URL: http://www.elsevier.nl/locate/entcs/volume51.html 17 pages

# On the decidability of fragments of the asynchronous $\pi$-calculus

Roberto M. Amadio Charles Meyssonnier [1,2]

*Laboratoire d'Informatique Fondamentale de Marseille, CMI, 39 rue Joliot-Curie, 13453, Marseille, France.*

**Abstract**

We study the decidability of a reachability problem for various fragments of the asynchronous $\pi$-calculus. We consider the combination of three main features: *name generation*, *name mobility*, and *unbounded control*. We show that the combination of name generation with either name mobility or unbounded control leads to an undecidable fragment. On the other hand, we prove that name generation without name mobility and with bounded control is decidable by reduction to the coverability problem for Petri Nets.

## 1 Introduction

We are interested in *properties* of the reduction relation such as reachability, deadlock, liveness,... for process calculi based on the *asynchronous* $\pi$-calculus [2,7,1].

We recall that 'asynchronous' here refers to a communication mechanism where messages are put in an unbounded and unordered buffer and that in the process calculus jargon this amounts to disallow the *output prefix*. By opposition, the *synchronous* $\pi$-calculus forces a synchronization between the sender and the receiver.

Our interest in the asynchronous $\pi$-calculus stems from the observation that the core of concurrent programming languages such as Pict [13], Join [4], or Tyco [17] are based on it and the remark that object-oriented programming languages enjoy a rather direct representation in these formalisms.

In this paper, we will mainly consider a *minimal* asynchronous, polyadic, simply sorted $\pi$-calculus *not* including external choice and we will concentrate on three main 'features' of this minimal calculus:

[1] {amadio,meyssonn}@cmi.univ-mrs.fr.

[2] The authors are partially supported by RNRT MARVEL.

- Name generation, *i.e.* the possibility of generating fresh names (values, channels,...).
- Name mobility, *i.e.* the possibility of transmitting names.
- Unbounded control, *i.e.* the possibility of dynamically adding new threads of control.

In the absence of name generation, our formalism can be mapped to Petri Nets (see, *e.g.* [15]). This encoding, that basically goes back to early work [5] on the translation of CCS [11] to Petri Nets, settles most interesting decision problems for the fragment *without* name generation. Therefore, the main issue that, in our opinion, remains to be clarified is whether there exist *decidable* fragments that include some form of name generation.

So far, most decidability results we are aware of concern the *synchronous* $\pi$-calculus with *bounded* control (see, *e.g.*, [3,12]). In the *asynchronous* case, our main results are as follows:

- The combination of name generation and name mobility leads to an undecidable fragment even assuming the control finite.
- The combination of name generation and unbounded control leads to an undecidable fragment even assuming that no name is transmitted (this refines a well-known undecidability result for CCS).
- Name generation without name mobility and with bounded control is decidable by reduction to Petri Nets. This is our main technical result which is based on an analysis of the use of generated names. The analysis, which appears to be original, distinguishes between 'persistent' and 'temporary' names and provides a method to reuse the same name for generated temporary names which are alive at different times.

We regard these results as a first step towards the systematic introduction of approximated decision methods for languages including name generation. We expect that a fruitful approach is to understand these methods by factoring the approximation through a translation into Petri Nets. Once the behaviour is mapped to a Petri Net further standard approximation techniques are available based, *e.g.*, on semi-linear sets (see, *e.g.*, [16], for an up to date survey).

## 2 Asynchronous $\pi$-calculus

As usual, we assume given a denumerable set of *names*, that we denote $a, b, \ldots$ Vectors of names (possibly empty) are denoted $\vec{a}, \vec{b}, \ldots$ We denote with $[\vec{b}/\vec{a}]$ a substitution on names. If $\vec{a} \equiv a_1, \ldots, a_n$ then we use $(\nu\vec{a})$ as a shorthand for $(\nu a_1) \ldots (\nu a_n)$.

We suppose that every name $a$ has an associated sort $st(a)$ and that names are used consistently with their sort. We will just rely on *simple* sorts as

defined by the following grammar

$$s ::= o \parallel Ch(s, \ldots, s) \tag{1}$$

where $o$ is some ground sort.

We consider a polyadic, simply sorted, asynchronous $\pi$-calculus with the standard operations of message creation $\overline{a}\vec{b}$, input prefix $a(\vec{b}).P$, parallel composition $P \mid Q$, name generation $(\nu a)P$, and parametric recursive definitions. The latter is preferred to *iteration* because it allows a better control on the creation and termination of parallel threads.

We denote with $A, B, \ldots$ parametric process identifiers. A *process* is presented by a finite system $\mathcal{E}$ of parametric equations $A(\vec{a}) = P$ and an initial configuration where we assume that: (i) every process identifier is defined by exactly one equation, and (ii) the names occurring free in $P$ are included in $\{\vec{a}\}$. It will be convenient to assume that every equation has the following normalised shape:

$$A(\vec{a}) = a(\vec{a'}).(\nu\vec{a''})(\Pi_{i\in I}\overline{a}_i\vec{a}_i \mid \Pi_{j\in J}A_j(\vec{a}_j)) \ . \tag{2}$$

Such an equation specifies a process that inputs a message and then generates new names, sends a number of messages, and runs a number of continuations. The sets $I$ and $J$ are assumed finite (possibly empty, in which case the parallel composition reduces to the terminated process 0). We note that in equation (2) the names $\vec{a}$, $\vec{a'}$, and $\vec{a''}$ are bound. We will assume that they are renamed so that they are all distinct.

