HD-automata for open bisimulation

Marino Miculan[1], Emilio Tuosto[2], and Kidane Yemane[3]

[1] Dept. of Mathematics and Informatics, University of Udine, Italy.
miculan@dimi.uniud.it
[2] Dept. of Computer Science, University of Leicester, UK. et52@mcs.le.ac.uk
[3] Dept. of Information Technology, Uppsala University, Sweden. kyemane@it.uu.se

Abstract. HD-automata are a syntax-independent operational model introduced for dealing with history-dependent formalisms. This kind of enriched automata, where states, transitions, and labels are equipped with *names* and *symmetries*, have been successfully applied for modelling early and late bisimulation in π-calculus and hyperbisimulation in Fusion calculus. However, current HD-automata are not adequate for modelling open bisimulation, because in HD-automata two names cannot be unified, while open bisimulation is closed under all possible name substitution respecting name distinctions.
In this paper we tackle the problem by integrating in the definition of *named sets*, the basic building blocks of HD-automata, a notion of *distinction:* names can coalesce if the distinction allows to. Then, we use HD-automata over named sets with distinctions for modelling the open bisimulation of π-calculus. Finally, we discuss the relationship between named sets with distinctions and their HD-automata, with the categorical counterparts based on presheaf categories.

1 Introduction

Over the last years several kinds of operational models of name passing calculi have been suggested; to mention some, coalgebras over presheaf category [8], indexed labeled transition systems [3, 7], and HD-automata [16, 6]. HD-automata are notable for their simplicity and very suited for implementation in automated verification of properties of name passing calculi.

The main aim of HD-automata is to give finite representations of otherwise infinite LTSs; similarly to ordinary automata, they consist of states and labeled transitions, however, states and transitions of HD-automata are equipped with names which are no longer treated as syntactic components of labels, but become an explicit part of the operational model. This allows to model the typical mechanisms of name passing calculi, such as creation and extrusion of names. Moreover, it allows for compact representation of agent behaviour by collapsing states that differ only for renaming of local names. In [16] HD-automata has been successfully used to model early and late semantics of π-calculus and in [6] hyperbisimulation of fusion calculi.

In spite of these successes, HD-automata are not easily applicable to model *open bisimulation* of π-calculus [22]. Open bisimulation is of great importance in

the theory of concurrency, because it is a congruence (differently from early and late bisimulations) and, although its definition may seem complicated, it has an "efficient" characterisation exploited in automated tools [22, 24].

The complexity comes from the fact that open bisimulation is closed under all possible substitutions of names, respecting the "freshness" of names coming from scope extrusions; this means that substitutions can *unify* names which are not defined as distinct names. Thus, the intended meaning of names in open bisimulation is rather different from that of late and early bisimulations (and ultimately of standard HD-automata): in late/early, different names are considered as different constants, and hence can never be unified; on the other hand, in open names are rather "abstract variables (ranging over constants) subject to separateness conditions". Given these different interpretations of names, it is not surprising that current HD-automata, which have been defined with late and early bisimulations in mind, cannot cope with open bisimulation.

In this paper, we propose a variant of HD-automata for handling open bisimulation, and such that the previous HD-automata can be seen as special case. The key point of our construction is that the definition of *named sets*, the basic building blocks of HD-automata, must be changed to account for the different meaning of names in open bisimulation (Section 3). More precisely, in our definition an element of a named sets is equipped with a set of names, a *distinction relation* over these names, and a group of permutations *respecting the distinction*. Also functions between named sets must preserve distinctions, but it is important to notice that names can coalesce *if the distinction allows to*. Traditional named sets can be seen as a particular case of these named sets with distinctions, just by considering *complete* distinctions, i.e., those declaring that names can never be unified.

Then, we develop a theory of HD-automata over named sets with distinctions, which we apply in Section 4 for modelling the open bisimulation of π-calculus. In this case, we present also a normalization aglgorithm which allows to verify whether two finitary processes are open bisimilar.

Finally, in Section 5 we discuss the relationship between named sets with distinctions and their HD-automata, with the categorical counterparts based on presheaf categories. Conclusions and final remarks are in Section 6.

2 The π-calculus

In this section we recall the syntax and semantics of the π-calculus and the definition of open bisimulation. As this is standard material, we refer to standard literature for most of the technical details, e.g. [14, 13, 23].

We assume a coutable set of names $\mathcal{N}$ ranged over by x, y and z. Set $\mathcal{N}$ represents the set of communication channels and the values sent and received; later, it will also be used for representing names of elements of *named sets* and, therefore, local names of states of HD-automata.

Definition 1 (The π-calculus). *The set of π-calculus processes, ranged over by P, Q, R are defined as follows*

$$P ::= \mathbf{0} \mid \alpha\,.\,Q \mid Q + R \mid Q \mid R \mid (\boldsymbol{\nu} y)Q \mid [x = y]Q \mid A\langle x_1, \ldots, x_r\rangle$$
$$\alpha ::= x(y) \mid \overline{x}y \mid \tau$$

Input prefix and restriction bind y in Q: free and bound names are as usual and are denoted by $\mathrm{fn}(\_)$ *and* $\mathrm{bn}(\_)$*, respectively,* $\mathrm{n}(\_) = \mathrm{fn}(\_) \cup \mathrm{bn}(\_)$.

Each process identifier A has rank r and a unique definition $A(y_1, \ldots, y_r) \stackrel{\text{def}}{=} Q$ (with y_i all distinct and $\mathrm{fn}(Q) \subseteq \{y_1 \ldots y_r\}$); moreover, we assume that each process identifier in Q is in the scope of a prefix (guarded recursion).

Processes of π-calculus can perform input, output of silent actions. Process $\mathbf{0}$ stands for the deadlocked process; $\alpha\,.\,Q$ can perform the action α and then continue as Q; $Q + R$ is the nondeterministic choice between Q and R; $Q \mid R$ is the process Q and R running in parallel; $(\boldsymbol{\nu} x)Q$ is the process Q where x is *restricted*; $[x = y]Q$ is the process Q if the names x and y are equal and otherwise is $\mathbf{0}$. Finally, $A\langle x_1, \ldots, x_n\rangle$ is the (recursive) call of processes.

Prefixes generate the observable behaviour that processes exhibit as they evolve. Observations vary depending upon the bisimulation one wants to study. For open bisimulation, the possible observables are *i)* $x(y)$ representing the input along x of a name; *ii)* $\overline{x}y$ representing the output of a name y on channel x; *iii)* $\overline{x}(y)$ (*bound output*) representing the output of a *private* name y on channel x; and *iv)* τ representing a silent action. In the actions above, we call x the *subject* of the action, and y the *object*. Binders induce the obvious notion of α-equivalence on processes. Processes are usually considered up to structral congruence which is the smallest congruence relation over process that includes α-equivalence, and the symmetric monoid laws for $+$ and $\mid$ see [19] for complete presentation of structural congruence.

