Finite State Verification for the Asynchronous π-Calculus[⋆]

Ugo Montanari and Marco Pistore

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

Abstract. The π-calculus is a development of CCS that has the ability of communicating channel names. The *asynchronous π-calculus* is a variant of the π-calculus where message emission is non-blocking.
Finite state verification is problematic in this context, since even very simple asynchronous π-processes give rise to infinite-state behaviors. This is due to phenomena that are typical of calculi with name passing and to phenomena that are peculiar of asynchronous calculi.
We present a finite-state characterization of a family of *finitary* asynchronous π-processes by exploiting *History Dependent transition systems with Negative transitions* (HDN), an extension of labelled transition systems particularly suited for dealing with concurrent calculi with name passing. We also propose an algorithm based on HDN to verify asynchronous bisimulation for finitary π-processes.

1 Introduction

A growing interest has been recently devoted to calculi and languages for distributed systems, and in particular to the new phenomena they evidence. One of these phenomena is *mobility*: in large distributed systems, like the internet, there is mobility of hardware (when a computer is moved to a different node) and mobility of code and data (when applets are downloaded from the network and executed locally, or when remote programs are executed on local data).

The π-calculus [7,6] is a foundational calculus with mobility. In the π-calculus, processes can handle channel names as messages, thus modeling changes in their neighborhood. Furthermore, name passing is enough to simulate higher order and object oriented concurrent calculi, thus also mobility of code and of data can be expressed in the π-calculus. In the original papers on π-calculus [7,6], communications are synchronous, i.e., the emission and the reception of a message are assumed to happen in the same instant. More recently, an asynchronous version of the π-calculus has been defined [5,2]. Here it is assumed that messages take time to move from the sender to the receiver, and that the sender is *not* blocked until the message is received.

⋆ Research partially supported by CNR Integrated Project "Metodi e Strumenti per la Progettazione e la Verifica di Sistemi Eterogenei Connessi mediante Reti di Comunicazione", and Esprit WG CONFER2.

While more expressive and more suitable to describe distributed systems, the calculi with name passing give rise to new problems, that cannot be solved by exploiting existing techniques for CCS-like process calculi. Here we focus on the problem of extending to (classes of) π-processes the techniques of finite state verification.

Finite state verification is successful in the case of concurrent systems, since interesting problems can be expressed by means of finite state systems. This is the case for instance of protocols, where the control part is often independent from the data part and can be verified with finite-state techniques.

In this paper we face the problem of finite state verification for the asynchronous π-calculus. This is not a trivial problem, since naive approaches lead to infinite state systems also for very simple asynchronous π-processes. Different techniques have to be exploited to obtain finite state representations for interesting classes of processes. Now we are going to describe these techniques.

As a first step, we give a new definition of bisimilarity for the asynchronous π-calculus. In the classical asynchronous bisimulations proposed in [5,1], a *lazy* approach is used for output messages: since an arbitrary amount of time can be required for a message to be delivered, messages are never forced to be emitted from the system. In this way, however, infinite state systems are obtained practically for all recursive processes: in fact, if new messages are produced but never delivered, the size of the system can grow unboundedly.

We propose a different definition of bisimulation, that we have called *hot-potato* bisimulation, where messages are emitted as soon as they are ready (a similar approach is proposed in [12] for asynchronous systems *without* mobility). In this way, the system cannot grow unboundedly due to messages that are ready to be emitted, but that are still undelivered. The classical, eager asynchronous bisimulation and the new hot-potato bisimulation coincide.

Another cause of infiniteness is the generation of fresh names. This is a general phenomenon for the π-calculus: processes have the ability of creating dynamically new channels with the environment, and fresh names have to be associated to the new channels. Standard transition systems are not very convenient for dealing with allocation and deallocation of names: name creation is handled via the exposure of an internal name, that is subject to alpha conversion, and this results in an infinite branching; moreover, if names are never removed, new states are produced at every cycle which includes a name generation. In [8] we propose an enhanced version of labelled transition systems, that we called *History Dependent (HD) transition systems*, and a corresponding HD-bisimulation; names appear explicitly in states and labels of HD, so that name creation and deallocation can be explicitly represented.

While HD and HD-bisimilarity are adequate to describe the π-calculus with synchronous communications, a more general model is needed for the asynchronous π-calculus. In this paper we define *History Dependent transition systems with Negative transitions* (HDN) and HDN-bisimulation. We show that the asynchronous π-calculus can be represented by means of HDN and that finite state HDN are obtained for an important family of π-processes. In our opinion, HDN are a rather general model for mobile calculi; for instance, in [10] they are applied also to the early/late [7] and to the open [13,11] semantics of π-calculus. We also believe that they can be applied to other calculi with mobility, like the join calculus [4].

Finally, we define an iterative method to calculate HDN-bisimilarity for a certain class of finite state HDN. This method resembles the partitioning approach for ordinary labelled transition systems [9], where a partition of the states is built and incrementally refined until all the states in the same block are equivalent. In general HDN-bisimilarity is not guaranteed to be transitive: thus it is not possible to build a partition of equivalent states. Therefore, our partitioning approach applies only to a class of *redundancy-consistent* HDN. Fortunately enough, the HDN corresponding to asynchronous π-calculus is redundancy-consistent. Hence the partitioning method applies to verify equivalence of finitary asynchronous π-processes.

2 The asynchronous π-calculus

Asynchronous processes are a subset of ordinary π-calculus processes. More precisely, output prefixes $\overline{a}b.P$ are not allowed in the asynchronous context: in $\overline{a}b.P$, in fact, process P is blocked until message $\overline{a}b$ is emitted, while in the asynchronous context message emission is required to be non-blocking. Output prefixes are replaced by *output particles* $\overline{a}b$, that represent an output communication of name b on channel a that is ready to be delivered. For the same reasons, outputs cannot appear in a choice point, so sums are restricted to τ and input prefixes.

Let $\mathfrak{N}$ be an infinite, countable set of *names*, ranged over by $a, \ldots, z$. *Asynchronous processes* are defined by the syntax:

$$P,Q \;::=\; \overline{a}b \;\big|\; G \;\big|\; P|P \;\big|\; (\boldsymbol{\nu} a)\,P \;\big|\; A(a_1,\ldots,a_n) \qquad \text{(processes)}$$
$$G,H \;::=\; \mathbf{0} \;\big|\; a(b).P \;\big|\; \tau.P \;\big|\; G{+}G \qquad \text{(guards)}$$

where we assume that a definition $A(b_1,\ldots,b_n) \overset{\text{def}}{=} G_A$ corresponds to each process identifier A. All the occurrences of b in $(\boldsymbol{\nu} b)\,P$ and $a(b).P$ are bound; *free* and *bound names* of process P are then defined as usual and we denote them with $\text{fn}(P)$ and $\text{bn}(P)$ respectively.

