Electronic Notes in Theoretical Computer Science 10 (1998)
URL: http://www.elsevier.nl/locate/entcs/volume10.html 19 pages

# An Introduction to History Dependent Automata [1]

Ugo Montanari and Marco Pistore

*Computer Science Department*
*University of Pisa*
*Corso Italia 40, 56100 Pisa, Italy*
{ugo,pistore}@di.unipi.it

**Abstract**

*Automata* (or *labeled transition systems*) are widely used as operational models in the field of process description languages like CCS [13]. There are however classes of formalisms that are not modelled adequately by the automata. This is the case, for instance, of the $\pi$-calculus [15,14], an extension of CCS where channels can be used as values in the communications and new channels can be created dynamically. Due to the necessity to represent the creation of new channels, infinite automata are obtained in this case also for very simple agents and a non-standard definition of bisimulation is required.

In this paper we present an enhanced version of automata, called *history dependent automata*, that are adequate to represent the operational semantics of $\pi$-calculus and of other *history dependent formalisms*. We also define a bisimulation equivalence on history dependent automata, that captures $\pi$-calculus bisimulation. The results presented here are discussed in more detail in [21].

## 1 Introduction

In the context of process algebras (e.g., Milner's CCS [13]), *automata* (or *labeled transition systems*) are often used as operational models. They allow for a simple representation of process behavior and many concepts and theoretical results for these process algebras are independent from the particular syntax of the languages, and can be formulated directly on automata. In particular, this is true for the behavioral equivalences and preorders which have been

[1] Research supported in part by CNR Integrated Project *Metodi e Strumenti per la Progettazione e la Verifica di Sistemi Eterogenei Connessi mediante Reti di Comunicazione* and Esprit Working Group *CONFER2*.

defined for these languages, like bisimulation equivalence [13,22]: in fact they take into account just the labeled actions an agent can perform.

Automata are also important from an algorithmic point of view: efficient and practical techniques and tools for verification [9,12] have been developed for *finite-state* automata. Finite state verification is successful here, differently than in ordinary programming, since the control part and the data part of protocols and hardware components can be often cleanly separated, and the control part is usually both quite complex and finite state.

There are classes of process description languages, however, whose operational semantics is not described in a satisfactory way by ordinary automata. A paradigmatic example is provided by $\pi$-calculus [15,14]. This calculus can be considered as a foundational calculus for concurrent functional languages, as $\lambda$-calculus for sequential functional languages. In $\pi$-calculus channel names can be used as messages in the communications, thus allowing for a dynamic reconfiguration of process acquaintances. More importantly, $\pi$-calculus names can model objects (in the sense of object oriented programming [24]) and name sending thus models higher order communication [23]. New channels between the process and the environment can be created at run-time and referred to in subsequent communications.

The operational semantics of $\pi$-calculus is given via a labeled transition system. This is not completely adequate to deal with the peculiar features of the calculus and complications arise in the representation of the creation of new channels. Consider process $p = (\nu y)\, \bar{x}y.q$; it communicates name $y$ on channel $x$ and then behaves like $q$. Channel $y$ is initially a local, restricted channel for process $p$, however the restriction is removed when the communication takes place, since it makes name $y$ known also outside the process. This communication represents the creation of a new channel. In the ordinary semantics of the $\pi$-calculus it is modelled by means of an infinite bunch of transitions of the form $p \xrightarrow{\bar{x}(w)} q\{w/y\}$, where $w$ is any name that is not already in use in $p$. This way to represent the creation of new names has some disadvantages: first of all, also very simple $\pi$-calculus agents, like $p$, give rise to infinite-state and infinite-branching transition systems. Moreover, equivalent processes do not necessarily have the same sets of channel names; so, there are processes $q$ equivalent to $p$ which cannot use $y$ as the name for the newly created channel. Special rules are hence needed in the definition of bisimulation, which is not the standard one for transition systems, and, as a consequence, standard theories and algorithms do not apply to $\pi$-calculus.

This is a general problem for the class of *history-dependent calculi.* A calculus is history dependent if the observations labeling the transitions of an agent may refer to informations — names in the case of $\pi$-calculus — generated in previous transitions of the agent.

In [21] *history-dependent automata* (*HD-automata* in brief) are proposed as a general model for history-dependent calculi. As ordinary automata, they are composed of states and of transitions between states. To deal with the peculiar problems of history-dependent calculi, however, states and transition are enriched with sets of local names: in particular, each transition can refer to the names associated to its source state but can also generate new names, which can then appear in the destination state. In this manner, the names are not global and static, as in ordinary labeled transition systems, but they are explicitly represented within states and transitions and can be dynamically created.

This permits to represent adequately the behavior of history-dependent processes. In particular, $\pi$-calculus agents can be translated into HD-automata and a first sign of the adequacy of HD-automata for dealing with $\pi$-calculus is that a large class of *finitary* $\pi$-calculus agents can be represented by finite-state HD-automata.

In [21] a general definition of bisimulation for HD-automata is also given. An important result is that this general bisimulation equates the HD-automata obtained from two $\pi$-calculus agents if and only if the agents are bisimilar according to the ordinary (strong, early) $\pi$-calculus bisimilarity relation.

These results do not hold only for the $\pi$-calculus: similar mappings exist also for other history-dependent calculi. In previous papers [20,18,19] we defined mappings to HD-automata for CCS with localities [4], for CCS with causality [7,6,11], and, to consider an example outside the field of process algebras, for the history-preserving semantics of Petri nets [2].

Papers [20,18,19] introduce the applications of HD-automata without resorting to categories. Report [21] defines HD-automata and HD-bisimulation both in a set theoretical style and following the uniform categorical approach of [10] based on spans of open maps.

In this paper we summarize some of the results of [21]. In particular, we define HD-automata in a categorical framework, by exploiting a classical categorical definition of ordinary automata. We also show that $\pi$-calculus agents can be translated into HD-automata. Finally, we introduce HD-bisimulation by applying to HD-automata the approach of open maps. We refer to [21] for the proof of the results presented here and for a deeper study of the properties of HD-automata.

## 2 The $\pi$-calculus

The $\pi$-calculus [15,14] is an extension of CCS in which channel names can be used as values in the communications, i.e., channels are first-order values. This possibility of communicating names gives to the $\pi$-calculus a richer expressive power that CCS: in fact it allows to generate dynamically new channels and to change the interconnection structure of the processes. The $\pi$-calculus has been successfully used to model object oriented languages [24], and also higher-order communications can be easily encoded in the $\pi$-calculus [23], thus allowing for code migration.

Many versions of $\pi$-calculus have appeared in the literature. The $\pi$-calculus we present here is *early* and *monadic*; it was first introduced in [16], but we present a slightly simplified version, following in part the style proposed in [23,14] for the polyadic $\pi$-calculus.

Let $\mathfrak{N}$ be an infinite, denumerable set of *names*, ranged over by $a, \ldots, z$, and let $Var$ be a finite set of *agent identifiers*, denoted by $A, B, \ldots$; the $\pi$-calculus *agents*, ranged over by $p, q, \ldots$, are defined by the syntax

$$p \ ::= \ \mathbf{0} \ \Big| \ \pi.p \ \Big| \ p|p \ \Big| \ p{+}p \ \Big| \ (\nu x)\, p \ \Big| \ [x{=}y]p \ \Big| \ A(x_1, \ldots, x_n)$$

where the *prefixes* $\pi$ are defined by the syntax

$$\pi \ ::= \ \tau \ \Big| \ \bar{x}y \ \Big| \ x(y).$$

The occurrences of $y$ in $x(y).p$ and $(\nu y)\, p$ are bound; *free names* of agent $p$ are defined as usual and we denote them with $\mathtt{fn}(p)$. For each identifier $A$ there is a definition $A(y_1, \ldots, y_n) \overset{\text{def}}{=} p_A$ (with $y_i$ all distinct and $\mathtt{fn}(p_A) \subseteq \{y_1, \ldots, y_n\}$); we assume that, whenever $A$ is used, its arity $n$ is respected. Finally we require that each agent identifier in $p_A$ is in the scope of a prefix (guarded recursion).

