History Dependent Automata for Service Compatibility [⋆]

Vincenzo Ciancia[1], Gian Luigi Ferrari[1], Marco Pistore[2], and Emilio Tuosto[3]

[1] Department of Computer Science, University of Pisa
{ciancia,giangi}@di.unipi.it
[2] Center for Information Technology - IRST, Fondazione Bruno Kessler
pistore@fbk.eu
[3] Department of Computer Science, University of Leicester
et52@mcs.le.ac.uk

Abstract. We use *History Dependent Automata* (HD-automata) as a syntax-indepentend formalism to check compatibility of services at binding time in *Service-Oriented Computing.*
Informally speaking, service requests are modelled as pairs of HD-automata $\langle C_o, C_r \rangle$; C_r describes the (abstract) behaviour of the searched service and C_o the (abstract) behaviour guaranteed by the invoker. Symmetrically, service publication consists of a pair of HD-automata $\langle S_o, S_r \rangle$ such that S_o provides an (abstraction of) of the behaviour guaranteed by the service and S_r yields the requirement imposed to invokers. An invocation $\langle C_o, C_r \rangle$ matches a published interface $\langle S_o, S_r \rangle$ when C_o simulates S_r and S_o simulates C_r.

1 Introduction

Over the last few years nominal calculi have been envisaged as a suitable model for *Service-Oriented Computing* (SOC). As a matter of fact, *names* provide a uniform mechanism for abstracting a variety of different concepts like addresses, links, continuations, distributed objects, localities, causal dependencies, cryptographic keys and session identifiers. Also, the dynamicity issues usually arising in distributed computing (e.g., network reconfiguration, link mobility) can benefit from the sophisticated linguistic mechanisms of nominal calculi such as binding and scope extrusion. The π-calculus [12, 22] is a small but illustrative example of nominal calculus. Many of the concepts outlined above can be formally described and explained in terms of the π-calculus.

In the nineties, Montanari and Pistore [14, 15, 20] introduced *History Dependent automata* (HD-automata) as a "syntax-independent" automata based model amenable to represent the behaviour of a whole spectrum of formalisms that stress the role of names to refer to suitable semantics concepts. For instance, CCS with causality and localities and some dialects of the π-calculus have been

⋆ Research partially supported by the EU FP6-IST IP 16004 SENSORIA and by the UK project HIDEA4SOC.

semantically described by HD-automata. Indeed, different versions of HD-automata have been defined. The simplest version can easily be translated to finite-state automata, but possibly with a larger number of states.

A more sophisticated variant consists of HD-automata *with symmetries* where states are equipped with name symmetries (i.e., groups of name permutations). HD-automata with symmetries yield two main benefits: a faithful representation of linguistic mechanisms like scope extrusion and a re-adaptation of the partition refinement algorithm [18] for semantic minimisation of HD-automata. Basically, semantic minimisation provides a "garbage collection" mechanism of names so that states can be eliminated if their behaviour is mimicked by other states, possibly "re-using" their names with different meanings. Noteworthy, a theory based on coalgebras in a category of "named sets" can be developed for this kind of HD-automata, which extends the applicability of the approach to other nominal calculi and guarantees the existence of the minimal automaton within the same bisimilarity class [16, 6].

Also, the coalgebraic theory constitutes the formal basis upon which several verification toolkits have been defined and implemented. In fact, on the one hand, the front-end towards the π-calculus and the translation algorithm for the simplest version of HD-automata have been implemented in the HAL tool [4, 5], which relies on the JACK verification environment [1] for handling semantic verification via standard finite-state automata (e.g., model checking). And, on the other hand, the minimisation algorithm, naturally suggested by the coalgebraic framework, has been implemented in the Mihda toolkit [7, 8].

Finally, other variations of HD-automata have been defined by introducing different algebraic operations [9], and are based on a algebraic-coalgebraic theory [13]. Moreover, HD-automata have been considered with respect to bisimulation-like behavioural equivalences (e.g., π-calculus bisimulation [12] or fusion calculus hyperbisimulation [19]).

In this paper, we introduce a semantic framework for HD-automata based on *simulation* and propose to use it as a mechanism for dealing with semantic-based discovery of services within the service-oriented context. Our main goal is to define a foundational framework to express *behaviour*-based service discovery. Current standards for service discovery (i.e., UDDI and WSDL) provide purely syntactic techniques. As a consequence, composing services only on the basis of syntactic WSDL interfaces may lead to composite services that fall short in meeting their requirements.

In our approach, service descriptions are annotated with HD-automata abstracting the behaviour of the service. Informally speaking, service requests are modelled as pairs of HD-automata $\langle C_o, C_r \rangle$; C_r describes the (abstract) behaviour of the searched service and C_o the (abstract) behaviour guaranteed by the invoker. Symmetrically, service publication consists of a pair of HD-automata $\langle S_o, S_r \rangle$ where S_o provides an (abstraction of) of the behaviour guaranteed by the service and S_r yields the requirement imposed to invokers. Operationally, a service invocation $\langle C_o, C_r \rangle$ *matches* a published interface $\langle S_o, S_r \rangle$ when C_o simulates S_r and S_o simulates C_r. Hence, in our approach the operation of service discovery becomes a semantic-based operation: the service registry is searched for a service matching the semantic abstractions.

In this paper we report our preliminary results in the exploitation of HD-automata as intermediate language to represent semantic-based discovery of services.

Structure of the Paper. § 2 fixes the notations, revisits the main notions underlying HD-automata, and gives the definition of HD-automata. § 3 recasts the definition of *simulation* relation given in [20] in our context. § 4 gives an example applying the framework to the problem of semantical versioning of protocols. § 5 summarises our work, draws conclusions and sketches some research directions.

2 Background

This section collects the main notations and some basic concepts used throughout the paper.

Let X, Y, Z, $\ldots$ be sets. Then:

- $\mathcal{P}(X)$ (resp. $\mathcal{P}_{fin}(X)$) is the set of subsets (resp. finite subsets) of X;
- $[X \to Y]$ is the set of maps f with domain $\mathrm{dom}\, f \overset{\mathrm{def}}{=} X$ and codomain $\mathrm{cod}\, f \overset{\mathrm{def}}{=} Y$; $\mathrm{Im}\ f \overset{\mathrm{def}}{=} \{f(x) \in Y \mid x \in X\}$ is the image of f;
- $[X \overset{inj}{\to} Y]$ is the set of injective maps from X to Y;
- $\iota : X \hookrightarrow Y$ is the inclusion map from X to Y (implicitely assuming that $X \in \mathcal{P}(Y)$) and, if $f \in [Y \to Z]$ then $f|_X = f \circ \iota$ is the restriction of f to X;
- $Aut(X) = \{f \in [X \to X] \mid f \text{ bijective}\}$ is the set of *automorphisms* (or *permutations*) of X.

