Modelling and Minimising the Fusion calculus using HD-automata*

Gianluigi Ferrari[1], Ugo Montanari[1], Emilio Tuosto[1], Björn Victor[2], and Kidane Yemane[2]

[1] Dipartimento di Informatica, Università di Pisa, Italy.
[2] Dept. of Information Technology, Uppsala University, Sweden.

Abstract. We present a partition refinement algorithm for the Fusion calculus. The algorithm is based on HD-automata and exploits a co-algebraic functor representing a symbolic semantics of the Fusion calculus. HD-automata have have successfully been applied for monadic versions of π-calculus and have lead to Mihda, an implementation based on a polymorphic lambda calculus.
Interestingly enough, HD-automata can be exploited also for representing Fusion calculus which, differently from π-calculus, allows for global fusions of names. Some adjustments are necessary in order to deal with the semantics and polyadicity and of Fusion.
However, the adaptation to HD-automata does not affect the control part of the algorithm; indeed the algorithm for Fusion is obtained by exploiting similar results for the pi-calculus. Moreover, we can implement the minimisation algorithm for Fusion simply by adapting a few modules of Mihda. Polymorphism and the modularity of Midha allows for a smooth implementation.

1 Syntax and Semantics of the Fusion Calculus

In this section we first review the syntax and operational semantics of the Fusion calculus with match and recursion, and then its strong bisimulation equivalence and congruence from [7]. The novel contribution of the present paper begins by exploring the canonical symbolic semantics.

We assume an infinite enumerable and totally ordered set $\mathcal{N}$ of *names* ranged over by $u, v, \ldots, z$. Like in the π-calculus, names represent communication channels, which are also the values transmitted. We let φ range over total equivalence relations over $\mathcal{N}$ (i.e. equivalence relations with $\mathrm{dom}(\varphi) = \mathcal{N}$) with only finitely many non-singular equivalence classes. We write $\tilde{x}$ for a (possibly empty) finite sequence $x_1 \cdots x_n$ of names and $\{\tilde{x} = \tilde{y}\}$ to mean the smallest such equivalence relation relating each x_i with y_i, and write $\mathbf{1}$ for the identity relation. If x and y are related by φ we write $x \; \varphi \; y$.

* Work supported by the PROFUNDIS FET-GC project.

Definition 1 (Fusion calculus syntax). *The* free actions*, ranged over by* α*, and the* agents*, ranged over by* $P, Q, \ldots$*, are defined by*

$$\begin{array}{lll} \alpha ::= u\tilde{x} & \text{(Input)} \\ \quad\;\; \overline{u}\tilde{x} & \text{(Output)} \\ \quad\;\; \varphi & \text{(Fusion)} \end{array} \qquad P ::= \mathbf{0} \;\mid\; \alpha \,.\, Q \;\mid\; Q + R \;\mid\; Q \mid R \;\mid\; (x)Q \;\mid\; [x = y]Q \;\mid\; A\langle\tilde{x}\rangle.$$

In the Fusion calculus prefixes correspond exactly to the free actions defined above. Input and output actions are collectively called *free communication* actions. The names $\tilde{x}$ in $\overline{u}\tilde{x}$ and $u\tilde{x}$ are the *objects* of the action, and the name u is the *subject*. We write a to stand for either u or $\overline{u}$, thus $a\tilde{x}$ is the general form of a communication action. Given a communication action γ, we let $\mathrm{sbj}(\gamma)$ and $\mathrm{obj}(\gamma)$ to respectively denote the subject and the object of γ. Fusion actions have neither subject nor objects.

An input action $u\tilde{x}$ means "input objects along the port u and replace $\tilde{x}$ with these objects." Contrary to the situation in the π-calculus this action does not bind $\tilde{x}$. In other words, the scope of $\tilde{x}$ is not bounded by the action. The output action $\overline{u}\tilde{x}$ is familiar from the π-calculus and means "output the objects $\tilde{x}$ along the port u". A fusion action $\{\tilde{x} = \tilde{y}\}$ represents an obligation to make $\tilde{x}$ and $\tilde{y}$ equal everywhere.

The name x is said to be *bound* in $(x)P$. We write $(\tilde{x})P$ for $(x_1)\cdots(x_n)P$. The *free names* in P, denoted $\mathrm{fn}(P)$, are the names in P with a non-bound occurrence; here $\mathrm{fn}(\varphi)$ is defined to be the names in the non-singular equivalence classes, i.e., in the relation $\varphi - \mathbf{1}$. Notice that since the syntax $\{x = x\}$ means $\mathbf{1}$, the free names of that fusion do not include x, but is the empty set. As usual we will not distinguish between alpha-variants of agents, i.e., agents differing only in the choice of bound names.

Definition 2 (Fusion calculus labels). *The* labels*, ranged over by* γ*, consist of the fusion actions and of communication actions of the form* $(z_1)\cdots(z_n)a\tilde{x}$ *(written* $(\tilde{z})a\tilde{x}$*), where* $n \geq 0$ *and all names in* $\tilde{z}$ *are also in* $\tilde{x}$*. If* $n > 0$ *we say it is a* bound *label, otherwise it is a* free *label.*

The labels of a transition may be free or bound (note that there are no bound fusion labels). In the communication labels above, $\tilde{z}$ are the *bound objects* and the names in $\tilde{x}$ that are not in $\tilde{z}$ are the *free objects*. Free labels have no bound objects. We further write $\mathrm{n}(\gamma)$ to mean all names occurring in γ, both bound and free.

We let M, N, L range over sequences of match operators, and we say that M (*strictly*) *implies* N, written $M \Rightarrow N$ ($M \not\Longleftarrow\!\!\Rightarrow N$), if the conjunction of all matches in M logically implies all elements in N (and $N \not\Rightarrow M$). Moreover, $M \Leftrightarrow N$ means that M and N imply each other.

We define $\varphi \backslash z$ to mean $\varphi \cap (\mathcal{N} - \{z\})^2 \cup \{(z, z)\}$, i.e., the equivalence relation φ with all references to z removed (except for the identity). For example, $\{x = z, z = y\} \backslash z = \{x = y\}$, and $\{x = y\} \backslash y = \mathbf{1}$.

We consider Fusion calculus agents up to the structural congruence equivalence defined below.

$$\text{PREF}\ \frac{-}{\alpha\,.\,P \xrightarrow{\alpha} P} \qquad \text{SUM}\ \frac{P \xrightarrow{\gamma} P'}{P+Q \xrightarrow{\gamma} P'}$$

$$\text{PAR}\ \frac{P \xrightarrow{\gamma} P'}{P \mid Q \xrightarrow{\gamma} P' \mid Q} \qquad \text{COM}\ \frac{P \xrightarrow{u\tilde{x}} P',\ \ Q \xrightarrow{\overline{u}\tilde{y}} Q',\ \ |\tilde{x}| = |\tilde{y}|}{P \mid Q \xrightarrow{\{\tilde{x}=\tilde{y}\}} P' \mid Q'}$$

$$\text{SCOPE}\ \frac{P \xrightarrow{\varphi} P',\ z\,\varphi\,x,\ z \neq x}{(z)P \xrightarrow{\varphi\backslash z} P'\{x/z\}} \qquad \text{PASS}\ \frac{P \xrightarrow{\gamma} P',\ z \notin \mathrm{n}(\gamma)}{(z)P \xrightarrow{\gamma} (z)P'}$$

$$\text{OPEN}\ \frac{P \xrightarrow{(\tilde{y})a\,\tilde{x}} P',\ z \in \tilde{x} - \tilde{y},\ a \notin \{z, \overline{z}\}}{(z)P \xrightarrow{(z\tilde{y})a\,\tilde{x}} P'}$$

$$\text{MATCH}\ \frac{P \xrightarrow{\gamma} P'}{[x=x]P \xrightarrow{\gamma} P'} \qquad \text{STRUCT}\ \frac{P \equiv P' \quad P \xrightarrow{\gamma} Q \quad Q \equiv Q'}{P' \xrightarrow{\gamma} Q'}$$

Table 1. Transition rules for the Fusion Calculus.

Definition 3 (Structural congruence). *The* structural congruence*, $\equiv$, between agents is the least congruence satisfying the abelian monoid laws for $+$ and $\mid$ operators (associativity, commutativity and* $\mathbf{0}$ *as identity), and the scope laws $(x)\mathbf{0} \equiv \mathbf{0}$, $(x)(y)P \equiv (y)(x)P$, $(x)(P+Q) \equiv (x)P + (x)Q$, $(x)MP \equiv M(x)P$ if $x \notin \mathrm{n}(M)$ and also the* scope extension *law $P \mid (z)Q \equiv (z)(P \mid Q)$ where $z \notin \mathrm{fn}(P)$, and the law for process identifiers: $A\langle\tilde{y}\rangle \equiv P\{\tilde{y}/\tilde{x}\}$ if $A(\tilde{x}) \stackrel{def}{=} P$.*

We proceed by giving the labelled transition semantics of Fusion calculus, with labels as defined in Definition 2.

Definition 4 (Fusion calculus semantics). *The labelled transition system of Fusion calculus is the least relation satisfying the inference rules in Table 1. In this definition structurally equivalent agents are considered the same, i.e., if $P \equiv P'$ and $Q \equiv Q'$ and $P \xrightarrow{\gamma} Q$ then also $P' \xrightarrow{\gamma} Q'$.*

Use of the SCOPE rule entails a substitution of the scoped name z for a nondeterministically chosen name x related to it by φ. For the purpose of the equivalence defined below it will not matter which such x replaces z. The only rule dealing with bound actions is OPEN. Using structural congruence, pulling the relevant scope to top level, we can still infer e.g. $P \mid (x)ayx\,.\,Q \xrightarrow{(x)ayx} P \mid Q$ using PREF and OPEN (provided $x \notin \mathrm{fn}(P)$, otherwise an alpha-conversion is necessary).

Definition 5 (Substitutions). *A substitution σ* agrees with *the fusion φ if $\forall x,y : x\,\varphi\,y \Leftrightarrow \sigma(x) = \sigma(y)$. A* substitutive effect *of a fusion (or generally an equivalence relation) φ is an idempotent substitution σ agreeing with φ such that $\forall x,y : \sigma(x) = y \Rightarrow x\,\varphi\,y$ (i.e., σ sends all members of the equivalence class to one representative of the class). For any action or equivalence relation γ we write σ_γ for its substitutive effect.*

The only substitutive effect of a communication action is the identity substitution. The substitutive effects of $\{x = y\}$ are $\{x/y\}$ and $\{y/x\}$. Note that not all substitutions which agree with φ are substitutive effects of φ. For example any injective substitution agrees with $\mathbf{1}$, but the only substitutive effect of $\mathbf{1}$ is the identity substitution.