If $\sigma : \mathfrak{N} \to \mathfrak{N}$, we denote with $p\sigma$ the agent $p$ whose free names have been replaced according to substitution $\sigma$ (possibly with changes in the bound names); we denote with $\{y_1/x_1 \cdots y_n/x_n\}$ the substitution that maps $x_i$ into $y_i$ for $i = 1, \ldots, n$ and which is the identity on the other names.

Notice that, with some abuse of notation, we can see substitution $\sigma$ in $p\sigma$ as a function on $\mathtt{fn}(p)$ rather than on $\mathfrak{N}$; in fact, $p\sigma$ and $p\sigma'$ coincide whenever $\sigma$ and $\sigma'$ coincide on $\mathtt{fn}(p)$. So, we say that substitution $\sigma$ is injective for $p$ if $\sigma : \mathtt{fn}(p) \to \mathfrak{N}$ is an injective function. We also say that agents $p$ and $q$ differ for a bijective substitution if there exists some bijective function $\sigma : \mathtt{fn}(p) \to \mathtt{fn}(q)$ such that $q = p\sigma$.

We define $\pi$-calculus agents up to a *structural congruence* $\equiv$, in the style of the Chemical Abstract Machine [1]. This structural congruence allows to identify all the agents which represent essentially the same system and which differ just for syntactical details; moreover it simplifies the presentation of the operational semantics. The structural congruence $\equiv$ is the smallest congruence that respects the following rules.

**(alpha)** $(\nu x)\, p \equiv (\nu y)\, (p\{y/x\})$ if $y \notin \mathtt{fn}(p)$

**(sum)** $p{+}\mathbf{0} \equiv p \quad p{+}q \equiv q{+}p \quad p{+}(q{+}r) \equiv (p{+}q){+}r$

**(par)** $p|\mathbf{0} \equiv p \quad p|q \equiv q|p \quad p|(q|r) \equiv (p|q)|r$

**(res)** $(\nu x)\, \mathbf{0} \equiv \mathbf{0} \quad (\nu x)\, (\nu y)\, p \equiv (\nu y)\, (\nu x)\, p$

$(\nu x)\, (p|q) \equiv p|(\nu x)\, q$ if $x \notin \mathtt{fn}(p)$

**(match)** $[x = x]p \equiv p \quad [x = y]\mathbf{0} \equiv \mathbf{0}$

By exploiting the structural congruence $\equiv$, each $\pi$-calculus agent can be seen as a set of *sequential processes* that act in parallel, sharing a set of channels, some of which are global (unrestricted) whereas some other are local

$$\tau.p \xrightarrow{\tau} p \qquad \bar{x}y.p \xrightarrow{\bar{x}y} p \qquad x(y).p \xrightarrow{xz} p\{z/y\}$$

$$\frac{p_1 \xrightarrow{\alpha} p'}{p_1{+}p_2 \xrightarrow{\alpha} p'} \qquad \frac{p_1 \xrightarrow{\alpha} p_1'}{p_1|p_2 \xrightarrow{\alpha} p_1'|p_2} \text{ if } \mathtt{bn}(\alpha) \cap \mathtt{fn}(p_2) = \emptyset$$

$$\frac{p_1 \xrightarrow{\bar{x}y} p_1' \quad p_2 \xrightarrow{xy} p_2'}{p_1|p_2 \xrightarrow{\tau} p_1'|p_2'} \qquad \frac{p_1 \xrightarrow{\bar{x}(y)} p_1' \quad p_2 \xrightarrow{xy} p_2'}{p_1|p_2 \xrightarrow{\tau} (\nu y)\,(p_1'|p_2')} \text{ if } y \notin \mathtt{fn}(p_2)$$

$$\frac{p \xrightarrow{\alpha} p'}{(\nu x)\,p \xrightarrow{\alpha} (\nu x)\,p'} \text{ if } x \notin \mathtt{n}(\alpha) \qquad \frac{p \xrightarrow{\bar{x}y} p'}{(\nu y)\,p \xrightarrow{\bar{x}(z)} p'\{z/y\}} \text{ if } x \neq y, z \notin \mathtt{fn}((\nu y)\,p)$$

$$\frac{p_A\{y_1/x_1 \cdots y_n/x_n\} \xrightarrow{\alpha} p'}{A(y_1,\ldots,y_n) \xrightarrow{\alpha} p'} \text{ if } A(x_1,\ldots,x_n) \overset{\text{def}}{=} p_A$$

Table 1
Early operational semantics.

(restricted). Each sequential process is a term of the form

$$s \ ::= \ \pi.p \ \Big| \ p{+}p \ \Big| \ A(x_1,\ldots,x_n)$$

that can be considered as a "program" describing all the possible behaviors of the sequential process. These sequential processes are then connected by means of the operators of parallel composition and restriction, that allow to describe the structure of the system in which the processes act.

The *actions* an agent can perform are defined by the syntax

$$\alpha \ ::= \ \tau \ \Big| \ xy \ \Big| \ \bar{x}y \ \Big| \ \bar{x}(z)$$

and are called respectively *synchronization*, *input*, *free output* and *bound output* actions; $x$ and $y$ are free names of $\alpha$ ($\mathtt{fn}(\alpha)$), whereas $z$ is a bound name ($\mathtt{bn}(\alpha)$); moreover $\mathtt{n}(\alpha) = \mathtt{fn}(\alpha) \cup \mathtt{bn}(\alpha)$. Name $x$ is called the *subject* and $y$ or $z$ the *object* of the action.

The transitions for the *early operational semantics* are defined by the axiom schemata and the inference rules of Table 1.

Some comments on the syntax and on the operational semantics of $\pi$-calculus are now in order. The syntax of $\pi$-calculus is similar to that of CCS: the most important difference is in the prefixes. The *output* prefix $\bar{x}y.p$ specifies not just the channel $x$ for the communication, but also the value $y$ that is sent on $x$; in the input prefixes $x(y).p$, name $x$ represents still the channel, whereas $y$ represents a formal variable in $p$, that is instantiated by the effectively received value when the input transition takes place.

The matching $[x{=}y]p$ represents a guard for agent $p$: agent $p$ can act only if $x$ and $y$ coincide; this behavior is obtained by exploiting the structural congruence: in fact $[x{=}x]p \equiv p$; no transition can be derived from $[x{=}y]p$ if $x \neq y$.

Notice that, in the case of the $\pi$-calculus, the actions a process can perform are different from the prefixes. This happens due to the input and to the bound output. In the case of the input, the prefix has the form $x(y)$, while the action has the form $xz$; in fact, $y$ represent a formal variable, whereas $z$ is the effectively received value[2]. The bound output transitions are specific of the $\pi$-calculus; they represent the communication of a name that was previously restricted, i.e., it corresponds to the generation of a new channel between the agent and the environment.

Now we present the definition of the early bisimulation for the $\pi$-calculus.