To avoid cumbersome parenthesis, $\overset{inj}{\to}$ has precedence over $\in$ and sometimes square brackets will be omitted from the denotation of functional domains.

We use P and R to range over (π-calculus) processes built on a countable set of *names* ω. Elements of $[\omega \to \omega]$ are name substitutions and $P\sigma$ denotes the agent obtained by applying the substitution $\sigma \in [\omega \to \omega]$ to P.

2.1 Automata as coalgebras

We will define HD-automata as *coalgebras* for a functor on the category of **NSet** (§ 2.2). To make the presentation more clear, we first summarise how classical automata can be specified as coalgebras for which very basic notions from category theory have to be introduced. (The interested reader is referred to, e.g., [21].)

Recall that a *category* is a collection of *objects* $a, b, \ldots$ and *morphisms* $f : a \to b$ from a to b ($\mathrm{dom}\, f \overset{\mathrm{def}}{=} a$ and $\mathrm{cod}\, f \overset{\mathrm{def}}{=} b$ resp. are the *domain* and *codomain* of f). A category is subject to the following axioms:

– if f and g are morphisms for which $\mathrm{cod}\, f = \mathrm{dom}\, g$, the *composition* of f and g, written $g \circ f$, is a morphism of the category and $\mathrm{dom}\, g \circ f = \mathrm{dom}\, f$ and $\mathrm{cod}\, g \circ f = \mathrm{cod}\, g$;
– morphism composition is associative: if f, g and h can be composed, $(h \circ g) \circ f = h \circ (g \circ f)$;
– for each object a there is the *identity morphism* $id_a : a \to a$; identities are such that $f \circ id_{\mathrm{dom}\, f} = f = id_{\mathrm{cod}\, f} \circ f$, for any morphism f.

A *functor* $\mathcal{F}$ *from* $\mathcal{A}$ *to* $\mathcal{B}$ (written $\mathcal{F} : \mathcal{A} \to \mathcal{B}$) trasforms objects and morphisms of the category $\mathcal{A}$ resp. into objects and morphisms of the category $\mathcal{B}$, preserving identities and composition. Formally, any object a in $\mathcal{A}$ is mapped to an object $\mathcal{F}a$ in $\mathcal{B}$ and any morphism $f : a \to b$ in $\mathcal{A}$ is sent to a morphism $\mathcal{F}f : \mathcal{F}a \to \mathcal{F}b$ in $\mathcal{B}$ such that:

$$\mathcal{F}id_a = id_{\mathcal{F}a}, \qquad \mathcal{F}(g \circ f) = (\mathcal{F}g) \circ (\mathcal{F}f).$$

For simplicity we limit ourselves to $\mathcal{S}et$, the category of sets and total functions with the usual function composition. The singleton set $\{\star\}$ is indicated as $\mathbf{1}$ and the disjoint union of $\mathbf{1}$ with a set X is denoted by $X + \mathbf{1}$.

Definition 1 ($\mathcal{F}$-Coalgebra). *Let $\mathcal{F} : \mathcal{S}et \to \mathcal{S}et$ be a fixed endofunctor on $\mathcal{S}et$ (i.e., a functor from $\mathcal{S}et$ to $\mathcal{S}et$). A pair (X, α) is a* coalgebra *for $\mathcal{F}$ (or $\mathcal{F}$-coalgebra) iff X is an object of $\mathcal{S}et$ (i.e., a set) and $\alpha : X \to \mathcal{F}X$ is an arrow of $\mathcal{S}et$ (i.e., a total function from X to $\mathcal{X}$).*

Given two $\mathcal{F}$-coalgebras (X, α) and (Y, β), a function $f : X \to Y$ in $\mathcal{S}et$ is an *$\mathcal{F}$-coalgebra (homo)morphism* iff

$$\begin{tikzcd} X \arrow[r, "f"] \arrow[d, "\alpha"'] & Y \arrow[d, "\beta"] \\ \mathcal{F}X \arrow[r, "\mathcal{F}f"'] & \mathcal{F}Y \end{tikzcd}$$

commutes, namely $\beta \circ f = \mathcal{F}f \circ \alpha$.

It is immediate to see that coalgebras for $\mathcal{F} = \mathcal{P}(L \times \_)$ coincide with transition systems labelled by L; in fact, if (X, α) is a $\mathcal{P}(L \times \_)$-coalgebra then X is the set of states and, for $x \in X$, $\alpha(x)$ are the outgoing transitions of x. *Vice versa*, given a L-labelled transition system T whose set of states is X one can define the coalgebra (X, α) by letting $\alpha : x \mapsto \{\langle l, y\rangle \mid \langle x, l, y\rangle$ is a transition in $T\}$ for each $x \in X$. Similarly, finitely branching L-labelled transition systems correspond to $\mathcal{P}_{fin}(L \times \_)$-coalgebras. Also, and more importantly, coalgebra morphisms enable the definition of *coalgebraic bisimulation* that nicely corresponds to the familiar notion of bisimulation in labelled transitions systems. As usual, we let $\pi_1 : X \times Y \to X$ and $\pi_2 : X \times Y \to Y$ be the projections on the cartesian product of X and Y.

Definition 2 (Coalgebraic bisimulation). *A* bisimulation *between two $\mathcal{F}$-coalgebras (X, α) and (Y, β) is a set $B \subseteq X \times Y$ such that there is $\theta : B \to \mathcal{F}B$*

making $\pi'_1 = \pi_1 \circ \iota$ *and* $\pi'_2 = \pi_2 \circ \iota$ *two homomorphism, where* $\iota : B \hookrightarrow X \times Y$*. In other words,*

$$\begin{tikzcd}
 & X \times Y \arrow[dl, "\pi_1"'] \arrow[dr, "\pi_2"] & \\
X \arrow[d, "\alpha"'] & B \arrow[u, hook] \arrow[l, "\pi'_1"'] \arrow[r, "\pi'_2"] \arrow[d, dotted, "\theta"] & Y \arrow[d, "\beta"] \\
\mathcal{F}X & \mathcal{F}B \arrow[l, "\mathcal{F}\pi'_1"] \arrow[r, "\mathcal{F}\pi'_2"'] & \mathcal{F}Y
\end{tikzcd}$$

commute, namely $\alpha \circ \pi'_1 = \mathcal{F}\pi'_1 \circ \theta$ *and* $\beta \circ \pi'_2 = \mathcal{F}\pi'_2 \circ \theta$*.*

The correspondence between coalgebra homomorphisms and bisimulations is made precise by the following theorem, that holds under mild conditions on the functor $\mathcal{F}$ (namely, preservation of *weak pullbacks*):

Theorem 1 ([21]). *Morphism* $f : X \to Y$ *is an homomorphisms between two* $\mathcal{F}$*-coalgebras* (X, α) *and* (Y, β) *iff* $\{(x, f(x)) \mid x \in X\}$ *is a bisimulation between* (X, α) *and* (Y, β)*.*

Similar results have been proved for HD-automata and in the next sections we recast the coalgebraic framework for HD-automata.