Lemma 1. *If $P \xrightarrow{\gamma} P'$, then $P\sigma \xrightarrow{\gamma\sigma} P'\sigma$, for any substitution σ.*

We end this section by reviewing the definition of *hyperbisimulation* [7].

Definition 6 (Hyperbisimulation). *A* fusion bisimulation *is a binary symmetric relation $\mathcal{S}$ between agents such that $(P, Q) \in \mathcal{S}$ implies:*
If $P \xrightarrow{\gamma} P'$ with $\mathrm{bn}(\gamma) \cap \mathrm{fn}(Q) = \emptyset$, then $Q \xrightarrow{\gamma} Q'$ and $(P'\sigma_\gamma, Q'\sigma_\gamma) \in \mathcal{S}$. Agents P and Q are fusion bisimilar*, written $P \stackrel{\cdot}{\sim} Q$, if $(P, Q) \in \mathcal{S}$ for some fusion bisimulation $\mathcal{S}$.*

A hyperbisimulation *is a substitution closed fusion bisimulation, i.e., a fusion bisimulation $\mathcal{S}$ with the property that $(P, Q) \in \mathcal{S}$ implies $(P\sigma, Q\sigma) \in \mathcal{S}$ for any substitution σ. Two agents P and Q are* hyperequivalent*, written $P \sim Q$, if they are related by a hyperbisimulation.*

The interesting point in this definition is the treatment of fusion actions. A fusion $\{x = y\}$ represents an obligation to make x and y equal everywhere. Therefore, if γ above is such a fusion, it only makes sense to relate P' and Q' when a substitution $\{y/x\}$ or $\{x/y\}$ has been performed. Note that it does not matter which substitution we choose, since $P\{x/y\} \stackrel{\cdot}{\sim} Q\{x/y\}$ implies $P\{y/x\} \stackrel{\cdot}{\sim} Q\{y/x\}$, by the simple fact that $P\{x/y\}\{y/x\} \equiv P\{y/x\}$ and that bisimulation is closed under injective substitutions.

Theorem 1. [7] *Hyperequivalence is the largest congruence in fusion bisimilarity.*

For further examples and explanations we refer the reader to [7, 3].

2 Canonical Symbolic Semantics of Fusion Calculus

In this section we present a *canonical* symbolic semantics for the Fusion calculus, along the lines of symbolic semantics for the π-calculus [11, 9]. Symbolic semantics are often used to give efficient characterizations of bisimulation equivalences for value-passing calculi.

In Table 2 we present the symbolic transition system. Again, structurally equivalent agents are considered the same. Like in [11] a symbolic transition is of the form $P \overset{M,\gamma}{\longmapsto} Q$, where M is the enabling condition of the action γ in the sense that M represents the equalities a minimal substitution σ_M must make true in order for $P\sigma_M$ to perform the corresponding action in the original labelled transition system. Formally, σ_M is a substitutive effect of the equivalence relation induced by M.

$$\text{S-PREF}\ \frac{-}{\alpha\,.\,P \overset{\emptyset,\alpha}{\longmapsto} P\sigma_\alpha} \qquad \text{S-SUM}\ \frac{P \overset{M,\gamma}{\longmapsto} P'}{P+Q \overset{M,\gamma}{\longmapsto} P'}$$

$$\text{S-PAR}\ \frac{P \overset{M,\gamma}{\longmapsto} P'}{P \mid Q \overset{M,\gamma}{\longmapsto} P' \mid Q\sigma_M\sigma_\gamma}$$

S-COM

$$\frac{P \overset{M,u\tilde{x}}{\longmapsto} P',\ \ Q \overset{N,\overline{v}\tilde{y}}{\longmapsto} Q',\ \ |\tilde{x}| = |\tilde{y}|,\ \ L = M \star N \star [u = v],\ \varphi = \{\tilde{x} = \tilde{y}\}\sigma_L}{P \mid Q \overset{L,\varphi}{\longmapsto} (P' \mid Q')\sigma_L\sigma_\varphi}$$

$$\text{S-SCOPE}\ \frac{P \overset{M,\varphi}{\longmapsto} P',\ z\,\varphi\,x,\ z \neq x,\ z \notin \mathrm{n}(M)}{(z)P \overset{M,\varphi\setminus z}{\longmapsto} P'\{x/z\}}$$

$$\text{S-PASS}\ \frac{P \overset{M,\gamma}{\longmapsto} P',\ z \notin \mathrm{n}(M,\gamma)}{(z)P \overset{M,\gamma}{\longmapsto} (z)P'}$$

$$\text{S-OPEN}\ \frac{P \overset{M,(\tilde{y})a\,\tilde{x}}{\longmapsto} P',\ z \in \tilde{x} - \tilde{y},\ a \notin \{z, \overline{z}\},\ z \notin \mathrm{n}(M)}{(z)P \overset{M,(z\tilde{y})a\,\tilde{x}}{\longmapsto} P'}$$

$$\text{S-MATCH}\ \frac{P \overset{M,\gamma}{\longmapsto} P'}{[x = y]P \overset{M\star[x=y],\gamma\sigma_{[x=y]}}{\longmapsto} P'\sigma_{M\star[x=y]}}$$

Table 2. Canonical symbolic transition system for the Fusion Calculus.

In Table 2 we write $M \star N$ for denoting the concatenation of M and N. In Section 5 we will give a different interpretation of $\star$. For the moment, since no confusion arises, we indifferently write MN or $M \star N$.

Following Pistore and Sangiorgi's work [9], our transition rules apply substitutions to the continuation of a transition: Like [9], a substitution σ_M, making the condition for the transition true, and in addition a substitution σ_γ, the substitutive effect of the action. The motivation for this is to make the definition of bisimulation simpler and more in line with the algorithms used in the HD framework (see Section 5). We show later in this section that bisimulation using the symbolic semantics coincides with the original non-symbolic version.

While we can in general use any σ_M which agrees with M and which is identity for all names not in M, and any substitutive effect σ_γ of the action, like Pistore and Sangiorgi [9] we choose *canonical* substitutions, which selects the *minimal* representative of each equivalence class of M or γ and maps all members to it. Henceforth, σ_M and σ_γ refer to canonical substitutions. Note that hyperequivalence (Definition 6) does not change, since we can use any substitutive effect there.

Using canonical substitutions gives us pleasant properties like the following:

Lemma 2. *If* $P \xmapsto{M,\gamma} P'$, *then* $\gamma = \gamma\sigma_M$ *and* $P' = P'\sigma_M = P'\sigma_\gamma = P'\sigma_M\sigma_\gamma$.

The definition of symbolic hyperbisimulation is similar to that of symbolic open bisimulation [11, 9], but does not have the complication of distinctions.

Definition 7 (Symbolic hyperbisimulation). *A binary symmetric process relation* $\mathcal{S}$ *is a* symbolic hyperbisimulation *if* $(P, Q) \in \mathcal{S}$ *implies:*

If $P \xmapsto{M,\gamma} P'$ *with* $\mathrm{bn}(\gamma) \cap \mathrm{fn}(Q) = \emptyset$ *then*
$Q \xmapsto{N,\gamma'} Q'$ *such that*
$M \Rightarrow N$,
$\gamma = \gamma'\sigma_M$, *(note* $\gamma = \gamma\sigma_M$*)*
and $(P', Q'\sigma_M) \in \mathcal{S}$ *(note* $P' = P'\sigma_M$*).*

P *is* symbolically hyperequivalent *to* Q, *written* $P \simeq Q$, *if* $(P, Q) \in \mathcal{S}$ *for some symbolic hyperbisimulation* $\mathcal{S}$.

Since the symbolic semantics applies the substitutive effects, we can leave most of that out of the bisimulation definition. It is still necessary to apply substitution corresponding to the stronger condition, σ_M, to the label and continuation of the transition of Q. (Note that $Q'\sigma_M = Q'\sigma_M\sigma_\gamma$.)

In the remainder of this section we establish the correspondence between symbolic hyperequivalence (Definition 7) and the standard hyperequivalence (Definition 6) by proving Theorem 2: $P \sim Q$ iff $P \simeq Q$.

Lemma 3.

1. $\sigma\sigma_{R\sigma} = \sigma_R\rho$, *for any substitution* σ *and some* ρ, *where* R *is an equivalence relation.*
2. *If* $M \Rightarrow N$ *then for any substitution* σ, $M\sigma = N\sigma\rho$, *for some substitution* ρ.
3. $\sigma_R\sigma_{S\sigma_R} = \sigma_S\sigma_R$, *where* R *and* S *are equivalence relations.*

Lemma 4.

1. *If* $P \xmapsto{M,\gamma} P'$, *then* $P\sigma \xmapsto{M\sigma,\gamma\sigma} P'\sigma\sigma_{M\sigma}\sigma_{\gamma\sigma}$.
2. *if* $P\sigma \xmapsto{N,\gamma'} P'$, *then* $P \xmapsto{M,\gamma} P''$ *with* $M\sigma \Leftrightarrow N$, $\gamma\sigma = \gamma'$, *and* $P' = P''\sigma\sigma_N\sigma_{\gamma'}$.

Proof. By transition induction, using Lemma 3. □

Lemma 5. $P \simeq Q$ *implies* $P\sigma \simeq Q\sigma$, *for any substitution* σ.

Proof. Straightforward diagram chasing, using Lemmas 3 and 4. □

Lemma 6.

1. *If* $P \xmapsto{M,\gamma} P'$, *then* $P\sigma_M \xrightarrow{\gamma} P''$ *s.t.* $P' = P''\sigma_\gamma$;
2. *if* $M \Rightarrow N$ *and* $P\sigma_M \xrightarrow{\gamma} P'$, *then* $P \xmapsto{N,\gamma'} P''$ *such that* $\gamma = \gamma'\sigma_M$ *and* $P'\sigma_\gamma = P''\sigma_M$.

Proof. Again by transition induction, using Lemmas 3 and 1. □

Theorem 2. *$P \sim Q$ iff $P \simeq Q$*

Proof of $\Rightarrow$: by showing $\mathcal{S} = \{(P, Q) : P \sim Q\}$ is a symbolic hyperbisimulation, using Lemmas 6 and 3. □

Proof of $\Leftarrow$: We already have closure under substitution (Lemma 5), and show that $\mathcal{S} = \{(P, Q) : P \simeq Q\}$ is a fusion bisimulation using Lemmas 6 and 3. □

3 Types for HD-automata

Verification of systems that are modelled as mobile processes (i.e. concurrent systems whose communication topology may change dynamically) is hard. For instance, in Fusion calculus, agents can perform transitions that generate new names, e.g. the OPEN rule in Table 1 establishes that an agent, $(z)P$, can create a new name and can extrude it. The interplay between the OPEN and the STRUCT rules introduce an infinite branching in the automata corresponding to agents that perform such transitions since z can be freely replaced with any name in $\mathcal{N} \setminus \mathrm{fn}(P)$.