**Definition 2.1 (early bisimulation)** A relation $\mathcal{R}$ over agents is an *early simulation* if whenever $p \mathrel{\mathcal{R}} q$ then:

for each $p \xrightarrow{\alpha} p'$ with $\mathtt{bn}(\alpha) \cap \mathtt{fn}(p,q) = \emptyset$ there is some $q \xrightarrow{\alpha} q'$ such that $p' \mathrel{\mathcal{R}} q'$.

A relation $\mathcal{R}$ is an *early bisimulation* if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are early simulations. Two agents $p$ and $q$ are *early bisimilar*, written $p \sim_\pi q$, if $p \mathrel{\mathcal{R}} q$ for some early bisimulation $\mathcal{R}$.

As for CCS-like calculi, a labeled transition system is used to give an operational semantics to the $\pi$-calculus. However, this way to present the operational semantics has some disadvantages. For instance, an infinite number of transitions correspond even to very simple agents, like $p = x(y).\bar{y}z.\mathbf{0}$: in fact, this agent can perform an infinite number of different input transitions $p \xrightarrow{xw} \bar{w}z.\mathbf{0}$, corresponding to all the possible choices of $w \in \mathfrak{N}$. It is clear that, except for $x$ and $z$, which are the free names of $p$, all the other names are indistinguishable as input values for the future behavior of $p$. However, this fact is not reflected in the operational semantics.

Also consider process $q = (\nu y)\,\bar{x}y.y(z).\mathbf{0}$. It is able to generate a new channel by communicating name $y$ in a bound output. The creation of a new name is represented in the transition system by means of an infinite bunch of transitions $q \xrightarrow{\bar{x}(w)} w(z).\mathbf{0}$, where, in this case, $w$ is any name different from $x$: the creation of a new channel is modelled by using all the names which are not already in use to represent it. As a consequence, the definition of bisimulation is not the ordinary one: in general two bisimilar process can have different sets free names, and the clause "$\mathtt{bn}(\alpha) \cap \mathtt{fn}(p,q) = \emptyset$" has to be added in Definition 2.1 to deal with those bound output transitions which use a name that is used only in one of the two processes. The presence of this clause makes it difficult to reuse standard theory and algorithms for bisimulation on the $\pi$-calculus — see for instance [5].

## 3 History-dependent automata

As explained in the Introduction, ordinary automata are insufficient to deal with history-dependent calculi. To address this problem, in this section we

[2] This is not true in all the versions of the $\pi$-calculus; in the case of the *late* and *open* versions, for instance, also the input actions have formal variables rather than values.

describe a richer structure, the *history-dependent automata* (*HD-automata* in brief), which are obtained by allowing names to appear explicitly in states, transitions and labels. As we will see, it is convenient to assume that the names which appear in a state, a transition or a label of a HD-automaton are *local* names and do not have a global identity. In this way, for instance, a single state of the HD-automaton can be used to represent all the states of a system that differ just for a bijective renaming. In this way, however, each transition is required to represent explicitly the correspondences between the names of source, target and label.

In this section we show that HD-automata can be defined in a categorical framework by extending the classical categorical definition of ordinary automata.

An ordinary automaton can be defined as a diagram

$$\begin{tikzcd} L & T \arrow[l, "l"'] \arrow[r, bend left, "s"] \arrow[r, bend right, "d"'] & Q & \{*\} \arrow[l, "i"'] \end{tikzcd}$$

in the category **Set** of sets. Sets $Q$, $T$ and $L$ represent respectively the *states*, the *transitions* and the *labels* of the automaton. Functions $s$, $d$ and $l$ associate to each transition respectively its *source*, its *destination* state and its *label.* If $t \in T$ is such that $s(t) = q$, $d(t) = q'$ and $l(t) = \lambda$, then we write in brief $t : q \xrightarrow{\lambda} q'$. The initial state of the automaton is designated by $i(*)$.

Given two automata $A_1$ and $A_2$ on the same set $L$ of labels, a *morphism* $m : A_1 \to A_2$ is a pair of arrows $m_Q : Q_1 \to Q_2$ and $m_T : T_1 \to T_2$ that respect sources, destinations, labels, and initial state, i.e., such that the two overlapped diagrams

$$\begin{tikzcd} & T_1 \arrow[dl, "l_1"'] \arrow[r, bend left, "s_1"] \arrow[r, bend right, "d_1"'] \arrow[dd, "m_T"'] & Q_1 \arrow[dd, "m_Q"] & \\ L & & & \{*\} \arrow[ul, "i_1"'] \arrow[dl, "i_2"] \\ & T_2 \arrow[ul, "l_2"] \arrow[r, bend left, "s_2"] \arrow[r, bend right, "d_2"'] & Q_2 & \end{tikzcd}$$

commute in the obvious way.

The category $\mathbf{Aut}_L$ of the automata on labels $L$ is defined by using automata with labels $L$ as objects and morphisms between such automata as arrows; identity arrow and composition between arrows are defined in the obvious way.

HD-automata can be defined in a similar way: we have just to replace the category **Set** with a category of *named sets.*

**Definition 3.1 (named sets)** A *named set* $\mathsf{E}$ is a set denoted by $E$, and a family of name sets indexed by $E$, namely $\{\mathsf{E}[e] \in Set\}_{e \in E}$ (i.e., $\mathsf{E}[\_]$ is a map form $E$ to $Set$).
Given two named sets $\mathsf{E}$ and $\mathsf{E}'$, a *named function* $\mathsf{m} : \mathsf{E} \to \mathsf{E}'$ is a function on the sets $m : E \to E'$ and a family of name embeddings (i.e., of injective

functions) indexed by $m$, namely $\{\mathsf{m}[e,e'] : \mathsf{E}'[e'] \hookrightarrow \mathsf{E}[e]\}_{\langle e,e'\rangle \in m}$.

$$\begin{tikzcd} \mathsf{E} \arrow[d, "\mathsf{m}"] & \ni & e \arrow[d, mapsto, "m"] & \mathsf{E}[e] \\ \mathsf{E}' & \ni & e' & \mathsf{E}'[e'] \arrow[u, hook, "{\mathsf{m}[e,e']}"'] \end{tikzcd}$$

A named set $\mathsf{E}$ is *finitely named* if $\mathsf{E}[e]$ is finite for each $e \in E$. A named set $E$ is *finite* if it is finitely named and set $E$ is finite.

The *category* **NSet** *of named sets* has named sets as objects and named functions as arrows; in particular:

- if $\mathsf{E}$ is a named set, then $\mathsf{id}_\mathsf{E}$ is the named function such that, for each $e \in E$, $id_E(e) = e$ and $\mathsf{id}_\mathsf{E}[e,e] = \mathrm{id}_{\mathsf{E}[e]}$;
- if $\mathsf{m} : \mathsf{E}_1 \to \mathsf{E}_2$ and $\mathsf{m}' : \mathsf{E}_2 \to \mathsf{E}_3$ are two named functions, then $\mathsf{m};\mathsf{m}' : \mathsf{E}_1 \to \mathsf{E}_3$ is the named function such that, for each $e \in E_1$, $m;m'(e) = m'(m(e))$ and, if $m(e) = e'$ and $m'(e') = e''$ then $\mathsf{m};\mathsf{m}'[e,e''] = \mathsf{m}'[e',e''];\mathsf{m}[e,e']$.

