APPENDIX 1.2.3

R. Amadio and C. Meyssonnier. On decidability of the control reachability problem in the asynchronous pi-calculus. *Nordic Journal of Computing*, 9(2):70–101, 2002.

LIF

Laboratoire d'Informatique Fondamentale
de Marseille

Unité Mixte de Recherche 6166
CNRS - Université de Provence - Université de la Méditerranée

On decidability of the control reachability problem in the asynchronous π-calculus

Roberto M. Amadio, Charles Meyssonnier

Rapport/Report 04-2002

31 Mai, 2002

On decidability of the control reachability problem in the asynchronous π-calculus

Roberto M. Amadio, Charles Meyssonnier

Laboratoire d'Informatique Fondamentale

UMR 6166
CNRS - Université de Provence - Université de la Méditerranée

Université de Provence

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

Abstract/Résumé

We study the decidability of the control reachability problem for various fragments of the asynchronous π-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 with *unique receiver* and *bounded* input (a condition weaker than bounded control) is decidable by reduction to the coverability problem for Petri Nets with transfer (and back).

Nous étudions la décidabilité d'un problème d'accessibilité pour plusieurs fragments du π-calcul asynchrone. Nous considérons la combinaison de trois aspects principaux : la *génération de noms*, la *mobilité de noms* et le *contrôle non borné*. Nous montrons que la combinaison de la génération de noms avec ou bien la mobilité de noms ou bien le contrôle non borné induit un fragment indécidable. D'autre part, nous démontrons que la génération de noms avec *recepteur unique* et avec input *borné* (une condition plus faible que le contrôle borné) est décidable par reduction au problème de couverture pour les Réseaux de Petri avec transfert (et vice versa).

Relecteurs/Reviewers: Silvano Dal Zilio, Denis Lugiez.

Notes: The authors are partly supported by IST Profundis. A preliminary version of this article appeared in Electronic Notes in Theoretical Computer Science 52.1 and as INRIA Research Report 4241. A revised version will appear in the Journal of Nordic Computing.

1 Introduction

In the last fifteen years a variety of formalisms or calculi have been proposed for the description of interactive systems including notions such as 'object', 'mobility', 'migration', 'cryptographic function',... A common aspect of these formalisms is the emphasis on the generation and the management of names. Indeed, formalisms such as the π-calculus or the join-calculus can be regarded as elaborations over well known models like CCS [23] or Petri Nets (see, *e.g.* [27]), respectively, where a notion of generation and mobility of names is added.

The programming languages community has undergone a similar evolution moving from static memory management languages like Fortran to dynamic memory management languages like Pascal and, nowadays, object oriented languages. The introduction of *references* in programming languages has required a considerable rethinking of the run time support going from the introduction of a heap memory to the development of garbage collection algorithms. The automatic verification community is now confronted to a similar problem. The current technology applies well to finite state systems but its extension to more expressive models is not so well-understood. For instance, if we consider the situation for the π-calculus, which has a prominent role among the calculi including name generation, the decidability result we are aware of concern the *synchronous* π-calculus with *bounded* control (see, *e.g.*, [9, 24]).

We believe that an interesting problem is to go beyond this restricted framework. In particular, we propose to look at *properties* of the reduction relation such as control reachability, boundedness, deadlock, liveness,... for process calculi based on the *asynchronous* π-calculus [8, 18, 4] – 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* π-calculus forces a synchronization between the sender and the receiver. We pause to note that the quest for decidability results for *equivalence* problems (trace, bisimulation,...) is discouraged by the negative results known for Petri Nets [17, 20] and the fact that, as we will see, even restricted fragments of the asynchronous π-calculus have an expressive power that goes well beyond Petri Nets.

Our interest in the asynchronous π-calculus stems from the observation that the core of concurrent programming languages such as PICT [25], JOIN [15], or TYCO [30] are based on it and the remark that object-oriented programming languages enjoy a rather direct representation in these formalisms. Thus advances in the analysis of this calculus are likely to have an impact on 'practical' program analysis. In this paper, we will mainly consider a *minimal* asynchronous, polyadic, simply sorted π-calculus *not* including external choice and we will concentrate on three main 'features' of this minimal calculus:

- 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. This encoding, basically going back to early work [16] on the translation of CCS 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.

Our main results are as follows:

1. The combination of name generation and name mobility leads to an undecidable fragment even assuming the control finite.

2. 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).

3. Name generation with *unique receiver* and *bounded input* conditions is decidable by reduction to the coverability problem for Petri Nets with transfer.

Roughly, the unique receiver condition entails that at most one thread can perform an input on a generated name, and the bounded input condition makes sure that there is a bound on the number of generated names on which an input can be performed (bounded control implies bounded input, but not vice versa; more on this in section 6.1).

Our third (and main) result generalizes a previous result we presented in [6] stating that name generation *without* name mobility and with bounded control is decidable by reduction to the coverability problem for Petri Nets. The generalization has a price as we have to move to a more general form of Petri Nets *with transfer*.

We also observe that the coverability problem for Petri Nets (Petri Net with transfer) can be encoded into the control reachability problem for bounded control systems without name mobility (bounded control systems without name mobility and with name generation). This shows that the reduction of the control reachability problem to the coverability problem for Petri Net (with transfer) is *not* an overkill and it also provides another example of the blow up in complexity due to the introduction of name generation (cf. remark 6.11).

2 Asynchronous π-calculus

We fix our notation for the asynchronous polyadic π-calculus. Simple sorts are defined by the following grammar:

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

where intuitively o is some ground sort and $Ch(s_1, \ldots, s_n)$ is the sort of channels carrying n data of sort $s_1, \ldots, s_n$. We assume a set of *names*, ranged over by a, b, c, $\ldots$ and suppose that (i) every name a comes with a fixed sort $st(a) = s$ and that (ii) for every sort there are denumerable many names of that sort.

We also assume a denumerable set of *parametric process identifiers* $A, B, \ldots$ and suppose that every identifier A comes with a fixed sort $st(A) = Ch(s_1, \ldots, s_n)$ so that n is the number of parameters and s_i the sort of the i^{th} parameter.

Vectors of names (possibly empty) are denoted $\vec{a}, \vec{b}, \ldots$ We denote with $[\vec{b}/\vec{a}]$ a substitution on names. In substitutions, a name (an identifier) can only be replaced by a name (an identifier) with the same sort.

We consider a polyadic, simply sorted, asynchronous π-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. If $\vec{a} \equiv a_1, \ldots, a_n$ then we use $(\nu\vec{a})$ as a shorthand for $(\nu a_1) \ldots (\nu a_n)$ and, without loss of generality, we assume that the names a_i are all distinct.