History Dependent automata (HD-automata in brief) have been proposed in [8, 5, 6, 1] as a new operational model for history dependent calculi, namely those calculi whose semantics is defined in terms of a labelled transition system where labels may carry information generated in the past transitions of the system and this "historical" information can influence the future behaviour of the system.

HD-automata aim at giving a finite representation of otherwise infinite label transition systems. Similarly to ordinary automata, HD-automata are made out of states and labelled transitions. Their peculiarity resides in the fact that states and transitions are equipped with names which are no longer dealt as syntactic components of labels, but become an explicit part of the operational model. This permits to model name creation/deallocation or name extrusion that are typical linguistic mechanisms of name passing calculi.

Noteworthy, names in states of HD-automata have *local meaning*. For instance, if $A(x, y, z)$ denotes an agent having three free names x, y and z, then agent $A(y, x, z)$ is different from $A(x, y, z)$, however, they can be both represented by means of a single state, say q, in a HD-automaton simply by considering a "swapping" operation on the names (corresponding to) x and y of q. More generally, states that differs only for renaming of their local names are identified in the operational model.

Local meaning of names requires a mechanism for describing how names correspond each other along transitions. Graphically, we can represent such correspondences using "wires" that connect names of label, source and target states of transitions. For instance, Figure 1 depicts a transition from source state s to destination state d. State s has three names, 1, 2 and 3 while d has two names 4 and 5 which correspond to name 1 of s and to the new name 0, respectively.

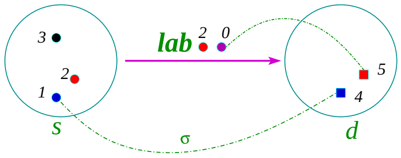

Fig. 1. A HD-automaton transition

The transition is labelled by lab and exposes two names: Name 2 of s and a fresh name 0. Notice that names 3 in s is "discharged" along such transition.

As described in Figure 1, HD-automata relies on the fact that names are local. This allows for a compact representation of agent behaviour by collapsing states that differ only for renaming of local names and encompasses the main characteristics of name-passing calculi, namely, creation/deallocation of names. Indeed, name creation is simply handled by associating in the target state a name not in the source state, while a name in a state s can be deallocated when it is not involved in any transition from s.

A HD-automaton associates a "history" to names of the states appearing in the computation, in the sense that it is possible to reconstruct the associations which lead to the state containing the name. Clearly, if a state is reached in two different computations, different histories could be assigned to its names.

3.1 A polymorphic λ-calculus

This section reviews some basic type-theoretic notions underlying the description of languages which support the organisation of applications into autonomous (compilable) modules exploiting explicit type information (e.g. the ML module language). We refer to [4] for a detailed introduction to these issues.

A polymorphic type describes a structure "having many types". Two powerful constructs to describe polymorphic types are the *general product and sum* types. A function type $t \to t'$ describes the type of a function mapping elements of t into elements of t'. Sometimes to specify the dependence of the result type on the value of the argument type (i.e. the type t' is an expression with free variable x of type t), the function type is written as $\prod_{x:t} t'$. This function type is called *general product* of t' over the index set t. The elements of this type are functions f such that $f(a) : t'[a/x]$, for each $a : t$.

A type $\langle t, t' \rangle$ describes a pair whose components are elements of type t and elements of type t'. When the value of the type of the first component determines the value of the type of the second component (i.e. the type t' is an expression with free variable x of type t) the pair type is written as $\displaystyle\sum_{x:t} t'$. This type is called *general sum* of t' over the index set t. The elements of this types are pairs $\langle a, b \rangle$ with $a : t$ and $b : t'[a/x]$. General sums are equipped with projections to extract their components.

Generic sum types encompass tuple types; indeed, $\sum_{x:t} t'$ is equivalent to $t \times t'$, if x does not occur free in t'. In these case, we will sometime write $t \times t'$ instead of $\sum_{x:t} t'$.

We now introduce $\lambda^{\rightarrow,\Pi,\Sigma}$ the predicative calculus with products and sums. We let U_1 and U_2 to denote the universes of *non*-polymorphic (basic) and polymorphic types, respectively. We assume that the universe of non-polymorphic types contains a collection of *type constructors* and it is closed under product and function space (notice that U_1 does not belong to U_2). This allows us to assume type constructors such as lists, trees or enumeration types over a type t as basic structures of our calculus. The universe of polymorphic types can be made as rich as the universe of basic types.

The syntax of (pre-)terms of $\lambda^{\rightarrow,\Pi,\Sigma}$ is given by:

$$
\begin{aligned}
M ::= U_1 \mid U_2 \mid bt \mid M \rightarrow M \mid \prod_{x:M} M \mid \sum_{x:M} M \\
\mid x \mid c \mid \lambda x : M.m \mid MM \\
\mid \langle x : M = M, M : M \rangle \mid \mathrm{I}(M) \mid \mathrm{II}(M)
\end{aligned}
$$

The first line of the grammar gives the syntax of the *type expressions*, the third line describes the structure associated with general sums; expressions $\mathrm{I}(M)$ and $\mathrm{II}(M)$ are the projections on the components of a pair (i.e. $\mathrm{I}(\langle M_1, M_2 \rangle) = M_1$ and $\mathrm{II}(\langle M_1, M_2 \rangle) = M_2$). The second line gives the productions of a (typed) λ-calculus. As usual, we will use τ to denote a term whenever the term is intended to be a type.

The type system is defined by type judgements $\Gamma \rhd M : \tau$ where Γ is type context of the form $\{x_1 : \tau_1, \ldots, x_n : \tau_n\}$, giving the types of variables $x_1, \ldots, x_n$. Any of the types τ_i can be a basic type or a polymorphic type. Type contexts have to satisfy some well-formedness constraints. Here, we do not present all the inference rules which express when a context is well formed. To give the flavour of the type system we present, instead, a sample of the typing rules. In particular, we will focus on general products and sums.

The following inference rules describes the conditions for forming and handling general products.

$$
\frac{\Gamma \rhd \tau : U_2}{\Gamma, x : \tau \rhd \tau' : U_2} \quad , \quad \Gamma \rhd \textstyle\prod_{x:\tau} \tau' : U_2 \qquad\qquad \frac{\Gamma \rhd M : \prod_{x:\tau} \tau' \qquad \Gamma \rhd N : \tau}{\Gamma \rhd MN : \tau'[N/x]}
$$

$$
\frac{\Gamma \rhd \tau : U_2 \qquad \Gamma, x : \tau \rhd \tau' : U_2 \qquad \Gamma, x : \tau \rhd M : \tau'}{\Gamma \rhd \lambda x : \tau.M \; : \; \prod_{x:\tau} \tau'}
$$

The four inference rules below provide the conditions for forming general sums and for handling the associated terms.

$$\frac{\Gamma \rhd \tau : U_2 \qquad \Gamma, x : \tau \rhd \tau' : U_2}{\Gamma \rhd \sum_{x:\tau} \tau' : U_2}$$

$$\frac{\Gamma \rhd M : \sum_{x:\tau} \tau'}{\Gamma \rhd \mathrm{I}(M) : \tau} \qquad\qquad \frac{\Gamma \rhd M : \sum_{x:\tau} \tau'}{\Gamma \rhd \mathrm{II}(M) : \tau'[\mathrm{I}(M)/x]}$$

$$\frac{\Gamma \rhd M : \tau \qquad \Gamma, x : \tau \rhd \tau' : U_2 \qquad \Gamma \rhd N[M/x] : \tau'[M/x]}{\Gamma \rhd \langle x : \tau = M, N : \tau' \rangle : \sum_{x:\tau} \tau'}$$

3.2 Types for HD-automata

Dependent types can faithfully describe HD-automata and the relationships among their states, transitions and names. For instance, the type for named sets yields the structure of a set of states and is a dependent sum type whose first component is any type T describing the elements of the named set[3] while the remaining components describe their characteristics, e.g. the number of names of each element and its symmetry. Notice that there are dependencies among those components, for instance names of elements have type $Q \to \omega$ and symmetries of elements have function type that maps elements in Q to finite sets of bijections on their names.

Table 3 borrows from [2] the types' definitions of named set and named functions. Let us remark that types allow us to abstract from meaningless details. For instance, we do not care about the type Q in `NS`, namely, elements of named sets have arbitrary type. Similarly to [2], instead of considering *abstract* names we exploit a concrete representation of names in terms of integers `I` (with the usual order).

A minor difference with respect to [2] is in the representation of names; in [2] names have been represented with natural numbers, ω. It will be more clear later that integers cope with polyadicity of Fusion calculus. Polyadicity of Fusion calculus is more appropriately dealt with integers than with naturals; indeed, negative integers will denote newly generated names and will "appear" in transition labels, while names local to states are strictly positive numbers. In ordert to represent finite sets of names, Table 3 defines the type `N`. Moreover, we will consider functions from totally ordered sets to `I`, hence we introduce the Definition 8 in Table 3. A named set A lives in a generalised sum type[4] whose

[3] Since our algorithm does not depend on the representation of states, named sets can (polymorphically) depend on *any* type.

[4] Formally, A should be written as $\langle Q_A, \langle \lesssim_A, \langle |\_|_A, \mathcal{G}_A \rangle \rangle \rangle$. However, when no confusion arises, we prefer the more compact notation $\langle Q_A, \lesssim_A, |\_|_A, \mathcal{G}_A \rangle$ that avoids writing

Types for names $\quad \mathtt{N} \triangleq \prod_{n:\omega} 1 \cdots n$

where ω is the type of positive integers.

Definition 8 **($\trianglelefteq$, Definition 3.1 of [2]).** *Given two totally ordered sets* $(A_1, \lesssim_1)$ *and* $(A_2, \lesssim_2)$*, we let* $\trianglelefteq$ *be the relation on functions from* A_1 *to* A_2 *such that* $f \trianglelefteq g$ *if, and only if,*

- *either,* $\forall q \in A_1.f(q) = g(q)$
- *or, there is* $q \in A_1$ *such that* $f(q) \neq g(q)$ *and* $\forall q' \in A_1.q' \lesssim_1 q \Rightarrow f(q') = g(q') \wedge f(q) \lesssim_2 g(q)$.

Types for named sets $\quad \mathtt{NS} \triangleq \sum_{Q:\mathtt{U}_2} \sum_{\lesssim:Q\times Q\to\mathtt{bool}} \sum_{|\_|:Q\to\omega} \prod_{q:Q} \wp_{\text{fin}}(\mathtt{N}(|q|) \to \mathtt{N}(|q|))$.