The symbolic semantics adopted here mimicks the one defined in [21, 22] where transitions take the form $P \stackrel{M,\alpha}{\longmapsto} P'$ where the *enabling condition* M is a sequence of name matching required to fire action α. In Figure 1 we report the complete transition systems. We simply comments on rules COM and CLOSE and refere the reader to [22] for details[4]. Rule COM and CLOSE permit syncrhonising between input and output actions having distinct subjects. However, notice that the enabling conditions of both rules require that the subjects should be fused in order to fire the synchronisation.

Open bisimulation, originally defined in [22], is finer than the early and late bisimulations [15]; unlike early and late bisimulation, open bisimulation is a congruence and, although its definition may seem more complex at first sight, it has an "efficient" characterisation exploited in automated tools [22, 24].

We use a symbolic version of open bisimulation whose equivalence with concrete version can found at [22]. This definition is based on the notion of *distinc-*

[4] The only difference withd respect to [22] is that, as in [21], we use recursion instead of iteration.

$$\text{PREF}\ \ \alpha\,.\,P \overset{\emptyset,\alpha}{\longmapsto} P \qquad \text{RES}\ \ \frac{P \overset{\mu}{\longmapsto} P'}{(\boldsymbol{\nu} z)P \overset{\mu}{\longmapsto} (\boldsymbol{\nu} z)P'}\ \text{if } z \notin \mathrm{n}(\mu)$$

$$\text{SUM}\ \ \frac{P \overset{\mu}{\longmapsto} P'}{P + Q \overset{\mu}{\longmapsto} P'} \qquad \text{PAR}\ \ \frac{P \overset{\mu}{\longmapsto} P'}{P \mid Q \overset{\mu}{\longmapsto} P' \mid Q}\ \text{if } \mathrm{bn}(\alpha) \notin \mathrm{fn}(Q)$$

$$\text{MATCH}\ \ \frac{P \overset{M,\alpha}{\longmapsto} P'}{[x=y]P \overset{N,\alpha}{\longmapsto} P'}\ \text{where } N \triangleq \begin{cases} M[x=y] \text{ if } x \neq y \text{ and } x, y \notin \mathrm{bn}(\alpha) \\ M \text{ if } x = y \end{cases}$$

$$\text{COM}\ \ \frac{P \overset{M,x(z)}{\longmapsto} P', \quad Q \overset{N,\overline{y}z'}{\longmapsto} Q'}{P \mid Q \overset{L,\tau}{\longmapsto} P'\{z'/z\} \mid Q'}\ \text{where } L \triangleq \begin{cases} MN[x=y] \text{ if } x \neq y \\ MN \text{ otherwise} \end{cases}$$

$$\text{CLOSE}\ \ \frac{P \overset{M,x(z)}{\longmapsto} P', \quad Q \overset{N,\overline{y}(z)}{\longmapsto} Q'}{P \mid Q \overset{L,\tau}{\longmapsto} (\boldsymbol{\nu} z)(P' \mid Q')}\ \text{where } L \triangleq \begin{cases} MN[x=y] \text{ if } x \neq y \\ MN \text{ otherwise} \end{cases}$$

$$\text{OPEN}\ \ \frac{P \overset{M,\overline{x}y}{\longmapsto} P'}{(\boldsymbol{\nu} y)P \overset{M,\overline{x}(y)}{\longmapsto} P'}\ \text{if } y \notin \mathrm{n}(M) \cup \{x\}$$

$$\text{REC}\ \ \frac{Q\{x_1/y_1, \ldots, x_r/y_r\} \overset{\mu}{\longmapsto} Q'}{A\langle x_1, \ldots, x_r\rangle \overset{\mu}{\longmapsto} Q'}\ \text{if } A(y_1, \ldots, y_r) \overset{\text{def}}{=} Q$$

Fig. 1. Open Transition rules for π-calculus

tion relations, or simply *distinctions*. A distinction is a finite symmetric irreflexive binary relation over names. More precisely, a distinction relation is a pair (n, d) where n is a finite set of names, and d is a symmetric relation $d \subseteq n \times n$ such that for all $x \in n$, $(x, x) \notin d$. In the following, (n, d) is written as $d^{(n)}$ (possibly dropping the superscript when unimportant or clear from the context) and $\mathrm{n}(d^{(n)}) = \{x \in \mathcal{N} \mid \exists y \in \mathcal{N}.\ (x, y) \in d\}$ denote the names *occurring* in $d^{(n)}$ (or simply, the names of $d^{(n)}$). The set of distinctions on $\mathcal{N}$ is denoted by $\mathrm{D}(\mathcal{N})$.

Definition 2 (Symbolic Open Bisimulation [21]). *Let $\{\sim_d\}_{d \in \mathrm{D}(\mathcal{N})}$ be an indexed relation of symmetric process relations. The $\{\sim_d\}_{d \in \mathrm{D}(\mathcal{N})}$ is an* open bisimulation *if, whenever $P \overset{M,\alpha}{\longmapsto} P'$, with $\mathrm{bn}(\alpha) \cap (\mathrm{fn}(Q) \cup \mathrm{n}(d)) = \emptyset$ and M respects d, there are N, β, and Q' such that $Q \overset{N,\beta}{\longmapsto} Q'$ and*

- *$M \triangleright N$*
- *$\alpha = \beta\sigma_M$, and*
- *$P' \sim_{d'} Q'_{\sigma_M}$,*
  where $d' = d\sigma_M \cup (\{y\} \times \mathrm{fn}(P, Q))$ if $\alpha = \overline{x}(y)$, otherwise $d' = d$.

We work with transition system $d \vdash P \overset{M,\alpha}{\longmapsto} d' \vdash P'$ on *constrained process* $d \vdash P$, namely processes equipped with distinctions on (a superset of) their free names. The distinction relation is intended to record what names are distinct from the rest. We consider transitions obtained by the following inference rule

$$\frac{P \overset{M,\alpha}{\longmapsto} P', \ \ \sigma_M \text{ respects } d}{d \vdash P \overset{M,\alpha\sigma_M}{\longmapsto} d'\sigma_M \vdash P'\sigma_M}$$

where d' is computed from d as in Definition 2 and the substitutive effect of M is applied to the action of the transition and the residual of the process (σ_M is an idempotent substitution s.t. $\sigma_M(x) = \sigma_M(y)$ iff $M \Rightarrow x = y$).

3 Named Sets and Named Functions With Distinctions

This section introduces *named sets with distinctions* and the corresponding *named functions*, which extend the definitions introduced in [17, 4, 5]. Named sets are enriched set where elements are equipped with names, which become first-class citizens in the operational model of HD-automata. On the top of named sets with distinctions we build a new version of HD-automata suitable for a faithful representation of the open semantics of π-calculus. Following [17], we define HD-automata in terms of *permutation algebras* (defined below) which basically specify how name permutations act on the elements equipped with names.

We use bijective substitutions (automorphisms) on $\mathcal{N}$, namely *permutations*, which form a group where the operation is the function composition and the identity is the identity function on $\mathcal{N}$. For a permutation ρ, the *kernel of* ρ is the set of names that are permuted by ρ, namely $\ker \rho \stackrel{\text{def}}{=} \{x \in \mathcal{N} \mid \rho(x) \neq x\}$. We say that ρ is a *finite kernel permutation* if $\ker \rho$ is finite. In the following, $Aut(\mathcal{N})$ denotes the (enumerable) group of finite kernel permutations and $\mathrm{SubGrp}(Aut(\mathcal{N}))$ is the set of subgroups of $Aut(\mathcal{N})$.