We define a *structural congruence* $\equiv$ that identifies all those processes that differ only for inessential details in the syntax of the programs. Formally, we define $\equiv$ as the smallest congruence that satisfies the following rules:

$$P \equiv Q \text{ if } P \text{ and } Q \text{ are alpha equivalent}$$
$$G{+}\mathbf{0} \equiv G \quad G{+}G' \equiv G'{+}G \quad G{+}(G'{+}G'') \equiv (G{+}G'){+}G''$$
$$P|\mathbf{0} \equiv P \quad P|P' \equiv P'|P \quad P|(P'|P'') \equiv (P|P')|P''$$
$$(\boldsymbol{\nu} a)\,\mathbf{0} \equiv \mathbf{0} \quad (\boldsymbol{\nu} a)\,(\boldsymbol{\nu} b)\,P \equiv (\boldsymbol{\nu} b)\,(\boldsymbol{\nu} a)\,P \quad (\boldsymbol{\nu} a)\,(P|Q) \equiv P|(\boldsymbol{\nu} a)\,Q \text{ if } a \notin \text{fn}(P)$$

The structural congruence is useful to obtain finite state representations for classes of processes. In fact, it can be used to garbage-collect terminated processes and unused restrictions.

$$\text{[TAU]}\ \tau.P \xrightarrow{\tau} P \qquad \text{[IN]}\ a(b).P \xrightarrow{a(b)} P \qquad \text{[OUT]}\ \overline{a}b \xrightarrow{\overline{a}b} \mathbf{0}$$

$$\text{[SUM]}\ \frac{G \xrightarrow{\alpha} G'}{G{+}H \xrightarrow{\alpha} G'} \qquad \text{[COMM]}\ \frac{P \xrightarrow{\overline{a}b} P' \quad Q \xrightarrow{a(c)} Q'}{P|Q \xrightarrow{\tau} P'|Q'\{b/c\}} \qquad \text{[CLOSE]}\ \frac{P \xrightarrow{\overline{a}(b)} P' \quad Q \xrightarrow{a(b)} Q'}{P|Q \xrightarrow{\tau} (\boldsymbol{\nu}b)\,(P'|Q')}$$

$$\text{[OPEN]}\ \frac{P \xrightarrow{\overline{a}b} P'}{(\boldsymbol{\nu}b)\,P \xrightarrow{\overline{a}(b)} P'}\ \text{if } a \neq b \qquad \text{[PAR]}\ \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}\ \text{if } \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset$$

$$\text{[RES]}\ \frac{P \xrightarrow{\alpha} P'}{(\boldsymbol{\nu}a)\,P \xrightarrow{\alpha} (\boldsymbol{\nu}a)\,P'}\ \text{if } a \notin \mathrm{n}(\alpha) \qquad \text{[CONG]}\ \frac{P \equiv P' \quad P' \xrightarrow{\alpha} Q' \quad Q' \equiv Q}{P \xrightarrow{\alpha} Q}$$

$$\text{[IDE]}\ \frac{G_A\{a_1/b_1 \cdots a_n/b_n\} \xrightarrow{\alpha} P}{A(a_1, \ldots, a_n) \xrightarrow{\alpha} P}\ \text{if } A(b_1, \ldots, b_n) \stackrel{\text{def}}{=} G_A$$

If $\sigma : \mathfrak{N} \to \mathfrak{N}$, we denote with $P\sigma$ the process P whose free names have been replaced according to substitution σ (possibly with changes in the bound names to avoid name clashing); 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. With some abuse of notation, we can see substitution σ in $P\sigma$ as a function on $\mathrm{fn}(P)$ rather than on $\mathfrak{N}$.

The *actions* that the processes can perform, are the following:

$$\alpha ::= \tau \;\big|\; a(c) \;\big|\; \overline{a}b \;\big|\; \overline{a}(c)$$

and are called respectively *synchronization*, *input*, *free output* and *bound output* actions; a and b are free names of α ($\mathrm{fn}(\alpha)$), whereas c is a bound name ($\mathrm{bn}(\alpha)$); moreover $\mathrm{n}(\alpha) = \mathrm{fn}(\alpha) \cup \mathrm{bn}(\alpha)$.

The operational semantics of the asynchronous π-calculus is defined by means of a labelled transition systems. The transitions for the *ground operational semantics* are defined by the axiom schemata and the inference rules of the table on the top of this page. We recall that in the *ground* semantics no name instantiation occurs in the input transitions. In [1] it is shown that ground semantics coincides with early and late semantics in the case of asynchronous π-calculus without matching.

2.1 Asynchronous bisimulation

In this section we introduce asynchronous bisimulation. As we will see, while the management of τ and output transitions in the bisimulation game is standard, a special clause is needed for the input transitions; different characterizations of asynchronous bisimulation are proposed in [1]: they differ just in the way input transitions are dealt with. Following [1], we first define $o\tau$-bisimulation, that just considers output and τ transitions; then we consider different notions of bisimulations that extend $o\tau$-bisimulation with a clause for input transitions.

Definition 1 ($o\tau$-bisimulation [1]). *A symmetric relation $\mathcal{R}$ on π-processes is a $o\tau$-*bisimulation *if $P \;\mathcal{R}\; Q$ and $P \xrightarrow{\alpha} P'$, where α is not an input transition, and* $\mathrm{bn}(\alpha) \cap \mathrm{fn}(P|Q) = \emptyset$*, imply $Q \xrightarrow{\alpha} Q'$ and $P' \;\mathcal{R}\; Q'$.*

Notice that clause "$\mathrm{bn}(\alpha) \cap \mathrm{fn}(P|Q) = \emptyset$" in the definition above assures that, when a channel name is extruded in a bound output transition, a fresh name (i.e., a name that is not used in P or Q) is used to denote that channel.

In an asynchronous context, messages can be received by a process in any moment, even if the process is not ready to consume them: in [5] this intuition is modeled by allowing every process to accept every input message, i.e., according to the semantics of [5], $P \xrightarrow{a(b)} P|\overline{a}b$ is a valid transition for every process P. This approach has some drawbacks; the most important for our purposes is that an infinite number of transitions can be performed by every process — even by process $\mathbf{0}$ — so finite state verification is not possible.

In this paper we follow instead the approach of [1]: an input transition $P \xrightarrow{a(b)} P'$ corresponds to the consumption of a message, i.e., to the execution of an input prefix. However, in the definition of asynchronous bisimulation, we cannot require that, given two bisimilar processes P and Q, each input transition $P \xrightarrow{a(b)} P'$ is matched by a transition $Q \xrightarrow{a(b)} Q'$: process Q can receive the message $\overline{a}b$ without consuming it, and be still equivalent to P. In asynchronous bisimulation [1], hence, a transition $P \xrightarrow{a(b)} P'$ can be matched either by a transition $Q \xrightarrow{a(b)} Q'$, and P' and Q' should be still bisimilar; or by a fictitious input transition of Q, that receives the message but does not consume it: this is modeled by requiring that $Q \xrightarrow{\tau} Q'$ (i.e., Q performs some internal work), and that P' is bisimilar to $Q'|\overline{a}b$ (process $Q'|\overline{a}b$ has received the message but has not yet consumed it).