**Definition 3.2 (HD-automata)** Let $\mathsf{Start}$ be the named set with $*$ as singleton element and $\mathsf{Start}[*] = \mathfrak{N}$. A *HD-automaton* is a diagram

$$\begin{tikzcd} \mathsf{L} & \mathsf{T} \arrow[l, "\mathsf{l}"'] \arrow[r, bend left, "\mathsf{s}"] \arrow[r, bend right, "\mathsf{d}"'] & \mathsf{Q} & \mathsf{Start} \arrow[l, "\mathsf{i}"'] \end{tikzcd}$$

in the category **NSet** of named sets.
A HD-automaton is *finitely named* if $\mathsf{L}$, $\mathsf{Q}$ and $\mathsf{T}$ are finitely named; it is *finite* if, in addition, $\mathsf{Q}$ and $\mathsf{T}$ are finite.
Given two HD-automata $\mathcal{A}_1$ and $\mathcal{A}_2$ on the same named set $\mathsf{L}$ of labels, a *morphism* $m : \mathcal{A}_1 \to \mathcal{A}_2$ is a pair of arrows $\mathsf{m}_Q : \mathsf{Q}_1 \to \mathsf{Q}_2$ and $\mathsf{m}_T : \mathsf{T}_1 \to \mathsf{T}_2$ that respects sources, destinations, labels, and initial state, i.e., such that the two overlapped diagrams

$$\begin{tikzcd} & \mathsf{T}_1 \arrow[dl, "\mathsf{l}_1"'] \arrow[r, bend left, "\mathsf{s}_1"] \arrow[r, bend right, "\mathsf{d}_1"'] \arrow[dd, "\mathsf{m}_\mathsf{T}"'] & \mathsf{Q}_1 \arrow[dd, "\mathsf{m}_\mathsf{Q}"] & \\ \mathsf{L} & & & \mathsf{Start} \arrow[ul, "\mathsf{i}_1"'] \arrow[dl, "\mathsf{i}_2"] \\ & \mathsf{T}_2 \arrow[ul, "\mathsf{l}_2"] \arrow[r, bend left, "\mathsf{s}_2"] \arrow[r, bend right, "\mathsf{d}_2"'] & \mathsf{Q}_2 & \end{tikzcd}$$

commute in the obvious way.
The category $\mathbf{HD}_\mathsf{L}$ of the HD-automata on labels $\mathsf{L}$ is defined as the full subcategory of **HD** whose objects have $\mathsf{L}$ as the set of labels and respect the following condition:

$$\mathsf{T}[t] = \mathrm{cod}(\mathsf{s}[t,s(t)]) \cup \mathrm{cod}(\mathsf{l}[t,l(t)]) \text{ for each } t \in T.$$

Let $t$ be a transition of a HD-automaton such that $s(t) = q$, $d(t) = q'$ and $l(t) = \lambda$ (in this case we write in brief $t : q \xrightarrow{\lambda} q'$). Then $\mathsf{s}[t,q]$ embeds the names of $q$ into the names of $t$, whereas $\mathsf{d}[t,q']$ embeds the names of $q'$ into the names of $t$; in this way, a partial correspondence is defined between the names of the source state and those of the target; the names which appear in the source and not in the target are discarded, or forgotten, during the transitions, whereas the names that appear in the target but not in the source are created during the transition. Condition "$\mathsf{T}[t] = \mathrm{cod}(\mathsf{s}[t, s(t)]) \cup \mathrm{cod}(\mathsf{l}[t, l(t)])$" corresponds to require that all the names that are created in the transition must appear explicitly in the label (name discarding, instead, can appear silently).

The *initial state* $q_0$ of a HD-automaton is designated by $i(*)$, whereas $\mathsf{i}[*, q_0]$ is the *initial embedding* that maps the names of the initial state into the set $\mathfrak{N}$ of global names.

## *3.1 Well-sorted HD-automata*

The HD-automata we have defined above are satisfactory for representing the operational semantics of many history dependent formalisms, like CCS with localities [20] and Petri nets with history-preserving bisimulation [19]. They are not completely adequate for the $\pi$-calculus.

In fact, let $p(a, b, c)$ and $q(a, b)$ be equivalent $\pi$-calculus agents with different sets of free names[3] and suppose the two agents perform a bound output. According to Definition 2.1, in checking bisimilarity we require that the object of the bound output is a new name for *both* agents. On the HD-automata this can be achieved by representing the bound output with an unique transition that introduces a new name.

If the two agents perform an input, however, all the names must be considered as possible input values. To represent the input on a HD-automaton, we have to consider a transition for each of the names which are present in the source state, and a transition corresponding to the input of a fresh name. In both the HD-automata corresponding to $p$ and $q$, hence, there are transitions corresponding to the input of names $a$ and $b$ and to the input of a fresh name. The transition for name $c$ appears only in the HD-automaton of $p$, since $c$ is not free in $q$: this transition of $p$ is matched in $q$ by the transition for the fresh name.

This shows that the objects of bound outputs and of inputs have different meanings: in the case of bound outputs they are *new* names, whereas the objects of inputs are either already present in the source state or *universal* names (i.e., they represent *all the other* names, including the names which are free only in the other agent). In the HD-automata, however, there is only one way to introduce fresh names in a transition; so we need to add a new component to the HD-automata to distinguish between bound-output-like transitions and fresh-input-like transitions. This new component, called *sorting*, allows to distinguish the names of a label that *must* appear in the source (the `old` names), those that *cannot* appear in the source (the `new` names) and those that *may* appear in the source (the `both` names).

[3] This can be easily obtained, for instance by getting $p(a, b, c) = q(a, b) + (\nu x)\, \bar{x}c.0$, where the component $(\nu x)\, \bar{x}c.0$ is deadlocked.

**Definition 3.3 (well-sorted HD-automata)** Let $\mathsf{L}$ be a named set of labels. A *sorting* $\Gamma$ for $\mathsf{L}$ associates to each label $\lambda \in L$ a function $\Gamma_\lambda : \mathsf{L}[\lambda] \to \{\mathtt{new}, \mathtt{old}, \mathtt{both}\}$.
The category $\mathbf{HD}_{\mathsf{L},\Gamma}$ of the well-sorted HD-automata on labels $\mathsf{L}$ and sorting $\Gamma$ is defined as the full subcategory of $\mathbf{HD}_{\mathsf{L}}$ whose objects respect following condition:

for each $t \in T$ and $n \in \mathsf{L}[l(t)]$, if $\mathsf{l}[t, l(t)](n) \in \mathrm{cod}(\mathsf{s}[t, s(t)])$ then $\Gamma_{l(t)}(n) \neq \mathtt{new}$ and if $\mathsf{l}[t, l(t)](n) \notin \mathrm{cod}(\mathsf{s}[t, s(t)])$ then $\Gamma_{l(t)}(n) \neq \mathtt{old}$.

According to the previous considerations, the names of a transition $t : q \xrightarrow{\lambda} q'$ are classified as follows:

- $\mathsf{T}[t]_{\mathtt{new}} = \{n \mid n' \in \mathsf{L}[\lambda],\, \Gamma_\lambda(n') = \mathtt{new},\, \mathsf{l}[t, \lambda](n') = n\}$ are the new names of transition $t$, i.e., the names which correspond to names of the label of sort $\mathtt{new}$;
- $\mathsf{T}[t]_{\mathtt{src}} = \mathrm{cod}(\mathsf{s}[t, q])$ are the names of transition $t$ that are already present in the source state;
- $\mathsf{T}[t]_{\mathtt{univ}} = \mathsf{T}[t]_{\mathtt{both}} \setminus \mathsf{T}[t]_{\mathtt{src}}$ are the *universal names* of transition $t$, i.e., the names which correspond to names of the label of sort $\mathtt{both}$ and which are not present in the source state.

Notice that $\mathsf{T}[t]_{\mathtt{src}}$, $\mathsf{T}[t]_{\mathtt{new}}$ and $\mathsf{T}[t]_{\mathtt{univ}}$ are a partition of $\mathsf{T}[t]$, i.e., they are disjoint and their union contains all the names of $t$.