Definition 3 (Permutation algebras). *If $\langle G, \cdot, id\rangle$ is a group, a set Q is a* G-set *if there exists $\bullet : G \times Q \to Q$, called* group action*, such that the following axioms hold:*

$$\forall \rho, \rho' \in G. \forall q \in Q.\ \rho \bullet (\rho' \bullet q) = (\rho \cdot \rho') \bullet q, \qquad \forall q \in Q.\ id \bullet q = q.$$

A permutation algebra *is a G-set for a subgroup G of $Aut(\mathcal{N})$.*

A permutation algebra specifies how to apply name permutations to the elements of its carrier set (through the interpretation of the group action).

The set of π-calculus processes up to structural congruence is a permutation algebra, indeed, $\bullet$ can be interpreted as the capture-avoiding application of substitutions to π-calculus terms and $\cdot$ is the usual function composition.

Named sets with distinctions (conservatively) extend traditional named sets. The definitions of named sets [17, 4, 5] simply associate a group of name permutations with their elements. Here, dns have in addition distinctions associated to their elements, i.e., relations yielding the constraints that names must satisfy.

Definition 4 (Named sets with distinctions). *A* named set with distinctions (dns) *is a triple $\langle Q, \mathrm{d}, \mathrm{g}\rangle$ where*

- *Q is a permutation algebra with respect to $G \in \mathrm{SubGrp}(\mathrm{Aut}(\mathcal{N}))$;*
- *$\mathrm{d} : Q \to \mathrm{D}(\mathcal{N})$ maps an element $q \in Q$ to a distinction $d^{(n)} \in \mathrm{D}(\mathcal{N})$;*

– $\mathrm{g} : Q \to \mathrm{SubGrp}(\mathrm{Aut}(\mathcal{N}))$ *maps elements* $q \in Q$ *to a subgroup* $\mathrm{g}(q)$ *of* G *such that, if* $\mathrm{d}(q) = d^{(n)}$ *and* $\rho \in \mathrm{g}(q)$*:*

$$\begin{array}{rcl} & \ker\rho & \subseteq \quad n \\ \textit{for all } x, y \in n : \quad & (x,y) \in \mathrm{d}(q) & \iff (\rho(x), \rho(y)) \in \mathrm{d}(q). \end{array}$$

For $q \in Q$*, if* $\mathrm{d}(q) = d^{(n)}$*, then* n *is the set of* names *of* q*, denoted as* $|q|$*.*

Basically, a dns is an enriched set where elements have names associated with them, elements of dnss are meant to represent terms with names. Therefore, an element q of a dns is equipped with a group of permutations $\mathrm{g}(q)$ that corresponds to equivalent syntactic representations of q.

According to Definition 4, permutations of q, must respect its name distinctions. For instance, suppose that $(x, y) \in \mathrm{d}(q)$ while $(z, y) \notin \mathrm{d}(q)$; then $\mathrm{g}(q)$ cannot contain a permutation ρ such that $\rho(z) = y$ or $\rho(z) = x$.

Notice that a traditional named set is a dns where, for any element, the associated distinction is $\emptyset^{(n)}$ for any $n \in \wp_{\mathrm{fin}}(\mathcal{N})$ which is trivially respected by any permutation.

We let E, F, G range over dnss and, given a dns $E = (Q, \mathrm{d}, \mathrm{g})$, we write Q_E, d_E, g_E for denoting Q, d and g, respectively.

As done for ordinary named set, we define *named functions* as mappings between dnss that preserve their structure. The typical representation of fresh names in HD-automata is through a distinguished element[5], say $\star \notin \mathcal{N}$; more precisely, we let $\mathcal{N}_\star$ denote the set $\mathcal{N} \cup \{\mathcal{N}_\star\}$.

Definition 5 (Named functions). *Given two dnss* E *and* F*, a* named function *(nf)* $H : E \to F$ *is a pair* (h, Σ) *where* $h : Q_E \to Q_F$ *is a function, and* $\Sigma : Q_E \to \wp({\mathcal{N}_\star}^{\mathcal{N}})$ *is such that, for all* $q \in Q_E$ *and* $\sigma \in \Sigma(q)$*,*

1. σ *is injective,* $\sigma(|h(q)|) \subseteq |q| \cup \{*\}$ *and* σ *is the identity outside* $|h(q)|$*;*
2. $\forall \sigma \in \Sigma(q).\ \forall x, y \in |h(q)|.\ (\sigma(x), \sigma(y)) \in \mathrm{d}_E(q) \Rightarrow (x, y) \in \mathrm{d}_F(h(q))$*;*
3. $\sigma; \mathrm{g}_E(q) \subseteq \Sigma(q)$*, (permutations are extended as the identity on* $\mathcal{N}_\star$*);*
4. $\mathrm{g}_F(h(q)); \sigma = \Sigma(q)$*.*

Named functions are ranged over by H, K and J. We write h_H and Σ_H for denoting the first and the second components of H. According to its definition, a nf H maps elements of a ns E to elements in F through the mapping h_H. However, names must be taken into account, therefore, the component Σ_H records "the history" of names when mapping $q \in E$ to $\mathrm{h}_H(q) \in F$. Such history is represented by means of a bunch of injective functions that send names in $\mathrm{h}_H(q)$ back either to names of q or to $\mathcal{N}_\star$ representing to a freshly generated name in $\mathrm{h}_H(q)$). Condition 2 states that nfs preserve distinctions, namely it ensures that names distinguished in q remain distinguished in $\mathrm{h}_H(q)$, (if they have a counterimage there). Notice that this condition does not prevent $\mathrm{d}_F(\mathrm{h}_H(q))$ to contain additional distinction pairs in $|\mathrm{h}_H(q)| \times |\mathrm{h}_H(q)|$.

[5] In [17], a different construction is exploited.

One might wonder why $\Sigma_H(q)$ is a set of mappings instead of a single function. The reason is that elements of nss are equipped with permutations that do not alter their meaning. Namely, for any $\rho \in \mathrm{SubGrp}(q)$, $q\rho$ is equivalent (under a given equivalence) to q (and similarly for $\mathrm{h}_H(q)$); the history Σ_H must be independent (or functional) with respect to the possible equivalent representations of q or of $\mathrm{h}_H(q)$. Condition 3 and 4 in Definition 5 are designed on this purpose. The former imposes that (opportunely) composing $\sigma \in \Sigma(q)$ with any equivalent representation of q yields a suitable history for the names; while the latter states that the representation of $\mathrm{h}_H(q)$ does not influence the history of names.

Actually, we can show that dnss and nfs are the objects and morphisms of a category, denoted by `DNSet`. This allows us to define HD-automata as a coalgebra for a suitable endofunctor of `DNSet`.

First we need to define morphisms composition.