We write Q_A, $\lesssim_A$, $|\_|_A$ and $\mathcal{G}_A$ in place of using the unwieldy (and less readable) notation of $\lambda^{\to,\prod,\Sigma}$ based on projections $\mathrm{I}(\_)$ and $\mathrm{II}(\_)$. Given a named set A, in the following we write we write $a \in A$ instead of $a : Q_A$ and $q \lesssim_A q'$ ($q \not\lesssim_A q'$) instead of $\lesssim_A (q, q') = \mathtt{true}$ ($\lesssim_A (q, q') = \mathtt{false}$).

Definition 9 **(Named functions).** *The type of* named functions *is defined as follows:*

$$\mathtt{NF} \triangleq \sum_{S:\mathtt{NS}} \sum_{D:\mathtt{NS}} \sum_{h:Q_S\to Q_D} \prod_{q:Q_S} \wp_{\mathit{fin}}(\mathtt{N}(|h(q)|_D) \to \mathtt{N}(|q|_S))$$

It is worth fixing some notations that will help reading. Given a named function $H : \mathtt{NF}$*, we use the following abbreviations:*

- $\mathrm{dom}(H) = \mathrm{I}(H)$,
- $\mathrm{cod}(H) = \mathrm{I}(\mathrm{II}(H))$,
- $\mathrm{h}_H = \mathrm{I}(\mathrm{II}(\mathrm{II}(H)))$,
- $\Sigma_H = \mathrm{II}(\mathrm{II}(\mathrm{II}(H)))$,

which correspond to the projections of the sum type $\mathtt{NF}$*. We implicitly assume that, for all elements* $q \in \mathrm{dom}(H)$

1. $\forall \sigma \in \Sigma_H(q).\mathcal{G}_{\mathrm{cod}(H)}(\mathrm{h}_H(q)); \sigma = \Sigma_H(q)$,
2. $\forall \sigma \in \Sigma_H(q).\sigma; \mathcal{G}_{\mathrm{dom}(H)}(q) \subseteq \Sigma_H(q)$,
3. *any function of* $\Sigma_H(q)$ *is injective.*

Definition 10 **(Kernel of named functions).** *The* kernel of named function H *is the named set such that:*

- *The underlying set is* $\ker \mathrm{h}_H$*;*
- *the order relation, taken two classes* $A, B \in \ker \mathrm{h}_H$*, yields* `true` *if, and only if,* $\mathrm{h}_H(a) \lesssim_{\mathrm{cod}(H)} \mathrm{h}_H(b)$*, for* $a \in A$ *and* $b \in B$*;*
- *the weight of a class* $A \in \ker \mathrm{h}_H$ *is* $|\mathrm{h}_H(a)|_{\mathrm{cod}(H)}$*, for* $a \in A$*;*
- *the group of* $A \in \ker \mathrm{h}_H$ *is* $\mathcal{G}_{\mathrm{cod}(H)}(\mathrm{h}_H(a))$*, for* $a \in A$.

Table 3. Types for named sets and functions

first component is a type Q_A, the second component is again a sum type with a function $\lesssim_A$ that represents a total order on Q_A and will be used for determining the canonical representative in a set of of states; function $|\_|_A$ is called the weight function (of A) and associates the number of names to elements in Q_A; the generalised product type of the last component assigns a set of permutations of names of q.

Definition 10 generalises the concept of *kernel* of functions to named functions. The kernel $\ker H$ of a named function H is obtained by considering the partition induced by h_H on its domain and by exploiting the named set structure of $\mathrm{cod}(H)$ for the order, weight and group components. Notice that, in Definition 10, those components do not depend on the choice of a or b, since any element in $Q_{\ker H}$ is a set whose elements have the same image through h_H. The kernel of a named function is exploited for specifying *blocks*, the main data structure of the minimisation algorithm. Basically, a block collects those states considered equivalent at a generic iteration. Hence, the block *corresponds* to an element of the partition induced by the kernel of the named function associated to a generic step of the iterative algorithm.

Transitions among states are represented by means of *named functions* (see Definition 9 in Table 3). The type of named functions is a generalised sum type containing the named sets for source and destination (notice that the type of the destination does not depend on the type of the source), a mapping h from the source to the destination and, for each q in the source, there is a set of functions from the names of $h(q)$ to names of q. (conditions *1*, *2* and *3* in [2]).

We refer the reader to [2] for a deeper discussion.

4 Types for Fusion calculus

This section introduces the definitions of labels, transitions and bundles needed for defining HD-automata for Fusion calculus.

Definition 11 (Arity of actions). *We define*

$$\mathrm{ar}(\tau) = 0, \qquad \mathrm{ar}((\tilde{z})a\tilde{y}) = |\tilde{y}|, \qquad \mathrm{ar}(\tilde{x} = \tilde{y}) = 2|\tilde{x}|,$$

and we say that $\mathrm{ar}(\gamma)$ *is the* arity *of* γ.

Definition 12 (Labels of HD-automata). *The* labels of transitions of HD-automata for Fusion calculus *have the following type:*

$$\mathtt{L} \stackrel{\triangle}{=} \mathtt{M} \times \mathtt{Act} \times \prod_{n:\omega} (\mathtt{N}(n) \to \omega),$$

where $\mathtt{M}$ *is the type of matchings and is defined as pair of lists of natural numbers. Given* $\mu : \mathtt{M}$*,* μ_1 *and* μ_2 *denote the components of* μ *and we assume that they*

many brackets. The same notational abuse is adopted for the other sum types that will later appear in the paper.

have the same length[5]*. The enumeration type* $\mathtt{Act} \triangleq \mathtt{tau}, \mathtt{in}, \mathtt{out}, \mathtt{fuse}$ *is the type of* actions *(that we consider ordered by their position). Given* $\mu : \mathtt{M}$*, the set* $\mathrm{n}(\mu)$ *denotes the set of names occurring in* μ_1 *and* μ_2*.*

In Definition 12, elements of $\mathtt{L}$ (ranged over by l) are tuples $\langle \mu, \mathfrak{a}, n, \eta \rangle$ where,

- μ represents the conditions of the corresponding Fusion calculus label,
- $\mathfrak{a}$ is the action identifier,
- n is the number of object names of the action,
- η maps object names to 0 if they are free occurrences, otherwise η maps names to a *strictly* positive number (see Definition 16).

The order on $\mathtt{Act}$ is exploited for defining the order on $\mathtt{L}$ and *representative transitions* (see Definition 16). Let us however remark that it is important only to have total orders (no matter how they are chosen) and we could have also used a different order on $\mathtt{Act}$.

Since matchings are in biunivocal correspondence with elements in $\mathtt{M}$, we will sometime confound metavariables μ and M.

We assume that $\mathtt{L}$ is totally ordered (e.g., we can assume the lexicographic order on $\mathtt{L}$). Components of a label l are denoted by l_M, $l_{\mathfrak{a}}$, $\mathrm{ar}(l)$ and l_π, respectively. In the following, it will be useful to adopt the following notation:

> If f is a function that assume values on integers then $f^{>} = \{x \in \mathrm{dom}(f) \mid f(x) > 0\}$, $f^{<} = \{x \in \mathrm{dom}(f) \mid f(x) < 0\}$ and $f^{=} = \{x \in \mathrm{dom}(f) \mid f(x) = 0\}$.

Definition 13 (Free/bound names of labels). *We let* $\mathrm{fn}(l) = \mathrm{n}(l_M) \cup {l_\eta}^{=}$ *be the set of* free *names of* l*; accordingly, the set of* bound *names of* l *is* $\mathrm{bn}(l) = {l_\eta}^{>}$*. Finally, the* weight *of* l *(written* $|l|$*) is the cardinality of the set* ${l_\eta}^{=}$*.*

A transition system (or an automaton) can be described coalgebraically. Intuitively, the coalgebra corresponding to the transition system is a function mapping each state to the set of transitions out of that state. Transitions for nominal calculi also specify how names of labels and target state are related to names of the starting state; they can be defined by the type $\mathtt{hdt}$ of *HD-transitions* as

$$\mathtt{hdt} \triangleq \prod_{D:\mathtt{NS}} \sum_{q \in D:} \sum_{l:\mathtt{L}} \sum_{\pi:\mathtt{N}(|l|)\to\omega} \mathtt{N}(|q|_D) \to \mathtt{I}.$$

HD-transitions are built on named sets and can be seen as 4-tuples. It is worth to assign a name to HD-transitions' projections, in order to enhance readability. Let $D : \mathtt{NS}$ be a named set and $t : \mathtt{hdt}(D)$ be a HD-transition on D, then

$$\tau_t \triangleq \mathrm{I}(t),\ \ell_t \triangleq \mathrm{I}(\mathrm{II}(t)),\ \pi_t \triangleq \mathrm{I}(\mathrm{II}(\mathrm{II}(t))),\ \sigma_t \triangleq \mathrm{II}(\mathrm{II}(\mathrm{II}(t)));$$

[5] We prefer to maintain the presentations as simple as possible instead of formally defining $\mathtt{M}$ as $\prod_{m:\omega} \omega^m \times \omega^m$, where ω^m is the type of lists of natural numbers having length m.

a HD-transition t over a named set D represents a transition to state τ_t with label ℓ_t; moreover, t is equipped with functions π_t and σ_t that respectively map indexes of ℓ_t and names of τ_t to suitable names (of the source state of the transition). Both π_t and σ_t are necessary for establishing a relationship between indexes in labels and names local to states or between names of different states.

Remark 1. In the case of bound labels, we must relate the subject name of the HD-transition to a name in the source state, while the bound object names do not appear in π_t (as will be more clear later). The information about bound names is given in σ_t. In fact, if ℓ_t represents a bound action, σ_t maps the bound names (of t) to negative integers. In this respect, negative integers should not be considered as names, but as placeholders that signal that the corresponding names of the target state must be considered new. No name is mapped to negative integers by σ_t (${\sigma_t}^{<} = \emptyset$) when ℓ_t is a free label.

HD-transitions can be totally ordered using the lexicographic order $\sqsubseteq$, where $t \sqsubseteq t'$ if, and only if,

$$\tau_t \lesssim_D \tau_{t'} \wedge (\tau_t = \tau_{t'} \Rightarrow \ell_t \leq \ell_{t'} \wedge (\ell_t = \ell_{t'} \Rightarrow \pi_t \trianglelefteq \pi_{t'} \wedge (\pi_t = \pi_{t'} \Rightarrow \sigma_t \trianglelefteq \sigma_{t'})));$$

the intuition is that $\sqsubseteq$ exploits the orders of the underlying components for imposing an order relation on HD-transitions. Such order will become useful later to define the action of the functor over HD-automata for π-agents.