## 4 Representing $\pi$-calculus agents as HD-automata

We are interested in the representation of $\pi$-calculus agents as HD-automata. First we define the named set of labels $\mathsf{L}_\pi$ for this language: we have to distinguish between synchronizations, inputs, free outputs and bound outputs. Thus the set of labels is

$$L_\pi = \{\mathtt{tau}, \mathtt{in}, \mathtt{in}_2, \mathtt{out}, \mathtt{out}_2, \mathtt{bout}\}$$

where $\mathtt{in}_2$ and $\mathtt{out}_2$ are used when subject and object names of inputs or free outputs coincide (these special labels are necessary, since the function from the names associated to a label into the names associated to a transition must be injective). No name is associated to $\mathtt{tau}$, one name ($n$) is associated to $\mathtt{in}_2$ and $\mathtt{out}_2$ and two names ($n_{\mathrm{sub}}$ and $n_{\mathrm{obj}}$) are associated to $\mathtt{in}$, $\mathtt{out}$ and $\mathtt{bout}$.

The sorting $\Gamma_\pi$ on $L_\pi$ is defined as follows:

| $\lambda$ | `tau` | `in` | | $\mathtt{in}_2$ | `out` | | $\mathtt{out}_2$ | `bout` | |
|---|---|---|---|---|---|---|---|---|---|
| $n \in L(\lambda)$ | – | $n_{\mathrm{sub}}$ | $n_{\mathrm{obj}}$ | $n$ | $n_{\mathrm{sub}}$ | $n_{\mathrm{obj}}$ | $n$ | $n_{\mathrm{sub}}$ | $n_{\mathrm{obj}}$ |
| $\Gamma_\lambda(n)$ | – | `old` | `both` | `old` | `old` | `old` | `old` | `old` | `new` |

This means that the subject names of the labels must be old names, whereas the object names must be old in the case of free output, new in the case of bound output and can be either old or new in the case of input.

To associate a HD-automaton to a $\pi$-calculus agent, we have to represent the derivatives of the agent as states of the automaton and their transitions as transitions in the HD-automaton; the names corresponding to a state are the free names of the corresponding agent, the names corresponding to a transition are the free names of the source state plus the new names (if any) appearing in the label of the transition. A label of $\mathsf{L}_\pi$ is associated to each transition in the obvious way.

This naive construction can be improved to obtain more compact HD-automata. Consider the agent $p = x(z).q(x, y, z)$; it can perform an infinite number of input transitions, corresponding to different received names. In the context of HD-automata, however, due to the local nature of names, the transitions of $p$ corresponding to the input of all the names different from $x$ and $y$ are indistinguishable; so it is sufficient to consider just three input transitions for $p$, i.e., the inputs of names $x$ and $y$, and the input of one representative of the fresh names.

Similarly, it is sufficient to consider just one bound output, whose extruded name is the representative of the names not appearing in the agent; finally, all the $\tau$ and the free output transitions have to be considered.

According to the following definition we choose to use the first name which does not appear free in $p$ — namely $\min(\mathfrak{N} \smallsetminus \mathtt{fn}(p))$ — as representative for the input and bound output transitions of $p$.

**Definition 4.1 (representative transitions)** A $\pi$-calculus transition $t : p \xrightarrow{\alpha} q$ is a *representative transition* if $\mathtt{n}(\alpha) \subseteq \mathtt{fn}(p) \cup \{\min(\mathfrak{N} \smallsetminus \mathtt{fn}(p))\}$.

The following lemma shows that the representative transitions express, up to $\alpha$-conversion, all the behaviors of an agent.

**Lemma 4.2** *Let $t : p \xrightarrow{\alpha} q$, with $\alpha = ax$ (resp. $\alpha = \bar{a}(x)$), be a non-representative $\pi$-calculus transition. Then there is some representative transition $t' : p \xrightarrow{\alpha'} q'$, with $\alpha' = ay$ (resp. $\alpha' = \bar{a}(y)$), such that $q' = q\{y/x\ x/y\}$.*

If only representative transitions are used when building a HD-automaton from a $\pi$-calculus agent, the obtained HD-automaton is finite-branching (i.e., with a finite set of transitions from each state of the automaton).

Another advantage of using local names is that two agents differing only for a bijective substitution can be collapsed in the same state in the HD-automaton: we assume to have a function $\mathtt{norm}$ that, given an agent $p$, returns a pair $(q, \sigma) = \mathtt{norm}(p)$, where $q$ is the representative of the class of agents differing from $p$ for bijective substitutions and $\sigma : \mathtt{fn}(q) \to \mathtt{fn}(p)$ is the bijective substitution such that $p = q\sigma$.

**Definition 4.3 (from $\pi$-calculus agents to HD-automata)** The HD-automaton $\mathcal{A}_p$ corresponding to a $\pi$-calculus agent $p$ is defined by the following

| $\alpha$ | $\tau$ | $xy$ | | $xx$ | $\bar{x}y$ | | $\bar{x}x$ | $\bar{x}(y)$ | |
|---|---|---|---|---|---|---|---|---|---|
| $\lambda$ | `tau` | `in` | | $\texttt{in}_2$ | `out` | | $\texttt{out}_2$ | `bout` | |
| $n \in \mathsf{L}[\lambda]$ | $-$ | $n_{\text{sub}}$ | $n_{\text{obj}}$ | $n$ | $n_{\text{sub}}$ | $n_{\text{obj}}$ | $n$ | $n_{\text{sub}}$ | $n_{\text{obj}}$ |
| $\kappa(n)$ | $-$ | $x$ | $y$ | $x$ | $x$ | $y$ | $x$ | $x$ | $y$ |

Table 2
Relations between $\pi$-calculus labels and labels of HD-automata.

rules:

- if $\texttt{norm}(p) = (p', \sigma')$ then:
  - $\cdot$ $p' \in Q$ is the initial state and $\mathsf{Q}[p'] = \texttt{fn}(p')$;
  - $\cdot$ $\sigma'$ is the initial embedding;
- if $q \in Q$, $t : q \xrightarrow{\alpha} q'$ is a representative transition and $\texttt{norm}(q') = (q'', \sigma)$, then:
  - $\cdot$ $q'' \in Q$ and $\mathsf{Q}[q''] = \texttt{fn}(q'')$;
  - $\cdot$ $t \in T$ and $\mathsf{T}[t] = \texttt{fn}(q) \cup \texttt{bn}(\alpha)$;
  - $\cdot$ $s(t) = q$, $d(t) = q''$, $\mathsf{s}[t, q] = \mathrm{id}_{\texttt{fn}(q)}$ and $\mathsf{d}[t, q''] = \sigma$;
  - $\cdot$ $l(t) = \lambda$ and $\mathsf{l}[t, \lambda] = \kappa$ are defined as in Table 2.

**Lemma 4.4** *For every $\pi$-calculus agent $p$, the HD-automaton $\mathcal{A}_p$ is well-sorted for labels $\mathsf{L}_\pi$ and sorting $\Gamma_\pi$.*

For each $\pi$-calculus agent $p$, the HD-automaton $\mathcal{A}_p$ is obviously finitely named. Now we will identify a class of agents that generate a finite HD-automaton. This is the class of *finitary* $\pi$-calculus agents, which is defined like the corresponding class of CCS agents.