Definition 6 (Composition of named functions). *The* composition $H;K$ *of two nfs* $H : E \to F$ *and* $K : F \to G$ *is* $\langle \mathrm{h}_H; \mathrm{h}_K, \Sigma_H; \Sigma_K \rangle$ *where* $\Sigma_H; \Sigma_K : Q_E \to \wp_{fin}({\mathcal{N}_\star}^{\mathcal{N}})$ *is such that*

$$\Sigma_H; \Sigma_K : q \mapsto \bigcup_{\sigma \in \Sigma_H(q)} \{\sigma; \sigma'[\star \mapsto \star] | \sigma' \in \Sigma_K(h_H(q))\}.$$

Proposition 1. *In Definition 6,* $H;K$ *is a nf. Composition of nfs is associative and has identities.*

Proof. The existence of identities is trivial. The proof of associativity of composition easily follows from associativity of composition of functions. □

Definition 7 (The category `DNSet`). *The* category `DNSet` *has dnss as objects and nfs as morphisms.*

Proposition 2 (Structure of `DNSet`). *The category* `DNSet` *has initial and final objects, and finite powerset functor defined as follows:*

1. *the initial object is given by* $\langle \emptyset, \emptyset \mapsto \emptyset^{(0)}, \emptyset \mapsto \emptyset \rangle$*;*
2. *terminal objects are given by* $I = \langle \{q\}, q \mapsto \emptyset^{(0)}, q \mapsto \emptyset \rangle$*;*
3. *the powerset functor is* $\wp_{fin}(E) = \langle \mathcal{P}_f(E_Q), \mathrm{d}, \mathrm{g} \rangle$ *where,*
   - $\mathcal{P}_f$ *is the covariant finite powerset on* **Set***,*
   - $\mathrm{d}(\{q_1, \ldots, q_j\}) = \mathrm{d}_E(q_1) \cup \ldots \cup \mathrm{d}_E(q_j)$*, where the union of two distinctions is defined as* $d^{(n)} \cup e^{(m)} \triangleq d \cup e^{(n \cup m)}$*.*
   - $\mathrm{g}(\{q_1, \ldots, q_j\}) = \{\rho \in \mathrm{g}_E(q_1) \cup \ldots \cup \mathrm{g}_E(q_j) \mid \rho \textit{ respects } \mathrm{d}(\{q_1, \ldots, q_j\})\}$.

The initial object is the empty dns, the final object is any singleton dns whose distinction is $\emptyset^{(0)}$ and permutation group is $\emptyset$, finally, the powerset functor is obtained by lifting the covariant powerset functor $\mathcal{P}$ on **Set**. Notice that the distinction of a finite subset of Q_E are obtained by considering the union of the distinctions of its elements.

Definition 8 (Pairing of dnss). *The* pairing $E \otimes F$ *of two dnss* E *and* F *is defined as* $E \otimes F \stackrel{\text{def}}{=} \langle Q_E \times Q_F, \mathrm{d}, \mathrm{g}\rangle$*, where* $\mathrm{d}(q, q') = \mathrm{d}_E(q) \cup \mathrm{d}_F(q')$ *and* $\mathrm{g}(q, q') = \{\rho \in \mathrm{g}_E(q) \cup \mathrm{g}_F(q') \mid \rho \text{ respects } \mathrm{d}(q, q')\}$.

Definition 9 (HD-automata). *Fixed a dns of labels* L*, a* HD-automaton *over* L *is a coalgebra for* $T_L(E) = \wp_{fin}(L \otimes E)$.

The minimisation algorithm on HD-automata (i.e., on a T_L-coalgebras) K is borrowed from [6] and is specified by the equations 1 and 2 below.

$$H_{(0)} \stackrel{\text{def}}{=} \langle q \mapsto \bot, q \mapsto \emptyset^{(0)}, q \mapsto \emptyset\rangle, \tag{1}$$

$$H_{(i+1)} \stackrel{\text{def}}{=} K; N(T(H_{(i)})), \tag{2}$$

where $T : \mathtt{DNSet} \to \mathtt{DNSet}$ is the functor specified as

$$T(E) = \begin{cases} T_L(E) & E \in \mathrm{obj}(\mathtt{DNSet}) \\ \langle \mathrm{h}, \Sigma\rangle & E = \langle h_E, \Sigma_E\rangle \in \mathtt{DNSet}(E, F) \text{ for } E, F \in \mathrm{obj}(\mathtt{DNSet}) \end{cases}$$

where, given $B \in \wp_{\mathrm{fin}}(L \otimes E)$,

$$\begin{aligned} \mathrm{h}(B) &= \{\langle l, \mathrm{h}_E(q)\rangle \mid \langle l, q\rangle \in B\} \\ \Sigma(B) &= \{\langle l, \mathrm{h}_E(q), \sigma; \sigma'\rangle \mid \langle l, q, \sigma'\rangle \in B \wedge \langle l, q', \sigma'\rangle \in \Sigma_E(q)\}. \end{aligned}$$

and N is a *normalisation functor* explained below. All the states of a HD-automaton are initially considered equivalent. Equation (2) specifies a generic iteration where K is composed with the nf at the i-th iteration after it has been applied to functors T and N. The algorithm builds the minimal realisation $\bar{H}$ of (finite) HD-automata by constructing (an approximation of) the final coalgebra morphism.

The normalisation functor removes the *redundant transitions*. Generally, redundant transitions are transitions describing behaviours that can be matched by "more general" transitions in the bisimulation game, where the meaning of "more general" depends on the behavioural equivalence at hand. For instance, in [4, 5] *redundancy* for the early semantics of π-calculus has been stated in terms of *active names* We generalise this concept by means of *redundant transitions* which occur when HD-automata are built out of a nominal calculus. During this phase, it is not possible to decide which are the redundant transitions[6]. Therefore, all the transitions are taken when HD-automata are built and redundant ones are removed during the minimisation. This is achieved by means of a *normalisation functor* which basically gets rid of redundant transitions.

Definition 10 (Normalisation functor). *A* normalisation functor N *is any functor such that* $N(E)$ *is isomorphic to a subset of* E.

The proof of the convergence of the algorithm is given in [6].

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

4 From Open π-calculus to HD-automata

This section describes how agents of π-calculus can be mapped onto HD-automata and what normalisation means for the HD-automata for open π-calculus. We first introduce labels and transitions and then define the normalisation functor for open π-calculus.

The labels of the canonical symbolic semantics of π-calculus consists of enabling conditions and actions; both of them can be represented as dnss.

Definition 11 (Matching dns). *Consider the function* $\widehat{\_}$ *inductively defined on matchings sequences as follows:*

$$\widehat{[x=y]} \stackrel{\text{def}}{=} \langle \{\diamond\}, \diamond \mapsto \emptyset^{(\{x,y\})}, \diamond \mapsto \{id, exch_{x,y}\}\rangle$$
$$\widehat{M[x=y]} \stackrel{\text{def}}{=} \langle \{\diamond\}, \diamond \mapsto \emptyset^{(n)}, \diamond \mapsto G\rangle$$

where $n = \mathrm{n}(M[x=y])$, $G = \{\rho^i \mid \exists \rho' \in \mathrm{g}_{\widehat{M}}(\diamond) \setminus \{id\}.\ \rho = \rho' \downarrow [x=y] \wedge i \geq 0 \wedge M\rho \triangleleft \triangleright M\}$ *and* $\downarrow$: $Aut(\mathcal{N}) \to Aut(\mathcal{N})$ *is defined as*

$$\rho \downarrow [x=y] = \begin{cases} \rho, & x,y \in \ker\rho \\ \rho; exch_{x,y}, & x,y \notin \ker\rho \\ \rho;[x\mapsto y];[y\mapsto\rho(x)], & x \in \ker\rho \wedge y \notin \ker\rho \\ \rho;[y\mapsto x];[x\mapsto\rho(y)], & y \in \ker\rho \wedge x \notin \ker\rho. \end{cases}$$

A matching dns *is a dns as described above and is ranged over by* $\mathtt{M}$.