Given a finite system of recursive equations as above, a *configuration* is a normalised process of the shape:

$$(\nu\vec{a})(\Pi_{i\in I}\overline{a_i}(\vec{a}_i) \mid \Pi_{j\in J}A_j(\vec{a}_j))$$

where as usual '$\Pi$' stands for the parallel composition. Let $P, Q$ be two configurations. We write $P \equiv Q$ if $P$ is syntactically equal to $Q$ up to renaming of bound names, permutation of name generations, and associativity and commutativity of parallel composition. We denote with $fn(P)$ the set of names occurring free in $P$.

Next we introduce the reduction relation on configurations. All we want to capture is the usual reduction rule

$$\overline{a}\vec{b} \mid a(\vec{c}).P \to [\vec{b}/\vec{c}]P$$

allowed to take place under name generation and parallel composition, up to a suitable structural equivalence. Our definition of reduction is a bit technical because it has to evaluate the actual parameters, unfold a recursive definition to find an input prefix matching a message, and then bring the name generations, the messages, and the continuations under the input prefix at top level. The advantages of this approach, is that we can then limit the structural rules to the ones stated above, give a compact normal form for configurations, and provide a simple translation to Petri Nets.

**Definition 2.1** If the equation associated to the process identifier $A$ is (2)

and

(i) $P \equiv (\nu \vec{b'})(A(\vec{b}) \mid \overline{c}(\vec{c}) \mid Q)$,

(ii) the sets $\{\vec{a}, \vec{a'}, \vec{a''}\}$ and $\{\vec{b'}\} \cup fn(P)$ are mutually disjoint,

(iii) $\sigma \equiv [\vec{b}/\vec{a}, \vec{c}/\vec{a'}]$,

(iv) and $\sigma(a) = c$

then

$$P \to (\nu \vec{b'}, \vec{a''})(\Pi_{i \in I} \sigma(\overline{a}_i \vec{a}_i) \mid \Pi_{j \in J} A_j(\sigma \vec{a}_j) \mid Q) \ . \tag{3}$$

We may wonder whether our normalised configurations can represent all usual processes of the $\pi$-calculus, say:

$$p ::= \overline{a}\vec{b} \parallel a(\vec{b}).p \parallel !(a(\vec{b}).p) \parallel (\nu a)p \parallel (p \mid p) \ .$$

Indeed, this can be easily checked. We note that, up to structural equivalence, a process $p$ can always be written as:

$$p \equiv (\nu \vec{a})(\Pi_{i \in I} \overline{a}_i \vec{a}_i \mid \Pi_{j \in J} a_j(\vec{a}_j).p_j \mid \Pi_{k \in K} !(a_k(\vec{a}_k).p_k)) \ .$$

We claim that we can build a configuration $P$ and a set of equations $\mathcal{E}$ whose behaviour is equivalent to $p$'s. We proceed by induction on the structure of $p$ to generate the set of equations. For every process $a_j(\vec{a}_j).p_j$ we introduce a fresh process identifier $A_j(\ldots)$ and the equation $A_j(\ldots) = a_j(\vec{a}_j).\ldots$, and we apply inductively the transformation to $p_j$. Similarly, for every process $!(a_k(\vec{a}_k).p_k))$ we introduce a fresh process identifier $A_k(\ldots)$ and the equation $A_k(\ldots) = a_k(\vec{a}_k).(A_k(\ldots) \mid \ldots)$, and we apply inductively the transformation to $p_k$.

Reassured about the expressivity of our formalism, we can now formally state the reachability problem we address in this paper.

**Definition 2.2** *Given a system of equations $\mathcal{E}$ containing a process identifier $A$ and a related initial configuration $P$, the reachability problem asks whether $P$ reduces to a configuration containing the process identifier $A$,* i.e. *$P \to^* (\nu \vec{a})(\ldots \mid A(\vec{b}) \mid \ldots)$, for some $\vec{a}, \vec{b}$.*

In section 3.4, we will relate this problem to the well known coverability problem for Petri Nets.

# 3 The fragment without name generation reduces to Petri Nets

We consider the fragment where the equation (2) is restricted to having the shape:

$$A(\vec{a}) = a(\vec{b}).(\Pi_{i \in I} \overline{c}_i \vec{d_i} \mid \Pi_{j \in J} A_j(\vec{e_j})) \ . \tag{4}$$

In this fragment no name generation is allowed. Given such a system of equations and an initial configuration $P$ we will recall below the standard construction of a Petri Net that simulates the reduction of the process.

### 3.1 Parameterless systems of equations

First we recall the notion of *parameterless* system of equations (a notation used, *e.g.*, in the context of CCS [11]). In this case, all names have sort $Ch()$ and an equation has the shape

$$A = \Sigma_{k\in K} a_k.(\Pi_{i\in I_k}\overline{a}_i \mid \Pi_{j\in J_k} A_j) \tag{5}$$

where $K$ is a finite set and $\Sigma$ stands for the *external choice* (external choice is just used here to represent an intermediate step towards the translation to Petri Nets). If $K$ is empty, we take conventionally the left hand side as the terminated process 0. No renaming is allowed and a process identifier is *literally* replaced by the right hand side of the equation defining it.

### 3.2 From parameterless systems of equations to Petri Nets

We fix a system of equations without parameters of the shape (5). Let $P$ be an initial configuration. Without loss of generality, we may assume that $P$ contains no name generators $\nu$; otherwise we replace the names bound by $\nu$ by fresh names. Let $N$ be the collection of names free in $P$. Since there is no name generation, these are all the names that can appear in a reachable configuration.

(1) We associate a distinct place to every name $a \in N$ and to every process identifier $A$. The intended interpretation is that a token at place $a$ corresponds to a message $\overline{a}$ while a token at place $A$ means that the control of a thread is at $A$. Following this interpretation we determine the initial marking.