We call *bundle* the collection of HD-transitions out of a state. Bundles are described by type B below:

$$\mathtt{B} \stackrel{\triangle}{=} \sum_{D:\mathtt{NS}} \wp_{\mathrm{fin}}(\mathtt{hdt}(D)).$$

A bundle over a named set D is a pair whose first component is D and the second component is a set of HD-transitions on D. Analogously to what done for previous types, we assign names to the components of a bundle $\beta : \mathtt{B}$; the *support of* β is $\mathrm{I}(\beta)$ and is denoted by D_β, while the *step of* β, denoted by $\mathcal{S}_\beta$, is $\mathrm{II}(\beta)$.

4.1 Normalisation

This section reports the auxiliary operations on bundles that are used in the specification of the functor for coalgebras over the Fusion calculus.

We start by showing how bundles can be *casted* to named sets.

First, $\sqsubseteq$ is lifted to bundles (over the same support). Indeed, $\sqsubseteq$ induces the following order on bundles.

Definition 14 (Bundles' ordering). *Let $\beta_1, \beta_2 : \mathtt{B}$ be two bundles such that $D_{\beta_1} = D_{\beta_2}$ and let t_i be the minimal HD-transition in $\mathcal{S}_{\beta_i}$ (for $i = 1, 2$). We say that β_1 is* smaller *than β_2 (and write $\beta_1 \precsim \beta_2$) if, and only if,*

- *either $\mathcal{S}_{\beta_1}$ is empty,*

– *or* $t_1 \sqsubseteq t_2$ *and* $t_1 \neq t_2$,
– *or else* $t_1 = t_2$ *and* $\langle D_{\beta_1}, \mathcal{S}_{\beta_1} \setminus \{t_1\}\rangle \precsim \langle D_{\beta_1}, \mathcal{S}_{\beta_2} \setminus \{t_2\}\rangle$.

Basically, $\precsim$ corresponds to the lexicographic order on the second components of the bundles. (In the following, when no confusion arises, we use 'transition' as synonim of HD-transition.)

Second, the names of a bundle β (written $\{\!|\, \beta \,|\!\}$) are those names that "appear" in transitions of β:

$$\{\!|\,\_\,|\!\} \stackrel{\triangle}{=} \lambda\beta : \mathtt{B}. \bigcup_{t \in \mathcal{S}_\beta} \mathrm{fn}(\ell_t) \cup \mathrm{rng}(\pi_t) \cup \mathrm{rng}(\sigma_t \mid_{\sigma_t{}^>});$$

namely, $\{\!|\, \beta \,|\!\}$ contains those names that are either in $\mathrm{fn}(\ell_t)$ or in the ranges of π_t and σ_t of HD-transitions t in β. Bundles also play the role of states along the iterations of the algorithm (Section 5.1), hence, the names of a bundle obey the same conventions of names of states and, according to Remark 1, the negative integers should not be considered names of the bundle; for this reason they are removed when appearing in some transition, i.e. we consider ongly the range of σ_t restricted to $\sigma_t{}^>$. For any bundle β, the weight of β, $\lfloor \beta \rfloor$, is the cardinality of $\{\!|\, \beta \,|\!\}$.

Finally, let β be a bundle and ρ be a permutation of $\{\!|\, \beta \,|\!\}$, $\beta\rho$ denotes the bundle whose support is D_β and whose step is

$$\{\langle \tau_t, \rho(\ell_t), \pi_t; \rho, \sigma_t; \rho\rangle \mid \text{ is a permutation of } \{\!|\, \beta \,|\!\} \ \wedge\ t \in \mathcal{S}_\beta\},$$

where $\rho(\ell_t)$ is the label obtained by applying ρ to the names in the matching component of ℓ_t. The symmetry of β, $\mathit{Sym}(\beta)$, consists of all the bijections of $\{\!|\, \beta \,|\!\}$ that leave (the step of) β unchanged, namely

$$\mathit{Sym}(\beta) = \{\rho \mid \rho \text{ is a permutation of } \{\!|\, \beta \,|\!\} \ \wedge\ \ \mathcal{S}_\beta = \mathcal{S}_{\beta\rho}\}.$$

The most important operation on bundles is *normalisation*; it is necessary because (*i*) we must establish a canonical choice of the step component of a bundle among a number of different equivalent ways; (*ii*) more importantly, *redundant* transitions must be removed.

Redundant transitions occur when HD-automata are built from Fusion calculus agents. During this phase, it is not possible to decide which transitions are required, and which are redundant[6]. The solution to this problem consists of adding all the possible redundant transitions when HD-automata are built, and to exploit a reduction function (at every step of the partition algorithm) to remove those that are unnecessary. As will be more clear in Section 5.1, during the iterations of the minimisation algorithm the sets of redundant transitions of bundles decrease. When the iterative construction terminates, only those transitions that are really redundant will be removed from the bundles.

[6] In general, to decide redundancy of transitions is as difficult as deciding bisimilarity.

The normalisation of a bundle is done in different steps. First, the bundle is reduced by removing all the possibly redundant transitions using the function *non_red*:

$$non\_red \stackrel{\triangle}{=} \lambda\beta : \mathtt{B}.\langle D_\beta,\ \ \mathcal{S}_\beta \setminus \{t \in \mathcal{S}_\beta | dominated(\beta)(t)\}\rangle$$

where, *dominated* expresses the condition for redundancy and, for the symbolic semantics of Fusion calculus, is defined as follows:

$$\begin{aligned} dominated \stackrel{\triangle}{=}\ & \lambda\beta : \mathtt{B}.\lambda t : \mathtt{hdt}(D_\beta).\exists\langle \tau_t, l, \pi, \sigma\rangle \in \mathcal{S}_\beta. \\ & \ell_{t_M} \not\Longleftarrow\!\!\Rightarrow l_M \ \wedge\ \ell_{t_\mathfrak{a}} = l_\mathfrak{a} \ \wedge\ \mathrm{ar}(\ell_t) = \mathrm{ar}(l) \ \wedge\ \sigma;\sigma_M = \sigma_t \ \wedge\ \pi;\sigma_M = \pi_t. \end{aligned}$$

The intuition is that t is dominated in a bundle β when β contains a transition t' reaching the same target state as t (i.e., $\tau_t = \tau_{t'}$) with the same action (i.e., $\ell_{t\mathfrak{a}} = \ell_{t'\mathfrak{a}}$) with the same arities and such that

- t' has weaker conditions than t (i.e., $\ell_{tM} \not\Longleftarrow\!\!\Rightarrow \ell_{t'M}$) and,
- under $M = \ell_{tM}$, the exposed names are the same (i.e., $\sigma_{t'};\sigma_M = \sigma_t \ \wedge\ \pi_{t'};\sigma_M = \pi_t$).

Finally, the function to normalise bundles is defined as follows:

$$\begin{aligned} norm \ \stackrel{\triangle}{=}\ & \lambda\beta : \mathtt{B}. \\ & let\ \beta' = rem\_dom\ \beta \\ & in\ min_{\sqsubseteq}(\beta'\theta \mid \theta : \{\!|\beta'|\!\} \to \mathtt{N}(\lfloor\beta'\rfloor) \text{ is a bijective substitution}), \end{aligned}$$

where $rem\_dom = rem\_dom \stackrel{\triangle}{=} \lambda\beta : \mathtt{B}.\mathcal{S}_\beta \setminus \{t \in \mathcal{S}_\beta \mid dominated\ \beta\ t\}$ removes the dominated transitions from a bundle. Basically, *norm*, applied to a bundle β, filters those transitions that are dominated in β (and hence they are redundant). The remaining transitions are collected in a bundle β' and the substitution which makes β' minimal is applied to it (we denote such substitution θ_β; substitution θ_β enjoys the property $(rem\_dom\ \beta)\theta_\beta = norm\ \beta$).

We have the following results.

Lemma 7. *If $\beta, \beta' : \mathtt{B}$ are such that $\mathcal{S}_\beta \subseteq \mathcal{S}_{\beta'}$, and let $\beta_1 = norm\ \beta$ and $\beta'_1 = norm\ \beta'$ then $\mathcal{S}_{\beta_1};\theta^{-1}(\beta) \subseteq \mathcal{S}_{\beta'_1};\theta^{-1}(\beta')$.*

Proof. Trivial. □

Lemma 8. *For each $\beta : \mathtt{B}$, $\mathcal{S}_{norm\ \beta} \subseteq \mathcal{S}_\beta;\theta^{-1}(\beta)$.*

Proof. By construction, *norm* β is the bundle obtained by removing dominated HD-transitions from β and by renaming its names through $\theta^{-1}(\beta)$. □

Intuitively, Lemma 8 states that β "includes" the bundle resulting from its normalisation. Sometimes we write $\beta_1 \subseteq \beta_2$ instead of $\mathcal{S}_{\beta_1} \subseteq \mathcal{S}_{\beta_2}$.

$$
\begin{array}{l}
T_{NS} \stackrel{\text{def}}{=} \lambda A : \mathtt{NS}. \\
\qquad\quad \langle\ B = Q_A \times \wp_{\text{fin}}(\mathtt{hdt}(Q_A)), \\
\qquad\qquad \lesssim\ = \lambda(\beta, \beta') : B \times B. \\
\qquad\qquad\quad \mathit{if}\ \mathcal{S}_\beta = \emptyset \\
\qquad\qquad\qquad \mathit{then}\ \mathtt{tt} \\
\qquad\qquad\qquad \mathit{else} \\
\qquad\qquad\qquad\quad \mathit{let}\ t = \mathit{min}_{\sqsubseteq}(\mathcal{S}_\beta)\ \mathit{in} \\
\qquad\qquad\qquad\quad \mathit{let}\ t' = \mathit{min}_{\sqsubseteq}(\mathcal{S}_{\beta'})\ \mathit{in} \\
\qquad\qquad\qquad\qquad t \sqsubseteq t' \wedge (t = t' \Rightarrow \mathcal{S}_\beta \setminus \{t\} \lesssim \mathcal{S}_{\beta'} \setminus \{t'\}), \\
\qquad\qquad |\_| = \lambda\beta : B.\ \lfloor\beta\rfloor, \\
\qquad\qquad \lambda\beta : B.\mathit{Sym}(\beta) \\
\qquad\quad \rangle \\
T_{NF} \stackrel{\text{def}}{=} \lambda H : \mathtt{NF}. \\
\qquad\quad \mathit{let}\ S = T_{NS}(\mathrm{dom}(H)) \\
\qquad\quad \mathit{and}\ D = T_{NS}(\mathrm{cod}(H)) \\
\qquad\quad \mathit{and}\ h' = \lambda\beta \in S. \\
\qquad\qquad\qquad \langle \mathrm{cod}(H),\ \{\langle \mathrm{h}_H(q), l, \pi, \sigma'; \sigma\rangle\} \mid \langle q, l, \pi, \sigma\rangle \in \mathcal{S}_\beta \quad \wedge \quad \sigma' \in \Sigma_H(q)\}\rangle \\
\qquad\quad \mathit{and}\ h = \lambda\beta \in S.\mathit{norm}\ h'(\beta) \\
\qquad\quad \mathit{and}\ \Sigma = \lambda\beta \in S.\{\rho; \theta^{-1}(h'(\beta)) | \rho \in \mathit{Sym}(h(\beta))\} \\
\qquad\quad \mathit{in}\ \langle S, D, h, \Sigma\rangle
\end{array}
$$

Table 4. The functor on Fusion calculus

4.2 The functor for Fusion calculus

Table 4 borrows from [2] the definition of the functor for Fusion calculus in $\lambda^{\to,\Pi,\Sigma}$.