**Definition 4.5 (finitary agents)** The *degree of parallelism* $\texttt{deg}(p)$ of a $\pi$-calculus agent $p$ is defined as follows:

$$\begin{array}{ll} \texttt{deg}(\mathbf{0}) = 0 & \texttt{deg}(\mu.p) = 1 \\ \texttt{deg}((\nu\alpha)\,p) = \texttt{deg}(p) & \texttt{deg}(p|q) = \texttt{deg}(p) + \texttt{deg}(q) \\ \texttt{deg}(p{+}q) = 1 & \texttt{deg}([x{=}y]p) = \texttt{deg}(p) \\ \texttt{deg}(A) = 1 & \end{array}$$

A $\pi$-calculus agent $p$ is *finitary* if $\max\{\texttt{deg}(p') \mid p \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_i} p'\} < \infty$.

**Theorem 4.6** *Let $p$ be a finitary $\pi$-calculus agent. Then the HD-automaton $\mathcal{A}_p$ is finite.*

An important class of finitary agents which can be characterized syntactically is the class of the agents with *finite control*, i.e., the agents without parallel composition in the body of recursive definitions. In this case, after an initialization phase during which a finite set of processes acting in parallel is created, no new processes can be generated.

## 5 Bisimulation for HD-automata

In this section we introduce a notion of bisimulation on HD-automata and give some basic properties of this bisimulation. We also show that the definition of bisimulation on $\pi$-calculus agents is captured exactly by the bisimulation on HD-automata.

### *5.1 Open maps and bisimulations*

Consider a morphism $m : A_1 \to A_2$ in the category of automata. Relation

$$\mathcal{R} = \{\langle q_1, q_2 \rangle \in Q_1 \times Q_2 \mid q_2 = m_Q(q_1)\}$$

is a simulation for $A_1$ and $A_2$. In fact, assume $q_1 \mathrel{\mathcal{R}} q_2$ and $t_1 : q_1 \xrightarrow{\lambda} q_1'$; then we have $t_2 : q_2 \xrightarrow{\lambda} q_2'$ and $q_1' \mathrel{\mathcal{R}} q_2'$ by taking $q_1' \mathrel{\mathcal{R}} q_2'$. Moreover $q_{01} \mathrel{\mathcal{R}} q_{02}$.

So, a morphism $m : A_1 \to A_2$ expresses the fact that all the transitions of $A_1$ can be simulated in $A_2$, starting from the initial states. In general, however, it is not true that all the transitions of $A_2$ can be simulated in $A_1$.

However, it is possible to define a particular class of "bisimulation" morphisms, such that the existence of such a morphism from $A_1$ to $A_2$ guarantees not only that the transitions of $A_1$ can be adequately simulated in $A_2$ but also the converse; i.e., the existence of a "bisimulation" morphism guarantees that $A_1$ and $A_2$ are bisimilar. In general, it is not true the converse, i.e., there exist bisimilar automata $A_1$ and $A_2$ such that no "bisimulation" morphism (nor generic morphisms) can be found between them. However, whenever two automata $A_1$ and $A_2$ are bisimilar, it is possible to find a common predecessor $A$ and a span of "bisimulation" morphisms $m_1 : A \to A_1$ and $m_2 : A \to A_2$ between them.

$$\begin{tikzcd}
 & A \arrow[dl, "m_1"'] \arrow[dr, "m_2"] & \\
A_1 & & A_2
\end{tikzcd}$$

This class of "bisimulation" morphisms have been defined in various manner in the literature, and different names have been given to them. Here we just consider the approach of *open maps* [10], that it is general enough to be applied not only to automata, but also to other models of concurrency, like Petri nets and event structures.

Assume a category $\mathbf{M}$ of *models*. Let $\mathbf{E}$ be the subcategory of $\mathbf{M}$ whose objects are the *experiments* that can be executed on $\mathbf{M}$ and whose arrows express how the experiments can be extended. If $X$ is an object of $\mathbf{E}$ and $M$ is an object of $\mathbf{M}$, an arrow $x : X \to M$ of $\mathbf{M}$ represents the execution of the experiment $X$ in the model $M$.

Consider an arrow $m : M \to N$ in $\mathbf{M}$. We can see this arrow as a simulation of model $M$ in model $N$. So, correctly, if an experiment $X$ can be executed in $M$ (there exists an arrow $x : X \to M$) and $N$ can simulate $M$ (there exists an arrow $m : M \to N$) then the experiment $X$ can be executed in $N$ (via the arrow $x; m : X \to N$).

Suppose now to extend the experiment $X$ to an experiment $Y$ (via an arrow $f : X \to Y$ in $\mathbf{E}$) and that an arrow $y : Y \to N$ exists such that the following diagram commutes in $\mathbf{M}$.

(1)
$$\begin{tikzcd} X \arrow[r, "x"] \arrow[d, "f"'] & M \arrow[d, "m"] \\ Y \arrow[r, "y"] & N \end{tikzcd}$$

This means that the execution of the experiment $X$ in $N$ (via $x; m$) can be extended to an execution of the experiment $Y$ in $N$ (via $y$).

This does not imply in general that also the execution of $X$ in $M$ can be extended to an execution of $Y$ in $M$ (which equates $y$ via $m$) but we can make this sure by requiring that there is an arrow $y'$ such that the diagram

(2)
$$\begin{tikzcd} X \arrow[r, "x"] \arrow[d, "f"'] & M \arrow[d, "m"] \\ Y \arrow[r, "y"] \arrow[ur, "y'"] & N \end{tikzcd}$$

commutes. Given $m$, if for each commuting diagram (1) there is an arrow $y'$ such that also (2) commutes, we say that $m$ is an $\mathbf{E}$-*open map*.

It is easy to check that the open maps form a subcategory of $\mathbf{M}$ (i.e., identities are open and open maps are closed for composition).

**Definition 5.1 (open bisimulation)** We say that two objects $M_1$ and $M_2$ of $\mathbf{M}$ are *open-bisimilar* with respect to $\mathbf{E}$ if and only if there is a span of $\mathbf{E}$-open maps $m_1, m_2$.

$$\begin{tikzcd} & M \arrow[dl, "m_1"'] \arrow[dr, "m_2"] & \\ M_1 & & M_2 \end{tikzcd}$$

In [10] it is shown that, if the category $\mathbf{Aut}_L$ is used as the category of the models and the full subcategory $\mathbf{Bran}_L$ of the *branches* (i.e., of those finite automata which consist of a linear sequence of transitions) is used as the category of experiments, then two automata are open-bisimilar if and only if they are bisimilar according to the classical definition.

### *5.2 Application to the HD-automata*

In the case of HD-automata, an experiment is a finite sequences of transitions and an extended experiment can be obtained by adding new transitions. Moreover, we require that no name is forgotten during an experiment, since this models the idea that the observer can remember all the names previously used in the experiment. However, this is not a crucial point for the validity of Theorem 5.4.

**Definition 5.2 (category of HD-experiments)** A HD-automaton $\mathcal{X}$ is a *HD-experiment* if:

- $Q = \{q_0, q_1, \ldots, q_n\}$ are the states and $T = \{t_1, \ldots, t_n\}$ are the transitions, and $s(t_i) = q_{i-1}$ and $d(t_i) = q_i$;
- for all $t \in T$, $\mathsf{d}[t, d(t)] : \mathsf{Q}[d(t)] \hookrightarrow \mathsf{T}[t]$ is bijective.

