Minimizing Transition Systems for Name Passing Calculi: A Co-algebraic Formulation*

Gianluigi Ferrari[1], Ugo Montanari[1], and Marco Pistore[2]

[1] Dipartimento di Informatica, Università di Pisa
[2] ITC-IRST, Trento

Abstract. We address the problem of minimizing labelled transition systems for name passing calculi. We show how the co-algebraic formulation of automata with naming directly suggests an effective minimization algorithm which reduces the number of states to be analyzed in the verification of properties for name passing process calculi.

1 Introduction

Automatic methods for verifying finite state automata have been showed to be surprisingly effective [3]. Indeed, finite state verification techniques have enjoyed substantial and growing use over the last years. For instance several communication protocols and hardware systems of considerable complexity have been formalized and proved correct by exploiting finite state verification techniques. Since the state space is still very large for many designs, much research has been devoted to find techniques to combat state explosion. *Semantic minimization* [18,9] is a rather general technique which reduces the number of states to be analyzed by producing a minimal automaton that is equivalent to the original one. The minimal automaton is is indistinguishable from the original system with respect to the set of properties specified in many logical systems (e.g. μ-calculus) but easier to handle due to its possibly much smaller size [6]. The possibility of generating a minimal system provides further advantages. First, the minimal automaton can be exploited to verify different properties of the same system. Second, systems are usually obtained by composing components. Hence, minimizing components before combining them yields smaller state spaces.

The advent of world-wide networks and wireless communications are contributing to a growing interest in dynamic and reconfigurable systems. Unfortunately, finite state verification of these systems is much more difficult. Indeed, in this case, even simple systems can generate infinite state spaces. An illustrative example is provided by the π-calculus [11]. Its primitives are simple but expressive: channel names can be created, communicated (thus giving the possibility of dynamically reconfiguring process acquaintances) and they are subjected to sophisticated scoping rules. The π-calculus has greater expressive power than ordinary process calculi, but the possibility of dynamically generating new names leads also to models which are infinite-state and infinite branching.

* Work partially supported by FET Global Project PROFUNDIS and MURST Project COMETA

M. Nielsen and U. Engberg (Eds.): Fossacs 2002, LNCS 2303, pp. 129–143, 2002.

Creation of new names is actually quite common also in practice. An example is provided by the creation of nonces to identify sessions in security protocols (see [10] for a critical review of the state-of-the-art on verification of security protocols). All these techniques are based on finite state verification, and typically can ensure error freedom only for a finite amount of the behaviour of protocols. Even if many protocols do not include iterations, however, an unbound number of principals may take part in the interleaved sessions of the protocol; moreover, many known attacks exploit the mixed interleaving of different sessions of the same protocol.

Hence, traditional finite state models of behaviour have important shortcomings to address the challenges of supporting verification of dynamic distributed systems via semantic equivalence. To support finite state verification of dynamic distributed systems two of the authors have introduced a novel foundational model called *History Dependent Automata* (*HD-automata*) [12,15]. HD-automata have been specifically designed to allocate and garbage collect names. The theory ensures that finite state, finite branching automata give a faithful representation of the behaviour of π-calculus processes. Furthermore, HD-automata are expressive enough to represent formalisms equipped with mobility, locality, and causality primitives [13,14]. The finite state representation of π-calculus processes given by the HD-automata has been exploited to check behavioral properties of the agents [8,7]. Few other approaches have been proposed for finite state verification of π-calculus processes [4,17]. The explicit management of names provided by HD-automata allows for an extremely compact representation of the behaviour of processes. HD-automata are not only a convenient format for representing the behaviour of π-calculus processes in a compact way, though. They may be considered the natural extension of automata to calculi with name passing and name generation also from a theoretical point of view. In particular, HD-automata can be defined as coalgebras on the top of permutation algebras of states [16]. The permutation algebra describes the acts of name permutations (i.e. renaming) on state transitions. This information is sufficient to describe in a semantically correct way the creation, communication, and deallocation of names: all the features needed to describe and reason about formalisms with operations over names.

General results concerning coalgebras [1,19,20,2,21] extend also to these kinds of coalgebras: they make sure that a final coalgebra exists and that it defines minimal realizations of HD-automata. Also, equivalent HD-automata, and hence equivalent π-calculus processes, have isomorphic minimal realizations.

Our aim in this paper is to tackle the problem of minimizing labelled transition systems for name passing calculi. Instead of presenting our construction in the abstract setting of category theory, namely the category of coalgebras for an endofunctor on some base category, we shall be working in the standard universe of sets and functions. The main point is to provide a concrete representation of the terminal coalgebra which directly suggests the data structures of a minimization module of a semantic-based verification environment. Indeed, a key goal of this paper is to provide a clear relationship between the structures of

the semantic world and the data structures for the implementation. A main contribution of this paper is to show how the (theoretical) coalgebraic formulation of automata with naming directly suggests an effective minimization algorithm. The work reported in this paper is closely related to the theoretical development of coalgebras, and, at the same time, it formally specifies an effective procedure to perform finite state verification of distributed systems with dynamic name generation via semantic equivalence.

2 Transition Systems, Co-algebras and Minimization

A transition system T is a structure $(S, L, \rightarrow)$, where S is the set of *states*, L is the set of (action) *labels* and $\rightarrow \subseteq S \times L \times S$ is the *transition relation*. Usually, one writes $s \xrightarrow{\ell} s'$ to indicate $(s, \ell, s') \in \rightarrow$. In this section we provide a concrete representation of the terminal coalgebra (of an endofunctor over `Set`) which will yield the minimal transition system. Hereafter, we use the following notations:

- `Set` is the collection of all sets. By convention, Q : `Set` denotes a set and $q : Q$ denotes an element in the set Q;
- `Fun` is the collection of functions among sets. The function space over sets will have the following structure:

$$\texttt{Fun} \ = \{H \mid H = \langle S : \texttt{Set} \ , D : \texttt{Set} \ , h : S \rightarrow D \rangle\}.$$

  By convention we use S_H, D_H and h_H to denote the components of an element of `Fun`.