2.2 Named sets and named functions

The definition of HD-automata relies on the notion of *named sets* (to represent the states) and *named functions* (to represent transitions).

Definition 3 (Named set). *A* named set *consists of a triple* $\langle Q, \|\_\|, G\rangle$ *where*

- Q *is a set of* states*;*
- $\|\_\| : Q \to \mathcal{P}_{fin}(\omega)$ *maps each state* $q \in Q$ *to the set of names of* $\|q\|$*;*
- $G : Q \to \mathcal{P}(Aut(\omega))$ *maps each state* $q \in Q$ *to* $G(q)$ *which is a subgroup of* $Aut(\|q\|)$ *called* symmetry of q.

We let $L, M, N, \ldots$ range over named sets; states, set of names and symmetry maps of a named set N are written as Q_N, $\|\_\|_N$ and G_N, resp. (subscripts will be removed if clear from the context).

By Definition 3, each state $q \in Q_N$ of a named set N is equipped with a finite set of names $\|q\|_N$ (called *support*) together with a group of permutations over such names. Intuitively, q represents a set of states "using" names in $\|q\|_N$ (the names of q) that are "indistinguishable" (according to a suitable equality of states) under the permutations in $G_N(q)$.

Example 1. The set of π-calculus agents can be given a named set structure by taking as elements sets q of agents and setting $G(q)$ to be the symmetry made of all permutations in $Aut(\omega)$ that applied to agents in q yield a structurally congruent agent still in q. Namely, if $P, R \in q$ then $P \equiv R\rho$ for a $\rho \in G(q)$. Observe that all the agents in q have the same set of free names which is actually the support of q, $\|q\|$.

A key feature of HD-automata is that names do not have a global meaning. In fact, names are deemed *local* to states and transitions. This makes possible garbage collection of unused names which is usually absent in ordinary transition systems. For instance, in the ordinary semantics of the π-calculus, the agent $R(x) = (\nu y)\bar{x}y.R(y)$ reaches an infinite number of agents because all $R(z)$ with fresh z are different. Instead, in the HD-automata representation of $R(x)$ it is the "history" of computation that establishes the freshness of names; hence, all $R(z)$ with fresh z collapse on a single state. *Named functions* yield the "history" of computations.

Definition 4 (Named function). *A* named function F between two named sets N and M *is a pair of functions* $\langle h, \Sigma \rangle$ *such that*

- $h : Q_N \to Q_M$*;*
- $\Sigma : Q_N \to \mathcal{P}(\omega \overset{inj}{\to} \omega)$ *such that for all* $q \in Q_N$*,* $\Sigma(q) \in \mathcal{P}_{fin}(\|h(q)\|_M \overset{inj}{\to} \|q\|_N)$ *and, for all* $\sigma \in \Sigma(q)$

$$\sigma \circ G_M(h(q)) = \Sigma(q) \tag{1}$$

$$G_N(q) \circ \sigma \subseteq \Sigma_F(q). \tag{2}$$

We write $F : N \to M$ *to denote a named function from* N *to* M *and, if* $F = \langle h, \Sigma \rangle$*,* h_F *(resp.* Σ_F*) denotes* h *(resp.* Σ*).*

Condition (1) intuitively states that a function in Σ_F traces the history of names of q when mapped via h_F. Remarkably, Σ_F contains in general many injective functions: one for each permutation in the symmetry of $h_F(q)$. In other words, Σ_F is obtained by saturating an injective function with $G_M(q)$. This avoids the possibility to have two different mappings, that only differ for a permutation which is already in the symmetry of an element.

Condition (2) might look a bit obscure at a first glance, however it can be explained as follows: if we interpret σ as the representative of the history of names along a transition, condition (2) states that permutations in the symmetry of q respect such a history, namely they do not "represent" a transition which is not encompassed by $\Sigma(q)$.

We conclude this section by defining identity and composition of named functions.

Definition 5 (Identity and composition). *Let* L*,* M*,* N *be named sets. The* identity named function on N *is* $id_N = \langle id_{Q_N}, \lambda q.G_N(q) \rangle$*. If* $F : L \to M$ *and* $H : M \to N$*, the* composition of F and H *is* $\langle h_H \circ h_F, \Sigma \rangle$ *where*

$$\Sigma(q) = \{\sigma \circ \sigma' \mid \sigma \in \Sigma_F(q) \wedge \sigma' \in \Sigma_H(h_F(q))\}.$$

The composition of F *and* H *is denoted as* $H \circ F$*.*

Summing up, named sets and named functions form the category **NSet** [6, 3]. In [11], it is shown how **NSet** is categorially equivalent to the category of *nominal sets* [10], which is the same as algebras over the permutation signature [17].

2.3 HD-automata

We define HD-automata as labelled transition systems where a set of *local* names is associated to each state. Locality of names is a key aspect of HD-automata and, intuitively, asserts that identity of names used in a given state is not related to that of names used in other states. This, combined with symmetries associated to states, enables the minimisation of HD-automata [17, 3]. Roughly speaking, the symmetry of a state q specifies how names of q can be interchanged without affecting its observable behaviour. This also allows the size of the state space of HD-automata to be reduced yielding a bisimulation checking algorithm. Here we neglect some technical details in favour of a simpler presentation. Specifically, we do not consider the *normalisation* operation (cf. [3, 6]).

We fix the named set $L = \langle Q_L, \|\_\|_L, G_L \rangle$ of *labels* where

- Q_L is the *set* of labels (ranged over by l);
- $\|l\|_L \in \mathcal{P}_{fin}(\omega)$ are the names "exposed" in a transition labelled by l;
- for each label $l \in Q_L$, $G_L(l)$ is usually the trivial group $\{id_{\|l\|_L}\}$.

Example 2. A label representing the π-calculus output can be given by a label $\mathbf{out} \in Q_L$ for which $\|\mathbf{out}\|_L = \{x, y\}$, yielding the *subject* and the *object* names of the output. Notice that an output where subject and object coincide can be represented by a label $\mathbf{out1} \in Q_L$ such that $\|\mathbf{out1}\|_L$ is a singleton. Similarly, a *bound output* (where the object is a private name extruded to the environment) can be represented by a label $\mathbf{bout}$ having a single name for the subject.

HD-automata can be defined as coalgebras for the functor $\mathcal{T}$ (given below) that equips labelled transition systems with the notion of *local names* and *binding* emerging from the base category **NSet**.

