2 \rangle : h \to h'$  be an arrow of  $\mathbb{E}$ . Then

$$F^{\Box}(\sigma) = \sum_{p} F(\langle \sigma_1, \sigma_2^* \rangle)$$

where  $\sigma_2^*$  is uniquely induced by the pushout depicted in the diagram below for each p, noting that  $\langle \sigma_1, \sigma_2^* \rangle : h; p \to h'; p^*$ .



Roughly,  $F^{\Box}$  may add any possible name equivalence to an equivalence relation on s. The box operator is used in Section 4.3 to define the functor for prefixes of the explicit fusion. Indeed, a fusion process with a prefix  $\pi$ . P has no equivalence relation on its names, while P may have any equivalence.

Shift operator  $F^{\delta}$ . The *shift operator* is the functor defined on objects and arrows as follows. Let  $h: s \to t$  be an object of  $\mathbb{E}$ . Then

$$F^{\delta}(h) = F^{\perp}(h) + \sum_{e} F([h, e])$$

where  $e : \{\star\} \to t$  is a function mapping  $\star$  in some element of t and  $[h, e] : s + \{\star\} \to t$  is the function uniquely induced by the coproduct, mapping all elements of s in h(s) and  $\star$  in  $e(\star)$ .

Let  $h: s \twoheadrightarrow t$  and  $h': s' \twoheadrightarrow t'$  be objects of  $\mathbb{E}$  and  $\sigma = \langle \sigma_1, \sigma_2 \rangle : h \to h'$  be an arrow of  $\mathbb{E}$ . Then

$$F^{\delta}(\sigma) = F^{\perp}(\langle \sigma_1, \sigma_2 \rangle) + \sum_e F(\langle (\sigma_1)_{\perp}, \sigma_2 \rangle)$$

noting that  $e; \sigma_2 : \{\star\} \to t'$  and  $\langle (\sigma_1)_{\perp}, \sigma_2 \rangle : [h, e] \to [h', e; \sigma_2].$ 

The shift operator is used to define the functor for the restriction operation in the explicit fusion. In fact,  $F^{\delta}$  is a variant of the functor for name generation used to model restriction in the  $\pi$ -calculus. The main point here is that objects are now equivalence relations. Hence, when generating a new name x, it is necessary to specify whether x is equivalent to any other name already occurring in the process, or it belongs to its own equivalence class.

#### 4.3 Explicit Fusion Syntax as an Initial Algebra

An abstract syntax for the explicit fusion calculus is captured by the initial algebra of the endo-functor  $\Sigma: Set^{\mathbb{E}} \to Set^{\mathbb{E}}$  that is defined below.

| $\Sigma F =$ | 1                                    | (inert process)        |
|--------------|--------------------------------------|------------------------|
| +            | F                                    | (replication)          |
| +            | $F \times F$                         | (parallel composition) |
| +            | $F^{\Box}$                           | (tau prefix)           |
| +            | $Names \times Names \times F^{\Box}$ | (input prefix)         |
| +            | $Names \times Names \times F^{\Box}$ | (output prefix)        |
| +            | $F^{\delta}$                         | (restriction)          |

We briefly comment on the component functors. For the purpose, we introduce the notation  $!_s$  representing the unique morphism of type  $s \twoheadrightarrow \{\star\}$  from s to the final object in *Set*, viewed as an object of  $\mathbb{E}$  (noting that  $!_{\{\star\}} = id_{\{\star\}}$ ). Moreover, we say that an object  $h : s \twoheadrightarrow t$  of  $\mathbb{E}$  is *included* in an object  $h_1 : s_1 \twoheadrightarrow t_1$ , and we write  $h \sqsubseteq h_1$ , if there exists an arrow  $\sigma : h \to h_1$ .