(2) To every equation we associate a set of transitions which are connected to the places as follows. If $A = a_1 \ldots + \cdots + a_n \ldots$ then we introduce $n$ transitions $t_1, \ldots, t_n$ and for $k = 1, \ldots, n$ an edge from place $A$ to transition $t_k$ and an edge from place $a_k$ to transition $t_k$. Moreover, if the continuation of $a_k$ has the shape

$$(\Pi_{i\in I}\overline{a_i} \mid \Pi_{j\in J} A_j)$$

then we add an edge from transition $t_k$ to place $a_i$ for $i \in I$ and from transition $t_k$ to place $A_j$ for $j \in J$.

### 3.3 From systems without name generation to parameterless systems

We fix a system of parametric equations without name generation of the shape (4). For the sake of notational simplicity we assume that all channels have a recursive sort $s = Ch(s)$, and that all process identifiers depend on $k$ parameters. Then:

- for every pair of channel names $a, b \in N$, we introduce a new channel name $a_b$ of sort $Ch()$.
- for every equation of the shape (4) and for every vector of names $\vec{a'} \in N^k$

we produce an equation

$$A_{\vec{a'}} = \Sigma_{b' \in N}(\sigma(a)_{b'}.(\Pi_{i \in I}\overline{\sigma(c_i)_{\sigma(d_i)}} \mid \Pi_{j \in J} A_{j,\sigma(\vec{e}_j)})) \ .$$

where $\sigma \equiv [\vec{a'}/\vec{a}, b'/b]$.

To summarize, we transform a parametric system into a system without parameters but with external choice, and in turn, we transform the latter system into a Petri Net.

### *3.4 From reachability to coverability, and back*

In terms of Petri Nets, the reachability problem we have formulated in definition 2.2 amounts to checking whether certain places, corresponding to a given process identifier, will contain a token. This is an instance of the *coverability* problem for which Lipton [10] has provided a $2^{O(\sqrt{n})}$ space lower bound and Rackoff [14] a $2^{O(n \log n)}$ space upper bound.

On the other hand, it is easy to see that the coverability problem for Petri Nets can be reduced to the reachability problem 2.2. Given a Petri Net, for every transition $t$ taking, say, one token from places $a_1, \ldots, a_n$ and putting one token in places $b_1, \ldots, b_m$, we introduce the equations (we omit the parameters):

$$A_t = a_1.A_t^1 \quad A_t^1 = a_2.A_t^2 \ldots \quad A_t^{n-1} = a_n.(\overline{b}_1 \mid \cdots \mid \overline{b}_m \mid A_t)$$

Thus a transition of the Petri Net is now simulated by serialising the reading of the tokens. If we want to know, whether, say, the place $a$ will ever contain a token we add the equation $A = a.B$. Then the initial configuration contains the process identifier $A_t$ for every transition $t$, a number of messages corresponding to the initial marking, and the process identifier $A$. To determine whether the place $a$ will contain a token it is then enough to check whether the initial configuration reaches one containing the process identifier $B$.

This reduction is polynomial and it shows that even without mobility and without name generation the reachability problem 2.2 we consider requires exponential space. We expect that our reachability problem could be generalized mimicking what has been done for Petri Nets [18]. On the other hand, the quest for decidability results on the equivalence problem (trace, bisimulation,...) is discouraged by the negative results known for Petri Nets [6,9].

## 4 The fragment with bounded control is undecidable

We say that a configuration has *bounded control* if there is a natural number that bounds the number of live threads running in parallel in any accessible configuration. One can imagine various syntactic conditions that imply this property and are efficiently checkable. To show our negative results, it will be enough to consider the fragment where the equation (2) is restricted to having

the shape:

$$A(\vec{a}) = a(\vec{b}).(\nu\vec{d})(\Pi_{i\in I}\overline{a}_i\vec{b}_i \mid A'(\vec{c}))$$
$$A(\vec{a}) = A_1(\vec{a}_1) \oplus A_2(\vec{a}_2) \ .$$

where $\oplus$ denotes the *internal choice*. This means that, up to internal choice, every control point has exactly one continuation and thus the control is basically *bounded* by the number of parallel threads present in the initial configuration.

**Remark 4.1** It is well known that internal choice is *definable* from parallel composition and name generation. In our case, there is just a little twist to fit the shape of the normalised equations (2). Thus we replace the equation $A(\ldots) = A_1(\ldots) \oplus A_2(\ldots)$ by the equations

$$A(\ldots) \ = t.(\nu c)(A'_1(c,\ldots) \mid A'_2(c,\ldots) \mid \overline{c} \mid \overline{t})$$
$$A'_i(\ldots) = c.A_i(\ldots) \quad \text{for } i = 1,2$$

where $t$ is a 'global' channel provided in the initial configuration with a message $\overline{t}$ (the $t$ channel plays the role of the CCS $\tau$ action).

A similar trick applies if we want to define the internal choice of two messages $\overline{a_1} \oplus \overline{a_2}$. Then we introduce an identifier $A$ and the equations:

$$A(\ldots) \ = t.(\nu c)(A'_1(c,\ldots) \mid A'_2(c,\ldots) \mid \overline{c} \mid \overline{t})$$
$$A'_i(\ldots) = c.\overline{a_i} \quad \text{for } i = 1,2 \ .$$

**Proposition 4.2** *The reachability problem for the fragment with bounded control is undecidable.*

**Proof.** The proof is loosely inspired by the encoding of the computation mechanism of Turing machines into a deduction system for Horn clauses without function symbols, also known as DATALOG. Readers familiar with the latter might find it inspiring to look at an 'existential' Horn clause $\forall\vec{x}\,(a(\vec{x}) \supset \exists\vec{y}\,b(\vec{x},\vec{y}))$ as a recursive process $A = a(\vec{x}).(\nu\vec{y})(\overline{b}(\vec{x},\vec{y}) \mid \overline{a}(\vec{x}) \mid A)$.