Definition 2 (ground asynchronous bisimulation [1]). *A symmetric relation $\mathcal{R}$ on π-processes is an* (ground) asynchronous bisimulation *if it is a $o\tau$-bisimulation such that $P \mathrel{\mathcal{R}} Q$ and $P \xrightarrow{a(b)} P'$ with $b \notin \mathrm{fn}(P|Q)$ imply*

- *either $Q \xrightarrow{a(b)} Q'$ and $P' \mathrel{\mathcal{R}} Q'$*
- *or $Q \xrightarrow{\tau} Q'$ and $P' \mathrel{\mathcal{R}} (Q'|\overline{a}b)$.*

Two processes P and Q are asynchronous bisimilar*, written $P \sim_a Q$, if $P \mathrel{\mathcal{R}} Q$ for some asynchronous bisimulation $\mathcal{R}$.*

In [1] some alternative characterizations of asynchronous bisimulation are proposed. One of them, namely 3-bisimulation, shows that it is possible to discover by only considering the behavior of P whether the input $P \xrightarrow{a(b)} P'$ is "redundant", and to require that only the "non-redundant" input transitions of P are matched in Q. The intuition is that an input transition is "redundant" if it is immediately followed by the emission of the received message.

Here we define a variant of 3-bisimulation, that we call 4-bisimulation. According to it, if process P performs an input $P \xrightarrow{a(b)} P'$, but it also can perform a τ transition $P \xrightarrow{\tau} P''$ such that P' and $P''|\overline{a}b$ are bisimilar, then the input transition is *redundant*, and should not be matched by an equivalent process Q.

Definition 3 (4-bisimulation). *A symmetric relation $\mathcal{R}$ on π-processes is a* 4-bisimulation *if it is a $o\tau$-bisimulation such that $P \mathrel{\mathcal{R}} Q$ and $P \xrightarrow{a(b)} P'$ with $b \notin \mathrm{fn}(P|Q)$ imply*

- *either* $Q \xrightarrow{a(b)} Q'$ *and* $P' \mathcal{R} Q'$
- *or* $P \xrightarrow{\tau} P''$ *and* $P' \mathcal{R} (P''|\overline{a}b)$.

Two processes P *and* Q *are* 4-bisimilar*, written* $P \sim_4 Q$*, if there is some 4-bisimulation* $\mathcal{R}$ *such that* $P \mathcal{R} Q$*.*

In our opinion 4-bisimulation is particularly interesting: each process can discover privately if a transition is redundant, and in when two transitions of different processes are matched, it is required that the labels are exactly the same.

Proposition 1. *Relations* $\sim_a$ *and* $\sim_4$ *coincide.*

3 "Hot-potato" bisimulation

Asynchronous bisimulation and its alternative characterizations discussed in the previous section are not amenable for finite state verification. In fact, infinite state systems are obtained for essentially all the interesting processes that can perform infinite computations. This happens since the messages generated during a computation are not forced to be emitted, even if their channels are not restricted; rather, they are simply put in parallel to the process. So, every process that continues to generate output messages, gives rise to an infinite state system.

We define now "hot-potato" bisimulation, that avoids this source of infiniteness. The key idea is to force the output particles to be emitted as soon as possible: consider process

$$P = (\boldsymbol{\nu} c)\,(\boldsymbol{\nu} e)\,(\overline{a}c|\overline{b}c|\overline{c}d|\overline{e}f|G).$$

Output particles $\overline{a}c$ and $\overline{b}c$ can be emitted directly. Particle $\overline{c}d$ can be emitted only after name c has been extruded by the emission of $\overline{a}c$ or of $\overline{b}c$. Particle $\overline{e}f$, finally, cannot be fired, since name e is restricted and there are no output particles that extrude it. In what follows, whenever we need to identify the firable output particles of a process P we use the notation $P \equiv F \lhd P'$, where F contains the firable output particles and the restrictions that are extruded by them, while P' contains the blocked output particles and the control part. So, for instance, process P can be decomposed as follows:

$$P \equiv (\boldsymbol{\nu} c)\,(\overline{a}c|\overline{b}c|\overline{c}d) \lhd (\boldsymbol{\nu} e)\,(\overline{e}f|G).$$

Up to structural congruence $\equiv$, the decomposition of P into F and P' is unique.

In hot-potato bisimulation the emission of a message takes precedence on input and synchronization transitions; that is, process P cannot perform any input or synchronization transition until messages $\overline{a}c$, $\overline{b}c$ and $\overline{c}d$ have been emitted. Moreover, rather than performing the emission of the output particles in a sequential way, the whole firable output F of $F \lhd P$ is emitted in one step.

Definition 4 (hp-bisimulation). *A symmetric relation* $\mathcal{R}$ *on* π*-processes is a* hot-potato bisimulation *(or* hp-bisimulation*) if* $P \mathcal{R} Q$ *and* $P \equiv F \lhd P'$ *with* $\mathrm{bn}(F) \cap \mathrm{fn}(P|Q) = \emptyset$ *then* $Q \equiv F \lhd Q'$ *and*

– *if* $P' \xrightarrow{\tau} P''$ *then* $Q' \xrightarrow{\tau} Q''$ *and* $P'' \mathcal{R} Q''$*;*
– *if* $P' \xrightarrow{a(b)} P''$ *and* $b \notin \mathrm{fn}(P'|Q')$ *then:*
  • *either* $Q' \xrightarrow{a(b)} Q''$ *and* $P'' \mathcal{R} Q''$ • *or* $P' \xrightarrow{\tau} P'''$ *and* $P'' \mathcal{R} (P'''|\overline{a}b)$*.*

Two processes P *and* Q *are* hp-bisimilar*, written* $P \sim_{hp} Q$*, if there is some hp-bisimulation* $\mathcal{R}$ *such that* $P \mathcal{R} Q$*.*

Theorem 1. *Relations* $\sim_a$ *and* $\sim_{hp}$ *coincide.*

4 History dependent transition systems

In this section we introduce a new operational model, the *History Dependent transition systems with Negative transitions*, HDN in brief; they are, in our opinion, more adequate than classical labelled transition systems for dealing with process calculi with name passing, like the asynchronous π-calculus.

As we have explained in the Introduction, classical labelled transition systems have difficulties in modelling the creation of fresh names: for instance, in the ordinary operational semantics of π-calculus, infinite bunches of transitions are necessary. This problem is addressed by HDN, where states and labels are enriched with sets of names, that are now an explicit component of the model. Moreover each state of a HDN is used to denote a whole family of π-processes that differ for injective renamings, and a single transition is sufficient to model the creation of a fresh name. This is obtained by representing explicitly the correspondence between the names of source, label and target of each transition; in the ordinary labelled transition system, the correspondence between these names is the syntactical identity, and this requires to distinguish states and transitions that differ for the syntactical identity of the names.