Basically, matching dnss represent enabling conditions of open π-calculus transitions. Despite of the involved mathematical definition of the group of $\widehat{M}$, the intuition behind it is simple. Since any two names identified by M can be exchanged, G contains those permutations that yield logically equivalent representations of M.

Definition 12 (Labels for π-calculus). *The* dns of actions for π-calculus on $n \in \wp_{\mathrm{fin}}(\mathcal{N})$ *is* $L_\pi = \langle L, \mathrm{d}, l \mapsto \{id\}\rangle$ *where* $L = \{xy, \overline{x}y, x(y), \overline{x}(y) \mid x,y \in \mathcal{N}\} \cup \{\tau\}$ *and* $\mathrm{d} : L \to \mathrm{D}(\mathcal{N})$ *is defined as*

$$\mathrm{d}(l) = \begin{cases} \emptyset^{(n)}, & l \notin \{x(y), \overline{x}(y) \mid x,y \in \mathcal{N}\} \\ \{(y,u) \mid u \in n\}^{(n)}, & \text{otherwise} \end{cases}$$

The dns of labels of π-calculus *is the pairing of matching sequence dnss and* L_π*, it is denoted by Lab and is ranged over by* λ*. We let* $\mathrm{bn}((M, x(y))) = \mathrm{bn}((M, \overline{x}(y))) = \{y\}$ *while* $\mathrm{bn}(\lambda) = \emptyset$*, otherwise.*

Roughly, *Lab* equips labels of (the symbolic semantics of) π-calculus with a dns structure. Notice that L contains both *bound* input actions and *free* input actions, respectively denoted by $x(y)$ and xy. This is necessary for HD-automata because the names in arrival states of input transitions must be associate either to $\mathcal{N}_\star$ (in which case the transition is a bound input) or to a name of the source state (which correspond to free input transitions).

Let us denote with P_π the set of constrained processes of π-calculus. It is trivial to equip P_π with a dns structure by defining $\mathrm{d} : P_\pi \to \mathrm{D}(\mathcal{N})$ as the constant mapping $\mathrm{d} : d \vdash P \mapsto d$ and $\mathrm{g} : P_\pi \to \mathrm{SubGrp}(\mathcal{N})$ as $\mathrm{g} : q \mapsto \{id\}$.

The set of reachable processes from $d \vdash P \in P_\pi$, $\partial(d \vdash P)$, is defined as follows:

$$\partial(d \vdash P) \stackrel{\text{def}}{=} \{d \vdash P\} \cup \bigcup_{d\vdash P \xrightarrow{M,\alpha} d\vdash' Q} \partial(d \vdash' Q).$$

The HD-automaton associated to $d \vdash P$ is denoted as $K_{d\vdash P}$ and is the T_L-coalgebra such that $\mathrm{dom}(K_{d\vdash P}) = \partial(d \vdash P)$ and $\mathrm{cod}(K_{d\vdash P}) = T_{NS}(\partial(d \vdash P))$. Function $\mathrm{h}_{K_{d\vdash P}}$ associates a reachable state of $d \vdash P$ its outgoing transitions:

$$\mathrm{h}_{K_{d\vdash P}}(e \vdash Q) = \begin{array}{r} \{(\lambda, e' \vdash Q') \mid \lambda \text{ is not input } \wedge\, e \vdash Q \xrightarrow{\lambda} e' \vdash Q'\} \\ \cup \\ \{((M, xy), e' \vdash Q') \mid e \vdash Q \xrightarrow{M,x(z)} e' \vdash Q' \wedge y \in \mathrm{fn}(Q)\} \\ \cup \\ \{((M, x(y)), e'' \vdash Q') \mid e \vdash Q \xrightarrow{M,x(y)} e' \vdash Q' \wedge e'' = e' \cup \{(y, z) \mid z \in \mathrm{fn}(Q)\}\} \end{array}$$

where transitions in the first set are straitforwardly translated while input transitions require to consider input of known names (second set) from bound input transitions, where y is supposed to be a new name.

The mapping $\Sigma_{K_{d\vdash P}}$ associates to any $e \vdash Q \in \partial(d \vdash P)$ the set

$$\bigcup_{\substack{(\lambda, e' \vdash Q') \in \\ \mathrm{h}_{K_{d\vdash P}}(e \vdash Q)}} \{\sigma \in \mathcal{N}_\star^{\mathcal{N}} \mid \mathrm{bn}(\lambda) = \emptyset \Rightarrow \sigma = id \vee \mathrm{bn}(\lambda) = \{y\} \Rightarrow \sigma = id; y \mapsto \mathcal{N}_\star\}.$$

The HD-automaton obtained by this definition is a T_L-coalgebra by construction. Observe that infinite HD-automata might be obtained using the construction above, however, for interesting classes of π-calculus processes, e.g., *finitary* processes[7], the construction yields finite HD-automata [17, 20].

The normalisation functor for the symbolic open bisimulation of π-calculus relies on the definition of reduntant transition given for constrained processes in [21]. The idea is that a transition $d \vdash P \xrightarrow{M,\alpha} e \vdash Q$ is redundant, when there exists a transition $d \vdash P \xrightarrow{N,\beta} e' \vdash Q'$ such that

- $M \not\lhd\rhd N$,
- $\alpha = \beta\sigma_M$ and
- $e \vdash Q$ and $e'\sigma_M \vdash Q'\sigma_M$ are open bisimilar.

In words, N is a condition weaker than M such than α can be recovered by instantiating β with the further identifications in M and the processes reached by the transitions are still bisimilar under the conditions in M.

[7] Finitary processes are those processes that have a bounded degree of parallelism. The class of finitary agents is expressive enough for non-trivial specification, as witnessed by e.g. the Handover protocol [18] in π-calculus, which can trivially be specified also in the finitary π-calculus.

Definition 13 (Normalisation functor). *The* normalisation functor for the open bisimilarity of π-calculus *is denoted by* $\mathtt{N} : \mathtt{DNSet} \to \mathtt{DNSet}$ *and, on nfs of the form* $\langle \mathrm{h}, \Sigma \rangle \in \mathtt{DNSet}(\wp_{fin}(L_\pi \otimes E), \wp_{fin}(L_\pi \otimes F))$ *it yields* $\langle h', \Sigma' \rangle$ *where* $h' : B \mapsto \{(\lambda, q) \mid (\lambda, q)$ *non redundant in* $B\}$ *and* $\Sigma'(B)$ *is defined as*

$$\bigcup_{(\lambda,q)\in h'(B)} \{\sigma \in \Sigma(B) \mid \mathrm{bn}(\lambda) = \emptyset \Rightarrow \sigma = id \vee \mathrm{bn}(\lambda) = \{y\} \Rightarrow \sigma = id; y \mapsto \star\}.$$

On the other arrows and on objects of $\mathtt{DNSet}$, $\mathtt{N}$ *is the identity.*

Basically, N filters those transitions out of a given state q that are redundant because of the presence of another transition having weaker conditions on names.