A *process* is presented by a finite system $\mathcal{E}$ of *parametric equations* $A(a_1, \ldots, a_n) = P$ 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 *parametric 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 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.

Also, we assume that 2 is well-sorted which means that: (i) if $st(A) = Ch(s_1, \ldots, s_n)$ then $\vec{a} \equiv a_1, \ldots, a_n$ and $st(a_i) = s_i$; (ii) if $st(a) = Ch(s_1, \ldots, s_n)$ then $\vec{a'} \equiv a'_1, \ldots, a'_n$ and $st(a'_i) = s_i$; and (iii) similar conditions hold for the messages $\overline{a}_i\vec{a}_i$ and the continuations $A_j(\vec{a}_j)$).

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 'Π' 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 (i) it has to evaluate the actual parameters, (ii) unfold a recursive definition to find an input prefix matching a message, and then (iii) bring at top level the name generations, the messages, and the continuations under the input prefix. The advantages of this approach are that we can then (i) limit the structural rules to the ones stated above, (ii) give a compact normal form for configurations, and (iii) provide a simple translation to Petri Nets.

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

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

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

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

(4) *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 π-calculus, say:

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

Indeed, this can be easily checked. We note that, up to structural equivalence, a process p in the usual notation 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 $A_j(\ldots) = a_j(\vec{a}_j).Q_j$ where Q_j is obtained by applying 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 $A_k(\ldots) = a_k(\vec{a}_k).(A_k(\ldots) \mid Q_k)$, where Q_k is obtained by applying inductively the transformation to p_k.

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

Definition 2.2 (control reachability) *Given a system of equations $\mathcal{E}$ containing a process identifier A and a related initial configuration P, the control reachability problem asks whether P reduces to a configuration containing the process identifier A,* i.e. $P \rightarrow^* (\nu \vec{a})(\ldots \mid A(\vec{b}) \mid \ldots)$, *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 2 is restricted to having the shape:

$$A(a_1, \ldots, a_n) = b(a'_1, \ldots, a'_m).(\Pi_{i \in I} \overline{b}_i \vec{b}_i \mid \Pi_{j \in J} A_j(\vec{b}_j)) \ . \tag{4}$$

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 [23]). We assume a finite set N of names with generic elements $x, y, \ldots$ and a finite set Id of identifiers with generic elements $X, Y, \ldots$ To every process identifier X we associate an equation of the shape:

$$X = \Sigma_{k \in K} x_k.(\Pi_{i \in I_k} \overline{x}_i \mid \Pi_{j \in J_k} X_j) \tag{5}$$

where K is a finite set and Σ stands for the *external choice*. This means that X can perform an input on exactly one of the channels $\{x_k\}_{k \in K}$ for which a corresponding message is available.

We emphasize that external choice is just used here to represent an intermediate step towards the translation to Petri Nets. Indeed, external choice is easily encoded in Petri Nets as explained in the following section 3.2. If K is empty, we take conventionally the right hand side as the terminated process 0. The reduction rule is then formulated as follows assuming the usual properties for parallel composition:

$$X \mid \overline{x}_k \mid P \to \Pi_{i\in I_k}\overline{x}_i \mid \Pi_{j\in J_k}X_j \mid P \ . \tag{6}$$

Note that here no renaming is needed and that the process identifier X is *literally* replaced by the right hand side of 5 defining it. Also, since there is no name generation, all names appearing in a reachable configuration belong to N.

3.2 From parameterless systems of equations to Petri Nets

We fix a system of equations $\mathcal{E}$ without parameters of the shape (5) and an initial configuration P. Let N and Id be the set of names and identifiers, respectively, occurring in $\mathcal{E}$ and P.

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

2. For every equation of the shape (5) we introduce the transitions $\{t_k\}_{k\in K}$ which are connected to the places as follows: for all $k \in K$, an edge from place X to transition t_k and an edge from place x_k to transition t_k. Moreover, if the continuation of x_k has the shape

   $$(\Pi_{i\in I_k}\overline{x_i} \mid \Pi_{j\in J_k}X_j)$$

   then we add an edge from transition t_k to place x_i for $i \in I_k$ and from transition t_k to place X_j for $j \in J_k$.

3.3 From systems without name generation to parameterless systems

We fix a system of parametric equations $\mathcal{E}_b$ without name generation of the shape (4) with initial configuration P. Let N_b be the set of names occurring *free* in P and let Id_b be the set of process identifiers occurring in $\mathcal{E}_b$.

We define the set of names N of the parameterless system as the least set such that:

$$\frac{a, a_1, \ldots, a_n \in N_b \text{ and } st(a) = Ch(s_1, \ldots, s_n), st(a_i) = s_i, i = 1, \ldots, n}{\langle a, a_1, \ldots, a_n\rangle \in N}$$

and similarly we define the set of identifiers Id of the parameterless system as the least set such that:

$$\frac{A \in Id_b, a_1, \ldots, a_n \in N_b \text{ and } st(A) = Ch(s_1, \ldots, s_n), st(a_i) = s_i, i = 1, \ldots, n}{\langle A, a_1, \ldots, a_n\rangle \in Id} \ .$$

We write $b_1, \ldots, b_n \downarrow a_1, \ldots, a_m$ if $n = m$, $b_i \in N_b$, and $st(b_i) = st(a_i)$ for $i = 1, \ldots, n$. For every equation of the shape (4) and for every vector $c_1, \ldots, c_n$ of names such that $c_1, \ldots, c_n \downarrow a_1, \ldots, a_n$ we produce a parameterless equation

$$\begin{aligned} \langle A, c_1, \ldots, c_n\rangle \quad = \quad & \Sigma_{\vec{c}'\downarrow\vec{a}'} \ \ \sigma_{\vec{c}'}\underline{\langle b, a'_1, \ldots, a'_m\rangle}. \\ & \sigma_{\vec{c}'}(\Pi_{i\in I}\langle b_i, \vec{b_i}\rangle \mid \Pi_{j\in J}\langle A_j, \vec{b_j}\rangle), \end{aligned}$$

where $\sigma_{\vec{c}'} \equiv [\vec{c}/\vec{a}, \vec{c}'/\vec{a}']$ and we take $\sigma\langle u_1, \ldots, u_l\rangle = \langle \sigma u_1, \ldots, \sigma u_l\rangle$.

It is readily verified that in this way we produce a parameterless equation for every identifier in Id and that the names used on the right hand side belong to N. We leave it to the reader to check that the parameterless system represents exactly the behaviour of the parametric system.

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

3.4 From control reachability to coverability, and back

In terms of Petri Nets, the control 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 [22] has provided a $2^{O(\sqrt{n})}$ space lower bound and Rackoff [26] a $2^{O(n \log n)}$ space upper bound (for a survey on these results see, *e.g.*, [12]).