We now turn to the technical development. We simulate a 2-counter machine (see, *e.g.* [8]) and reduce the halting problem to the reachability problem 2.2. We assume that the 2-counter machine contains instructions of the form:

$$(1)\ q : C_k := C_k + 1;\ \ goto\ q'$$
$$(2)\ q : (C_k = 0) \to goto\ q', C_k := C_k - 1; goto\ q''$$

where $C_1, C_2$ denote the two counters. An instruction of type (1) increments the counter $k$ and jumps to another point of the control. An instruction of type (2) tests whether the counter $C_k$ is 0 and if it is the case it jumps to a control point $q'$, otherwise it decrements the counter and jumps to control point $q''$.

A counter is represented as a stack of cells where the bottom cell contains 0 and all the others contain 1. Thus the value 2 is represented by the stack

011. For every state, we assume a channel $q$ of sort $Ch()$. Moreover, for every counter $C_k$ we assume channels

$$Top_k \text{ of sort } Ch(Ch(Ch(), Ch(), Ch())) \qquad \text{and}$$
$$Adj_k \text{ of sort } Ch(Ch(Ch(), Ch(), Ch()), Ch()) \ .$$

Every cell of the stack is assigned a distinct channel $a$ of sort $Ch(Ch(), Ch(), Ch())$. We associate to every such channel three more distinct channels $a_0, a_1, a_t$ and a message $\overline{a}(a_0, a_1, a_t)$. Moreover:

- If the channel $a$ refers to the bottom cell then we introduce a message $\overline{a_0}$, and otherwise we introduce a message $\overline{a_1}$.
- If the channel $a$ refers to the cell at the top of the stack we introduce a message $\overline{Top_k}a$.
- If the channels $a$ and $b$ refer to two adjacent cells (the first under the second) then we introduce a message $\overline{Adj}_k(a, b_t)$.

For instance, the stack 011 could be represented by the following messages:

$$\overline{a}(a_0, a_1, a_t) \mid \overline{a_0} \mid \overline{Adj}_k(a, b_t) \mid \text{(bottom cell)}$$
$$\overline{b}(b_0, b_1, b_t) \mid \overline{b_1} \mid \overline{Adj}_k(b, c_t) \mid \text{(second cell)}$$
$$\overline{c}(c_0, c_1, c_t) \mid \overline{c_1} \mid \overline{Top_k}c \qquad \text{(top cell)}$$

We now consider the problem of implementing on this data structure the 2-counter machine operations. An instruction of type (1) is translated as:

$$A = q.Top_k(a).(\nu a', a'_0, a'_1, a'_t)(\overline{q}' \mid \overline{Adj_k}(a, a'_t) \mid \overline{Top_k}(a') \mid \overline{a'}(a'_0, a'_1, a'_t) \mid \overline{a'_1} \mid A),$$

and an instruction of type (2) becomes:

$$\begin{aligned} A = {} & q.Top_k(a).a(a_0, a_1, a_t). \\ & (a_0.(\overline{q'} \mid \overline{Top_k}(a) \mid \overline{a}(a_0, a_1, a_t) \mid \overline{a_0} \mid A) \oplus \qquad (\text{if } C_k = 0) \\ & a_1.Adj_k(b, b_t).(\overline{a_t} \mid b_t.(\overline{q''} \mid \overline{Top_k}(b) \mid A))) \qquad (\text{if } C_k > 0) \ . \end{aligned}$$

Note that in the equations above we have omitted the parameters (which can be easily inferred) as well as the intermediate process identifiers. The case $(C_k > 0)$ reveals the role of the channel $a_t$: it is used to simulate via a communication an equality test between $a_t$ and $b_t$ so as to make sure that the received channel $b$ corresponds to the cell preceding $a$'s. □

### *4.1 Undecidability with generated values and conditional*

The encoding above relies on channel mobility and moreover processes may input on received channel names. A frequently used extension of the $\pi$-calculus includes a *conditional* on name equality. To formalise this extension, we assume equations may have the shape:

$$A(\vec{a}) = [a = a']A'(\vec{a'}), A''(\vec{a''}) \tag{6}$$

with the expected meaning that we branch on $A'$ if $a \equiv a'$ and on $A''$ otherwise.

Now if we allow a conditional on names of basic sort $o$ then a simpler encoding is possible where all transmitted names have sort $o$. We assume additional channels $Cont_k$ to indicate the contents of a cell (values 0 or 1). The sorts are now as follows:

$$Top_k \text{ of sort } Ch(o),\ Adj_k \text{ of sort } Ch(o,o),\ \text{and } Cont_k \text{ of sort } Ch(o,o)\ .$$

An instruction of type (1) is translated as:

$$A = q.Top_k(a).(\nu a')(\overline{q'} \mid \overline{Adj_k}(a,a') \mid \overline{Cont_k}(a',1) \mid \overline{Top_k}(a') \mid A),$$

and an instruction of type (2) is translated as:

$$\begin{aligned} A = {} & q.Top_k(a).Cont_k(a',v).[a'=a] \\ & ([v=0](\overline{q'} \mid \overline{Top_k}(a) \mid \overline{Cont_k}(a,0) \mid A), \\ & Adj_k(a',a'').[a''=a](\overline{q'} \mid \overline{Top_k}(a') \mid A))\ . \end{aligned}$$