Let H and K be functions (i.e. elements of `Fun`), then the *composition* of H and K $(H; K)$ is defined provided that $S_K = D_H$ and it is the function given by $S_{H;K} = S_H$, $D_{H;K} = D_K$, and $h_{H;K} = h_K \circ h_H$.

Sometimes, we shall need to work with surjective functions. Let H be a function, then $\widehat{H}$ is the function given by:

- $S_{\widehat{H}} = S_H$, $D_{\widehat{H}} = \{q' : D_H \mid \exists q : S_H, h_H(q) = q'\}$ and $h_{\widehat{H}} = h_H$.

Transition systems are described co-algebraically employing two components: a set Q (the state space) together with a function $K : Q \rightarrow \wp(L \times Q)$ where $\wp(X)$ is the finite powerset of X. The idea is that function K determines the behaviour of the transition system: $K(q)$ is the set of pairs (ℓ, q') such that $q \xrightarrow{\ell} q'$. Functor $T(X) = \wp(L \times X)$ operates on both sets and functions, and characterizes a whole category of labelled transition systems, i.e. of coalgebras. In this paper, we aim at developing a concrete co-algebraic description of the minimization procedure, hence, we rephrase coalgebras in terms of certain structures called *bundles*.

Let L be the set of labels (ranged over by ℓ), then a *bundle* β over L is a structure $\langle D : \texttt{Set} \ , Step : \wp(L \times D) \rangle$. Given a fixed set of labels L, by convention, B denotes the collection of bundles and $\beta : B$ identifies the bundle β. Intuitively, the notion of bundle has to be understood as giving the data

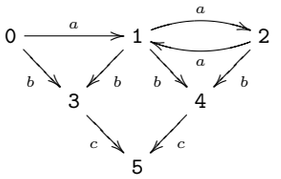
structure representing all the state transitions out of a given state. It details which states are reachable by performing certain actions.

The following clauses define functor T.

- $T(Q) = \{\beta : B \mid D_\beta = Q\}$, for each $Q : \mathtt{Set}$;
- For each $H : \mathtt{Fun}$, $T(H)$ is defined as follows:
  - $S_{T(H)} = T(S_H)$ and $D_{T(H)} = T(D_H)$;
  - $h_{T(H)}(\beta : T(S_H)) = \langle D_H, \{\langle \ell, h_H(q)\rangle \mid \langle \ell, q\rangle : Step_\beta\}\rangle$.

Definition 1. *Let L be a set of labels. Then a labelled transition system over L is a co-algebra for functor T, namely it is a function K such that $D_K = T(S_K)$.*

Aa already mentioned a co-algebra K for functor T represents a transition system where S_K is the set of states, and $h_K(q) = \beta$, with $D_\beta = S_K$ and $Step_\beta = \{(\ell, q') \mid q \xrightarrow{\ell} q'\}$. Figure 1 illustrates a labelled transition system and its coalgebraic formulation via the mapping h_K.



$$
\begin{aligned}
h_K(0) &= \langle S_k, \{\langle a,1\rangle, \langle b,3\rangle\}\rangle \\
h_K(1) &= \langle S_k, \{\langle a,2\rangle, \langle b,3\rangle, \langle b,4\rangle\}\rangle \\
h_K(2) &= \langle S_k, \{\langle a,1\rangle, \langle b,4\rangle\}\rangle \\
h_K(3) &= \langle S_k, \{\langle c,5\rangle\}\rangle \\
h_K(4) &= \langle S_k, \{\langle c,5\rangle\}\rangle \\
h_K(5) &= \langle S_k, \emptyset\rangle
\end{aligned}
$$

Fig. 1. A labelled transition system and its coalgebraic specification

General results (e.g. [1]) ensure the existence of the final coalgebra for a large class of functors . These results apply to our formulation of transition systems. In particular, it is interesting to see the result of the iteration along the terminal sequence [21] of functor T.

Let K be a transition system, and let $H_0, H_1, \ldots, H_{n+1}, \ldots$ be the sequence of functions computed by $H_{n+1} = K;\widehat{T(H_n)}$, where H_0 is the unique function from S_K to the one-element set $\{*\}$ given by $S_{H_0} = S_K$; $D_{H_0} = \{*\}$; and $h_{H_0}(q : S_{H_0}) = *$. Finiteness of $\wp$ ensures convergence of the iteration along the terminal sequence. We can say much more if the transition system is finite state.

Theorem 1. *Let K be a finite-state transtion system. Then,*

- *The iteration along the terminal sequence converges in a finite number of steps, i.e. $D_{H_{n+1}} \equiv D_{H_n}$,*
- *The isomorphism mapping $F : D_{H_n} \to D_{H_{n+1}}$ yields the minimal realization of transition system K.*

Comparing the co-algebraic construction with the standard algorithm [9,6] which constructs the minimal labelled transition system we can observe:

- at each iteration i the elements of D_{H_i} are the blocks of the minimization algorithm (i.e. the i-th partition). Notice that the initial approximation D_{H_0} contains a single block: in fact H_0 maps all the states of the transition system into $\{*\}$.
- at each step the algorithm creates a new partition by identifying the *splitters* for states q and q'. This corresponds in our co-algebraic setting to the fact that $H_i(q) = H_i(q')$ but $H_{i+1}(q) \neq H_{i+1}(q')$.
- the iteration proceeds until a stable partition of blocks is reached: then the iteration along the terminal sequence converges.