On the other hand, it is easy to see that the coverability problem for Petri Nets can be reduced to the control 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):

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

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 control reachability problem 2.2 we consider requires exponential space. A variant of this translation for Petri Nets with transfer will be presented in section 6.11.

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 in this and the following section 5, it will be enough to consider the fragment where (2) is restricted to having the shapes:

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

where $\oplus$ denotes the *internal choice* (which is only used in the following section 5). 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.

4.1 Definability of internal choice

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:

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

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

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

4.2 2-counter machines

In the undecidability proofs, we simulate 2-counter machines (see, *e.g.*, [19]) and reduce the halting problem to the control reachability problem 2.2. We assume that a 2-counter machine contains instructions of the form:

$$\begin{array}{lll} (1) & q & : C_k := C_k + 1;\ goto\ q' \\ (2) & q & : (C_k = 0) \rightarrow goto\ q', C_k := C_k - 1; goto\ q'' \end{array}$$

where C_1, C_2 denote the two counters. An instruction of type (1) increments the counter C_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''.

4.3 Undecidability with generated channels

We now turn to the main result of this section.

Proposition 4.1 *The control 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 $A = a(\vec{x}).(\nu \vec{y})(\overline{b}(\vec{x},\vec{y}) \mid \overline{a}(\vec{x}) \mid A)$. More recent variations over this theme can be found in the framework of Dolev-Yao model of cryptographic protocols [11, 5].

We now turn to the technical development. 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

$$\begin{array}{lll} Top_k & \text{of sort } Ch(Ch(Ch(), Ch(), Ch())) & \text{and} \\ Adj_k & \text{of sort } Ch(Ch(Ch(), Ch(), Ch()), Ch()) \ . & \end{array}$$

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:

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

We now consider the problem of implementing on this data structure the 2-counter machine operations. We build a system of equations as follows:

- For every instruction of type (1) we introduce an equation:

$$\begin{array}{ll} A_q & = q.Top_k(a).(\nu a', a'_0, a'_1, a'_t) \\ & (\overline{q}' \mid \overline{Adj}_k(a, a'_t) \mid \overline{Top_k}(a') \mid \overline{a}'(a'_0, a'_1, a'_t) \mid \overline{a'_1} \mid A_q) \ . \end{array}$$

- For every instruction of type (2) we introduce two equations:

$$\begin{array}{ll} A^1_q & = q.Top_k(a).a(a_0, a_1, a_t).a_0. \\ & (\overline{q'} \mid \overline{Top_k}(a) \mid \overline{a}(a_0, a_1, a_t) \mid \overline{a_0} \mid A^1_q), \end{array}$$

$$\begin{array}{ll} A^2_q & = q.Top_k(a).a(a_0, a_1, a_t).a_1.Adj_k(b, b_t). \\ & (\overline{a_t} \mid b_t.(\overline{q''} \mid \overline{Top_k}(b) \mid A^2_q)) \ . \end{array}$$

For the sake of readability we have omitted the parameters (which can be easily inferred) as well as the intermediate process identifiers. The first equation corresponds to the case $C_k = 0$ and the second to the case $C_k > 0$. This second case 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 adjacent to a.

Finally, we introduce the equation $A = q_H.B$ where q_H is the halting state of the 2-counter machine. The *initial configuration* includes the identifiers A_q or A^i_q, $i = 1, 2$ for every state q of the 2-counter machine as well as the identifier A.

Moreover for $k = 1, 2$ we have the messages:

$$\overline{a}(a_0, a_1, a_t) \mid \overline{a_0} \mid \overline{Top_k}a$$

corresponding to two empty stacks, and the message $\overline{q_o}$ corresponding to the initial state q_o of the 2-counter machine.

We claim that the halting problem for the 2-counter machine reduces to the reachability of the control point B. The system we have described can simulate every reduction of the

2-counter machine. Moreover, in the simulation of instructions of type (2) a certain non-determinism arises because the control may be caught by A^1_q or A^2_q and in the second case an arbitrary message $\overline{Adj}_k(b, b_t)$ may be received. Note however that if something goes wrong, *i.e.* either we pick up the wrong branch or the wrong 'adjacent message' the computation gets stuck before the message corresponding to the next state ($\overline{q'}$ or $\overline{q''}$) is released. Thus the system may engage in more computations than the 2-counter machine but these computations lead immediately to a deadlock. □

Remark 4.2 *The encoding above relies on channel mobility and moreover processes may input on received channel names. In turn this feature is used to program a weak form of name equality that allows to encode the 'decrement' instruction of 2-counter machines. The decidability result in section 6 suggests that this is an essential feature as weaker forms of name mobility with bounded control turn out to be decidable.*

4.4 Undecidability with generated values and conditional

The encoding presented above can be further simplified if we consider a π-calculus with a *conditional* on name equality. To formalise this extension, we assume equations may have the shape:

$$A(\vec{a}) = [a = a']A'(\vec{a}'), A''(\vec{a}'') \tag{7}$$

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 (no channel mobility). We assume additional channels $Cont_k$ to indicate the contents of a cell (values 0 or 1). The sorts are now as follows:

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

For every instruction of type (1) we introduce the equation:

$$A_q = q.Top_k(a).(\nu a)'(\overline{q'} \mid \overline{Adj_k}(a, a') \mid \overline{Cont}_k(a', 1) \mid \overline{Top_k}(a') \mid A_q),$$

and for every instruction of type (2) we introduce two equations:

$$\begin{aligned} A^1_q &= q.Top_k(a).Cont_k(a', v).[a' = a][v = 0] \\ &\quad (\overline{q'} \mid \overline{Top_k}(a) \mid \overline{Cont_k}(a, 0) \mid A^1_q), \end{aligned}$$

$$\begin{aligned} A^2_q &= q.Top_k(a).Cont_k(a', v).[a' = a][v = 1]Adj_k(a', a'').[a'' = a] \\ &\quad (\overline{q'} \mid \overline{Top_k}(a') \mid A^2_q)\ . \end{aligned}$$

Then the encoding follows the pattern presented in the proof of proposition 4.1.

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 2 is restricted to having the shape:

$$A(\vec{a}) = a.(\nu \vec{d})(\Pi_{i \in I} \overline{a}_i \mid \Pi_{j \in J} A_j(\vec{c}_j))\ . \tag{8}$$

In the absence of name mobility, generated names cannot be extruded and therefore name generation is essentially CCS *restriction*. Milner [23] 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 control reachability property), and (iii) relabelling by parametric equations.