The functor for the inert process just returns a singleton at each stage; whilst the functor for replication just returns an additional copy of the processes at each stage, as if prefixing them with a suitable operator. 
 Table 1. Inference rules for Abstract Syntax

$$\mathbf{0} \in F(id_{\emptyset}) \qquad \frac{p \in F(h) \quad h \sqsubseteq h_{1}}{p \in F(h_{1})} \qquad \frac{p \in F(h)}{!p \in F(h)}$$

$$\frac{p \in F^{\Box}(h)}{\tau \cdot p \in F(h)} \qquad \frac{p \in F^{\delta}(h)}{(\star)p \in F(h)} \qquad \frac{p \in F(h) \quad q \in F(h)}{p \mid q \in F(h)}$$

$$\frac{a \in Names(h) \quad b \in Names(h) \quad p \in F^{\Box}(h)}{\overline{a}\langle b \rangle \cdot p \in F(h)}$$

$$\frac{a \in Names(h) \quad b \in Names(h) \quad p \in F^{\Box}(h)}{a\langle b \rangle \cdot p \in F(h)}$$

The three functors for prefixing are similar, the idea being of "hiding" equivalences of a process P when prefixed as  $\pi . P$ , similarly to what is done in the definition of Eq(P). To this purpose, we use the functor  $F^{\Box}$ : it returns, at stage  $h: s \to t$ , all the processes occurring in F(h') for all h' such that  $h \sqsubseteq h' \sqsubseteq !_s$ .

The parallel composition of two elements  $a \in F(h_1)$  and  $b \in F(h_2)$  can only be found at a stage h such that  $h_1 \sqsubseteq h$  and  $h_2 \sqsubseteq h$ . Indeed, being a presheaf, the product  $F \times F$  is a functor of type  $\mathbb{E} \to Set$  defined as  $(F \times F)(h) = F(h) \times F(h)$ . Hence, we can not find the product of two elements  $h_1$  and  $h_2$  belonging to different stages, but we have to "inject" them into a common stage h, where both can be found. Note that, since arrows can never remove fusions from objects of  $\mathbb{E}$ , this requirement is equivalent to stating that  $Eq(P | Q) = (Eq(P) \cup Eq(Q))^*$ , carrying on the intuition that the stage of an element always includes its fusions.

Next, we consider  $F^{\delta}$ . We can exemplify its meaning using the process (x)(x = y). This process belongs to  $F^{\delta}(id_{\{y\}})$ , since its set of free names is indeed included in  $\{y\}$ . However, it is obtained from the process x = y, which does not belong to  $F(id_{\{x,y\}})$ , but rather to  $F(!_{\{x,y\}})$ . Hence, to capture the process (x)(x = y) in the abstract syntax,  $F(!_{\{y,x\}})$  has to be a subset of  $F^{\delta}(id_{\{y\}})$ .

The previous discussion is summed up by Table 1. The different stages h account for the equivalences. Note e.g. that **0** belongs to  $F(id_{\emptyset})$ , for  $id_{\emptyset}$  the identity of the empty set. It thus belongs also to  $F(id_{\{x\}})$  and to  $F(!_{\{x,y\}})$ , for any name x, y. However, in the latter stage it represents the process **0** | x = y.

#### 4.4 Including Fusions, Syntactically

We could have introduced explicitly the fusions in our syntactical functor, simply considering the functor Fus to represent them

$$\Sigma'F = \Sigma F$$
 (fusionless calculus)  
+ Fus (explicit fusions)

This ensures, by definition, that a process x = y (for x, y different names) can not be found in F(h) unless  $(x, y) \in Ker_h$ , i.e. that the syntax reflects the equivalences of the objects in the index category. These constraints propagate to the parallel composition of fusions with arbitrary processes, and they induce exactly the equivalence relation Eq on the abstract syntax. Nevertheless, this choice requires a more complex relationship between concrete and abstract functor, and we thus preferred to deal with the fusions only at the stage level.

# 5 Behavioural Functor

In this section we introduce a behavioural endo-functor B on  $Set^{\mathbb{E}}$ . This automatically derives a notion of bisimulation that remarkably coincides with *insideoutside* bisimulation. In order to prove this, we will first provide a more concrete characterization of B-coalgebras in terms of  $\mathbb{E}$ -transition systems and then a corresponding notion of bisimulation.