Let $\overset{\frown}{N}$, the set of *(L-)transitions* on a named set N, be defined as

$$\overset{\frown}{N} = \left\{ \langle q, l, n, \pi, \vartheta \rangle \mid q \in Q_N, l \in Q_L, n \in \mathcal{P}_{fin}(\omega), \pi : \|l\|_L \overset{inj}{\to} n, \vartheta : \|q\|_N \overset{inj}{\to} n + \mathbf{1} \right\}.$$

Each transition consists of a destination state q, a label l, a set of names n (representing the observable names of the whole transition), and two injections π and ϑ. The former maps the observable names of the label l into n, while ϑ provides the history of the names of the destination state along the transition, by mapping them to n. Remarkably, Im $\vartheta \subseteq n + \mathbf{1}$ accounts for the generation of a fresh name. Specifically, if a name of q is mapped on $\star \in \mathbf{1}$ then it is fresh (for simplicity we assume that at most one fresh name can be generated).

Transitions on N form a named set $Tr[N] = \langle \overset{\frown}{N}, \|\_\|_{\overset{\frown}{N}}, G_{\overset{\frown}{N}} \rangle$ where

- $\|\langle q, l, n, \pi, \vartheta \rangle\|_{\overset{\frown}{N}} = n$;
- $G_{\overset{\frown}{N}}(\langle q, l, n, \pi, \vartheta \rangle) = \{\rho \in Aut(n) \mid \rho|_{\mathrm{Im}\ \pi} = id_{\mathrm{Im}\ \pi} \wedge \rho^* \circ \vartheta \circ G_N(q) = \vartheta \circ G_N(q)\}$, where $\rho^* \in Aut(n + \mathbf{1})$ is the map $\rho + id_{\mathbf{1}}$ (i.e., $\rho^*|_n = \rho$ and $\rho^*|_{\mathbf{1}} = id_{\mathbf{1}}$).

The symmetry of a transition is given by the permutations that preserves the mappings $\vartheta \circ G_N(q)$, that is the meaning of the names as given by the symmetry of the target state q (and the label l).

Proposition 1. *If N is named set, then $Tr[N]$ is a named set.*

The functor $\mathcal{T} : \mathbf{NSet} \to \mathbf{NSet}$ is specified by describing its action on objects (i.e., named sets) and morphisms (i.e., named functions) of $\mathbf{NSet}$. The action of the functor $\mathcal{T}$ on an object N is

$$\mathcal{T}N = \langle \mathcal{P}_{fin}(Q_{Tr[N]}), \|{-}\|_{\mathcal{T}N}, G_{\mathcal{T}N} \rangle$$

where the set of names of $T \in \mathcal{P}_{fin}(Q_{Tr[N]})$ is the union of the set of names of all transitions in T, and the symmetry of T is the set of permutations for which transitions in T "are preserved". Formally,

$$\|T\|_{\mathcal{T}N} = \bigcup_{\langle q,l,n,\pi,\vartheta\rangle \in T} n, \qquad G_{\mathcal{T}N}(T) = \bigcap_{t \in T} G_{\widehat{N}}(t).$$

The action of $\mathcal{T}F = \langle h, \Sigma \rangle$ on a named function $F : N \to M$ is given by

$$h(T) = \bigcup_{\langle q,l,n,\pi,\vartheta\rangle \in T} \big\{ \langle h_F(q), l, (\mathrm{Im}\ \vartheta \circ \sigma \cup \mathrm{Im}\ \pi) \setminus \mathbf{1}, \pi, \vartheta \circ \sigma \rangle \mid \sigma \in \Sigma_F(q) \big\}$$

$$\Sigma(T) = \iota \circ G_{\mathcal{T}N}(T), \text{ where } \iota : \|h(T)\|_M \hookrightarrow \|T\|_N$$

Namely, $\mathcal{T}$ maps transitions on N to transitions on M replacing q with $h_F(q)$ in each $\langle q, l, \mathcal{N}, \pi, \vartheta \rangle$ and relating names of $h_F(q)$ to names of q via $\Sigma_F(q)$. It is worth noticing that the names of the transitions T are a superset of the names $h(T)$ because they are the union of the images of $\vartheta \circ \sigma$ (and π) of transitions in $h(T)$. In fact, Im $\vartheta \subseteq n$ by definition of ϑ and the composition with the function σ can only restrict the image of $\vartheta \circ \sigma$ (which happens when some name of q is discarded by F).

Proposition 2. *If F is a named function, then $\mathcal{T}F$ is a named function.*

Definition 6. *An HD-automata is a named function $H : N \to \mathcal{T}N$, namely it is a coalgebra for the functor $\mathcal{T}$ on* $\mathbf{NSet}$.

3 Simulation for HD-automata

This section recasts the definition of the *simulation* relation originally presented in [20] (Definition 7.11, Chapter 7) in our context.

Definition 7 (HD-Simulation). *Let $H : N \to \mathcal{T}N$ and $K : M \to \mathcal{T}M$ be two HD-automata. A relation $\mathfrak{S} \subseteq Q_N \times Aut(\omega) \times Q_M$ is an* HD-simulation *iff whenever $(q, \delta, q') \in \mathfrak{S}$ for any $\langle q_1, l, n_1, \pi_1, \vartheta_1 \rangle \in h_H(q)$ there are $\langle q_2, l, n_2, \pi_2, \vartheta_2 \rangle \in h_K(q')$ and $\delta' \in Aut(\omega)$ such that*

$$\begin{tikzcd}
l \arrow[rrrrr, dash, "id"] \arrow[d, "\pi_1"] &&&&& l \arrow[d, dotted, "\pi_2"'] \\
n_1 \arrow[r, hook, "\iota_1"] & h_H(q_1)_{\mathcal{T}H} \arrow[r, Rightarrow, "\sigma_1\in\Sigma_H(q_1)"'] & q_1 \arrow[r, dash, "\delta"] \arrow[loop above, Rightarrow, "\rho_1\in G_N(q_1)"] & q_2 \arrow[loop above, dotted, Rightarrow, "\rho_2\in G_M(q_2)"] & h_K(q_2)_{\mathcal{T}K} \arrow[l, dotted, Rightarrow, "\sigma_2\in\Sigma_K(q_2)"] & n_2 \arrow[l, dotted, hook', "\iota_2"'] \\
q_1' \arrow[u, "\vartheta_1"] \arrow[rrrrr, dotted, dash, "\delta'"'] \arrow[loop right, Rightarrow, "\rho_1'\in G_N(q_1')"] &&&&& q_2' \arrow[u, dotted, "\vartheta_2"'] \arrow[loop left, dotted, Rightarrow, "\rho_2'\in G_M(q_2')"]
\end{tikzcd}$$