A morphism $\langle \mathsf{m_Q}, \mathsf{m_T} \rangle : \mathcal{X} \to \mathcal{X}'$ is *name preserving* if $\mathsf{m_Q}$ and $\mathsf{m_T}$ are bijections on the names, i.e., $\mathsf{m_Q}[q, m_Q(q)]$ is a bijection between $\mathsf{Q}'[m_Q(q)]$ and $\mathsf{Q}[q]$ for all $q \in Q$, and similarly for $\mathsf{m_T}$.
The *category* **Exp** *of HD-experiments* is the subcategory of **HD** with HD-experiments as objects and name preserving morphisms as arrows.
Category $\mathbf{Exp}_{\mathsf{L},\Gamma}$ is the full subcategory of **Exp** whose objects are $\mathbf{HD}_{\mathsf{L},\Gamma}$-automata.

Now we can apply the general definition of open-bisimilarity in our case.

**Definition 5.3 (HD-bisimilarity)** Two well-sorted HD-automata $\mathcal{A}$ and $\mathcal{B}$ on the same labels $L$ and sorting $\Gamma$ are *HD-bisimilar*, written $\mathcal{A} \sim \mathcal{B}$, if they are open-bisimilar w.r.t. experiments $\mathbf{Exp}_{\mathsf{L},\Gamma}$.

The definition of HD-bisimilarity can be applied also in the case of HD-automata obtained from $\pi$-calculus agents. The induced equivalence on the agents coincides exactly with the strong, early bisimilarity relation $\sim_\pi$.

**Theorem 5.4** *Let $p_1$ and $p_2$ be $\pi$-calculus agents. Then $p_1 \sim_\pi p_2$ iff $\mathcal{A}_{p_1} \sim \mathcal{A}_{p_2}$.*

It is also possible to give an explicit definition of HD-bisimulation, in terms of relations on the states, rather that in terms of bisimulation morphisms. The explicit definition is reported in Appendix A.

## 6 Concluding remarks

In this paper we have briefly described history dependent automata, an operational model adequate to deal with history dependent calculi. In particular, we have represented $\pi$-calculus agents via HD-automata and strong, early $\pi$-calculus bisimilarity via a general definition of bisimulation equivalence on HD-automata. All these results will appear in more detail in [21].

We want to stress that HD-automata can be applied successfully also to the late semantics of $\pi$-calculus (only the translation of Definition 4.3 has to be changed) or to other examples of history dependent calculi, as for instance CCS with localities [4,20] or with causality [6,18]. It is also possible to define a *weak* HD-bisimulation that applies to all these cases. Also, HD-automata can be applied to formalisms outside the field of process algebras; this is the case for the history-preserving semantics of Petri nets [2,19].

HD-automata are very promising for the development of automatic verification tools for history dependent calculi. In fact, HD-automata can be used as a common format in which various history-dependent calculi can be translated, so that general algorithms on HD-automata can be re-used for all these calculi. We are developing a verification environment which is based on the approach above. The environment provides a number of front ends translating the different history dependent formalisms into HD-automata, and a set of tools to edit, visualize, compose and check for equivalence the obtained HD-automata. It is also possible to associate ordinary automata to the HD-automata, in such a way that bisimilar HD-automata are mapped into bisimilar automata, and finite HD-automata are mapped into finite automata. In this way, classical algorithms and tools for ordinary automata [3] can be re-used. A preliminary report on the development of the tool appeared in [8].

## References

[1] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217-248, 1992.

[2] E. Best, R. Devillers, A. Kiehn and L. Pomello. Fully concurrent bisimulation. *Acta Informatica*, 28:231–264, 1991.

[3] A. Bouali, S. Gnesi and S. Larosa. The integration project for the JACK environment. *Bullettin of the EATCS*, 54, 1994.

[4] G. Boudol, I. Castellani, M. Hennessy and A. Kiehn. Observing localities. *Theoretical Computer Science*, 114:31–61, 1993.

[5] M. Dam. On the decidability of process equivalences for the $\pi$-calculus. *Theoretical Computer Science*, 183:215-228, 1997.

[6] Ph. Darondeau and P. Degano. Causal trees. In *Proc. ICALP'89*, LNCS 372. Springer Verlag, 1989.

[7] P. Degano, R. De Nicola and U. Montanari. CCS is an (augmented) contact free C/E system. In *Proc. Adv. Sch. on Mathematical Models for the Semantics of Parallelism*, LNCS 280. Springer Verlag, 1986.

[8] G. Ferrari, G. Ferro, S. Gnesi, U. Montanari, M. Pistore and G. Ristori. An automata based verification environment for mobile processes. In *Proc. TACAS'97*, LNCS 1217. Springer Verlag, 1997.

[9] P. Inverardi and C. Priami. Evaluation of tools for the analysis of communicating systems. *Bulletin of the EATCS*, 45:158–185, 1991.

[10] A. Joyal, M. Nielsen and G. Winskel. Bisimulation from open maps. In *Proc. LICS'93*. Full version as BRICS RS-94-7. Department of Computer Science, University of Aarhus. 1994.

[11] A. Kiehn. Local and global causes. Tech. Rep. 42/23/91, Institut für Informatik, TU München, 1991.

[12] E. Madelaine. Verification tools for the CONCUR project. *Bulletin of the EATCS*, 47:110–126, 1992.

[13] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[14] R. Milner. The polyadic $\pi$-calculus: a tutorial. In *Logic and Algebra of Specification*, NATO ASI Series F, Vol. 94. Springer Verlag, 1993.

[15] R. Milner, J. Parrow and D. Walker. A calculus of mobile processes (parts I and II). *Information and Computation*, 100:1–77, 1992.

[16] R. Milner, J. Parrow and D. Walker. Modal logic for mobile processes. *Theoretical Computer Science*, 114:149-171, 1993.

[17] U. Montanari and M. Pistore. Checking bisimilarity for finitary $\pi$-calculus. In *Proc. CONCUR'95*, LNCS 962. Springer Verlag, 1995.

[18] 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.

[19] U. Montanari and M. Pistore. Minimal transition systems for history-preserving bisimulation. In *Proc. STACS'97*, LNCS 1200. Springer Verlag, 1997.

[20] U. Montanari, M. Pistore and D. Yankelevich. Efficient minimization up to location equivalence. In *Proc. ESOP'96*, LNCS 1058. Springer Verlag, 1996.

[21] U. Montanari and M. Pistore. History dependent automata. Tech. Rep. in preparation. Dipartimento di Informatica, Università di Pisa, 1998.

[22] D. Park. *Concurrency and Automata on Infinite Sequences*, LNCS 104. Springer Verlag, 1980.

[23] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD Thesis CST-99-93, University of Edinburgh, 1992.

[24] D. Walker. Objects in the $\pi$-calculus. *Information and Computation*, 116:253–271, 1995.

## A Explicit definition of HD-bisimulation

Here we want to give an explicit definition of bisimulation on HD-automata which is equivalent to the one given in Definition 5.3 by exploiting the open maps. This definition is less satisfactory than the one given via open maps, since, as we will see, it has to deal explicitly with the different sorts of names that appear in a transition. However, the explicit definition makes it clear that the bisimulation of two HD-automata can be effectively decided whenever the two HD-automata are finite.

Due to the private nature of the names appearing in the states of HD-automata, bisimulations cannot simply be relations on the states; they must also deal with name correspondences: a HD-bisimulation is a set of triples of the form $\langle q_1, \delta, q_2 \rangle$ where $q_1$ and $q_2$ are states of the automata and $\delta$ is a partial bijection between the names of the states. The bijection is partial since we allow for equivalent states with different numbers of names (for instance, equivalent $\pi$-calculus agents can have different sets of free names). In what follows, we represent a *partial bijection* $f$ from set $A$ to set $B$ with $f : A \hookrightarrow B$.