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

PROOF. Again we simulate a 2-counter machine (cf. section 4.2) and reduce the halting problem to the control reachability problem 2.2. The basic issue is to represent a stack. To this end we define the following system of equations (inspired by [23]). 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 in the parameters below.

$$\begin{array}{ll}
B(i,z,d) & = B_i(i,z,d) \oplus B_z(i,z,d) \\
B_i(i,z,d) & = i.(\overline{i}^a \mid CB(i,z,d)) \\
B_z(i,z,d) & = z.(\overline{z}^a \mid B(i,z,d)) \\
 & \\
C(i,z,d,z',d') & = C_i(i,z,d,z',d') \oplus C_d(i,z,d,z',d') \\
C_i(i,z,d,z',d') & = i.(\overline{i}^a \mid CC(i,z,d,z',d')) \\
C_d(i,z,d,z',d') & = d.((\overline{d'} \oplus \overline{z'}) \mid D(i,z,d,z',d')) \\
 & \\
D(i,z,d,z',d') & = D_d(i,z,d,z',d') \oplus D_z(i,z,d,z',d') \\
D_d(i,z,d,z',d') & = (d'^a.(\overline{d}^a \mid C(i,z,d,z',d'))) \\
D_z(i,z,d,z',d') & = (z'^a.(\overline{d}^a \mid B(i,z,d))) \\
 & \\
CB(i,z,d) & \equiv (\nu i'',z'',d'')(C(i,z,d,z'',d'') \mid B(i'',z'',d'')) \\
CC(i,z,d,z',d') & \equiv (\nu i'',z'',d'')(C(i,z,d,z'',d'') \mid C(i'',z'',d'',z',d'))\ .
\end{array}$$

A process B receives on either i or z and acknowledges on either i^a or z^a, respectively. A process C receives on either i or d and either acknowledges on i^a or sends on either d' or z', respectively. A process D receives on either d'^a or z'^a and sends on d^a. The processes CB and CC create a new C which is suitably linked to either B or C, respectively.

We describe the intended dynamics of a *decrement* instruction. 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 in a specific case:

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

The D is propagated towards the right till it meets B and when this happens it becomes B and shortcuts the last B, then the call comes back and the D's are turned again into C's.

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 stacks representing the counters that are initialised by:

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

For every instruction of type (1) we introduce the equation:

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

and for every instruction of type (2) we introduce two equations:

$$\begin{array}{ll} A_q^1 & = q.(\overline{z}_k \mid z_k^a.(\overline{q'} \mid A_q^1)) \\ A_q^2 & = q.(\overline{d}_k \mid d_k^a.(\overline{q''} \mid A_q)) \ . \end{array}$$

Moreover, we introduce the equation $A = q_H.B$.

The *initial configuration* includes the process S above corresponding to the two empty stacks, the identifiers A_q or A_q^i $(i = 1, 2)$ for every state q of the 2-counter machine, the identifier A, and the message $\overline{q}_o$ corresponding to the initial state q_o of the 2-counter machine.

We claim that the halting problem for the 2-counter machine reduces to the reachability of the control point B. 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 choices. This may happen (i) when the control sends an increment or test_zero request to a counter, (ii) when A_q^1 and A_q^2 compete to take the control, or (iii) when a decrement instruction propagates towards the right in a counter. In all cases the control is stuck. In the first and second case this is immediate and in the third case this happens because the control waits for an acknowledgement which is delivered only after the propagation is completed and the D's are turned back into C's. □

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. *Thus we have also shown that 'bounded messages' with unbounded control is undecidable.*

6 The fragment with unique receiver and bounded input is decidable

In [6], we have shown that control reachability for the fragment without name mobility and with bounded control is decidable by reduction to the coverability problem for Petri nets. We briefly recall our main arguments:

- 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 and all messages addressed to that name can be collected.

- We generalise the reduction to Petri Nets presented in section 3 and simulate *name generation* by the reusing of 'dead' names and *name collection* by a *reset* on the corresponding places. It turns out that by a *static* analysis it is possible to decide whether a generated name is persistent or temporary (it eventually dies) and in the latter case to give a *bound* on the number of messages emitted on the temporary name. Because the reset operation on a *bounded* place is already definable in Petri Nets, we can show that the control reachability problem for the considered fragment reduces to the coverability problem for Petri Nets.

This result is not very satisfying because: (i) it completely forbids name mobility and (ii) it requires bounded control. In section 3, we have seen that in the absence of name generation the control reachability problem is decidable (even if the control is unbounded) thus there is clearly some space for improvement. In this section we present a more general decidability result for a fragment including some form of name mobility and unbounded control. First, we introduce three basic ingredients: unique receiver condition (section 6.1), bounded input condition (section 6.2), and Petri Nets and parameterless systems with transfer (section 6.3). We informally discuss how they contribute to our decision result before turning to the formal development in the following section 6.4.

6.1 Unique receiver condition and name collection

The formalization of the decidable fragment relies on the *unique receiver* condition: a syntactic condition on the π-calculus studied in [2] (the unique receiver condition presented here is actually more liberal than the one in [2]). The condition ensures that any *generated* name is associated to a unique thread that can possibly perform input actions on it. To make sure that reduction preserves this property, we also require that received names cannot be used in input position; a property also known as *local* mobility.

In practice, the *unique receiver* condition is usually satisfied as one leans naturally towards an 'object-oriented' style of programming where interaction arises when an object calls the method of another *uniquely determined* object. In theory, it can be shown [2] that the join-calculus can be represented up to 'asynchronous' bisimulation in the asynchronous π-calculus with unique receiver. Since in turn the asynchronous π-calculus can be translated into the join-calculus in a fully abstract way [14], we can claim that the *unique receiver* condition does *not* impair the expressivity of the programming model.

In order to formulate the unique receiver condition it is convenient to distinguish the parameters of a process identifier in two lists separated by ';' as in $A(\vec{a}_I;\vec{a}_O)$, with the intended meaning that the names $\vec{a}_I$ can be used in input and output whereas the names $\vec{a}_O$ can be used in output only. In the following we will refer to $\vec{a}_I$ as the i/o parameters and to $\vec{a}_O$ as the output parameters of the process identifier A.