As usual [10], we represent T as a pair (T_{NS}, T_{NF}), where T_{NS} maps objects to objects, while T_{NF} maps morphisms to morphisms. When applied to a named set A, T_{NS} returns a named set whose components are described below:

- the underlying set is characterised by the type of bundles over A;
- the order relation is the order on bundles induced by the order of A (Definition 14);
- the weight function is the function that gives the number of names appearing in the HD-transitions of the bundle;

The named function resulting from applying T_{NF} to H is a function such that

- the domain and the codomain are obtained by applying T_{NS} to $\mathrm{dom}(H)$ and $\mathrm{cod}(H)$, respectively;
- the function $\mathrm{h}_{T_{NF}(H)}$ maps each bundle in the domain to a normalised bundle in the codomain;
- for each bundle β, the set of its name correspondences is obtained by composing symmetry $\mathit{Sym}(\mathrm{h}_{T_{NF}(H)}(\beta))$ with $\theta^{-1}(\beta)$ to undo the normalisation of names performed by computing *norm*.

5 From Fusion calculus to HD-automata

We now construct the HD-automaton corresponding to the canonical semantics of the Fusion calculus (Section 2). We mimic the definitions specified for π-calculus in [2]. This approach allows us to smoothly reuse the minimisation algorithm based on HD-automata (see Section 5.1).

We assume a total order on matchings (e.g., the lexicographic order).

Definition 15 (Representative matchings). *A sequence of matchings M is* representative *iff either $M = \mathbf{1}$ or $M = [x_1 = y_1] \dots [x_n = y_n] \neq \mathbf{1}$, and*

- $M \setminus [x_i = y_i] \not\Rightarrow [x_i = y_i]$, *for all* $i \in \{1, \dots, n\}$
- $x_i < y_i \wedge (i < j \Rightarrow x_i \leq x_j \wedge y_i \leq y_j \wedge y_i \neq x_j)$, *for all* $i, j \in \{1, \dots, n\}$.

Proposition 1. *If both M and N are representative matchings and $M \Longleftrightarrow N$ then $M = N$.*

Proof. By induction on the lenght of M. $\square$

Proposition 2. *For any matching M there exists a unique representative matching M' such that $M \Longleftrightarrow M'$.*

Proof. Given $x \in \mathrm{n}(M)$, let

$$\begin{array}{lcl} s_M(x) & = & \{y \in \mathrm{n}(M) \mid y > x \wedge [x = y] \text{ in } M\} \\ U & = & \{x \in \mathrm{n}(M) \mid \forall y \in \mathrm{n}(M).x \notin s_M(y)\}. \end{array}$$

By definition, set $s_M(x)$ contains all names, different from x, directly equated to x in M and set U is the set of all those names not appearing in $s_M(y)$ for any $y \in M$. Notice that x is in U only if, for any $y < x$, $x \notin s_M(y)$. For any $x \in U$, we define the set of names *equal* to x in M as

$$e_M(x) = s_M(x) \cup \bigcup_{y \in s_M(x)} s_M(y).$$

Let us remark that e_M is well define because $y \in s_M(x) \Rightarrow x \notin s_M(y)$. Moreover, $e_M(x)$ contains all the names in M that can be provably be equated to x. We define

$$M' = \begin{cases} \mathbf{1}, & if\ \forall x \in U.s_M(x) = \emptyset \\ x_1[M] \cdots x_n[M], & if\ U = \{x_1, \dots, x_n\} \wedge 1 \leq i < j \leq n \Rightarrow x_i < x_j \end{cases},$$

where, letting $s_M(x_i) = \{y_1, \dots, y_m\}$ and $h < k \Rightarrow y_h < y_k$, $x_i[M] = [x_i = y_1] \cdots [x_i = y_m]$. Then M' (by construction) is representative and $M \Longleftrightarrow M'$. Assume that $M' = \mathbf{1}$; then, for any $x \in \mathrm{n}(M)$, $s_M(x) = \emptyset$ (in fact, if $y \in s_M(x)$ then $M' \neq \mathbf{1}$, that contradicts the hypothesis $M' = \mathbf{1}$). Hence, for any x, $[x = x]$ is the only possible match in M that involves x, namely $M \Longleftrightarrow M' = \mathbf{1}$.
Assume that $M' = x_1[M] \cdots x_n[M] \neq \mathbf{1}$. We first prove that, for all $i \in \{1, \dots, n\}$ and $y \in x_i[M]$, $M \Rightarrow [x_i = y]$; indeed, either $y \in s_M(x_i)$ i.e., one of

$[x_i = y]$ or $[y = x_i]$ appear in M, or else there are $[x_i = y'_1], [y'_1 = y'_2], \ldots, [y'_h = y]$ in M (by construction).

We prove that $M' \Rightarrow M$. Let $[x = y]$ be a matching in M; without loss of generality, we can assume that $x < y$. By definition of e_M, $y \in e_M(\bar{x})$ for $\bar{x} = min\{u \in \mathrm{n}(M) \mid [u = x'_1], [x'_1 = x'_2], \ldots, [x'_k = x]\}$; moreover, $\bar{x} \in U$. Thence, $M' \Rightarrow \bar{x}[M] \Rightarrow [x = y]$.

Unicity of M' is implied by Proposition 1. □

Notice that the proof of Proposition 2 provides a constructive algorithm for determining the representative matchings from non representative ones. Before giving the translation, we introduce a slightly different interpretation of Fusion calculus canonical semantics. Namely, we consider the LTS specified in Table 1 where the operation ' $\star$ ' is interpreted as follows:

$$M' = M \star N \quad \text{such that} \quad \begin{array}{l} -\ M' \text{ is representative and} \\ -\ M' \Longleftrightarrow MN. \end{array} \tag{1}$$

Proposition 1 and 2 ensure that ' $\star$ ' is well defined (and commutative). Let us remark that the second interpretation of ' $\star$ ' yields an equivalent symbolic semantics for Fusion calculus. For the sake of clarity, let $\longmapsto_1$ and $\longmapsto_2$ respectively denote the symbolic transition relation under the interpretation of ' $\star$ ' as juxtaposition and as in (1). A strict connection between $\longmapsto_1$ and $\longmapsto_2$ exists, indeed, it is evident that

$$P \xmapsto{M,\gamma}_1 Q \iff P \xmapsto{M',\gamma}_2 Q, \tag{2}$$

where, M' is the representative matching such that $M' \Longleftrightarrow M$. Equivalence 2 is trivially implied by the fact that

1. each proof for a derivation in $\longmapsto_1$ is a proof for a derivation in $\longmapsto_2$;
2. $\sigma_M = \sigma_{M'}$ (since $M' \Longleftrightarrow M$);
3. Lemma 1 holds for $\longmapsto_2$.

We also conclude that the symbolic bisimultions induced by $\longmapsto_1$ and $\longmapsto_2$ coincide.

Basically, $\longmapsto_2$ is obtained by factorising all those transitions of $\longmapsto_1$ that differ only for the syntactic representation of their matching conditions and by considering all of them as a single $\longmapsto_2$-transition whose matching is representative.

The new interpretation of ' $\star$ ' allows us to further refine the canonical semantics of Fusion calculus. Indeed, we can define *representative transitions*.

Definition 16 (Representative transitions). *A transition $P \xmapsto{M,\gamma} Q$ is* representative *iff, M is a representative matching and one of the following condition holds:*

- $\gamma = \tau$

– $\gamma = a\tilde{x}$
– $\gamma = \{\tilde{x} = \tilde{y}\}$
– $\gamma = (y_1, \ldots, y_n)a\tilde{x}$ *and* $y_i = min(\mathcal{N} \setminus \mathrm{fn}(P) \setminus \{y_1, \ldots, y_{i-1}\})$, *for all* $i \in \{1, \ldots, n\}$.

Intuitively, a representative transition is exploited to single out a transition labelled γ in a canonical way from a bunch of transitions such that the conditions that enable γ to be fired are "minimal". We use only representative transitions in HD-automata that correspond to Fusion calculus agents; the following lemma ensures that these transitions are enough for characterising agents' behaviour.

Lemma 9 (Lemma 7.6 of [8]). *If* $P \xmapsto{M,\gamma}_1 Q$ *is a transition from* P*, then there is a representative transition* $P \xmapsto{M',\gamma'}_2 Q'$ *such that*

– $\sigma_M = \sigma_{M'}$, *and*
– *there is a bijection* $\rho : \mathrm{bn}(\gamma) \to \mathrm{bn}(\gamma')$ *such that* $\gamma = \rho(\gamma')$ *and* $Q' = Q\rho$,

where $\rho(\gamma)$ *is the synctactic substitutions of names in* γ *according to* ρ.

Proof. By the property of $\longmapsto_2$ previously discussed, if $P \xmapsto{M,\gamma}_1 Q$ then $P \xmapsto{M',\gamma}_2 Q$ where M' is representative and $M' \Longleftrightarrow M$. If γ is not a bound action the proof is finished, otherwise, let γ be $(\tilde{z})a\tilde{x}$; then we can α-rename all the occurrences of names $\tilde{z}$ as specified in the last item of Definition 16. Hence, by the S-CONG, we have, $P \equiv\xmapsto{M',\gamma'}_2\equiv Q\rho$ where $\rho = [{}^{y_i}/_{z_i};\ i = 1, \ldots, |\tilde{z}|\ ]$ and $\gamma' = (\tilde{y})a\tilde{x}\rho$. $\square$

By Lemma 9 we can safely confound $\longmapsto_1$ and $\longmapsto_2$ and in the following we simply write $\longmapsto$. In some sense, representative transitions constitute a canonical form of symbolic transitions of an agent such that those transitions differ only on the choice of bound names.