## 5 The fragment without name mobility is undecidable

We consider the fragment where all names have sort $Ch()$, *i.e.*, no name mobility is allowed. Then the equation (2) is restricted to having the shape:

$$A(\vec{a}) = a.(\nu \vec{d})(\Pi_{i\in I}\overline{a}_i \mid \Pi_{j\in J}A_j(\vec{c}_j))\ . \tag{7}$$

In the absence of name mobility, generated names cannot be extruded and therefore name generation is essentially CCS *restriction*. Milner [11] shows that *synchronous* CCS with *restriction*, *relabelling*, and *external choice* is powerful enough to simulate a 2-counter machine. We will show that this simulation can be still carried on while dropping external choice and relabelling and using just *asynchronous* communication. Schematically, we replace (i) synchronous communication by asynchronous communication plus an acknowledgement, (ii) external choice by internal choice (of course, this is possible because we are just looking at a reachability property), and (iii) relabelling by parametric equations.

**Proposition 5.1** *The reachability problem for the fragment with name generation and without name mobility is undecidable.*

**Proof.** Again we simulate a 2-counter machine in the form described in the proof of proposition 4.2 and reduce the halting problem to the reachability problem 2.2. The basic issue is to represent a stack. To this end we define the following system of equations (inspired by [11]). The channel $i$ stands for increment, $z$ for counter is zero, and $d$ for decrement. Each of these channels comes with a corresponding 'acknowledgement' channel $i^a$, $z^a$, and $d^a$ which

are kept implicit below.

$$
\begin{aligned}
B(i,z,d) &= B_i(i,z,d) \oplus B_z(i,z,d)\\
B_i(i,z,d) &= i.(\overline{i}^a \mid CB(i,z,d))\\
B_z(i,z,d) &= z.(\overline{z}^a \mid B(i,z,d))
\end{aligned}
$$

$$
\begin{aligned}
C(i,z,d,z',d') &= C_i(i,z,d,z',d') \oplus C_d(i,z,d,z',d')\\
C_i(i,z,d,z',d') &= i.(\overline{i}^a \mid CC(i,z,d,z',d'))\\
C_d(i,z,d,z',d') &= d.((\overline{d'} \oplus \overline{z'}) \mid D(i,z,d,z',d'))
\end{aligned}
$$

$$
\begin{aligned}
D(i,z,d,z',d') &= D_d(i,z,d,z',d') \oplus D_z(i,z,d,z',d')\\
D_d(i,z,d,z',d') &= (d'^a.(\overline{d}^a \mid C(i,z,d,z',d')))\\
D_z(i,z,d,z',d') &= (z'^a.(\overline{d}^a \mid B(i,z,d)))
\end{aligned}
$$

$$
\begin{aligned}
CB(i,z,d) &\equiv (\nu i'',z'',d'')(C(i,z,d,z'',d'') \mid B(i'',z'',d''))\\
CC(i,z,d,z',d') &\equiv (\nu i'',z'',d'')(C(i,z,d,z'',d'') \mid C(i'',z'',d'',z',d'))\ .
\end{aligned}
$$

A process $C$ receives on $i,d$ and sends on $z',d'$. A process $B$ receives on $i,z$. When decrementing, a process $C$ sends messages to its neighbour. The message goes on $d$ if the neighbour is $C$ and on $z$ if the neighbour is $B$. Here is a schematic intuition of what happens:

$$
\begin{aligned}
&DCCCBB \to DDCCBB \to DDDCBB \to DDDDBB \to\\
&DDDBBB \to DDCBBB \to DCCBBB \to CCCBBB\ .
\end{aligned}
$$

The $D$ is propagated towards the right till it meets $B$ and when this happens it becomes $B$ and shortcuts the last $B$.

Note the peculiar way in which we use the internal choice. If a 'server' can receive requests on two channels then it guesses non-deterministically on which channel the next message is coming. Symmetrically, a 'client' with two requests internally guesses which request is going to be served. If client and server guess consistently we obtain the desired behaviour. Otherwise client and server get stuck.

We translate a program of a 2-counter machine as a 'finite' control process that acts as a client for two counters' processes initialised by:

$$B(i_1,z_1,d_1) \mid B(i_2,z_2,d_2)\ .$$

The instructions of type (1) and (2) are simulated as follows:

$$(1)\ A_q = q.(\overline{i_k} \mid i_k^a.(\overline{q'} \mid A_q)) \ ,$$

$$\begin{aligned}(2)\ A_q &= A_q^z \oplus A_q^d \\ A_q^z &= q.(\overline{z}_k \mid z_k^a.(\overline{q'} \mid A_q)) \\ A_q^d &= q.(\overline{d}_k \mid d_k^a.(\overline{q''} \mid A_q)) \ .\end{aligned}$$

It is clear that by a suitable selection of internal choices we can simulate the behaviour of the 2-counter machine. On the other hand, suppose an attempted communication gets stuck because of wrong internal choices. This may happen (i) when the control sends a request to a counter, or (ii) when a decrement instruction propagates towards the right in a counter. In both cases the control is stuck. In the first case this is clear, in the second case this happens because the control waits for an acknowledgement which is delivered only after the propagation is completed. □

**Remark 5.2** *In all the equations above, an input is followed, up to internal choice, by exactly one output. This implies that the number of messages present in a reachable configuration is* bounded.