We now apply the iteration along the terminal sequence to the coalgebraic formulation of the transition system of Figure 1. The initial approximation is the function H_0 defined as follows

$$H_0 = \langle S_{H_0} = S_K, D_{H_0} = \{*\}, h_{H_0}(q) = * \rangle$$

We now construct the first approximation H_1. We have that

$$h_{H_1}(q) = \langle D_{H_0}, \{\langle \ell, h_{H_0}(q') \rangle : q \xrightarrow{\ell} q'\} \rangle$$

$$T(H_0)\langle \{1,2,3,4,5\}, \{\langle a,2\rangle, \langle b,3\rangle, \langle b,4\rangle\}\rangle = \langle \{*\}, \{\langle a,*\rangle, \langle b,*\rangle\}\rangle$$

In our example we obtain the function h_{H_1} and the destination state $D_{H_1} = \{\beta_1, \beta_2, \beta_3\}$ as detailed below.

$$\begin{array}{rl} h_{H_1}(0) = \langle \{*\}, \{\langle a,*\rangle, \langle b,*\rangle\}\rangle & \\ h_{H_1}(1) = \langle \{*\}, \{\langle a,*\rangle, \langle b,*\rangle\}\rangle & \beta_1 = \langle \{*\}, \{\langle a,*\rangle, \langle b,*\rangle\}\rangle \\ h_{H_2}(2) = \langle \{*\}, \{\langle a,*\rangle, \langle b,*\rangle\}\rangle & \beta_2 = \langle \{*\}, \{\langle c,*\rangle\}\rangle \\ h_{H_1}(3) = \langle \{*\}, \{\langle c,*\rangle\}\rangle & \beta_3 = \emptyset \\ h_{H_1}(4) = \langle \{*\}, \{\langle c,*\rangle\}\rangle & \\ h_{H_1}(5) = \langle \{*\}, \emptyset\rangle & \end{array}$$

We continue to apply the iterative construction, we obtain:

$$\begin{array}{rl} h_{H_2}(0) = \langle D_{H_1}, \{\langle a,\beta_1\rangle, \langle b,\beta_2\rangle\}\rangle \\ h_{H_2}(1) = \langle D_{H_1}, \{\langle a,\beta_1\rangle, \langle b,\beta_2\rangle\}\rangle \\ h_{H_2}(2) = \langle D_{H_1}, \{\langle a,\beta_1\rangle, \langle b,\beta_2\rangle\}\rangle \\ h_{H_2}(3) = \langle D_{H_1}, \{\langle c,\emptyset\rangle\}\rangle \\ h_{H_2}(4) = \langle D_{H_1}, \{\langle c,\emptyset\rangle\}\rangle \\ h_{H_2}(5) = \langle D_{H_1}, \emptyset\rangle \end{array}$$

Since $D_{H_2} \equiv D_{H_1}$ the iterative construction converges, thus providing the minimal labelled transition system illustrated in Figure 2, where $\bullet_1 = \{0,1,2\}$, $\bullet_2 = \{3,4\}$ and $\bullet_3 = \{5\}$.

3 A Co-algebraic Formulation of HD-Automata

In this section we extend the co-algebraic minimization to *HD-automata*. HD-automata [15] are specifically designed to provide a compact representation of the

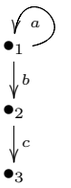


Fig. 2. Minimal labelled transition system

behavior of name passing process calculi. Names appear explicitly in the states of an HD-automaton: the idea is that the names associated to a state are the names which may play a role in the state evolution. A set $\{v_1, \ldots, v_{|q|}\}$ of local names is associated with each state q, and a permutation group G_q on $\{v_1, \ldots, v_{|q|}\}$ defines what renamings leave unchanged the behavior of q. Moreover, the identity of names is local to the state: states which differ only for the order of their names are identified. Due to the usage of local names, whenever a transition is performed a name correspondence between the name of the source state and the names of the target state is explicitly required.

3.1 Named Sets

Now we introduce the notion of *named sets*. Hereafter, we use $A \stackrel{bij}{\rightarrow} B$ ($A \stackrel{inj}{\rightarrow} B$) to denote a bijective (injective) function from A to B.

Definition 2. *A named set A is a structure*

$$A = \langle\, Q : \mathtt{Set}, |\_| : Q \longrightarrow \omega, \leq : Q \times Q \longrightarrow Bool, G : \prod_{q:Q} \wp(\{v_1..v_{|q|}\} \stackrel{bij}{\longrightarrow} \{v_1..v_{|q|}\}) \,\rangle$$

where $\forall q : Q_A$, $G_A(q)$ is a permutation group and $\leq_A$ is a total ordering.

A named set is a set of states equipped with a mechanism to give local meaning to names occurring in each state. In particular, function $|\_|$ yields the number of local names of states. Moreover, the permutation group $G_A(q)$ allows one to describe directly the renamings that do not affect the behaviour of q, i.e., symmetries among the local names of q. Finally, we assume that states are totally ordered. By convention we write $\{q : Q_A\}_A$ to indicate the set $\{v_1..v_{|q|_A}\}$ and we use `NSet` to denote the universe of named sets. In the definition above, the general product $\prod$ is employed (as usual in type theory) to type functions f such that the type of $f(q)$ is dependent on q.

Definition 3. *A named function H is a structure*

$$H = \langle\, S : \mathtt{NSet}, D : \mathtt{NSet}, h : Q_S \longrightarrow Q_D, \Sigma : \prod_{q:Q_S} \wp(\{h(q)\}_D \stackrel{inj}{\longrightarrow} \{q\}_S) \,\rangle$$

where $\forall q : Q_{S_H}, \forall \sigma : \Sigma_H(q)$, $G_{D_H}(h_H(q)); \sigma = \Sigma_H(q)$ and $\sigma; G_{S_H}(q) \subseteq \Sigma_H(q)$.

As in the case of standard transition systems, functions are used to determine the next state of transitions. As states are equipped with local names, a name correspondence (the mapping H_h) is needed to describe how names in the destination state are mapped into names of the source state. However, since names of corresponding states $(q, h_H(q))$ in h_H are defined up to permutation groups, we must equip H with a *set* $\Sigma_H(q)$ of injective functions. Since the name correspondence must be functional, the whole set $\Sigma_H(q)$ must be generated by saturating any of its elements by the permutation group of $h_H(q)$, and the result must be invariant with respect to the permutation group of q.