Proposition 3. *The functor* $\mathtt{N}$ *is monotonic on nfs.*

Theorem 1. *The minimisation algorithm converges on finite HD-automata for* π*-calculus.*

Proof. The proof follows by the monotonicity of T and $\mathtt{N}$ and following the proof of Theorem 1 in [6].

Let us remark that normalisation through $\mathtt{N}$ is strictly related to symbolic open bisimulation, indeed it expactly applies redundancy conditions given in [21]. Hence, a tight relationship can be established between symbolic open hyper-bisimulation and the outcome of the minimisation algorithm.

Theorem 2 (Minimisation and open bisimulation). *Two finitary* π*-calculus processes are symbolic open bisimilar iff they have the same minimal realisation.*

Theorem 2 re-casts the results for early bisimulation of π-calculus [4, 5] and for hyperbisimulation of Fusion calculus [6] and the proof can be obtained mimicking the proofs therein.

5 Presheaf categories and HD-automata

In this section we give a more abstract presentation of the operational semantics of open π-calculus, following the pattern in [7] for late and early π-calculus.

First, we have to recall some constructions and results from [1, 12]. The main insight of the model of open bisimulation given in [1] is that the domains must be staged according to the knowledge represented by distinction relations; thus the index category must be a category of distinctions. In fact, it is easy to see that distinctions are the objects of a small category:

Definition 14 (The category $\mathbb{D}$). *The category* $\mathbb{D}$ *of* distinctions relations *is the full subcategory of Rel of irreflexive, symmetric binary relations over* $\mathcal{N}$ *with a finite carrier set.*

Here *Rel* is the category of relations and monotone functions, hence a morphisms $f : d^{(n)} \to e^{(m)}$ is any function $f : n \to m$ (i.e., a name substitution) which preserves the distinction relation (if $(x, y) \in d$ then $(f(x), f(y)) \in e$). In other words, substitutions cannot map two related (i.e., definitely distinct) names to the same name of a later stage, while unrelated names can coalesce.

Structure of $\mathbb{D}$. The category $\mathbb{D}$ has products, coproducts and pullbacks. Note that $\mathbb{D}$ has no terminal object, but it has initial object $(\emptyset, \emptyset)$. In fact, $\mathbb{D}$ inherits meets, joins and partial order from $\wp(\mathcal{N})$:

- $d^{(n)} \wedge e^{(m)} = (d \cap e)^{(m \cap n)}$, and $d^{(n)} \vee e^{(m)} = (d \cup e)^{(m \cup n)}$
- $d^{(n)} \leq e^{(m)}$ iff $d \wedge e = d$, that is, iff $d \subseteq e$.

For each n, let us denote $\mathbb{D}_n$ the full subcategory of $\mathbb{D}$ whose objects are all relations over n. Then, $\mathbb{D}_n$ is a complete Boolean algebra. Let $\bot^{(n)} \triangleq (n, \emptyset)$ and $\top^{(n)} \triangleq (n, n^2 \setminus \Delta_n)$ be the *empty* and *complete* distinction on n, respectively, where $\Delta : \mathbb{F} \to Rel$ is the *diagonal* functor defined as $\Delta_n = (n, \{(i,i) \mid i \in n\})$.

$\mathbb{D}$ can be given another monoidal structure. Let us define $\oplus : \mathbb{D} \times \mathbb{D} \to \mathbb{D}$ as

$$d_1^{(m)} \oplus d_2^{(n)} = (m+n, d_1 \cup d_2 \cup \{(i,j),(j,i) \mid i \in m, j \in n\}).$$

Then, it is easy to see that $(\mathbb{D}, \oplus, \bot^{(0)})$ is a symmetric monoidal category.

By applying coproduct and tensor to $\bot^{(1)}$ we get two distinguished *dynamic allocation* functors $\delta^-, \delta^+ : \mathbb{D} \to \mathbb{D}$, as $\delta^- \triangleq \bot^{(1)} + \_$ and $\delta^+ \triangleq \bot^{(1)} \oplus \_$. More explicitly, the action of δ^+ on objects is $\delta^+(d^{(n)}) = d_{+1}^{(n+1)}$ where $d_{+1} = d \cup \{(*,i),(i,*) \mid i \in n\}$. Thus both δ^- and δ^+ add an extra element to the carrier, but, as the superscript $^+$ is intended to suggest, δ^+ adds in *extra* distinctions.

Functors over $\mathbb{D}$ $\mathbf{Set}^{\mathbb{D}}$ is the category of functors from $\mathbb{D}$ to $\mathbf{Set}$ (often called *presheaves (over* $\mathbb{D}^{op}$*)*) and natural transformations. The structure of $\mathbb{D}$ lifts to $\mathbf{Set}^{\mathbb{D}}$, which has the necessary type constructors for representing the behaviour of an open semantics in π-calculus:[8]

1. *Products and coproducts*, which are computed pointwise (as with all limits and colimits in functor categories); e.g. $(P \times Q)_{d^{(n)}} = P_{d^{(n)}} \times Q_{d^{(n)}}$. The terminal object is the constant functor $\mathcal{K}_1 = \mathbf{y}(\bot^{(\emptyset)})$: $\mathcal{K}_1(d) = 1$.
2. A presheaf of *atoms* $Atom \in \mathbf{Set}^{\mathbb{D}}$, $Atom = \mathbf{y}(\bot^{(1)}) = \mathbf{y}(\top^{(1)})$. The action on objects is $Atom(d^{(n)}) = n$.
3. Two *dynamic allocation* functors $\delta^-, \delta^+ : \mathbf{Set}^{\mathbb{D}} \to \mathbf{Set}^{\mathbb{D}}$, induced by each $\kappa \in \{\delta^+, \delta^-\}$ on $\mathbb{D}$ as $\_ \circ \kappa : \mathbf{Set}^{\mathbb{D}} \to \mathbf{Set}^{\mathbb{D}}$.
4. Let $\wp_f$ be the finite (covariant) powerset functor on $\mathbf{Set}$; then $\wp_f \circ \_ : \mathbf{Set}^{\mathbb{D}} \to \mathbf{Set}^{\mathbb{D}}$ is the *finite powerset* operator on $\mathbb{D}$-presheaves.
5. *Exponentials* are defined as usual in functor categories:

$$(B^A)_d \triangleq \mathbf{Set}^{\mathbb{D}}(A \times \mathbb{D}(d, \_), B)$$
$$(B^A)_f(m) \triangleq m \circ (id_A \times (\_ \circ f)) \quad \text{for } f : d \to e \text{ in } \mathbb{D}, m : A \times \mathbb{D}(d, \_) \longrightarrow B$$

Using these constructions, we can define the behaviour functor $\mathcal{B}_o : \mathbf{Set}^{\mathbb{D}} \to \mathbf{Set}^{\mathbb{D}}$ suitable for open bisimulation in π-calculus as follows:

$$\mathcal{B}_o P = \mathcal{P}_f(P + N \times N \times P + N \times \delta^+ P + N \times \delta^- P)$$

[8] We shall use the same symbols for the lifted structure, but ensuring the reader has enough information to deduce which category we are working in.