All these features are also present in HD [8]. The original element of HDN is the presence of *negative transitions*: these are used to determine whether a transition is redundant or not. The intuition is that a transition is redundant if there is a negative transition from the same state, with the same label, and such that the two target states are bisimilar. That is, a negative transition from a state cancels the "equivalent" positive transitions from that state.

Definition 5 (HDN). *A* History Dependent transition system with Negative transitions*, or HDN, is a tuple* $\mathcal{A} = (\mathcal{Q}, \mathcal{L}, \boldsymbol{\mu}, \longmapsto, \rightsquigarrow)$ *where:*

– $\mathcal{Q}$ *is a set of* states *and* $\mathcal{L}$ *is a set of* labels*; we assume that* $\mathcal{Q} \cap \mathcal{L} = \emptyset$*;*
– $\boldsymbol{\mu} : \mathcal{L} \cup \mathcal{Q} \to \mathcal{P}_{\mathrm{fin}}(\mathfrak{N})$ *associates to each state and label a finite set of names;*
– $\longmapsto$ *is the* (positive) transition relation *and* $\rightsquigarrow$ *is the* negative transition relation*; if* $Q \overset{\lambda}{\longmapsto}{}^{\sigma} Q'$ *(resp.* $Q \overset{\lambda}{\rightsquigarrow}{}^{\sigma} Q'$*) then:*
  • $Q, Q' \in \mathcal{Q}$ *are the* source *and* target *states,*
  • $\lambda \in \mathcal{L}$ *is the label,*
  • $\sigma : \boldsymbol{\mu}(Q') \hookrightarrow \boldsymbol{\mu}(Q) \cup \boldsymbol{\mu}(\lambda)$ *is an injective embedding of the names of the target state into the names of the source state and of the label.*

We assume that the set of labels is closed for injective renamings, i.e., for each label $\lambda \in \mathcal{L}$ and each injective renaming $\rho : \boldsymbol{\mu}(\lambda) \hookrightarrow \mathfrak{N}$, we assume that a label $\lambda\rho \in \mathcal{L}$ is defined. The following properties of renamings on labels must be satisfied: $\boldsymbol{\mu}(\lambda\rho) = \rho(\boldsymbol{\mu}(\lambda))$, $(\lambda\rho)\rho' = \lambda(\rho;\rho')$, and $\lambda\rho = \lambda$ if $\rho = \mathrm{id}_{\boldsymbol{\mu}(\lambda)}$.

4.1 A HDN for the asynchronous π-calculus

In this section we define the HDN Π corresponding to the asynchronous π-calculus; the "hot potato" semantics is exploited to this purpose.

In this case, the states $\mathcal{Q}_\Pi$ have two forms: they are $(0,P)$ and $(1,P)$; in a state of the form $(0,P)$ the emission of the output message has still to be performed, while in a state $(1,P)$ it has already happened and process P can perform input and synchronization transitions. In both cases, the names associated to the state are $\mathrm{fn}(P)$.

In Π all the π-processes that differ only for an injective renaming are collapsed into a single state. To this purpose, we assume to have canonical representatives for each class of processes that differ for injective renamings, and a function norm that, given a process P, returns a pair $\mathrm{norm}(P) = (Q,\sigma)$, where Q is the canonical representative of the class of processes that differ from P for an injective renaming, and $\sigma : \mathrm{fn}(Q) \hookrightarrow \mathrm{fn}(P)$ is the injective renaming such that $P = Q\sigma$.

The transitions in Π from a state $(1,P)$ correspond to the synchronization and input actions of process P. While all the τ transitions of P have to be represented in Π, it is not necessary to take all the input transitions; rather, it is sufficient to take just one canonical representative for each bunch of input transitions. In this case, a policy for allocating the fresh names has to be chosen. Since $\mathfrak{N}$ is countable, we can take the first name that does not already appear in process P whenever a transition from P requires the generation of a fresh name. So, we say that transition $P \xrightarrow{a(b)} P'$ is *canonical* if $b = \min(\mathfrak{N} \setminus \mathrm{fn}(P))$.

Whenever a process P can perform both an input transition $P \xrightarrow{a(b)} P'$ and a τ transition $P \xrightarrow{\tau} P''$, we have to take into account that the input transition is redundant if P' and $P''|\overline{a}b$ are bisimilar. To this purpose, a negative transition with label $a(b)$ is added to Π.

In Π there is exactly one transition from state $(0,P)$, that corresponds to the emission of the firable messages. If $P \equiv F \lhd P'$, then F is observed as the label of the transition. Since component F of a process P is unique only up to structural congruence, we assume to have canonical representatives for these composed output messages, and we call $P \equiv F \lhd P'$ a *canonical decomposition* if F is a canonical representative.

Notice that the names $\boldsymbol{\mu}(F)$ that correspond to label F are not only the free names of F, but also its restricted ones. So, if the injective substitution ρ is applied to F, not only the free names are changed according to ρ, but also the restricted ones.

Definition 6 (HDN for the asynchronous π-calculus). *The HDN Π for the "hot potato" asynchronous π-calculus is defined as follows:*

- $\mathcal{Q}_\Pi = \{(0,P) \mid P$ *is a canonical π-process*$\} \cup \{(1,P) \mid P$ *is a canonical π-process without firable messages*$\}$ *and* $\boldsymbol{\mu}\big((0,P)\big) = \boldsymbol{\mu}\big((1,P)\big) = \mathrm{fn}(P)$*;*
- $\mathcal{L}_\Pi = \{\tau\} \cup \{a(b) \mid a,b \in \mathfrak{N}\} \cup \{F \mid F$ *is canonical*$\}$ *and* $\boldsymbol{\mu}(\tau) = \emptyset$, $\boldsymbol{\mu}(a(b)) = \{a,b\}$, $\boldsymbol{\mu}(F) = \mathrm{fn}(F) \cup \mathrm{bn}(F)$*;*
- *if* $(0,Q) \in Q_\Pi$, $Q \equiv F \lhd Q'$ *is a canonical decomposition, and* $\mathrm{norm}(Q') = (Q'',\sigma)$*, then* $(0,Q) \xmapsto{F}{}^{\sigma}_{\Pi} (1,Q'')$*;*
- *if* $(1,Q) \in Q_\Pi$, $Q \xrightarrow{\tau} Q'$ *is a transition, and* $\mathrm{norm}(Q') = (Q'',\sigma)$*, then* $(1,Q) \xmapsto{\tau}{}^{\sigma}_{\Pi} (0,Q'')$*;*
- *if* $(1,Q) \in Q_\Pi$, $Q \xrightarrow{a(b)} Q'$ *is a canonical transition, and* $\mathrm{norm}(Q') = (Q'',\sigma)$*, then* $(1,Q) \xmapsto{a(b)}{}^{\sigma}_{\Pi} (0,Q'')$*;*
- *if* $(1,Q) \in Q_\Pi$, $Q \xrightarrow{a(b)} Q'$ *is a canonical transition,* $Q \xrightarrow{\tau} Q''$ *is a transition, and* $\mathrm{norm}(Q''|\overline{a}b) = (Q''',\sigma)$*, then* $(1,Q) \overset{a(b)}{\leadsto}{}^{\sigma}_{\Pi} (0,Q''')$.