Named functions can be composed in the obvious way. Let H and K be named functions. Then $H; K$ is defined only if $D_H = S_K$, and

- $S_{H;K} = S_H$, $D_{H;K} = D_K$,
- $h_{H;K} : Q_{S_H} \longrightarrow Q_{D_K} = h_H; h_K$,
- $\Sigma_{H;K}(q : Q_{S_H}) = \Sigma_K(h_H(q)); \Sigma_H(q)$

Let H be a named function, $\widehat{H}$ denotes the surjective component of H:

- $S_{\widehat{H}} = S_H$ and $Q_{D_{\widehat{H}}} = \{q' : Q_{D_H} \mid \exists q : Q_{S_H}.h_H(q) = q'\}$,
- $|q|_{D_{\widehat{H}}} = |q|_{D_H}$,
- $G_{D_{\widehat{H}}}(q) = G_{D_H}(q)$,
- $h_{\widehat{H}}(q) = h_H(q)$,
- $\Sigma_{\widehat{H}}(q) = \Sigma_H(q)$

3.2 Bundles over π-Calculus Actions

To deal with HD-automata and named sets, the notion of bundle must be enriched. First we have to define the set of labels of transitions. Labels of transitions must distinguish among the different meanings of names occurring in π-calculus actions, namely synchronization, bound output (scope extrusion) free output, bound input, output where the subject and the object coincide, and so on. The set of π-calculus labels L_π is the set $\{TAU, BOUT, OUT, BIN, IN\}$. However, this is not enough. We still have to specify how many names are associated to labels. For instance, no name is associated to TAU (synchronization) labels, whereas one name, is associated to bound output $BOUT$ labels. Let $|\_|$ be the weight map associating to each π-label the set of indices of distinct names the label refers to. The weight map is defined as follows:

$$|TAU| = \emptyset \quad |BOUT| = |BIN| = \{1\} \quad |OUT| = |IN| = \{1, 2\}$$

Definition 4. *A bundle β consists of the structure*

$$\beta = \langle\, \mathcal{D} : \texttt{NSet}, Step : \wp(qd\ \mathcal{D}) \,\rangle$$

where $qd\,\mathcal{D}$ is the set of quadruples of the form $\langle\, \ell, \pi, \sigma, q \,\rangle$ given by

$$qd\,\mathcal{D} = \{\langle\, \ell : L_\pi, \pi : |\ell| \xrightarrow{inj} \{v_1..\}, q : Q_\mathcal{D}, \sigma : \prod_{\ell \in L_\pi} \{q\}_\mathcal{D} \xrightarrow{inj} Q\ell \,\rangle\}.$$

and

$$Q\ell = \begin{cases} \{*, v_1..\} & \text{if } \ell \in \{BOUT, BIN\} \\ \{v_1..\} & \text{if } \ell \notin \{BOUT, BIN\} \end{cases}$$

under the constraint that $G_{\mathcal{D}_\beta}(q); S_q = S_q$, *where* $S_q = \{\langle\ \ell, \pi, q, \sigma\ \rangle \in Step_\beta\}$ *and* $\rho; \langle\ \ell, \pi, q, \sigma\ \rangle = \langle\ \ell, \pi, q, \rho; \sigma\ \rangle$.

The intuition is that a bundle provides the abstraction to describe the successor set of a state. More precisely, if $\langle\ \ell, \pi, q, \sigma\ \rangle \in qd\mathcal{D}$, then q is the destination state; ℓ is the label of the transition; π associates to the label the names observed in the transition; and σ states how names in the destination state are related with the names in the source state. Notice that the distinguished element $*$ belongs to the names of the source state when a new name is generated in the transition.

Some additional operations over bundles will be helpful. Given a function f, define f^* to be its $*$-extension as follows:

$$f^*(x) = \begin{cases} * & \text{if } x = * \\ f(x) & \text{otherwise} \end{cases}$$

Let $\{\!|\, \beta \,|\!\}$ be the function over bundles given by

$$\{\!|\, \beta \,|\!\} = \bigcup_{\langle\ \ell, \pi, q, \sigma\ \rangle \in Step_\beta} rng(\pi) \cup rng(\sigma) \setminus \{*\}$$

where rng yields the range of functions. Function $\{\!|\, \beta \,|\!\}$ gives the set of names which occur in the bundle β. Hereafter, we will only consider bundles β such that $\{\!|\, \beta \,|\!\}$ is finite. Moreover, we will use $\lfloor \beta \rfloor$ to indicate the number of names which occur in the bundle β (i.e. $\lfloor \beta \rfloor = |\{\!|\, \beta \,|\!\}|$).

The most important construction on bundles is the *normalization* operation. This operation is necessary for two different reasons. The first reason is that there are different equivalent ways for picking up the step components (i.e. quadruples $\langle\ \ell, \pi, q, \sigma\ \rangle$) of a bundle. We assume to have an ordering relation over the quadruples in $qd\,\mathcal{D}$, which yields an ordering $\sqsubseteq$ over the bundles on $\mathcal{D}$. This ordering relation will be used to define canonical representatives of bundles. The ordering on quadruples can be defined non ambiguously only assuming an ordering on $\mathcal{D}$. This is why we introduced an ordering relation on named sets in the first place.

The second, more important, reason for normalizing a bundle is for removing from the step component of a bundle all the input transitions which are *redundant*. Consider for instance the case of a state q having only one name v_1 and assume that the following two tuples appear in a bundle:

$$\langle\ IN, xy, q, \{v_1 \to y\}\ \rangle \qquad \text{and} \qquad \langle\ BIN, x, q, \{v_1 \to *\}\ \rangle.$$

Then, the first tuple is redundant, as it expresses exactly the same behavior of the second tuple, except that a "free" input transition is used rather than a "bound" one. Hence, the transformation removes the first tuple from the bundle. We have to remark that redundant transitions occur when building the HD-automaton for a π-calculus agent. Indeed, it is not possible to decide which free

input transitions are required, and which transitions are covered by the bound input transition[1]. The solution to this problem consists of adding a superset of the required free input transitions when the HD-automaton is built, and to exploit a reduction function to remove the ones that are unnecessary. During the iterative execution of the minimization algorithm, bundles are split: this means that the set of redundant components of bundles decreases. Hence, when the iterative construction terminates, only those free inputs that are really redundant have been removed from the bundles.