## 6 The fragment without mobility and with bounded control is decidable

We consider the fragment where all names have the sort $Ch()$, and the equation (2) is restricted to the shape:

$$(8) \qquad A(\vec{a}) = a.(\nu \vec{d})(\Pi_{i \in I} \overline{a_i} \mid B(\vec{b})) \ .$$

For the sake of simplicity, we assume that all the equations in a given system depend on $k$ parameters. We note that in systems without name mobility and with bounded control there is a bound on the number of 'live' names appearing in any reachable configuration. Indeed, the only form of name transmission allowed in these systems is *via* the recursion parameters: once a name disappears from the recursion parameters, no input can ever be performed on that name again. Therefore, without loss of generality we suppose that in the equation (8) above $\{\vec{d}\} \subseteq \{\vec{b}\}$.

The basic idea is to generalise the reduction to Petri Nets presented in section 3 and to replace name generation by the reusing of 'dead' names. We will begin by transforming the system into an equivalent parameterless system of equations *with reset* (and without name generation), which in turn we transform into a Petri Net with reset arcs. The latter can be reduced to a standard Petri Net, provided that the number of tokens in resetable places is bounded (in general Petri Nets with reset arcs are undecidable).

In the following, a parameterless system of equations with reset is a variant of the parameterless system presented in section 3.1. In such a system, the equations have the shape

$$A = a.\mathrm{reset}\,\vec{d}.(\Pi_{i\in I}\overline{a_i} \mid B)\ , \tag{9}$$

and the semantics of the *reset* operator is to erase all messages sent on names belonging to its argument.

### *6.1 Lifetime analysis of names*

In order to reduce a Petri Net with reset arcs to a standard Petri Net, we need a bound on the number of tokens in any resetable place. This leads us to distinguishing two kinds of names in the original system: *persistent* names, for which there is no bound, but which never need to be reset, and *temporary* names. We will give a bound on the number of messages sent on any temporary name.

To this end, we introduce the *parameter flow graph* of the system, which is defined as follows.

**Definition 6.1** *The* parameter flow graph *of a system $\mathcal{E}$ is a directed graph $\mathcal{G} = (\mathcal{L}, \mapsto)$, where:*

- *The set of nodes $\mathcal{L}$ is given by the* parameter positions $\{A^i \mid A$ *identifier in* $\mathcal{E}$ *and* $i \in [1,k]\}$.
- $A^i \mapsto B^j$ *is an edge of the graph if if the equation associated to $A$ in the system $\mathcal{E}$ is*

$$A(\vec{a}) = a.(\nu\vec{d})(\ldots \mid B(\vec{b}))\ ,$$

*and the $i$-th component of $\vec{a}$ is equal to the $j$-th component of $\vec{b}$.*

Positions leading to a cycle in $\mathcal{G}$ will be referred to as *persistent positions*, while the others will be called *temporary positions*. Accordingly, when a name occurs in $A(\vec{a})$ we will call that name persistent if it is used in at least one persistent position, and temporary if it is used only in temporary positions.

Note the peculiar structure of $\mathcal{G}$: if we consider the class of positions associated to one process identifier, all edges from vertices in this class lead to vertices in a unique class, due to our syntactic definition of finite control. Also, since all names in $\{\vec{a}\}$ are distinct, we cannot have, for $i \neq j$, $A^i \mapsto B^l$ and $A^j \mapsto B^l$. It follows from these observations that the set of vertices reachable from a temporary position is a finite tree. If $e$ is the number of equations in $\mathcal{E}$ then the size of the tree is bounded by $e \cdot k$ which is the number of parameter positions. Moreover, if $m$ is the maximum number of outputs on any parameter in any equation of $\mathcal{E}$, then the number of outputs performed on *any* temporary name is bounded by $e \cdot k \cdot m$, which is polynomial in the size of $\mathcal{E}$.

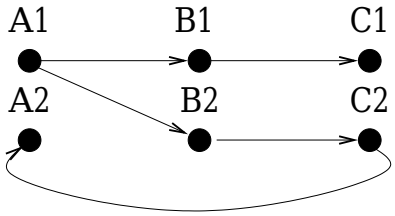


Fig. 1. The parameter flow graph for $\mathcal{E}$

**Example 6.2** *Let us consider the system $\mathcal{E}$ defined by the equations*

$$\begin{aligned} A(a,b) &= b.(\overline{a} \mid B(a,a)) \\ B(a,b) &= a.C(a,b) \\ C(a,b) &= b.(\nu c)(\overline{c} \mid A(c,a)) \ , \end{aligned}$$

*and the initial configuration*

$$P \equiv \overline{a} \mid \overline{a} \mid \overline{b} \mid A(a,b) \ .$$

*In this system, all newly generated names are temporary names used in position $A^1$. Since the tree rooted in $A^1$ has 6 nodes, and no equation in $\mathcal{E}$ performs more than 1 output on any of its parameters, we can take 6 as a bound on the number of messages sent on any temporary name.*

### *6.2 From systems without mobility and with bounded control to parameterless systems with reset*

We fix a system $\mathcal{E}$ of equations of the type (8), and an initial configuration $P$, which does not contain any generated names. Let $N_0$ be the set of names free in $P$, and $n$ the number of process identifiers in $P$. Without loss of generality, we may suppose that every process identifier in $\mathcal{E}$ relates to a *unique* thread of the initial configuration (if process identifiers are shared among different threads then we can always rename them so as to satisfy this condition).

We will construct a system $\mathcal{E}'$ of equations of the shape (9) and show that the reachability problem for $\mathcal{E}$ and $P$ reduces to a finite number of reachability problems for $\mathcal{E}'$ and a suitable initial configuration $P'$.