A coalgebra $(P, k : P \dot{\longrightarrow} \mathcal{B}_o P)$ for $P \in \mathbf{Set}^{\mathbb{D}}$ induces transition relation over the state space given by the elements of $\int P$. We can describe these transition systems as follows:

Definition 15 ($\mathbb{D}$-transition systems). *A $\mathbb{D}$-transition system consists of a presheaf $P : \mathbf{Set}^{\mathbb{D}}$ and a graph such that for a set $Act = \{a(), \overline{a}(), \overline{a}b, \tau\}$ with obvious action for $\sigma : \mathcal{N} \longrightarrow \mathcal{N}$.*

- *The nodes are labelled by pairs (p, d) where $p \in P_d$*
- *The edges are labelled by elements of the set Act such that if*
  - $(p, d) \xrightarrow{a()} (p', d')$ *then* $d' = \delta^- d$ *and* $a \in d$
  - $(p, d) \xrightarrow{\overline{a}()} (p', d')$ *then* $d' = \delta^+ d$ *and* $a \in d$
  - $(p, d) \xrightarrow{\overline{a}b} (p', d')$ *then* $d' = d$ *and* $a, b \in d$
  - $(p, d) \xrightarrow{\tau} (p', d')$ *then* $d' = d$
- *If* $(p, d) \xrightarrow{\alpha} (p', \kappa d)$ *where* $\kappa \in \{Id, \delta^-, \delta^+\}$ *and* $\sigma : d \to d'$ *then* $(P_\sigma(p), d') \xrightarrow{Act_\sigma(\alpha)} (P_\sigma(p'), \kappa d')$.
- *conversely: given* $\sigma : d \to d'$, *if* $(P_\sigma(p), d') \xrightarrow{\alpha'} (p', \kappa d')$ *where* $\kappa \in \{Id, \delta^-, \delta^+\}$, *then there exist* q *such that* $(p, d) \xrightarrow{\alpha} (q, \kappa d)$ *and* $\alpha' = Act_\sigma(\alpha)$, $p' = P_\sigma(q)$.

Proposition 4. *$\mathbb{D}$-transition systems are in one-to-one correspondence with $\mathcal{B}_o$-coalgebras.*

Proof. The structure map of the coalgebra gives us exactly the transition relation while the naturality of the coalgebra structure map corresponds exactly to the closure condition of open transitions under distinction preserving renamings.

Pullback-preserving functors In order to relate the category of `DNSet` with presheaves over $\mathbb{D}$, we need to restrict our attention to "well-behaving" presheaves, that is presheaves whose elements have finite support. Let us recall a general definition of support and a result from [12]:

Definition 16 (support). *Let $\mathcal{C}$ be a category, $F : \mathcal{C} \to \mathbf{Set}$ a functor, C an object of $\mathcal{C}$, and $a \in F_C$. A subobject $i : D \rightarrowtail C$ of C* supports *a* (at C) *if there exists a (not necessarily unique) $b \in F_D$ such that $a = F_i(b)$.*

Proposition 5. *Let $\mathcal{C}$ have pullbacks, $F : \mathcal{C} \to \mathbf{Set}$ be pullback-preserving, C be in $\mathcal{C}$, and $x \in F_C$. If both C_1, C_2 support x, then $C_1 \wedge C_2$ supports x.*

As a corollary of this proposition, if the objects of $\mathcal{C}$ are finite objects (as in the case of $\mathbb{D}$), there must be a minimum stage supporting x; let us denote this object as $\mathrm{supp}^(x)$.

In virtue of these results, we restrict our attention to the full subcategory of $\mathbf{Set}^{\mathbb{D}}$ of pullback preserving functors; we denote this category by $\mathcal{D}$.

Clearly, functor $\mathcal{B}_o$ restricts to $\mathcal{D}$, thus leading to a class of $\mathcal{B}_o$ coalgebras whose carrier is a pullback-preserving functor. Similarly, the definition of $\mathbf{Set}^{\mathbb{D}}$-transition systems restricts naturally to the subcategory $\mathcal{D}$, yielding a new notion

of labelled transition system which we call $\mathcal{D}$-*LTS*. Also, the correspondence of Proposition 4 restricts to these subclasses of coalgebras and LTS.

The next theorem will constitute a step towards relating coalgebra over a functor of $\mathcal{D}$ and HD-automata.

Theorem 3. *The category of* `DNSet` *is essentially equivalent to* $\mathcal{D}$.

Proof. To a dns $(X, \mathrm{d}, \mathrm{g})$, we associate a functor $P : \mathbb{D} \to Set$ defined as

$$P_d = \{(x, s) \mid x \in X, s \in \mathbb{D}(\mathrm{d}(x), d)\}/_\sim$$

where $(x, s) \sim (x, s')$ iff $s \circ \pi = s' \circ \pi'$ for some $\pi, \pi' \in \mathrm{g}(x)$. It is easy to check that this P is pullback-preserving, and hence an object of $\mathcal{D}$.

Conversely, given a presheaf P in $\mathcal{D}$, we can define a named set with distinctions $(X, \mathrm{d}, \mathrm{g})$ where

- $X = (\int P)/_\approx = (\Sigma_{d \in \mathbb{D}} P_d)/_\approx$, where $(d, x) \approx (d', x')$ iff there exist $s : d \to d''$, $s' : d' \to d''$ such that $P_s(x) = P_{s'}(x')$.
- for $(d, x) \in X$, let us define $\mathrm{d}((d, x))$ as the minimal distinction supporting x, which exists because P is pullback preserving.
- for $(d, x) \in X$, we define the subgroup $\mathrm{g}((d, x)) = \{\pi \in Aut(\mathrm{d}((d, x))) \mid \mathrm{im}(\pi_{|d}) = d \text{ and } P_{\pi_{|d}}(x) = x\}$

The resulting named set $(X, \mathrm{d}, \mathrm{g})$ is mapped, by the construction defined above, to a presheaf Q which is isomorphic to P. □

Now, notice that a HD-automata P for open semantics is given by a graph whose states, labels, and transitions are drawn from a named set subject to the following conditions:

1. Only one transition per source, label, target.
2. Input is parametric on the data.
3. A bound output transition tansits into a bigger state.

Thus, following the proof given by Fiore and Staton in [7] in the case of $Sh(\mathbb{I}^{op})$ and traditional HD-automata, we can see that there is a one-to-one correspondence between HD-automata with state in P and $\mathcal{D}$-LTS.

6 Conclusions and Related work

In this paper we have presented *HD-automata with distinctions*, based on *named sets with distinctions*. These notions extend the traditional ones of HD-automata with symmetries and named sets. HD-automata with distinctions and are suitable for a faithful representation of the open semantics of π-calculus. We have indeed applied our constructions for mapping symbolic open semantics of π-calculus to HD-automata with distinctions. Finally, we also have established a connection between named sets with distinctions and pullback-preserving functors over the index category of distinctions.