Definition 7 (finitary processes). *Let P be an asynchronous π-process and let* $\mathrm{norm}(P) = (Q,\sigma)$*. Process P is* finitary *if and only if a finite number of states in Π are reachable from* $(0,Q)$*.*

4.2 HDN-bisimulation

In this section we define bisimulation on HDN. In this case, bisimulations cannot simply be relations on the states; they must also deal with name correspondences: a *HDN-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 HDN and δ is a partial bijection between the names of the states. The bijection is partial since bisimilar states of a HDN can have a different number of names (in fact, bisimilar π-processes can have different sets of free names).

Notation 1. *We represent with $f : A \rightharpoonup B$ a* partial bijection *from set A to set B and with $f : A \longleftrightarrow B$ a* total bijection *from set A to set B. We denote with $f;g$ the* concatenation *of f and g and with f^{-1} the* inverse *of f.*

Suppose that we want to check if states Q_1 and Q_2 are bisimilar via the partial bijection $\delta : \boldsymbol{\mu}(Q_1) \rightharpoonup \boldsymbol{\mu}(Q_2)$ and suppose that Q_1 can perform a transition $Q_1 \xmapsto{\lambda_1}{}^{\sigma_1} Q_1'$. There are two alternatives:

- State Q_2 matches the transition of Q_1 with a transition $Q_2 \xmapsto{\lambda_2}{}^{\sigma_2} Q_2'$ such that labels λ_1 and λ_2 coincide up to a bijective renaming ρ, and states Q_1' and Q_2' are still bisimilar via a partial bijection δ'. Clearly, name correspondences δ, ρ and δ' have to be related. More precisely, ρ has to coincide with δ on the names that appear both in the label and in the source state (in fact, ρ is used to extend δ to the fresh names that are introduced in the transition) and all the pairs of names that appear in δ' must appear, via the embeddings σ_1 and σ_2, either in δ or in ρ.

– Transition $Q_1 \overset{\lambda_1}{\longmapsto}^{\sigma_1} Q_1'$ is redundant, i.e., there is some negative transition $Q_1 \overset{\lambda_1'}{\rightsquigarrow}^{\sigma_1'} Q_1''$ such that labels λ_1 and λ_1' coincide up to a bijective renaming ρ, and states Q_1' and Q_1'' are bisimilar via a partial bijection δ'. Also in this case, name correspondences $\mathrm{id}_{\boldsymbol{\mu}(Q_1)}$, ρ and δ' are related.

Definition 8 (redundant transitions). *Let $\mathcal{R}$ be a symmetric set of triples on HDN $\mathcal{A}$. Transition $Q_1 \overset{\lambda_1}{\longmapsto}^{\sigma_1} Q_1'$ is* redundant *for $\mathcal{R}$, written $Q_1 \overset{\lambda_1}{\longmapsto}^{\sigma_1} Q_1' \in \mathrm{red}[\mathcal{R}]$, if there exists some negative transition $Q_1 \overset{\lambda_1'}{\rightsquigarrow}^{\sigma_1'} Q_1''$ and some $\rho : \boldsymbol{\mu}(\lambda_1) \longleftrightarrow \boldsymbol{\mu}(\lambda_1')$ such that*

– $\rho \cap (\boldsymbol{\mu}(Q_1) \times \boldsymbol{\mu}(Q_1)) = \mathrm{id}_{\boldsymbol{\mu}(Q_1)} \cap (\boldsymbol{\mu}(\lambda_1) \times \boldsymbol{\mu}(\lambda_1'))$;
– $\lambda_1' = \lambda_1 \rho$;
– $\langle Q_1', \delta', Q_1'' \rangle \in \mathcal{R}$ *for some* $\delta' \subseteq \sigma_1; (\mathrm{id}_{\boldsymbol{\mu}(Q_1)} \cup \rho); {\sigma_1'}^{-1}$.

If transition $Q_1 \overset{\lambda_1}{\longmapsto}^{\sigma_1} Q_1'$ is not redundant for $\mathcal{R}$, then we say that it is non-redundant *for $\mathcal{R}$ and we write $Q_1 \overset{\lambda_1}{\longmapsto}^{\sigma_1} Q_1' \notin \mathrm{red}[\mathcal{R}]$.*

Definition 9 (HDN-bisimulation). *A symmetric set of triples $\mathcal{R}$ on HDN $\mathcal{A}$ is a HDN-bisimulation if $\langle Q_1, \delta, Q_2 \rangle \in \mathcal{R}$ implies that for each transition $Q_1 \overset{\lambda_1}{\longmapsto}^{\sigma_1} Q_1' \notin \mathrm{red}[\mathcal{R}]$ there exists some transition $Q_2 \overset{\lambda_2}{\longmapsto}^{\sigma_2} Q_2'$ and some $\rho : \boldsymbol{\mu}(\lambda_1) \longleftrightarrow \boldsymbol{\mu}(\lambda_2)$ such that*

– $\rho \cap (\boldsymbol{\mu}(Q_1) \times \boldsymbol{\mu}(Q_2)) = \delta \cap (\boldsymbol{\mu}(\lambda_1) \times \boldsymbol{\mu}(\lambda_2))$;
– $\lambda_2 = \lambda_1 \rho$;
– $\langle Q_1', \delta', Q_2' \rangle \in \mathcal{R}$ *for some* $\delta' \subseteq \sigma_1; (\delta \cup \rho); \sigma_2^{-1}$.

Proposition 2. *If $\mathcal{R}_i$ with $i \in I$ are HDN-bisimulations for some HDN then also $\bigcup_{i \in I} \mathcal{R}_i$ is a HDN-bisimulation.*

This proposition guarantees the existence of the *largest bisimulation* for a HDN $\mathcal{A}$. We denote with $\sim_{\mathcal{A}}$ this largest HDN-bisimulation. Moreover, if $\langle Q_1, \delta, Q_2 \rangle \in \sim_{\mathcal{A}}$ then we say that states Q_1 and Q_2 are *HDN-bisimilar according to δ.*

The following theorem shows that HDN-bisimulation on Π captures exactly asynchronous bisimulation.