We assume, for every $j \in [1,n]$, pairwise disjoint sets $P_j$ and $T_j$, of respective cardinalities $k$ and $2k$, which will represent the $j$-th thread's private name space ($P_j$ is used for persistent names and $T_j$ for the temporary ones). The parameterless system $\mathcal{E}'$ will be defined over the name space $N = N_0 \cup (\cup_{j\in[1,n]} P_j \cup T_j)$.

**Definition 6.3** The vector of names $(a_1, \ldots, a_k)$ is compatible with the process identifier $A$ of the $j^{th}$ thread (written $(a_1, \ldots, a_k) \downarrow A, j$) if for all $a \in$

$\{a_1, \ldots, a_k\}$

$$a \in \begin{cases} N_0 \cup P_j \text{ if } \exists\, i \ \ (a_i = a \text{ and } A^i \text{ is a persistent position}) \\ N_0 \cup T_j \text{ otherwise.} \end{cases}$$

Next we define the system $\mathcal{E}'$ associated to $(P, \mathcal{E})$.

**Definition 6.4** Fix an equation of the shape (8) in $\mathcal{E}$ relating, say, to the thread $j$. Then:

(i) for every $\vec{a'}$ such that $\vec{a'} \downarrow A, j$,

(ii) for every *injective* substitution $[\vec{d'}/\vec{d}]$ such that $\{\vec{d'}\} \subseteq P_j \cup T_j$, $\{\vec{d'}\} \cap \{\vec{a'}\} = \emptyset$, $\sigma = [\vec{a'}/\vec{a}, \vec{d'}/\vec{d}]$, and $\sigma\vec{b} \downarrow B, j$

we introduce an equation

$$A_{\vec{a'}} = \sigma(a).\mathrm{reset}\, \vec{r}.(\Pi_{i \in I'} \sigma a_i \mid B_{\sigma\vec{b}})$$

where $\{\vec{r}\} = T_j \backslash \{\sigma\vec{b}\}$ and $I' = \{i \in I \mid \sigma a_i \notin \{\vec{r}\}\}$.

Roughly, we consider all compatible instances $\vec{a'}$ of a process identifier $A$ of a thread $j$, we replace the generated names $\vec{d}$ by unused names in $P_j \cup T_j$ (a simple cardinality argument show that they exist), and we reset all the channels on temporary names that are not used in the continuation.

Next, we introduce a binary relation $\mathcal{R}$ relating configurations and parameterless configurations.

**Definition 6.5** Let $Q \equiv (\nu\vec{d_0})(\nu\vec{d})(\Pi_{i \in I}\overline{a}_i \mid \Pi_{j \in [1,n]} A_j(\vec{a}_j))$ be a configuration where we assume that:

(i) $\{\vec{d_0}\} \cap (\bigcup_{j \in [1,n]} \{\vec{a}_j\}) = \emptyset$,

(ii) the identifier $A_j$ relates to the $j^{th}$ thread, and

(iii) if $d \in \{\vec{d}\}$ then $d$ occurs in *exactly one* set of parameters $\{\vec{a_j}\}$.

Then $Q\ \mathcal{R}\ (\Pi_{i \in I'} \sigma a_i \mid \Pi_{j \in [1,n]} A_{j,\sigma\vec{a}_j})$ whenever:

- $I' = \{i \in I \mid a_i \in N_0 \cup (\bigcup_{j \in [1,n]} \{\vec{a}_j\})\}$ and
- $\sigma$ is an *injective* substitution from $\vec{d}$ to $\bigcup_{j \in [1..n]} (T_j \cup P_j)$ such that $\sigma\vec{a}_j \downarrow A_j, j$, for $j \in [1, n]$.

Here we follow the same approach as in the previous definition 6.4: we replace the generated names occurring in the parameters of exactly one process identifier by compatible names in the set $T_j \cup P_j$, while removing useless restrictions and messages.

Given an initial configuration $P$, we can easily compute a $P'$ such that $P\ \mathcal{R}\ P'$. Then we have to check that the relation $\mathcal{R}$ is sufficiently general to keep the two configurations in lockstep.

**Lemma 6.6** *If $Q\ \mathcal{R}\ Q'$, we have:*

- *if $Q \to R$ then there is $R'$ such that $R\ \mathcal{R}\ R'$ and $Q' \to R'$.*

- *if* $Q' \to R'$ *then there is* $R$ *such that* $R \mathcal{R} R'$ *and* $Q \to R$.

The proof of this lemma is a simple, although laborious, manipulation of definitions 2.1, 6.4, and 6.5. We can then reduce the reachability problem for $(P, \mathcal{E})$ to a finite number of reachability problems for $(P', \mathcal{E}')$.

**Proposition 6.7** *The reachability of the process identifier* $A$ *in* $(P, \mathcal{E})$ *is equivalent to the reachability of one of the (finitely many) parameterless identifiers* $A_{\vec{a}}$ *in* $(P', \mathcal{E}')$.

**Proof.** We apply lemma 6.6 inductively on the length of the considered reduction chain and exploit the definition of the relation $\mathcal{R}$. □

### *6.3 From parameterless systems with reset to Petri Nets with reset arcs*

In this section, we show how to extend the reduction from parameterless systems to Petri Nets described in section 3.2, to a reduction from parameterless systems with *reset* to Petri Nets with *reset arcs*.

We suppose given a parameterless system of equations with reset $\mathcal{E}'$, and an initial configuration $P'$ without name generation. Let $N$ be the finite name space over which the system is defined (note that this may be strictly larger than the collection of names free in $P'$).