Related work. Symbolic semantics [10, 2, 11] is a well established approach both for theoretical investigation and for verification of nominal calculi. For instance, symbolic approaches been exploited to provide a convenient characterisation of *open bisimilarity* [22] and in the design of the corresponding bisimulation checker, e.g., *Mobility WorkBench* [24]. The main advantage of the symbolic semantics is that it yields transition systems which are smaller than the correspondig "concret" transition systems. Usually, symbolic semantics takes a *syntax-based* approach and generalises standard operational semantics by keeping track of equalities among names: transitions are derived in the context of such constraints.

An alternative class of models for nominal calculi are the so-called *syntax-free* models where names are explicitly dealt with regardless of the syntactic structure of the calculi. *Indexed LTSs* [3] are examples of syntax-free models of nominal calculi developed following the approach based on name permutations.

HD-automata [20, 17] have an added value with respect to indexed LTS because they are equipped with powerful verification techniques. In [20, 17], states of HD-automata have been equipped with name symmetries which further reduces the size of the automata and guarantee the existence of the minimal realization. HD-automata serves as an operational model of history dependent formalism and in particular for nominal calculi, the leading example being π-calculus. All these formalism have common property that they may generate a resource and refere it at later computation stage [16]. For π-calculus this means that an agent generate a new name and refere to it at a latter stage.

The computation of the minimal HD-automata presented in this paper is derived by exploiting a coalgebraic presentation of the partition refinement algorithm [4], furtherly refined in [6]. The minimisation algorithm for the early semantics of π-calculus has been implemented in the Mihda toolkit [5]. Hence, the integration of symbolic techniques and syntax-free models would provide more powerful verification methods. Notice that minimisation algorithms for syntax-based models have been already developed (e.g., for the open semantics of the π-calculus[21]).

In [16] HD-automata has been successfully used to model early and late semantics of π-calculus and in [6] hyperbisimulation of fusion calculi. Open bisimulation in π-calculus on the other hand, is more complicated than early and late bisimulation, because open bisimulation is closed under all distinction preserving renaming. HD-automata introduced in [16] can not cope with this problem as they have no means of representing distinction on names. The solution we offer is based on understanding that the extend the state of HD-automata from a named set to a named relation. The closure under all distinction preserving renaming are already handled in the symbolic semantics as in [6]. This is to be contrasted with Pistore [20] where he introduce negative transition.

A related work worth mentioning is the coalgebraic model of open bisimulation by Ghani et al. [9], where they model open bisimulation using coalgebras over presheafs over $\mathbb{D}$. Another related work is [3], where the notion of $\mathbb{F}$-LTS is introduced. An extensive study of indexed LTS, such as $\mathbb{I}$-LTS, i-LTS, $\mathbb{B}$-LTS, is carried out in [7], where a correspondence between i-LTS and (standard) HD-automata is shown. However, none of these LTSs deal with open bisimulation.

References

1. J. Adamek, editor. *Coalgebraic Methods in Computer Science*, ENTCS. Elsevier, 2004.
2. M. Boreale and R. De Nicola. A Symbolic Semantics for the π-calculus. *Information and Computation*, 126(1):34–52, April 1996.
3. G. Cattani and P. Sewell. Models for Name-Passing Processes: Interleaving and Causal (Extended Abstract). *Information and Computation*, 190(2):136–178, May 2004.
4. G. Ferrari, U. Montanari, and M. Pistore. Minimizing transition systems for name passing calculi: A co-algebraic formulation. In M. Nielsen and U. Engberg, editors, *Foundations of Software and Computer Science*, volume 2303 of *Lecture Notes in Computer Science*, pages 129–143. Springer, 2002.
5. G. Ferrari, U. Montanari, and E. Tuosto. Coalgebraic Minimisation of HD-automata for the π-Calculus in a Polymorphic λ-Calculus. *Theoretical Computer Science*, 331:325–365, 2005.
6. G. Ferrari, U. Montanari, E. Tuosto, B. Victor, and K. Yemane. Modelling and Minimising the Fusion Calculus Using HD-Automata. In J. Fiadeiro, N. Harman, M. Roggenbach, and J. Rutten, editors, *Algebra and Coalgebra in Computer Science*, volume 3629 of *LNCS*, pages 142 – 156. Springer-Verlag, 2005.
7. M. Fiore and S. Staton. Comparing operational models of name-passing process calculi. In Adamek [1].
8. M. Fiore and D. Turi. Semantics of name and value passing. In H. Mairson, editor, *Proc. 16th LICS*, pages 93–104, Boston, USA, 2001. The Institute of Electrical and Electronics Engineers, Inc., IEEE Computer Society Press.
9. N. Ghani, K. Yemane, and B. Victor. Relationally staged computation in calculi of mobile processes. In Adamek [1].
10. M. Hennessy and H. Lin. Symbolic Bisimulations. *Theoretical Computer Science*, 138(2):353–389, February 1995.
11. H. Lin. Complete Inference Systems for Weak Bisimulation Equivalences in the π-Calculus. *Information and Computation*, 180(1):1–29, January 2003.
12. M. Miculan and K. Yemane. A unifying model of variables and names. In V. Sassone, editor, *Proc. FOSSACS'05*, volume 3441 of *Lecture Notes in Computer Science*, Apr. 2005.
13. R. Milner. *Communicating and Mobile Systems: the π-calculus*. Cambridge University Press, 1999.
14. R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, I and II. *Information and Computation*, 100(1):1–40,41–77, September 1992.
15. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part I/II. *Journal of Information and Computation*, 100:1–77, Sept. 1992.
16. U. Montanari and M. Pistore. An introduction to history dependent automata. In A. Gordon, A. Pitts, and C. Talcott, editors, *Conference Record of HOOTS II*, volume 10 of *ENTCS*. Elsevier Science Publishers, 1997.
17. U. Montanari and M. Pistore. π-calculus, structured coalgebras, and minimal HD-automata. In M. Nielsen and B. Rovan, editors, *Proc. MFCS 2000*, volume 1893 of *Lecture Notes in Computer Science*, pages 569–578. Springer-Verlag, 2000.
18. F. Orava and J. Parrow. An algebraic verification of a mobile network. *Formal Aspects of Computing*, 4(5):497–543, 1992.
19. J. Parrow. An introduction to the pi-calculus. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, pages 479–543. Elsevier Science, 2001.

20. M. Pistore. *History Dependent Automata*. PhD thesis, Computer Science Department, Università di Pisa, 1999.
21. M. Pistore and D. Sangiorgi. A Partition Refinement Algorithm for the π-Calculus. In R. Alur, editor, *Proceedings of CAV '96*, volume 1102 of *LNCS*, pages 38–49, 1996.
22. D. 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.
23. D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.
24. B. Victor and F. Moller. The Mobility Workbench — a tool for the π-calculus. In D. Dill, editor, *Proceedings of CAV '94*, volume 818 of *LNCS*, pages 428–440. Springer, 1994.