Suppose that we want to check if states $q_1$ and $q_2$ are (strongly) bisimilar via the partial bijection $\delta : \mathsf{Q}[q_1] \hookrightarrow \mathsf{Q}[q_2]$ and suppose that $q_1$ can perform a transition $t_1 : q_1 \xrightarrow{\lambda} q_1'$. We assume for the moment that all the names of the label $\lambda$ are of sorts `old` or `new`. Then we have to find a transition $t_2 : q_2 \xrightarrow{\lambda} q_2'$ that matches $t_1$, i.e., not only the two transitions must have the same label, but also the names associated to the labels must be used consistently. This means that:

*a*) if a name $n$ of the label is of sort `old`, then the corresponding names in the source states $q_1$ and $q_2$ must be in correspondence by $\delta$ (such names surely exist in $q_1$ and $q_2$, if the HD-automata are well-sorted);

*b*) if a name $n$ of the label is of sort `new`, then the corresponding names in the transitions $t_1$ and $t_2$ are put in correspondence (if the HD-automata are well-sorted, no names corresponding to $n$ appear in the source states).

This behavior is obtained by requiring that a partial bijection $\zeta : \mathsf{T}[t_1] \hookrightarrow \mathsf{T}[t_2]$ exists such that: *i*) $\zeta$ coincides with $\delta$ if restricted to the names of the source states (obviously, via the embeddings $\mathsf{s}[t_1, q_1]$ and $\mathsf{s}[t_2, q_2]$); *ii*) the names associated to the labels are the same, via $\zeta$, and *iii*) the destination states $q_1'$ and $q_2'$ are bisimilar via a partial bijection $\delta'$ which is compatible with $\zeta$ (i.e., if two names are related by $\delta'$ in the destination states, then the corresponding names in the transitions are related by $\zeta$).

The situation is more complex if a name $n$ of the label $\lambda$ is of sort `both`. We can distinguish three more cases:

*c*) the name $n_1$ in $t_1$ corresponding to $n$ is already present in $q_1$ *and* is associated via $\delta'$ to a name of $q_2$; in this case a matching transition $t_2$ from $q_2$ must use for $n$ this associated name;

*d*) the name $n_1$ in $t_1$ corresponding to $n$ is already present in $q_1$ *and* it is *not* associated via $\delta$ to a name of $q_2$; in this case $t_1$ must be matched by a transition $t_2$ from $q_2$ which uses an universal name for $n$; the meaning of this is that name $n_1$ is handled as a special case in state $q_1$, but is handled by the default transition in $q_2$;

*e*) the name $n_1$ in $t_1$ corresponding to $n$ is *not* already present in $q_1$ (i.e., $n_1$ is an universal name); in this case we require that $t_1$ is matched:

*e*1) by a transition from $q_2$ which uses an universal name for $n$ (the two universal names are put in correspondence), and

$e2$) for each name $n_2$ of $q_2$ not appearing (via $\delta$) in $q_1$, by a transition from $q_2$ that uses $n_2$ for $n$ (in this case, a new correspondence is set for $n_1$ and $n_2$); the meaning of this is that the default transition in $q_1$ must match also the special cases of $q_2$ which are not contemplated by $q_1$.

This more complex behavior is obtained by requiring that, for each transition $t_1 : q_1 \xrightarrow{\lambda} q_1'$ and for each possible partial bijection $\xi$ between the universal names of $t_2$ and the names of $q_2$ which do not already correspond to names of $q_1$, there is some transition $t_2 : q_2 \xrightarrow{\lambda} q_2'$ and some partial bijection $\zeta : \mathsf{T}[t_1] \hookrightarrow \mathsf{T}[t_2]$ extending $\xi; \mathsf{s}[t_2, q_2]$ such that *i*) $\zeta$ satisfies the rules *a*-*e*) above[4]; *ii*) the names associated to the labels are the same, via $\zeta$, and *iii*) the destination states $q_1'$ and $q_2'$ are bisimilar via a partial bijection $\delta'$ which is compatible with $\zeta$. Notice that to a name of $t_1$ can correspond no name of $t_2$ via $\zeta$, if no name is associated to it via $\delta$ and the name does not appear in the label.

**Definition A.1 (HD-bisimulation)** Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two HD-automata in $\mathbf{HD}_{\mathsf{L},\Gamma}$. A *HD-simulation* for $\mathcal{A}_1$ and $\mathcal{A}_2$ is a set of triples $\mathcal{R} \subseteq \{\langle q_1, \delta, q_2\rangle \mid q_1 \in Q_1, q_2 \in Q_2, \delta : \mathsf{Q}_1[q_1] \hookrightarrow \mathsf{Q}_2[q_2]\}$ such that, whenever $\langle q_1, \delta, q_2\rangle \in \mathcal{R}$ then:

for each $t_1 : q_1 \xrightarrow{\lambda} q_1'$ in $\mathcal{A}_1$ and for each $\xi : \mathsf{T}_1[t_1]_{\mathtt{univ}} \hookrightarrow \mathsf{Q}_2[q_2]$ such that $\mathrm{cod}(\xi) \cap \mathrm{cod}(\delta) = \emptyset$, there exist some $t_2 : q_2 \xrightarrow{\lambda} q_2'$ in $\mathcal{A}_2$ and some $\zeta : \mathsf{T}_1[t_1] \hookrightarrow \mathsf{T}_2[t_2]$ such that:

- $\delta = \mathsf{s}_1[t_1, q_1]; \zeta; \mathsf{s}_2[t_2, q_2]^{-1}$;
- $\xi = \zeta|_{\mathsf{T}_1[t_1]_{\mathtt{univ}}}; \mathsf{s}_2[t_2, q_2]^{-1}$;
- $\mathsf{l}_1[t_1, \lambda]; \zeta = \mathsf{l}_2[t_2, \lambda]$;
- $\langle q_1', \delta', q_2'\rangle \in \mathcal{R}$ where $\delta' \subseteq \mathsf{d}_1[t_1, q_1']; \zeta; \mathsf{d}_2[t_2, q_2']^{-1}$.

A *HD-bisimulation* for $\mathcal{A}_1$ and $\mathcal{A}_2$ is a set of triples $\mathcal{R}$ such that $\mathcal{R}$ is a HD-simulation for $\mathcal{A}_1$ and $\mathcal{A}_2$ and $\mathcal{R}^{-1} = \{\langle q_2, \delta^{-1}, q_1\rangle \mid \langle q_1, \delta, q_2\rangle \in \mathcal{R}\}$ is a HD-simulations for $\mathcal{A}_2$ and $\mathcal{A}_1$.
A HD-bisimulation on for $\mathcal{A}$ is a HD-bisimulation for $\mathcal{A}$ and $\mathcal{A}$.
The HD-automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are *(strongly) HD-bisimilar* (written $\mathcal{A}_1 \sim \mathcal{A}_2$) if there exists some HD-bisimulation for $\mathcal{A}_1$ and $\mathcal{A}_2$ such that $\langle i_1(*), \delta, i_2(*)\rangle \in \mathcal{R}$ for $\delta \subseteq \mathsf{i}_1[*, i_1(*)]; \mathsf{i}_1[*, i_1(*)]^{-1}$.

**Theorem A.2** *Two HD-automata are bisimilar according to Definition 5.3 iff they are bisimilar according to Definition A.1.*

[4] For rule $e$): let $n_1$ be an universal name of $t_1$; if $\xi(n_1) = n_2 \in \mathsf{Q}[q_2]$ then $n_1$ must be matched by $n_2$ (case $e2$); if $\xi(n_1)$ is undefined, an universal name of $t_2$ must match $n_1$ (case $e1$).