Definition 6.1 *An equation of the shape:*

$$A(\vec{a}_I;\vec{a}_0) = a(\vec{a}').(\nu\vec{a}'')(\Pi_{k\in K}\overline{a}_k\vec{a}_k \mid \Pi_{j\in J}A_j(\vec{a}_{I,j};\vec{a}_{O,j})) \tag{9}$$

satisfies the unique receiver condition*, if besides the usual conditions on parametric equations (2), the following holds:*

(1) *the input is performed on an i/o parameter:* $a \in \{\vec{a}_I\}$,

(2) *the input parameters in the continuations* $\{\vec{a}_{I,j}\}_{j\in J}$ *are all distinct, and*

(3) *the generated names belong to the i/o parameters of the continuations and the latter are included in the i/o parameters or the generated names:*

$$\{\vec{a}''\} \subseteq \bigcup_{j \in J} \{\vec{a}_{I,j}\} \subseteq \{\vec{a}_I\} \cup \{\vec{a}''\} \ . \tag{10}$$

An equation of the shape:

$$A(\vec{a}_I; \vec{a}_0) = B(\vec{b}_I; \vec{b}_0) \oplus C(\vec{c}_I; \vec{c}_0) \tag{11}$$

satisfies the unique receiver condition, *if (besides the usual conditions) the names $\vec{b}_I$ are all distinct, the names $\vec{c}_I$ are all distinct, and $\{\vec{b}_I\} \cup \{\vec{c}_I\} \subseteq \{\vec{a}_I\}$.*

We include internal choice because it is useful in practice to approximate the behaviour of a conditional on name equality and because we need it to encode Petri Nets with transfer (cf. section 6.5).

The separator ';' plays no role in the reduction relation and therefore definition 2.1 applies to these refined systems.

Roughly, the conditions above make sure that in a reachable configuration a generated name can appear at most once (up to internal choice) among the i/o parameters of at most one process identifier.

The unique receiver condition provides a simple local criterion to *collect generated names.* Once a generated name, say a, is removed from the i/o parameters of the related thread we are sure that it can never appear again among the i/o parameters (no other thread can have that name among the i/o parameters and no thread can put a received name among its i/o parameters).

Thus the name a is *dead* and some action has to be taken so that it can be reused at a later time. The name a can appear in the current configuration in three positions:

1. as the address of a message as in $\overline{a}\vec{b}$: these messages can be just thrown away since no thread will ever receive them.

2. as the contents of a message, say $\overline{b}a$: these messages can still be received and retransmitted indefinitely but their contents can never be exploited. Therefore we can turn the message into $\overline{b}g$ where g is some *generic* name on which no input action is ever performed.

3. as the output parameters of a process identifier, say $A(\ldots; a)$: the same reasoning as in the previous case applies and we turn these process identifiers into $A(\ldots; g)$.

We will explain in section 6.3 how these 'collection operations' can be represented in Petri Nets *with transfer.*

6.2 Bounded input condition and name generation

It is convenient to assume that the *initial configuration* has the shape:

$$\mathit{Init} \equiv \Pi_{k \in K} \overline{a}_k \vec{a}_k \mid \Pi_{j \in J} A_j(\vec{a}_{I,j}; \vec{a}_{O,j}) \mid \Pi_{h \in H} A_h(\vec{a}_{I,h}; \vec{a}_{O,h}) \tag{12}$$

where all process identifiers occurring in the configuration are distinct and the related system of equations $\mathcal{E}$ can be *partitioned* in systems $\mathcal{E}_j$, $\mathcal{E}_h$ relating to identifiers A_j, A_h, for $j \in J$

and $h \in H$, respectively. This means that a process identifier occurring in a reachable configuration is uniquely associated to exactly one process identifier in the initial configuration.

We note that there is no loss of generality in these hypotheses. If the initial configuration contains generated names, then replace these names by fresh constants. If it contains a repeated process identifier then just introduce a fresh process identifier and duplicate the related equations. Finally, if the system of equations cannot be partitioned then again introduce fresh process identifiers and duplicate the related equations. The partitioning of the system $\mathcal{E}$ is instrumental to the following definition.

Definition 6.2 *An initial configuration such as (12) and the related system of equations $\mathcal{E}$ satisfy the* bounded input condition *if the following holds:*

(1) *The equations in $\mathcal{E}_j$, $j \in J$ are either of the shape (9) with exactly* one *continuation or of the shape (11).*

(2) *In the equations in $\mathcal{E}_h$, $h \in H$* no *name generation is allowed.*

Note that the initial configuration (12) is not 'bounded control' in the sense adopted in section 4. The definition 6.2 requires that threads that can generate names have exactly one continuation whereas the other threads can have an arbitrary number of continuations.

A problem dual to collection is the one of computing *fresh* names, *i.e.*, for all practical purposes, names that are not currently used elsewhere in the current configuration. The *bounded input condition* is helpful in solving this problem: if we allocate to each thread generating names a sufficiently large *finite* set of names, then at any point in the computation the thread can figure out the names it is currently using and determine, by complementation, the names not used.

6.3 Petri Nets and parameterless systems with transfer

Petri Nets *with transfer* are standard Petri Nets where moreover a transition may entail the transfer of the tokens in a place p to a distinct place q. Thus, if we regard places as counters, the transfer is expressed by the assignments: $q := q + p; p := 0$.

Definition 6.3 *A* Petri Net with transfer *is a tuple $(P, T, Pre, Post, Trans)$, where P is a finite set of* places*, T is a finite set of* transitions*, Pre is a function from $P \times T$ into $\mathbb{N}$ (the natural numbers), $Post$ is a function from $T \times P$ into $\mathbb{N}$, and $Trans$ is a function from T into $2^{P \times P}$.*

Additionally, it is convenient to assume that the transfer operations do not interfere with the transition and with each other. Thus we require that for all $t \in T$ and $(p, p') \in Trans(t)$: (1) $p \neq p'$, (2) $Pre(p,t) = Pre(p',t) = Post(t,p) = Post(t,p') = 0$, and (3) for all $(q, q') \in Trans(t)$, either $(q, q') = (p, p')$ or $\{p, p'\} \cap \{q, q'\} = \emptyset$.

A *marking* for such a net is a function from P into $\mathbb{N}$. The transition t will be said to be *enabled* with respect to the marking M if and only if $\forall p \in P\ M(p) \geq Pre(p,t)$, and when this is the case we have the reduction $M \to M'$, where for all $p \in P$,

$$M'(p) = \begin{cases} 0 & \text{if } (p,p') \in Trans(t) \text{ for some } p', \\ M(p) + M(p') & \text{if } (p',p) \in Trans(t) \text{ for some } p', \\ M(p) - Pre(p,t) + Post(t,p) & \text{otherwise.} \end{cases}$$

The Petri Nets with transfer we consider here are a particular case of the *generalized self-modifying nets* introduced in [10]. The reachability problem for Petri Nets with transfer is undecidable. The proof given in [7] for Petri Nets with reset still applies – in Petri Nets with reset, a transition can empty the contents of a place; for some problems, the reset operation can be simulated by transferring the contents of a place to a *sink* place. On the other hand, the coverability problem is decidable. The proof of this result has a rather long history. A first approach is based on a 'forward' analysis via the generation of the Karp-Miller tree [21]. Another approach presented in [1] uses a 'backward' analysis. Both methods rely on properties of well-orders. The latter in particular exploits the fact that the backward analysis generates a growing chain of ideals with an effectively computable basis. By the properties of well-orders this chain stabilizes and its limit can be effectively computed. This method applies to a variety of systems including Petri Nets with reset and/or with transfer [1, 10, 13].