Theorem 2. *Let P_1 and P_2 be two π-processes and let* $\mathrm{norm}(P_1) = (Q_1, \sigma_1)$ *and* $\mathrm{norm}(P_2) = (Q_2, \sigma_2)$. *Then $P_1 \sim_a P_2$ if and only if* $(0, Q_1)$ *and* $(0, Q_2)$ *are HDN-bisimilar in Π according to $\sigma_1; \sigma_2^{-1}$.*

4.3 Iterative characterization of HDN-bisimulation

In this section we show that, for a class of finite state HDN, the largest bisimulation can be effectively built with an iterative algorithm that resembles the partition refinement techniques of classical labelled transition systems [9].

As a first step, we characterize HDN-bisimulations on HDN $\mathcal{A}$ as the pre-fixed points of a monotone functor $\Phi_{\mathcal{A}}$.

Definition 10 (functor $\Phi_{\mathcal{A}}$). *Functor $\Phi_{\mathcal{A}}$ on symmetric set of triples $\mathcal{R}$ on HDN $\mathcal{A}$ is defined as follows: $\langle Q_1, \delta, Q_2 \rangle \in \Phi_{\mathcal{A}}(\mathcal{R})$ if and only if for each $Q_1 \xmapsto{\lambda_1}{}^{\sigma_1} Q_1' \notin \mathrm{red}[\mathcal{R}]$ there exists some transition $Q_2 \xmapsto{\lambda_2}{}^{\sigma_2} Q_2'$ and some $\rho : \boldsymbol{\mu}(\lambda_1) \longleftrightarrow \boldsymbol{\mu}(\lambda_2)$ such that*

- $\rho \cap (\boldsymbol{\mu}(Q_1) \times \boldsymbol{\mu}(Q_2)) = \delta \cap (\boldsymbol{\mu}(\lambda_1) \times \boldsymbol{\mu}(\lambda_2))$;
- $\lambda_2 = \lambda_1 \rho$;
- $\langle Q_1', \delta', Q_2' \rangle \in \mathcal{R}$ *where* $\delta' \subseteq \sigma_1; (\delta \cup \rho); \sigma_2^{-1}$.

Fact 1. *Set of triples $\mathcal{R}$ is a HDN-bisimulation for $\mathcal{A}$ if and only if it is a pre-fixed point of functor $\Phi_{\mathcal{A}}$.*

Lemma 1. *Functor $\Phi_{\mathcal{A}}$ is monotone. Moreover, if the HDN $\mathcal{A}$ is finite branching, functor $\Phi_{\mathcal{A}}$ is continuous.*

The fact that functor $\Phi_{\mathcal{A}}$ is continuous for finite branching HDN (and hence in particular for finite state HDN), guarantees that the largest bisimulation $\sim_{\mathcal{A}}$ can be obtained by the iterated application of $\Phi_{\mathcal{A}}$ starting from the universal set of triples $\mathcal{U}_{\mathcal{A}} = \{\langle Q_1, \delta, Q_2 \rangle \mid Q_1, Q_2 \in \mathcal{Q},\ \delta : \boldsymbol{\mu}(Q_1) \hookrightarrow \boldsymbol{\mu}(Q_2)\}$.

Corollary 1. *Let $\mathcal{A}$ be a finite branching HDN. Then $\sim_{\mathcal{A}} = \bigcap_{n \in \mathbb{N}} \Phi^n_{\mathcal{A}}(\mathcal{U}_{\mathcal{A}})$.*

This result can be exploited to obtain an algorithm that builds $\sim_{\mathcal{A}}$ whenever $\mathcal{A}$ is a finite state HDN. However, this approach is not very efficient, since it involves the manipulation of large sets of triples: even in the case $\sim_{\mathcal{A}}$ is very small, the algorithm starts from a set of triples $\mathcal{U}_{\mathcal{A}}$ that is very large.

A similar situation also happens in the case of bisimulation for ordinary finite state transition systems: all the states are considered equivalent in the beginning, and this universal relation is refined by the repeated application of a functor. In that case, however, all the approximations built by the algorithm are equivalences and can be efficiently represented by partitions of the states in equivalence blocks. So, for instance, the initial relation is represented in a compact way by the singleton partition, where all the states are in the same block.

To develop an efficient algorithm for HDN-bisimulation, it would be important to apply partitioning-like techniques also in this context. Unfortunately, in general the approximations $\Phi^n_{\mathcal{A}}(\mathcal{U}_{\mathcal{A}})$, and in particular the largest HDN-bisimulation $\sim_{\mathcal{A}}$, are not transitively closed. Consider if fact the following very simple HDN $\mathcal{A}$, where no names are associated to states and labels:

$$\mathcal{A} \; = \; \boxed{\begin{tikzcd} P \arrow[d, maps to, "\alpha"'] & & Q \arrow[dl, maps to, "\alpha"'] \arrow[dr, maps to, squiggly, "\alpha"] & & R \\ P' & Q' & & Q'' & \end{tikzcd}}$$

It holds that $\langle P, \emptyset, Q \rangle \in \sim_{\mathcal{A}}$ (since P and Q have the same positive transitions) and that $\langle Q, \emptyset, R \rangle \in \sim_{\mathcal{A}}$ (since the only positive transition of Q is clearly redundant). It is not true, however, that $\langle P, \emptyset, R \rangle \in \sim_{\mathcal{A}}$, since R is not able to match the positive transition of P.

These problems occur since in the definition of $\Phi_{\mathcal{A}}$, as well as in the definition of HDN-bisimulation, it is not required that a non-redundant transition of Q_1 is matched by a *non-redundant* transition of Q_2. Now we define functor $\Psi_{\mathcal{A}}$ where this correspondence between non-redundant transitions is forced. Therefore, the approximations obtained by iterating functor $\Psi_{\mathcal{A}}$ are transitively closed. However, this functor differs from $\Phi_{\mathcal{A}}$ in general and, even worse, it is non-monotone (so there is no guarantee that the approximations will ever converge).

Definition 11 (functor $\Psi_{\mathcal{A}}$). *Functor $\Psi_{\mathcal{A}}$ on a symmetric set of triples $\mathcal{R}$ is defined as follows: $\langle Q_1, \delta, Q_2 \rangle \in \Psi_{\mathcal{A}}(\mathcal{R})$ if and only if for each $Q_1 \overset{\lambda_1}{\longmapsto}{}^{\sigma_1} Q_1' \notin \mathrm{red}[\mathcal{R}]$ there exists some transition $Q_2 \overset{\lambda_2}{\longmapsto}{}^{\sigma_2} Q_2' \notin \mathrm{red}[\mathcal{R}]$ and some $\rho : \boldsymbol{\mu}(\lambda_1) \longleftrightarrow \boldsymbol{\mu}(\lambda_2)$ such that*

- $\rho \cap (\boldsymbol{\mu}(Q_1) \times \boldsymbol{\mu}(Q_2)) = \delta \cap (\boldsymbol{\mu}(\lambda_1) \times \boldsymbol{\mu}(\lambda_2))$;
- $\lambda_2 = \lambda_1 \rho$;
- $\langle Q_1', \delta', Q_2' \rangle \in \mathcal{R}$ *where* $\delta' \subseteq \sigma_1; (\delta \cup \rho); \sigma_2^{-1}$.