The behavioural endo-functor B describes the type of the transition system.

| BF | =    | $\mathcal{P}_f($                     |                       |
|----|------|--------------------------------------|-----------------------|
|    |      | $F^{\Box}$                           | (tau action)          |
|    | +    | $Names \times Names \times F^{\Box}$ | (input action)        |
|    | +    | $Names \times Names \times F^{\Box}$ | (output action)       |
|    | $^+$ | $Names \times F^{\perp^{\square}}$   | (bound input action)  |
|    | +    | $Names \times F^{\perp^{\square}}$   | (bound output action) |
|    |      | )                                    |                       |

First of all, note the definition of input. This is quite different from the standard works on presheaf semantics for  $\pi$ -calculus, such as [15], where an exponential type was used for the input action. In the explicit fusion calculus the input and output prefixes are completely symmetric and thus input is non-binding. For this reason, we can safely tackle it in the same way as output.

Moreover, the destination states for  $\tau$  actions are in  $F^{\Box}$ . This can be understood by noting that when a process performs a  $\tau$  transition, then the destination has the same names of the starting state, yet possibly with more fusions. As an example, consider the fusion process  $\tau . (x = y \mid Q)$ . Performing a  $\tau$  transition, this process arrives into the state  $x = y \mid Q$ , where x and y are now identified. Analogously for input and output actions. Instead, in the case of bound input and bound output, the arriving state has one more name and (possibly) more fusions (that is, it has to belong to  $F^{\perp \Box}$ ).

#### 5.1 *B*-coalgebras as $\mathbb{E}$ -transition Systems

A *B*-coalgebra is a pair  $\langle F, \beta \rangle$ , for *F* an object of  $Set^{\mathbb{E}}$  (i.e.,  $F : \mathbb{E} \to Set$  is a functor) and  $\beta : F \to B(F)$  is an arrow of  $Set^{\mathbb{E}}$ , i.e., a natural transformation between *F* and B(F). In other words,  $\beta : F \to B(F)$  is a family of functions  $(\beta)_h : F(h) \to B(F)(h)$  for all  $h \in \mathbb{E}$ , satisfying suitable naturality requirements. Now, let  $\int F$  denote the set of the *elements* of a functor *F*, namely, the disjoint union  $\sum_{h \in \mathbb{E}} F(h)$ . It is easy to note that for any functor *F* and  $h \in \mathbb{E}$  the following inclusions  $F^{\perp}(h) \subseteq \int F$ ,  $F^{\Box}(h) \subseteq \int F$  and  $F^{\delta}(h) \subseteq \int F$  hold, so

that by construction every *B*-coalgebra induces a transition relation whose state space is represented by  $\int F$ .

In the following, we denote the elements of a functor F as pairs  $h \vdash f$ , for  $f \in F(h)$  and we write  $h \vdash f \in F(h')$  whenever  $F(h) \subseteq F(h')$ .

**Definition 3 (E-transition system).** An  $\mathbb{E}$ -transition system consists of a presheaf  $F : \mathbb{E} \to Set$  and a transition relation  $\to$  such that

- 1. states are elements of  $\int F$ ;
- 2. transitions are typed such that
  - $if h \vdash f \xrightarrow{\tau} h' \vdash f' \text{ then } h' \vdash f' \in F^{\Box}(h),$  $- if h \vdash f \xrightarrow{z\langle y \rangle} h' \vdash f' \text{ then } h' \vdash f' \in F^{\Box}(h) \text{ and } z, y \in Names(h),$  $- if h \vdash f \xrightarrow{z(x)} h' \vdash f' \text{ then } h' \vdash f' \in F^{\bot^{\Box}}(h) \text{ and } z \in Names(h).$

- if  $h \vdash f \longrightarrow h' \vdash f'$  then  $h' \vdash f' \in F^{\perp}$  (h) and  $z \in Names(h)$ ,