Definition 17 (Representative symbolic hyperbisimulation). *A binary symmetric process relation* $\mathcal{S}$ *is a* representative symbolic hyperbisimulation *if* $(P, Q) \in \mathcal{S}$ *implies:*
If $P \xmapsto{M,\gamma} P'$ *with* $\mathrm{bn}(\gamma) \cap \mathrm{fn}(Q) = \emptyset$ *then there exists a representative transition* $Q \xmapsto{N,\gamma'} Q'$ *such that*

– $M \Rightarrow N$,
– $\gamma = \gamma'\sigma_M$, *(note* $\gamma = \gamma\sigma_M$*)*
– *and* $(P', Q'\sigma_M) \in \mathcal{S}$ *(note* $P' = P'\sigma_M$*).*

We write $P \cong Q$ *whenever* (P, Q) *belongs to some representative symbolic hyperbisimulation.*

Notice that the previous definition differs from Definition 7 only because the transition that Q uses to bisimulates the one from P must be representative. The following lemma trivially holds.

Lemma 10. *Any representative symbolic hyperbisimulation is a symbolic hyperbisimulation.*

Moreover, the converse holds:

Proposition 3. *For any two Fusion calculus agents P and Q, $P \simeq Q \iff P \cong Q$*

Proof. The proof easily follows from Proposition 9 and Lemma 10.

Proposition 3 directly implies that it suffices to consider representative transitions for characterising the observational semantics of agents; indeed,

Corollary 1. *For any Fusion calculus agent $P \simeq P \iff P \cong P$.*

We assume to have a function norm(_) that, given a Fusion calculus agent P, returns a pair $\mathrm{norm}(P) = \langle \bar{P}, \theta_P \rangle$, where $\bar{P}$ is the representative element of the class of those agents that differ from P for a bijective substitution θ_P, such that $\bar{P} = P\theta_P$ and $\overline{P\rho} = \bar{P}$, for any bijective substitution ρ.

Let P be a Fusion calculus agent; $\langle \bar{P}, \theta_P \rangle = \mathrm{norm}(P)$ denotes the pair made of $\bar{P}$ (the canonical representative for P) and the injective substitution θ_P mapping names of P into names of $\bar{P}$ (i.e. $\mathrm{norm}(P) = \langle \bar{P}, \theta_P \rangle$). Moreover, $K[P]$ denotes the HD-automaton associated to P. The coalgebraic specification of the corresponding HD-automaton of P can be easily obtained as a named function $K[P]$ with $\mathrm{dom}(K[P]) = D[P]$ and $\mathrm{cod}(K[P]) = T_{NS}(D[P])$.

We first determine $Q_{D[P]}$, the set of the states, as follows:

$$Q_{D[P]} \stackrel{\mathrm{def}}{=} \bigcup_{P' \in S[P]} Q_{D[P']} \cup S[P],$$

where $S[P] \stackrel{\mathrm{def}}{=} \{\bar{P}\} \cup \{\overline{P'} \mid P \xmapsto{M,\gamma} P'$ is a representative transition $\wedge\ \mathrm{norm}(P') = \langle \overline{P'}, \theta_{P'} \rangle\}$. Intuitively, $Q_{D[P]}$ is the set of (the canonical representative) processes that can be reached from P (through representative transitions). It is trivial to equip $Q_{D[P]}$ with a named set structure, indeed

- the order $\lesssim_{D[P]}$ on $Q_{D[P]}$ is the lexicographic order on processes;
- for any $q \in Q_{D[P]}$, the weight function $|q|_{D[P]}$ yields the cardinality of $\mathrm{fn}(q)$;
- for any $q \in Q_{D[P]}$, the group component $\mathcal{G}_{D[P]}(q)$ is the identity on $\mathrm{fn}(q)$ or is $\emptyset$, depending on $\mathrm{fn}(q) \neq \emptyset$.

Function $\mathrm{h}_{K[P]}$ associates, to each state, the bundle of its outgoing transitions and is defined as

$$\mathrm{h}_{K[P]} = \lambda q.norm\ \langle Q_{D[P]}, \{t_{M,\gamma} \mid q \xmapsto{M,\gamma} q' \text{ is a representative transition }\}\rangle$$

where, if $\langle \overline{q'}, \theta_{q'} \rangle = \mathrm{norm}(q')$, HD-transition $t_{M,\gamma}$ is a tuple $\langle \overline{q'}, l, \pi, \sigma \rangle$ where l, π and σ depend on γ. We first consider the case where γ is a communication action, namely, $\gamma = (\tilde{z})a\tilde{y}$. Let us recall that $l : \mathtt{L}$ (see Definition 12), hence, the label of $t_{M,\gamma}$ is $l = \langle M, \mathfrak{a}, \mathrm{ar}(\gamma), \eta \rangle$, where the label identifier $\mathfrak{a}$ is `in` or `out` depending on whether a is an input or an output action, respectively. The last two components of l tell which are the free and bound names of $t_{M,\gamma}$. Indeed, $\eta : \{1, \ldots, \mathrm{ar}(\gamma)\} \to \omega$ and is defined as

$$\eta(i) = \begin{cases} 0, & 1 \leq i \leq \mathrm{ar}(\gamma), y_i \notin \tilde{z} \\ -j, & 1 \leq i \leq \mathrm{ar}(\gamma), 1 \leq j \leq |\tilde{z}|, z_j = y_i \end{cases} .$$

Function π of $t_{M,\gamma}$ establishes the correspondence among the free occurrences of names in γ and the names of the source state:

$$\pi : \begin{cases} 1 \mapsto \mathrm{sbj}(\gamma) \\ j+1 \mapsto y_j, \ j \in \eta^{=}\} \end{cases} .$$

Analogously, σ traces the correspondence among names of $\overline{q'}$ and names of the source state:

$$\sigma = \theta_{q'}^{-1}[\theta_{q'}(y_i) \mapsto \eta(i) \mid i \in \eta^{<}],$$

which amounts to say that the names not restricted by γ are preserved in $\overline{q'}$, while, restricted names are mapped in the negative indexes determined by η.

If $\gamma = \{\tilde{x} = \tilde{y}\}$ then $l = \langle M, \texttt{fuse}, \mathrm{ar}(\gamma), \eta\rangle$, where for any $i \in \{1, \ldots \mathrm{ar}(\gamma)\}$, $\eta(i) = 0$ (since all occurrences of names in γ are free). Function π is defined as follows:

$$\pi = \begin{cases} 2i \mapsto y_i \\ 2i-1 \mapsto x_i \end{cases}, \qquad 1 \leq i \leq \frac{\mathrm{ar}(\gamma)}{2}$$

while σ simply is $\theta_{q'}^{-1}$.

Accordingly, $t_{M,\tau} = \langle \overline{q'}, l, \emptyset, \theta_{q'}^{-1}\rangle$ where $l = \langle M, \texttt{tau}, 0, \emptyset\rangle$.

We can state the following correspondence theorem.

Theorem 3. *Let $P \xmapsto{M,\gamma} Q$ be a representative transition of the Fusion calculus agent P and let* $\mathrm{norm}(P) = (\bar{P}, \theta)$. *Then exists $t_{M',\gamma'} \in \mathcal{S}_{\mathrm{h}_{K[P]}(\bar{P})}$ such that*

- *$M' \Rightarrow M\theta$,*
- *$\gamma' = \gamma\theta; \sigma_{M'}$*
- *$\tau_{t_{M',\gamma'}} = \bar{Q}'$, where* $\mathrm{norm}(Q\sigma_{M'}) = (\bar{Q}', \theta')$.

Proof. By construction and case analysis.

Finally, we define $\Sigma_{K[P]}(q) = \{\rho; \theta^{-1}(\beta_q) | \rho \in \mathcal{G}_{\mathrm{cod}(K[P])}(\mathrm{h}_{K[P]}(q))\}$, for any state q.

The HD-automaton obtained by this definition is a T-coalgebra by construction and it is a valid HD-automaton.

The construction above may yield infinite HD-automata. However, there are interesting classes of Fusion calculus agents that generate finite HD-automata: This is the case of *finitary* agents. The *degree of parallelism* $\deg(P)$ of a Fusion calculus agent P is defined as follows:

$$\begin{array}{rr} \deg(\mathbf{0}) = 0 & \deg(\alpha \,.\, P) = 1 \\ \deg((x)P) = \deg(P) & \deg(P \mid Q) = \deg(P) + \deg(Q) \\ \deg([x=y]P) = \deg(P) & \deg(P + Q) = \max(\deg(P), \deg(Q)) \end{array}$$

$$\deg(A\langle y_1, \ldots, y_n\rangle) = \deg(P[{}^{y_1,\ldots,y_n}/{}_{x_1,\ldots,x_n}]), \quad \text{if } A(x_1, \ldots, x_n) \stackrel{\mathrm{def}}{=} P$$

Agent P is *finitary* if $\max\{\deg(P') \mid P \xmapsto{M_1,\gamma_1} \cdots \xmapsto{M_i,\gamma_i} P'\} < \infty$. In [6, 8] the following result has been proved:

Theorem 4 (Theorem 47 of [6]). *Let P be a finitary agent. Then the HD-automaton $K[P]$ is finite.*

5.1 The minimisation algorithm in $\boldsymbol{\lambda}^{\rightarrow,\Pi,\Sigma}$

This section specifies the minimisation algorithm for HD-automata and proves that the partition refinement algorithm converges on finite HD-automata.

A transition system is a HD-automaton and can be coalgebraically specified as a named function K such that $\mathrm{cod}(K) = T_{NS}(\mathrm{dom}(K))$. We now point out the minimisation algorithm. Given a T-coalgebra $K : \mathtt{NF}$ with named set A as source, we let $\bot : \mathtt{NS} = \langle unit, \lambda x, y : unit.\mathtt{tt}, \lambda x : unit.0, \emptyset\rangle$ be the *empty* named set (i.e. the named set on the vacuum type *unit* having () as its unique element). The minimisation algorithm is defined by the equations below.

$$\text{Initial approximation: } H_0 \stackrel{\text{def}}{=} \langle A,\ \bot,\ \lambda q : A.(),\ \lambda q : A.\emptyset\rangle \tag{3}$$

$$\text{Iterative construction: } H_{i+1} \stackrel{\text{def}}{=} K; T_{NF}(H_i). \tag{4}$$

In words, all the states of automaton K are initially considered equivalent, indeed, $\ker H_0$ gives rise to a single equivalence class containing the whole $\mathrm{dom}(K)$. At the generic $(i + 1)$-th iteration, the image through T_{NF} of the i-th iteration is composed with K as prescribed in (4). The algorithm builds the minimal realisation $\bar{H}$ of (finite) HD-automata by constructing (an approximation of) the final coalgebra morphism. The kernel of $\bar{H}$ yields the equivalence classes where equivalent states are grouped. The active names of each state q are those in the ranges of $\Sigma_{\bar{H}}(q)$.