The normalization of a bundle β is done in different steps. In the first step, the bundle is reduced by removing all the possibly redundant input transitions. Reduction function $red(\beta)$ on bundles is defined as follows:

- $\mathcal{D}_{red(\beta)} = \mathcal{D}_\beta$,
- $Step_{red(\beta)} = Step_\beta \setminus \{\langle\, IN, xy, q, \sigma \,\rangle \mid \langle\, BIN, x, q, \sigma' \,\rangle : Step_\beta \wedge \sigma' = (\sigma; \{y \to *\})\}$.

Once the redundant input transitions have been removed, it is possible to associate to bundle β the set of its "active names" $an_\beta = \{\!|\, red(\beta) \,|\!\}$. These are the names that appear either in a destination state or in a label of a non-redundant transition of the bundle. Finally, the *normalization* function $norm(\beta)$ is defined as follows:

- $\mathcal{D}_{norm(\beta)} = \mathcal{D}_\beta$
- $Step_{norm(\beta)} = min_{\sqsubseteq}\big(Step_\beta \setminus \{\langle\, IN, xy, q, \sigma \,\rangle \mid y \notin an_\beta\}\big)$,

where $min_{\sqsubseteq}$ is the function that returns the minimal permutation of a given bundle with respect to order $\sqsubseteq$. More precisely, given a bundle β and a permutation $\theta : \{\!|\, \beta \,|\!\} \xrightarrow{bij} \{\!|\, \beta \,|\!\}$, bundle $\beta;\theta$ is defined as $\mathcal{D}_{\beta;\theta} = \mathcal{D}_\beta$, $step_{\beta;\theta} = \{(\ell, \pi; \theta, q, \sigma; \theta) \mid (\ell, \pi, q, \sigma) : \beta\}$. The bundle $min_{\sqsubseteq}\overline{\beta}$ is the minimal bundle in $\{\overline{\beta};\theta \mid \theta : \{\!|\, \overline{\beta} \,|\!\} \xrightarrow{bij} \{\!|\, \overline{\beta} \,|\!\}\}$, with respect to the total ordering $\sqsubseteq$ of bundles over $\mathcal{D}$. In the following, we use $perm(\beta)$ to denote the canonical permutation that associates $Step_{norm(\beta)}$ and $Step_\beta \setminus \{\langle\, IN, xy, q, \sigma \,\rangle \mid y \notin an_\beta\}$.

We remark that, while *all* the IN transitions covered by BIN transitions are removed in the definition of $red(\beta)$, only those corresponding to the reception of non-active names are removed in the definition of $norm(\beta)$. In fact, even if an input transition is redundant, it might be the case that it corresponds to the reception of a name that is active due to some other transitions.

Finally, we need a construction which extracts in a canonical way a group of permutations out of a bundle. Let β be a bundle, define $Gr\ \beta$ to be the set $\{\rho \mid Step_\beta; \rho^* = Step_\beta\}$.

Proposition 1. *$Gr\ \beta$ is a group of permutations.*

3.3 The Minimization Algorithm

We are now ready to introduce the functor T that defines the co-algebras for HD-automata. The action of functor T over named sets is given by:

[1] In the general case, to decide whether a free input transition is required it is as difficult as to decide the bisimilarity of two π-calculus agents.

- $Q_{T(A)} = \{\beta : Bundle \mid \mathcal{D}_\beta = A, \beta \text{ normalized}\}$,
- $|\beta|_{T(A)} = \lfloor \beta \rfloor$,
- $G_{T(A)}(\beta) = Gr\ \beta$,
- $\beta_1 \leq_{T(A)} \beta_2$ iff $Step_{\beta_1} \sqsubseteq Step_{\beta_2}$,

while the action of functor T over named functions is given by:

- $S_{T(H)} = T(S_H)$, $D_{T(H)} = T(D_H)$,
- $h_{T(H)}(\beta : Q_{T(S_H)}) : Q_{T(D_H)} = norm(\beta')$,
- $\Sigma_{T(H)}(\beta : Q_{T(S_H)}) = Gr(norm(\beta')); (perm(\beta'))^{-1}; inj : \{\!|\, norm(\beta') \,|\!\} \longrightarrow \{\beta\}_{T(S_H)}$ where $\beta' = \langle D_H, \{\langle\, \ell, \pi, h_H(q), \sigma'; \sigma \,\rangle \mid \langle\, \ell, \pi, q, \sigma \,\rangle : Step_\beta, \sigma' : \Sigma_H(q)\}\rangle$.

Notice that functor T maps every named set A into the named set $T(A)$ of its *normalized* bundles. Also a named function H is mapped into a named function $T(H)$ in such a way that every corresponding pair $(q, h_H(q))$ in h_H is mapped into a set of corresponding pairs $(\beta, norm(\beta'))$ of bundles in $h_{T(H)}$. The quadruples of bundle β' are obtained from those of β by replacing q with $h_H(q)$ and by saturating with respect to the set of name mappings in $\Sigma_H(q)$. The name mappings in $\Sigma_{T(H)}\beta$ are obtained by transforming the permutation group of bundle $norm(\beta')$ with the inverse of the canonical permutation of β' and with a fixed injective function inj mapping the set of names of $norm(\beta')$ into the set of names of β, defined as $i < j$, $inj(v_i) = v_{i'}$ and $inj(v_j) = v_{j'}$ implies $i' < j'$. Without bundle normalization, the choice of β' among those in $\beta'; \theta$ would have been arbitrary and not canonical with the consequence of mapping together fewer bundles than needed.

Definition 5. *A transition system over named sets and π-actions is a named function K such that $D_K = T(S_K)$.*

HD-automata are particular transition systems over named sets. Formally, an HD-automaton A is given by:

- the elements of the state Q_A are π-agents $p(v_1..v_n)$ ordered lexicographically: $p_1 \leq_A p_2$ iff $p_1 \leq_{lex} p_2$
- $|p(v_1..v_n)|_A = n$,
- $G_A q = \{id : \{q\}_A \longrightarrow \{q\}_A\}$, where id denotes the identity function,
- $h : Q_A \longrightarrow \{\beta \mid \mathcal{D}_\beta = A\}$ is such that $\langle\, \ell, \pi, q', \sigma \,\rangle \in Step_{h(q)}$ represent the π-calculus transitions from agent q.