Fig. 1. HD-automata simulation

– *for all* $\rho_1 \in G_N(q_1)$*,* $\sigma_1 \in \Sigma_H(q_1)$ *and* $\rho_1' \in G_{\mathcal{T}N}(q_1')$ *there are* $\rho_2 \in G_M(q_2)$*,* $\sigma_2 \in \Sigma_K(q_2)$ *and* $\rho_2' \in G_{\mathcal{T}M}(q_2)$ *such that*
  1. $\delta \circ \rho_1 \circ \sigma_1|_{n_1} \circ \vartheta_1 \circ \rho_1' = \rho_2 \circ \sigma_2|_{n_2} \circ \vartheta_2 \circ \rho_1' \circ \delta'$ *on the names* x *of* q_1 *for which* $\vartheta_1(\rho_1'(x)) \notin \mathbf{1}$*;*
  2. $\vartheta_1(\rho_1'(x)) \in \mathbf{1} \iff \vartheta_2(\rho_2(\delta'(x))) \in \mathbf{1}$*;*
  3. $\delta \circ \rho_1 \circ \sigma_1|_{n_1} \circ \pi_1 = \rho_2 \circ \sigma_2|_{n_2} \circ \pi_2$*;*

– $(q_1', \delta', q_2') \in \mathfrak{S}$*.*

where δ *(resp.* δ'*) is used as map from the names of* q_1 *(resp.* q_1'*) to the names of* q_2 *(resp.* q_2'*) in the compositions above.*

Definition 7 can be explained using the diagram in Figure 1 where

– bijections are represented by lines,
– elements of sets of maps Σ are represented by double lines,
– universally (resp. existentially) quantified maps are represented by solid (resp. dotted) lines and
– all the arrows involving each of the states q_1, q_1', q_2 and q_2' are meant to be maps from/to the names of the state.

For instance, ρ_2 is a double dotted line because it is an existentially quantified permutation in the symmetry of q_2.

In Figure 1 the sub-diagram consisting of π_1, ϑ_1 and $\sigma_1|_{n_1} = \sigma_1 \circ \iota_1$ describes how the coalgebra H maps the names of the transition $\langle q_1', l, \pi, \vartheta_1 \rangle \in h_H(q_1)$ on the names of q_1 through the injective maps in $\Sigma_H(q_1)$ (and similarly for the sub-diagram consisting of π_2, ϑ_2 and $\sigma_2|_{n_2} = \sigma_2 \circ \iota_2$).

Intuitively, Definition 7 states that any transition from q_1 is matched by a transition from q_2. However, names of such transitions and their relationship with names of q_1 and q_2 are of concern. As a matter of fact, symmetries may yield several different representations of equivalent states and transitions. Therefore, the conditions on δ state that names of q_1 and q_2 must be related so that the simulation is independent of their symmetries. More precisely, for any possible representation ρ_1 of q_1 and meaning σ_1 of the names in the derivative of q_1 (in the coalgebra H) it is possible to find corresponding representation ρ_2 and σ_2 for q_2 (in the coalgebra K) in such a way that:

– the history of names in transitions are compatible;
– freshness of names is preserved;
– observed names are the same up to permutations of the symmetries.

Notice that the conditions above resp. correspond to conditions 1, 2 and 3 of Definition 7.

Given an HD-simulation $\mathfrak{S}$ and $\langle q, \delta, q' \rangle \in \mathfrak{S}$, Definition 7 demands δ to be an automorphism of ω. In practice, name correspondences δ can be maps in $[\|q\| \to \|q'\|]$ such that $G(\delta) \overset{\text{def}}{=} \{(x, \delta(x)) \mid x \in \|q\|\}$ is a *partial bijection* in $\|q\| \times \|q'\|$, namely for all $(x', y') \in G(\delta)$, $x = x' \iff y = y'$.

Proposition 3. *For each HD-simulation $\mathfrak{S}$ there is an HD-simulation $\mathfrak{S}'$ such that for all $\langle q, \delta, q' \rangle \in \mathfrak{S}$ there is $\langle q, \delta', q' \rangle \in \mathfrak{S}'$ such that $\delta' \in [\|q\| \to \|q'\|]$ is a partial bijection.*

Moreover, since each partial bijection on ω can be extended to an automorphism, the following proposition holds.

Proposition 4. *Let $\mathfrak{S}$ be a set of triples $\langle q, \delta, q' \rangle$ where q and q' are states of two HD-automata and $\delta \in [\|q\| \to \|q'\|]$ is a partial bijection. If $\mathfrak{S}$ satisfies the conditions of Definition 3 then*

$$\mathfrak{S}' \overset{\text{def}}{=} \{\langle q, \hat{\delta}, q' \rangle \mid \langle q, \delta, q' \rangle \in \mathfrak{S} \land \hat{\delta} \in Aut(\omega) \land \hat{\delta}|_{\|q\|} = \delta\}$$

is an HD-simulation.

4 A Motivating Example

This section gives an example illustrating the main features of our approach, namely *symmetrical* and behavioural service matching with explicit handling of names.

Typically, in a distributed setting such as SOC, services and invokers agree on the adopted communication protocol during their very first interaction, by selecting a version identifier that is uniquely associated to a known protocol. In other words, an exact matching of the identifiers would establish a sort of “contract” about the communication protocol, avoiding unexpected requests or replies.

However, distributed protocols commonly evolve from one version to another of a service, making the usage of version identifiers a very fragile mechanism. In our framework, version identifiers are replaced by *behaviours* and simulation is used to establish the matching. This allows new deployed versions of services to slightly deviate from communication protocols in certain cases (e.g., by adding functionality to certain stages).

4.1 The scenario

Consider a service offering a "forwarding" mechanism whereby invokers send an address a and a message m to be sent to a. The service sends m to a after checking some conditions that we neglect for simplicity. Also, suppose that the service handles *sessions* through *cookies*. For instance, at the beginning of the interaction, a unique cookie is assigned to invokers. (Note that session handling is a requirement imposed by the service to invokers, and it is a completely separate issue from the effective service offered.) At any time during the evolution of the interaction, the service may require to check the invoker's cookie. Let this version of the service be called `protocol-A`. A different version, called `protocol-B`, allows the service to *refresh* cookies (i.e., to send a fresh cookie which should replace the old one) at any stage of the protocol.

Since cookies are expected to be fresh (in practice, being randomly generated with a very low probability of collision), it is necessary to explicitly handle fresh resource generation. The absence of garbage collection of *old* session cookies would yield infinite state systems, since an unbound number of (unused) cookies should be maintained. Remarkably, since clients must record at least the *last* received cookie (which is always fresh), it is not possible to model either of the protocols without memory at all. Therefore HD-automata are a reasonable choice, being close to ordinary labelled transition systems, with the added benefit of name handling and garbage collection.