The proof of the convergence of the algorithm is based on the fact that T_{NF} is a monotonic functor over a set whose chains are finite. Hence we first show that T is monotonic.

Proposition 4 (Monotonicity of T). *T_{NF} is monotonic.*

Monotonicity is preserved by composition of named functions:

Proposition 5 (Composition and $\preceq$). *Let H_1 and H_2 be two named functions such that $H_1 \preceq H_2$. For any $K : \mathtt{NF}$ if $\mathrm{cod}(K) = \mathrm{dom}(H_1) = \mathrm{dom}(H_2)$ then $K; H_1 \preceq K; H_2$.*

Finally, we can prove the convergence of the iterative algorithm:

Theorem 5 (Convergence). *The iterative algorithm described by (3) and (4) is convergent on finite state automata.*

We omit the proofs of the above results because they are basically the same of [2]. Proposition 4 and Theorem 5 rely on the functoriality of T_{NF}, which in turn derive from Lemmas 7 and 8.

A tight relationship can be established between Fusion calculus bisimilarity and the outcome of the minimisation algorithm. On the one hand, if P and Q are two bisimilar Fusion calculus agents then the same minimal realisation for $K[\bar{P}]$ and $K[\bar{Q}]$ is achieved by the minimisation algorithm (if $K[\bar{P}]$ and $K[\bar{Q}]$ are finite). Namely, the morphisms $\bar{H}_P$ and $\bar{H}_Q$ are equivalent, i.e. $\ker \bar{H}_P = \ker \bar{H}_Q$ which implies that corresponding classes of $\ker \bar{H}_P$ and $\ker \bar{H}_Q$ have the same number of active names and the same symmetries. It could be the case that the name correspondences in $\Sigma_{\bar{H}_P}(\bar{P})$ and $\Sigma_{\bar{H}_Q}(\bar{Q})$ are different since $\bar{P}$ and $\bar{Q}$ can have different numbers of inactive names, however $\Sigma_{\bar{H}_P}(\bar{P}); \theta_P = \Sigma_{\bar{H}_Q}(\bar{Q}); \theta_Q$ (where $\mathrm{norm}(P) = \langle \bar{P}, \theta_P \rangle$ and $\mathrm{norm}(Q) = \langle \bar{Q}, \theta_Q \rangle$). Indeed, P and Q must have the same set of active names otherwise they could not be bisimilar. On the other hand, if $K[P]$ and $K[Q]$ are finite and they both have $\bar{H}$ as their minimal realisation, then P and Q are bisimilar provided that $\Sigma_{\bar{H}_P}(\bar{P}); \theta_P = \Sigma_{\bar{H}_Q}(\bar{Q}); \theta_Q$. In fact, names in $K[P]$ and $K[Q]$ are local to the automata, hence, in order to check bisimilarity of the corresponding processes, the mappings of local names to global names must be the same. Equality $\Sigma_{\bar{H}_P}(\bar{P}); \theta_P = \Sigma_{\bar{H}_Q}(\bar{Q}); \theta_Q$ expresses this condition because it requires that correspondences between the names of P of the names and its canonical representative (i.e. $\bar{P}$) are the same of those of Q and the canonical representative of Q (i.e. $\bar{Q}$).

5.2 Applying the Algorithm

We apply the mapping in the previous section and the functor to a simple example. Let us consider the following Fusion calculus agent and its (canonical) transition system:

$$P = [x = y]\overline{u}\,u.\mathbf{0} + \overline{u}\,u.(x)\overline{x}\,u.\mathbf{0}$$
$$Q = (x)\overline{x}\,u.\mathbf{0}$$

$$\begin{tikzcd} & S = \mathbf{0} \\ P \arrow[ur, "{[x=y], \overline{u}\,u}"] \arrow[dr, "{\emptyset, \overline{u}\,u}"'] & \\ & Q \end{tikzcd}$$

(notice that Q is bisimilar to S).

First, we built the HD-automaton for P, $K[P]$. In orderd to construct $K[P]$ we must normalise the states of the transition system:

	$\bar{\_}$	$\mathrm{fn}(\_)$	$\theta(\_)$	$\mathcal{G}_{K[P]}\_$
P	$[a = b]\overline{c}\,c.\mathbf{0} + \overline{c}\,c.(d)\overline{d}\,c.\mathbf{0}$	$\{a, b, c\}$	$\begin{cases} x \mapsto a \\ y \mapsto b \\ u \mapsto c \end{cases}$	$\{\mathrm{id}_{a,b,c}\}$
Q	$(b)\overline{b}\,a.\mathbf{0}$	$\{a\}$	$\{u \mapsto a\}$	$\{\mathrm{id}_a\}$
S	$\mathbf{0}$	$\{\}$	$\{\}$	$\{\emptyset\}$

The HD-automaton for P is described by the named function $K[P]$ which is a coalgebra over the named set $\mathcal{D}_{\mathcal{P}}$ of the states $\bar{P}$, $\bar{Q}$ and $\bar{S}$. The codomain of $K[P]$ is the named set of bundles over $\mathcal{D}_{\mathcal{P}}$; below we list the remaining component of $K[P]$:

$$\mathrm{h}_{K[P]} : \begin{cases} \bar{P} \mapsto \left\{ \begin{array}{c} \langle \bar{S}, \mathtt{out}_{[a=b]} c\ c, \emptyset \rangle, \\ \langle \bar{Q}, \mathtt{out}_{\emptyset} c\ c, \{a \mapsto c\} \rangle \end{array} \right\} \\ \bar{Q} \mapsto \{\} \\ \bar{S} \mapsto \{\} \end{cases} \qquad \Sigma_{K[P]} : \begin{cases} \bar{P} \mapsto \{\mathrm{id}_{a,b,c}\} \\ \bar{Q} \mapsto \{\emptyset\} \\ \bar{S} \mapsto \{\emptyset\} \end{cases}$$

notice that the normalisation of the bundle of $\bar{P}$ does not change it, hence, $\Sigma_{K[P]}(\bar{P})$ contains only the identity. Moreover, groups of all bundles are trivially computed.

We can now apply the algorithm.

$\boxed{H_0}$ is the function that maps $\bar{P}$, $\bar{Q}$, $\bar{S}$, in $\perp$ and assigns to all of them the empty set of name correspondence.

$\boxed{H_1}$ is computed by $K[P]; T_2(H_0)$ hence, we have

$$\mathrm{h}_{K[P]} : \begin{cases} \bar{P} = \mathrm{h}_{T(H_0)}(\mathrm{h}_{K[P]}(\bar{P})) \mapsto \{\langle \perp, \mathtt{out}_{\emptyset}, a\ a, \emptyset \rangle\} \\ \bar{Q} = \mathrm{h}_{T(H_0)}(\mathrm{h}_{K[P]}(\bar{Q})) \mapsto \emptyset \\ \bar{S} = \mathrm{h}_{T(H_0)}(\mathrm{h}_{K[P]}(\bar{S})) \mapsto \emptyset \end{cases} \qquad \Sigma_{K[P]} : \begin{cases} \bar{P} \mapsto \{a \mapsto c\} \\ \bar{Q} \mapsto \{\emptyset\} \\ \bar{S} \mapsto \{\emptyset\} \end{cases}$$

$\boxed{H_2}$ is computed by $K[P]; T_2(H_1)$ hence, we have

$$\mathrm{h}_{K[P]} : \begin{cases} \bar{P} = \mathrm{h}_{T(H_1)}(\mathrm{h}_{K[P]}(\bar{P})) \mapsto \{\langle \mathrm{h}_{H_1}(\bar{P}), \mathtt{out}_{\emptyset}, a\ a, \emptyset \rangle\} \\ \bar{Q} = \mathrm{h}_{T(H_1)}(\mathrm{h}_{K[P]}(\bar{Q})) \mapsto \emptyset \\ \bar{S} = \mathrm{h}_{T(H_1)}(\mathrm{h}_{K[P]}(\bar{S})) \mapsto \emptyset \end{cases} \qquad \Sigma_{K[P]} : \begin{cases} \bar{P} \mapsto \{a \mapsto c\} \\ \bar{Q} \mapsto \{\emptyset\} \\ \bar{S} \mapsto \{\emptyset\} \end{cases}$$

and the algorithm terminates (since H_1 and H_2 yields isomorphic automata).

Let us remark that in the minimal automata:

- the free name of $\bar{Q}$ does not appear, since it is not used;
- therefore, $\bar{S}$ and $\bar{Q}$ has been identified;
- hence, the transition to $\bar{S}$ becomes redundant, thence it is removed in the final automaton.

References

1. Gianluigi Ferrari, Ugo Montanari, and Marco Pistore. Minimizing transition systems for name passing calculi: A co-algebraic formulation. In Mogens Nielsen and Uffe Engberg, editors, *FOSSACS 2002*, volume LNCS 2303, pages 129–143. Springer Verlag, 2002.
2. Gianluigi Ferrari, Ugo Montanari, and Emilio Tuosto. Coalgebraic minimisation of hd-automata for the π-calculus in a polymorphic λ-calculus. Submitted for publication, August 2003.

3. Cosimo Laneve and Björn Victor. Solos in concert. In Jiří Wiederman, Peter van Emde Boas, and Mogens Nielsen, editors, *Proceedings of ICALP '99*, volume 1644 of *LNCS*, pages 513–523. Springer, July 1999.
4. John C. Mitchell. *Foundations for Programming Languages.* MIT press, 1996.
5. Ugo Montanari and Marco Pistore. History dependent automata. Technical report, Computer Science Department, Università di Pisa, 1998. TR-11-98.
6. Ugo Montanari and Marco Pistore. π-calculus, structured coalgebras, and minimal HD-automata. In Mogens Nielsen and Branislav Roman, editors, *MFCS: Symposium on Mathematical Foundations of Computer Science*, volume 1983 of *LNCS*. Springer Verlag, 2000. An extended version will be published on Theoretical Computer Science.
7. Joachim Parrow and Björn Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proceedings of LICS '98*, pages 176–185. IEEE, Computer Society Press, July 1998.
8. Marco Pistore. *History dependent automata.* PhD thesis, Computer Science Department, Università di Pisa, 1999.
9. Marco Pistore and Davide Sangiorgi. A partition refinement algorithm for the π-calculus. In Rajeev Alur, editor, *Proceedings of CAV '96*, volume 1102 of *LNCS*, pages 38–49, 1996. Full version to appear in *Journal of Information and Computation.*
10. David E. Rydeheard and Rod M. Burstall. *Computational Category Theory.* Prentice-Hall, New York, 1988.
11. Davide Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Informatica*, 33:69–97, 1996. Earlier version published as Report ECS-LFCS-93-270, University of Edinburgh. An extended abstract appeared in the *Proceedings of CONCUR '93*, LNCS 715.