Parameterless systems with transfer relate to Petri Nets with transfer just as in section 3 parameterless systems relate to Petri Nets. With reference to section 3.1, we now allow more general equations of the shape

$$X = \Sigma_{k\in K} x_k.\{T(u_h, v_h)\}_{h\in H_k}.(\Pi_{i\in I_k}\overline{x}_i \mid \Pi_{j\in J_k} X_j) \tag{13}$$

where u_h, v_h are either names or identifiers and T stands for transfer. The inteded meaning is that following an input x_k a sequence of transfer operations $\{T(u_h, v_h)\}_{h\in H_k}$ is performed. As expected, executing the transfer operation $T(u, v)$ amounts to remove all instances of u in the current configuration and create as many instances of v. *Mutatis mutandis*, we also require that these transfer operations do not interfere. Then, adapting the translation in section 3.2, we can reduce parameterless systems with transfer to Petri Nets with transfer.

The interest of the transfer operation in our context comes from the observation that if we represent messages $\overline{a}\vec{b}$ and control positions $A(\vec{a};\vec{b})$ as counters, as we did in translating systems without name generation to Petri Nets (section 3.2), then the collection operations described above in section 6.1 can be regarded as *transfer* operations.

6.4 Reduction to Petri Nets with transfer

We now turn to the core of the technical development. We fix an initial configuration P and a related system of equations $\mathcal{E}$ as in definition 6.2.

6.4.1 Names and identifiers

The first problem is to determine the names and identifiers of the parameterless system with transfer.

1. Let N_o be the collection of names occurring free in P.
2. Let St be the set of sorts of names occurring in $P, \mathcal{E}$.
3. Let $G = \{g_s \mid s \in St\}$ be a set of fresh *generic* names, one for each sort in St, and such that $st(g_s) = s$.
4. We assume a fresh name z that will correspond to a *sink* place without outgoing arcs. The reset of a place p is performed by transferring the contents of p to z.

5. For every thread $j \in J$ of the initial configuration (12) that can generate names we assume a finite set of *fresh names* N_j. The requirements on the size of N_j will be given in the following section 6.4.2. Let $N_\nu = \bigcup_{j\in J} N_j$.

6. We define the space of *names* N of the parameterless system as the least set such that $z \in N$ and if $a \in N_o \cup N_\nu$, $st(a) = Ch(s_1, \ldots, s_n)$, $a_i \in N_o \cup N_\nu \cup G$ then $\langle a, a_1, \ldots, a_n\rangle \in N$. Note that N is finite.

7. We specialise the notion of vector compatibility presented in section 3.3. We write

$$\vec{c}\downarrow_O \vec{a} \quad \text{if} \quad \vec{c}\downarrow \vec{a}, \{\vec{c}\} \subseteq N_o \cup N_\nu \cup G,$$

and we write

$$\vec{c}\downarrow_I^j \vec{a} \quad \text{if} \quad \vec{c}\downarrow \vec{a}, \{\vec{c}\} \subseteq N_o \cup N_j$$

and moreover if c_i and c_k are two distinct occurrences in $\vec{c}$ of names in N_j then $c_i \neq c_k$. If A is defined by $A(\vec{a}_I; \vec{a}_O) = \ldots$ and belongs to system $\mathcal{E}_l$ then we write

$$\vec{c}_I, \vec{c}_O \downarrow A \quad \text{if } \vec{c}_I \downarrow_I^l \vec{a}_I \text{ and } \vec{c}_O \downarrow_O \vec{a}_O \ .$$

If the system $\mathcal{E}_l$ is without name generation then we take conventionally $N_l = \emptyset$ and therefore $\{\vec{c}_I\} \subseteq N_o$.

8. We define the set of *identifiers* Id of the parameterless system as the least set such that if A is an identifier in the system $\mathcal{E}$ and $\vec{a} \downarrow A$ then $\langle A, \vec{a}\rangle \in Id$. Note that Id is finite.

Example 6.4 *Consider the system and initial configuration ($i = 1, 2$):*

$$\begin{array}{ll} S_i(a;\_) & = a(a', b).(\nu a'', b')(\overline{a'}(a'', b') \mid \overline{b} \mid R_i(a'', b';\_)), \\ R_i(a, b;\_) & = b.S_i(a;\_), \\ \mathit{Init} & \equiv S_1(a;\_) \mid \overline{a}(a', b) \mid R_2(a', b;\_) \ . \end{array}$$

Here $N_o = \{a, a', b\}$, $St = \{s_1 = Ch(s_1, Ch()), s_2 = Ch()\}$, and $G = \{g_{s_1}, g_{s_2}\}$ (note that s_1 is a recursive sort; this is a bit more general than the simple sorts we have considered in this paper, however our construction still applies).

Then we define for $i = 1, 2$ the sets of names and identifiers as follows:

$$\begin{array}{lcl} N_i & = & \{c_{i,j} \mid 1 \leq j \leq 3, st(c_{i,j}) = s_1\} \cup \{d_{i,j} \mid 1 \leq j \leq 2, st(d_{i,j}) = s_2\}, \\ N_\nu & = & N_1 \cup N_2, \\ N & = & \{z\} \cup \{\langle u\rangle \mid u \in N_o \cup N_\nu, st(u) = s_2\} \cup \\ & & \{\langle u, v, w\rangle \mid u \in N_o \cup N_\nu, v, w \in N_o \cup N_\nu \cup G, \\ & & st(u) = st(v) = s_1, st(w) = s_2\}, \\ Id & = & \{\langle S_i, u\rangle \mid 1 \leq i \leq 2, u \in N_o \cup N_i, st(u) = s_1\} \cup \\ & & \{\langle R_i, u, v\rangle \mid 1 \leq i \leq 2, u, v \in N_o \cup N_i, st(u) = s_1, st(v) = s_2\} \ . \end{array}$$

6.4.2 Parameterless equations

Every parametric equation in $\mathcal{E}$ of the shape (9) generates a certain number of parameterless equations of the shape (13). We refer to definition 6.2 of bounded input. If no name generation is involved then the parameterless equations are generated as described in section 3.3. Because generated names cannot appear among the i/o parameters no name collection is necessary.

Equations of the shape (11) involving internal choice produce a finite family of parameterless equations of the shape $X = Y \oplus Z$. The latter are easily represented in a Petri Net: introduce two transitions t_Y, t_Z, an edge from X to t_Y, an edge from X to t_Z, an edge from t_Y to Y, and an edge from t_Z to Z.