In order to make the presentation clearer, in the following:

- states and labels are written together with their set of local names; for instance, $\mathbf{q}\{x, y\}$ or $\mathbf{addr}\{a\}$ denote a state q with two names x and y or a label **addr** whose observed name is a;
- the HD-automata transition

$$\begin{tikzcd} \overset{q_0}{X} \arrow[r, "\mathbf{l}\ Z{,}\ \vartheta"] & \overset{q_1}{Y} \end{tikzcd}$$

  represents a transition from $q\ X$ to $q'\ Y$ with label $\mathbf{l}\ Z$ and mapping θ from the names Y of q' to those X of q such that π is the identity map on Z and ϑ behaves as the identity of $Y \cap X$; moreover, θ is omitted if it is the identity;
- we write

$$\begin{tikzcd} \overset{q_0}{X} \arrow[r, "\substack{\mathbf{l_1}\ Z_1{,}\ \theta_1/\ldots/\\ \mathbf{l_n}\ Z_n{,}\ \theta_n}"] & \overset{q_1}{Y} \end{tikzcd}$$

  for a set of transitions from $q\ X$ to $q\ Y$ with labels $\mathbf{l_1}\ Z_1, \vartheta_1$,..., $\mathbf{l_n}\ Z_n, \vartheta_n$.

4.2 Binding services using HD-automata simulation

We assume that a service publishes (together with its signature) a pair of HD-automata $\langle S_o, S_r\rangle$ built on top of a fixed named set of labels. Intuitively, S_o yields the (abstracted) behaviour of the service (its *offer*) and S_r is the requirement that the service imposes to invokers to allow them to bind and make invocations.

We show here how this approach works for the scenario discussed in § 4.1 for which we define the named set of labels $\langle Lab, \|\_\|, G\rangle$ as follows:

– Lab is the set $\{\mathbf{setC}, \mathbf{readC}, \mathbf{addr}, \mathbf{msg}, \mathbf{quit}\}$ (where **setC** and **readC** are for setting a new cookie or reading a new one, **addr** is for communicating an address, **msg** is for communicating a message and **quit** determines the end of the protocol);
– $\|\_\|$ maps **setC** and **quit** to the empty set while **addr** and **msg** are mapped to a singleton;
– as assumed for named sets of labels, G maps each label to the trivial symmetry containing only the identity.

The HD-automaton representing the service requirements is S_r below:

$$\begin{tikzcd}
\overset{sr_0}{\{a', m'\}} \arrow[r, "\mathbf{setC}\{\}{,}\vartheta_{c'}"] & \overset{sr_1}{\{a', m', c'\}} \arrow[loop above, "\mathbf{readC}\{c'\}/\ \mathbf{setC}\{\}{,}\vartheta_{c'}/\ \mathbf{addr}\{a'\}"] \arrow[r, "\mathbf{msg}\{m'\}"] & \overset{sr_2}{\{a', m', c'\}} \arrow[r, "\mathbf{quit}\{\}"] & \overset{sr_3}{\{\}}
\end{tikzcd}$$

where $\vartheta_{c'}$ maps c' to $\star$ and a' (resp. m') to a' (resp. m'). HD-automaton S_r formalises `protocol-B` service requirements and requires invokers to accept a cookie and to be ready to reset it (**setC**) or provide it (**readC**) at any time during the protocol before the message is sent. After sending the message, the invoker has to quit.

Remark 1. It is a simple observation that requirements for `protocol-A` can be obtained by removing the transition labelled $\mathbf{setC}\{\}$ from sr_1 in S_r.

The service offers the behaviour S_o below to invokers that fulfill S_r:

$$\begin{tikzcd}
\overset{so_0}{\{a_1, a_2, c'\}} \arrow[loop above, "\mathbf{setC}\{\}{,}\vartheta_{c'}"] \arrow[r, "\mathbf{addr}\{a_1\}/\ \mathbf{addr}\{a_2\}"] & \overset{so_1}{\{a_1, a_2, c'\}} \arrow[loop above, "\mathbf{setC}\{\}{,}\vartheta_{c'}/\ \mathbf{readC}\{c'\}/\ \mathbf{addr}\{a_1\}/\ \mathbf{addr}\{a_2\}"] \arrow[r, "\mathbf{msg}\{m'\}"] & \overset{so_2}{\{a_1, a_2, m', c'\}} \arrow[r, "\mathbf{quit}\{\}"] & \overset{so_3}{\{\}}
\end{tikzcd}$$

where the cookie can be refreshed any number of times before and after getting an (states so_0 and so_1). Once an address is sent, the service moves in state so_1 where cookies can be refreshed or required for checking an unbound number of times. In so_1 the service is also keen to accept addresses (either a_1 or a_2) before sending the message. Finally, if the message is sent, the service halts.

Remark 2. Noteworthy, so_1 could be replaced by a state so_1' equipped with the symmetry $\{id_{a_1,a_2}, (a_1\ a_2)\}$ (where $(a_1\ a_2)$ is the transposition of a_1 and a_2). However, for simplicity we prefer to stick with the current more explicit representation.

Remark 3. Again, removing the transition $\mathbf{setC}\{\}$ from so_1 in S_o yields the HD-automaton for `protocol-A`.

Symmetrically to service publication, service invocations have to specify a pair $\langle C_r, C_o\rangle$ of HD-automata so that C_r describes the behaviour required by the invoker to the service and C_o yields the offered guarantees. An invocation matching $\langle S_r, S_o\rangle$ is represented by $\langle C_o, C_r\rangle$ where C_o is:

$$\begin{tikzcd}
 & & \begin{array}{c} co_2 \\ \{a_1, a_2, m, c\} \end{array} \arrow[loop above, "\begin{array}{c}\mathbf{readC}\{c\}/\\ \mathbf{setC}\{\}, \vartheta_c/\\ \mathbf{addr}\{a_2\}\end{array}"] \arrow[dd, "\mathbf{msg}\{m\}"] & \\
\begin{array}{c} co_0 \\ \{a_1, a_2, m\} \end{array} \arrow[r, "\mathbf{setC}\{\}, \vartheta_{c'}"] & \begin{array}{c} co_1 \\ \{a_1, a_2, m, c\} \end{array} \arrow[loop below, "\begin{array}{c}\mathbf{readC}\{c\}/\\ \mathbf{setC}\{\}, \vartheta_c\end{array}"] \arrow[ur, "\mathbf{addr}\{a_1\}"] \arrow[dr, "\mathbf{msg}\{m\}"] & & \\
 & & \begin{array}{c} co_3 \\ \{a_1, a_2, m, c\} \end{array} \arrow[loop below, "\begin{array}{c}\mathbf{readC}\{c\}/\\ \mathbf{setC}\{\}, \vartheta_c\end{array}"] \arrow[r, "\mathbf{quit}\{\}"] & \begin{array}{c} co_4 \\ \{\} \end{array}
\end{tikzcd}$$