- where the symmetric rules for output are omitted;
- 3. transitions are preserved by morphisms  $\sigma = \langle \sigma_1, \sigma_2 \rangle : h \to i$ , that is
  - $if h \vdash f \xrightarrow{\tau} h' \vdash f' then i \vdash F(\sigma)(f) \xrightarrow{\tau} i; p^* \vdash F(\langle \sigma_1, \sigma_2^* \rangle)(f') for p such that h' = h; p,$
  - $if h \vdash f \xrightarrow{z\langle y \rangle} h' \vdash f' \text{ then } i \vdash F(\sigma)(f) \xrightarrow{\sigma_1(z\langle y \rangle)} i; p^* \vdash F(\langle \sigma_1, \sigma_2^* \rangle)(f') \text{ for } p \text{ such that } h' = h; p,$
  - $\begin{array}{rrrr} if \ h \ \vdash \ f \ \xrightarrow{z(\star)} \ h' \ \vdash \ f' \ then \ i \ \vdash \ F(\sigma)(f) \ \xrightarrow{(\sigma_1)_{\perp}(z(\star))} \ i; p^* \ \vdash \\ F(\langle (\sigma_1)_{\perp}, (\sigma_2)_{\perp}^* \rangle)(f') \ for \ p \ such \ that \ h' = h_{\perp}; p \end{array}$

where p,  $p^*$ , and  $\sigma_2^*$  are as in the definition of the box operator (see Section 4.2) and where the symmetric rules for output are omitted;

- 4. transitions are reflected by morphisms  $\sigma = \langle \sigma_1, \sigma_2 \rangle : h \to i$ , that is
  - $if i \vdash F(\sigma)(f) \xrightarrow{\tau} i' \vdash f' \text{ then } h \vdash f \xrightarrow{\tau} h'' \vdash f'' \text{ such that } h'' = h; p \text{ and } i' = i; p^* \text{ and } F(\langle \sigma_1, \sigma_2^* \rangle)(f'') = f',$
  - $\begin{array}{l} \ if \ i \vdash F(\sigma)(f) \xrightarrow{z\langle y \rangle} i' \vdash f' \ then \ h \vdash f \xrightarrow{u\langle v \rangle} h'' \vdash f'' \ such \ that \ h'' = h; p, \\ i' = i; p^*, \ F(\langle \sigma_1, \sigma_2^* \rangle)(f'') = f' \ and \ \sigma_1(u\langle v \rangle) = z\langle y \rangle, \end{array}$
  - $\begin{array}{l} if \ i \vdash F(\sigma)(f) \xrightarrow{z(\star)} i' \vdash f' \ then \ h \vdash f \xrightarrow{u(v)} h'' \vdash f'' \ such \ that \ h'' = h_{\perp}; p, \\ i' = i; p^*, \ F(\langle (\sigma_1)_{\perp}, (\sigma_2)_{\perp}^* \rangle)(f'') = f' \ and \ (\sigma_1)_{\perp}(u(\star)) = z(\star). \end{array}$

where p,  $p^*$ , and  $\sigma_2^*$  are as in the definition of the box operator (see Section 4.2) and where the symmetric rules for output are omitted.

The first condition requires that the states are indexed by the object of  $\mathbb{E}$ . The second condition requires that the transitions have the right type (according to the definition of B). As for the third and fourth condition, they boil down to require that for arrow  $\sigma : h \to i$  (that is, for each renaming of names  $\sigma_1$  and each enlargement of equivalence classes  $\sigma_2$ ), the set of transitions leaving from each state  $h \vdash f$  has to be precisely the same as for the set of transitions leaving from  $i \vdash f'$ , for f' the state  $F(\sigma)(f)$  in the stage i.

We now give the first result of our contribution.

**Proposition 2.**  $\mathbb{E}$ -transition systems are in one to one correspondence with *B*-coalgebras.

*Proof.* Consider the requirements of  $\mathbb{E}$ -transition system. The first condition just states that we are working with a presheaf. The second condition imposes the correct type. The third and fourth requirement just impose that the transition structure is a natural transformation between F and B(F), for each functor F.

We now move to introduce a suitable notion of E-bisimulation.