Therefore, we concentrate on the case where the equation has the shape:

$$A(\vec{a}_I; \vec{a}_0) = b(\vec{a}').(\nu \vec{a}'')(\Pi_{k \in K} \overline{b}_k \vec{b}_k \mid B(\vec{b}_I; \vec{b}_O)), \tag{14}$$

where the process identifier A relates to the system $\mathcal{E}_j$ with name generation, for some $j \in J$.

1. For every $\vec{c}_I, \vec{c}_O \downarrow A$ we define:

$$\begin{array}{lll} \sigma_I & = [\vec{c}_I/\vec{a}_I] & \text{(actual i/o parameters)} \\ D & = \sigma_I(\{\vec{a}_I\} \backslash \{\vec{b}_I\}) \cap N_j & \text{(dead names in } \vec{c}_I) \ . \end{array}$$

2. For $\vec{c}' \downarrow_O \vec{a}'$, we also define;

$$\begin{array}{lll} \vec{c}'' & = choose(N_j, \vec{a}'', \{\vec{c}_I, \vec{c}_O, \vec{c}'\}) & \text{(fresh names in } N_j) \\ \sigma_{\vec{c}'} & = [\vec{c}_I/\vec{a}_I, \vec{c}_O/\vec{a}_O, \vec{c}'/\vec{a}', \vec{c}''/\vec{a}''] & \text{(combined substitution)} \\ \sigma_D(b) & = \left\{ \begin{array}{ll} g_s & \text{if } \sigma_{\vec{c}'}(b) \in D \cup G, st(b) = s \\ b & \text{otherwise} \end{array} \right. & \text{(dead names to generic)} \\ K_D & = \{k \in K \mid \sigma_{\vec{c}'}(b_k) \in D \cup G\} & \text{(dead names to sink)} \ . \end{array}$$

   Here *choose* is a function selecting a vector of names $\vec{c}''$ such that $\vec{c}'' \downarrow_I^j \vec{a}''$ and $\{\vec{c}''\} \cap \{\vec{c}_I, \vec{c}_O, \vec{c}'\} = \emptyset$. We leave it to the reader to check that we can always select a finite N_j satisfying this property.

   The substitution σ_D and the set K_D are applied below to collect dead names.

3. For every $\vec{c}_I, \vec{c}_O \downarrow A$ we introduce the *parameterless equation*:

$$\begin{array}{rcl} \langle A, \vec{c}_I, \vec{c}_O \rangle & = & \Sigma_{\vec{c}' \downarrow_O \vec{a}'} \ \sigma_{\vec{c}'}(\langle b, \vec{a}' \rangle).Col(D). \\ & & \sigma_{\vec{c}'}(\Pi_{k \in K \backslash K_D} \overline{\langle b_k, \sigma_D(\vec{b}_k) \rangle} \mid \Pi_{k \in K_D} \overline{z} \mid B(\vec{b}_I; \sigma_D(\vec{b}_O))) \ . \end{array}$$

   We use the substitution σ_D to map dead names in argument position to generic names and we select in K_D the messages addressed to dead names. $Col(D)$ is an abbreviation for a sequence of transfer instructions collecting dead names in D. Intuitively, $Col(D)$ collects dead names in the surrounding terms and threads, *i.e.*, in the 'evaluation context'.

4. Formally, for all $a \in D$, $Col(D)$ contains the following *transfer instructions*:

$$\begin{array}{ll} T(\langle a, \vec{b} \rangle, z) & \forall \langle a, \vec{b} \rangle \in N, \\ T(\langle b, \vec{c} \rangle, \langle b, \tau_D \vec{c} \rangle) & \forall \langle b, \vec{c} \rangle \in N, \text{ with } \{\vec{c}\} \cap D \neq \emptyset, b \notin D, \\ T(\langle B, \vec{c} \rangle, \langle B, \tau_D \vec{c} \rangle) & \forall \vec{c} \downarrow B, \text{ with } \{\vec{c}\} \cap D \neq \emptyset, \end{array}$$

$$\text{where: } \tau_D(c) = \left\{ \begin{array}{ll} g_s & \text{if } c \in D, st(c) = s, \\ c & \text{otherwise.} \end{array} \right.$$

   We note that here name collection is performed *eagerly*: names generated by the thread j are collected as soon as they exit its i/o parameters. Another possibility to be explored is to collect names *lazily* when they are generated/reused.

5. Finally, the *initial configuration* (12) is translated into the parameterless notation as follows:

$$\langle Init\rangle \equiv \Pi_{k\in K}\overline{\langle a_k, \vec{a}_k\rangle} \mid \Pi_{j\in J}\langle A_j, \vec{a}_{I,j}, \vec{a}_{O,j}\rangle \mid \Pi_{h\in H}\langle A_h, \vec{a}_{I,h}, \vec{a}_{O,h}\rangle \ . \tag{15}$$

Example 6.5 (continued) *We further develop example 6.4 and concentrate on the equations associated to the identifier S_1 for $c_{1,1} \downarrow S_1$. Then $\sigma_I = [c_{1,1}/a]$, $D = \{c_{1,1}\}$, and the equation has the shape:*

$$\langle S_1, c_{1,1}\rangle = \Sigma_{\vec{c}'\downarrow a',b}\ P_{\vec{c}'}$$

If $\vec{c}' \equiv c'_1, c'_2 \in N_o \cup N_\nu \cup G$ we have:

$$P_{\vec{c}'} \equiv \langle c_{1,1}, c'_1, c'_2\rangle.Col(\{c_{1,1}\}).Q_{\vec{c}'} \ .$$

The following table shows the shape of $Q_{\vec{c}'}$ for two different choices of $\vec{c}'$ and $\vec{c}''$:

$\vec{c}'$	$\vec{c}''$	$Q_{\vec{c}'}$
$c_{1,2}, d_{1,1}$	$c_{1,3}, d_{1,2}$	$\overline{\langle c_{1,2}, c_{1,3}, d_{1,2}\rangle} \mid \overline{\langle d_{1,1}\rangle} \mid \langle R_1, c_{1,3}, d_{1,2}\rangle$
$g, d_{2,1}$	$c_{1,2}, d_{1,1}$	$\overline{z} \mid \overline{\langle d_{2,1}\rangle} \mid \langle R_1, c_{1,2}, d_{1,1}\rangle$.