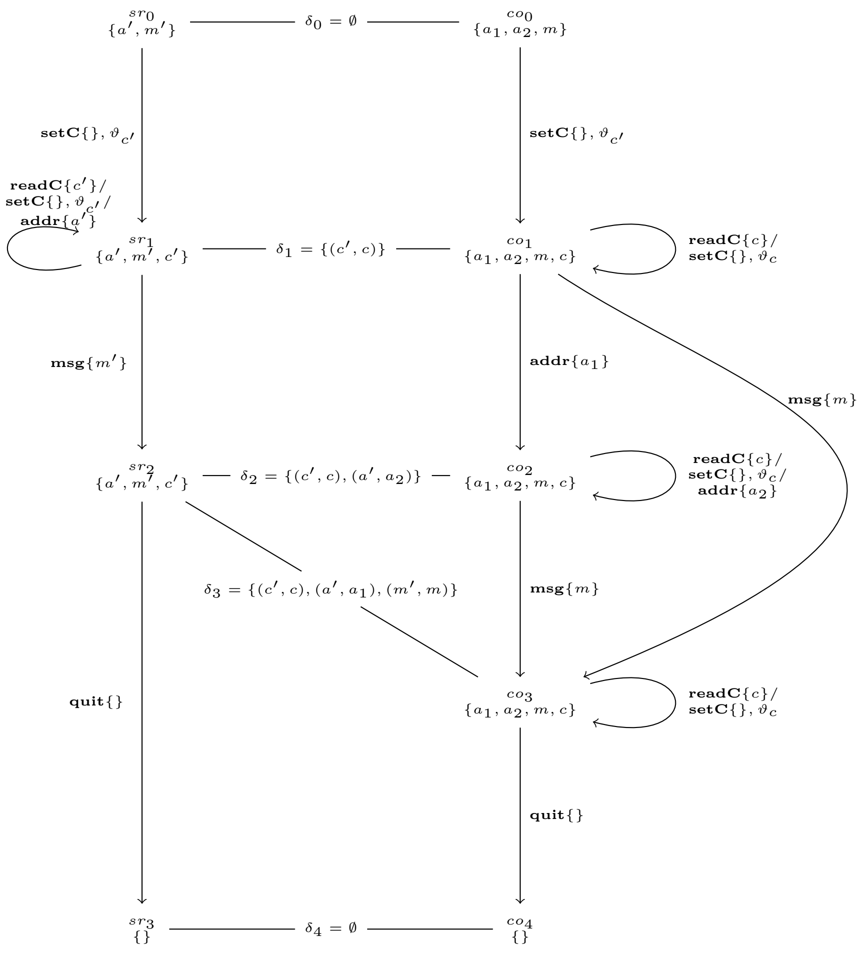
and C_r is

$$\begin{tikzcd}
\begin{array}{c} cr_0 \\ \{a_1, a_2\} \end{array} \arrow[r, "\mathbf{setC}\{\}, \vartheta_{c'}"] & \begin{array}{c} cr_1 \\ \{a_1, a_2, c\} \end{array} \arrow[r, "\mathbf{addr}\{a_1\}"] & \begin{array}{c} cr_2 \\ \{a_1, a_2, c\} \end{array} \arrow[r, "\mathbf{addr}\{a_2\}"] & \begin{array}{c} cdr_3 \\ \{a_1, a_2, c, m\} \end{array} \arrow[r, "\mathbf{msg}\{m\}"] & \begin{array}{c} cr_4 \\ \{a_1, a_2, c, m\} \end{array}
\end{tikzcd}$$

C_r simply requires to the service the capacity of executing the sequence of transitions setting the cookie, receiving the addresses and forwarding. (Notice that the invocation does not require the service to stop.)

C_o guarantees that the client accepts a request to set the cookie, then in each state is capable to provide the previously set cookie upon request, or to refresh it, thus respecting the protocol imposed by the service provider.

While it is immediate to see that S_o simulates C_r, it is less obvious that C_o simulates S_r. To show this, we build an explicit simulation between S_r and C_o represented in the following figure.



sr_0 $\{a', m'\}$
$\delta_0 = \emptyset$
co_0 $\{a_1, a_2, m\}$
setC$\{\}, \vartheta_{c'}$
setC$\{\}, \vartheta_{c'}$
readC$\{c'\}$/ **setC**$\{\}, \vartheta_{c'}$/ **addr**$\{a'\}$
sr_1 $\{a', m', c'\}$
$\delta_1 = \{(c', c)\}$
co_1 $\{a_1, a_2, m, c\}$
readC$\{c\}$/ **setC**$\{\}, \vartheta_c$
msg$\{m'\}$
addr$\{a_1\}$
msg$\{m\}$
sr_2 $\{a', m', c'\}$
$\delta_2 = \{(c', c), (a', a_2)\}$
co_2 $\{a_1, a_2, m, c\}$
readC$\{c\}$/ **setC**$\{\}, \vartheta_c$/ **addr**$\{a_2\}$
$\delta_3 = \{(c', c), (a', a_1), (m', m)\}$
msg$\{m\}$
quit$\{\}$
co_3 $\{a_1, a_2, m, c\}$
readC$\{c\}$/ **setC**$\{\}, \vartheta_c$
quit$\{\}$
sr_3 $\{\}$
$\delta_4 = \emptyset$
co_4 $\{\}$

A simple check shows that

$$\mathfrak{S} = \{\langle sr_0, \delta_0, co_0\rangle, \langle sr_1, \delta_1, co_1\rangle, \langle sr_2, \delta_2, co_2\rangle, \langle sr_2, \delta_3, co_3\rangle, \langle sr_3, \delta_4, co_4\rangle\}$$

yields an HD-simulation (by Proposition 4).

5 Conclusions and Future Work

We have introduced the foundations of a notion of *behavioural* matching of services, that keeps in account resource generation in a *finitistic* way employing HD-automata. Our framework inherits the algorithmic properties of HD-automata and it can support effective usages in design environments for SOC.

As a a case study here we considered the versioning problem of protocols. In the usual approach, versioning relies on version identifiers. Local deviations to a protocol, hence, either lead to incompatibility between clients and servers, or to servers mimicking a version identifier without giving real guarantees on the fact that the protocol is respected. We tackled this problem exploiting the semantic HD-automata machinery for the fresh generation of names.

A pair of matching interfaces (i.e., $\langle C_o, C_r \rangle$ and $\langle S_o, S_r \rangle$ where C_o simulates S_r and S_o simulates C_r) determines a contract. A possible research direction concerns the synthesis of a *monitor* out of a contract so that the fulfillment of the contract can be enforced during the execution of the communication protocol. This is an interesting research direction as, during the execution of the communication protocol, only traces can be observed (while at service binding time the interfaces of two services can be represented by abstractions of their interactive behaviours). We argue that a monitor can be automatically synthesised from contracts.