**Definition 4.** Let  $(F, \rightarrow)$  and  $(G, \rightarrow)$  be two  $\mathbb{E}$ -transition systems. Let  $R \subseteq \bigoplus_{h \in \mathbb{E}} F(h) \times G(h)$  be an  $\mathbb{E}$ -sorted family of symmetric relations. We say that R is an  $\mathbb{E}$ -bisimulation if whenever  $fR_hg$  then

 $- if \sigma : h \to i, \text{ then } F(\sigma)(f)R_iG(\sigma)(g), \\ - if h \vdash f \xrightarrow{\alpha} h' \vdash f' \text{ then } h \vdash g \xrightarrow{\alpha} h' \vdash g' \text{ and } f'R_{h'}g'.$ 

We finally state the main result of the paper.

**Proposition 3.**  $\mathbb{E}$ -bisimulations are in one to one correspondence with coalgebraic bisimulations for the endo-functor  $B : Set^{\mathbb{E}} \to Set^{\mathbb{E}}$ .

*Proof.* Let us now consider the definition of the coalgebraic bisimulation for the endo-functor  $B: Set^{\mathbb{E}} \to Set^{\mathbb{E}}$ .

A presheaf R and two natural transformations  $a : R \to F$  and  $b : R \to G$  form a bisimulation (R, a, b) between  $(F, \alpha)$  and  $(G, \beta)$  if (R, a, b) is a monic span<sup>1</sup> in  $Set^{\mathbb{E}}$  and there exists in  $Set^{\mathbb{E}}$  a natural transformation  $\gamma : R \to B(R)$  such that the following commutes.



First of all note that in any category C with binary product, (R, a, b) is a monic span if and only if the induced morphism  $\langle a, b \rangle : R \to F \times G$  is a monomorphism. In Definition 4 we require that  $R \subseteq \bigoplus_{h \in \mathbb{E}} F(h) \times G(h)$ , and this is equivalent to restricting to those  $\langle a, b \rangle$  that are injections of R into  $F \times G$ , instead of simply monomorphisms. Therefore, the one to one correspondence holds only up to isomorphism, as it is standard in the theory of coalgebras.

Then, the first requirement of Definition 4 coincides with the fact that R is a presheaf and a, b are natural transformations. Indeed, if  $fR_hg$  (i.e.,  $(f,g) \in R(h)$ ) and  $\sigma : h \to i$ , then  $R(\sigma)(f,g) \in R(i)$ , because R is a functor  $\mathbb{E} \to Set$ . Notice that  $R(\sigma)(f,g) = (F(\sigma)(f), G(\sigma)(g))$ , when considering  $\langle a, b \rangle$  as the injection of R into  $F \times G$ . Indeed, since  $a : R \to F$  is a natural transformation, then  $a_i(R(\sigma)(f,g)) = F(\sigma)a_h(f,g) = F(\sigma)(f)$  (because  $\langle a, b \rangle$  is the injection and not simply a mono). Similarly  $b_i(R(\sigma)(f,g)) = G(\sigma)b_h(f,g) = G(\sigma)(g)$ .

<sup>&</sup>lt;sup>1</sup> The triple (R, a, b) is a monic span (or monic pair) in a category C if for all  $h, i \in C$  it holds that h = i whenever h; a = i; a and h; b = i; b do.

The second requirement of Definition 4 (together with the fact that R is symmetric) coincides with requiring the existence of a  $\gamma$  making the above diagram commute. This is a standard reasoning in the theory of coalgebras. The interested reader is referred to [33, Example 2.1].

The correspondence between inside-outside bisimulations and *B*-bisimulatios is now evident. Both definitions require that the same equations must hold in the compared processes (this is implicitly expressed in  $\mathbb{E}$ -bisimulations by requiring that the relation is indexed over  $\mathbb{E}$ ). Moreover, both definitions require that the relation is closed with respect to name fusions (this is equivalent to requiring that the relation is closed under all arrows of  $\mathbb{E}$ ). And finally, the cases of bound input and bound output (expressed in inside-outside bisimulation by bn  $(\alpha) \cap \text{fn}(Q) = \emptyset$ ) is safely tackled by considering as the bound name a new name for both processes.