Like in section 3.2, we build a Petri Net that has one place for each parameterless process identifier in $\mathcal{E}'$, and one place for each name in $N$ (remember that we do not consider mobility). The intended interpretation is still that a token in place $a$ corresponds to a message $\overline{a}$, while a token in place $A$ corresponds to the presence of a parameterless process identifier $A$ in the current configuration.

The transitions are set as in section 3.2, except that we no longer have to care for external choice (*i.e.* there is only one transition per equation), and that if the equation associated to $A$ is $A = \ldots \operatorname{reset} \vec{r} \ldots$, then for each $a \in \{\vec{r}\}$ we add a reset arc going from transition $t_A$ to the place $a$.

Note that, thanks to the analysis performed in subsection 6.1, we can guarantee that all the places pointed to by reset arcs are bounded.

**Proposition 6.8** *The reachability of* $A$ *in* $(P', \mathcal{E}')$ *is equivalent to the coverability of place* $A$ *with* 1 *token in the Petri Net with reset arcs described above.*

### *6.4 From Petri Nets with reset arcs to Petri Nets*

Finally, we recall how to simulate a Petri Net with reset arcs $\mathcal{N}$ with a standard Petri Net $\mathcal{N}_0$, provided that all resetable places are bounded (this is a standard result for Petri Nets).

For each resetable place $p$, we add a complementary place $p'$. If $b$ is the bound on the number of tokens in place $p$, in all reachable markings $M$ we will maintain the invariant $M(p) + M(p') = b$.

To this end, we modify the existing transitions so as to add as many outgoing arcs to $p'$ as the number of incoming arcs from $p$, and as many incoming arcs from $p'$ as the number of outgoing arcs to $p$.

Then, for any transition $t$ that points a reset arc at $p$, we replace $t$ by the transitions $t_0, \ldots, t_b$, where, $t_i$ is connected to the places of the net by the same arcs as $t$, plus an arc of weight $i$ incoming from $p$, an arc of weight $b-i$ incoming from $p'$, and an arc of weight $b$ outgoing to $p'$.

**Proposition 6.9** *A marking $M$ is reachable in $\mathcal{N}$ if and only if the marking $M'$ is reachable in $\mathcal{N}_0$, where*

- *for any place $p$ of $\mathcal{N}$, $M'(p) = M(p)$,*
- *and for any resetable place $p$ of $\mathcal{N}$, $M'(p') = b - M(p)$.*

To summarize, given a system in the fragment without mobility and with bounded control, by composing the three reductions presented above, we reduce the reachability problem for that system to a finite number of coverability problems for a standard Petri Net.

**Theorem 6.10** *The reachability problem for the fragment without mobility and with bounded control is decidable.*

Our decision result could be extended from equations of the shape (8) to equations of the following shape:

$$A(\vec{a}) = a.(\nu\vec{c})(\Pi_{i\in I}\overline{a}_i \mid B(\vec{b}) \mid \Pi_{j\in J}A_j) \tag{10}$$

where $A_j$ are *parameterless* process identifiers that refer to parameterless equations of the shape (5) whose free names do not intersect the generated names $\vec{c}$.

An interesting open problem, concerns the decidability of the fragment with name generation, bounded control, and *weak* forms of name mobility where, *e.g.*, a process cannot receive on received names.

## References

[1] R. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous $\pi$-calculus. *Theoretical Computer Science*, 195:291–324, 1998.

[2] G. Boudol. Asynchrony and the $\pi$-calculus. Technical report, *RR 1702, INRIA, Sophia-Antipolis*, 1992.

[3] M. Dam. Model checking mobile processes. *Information and Computation*, 1996. Preliminary version appeared in Proc. Concur'93.

[4] C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy. A calculus of mobile agents. In *Proc. CONCUR 96, Springer Lect. Notes in Comp. Sci. 1119*, 1996.

[5] U. Golz and A. Mycroft. On the relationship of CCS and Petri Nets. *Proc. ICALP84, Springer Lect. Notes in Comp. Sci. 172:196–208*, 1984.

[6] M. Hack. *Decidability questions for Petri Nets.* Garland publishing Co., 1979.

[7] K. Honda and M. Tokoro. An object calculus for asynchronous communication. *Proc. ECOOP 91, Geneve, Springer Lect. Notes in Comp. Sci. 612*, pages 133–147, 1991.

[8] J. Hopcroft and J. Ullman. *Introduction to automata theory, languages, and computation.* Addison-Wesley, 1979.

[9] P. Jancar. Undecidability of bisimilarity for Petri Nets and related problems. *Theoretical Computer Science*, 148:281–301, 1995.

[10] R. Lipton. The reachability problem requires exponential space. Technical Report TR 66, Yale University, 1976.

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

[12] U. Montanari and M. Pistore. Checking bisimilarity for finitary $\pi$-calculus. In *CONCUR '95, Springer Lect. Notes in Comp. Sci. 962*, 1995.

[13] B. Pierce and D. Turner. Pict: a programming language based on the $\pi$-calculus. University of Cambridge, 1996.

[14] C. Rackoff. The covering and boundedness problem for vector addition systems. *Theoretical Computer Science*, 6:223–231, 1978.

[15] C. Reutenauer. *Aspects mathématiques des réseaux de Petri.* Masson Editeur, 1988. Also available in english: The mathematics of Petri Nets, Prentice-Hall.

[16] G. Sutre. *Abstraction et accéleration de systèmes infinis.* PhD thesis, ENS Cachan, 2000.

[17] V. Vasconcelos and R. Bastos. Core-TyCO, the language definition, version 0.1. Technical report TD98-3, University of Lisbon, 1998.

[18] H. Yen. A unified approach for deciding the existence of certain Petri Nets paths. *Information and Control*, 96(1):119–137, 1992.