We remark that bundle $Step_{h(q)}$ should not contain all the transitions from q, but only a representative subset. For instance, it is not necessary to consider a free input transition where the received name is not active provided that there is a bound input transition which differs from it only for the bound name. Finally, by using renaming σ in the element of the bundles, it is possible to identify all those π-agents that differ only for an injective renaming. In the following, we represent as $q \xrightarrow[\pi]{\ell}_\sigma q'$ the "representative" transitions from agent q that are used in the construction of the HD-automaton.

We can now define the function K.

- $S_K = A$,
- $h_K(q) = norm(h(q))$,
- $\Sigma_K(q) = Gr(h_K(q)); (perm(h(q)))^{-1}; inj : \{\!|\, h(q) \,|\!\} \longrightarrow \{q\}_A$

We now construct the minimal HD-automata by an iterative procedure. We first need to define the initial approximation. Given a HD-automata K, the initial approximation H_0 is defined as follows:

- $S_{H_0} = S_K$, $D_{H_0} = unit$ where $Q_{unit} = \{*\}$, $|*|_{unit} = 0$ (and hence $\{*\} = \phi$), $G_{unit}\, * = \phi$, and $* \leq_{unit} *$,
- $h_{H_0}(q : Q_{s_{H_0}}) = *$,
- $\Sigma_{H_0} q = \{\phi\}$

The formula which details the iterative construction is given by

$$H_{n+1} = \widehat{K; T(H_n)}.$$

Two of the authors have developed in [16] a final coalgebra model for the π-calculus. In particular, the early transition system is modelled as a coalgebra on a suitable category of permutation algebras, and early bisimilarity is obtained as the final coalgebra of such a category. HD-automata are then proposed as a compact representation of these transition systems. It is possible to adapt the results of [16] to our framework, and hence, to prove convergence of the iteration along the terminal sequence. Furthermore, if we consider finite state HD-automata (i.e. finite state HD automata associated to finitary π-calculus processes) we can prove a stronger result.

Theorem 2. *Let K be a finite state HD-automaton. Then*

- *The iteration along the terminal sequence converges in a finite number of steps: n exists such that $D_{H_{n+1}} \equiv D_{H_n}$,*
- *The isomorphism mapping $F : D_{H_n} \rightarrow D_{H_{n+1}}$ yields the minimal realization of the transition system K up to strong early bisimilarity.*

Sketch of the proof. It is possible to see that at every iteration either a block is split (i.e. there are q and q' with $h_{H_n}(q) = h_{H_n}(q')$ but $h_{H_{n+1}}(q) \neq h_{H_{n+1}}(q')$); or, if no block is split, some block acquires additional names (i.e. there is q with $\mid h_{H_n}(q) \mid< \mid h_{H_{n+1}}(q) \mid$); or, if there are no additional names the group of permutations of some block decreases (i.e. there is q with $G_{h_{H_n}}(q) \supset G_{h_{H_{n+1}}}(q)$). Since blocks cannot be indefinitely split (there is a finite number of states), the number of names cannot indefinitely increase (each block cannot have a number of names larger than any state in it) and groups of permutations cannot indefinitely decrease (they are finite) every terminal sequence is well-founded.

The following functional expression (in a extended λ-calculus) makes the iteration step of the normalization algorithm explicit.
$h_{H_{n+1}} = (\lambda q.norm\ \langle A, \{\langle\, \ell, \pi, q', \sigma\, \rangle \mid q \xrightarrow[\pi]{l}_{\sigma} q'\}\rangle)$;

$$\lambda\beta.norm\ \langle D_{H_n}, \{\langle\ \ell, \pi, h_{H_n}(q), \sigma'; \sigma\ \rangle \mid \langle\ \ell, \pi, q, \sigma\ \rangle : Step_\beta, \sigma' : \Sigma_{H_n}(q)\}\rangle$$

$$h_{H_{n+1}}(q) = norm\ \langle D_{H_n}, \{\langle\ \ell, \pi, h_{H_n}(q'), \sigma'; \sigma\ \rangle \mid q \xrightarrow[\pi]{\ell}_\sigma q', \sigma' : \Sigma_{H_n}(q')\}\rangle.$$

Notice that the normalization on the transition system is absorbed by the normalization on the resulting bundle.

4 Minimizing Behaviours of π-Calculus Processes

In this section we provide an example of minimization by considering a transition system which describes the behaviour of a π-calculus process. Let $S(x, y, z)$ be the π-calculus process

$$S(x, y, z) = x!y.R(x, y, z) + y!x.R(x, y, z)$$
$$R(x, y, z) = x?(w).S(x, y, w) + y?(w).S(y, x, z)$$

Here, we use $x!y$ (resp $x?(y)$) to denote output (resp. input) actions. Process $S(x, y, z)$ syntactically contains name z, however this name is not active in any system evolution. As we will see, name z disappears in the minimal realization for $S(x, y, z)$. The behaviour of the process $S(x, y, z)$ is illustrated by the labelled transition system displayed in the upper part of Figure 3.

The transition system has been automatically generated using the HAL environment [7,8]. The HAL environment is an integrated tool set for the specification, verification and analysis of mobile system specified in the π-calculus. HAL contains several modules to check bisimilarity; however, it does not include any module to minimize the state space of π-calculus processes under analysis. Notice that we extended labels of transitions. For instance, label $IN2$ is used to describe an input action where subject and object names coincide.

Hereafter, we adopt the following notations for name permutations. Term id_n denotes the identity permutation over n names and term $exch_2$ denotes the permutation that exchanges two names.

Let S_K be the set $\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$, where $G_q = id_{|q|}\ \ \forall q : S_K$. The occurrences of names in the states are given by $|q_0| = |q_1| = |q_4| = 3$, $|q_2| = |q_3| = |q_5| = |q_6| = 2$, where $|q| = n$ means that state q contains names $\{v_1, v_2, \ldots, v_n\}$. The mapping h_K is displayed in the lower part of Figure 3.