#### 5.2 A Further Abstraction

We round up the section by showing an alternative definition of the behavioural functor. The definition would allow one to observe directly the classes of name equivalences, instead of the names themselves.

Let  $Equiv : \mathbb{E} \to Set$  be the functor defined as  $Equiv(h : s \to t) = t$  and  $Equiv(\langle \sigma_1, \sigma_2 \rangle) = \sigma_2$ . The behavioural endofunctor  $B : Set^{\mathbb{E}} \to Set^{\mathbb{E}}$  is formally defined as

| BF | = | $\mathcal{P}_f($                     |                       |
|----|---|--------------------------------------|-----------------------|
|    |   | $F^{\Box}$                           | (tau action)          |
|    | + | $Equiv \times Equiv \times F^{\Box}$ | (input action)        |
|    | + | $Equiv \times Equiv \times F^{\Box}$ | (output action)       |
|    | + | $Equiv \times F^{\perp^{\square}}$   | (bound input action)  |
|    | + | $Equiv \times F^{\perp^{\square}}$   | (bound output action) |
|    |   | )                                    |                       |

¿From one side, this implicitly mimics the rule FUS, by forcing all processes to perform actions with equivalent names. On the other hand, this could be useful as an efficient characterization, since there would only be one transition for any two actions from a process P, as long as they are identified by the equivalence class Eq(P). We leave the exploration of this functor as future work.

### 6 Conclusions and Further Work

In this paper, we introduced the category  $\mathbb{E}$  of equivalence classes, and we started the study of the presheaf category  $Set^{\mathbb{E}}$ . Some preliminary results, summed up in the last section, show that the category  $Set^{\mathbb{E}}$  seems the right universe for providing denotational models for the fusion calculus.

Much work remains to be done. First of all, our propositions just proved that  $Set^{\mathbb{E}}$  is the right category for discussing inside-outside bisimulation, but it is yet to be proved that the operational semantics induced by the rules in

Fig. 1(c) coincides with the unique morphism from the initial  $\Sigma$ -algebra  $T_{\Sigma}$  to  $B(T_{\Sigma})$  of our construction. The main problem concerns the fact that the operational semantics of explicit fusion calculus is not compositional with respect to name fusion, and thus it does not correspond to any arrow in  $Set^{\mathbb{E}}$  (natural transformation). A possible way out of the empasse could be to consider the category of functors from  $\mathbb{E}$  to Set and lax natural transformations. Another solution might consist in considering the *context transition system* in the spirit of [29,23]: for all  $\sigma \in \mathbb{E}$ ,  $p \xrightarrow{\sigma,l} p'$  if and only if  $\sigma(p) \xrightarrow{l} p'$ . It is evident that such an operational semantics is clearly compositional with respect to all fusions, and thus it is trivially an arrow in  $Set^{\mathbb{E}}$ .

The latter solution could lead us to a coalgebraic characterization of the so called *efficient bisimulation* [38, Definition 9]. There, as in the case of *open bisimulation* [34], instead of considering transitions  $p \xrightarrow{\sigma,l} p'$  for all possible  $\sigma$ , only the *minimal*  $\sigma$ 's are considered. This is also similar to *reactive systems*, as proposed by Leifer and Milner [23]. The exact correspondence with this approach is shown in [3] for the case of open  $\pi$ -calculus, and can be trivially extended to explicit fusion calculus. Unfortunately, the definition of efficient bisimulation is asymmetric and thus it seems hard to characterize it through canonical coalgebras. Probably, *normalized coalgebras* [2] can be fruitfully employed for this aim.

In more general terms, we would like to have a better understanding of the properties of  $Set^{\mathbb{E}}$ . In particular, we plan to check if alternative characterisations exist, mimicking the correspondence between nominal sets and Schanuel topos holding for the subcategory of  $Set^{\mathbb{I}}$  of pullback-preserving functors. As a start, we noticed that in  $\mathbb{E}$  pullbacks exist only along regular monomorphims.