Consider HDN $\mathcal{A}$ on the previous page and let $\mathcal{R} = \{\langle P', \emptyset, Q' \rangle\}$ and $\mathcal{S} = \{\langle P', \emptyset, Q' \rangle, \langle Q', \emptyset, Q'' \rangle\}$. Then clearly $\mathcal{R} \subseteq \mathcal{S}$. However, $\langle P, \emptyset, Q \rangle \in \Psi_{\mathcal{A}}(\mathcal{R})$ but $\langle P, \emptyset, Q \rangle \notin \Psi_{\mathcal{A}}(\mathcal{S})$, so $\Psi_{\mathcal{A}}(\mathcal{R}) \not\subseteq \Psi_{\mathcal{A}}(\mathcal{S})$. In the case of HDN $\mathcal{A}$, therefore, functor $\Psi_{\mathcal{A}}$ is not monotone.

While in general different, there are classes of finite branching HDN on which functors Φ and Ψ compute the same sequence of approximations. This situation is very convenient, since in this case the advantages of both functors hold; that is, the functor is continuous, so that the largest HDN-bisimulation is captured by iterating it; and the approximations are transitively closed, which implies that also the largest bisimulation is transitively closed. Fortunately enough, all the interesting HDN that we have considered are redundancy-consistent. In particular, this is the case of Π.

Definition 12 (redundancy-consistent HDN). *The finite branching HDN $\mathcal{A}$ is* redundancy-consistent *if $\Phi^n_{\mathcal{A}}(\mathcal{U}_{\mathcal{A}}) = \Psi^n_{\mathcal{A}}(\mathcal{U}_{\mathcal{A}})$, for all $n \in \mathbb{N}$.*

Proposition 3. *All the approximations $\mathcal{R} = \Psi^n_{\mathcal{A}}(\mathcal{U}_{\mathcal{A}})$ of a redundancy-consistent HDN $\mathcal{A}$ are transitively closed, i.e., $\langle Q_1, \delta_{12}, Q_2 \rangle \in \mathcal{R}$ and $\langle Q_2, \delta_{23}, Q_3 \rangle \in \mathcal{R}$ imply $\langle Q_1, (\delta_{12}; \delta_{23}), Q_3 \rangle \in \mathcal{R}$.*

Theorem 3. *The HDN Π is redundancy-consistent.*

Each transitively closed set of triples $\mathcal{R}$ induces a partition of the states in equivalence classes. However, to characterize $\mathcal{R}$ it is still necessary to represent all the name correspondences between all the pairs of states in the same block. Now we show that these correspondences can be represented in a compact way by exploiting *active names.* At every step of the iteration of functor $\Psi_{\mathcal{A}}$ there are names of a state that have played no active roles in the game of matching transitions, since they are not appeared yet in the labels of the transitions considered for that state. Therefore, any correspondence can exist between the "inactive" names of equivalent states.

Definition 13 (active names). *Let $\mathcal{A}$ be a HDN. The family of functions* $\mathrm{an}^n_{\mathcal{A}} : \mathcal{Q} \to \mathcal{P}_{\mathrm{fin}}(\mathfrak{N})$*, with* $n \in \mathbb{N}$*, is defined as follows:* $\mathrm{an}^0_{\mathcal{A}}(Q) = \emptyset$*, and*

$$\mathrm{an}^{n+1}_{\mathcal{A}}(Q) = \bigcup_{\left\{Q \stackrel{\lambda}{\longmapsto}^{\sigma} Q \notin \mathrm{red}[\Psi^n_{\mathcal{A}}(\mathcal{U}_{\mathcal{A}})]\right\}} \Big(\boldsymbol{\mu}(\lambda) \cup \sigma\big(\mathrm{an}^n_{\mathcal{A}}(Q')\big)\Big) \cap \boldsymbol{\mu}(Q).$$

Notice that only the transitions that are non-redundant w.r.t. $\Psi^n_{\mathcal{A}}(\mathcal{U}_{\mathcal{A}})$ are considered for computing the active names at the $(n{+}1)$-th step: only those transitions, in fact, are considered in the $(n{+}1)$-th application of functor $\Psi_{\mathcal{A}}$. Also, the intersection $- \cap \boldsymbol{\mu}(Q)$ is necessary, since a transition can introduce new names that do not appear in the source state.

The following proposition expresses the important properties of active names: any name correspondence between two equivalent states is a total correspondence between the active names of the two states; moreover, any correspondence is possible between the non-active names of two equivalent states.

Proposition 4. *Let $\mathcal{A}$ be a redundancy-consistent HDN. Then:*

1. *if $\langle P, \delta, Q\rangle \in \Psi^n_{\mathcal{A}}(\mathcal{U}_{\mathcal{A}})$ then $\delta \cap \big(\mathrm{an}^n_{\mathcal{A}}(P) \times \mathrm{an}^n_{\mathcal{A}}(Q)\big)$ is a total bijection between $\mathrm{an}^n_{\mathcal{A}}(P)$ and $\mathrm{an}^n_{\mathcal{A}}(Q)$;*
2. *if $\langle P, \delta, Q\rangle \in \Psi^n_{\mathcal{A}}(\mathcal{U}_{\mathcal{A}})$ and $\delta \cap \big(\mathrm{an}^n_{\mathcal{A}}(P) \times \mathrm{an}^n_{\mathcal{A}}(Q)\big) = \delta' \cap \big(\mathrm{an}^n_{\mathcal{A}}(P) \times \mathrm{an}^n_{\mathcal{A}}(Q)\big)$ then $\langle P, \delta', Q\rangle \in \Psi^n_{\mathcal{A}}(\mathcal{U}_{\mathcal{A}})$.*

We can exploit the properties of active names to obtain a more compact representation of $\Psi^n_{\mathcal{A}}(\mathcal{U}_{\mathcal{A}})$: only the correspondences of the active names are explicitly represented for each pair of states in the same equivalence class. There are cases in which the introduction of active names leads to a dramatic reduction of the correspondences that have to be represented explicitly. An extreme example is the universal relation $\mathcal{U}_{\mathcal{A}}$: while all the name correspondences between each pair of states appear in $\mathcal{U}_{\mathcal{A}}$, none of them has to be represented explicitly, since no name is active at this point.