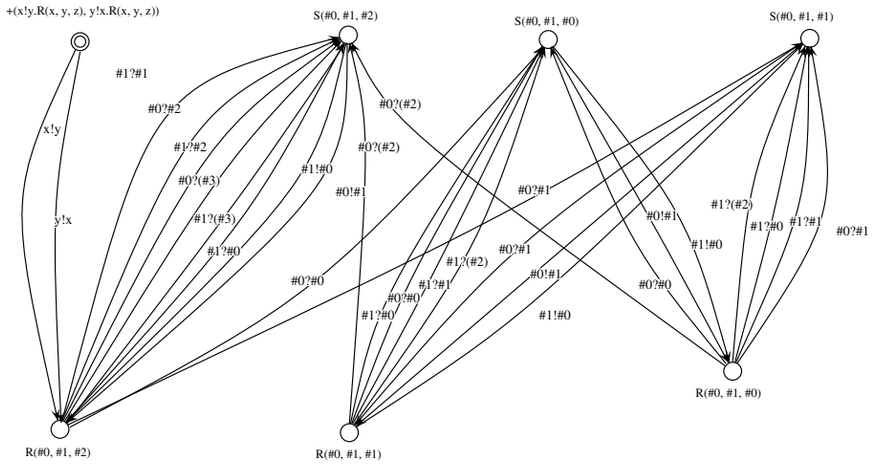
Finally, let $\Sigma_K q$ be the injection of the names occurring in $h_k q$ into the corresponding names of q (i.e., either id_2 or id_3).

The initial approximation is the function H_0 defined as follows

$$H_0 = \langle S_{H_0} = S_K, D_{H_0} = unit, h_{H_0} = \lambda q : S_K.*, \Sigma_{H_0} q = \emptyset\rangle$$

Applying the definition of the iterative algorithm we have

$$q_0, q_2, q_3, q_4 \xrightarrow{h_{H_1}} \beta_1 \quad q_1, q_5, q_6 \xrightarrow{h_{H_1}} \beta_2$$



$$
\begin{aligned}
h_K(q_0) = \langle S_k, \{\ & \langle OUT, v_1v_2, q_1, id_3\rangle, \\
& \langle OUT, v_2v_1, q_1, id_3\rangle\}\rangle \\
h_K(q_1) = \langle S_k, \{\ & \langle IN2, v_1, q_2, id_2\rangle, \\
& \langle IN, v_1v_2, q_3, id_2\rangle, \\
& \langle IN, v_1v_3, q_4, id_3\rangle, \\
& \langle BIN, v_1, q_4, id_2 \cup (v_3, *)\rangle, \\
& \langle IN, v_2v_1, q_4, exch_2 \cup (v_3, v_3)\rangle, \\
& \langle IN2, v_2, q_4, exch_2 \cup (v_3, v_3)\rangle, \\
& \langle IN, v_2v_3, q_4, exch_2 \cup (v_3, v_3)\rangle, \\
& \langle BIN, v_2, q_4, exch_2 \cup (v_3, v_3)\rangle\}\rangle \\
h_K(q_2) = \langle S_k, \{\ & \langle OUT, v_1v_2, q_6, id_2\rangle, \\
& \langle OUT, v_2v_1, q_6, id_2\rangle\}\rangle \\
h_K(q_3) = \langle S_k, \{\ & \langle OUT, v_1v_2, q_5, id_2\rangle, \\
& \langle OUT, v_2v_1, q_5, id_2\rangle\}\rangle \\
h_K(q_4) = \langle S_k, \{\ & \langle OUT, v_1v_2, q_1, id_3\rangle, \\
& \langle OUT, v_2v_1, q_1, id_3\rangle\}\rangle \\
h_K(q_5) = \langle S_k, \{\ & \langle IN2, v_1, q_2, id_2\rangle, \\
& \langle IN, v_1v_2, q_3, id_2\rangle, \\
& \langle BIN, v_1, q_4, id_2 \cup (v_3, *)\rangle, \\
& \langle IN, v_2v_1, q_2, exch_2\rangle, \\
& \langle IN2, v_2, q_2, exch_2\rangle, \\
& \langle BIN, v_2, q_2, exch_2\rangle\}\rangle \\
h_K(q_6) = \langle S_k, \{\ & \langle IN2, v_1, q_2, id_2\rangle, \\
& \langle IN, v_1v_2, q_3, id_2\rangle, \\
& \langle BIN, v_1, q_4, id_2 \cup (v_3, *)\rangle, \\
& \langle IN, v_2v_1, q_3, exch_2\rangle, \\
& \langle IN2, v_2, q_3, exch_2\rangle, \\
& \langle BIN, v_2, q_3, exch_2\rangle\}\rangle
\end{aligned}
$$

Fig. 3. An example of HD-automaton and its coalgebraic specification.

where bundles β_1 and β_2 and their associated permutation groups are defined as follows:

$$\begin{aligned}
\beta_1 &= \langle D_{H_0}, \{\ \langle OUT, v_1v_2, *, \emptyset\rangle, \langle OUT, v_2v_1, *, \emptyset\rangle\}\rangle \\
\beta_2 &= \langle D_{H_0}, \{\ \langle IN2, v_1, *, \emptyset\rangle, \\
&\qquad \langle IN, v_1v_2, *, \emptyset\rangle, \\
&\qquad \langle BIN, v_1, *, \emptyset\rangle, \\
&\qquad \langle IN, v_2v_1, *, \emptyset\rangle, \\
&\qquad \langle IN2, v_2, *, \emptyset\rangle, \\
&\qquad \langle BIN, v_2, *, \emptyset\rangle\}\rangle
\end{aligned}$$

$$\begin{aligned}
\lfloor \beta_1 \rfloor = 2 \qquad & G\beta_1 = \{id_2, exch_2\} \\
\lfloor \beta_2 \rfloor = 2 \qquad & G\beta_2 = \{id_2, exch_2\} \\
\Sigma_{H_1} q = (id_2, exch_2) \quad & \text{for all } q \in S_K.
\end{aligned}$$

Notice that the normalization construction has removed redundant transitions from bundles. Hence, the overall number of transitions decreases.