As a next step, we would like to address those calculi featuring distinctions, such as D-Fusion [4] and the open semantics for  $\pi$ -calculus [34]. A suitable denotational model could be obtained by considering the category  $\mathbb{D}$  of irreflexive graphs [24]. Given the injection  $IN_{\mathbb{D}} : \mathbb{D} \to Set$ , we should then study the comma category  $IN_{\mathbb{I}} \downarrow IN_{\mathbb{D}}$ , thus equipping each equivalence class (and each pair of names belonging to them) with a suitable irreflexive relation.

## References

- Bonchi, F., König, B., Montanari, U.: Saturated semantics for reactive systems. In: Proc. of LICS, pp. 69–80. IEEE, Los Alamitos (2006)
- Bonchi, F., Montanari, U.: Coalgebraic models for reactive systems. In: Kok, J.N., Koronacki, J., Lopez de Mantaras, R., Matwin, S., Mladenič, D., Skowron, A. (eds.) ECML 2007. LNCS (LNAI), vol. 4701, pp. 364–380. Springer, Heidelberg (2007)
- Bonchi, F., Montanari, U.: Symbolic semantics revisited. In: Amadio, R. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 395–412. Springer, Heidelberg (2008)
- Boreale, M., Buscemi, M.G., Montanari, U.: D-fusion: A distinctive fusion calculus. In: Chin, W.-N. (ed.) APLAS 2004. LNCS, vol. 3302, pp. 296–310. Springer, Heidelberg (2004)
- Boreale, M., Sangiorgi, D.: Some congruence properties for pi-calculus bisimilarities. Theoret. Comput. Sci. 198(1-2), 159–176 (1998)

- Buscemi, M.G., Montanari, U.: A first order coalgebraic model of pi-calculus early observational equivalence. In: Brim, L., Jančar, P., Křetínský, M., Kucera, A. (eds.) CONCUR 2002. LNCS, vol. 2421, pp. 449–465. Springer, Heidelberg (2002)
- Buscemi, M.G., Montanari, U.: Cc-pi: A constraint-based language for specifying service level agreements. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 18–32. Springer, Heidelberg (2007)
- Buscemi, M.G., Montanari, U.: A compositional coalgebraic model of fusion calculus. J. Log. Algebr. Program. 72(1), 78–97 (2007)
- Ciancia, V., Montanari, U.: A name abstraction functor for named sets. In: Proc. of CMCS. Elect. Notes in Th. Comput. Sci (to appear, 2008)
- Ferrari, G.L., Montanari, U., Pistore, M.: Minimizing transition systems for name passing calculi: A co-algebraic formulation. In: Nielsen, M., Engberg, U. (eds.) FOSSACS 2002. LNCS, vol. 2303, pp. 129–158. Springer, Heidelberg (2002)
- Ferrari, G.L., Montanari, U., Tuosto, E.: Coalgebraic minimization of HDautomata for the pi-calculus using polymorphic types. Theoret. Comput. Sci. 331(2-3), 325–365 (2005)
- Ferrari, G.L., Montanari, U., Tuosto, E., Victor, B., Yemane, K.: Modelling fusion calculus using HD-automata. In: Fiadeiro, J.L., Harman, N.A., Roggenbach, M., Rutten, J. (eds.) CALCO 2005. LNCS, vol. 3629, pp. 142–156. Springer, Heidelberg (2005)
- 13. Fiore, M.P., Moggi, E., Sangiorgi, D.: A fully abstract model for the  $\pi$ -calculus. Inf. Comput. 179(1), 76–117 (2002)
- Fiore, M.P., Plotkin, G.D., Turi, D.: Abstract syntax and variable binding. In: Proc. of LICS, pp. 193–202 (1999)
- Fiore, M.P., Turi, D.: Semantics of name and value passing. In: Proc. of LICS, pp. 93–104. IEEE, Los Alamitos (2001)
- Gabbay, M., Pitts, A.M.: A new approach to abstract syntax with variable binding. Formal Asp. Comput. 13(3-5), 341–363 (2002)
- Gadducci, F., Miculan, M., Montanari, U.: About permutation algebras (pre)sheaves and named sets. Higher-Order and Symbolic Computation 19(2-3), 283–304 (2006)
- Gadducci, F., Montanari, U.: Graph processes with fusions: Concurrency by colimits, again. In: Kreowski, H.-J., Montanari, U., Orejas, F., Rozenberg, G., Taentzer, G. (eds.) Formal Methods in Software and Systems Modeling. LNCS, vol. 3393, pp. 84–100. Springer, Heidelberg (2005)
- Ghani, N., Yemane, K., Victor, B.: Relationally staged computations in calculi of mobile processes. In: The Programming Language Ada. LNCS, vol. 106, pp. 105–120. Springer, Heidelberg (1981)
- Giarratana, V., Gimona, F., Montanari, U.: Observability concepts in abstract data type specifications. In: Mazurkiewicz, A. (ed.) MFCS 1976. LNCS, vol. 45, pp. 576–587. Springer, Heidelberg (1976)
- Hofmann, M.: Semantical analysis of higher-order abstract syntax. In: Proc. of LICS, pp. 204–213. IEEE, Los Alamitos (1999)
- Lanese, I., Montanari, U.: Mapping fusion and synchronized hyperedge replacement into logic programming. Theory Pract. Log. Program. 7(1-2), 123–151 (2007)
- Leifer, J.J., Milner, R.: Deriving bisimulation congruences for reactive systems. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 243–258. Springer, Heidelberg (2000)
- Miculan, M., Yemane, K.: A unifying model of variables and names. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 170–186. Springer, Heidelberg (2005)