Also in the cases where a large number of correspondences exist between two equivalent states, a compact representation can be found for them. In fact, let $\Delta^n_{\mathcal{A}}(P, Q)$ be the set of name correspondences that exist, according to $\Psi^n_{\mathcal{A}}(\mathcal{U}_{\mathcal{A}})$, between the active names of P and the active names of Q:

$$\Delta^n_{\mathcal{A}}(P, Q) = \big\{\delta \cap \big(\mathrm{an}^n_{\mathcal{A}}(P) \times \mathrm{an}^n_{\mathcal{A}}(Q)\big) \mid \langle P, \delta, Q\rangle \in \Psi^n_{\mathcal{A}}(\mathcal{U}_{\mathcal{A}})\big\}.$$

The following proposition shows that $\Delta^n_{\mathcal{A}}(Q, Q)$ is a permutation group on the active names of Q; it is hence sufficient to represent it by means of a set of generators. Moreover $\Delta^n_{\mathcal{A}}(P, Q)$ can be recovered, starting from any of its elements δ, by composing δ with all the elements of $\Delta^n_{\mathcal{A}}(Q, Q)$; it is hence sufficient to represent explicitly only one element of $\Delta^n_{\mathcal{A}}(P, Q)$.

Proposition 5. *Let $\mathcal{A}$ be a redundancy-consistent HDN. Then:*

1. *if $\Delta^n_{\mathcal{A}}(Q, Q)$ is a permutation group on $\mathrm{an}^n_{\mathcal{A}}(Q)$;*
2. *if $\delta \in \Delta^n_{\mathcal{A}}(P, Q)$ then $\Delta^n_{\mathcal{A}}(P, Q) = \{\delta; \delta' \mid \delta' \in \Delta^n_{\mathcal{A}}(Q, Q)\}$.*

```
1 Normalize processes $P_1$ and $P_2$. Let $(Q_i, \sigma_i) = \mathrm{norm}(P_i)$ for $i = 1, 2$.
2 Generate the part of the HDN $\Pi$ that is reachable from $(0, Q_1)$ and $(0, Q_2)$.
3 Initialization:
  3.1 For each (reachable) state $Q$ of $\Pi$, initialize an[$Q$] to the empty set.
  3.2 Initialize part to the singleton partition on the (reachable) states of $\Pi$.
  3.3 For each pair of (reachable) states $Q$ and $Q'$, initialize Delta[$Q, Q'$] to the
      empty set of name relations.
4 Repeat the following steps until partition part becomes stable:
  4.1 Compute the non-redundant transitions according to part.
  4.2 Update the sets of active names an[$Q$] for all the states $Q$.
  4.3 Refine part according to functor $\Psi_\Pi$. For each pair of states $Q$ and $Q'$
      that are still in the same block of part, put in Delta[$Q, Q'$] (a compact
      representation of) the valid relations between an[$Q$] and an[$Q'$].
5 Check if $Q_1$ and $Q_2$ are in the same class and if $\sigma_1; \sigma_2^{-1}$ is in Delta[$Q_1, Q_2$].
```

We are currently working on the implementation of an algorithm that exploits these techniques to check bisimilarity of finitary asynchronous π-processes. In the table above we sketch the main steps that have to be performed to check whether processes P_1 and P_2 are equivalent. We plan to integrate it within HAL, a verification environment for calculi with name passing [3].

We conclude this section with some comments on the complexity of the algorithm. It is not possible, in general, to find an upper bound for the number of states and transitions of the HDN corresponding to a finitary π-process P in function of the syntactical length of P: in fact this problem is equivalent to find an upper bound to the length of the tape used by a given Turing machine, which is an undecidable problem. Once the HDN is built, the complexity in time for building the largest HDN-bisimulation is polynomial in the number s of states and t of transitions of the HDN, and exponential in the maximum number n of the names that appear in the states. The polynomial complexity in s and t is typical of the partitioning algorithms: each iteration of step 4 of the algorithm refines the partition of states, and at most $s - 1$ refinements are possible, after that all the states are in different blocks. However, the algorithm has to deal with correspondences between names, and there can be up to $2^{\mathbf{O}(n \cdot \log n)}$ of those correspondences between two states, hence the algorithm is exponential in n. Even if these correspondences are represented in a compact way by means of permutation groups, the exponential in the number of names cannot be avoided: some of the operations on permutation groups used in the algorithm are in fact exponential in the number n of elements.

5 Concluding remarks

In this paper we have presented the model of *history dependent transition systems with negative transitions (HDN).* They are an extended version of labelled transition systems and are adequate for asynchronous calculi with name passing.

We have also defined a finitary characterization of bisimilarity for the π-calculus; this characterization can be modeled by HDN and, as a consequence, finite state representations can be computed for a family of π-processes.

In this paper we have considered only the asynchronous π-calculus without matching. In [10], however, HDN are applied also to the asynchronous π-calculus *with matching*, as well as to the early, late [7], and open [13,11] semantics of the π-calculus with synchronous communications.

We are also working to extend the approach described in this paper to the *weak* asynchronous bisimulation. The alternative characterization given by 4-bisimulation works also for the weak semantics: it is sufficient to replace the strong transitions $\xrightarrow{\alpha}$ with weak transitions $\xRightarrow{\alpha}$. Unfortunately, weak hot-potato bisimulation does not coincide with weak asynchronous bisimulation: it is not safe to force weak outputs to be emitted as soon as they are ready, since in this case the firing of an output can discard possible behaviors. For instance, in process $\tau.\overline{a}b + a(c).\mathbf{0}$ the input transition is not performed at all if the output transition $\tau.\overline{a}b + a(c).\mathbf{0} \xRightarrow{\overline{a}b} \mathbf{0}$ has the precedence. To apply successfully the HDN also to the weak asynchronous π-calculus it is necessary to find conditions that allow a weak output transition to be fired without discarding behaviors.

References

1. R. Amadio, I. Castellani and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. *Theoretical Computer Science*, 192(2):291–324, 1998.
2. G. Boudol. Asynchrony and the π-calculus. Research Report 1702, INRIA, Sophia-Antipolis, 1991.
3. 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.
4. C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget and D. Rémy. A calculus of mobile agents. In *Proc. CONCUR'96*, LNCS 1119. Springer Verlag, 1996.
5. K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *Proc. ECOOP'91*, LNCS 612. Springer Verlag, 1991.
6. R. Milner. The polyadic π-calculus: a tutorial. In *Logic and Algebra of Specification*, NATO ASI Series F, Vol. 94. Springer Verlag, 1993.
7. R. Milner, J. Parrow and D. Walker. A calculus of mobile processes (parts I and II). *Information and Computation*, 100(1):1–77, 1992.
8. U. Montanari and M. Pistore. An introduction to history dependent automata. In *Proc. HOOTS II*, ENTCS 10. Elsevier, 1998.
9. R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.
10. M. Pistore. *History Dependent Automata.* PhD Thesis. Dipartimento di Informatica, Università di Pisa, 1999.
11. M. Pistore and D. Sangiorgi. A partition refinement algorithm for the π-calculus. In *Proc. CAV'96*, LNCS 1102. Spinger Verlag, 1996.
12. J. Rathke. Resource based models for asynchrony. In *Proc. FoSSaCS'98*, LNCS 1378. Springer Verlag, 1998.
13. D. Sangiorgi. A theory of bisimulation for π-calculus. *Acta Informatica*, 33:69–97, 1996.

10.1007/b107031130018