Applying again the definition of the iterative algorithm we have

$$q_0, q_2, q_3, q_4 \xrightarrow{h_{H_2}} \beta'_1 \qquad q_1, q_5, q_6 \xrightarrow{h_{H_2}} \beta'_2$$

where bundles β'_1 and β'_2 and their associated permutation groups are defined as follows:

$$\begin{aligned}
\beta'_1 &= \langle D_{H_1}, \{\ \langle OUT, v_1v_2, \beta_2, id_2\rangle, \langle OUT, v_2v_1, \beta_2, id_2\rangle, \\
&\qquad \langle OUT, v_1v_2, \beta_2, exch_2\rangle, \langle OUT, v_2v_1, \beta_2, exch_2\rangle\}\rangle \\
\beta'_2 &= \langle D_{H_1}, \{\ \langle IN2, v_1, \beta_1, id_2\rangle, \\
&\qquad \langle IN, v_1v_2, \beta_1, id_2\rangle, \\
&\qquad \langle BIN, v_1, \beta_1, id_2\rangle, \\
&\qquad \langle IN, v_2v_1, \beta_1, exch_2\rangle, \\
&\qquad \langle IN2, v_2, \beta_1, exch_2\rangle, \\
&\qquad \langle BIN, v_2, \beta_1, exch_2\rangle\}\rangle
\end{aligned}$$

$$\begin{aligned}
\lfloor \beta'_1 \rfloor = 2 \qquad & G\beta'_1 = \{id_2, exch_2\} \\
\lfloor \beta'_2 \rfloor = 2 \qquad & G\beta'_2 = \{id_2, exch_2\} \\
\Sigma_{H_2} q = (id_2, exch_2) \quad & \text{for all } q \in S_K.
\end{aligned}$$

Since $D_{H_1} \equiv D_{H_2}$, the iterative algorithm terminates and H_2 defines the minimal HD-automaton. Notice that the minimal HD-automaton contains 2 states and 10 transitions rather than 7 states and 28 transitions of the original HD-automaton.

5 Concluding Remarks

In this paper we have developed a minimization procedure for finite state verification of name passing calculi. The minimization algorithm is automatically derived from the co-algebraic formulation. We plan to experiment with the minimization algorithm to evaluate its usefulness in practice by equipping the HAL verification environment with a module performing minimization of HD-automata. We are currently developing a prototype implementation of the minimization algorithm for HD-automata in O-Caml.

References

1. P. Aczel, N. Mendler, A Final Coalgebra Theorem, in *Category Theory and Computer Science*, LNCS 389, 1989.
2. A. Corradini, R. Heckel, U. Montanari, Compositional SOS and Beyond: A Coalgebraic View of Open Systems, Theoretical Computer Science, to appear.
3. E. Clarke and J. Wing Eds. Formal Methods: State of the Art and Future Directions. ACM Comp. Surv., December 1996.
4. M. Dam, On the decidability of process equivalences for the π-calculus. *Theoretical Computer Science*, 183(2):215–228, 1997.
5. B. Jacobs, J. Rutten A tutorial on (co)algebras and (co)induction. Bulletin of EATCS, 62, 222-259, 1997.
6. J.-C., Fernandez, An implementation of an efficient algorithm for bisimulation equivalence. Science of Computer Programming 13, 219-236, 1990.
7. G. Ferrari, G. Ferro, S. Gnesi, U. Montanari, M. Pistore, G. Ristori. An Automata-based Verification Environment for Mobile Processes In TACAS'97, LNCS 1217, 1997.
8. G. Ferrari, S. Gnesi, U. Montanari, M. Pistore, G. Ristori, Verifying Mobile Processes in the HAL Environment. CAV'98 , LNCS 1427, 1998.
9. P. Kannellakis, S. Smolka, CCS expressions, finite state processes, and three problems of equivalence. Information and Computation, 86, 43-68, 1990.
10. C. Meadows Open Issues in Formal Methods for Cryptographic Protocol Analysis, Proceedings of DISCEX 2000, IEEE Press, 237-250, 2000.
11. R. Milner, J. Parrow and D. Walker. A calculus of mobile processes, Part I and II. *Information and Computation*, 100(1):1–77, 1992.
12. U. Montanari and M. Pistore. Checking bisimilarity for finitary π-calculus. In *Proc. CONCUR'95*, LNCS 962. Springer Verlag, 1995.
13. U. Montanari and M. Pistore. History dependent verification for partial order systems. In *Partial Order Methods in Verification*, DIMACS Series, Vol. 29. American Mathematical Society, 1997.
14. U. Montanari, M. Pistore and D. Yankelevich. Efficient minimization up to location equivalence. In *Proc. ESOP'96*, LNCS 1058. Springer Verlag, 1996.
15. U. Montanari and M. Pistore. History-Dependent Automata. IRST Technical Report 0112-14, Istituto Trentino di Cultura, December 2001.
16. U. Montanari and M. Pistore. π-calculus, structured coalgebras and minimal HD-automata. In *Proc. MFCS'2000*, LNCS 1893, Springer Verlag, 2000.
17. M. Pistore and D. Sangiorgi, A partition refinement algorithm for the π-calculus. *Information and Computation* 164, 263-321, 2001.
18. R. Paige, R.E. Tarjan. Three partition refinement algorithms, SIAM Journal on Computing 16(6) 973–989, 1987.
19. J.J.M.M. Rutten. Universal coalgebra: a theory of systems. Technical Report CS-R9652, CWI, 1996. To appear in *Theoretical Computer Science*.
20. D. Turi, G.D. Plotkin, Towards a Mathematical Operational Semantics, In Proc. Logic in Computer Science 97, IEEE Press, 280-291, 1997.
21. J. Warell, Terminal Sequences for Accessible Endofunctors, In Proc. CMCS'99, ENCTS 19, 1999.