- Milner, R.: Communication and Concurrency. Prentice-Hall, Englewood Cliffs (1989)
- Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, I and II. Inform. and Comput. 100(1), 1–40 (1992)
- Montanari, U., Pistore, M.: An introduction to history dependent automata. In: Proc. of HOOTS. Elect. Notes in Th. Comput. Sci, vol. 10, pp. 170–188 (1997)
- Montanari, U., Pistore, M.: Structured coalgebras and minimal HD-automata for the pi-calculus. Theoret. Comput. Sci. 340(3), 539–576 (2005)
- Montanari, U., Sassone, V.: Dynamic congruence vs. progressing bisimulation for CCS. Fundamenta Informaticae 16(1), 171–199 (1992)
- Parrow, J., Victor, B.: The fusion calculus: Expressiveness and symmetry in mobile processes. In: Proc. of LICS, pp. 176–185. IEEE, Los Alamitos (1998)
- Pistore, M.: History Dependent Automata. PhD thesis, Università di Pisa, Dipartimento di Informatica, Available at University of Pisa as PhD. Thesis TD-5/99 (1999)
- 32. Plotkin, G.: A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, Computer Science Department (1981)
- Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. Theoret. Comput. Sci. 249(1), 3–80 (2000)
- 34. Sangiorgi, D.: A theory of bisimulation for the  $\pi$ -calculus. Acta Inform. 33(1), 69–97 (1996)
- Scott, D., Strachey, C.: Toward a mathematical semantics for computer languages. In: Programming Research Group Technical Monograph, Oxford University, Computing Laboratory, vol. PRG-6 (1971)
- 36. Stark, I.: A fully abstract domain model for the  $\pi$ -calculus. In: Proc. of LICS, pp. 36–42. IEEE, Los Alamitos (1996)
- Turi, D., Plotkin, G.D.: Towards a mathematical operational semantics. In: Proc. of LICS, pp. 280–291. IEEE, Los Alamitos (1997)
- Wischik, L., Gardner, P.: Strong bisimulation for the explicit fusion calculus. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, pp. 484–498. Springer, Heidelberg (2004)
- Wischik, L., Gardner, P.: Explicit fusions. Theoret. Comput. Sci. 340(3), 606–630 (2005)