Laudatio

HD-automata appeared for the first time in CONCUR'95, in an article by Ugo Montanari and the third author of this paper [14]. At that time, HD-automata where called π-automata, and they where simply seen as an efficient structure in an algorithm for detecting names that where syntactically present in π-calculus agents but that did not play any semantic role.

Since then, HD-automata have evolved into a reference model for nominal calculi, both from a theoretical and from a practical point of view. HD-automata can be defined by extending to structures with names several frameworks initially proposed for classical automata, ranging from the the *categorical* setting of Nielsen and Winskel (see [20]), to Rutten's *coalgebraic* setting (see this paper, [3] and the recent work [2]), to the *bialgebraic* setting of Turi and Plotkin (see [11, 17]). The semantic framework of HD-automata has been used to model and analyse concepts such as locality, causality, link mobility and cryptography [5]; more recently, they have also been used to capture concepts typical of SOC, as shown in this paper. Moreover, HD-automata have been exploited as the formal basis for verification toolkits such as HAL and Mihda [4, 7].

Along the years, the research on HD-automata has involved several scientists, including PhD students of Ugo, of whom we would like to remember Gioia Ristori, who will unfortunately not be able to celebrate Ugo's 65th birthday. If HD-automata have become a reference model for nominal calculi, it is thanks to the work of all these scientists. However, it is also and primarily thanks to Ugo, to his intuition that HD-automata were much more than the efficient data structure of CONCUR'95, to his idea that being able to manage names in a syntax-independent way is the key to understand and analyse nominal calculi, and to his capability to guide the research and to prove that this intuition was true.

Bibliography

[1] Amar Bouali, Stefania Gnesi, and Salvatore Larosa. The Integration Project for the JACK Environment. In *Bulletin of the EATCS*, volume 54, pages 207–223. 1994.

[2] Vincenzo Ciancia and Ugo Montanari. A name abstraction functor for named sets. In *Coalgebraic Methods in Computer Science 2008*, 2008. to appear.

[3] Gian Luigi Ferrari, Ugo Montanari, and Emilio Tuosto. Coalgebraic minimization of hd-automata for the pi-calculus using polymorphic types. *Theoretical Computer Science*, 331(2-3):325–365, 2005.

[4] Gianluigi Ferrari, Giovanni Ferro, Stefania Gnesi, Ugo Montanari, Marco Pistore, and Gioia Ristori. An Automata Based Verification Environment for Mobile Processes. In Ed Brinksma, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1217 of *Lecture Notes in Computer Science*, pages 275–289. Springer-Verlag, April 1997.

[5] Gianluigi Ferrari, Stefania Gnesi, Ugo Montanari, and Marco Pistore. A Model Checking Verification Environment for Mobile Processes. *ACM Transactions on Software Engineering and Methodology*, 12(4):440–473, 2003.

[6] Gianluigi Ferrari, Ugo Montanari, and Marco Pistore. Minimizing Transition Systems for Name Passing Calculi: A Co-algebraic Formulation. In Mogens Nielsen and Uffe Engberg, editors, *FOSSACS 2002*, volume 2303 of *Lecture Notes in Computer Science*, pages 129–143. Springer-Verlag, 2002.

[7] Gianluigi Ferrari, Ugo Montanari, and Emilio Tuosto. From Co-algebraic Specifications to Implementation: The Mihda toolkit. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *Formal Methods for Components and Objects: First International Symposium, FMCO*, volume 2852 of *Lecture Notes in Computer Science*, pages 319 – 338, Leiden (Netherlands), November 2002. Springer-Verlag.

[8] Gianluigi Ferrari, Ugo Montanari, and Emilio Tuosto. Coalgebraic Minimisation of HD-automata for the π-Calculus in a Polymorphic λ-Calculus. *Theoretical Computer Science*, 331:325–365, 2005.

[9] Gianluigi Ferrari, Ugo Montanari, Emilio Tuosto, Björn Victor, and Kidane Yemane. Modelling and Minimising the Fusion Calculus Using HD-Automata. In José Luiz Fiadeiro, Neil Harman, Markus Roggenbach, and Jan Rutten, editors, *Algebra and Coalgebra in Computer Science*, volume 3629 of *Lecture Notes in Computer Science*, pages 142 – 156. Springer-Verlag, 2005.

[10] Murdoch Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13(3-5):341–363, 2002.

[11] Fabio Gadducci, Marino Miculan, and Ugo Montanari. About permutation algebras, (pre)sheaves and named sets. *Higher-Order and Symbolic Computation*, 19(2-3):283–304, 2006.

[12] Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, I and II. *Information and Computation*, 100(1):1–40,41–77, September 1992.

[13] Ugo Montanari and Marzia Buscemi. A First Order Coalgebraic Model of π-Calculus Early Observational Equivalence. In Luboš Brim, Petr Jančar, Mojmír Křetinský, and Antonín Kučera, editors, *International Conference in Concurrency Theory*, volume 2421 of *Lecture Notes in Computer Science*, pages 449–465. Springer-Verlag, August 2002.

[14] Ugo Montanari and Marco Pistore. Checking Bisimilarity for Finitary π-Calculus. In Insup Lee and Scott A. Smolka, editors, *International Conference in Concurrency Theory*, volume 962 of *Lecture Notes in Computer Science*, pages 42–56, Philadelphia, PA, USA, August 1995. Springer-Verlag.

[15] Ugo Montanari and Marco Pistore. History Dependent Automata. Technical report, Dipartimento di Informatica, Università di Pisa, 1998. TR-11-98.

[16] Ugo Montanari and Marco Pistore. π-Calculus, Structured Coalgebras, and Minimal HD-Automata. In Mogens Nielsen and Branislav Roman, editors, *Mathematical Foundations of Computer Science*, volume 1983 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000. An extended version will be published on Theoretical Computer Science.

[17] Ugo Montanari and Marco Pistore. Structured coalgebras and minimal hd-automata for the π-calculus. *Theoretical Computer Science*, 340:539–576, 2005.

[18] Robert Paige and Robert Tarjan. Three Partition Refinement Algorithms. *SIAM Journal on Computing*, 16(6):973–989, December 1987.

[19] Joachim Parrow and Bjorn Victor. The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes. In *Annual Symposium on Logic in Computer Science*. IEEE Computer Society, 1998.

[20] Marco Pistore. *History Dependent Automata*. PhD thesis, Università di Pisa, Dipartimento di Informatica, 1999. available at University of Pisa as PhD. Thesis TD-5/99.

[21] Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, October 2000.

[22] Davide Sangiorgi and David Walker. *The π-Calculus: a Theory of Mobile Processes*. Cambridge University Press, 2002.