Note that we only collect the names of the first thread, e.g., *in the second case the name $d_{2,1}$ that belongs to the second thread is received in input in $\vec{c}$ and a message $\overline{\langle d_{2,1}\rangle}$ is emitted. It is up to the second thread to make sure that the name $d_{2,1}$ is alive.*

The sequence of transfer instructions associated to $Col(\{c_{1,1}\})$ is:

$$\begin{array}{l}\{T(\langle c_{1,1}, u, v\rangle, z) \mid \langle c_{1,1}, u, v\rangle \in N\}\cup \\ \{T(\langle u, c_{1,1}, v\rangle, \langle u, g_{s_1}, v\rangle) \mid \langle u, c_{1,1}, v\rangle \in N, u \neq c_{1,1}\}\cup \\ \{T(\langle S_1, c_{1,1}\rangle, \langle S_1, g_{s_1}\rangle)\}\cup \\ \{T(\langle R_1, c_{1,1}, u\rangle, \langle R_1, g_{s_1}, u\rangle) \mid u \in N \cup N_1, st(u) = s_2\} \ .\end{array}$$

Finally, the initial configuration is:

$$\langle Init\rangle \equiv \langle S_1, a\rangle \mid \overline{\langle a, a', b\rangle} \mid \langle R_2, a', b\rangle \ .$$

6.4.3 Bisimulation relation and decidability

A configuration reachable from the initial configuration (12) has the shape:

$$P \equiv (\nu\vec{a})(\Pi_{k\in K}\overline{a_k}(\vec{a}_k) \mid \Pi_{j\in J}A_j(\vec{a}_{I,j}; \vec{a}_{O,j}) \mid \Pi_{h\in H}A_h(\vec{a}_{I,h}; \vec{a}_{O,h})) \ .$$

Let us set $L_j = \{\vec{a}\} \cap \{\vec{a}_{I,j}\}$, $L = \bigcup_{j\in J} L_j$, $Z = \{\vec{a}\}\backslash L$, and $K_Z = \{k \in K \mid a_k \in Z\}$. Because of the unique receiver condition (definition 6.1) we claim that the sets $Z, \{L_j\}_{j\in J}$ form a partition of $\{\vec{a}\}$.

Then we introduce a *relation* $\mathcal{R}$ connecting configurations with the property above to parameterless configurations as follows:

$$\begin{array}{rll} P\ \mathcal{R}\ Q & \text{if } Q \equiv & \tau(\Pi_{k\in K\backslash K_Z}\overline{\langle a_k, \vec{a}_k\rangle}) \mid \Pi_{k\in K_Z}\overline{z} \mid \\ & & \tau(\Pi_{j\in J}\langle A_j, \vec{a}_{I,j}, \vec{a}_{O,j}\rangle \mid \Pi_{h\in H}\langle A_h, \vec{a}_{I,h}, \vec{a}_{O,h}\rangle)\end{array}$$

where $\tau : \{\vec{a}\} \to N_\nu \cup G$ is a substitution such that $\tau(a) \in N_j$ if $a \in L_j$ and $\tau(a) = g_s$ if $a \in Z$ and $st(a) = s$, and moreover τ is *injective* on the live names L. The rationale for the definition of the relation $\mathcal{R}$ is given by the two following lemmas.

Lemma 6.6 *The relation $\mathcal{R}$ is a bisimulation.*

PROOF HINT. Suppose $P \mathcal{R} Q$, we show that:
(1) If $P \to P'$ then for some Q', $Q \to Q'$ and $P' \mathcal{R} Q'$.
(2) If $Q \to Q'$ then for some P', $P \to P'$ and $P' \mathcal{R} Q'$. This follows from a direct analysis presented in appendix A. □

Lemma 6.7 *The initial configuration Init (12) reaches a control $A(\ldots)$ iff the related parameterless initial configuration $\langle Init \rangle$ (15) reaches a control $\langle A, \ldots \rangle$.*

PROOF. We note that $Init \mathcal{R} \langle Init \rangle$. From the definition of the relation $\mathcal{R}$ it follows that if $P \mathcal{R} Q$ then P includes a control $A(\ldots)$ iff Q includes a control $\langle A \ldots \rangle$. We conclude by applying the previous lemma 6.6. □

We finally arrive at the announced decidability result.

Theorem 6.8 *The control reachability problem for input bounded systems with unique receiver is decidable.*

PROOF. From lemma 6.7 the control reachability problem for, say, the identifier A, reduces to control reachability for the identifiers of the shape $\langle A, \vec{a} \rangle$ in the parameterless system. Because the name space described in section 6.4.1 is finite there are finitely many of them. In turn, by the discussion in section 6.3, the reachability of an identifier $\langle A, \vec{a} \rangle$ is equivalent to determine whether the place $\langle A, \vec{a} \rangle$ in the corresponding Petri Net with transfer will contain a token. As recalled in section 6.3 this property is decidable. □

6.5 Encoding Petri Nets with transfer

We now turn to showing how the coverability problem for Petri Nets with transfer can be encoded into the control reachability problem for parametric systems with name generation, bounded control, and without name mobility – these systems are a particularly simple case of the systems that we have shown to be decidable.

Suppose given a Petri Net with transfer $\mathcal{P} = (P, T, Pre, Post, Trans)$ (cf. definition 6.3), with $P = \{p_i\}_{i \in I}$. The net $\mathcal{P}$ is simulated by *one* thread defined by the system of equations in 1. For the sake of simplicity, we only consider the case where each transition has at most one transfer arc (multiple transfer arcs can clearly be encoded at the cost of additional equations by sequentializing the transfers). Of course, for transitions t without a transfer arc, we suppress the equations $TRANS_t$, $STEP_t$ and $RESET_t$, and the control flows directly from $PRE_{t,m}$ to $POST_t$. Similarly, if $m = 0$ then we start directly with $TRANS_t$. Note that this definition relies on name generation and internal choice but *not* on name mobility.

The notation $Post(t, p_i) \cdot \overline{x_i}$ stands for $Post(t, p_i)$ copies of the message $\overline{x_i}$. It is assumed that $\vec{x}$ is a vector of $\sharp I$ distinct names x_i (for $i \in I$), that $\vec{x'}$ is obtained by substituting x' for x_k in $\vec{x}$, and that $(i_j)_{j \in \{1,\ldots,m\}}$, k, and l are the respective indices in I of the preplaces of t, the origin of the transfer arc, and the destination of the transfer arc, (*i.e.*, $Pre(p_i, t) = \sharp \{j \in \{1, \ldots, m\} \mid i_j = i\}$ for all $i \in I$, and $Trans(t) = \{(p_k, p_l)\}$).

The intended meaning is that a token in place p_i is represented by a message $\overline{x_i}$, and we accordingly define the *translation* of a marking M as the configuration

$$\phi(M) \equiv (\nu \vec{x}) \, (\Pi_{i \in I} M(p_i) \cdot \overline{x_i} \mid LOOP(\vec{x})) \ ,$$

$$
\begin{array}{rll}
LOOP(\vec{x}) & = \oplus_{t\in T} PRE_t(\vec{x}) & \text{(choose a transition)} \\
 & & \\
PRE_{t,1}(\vec{x}) & = x_{i_1}.PRE_{t,2}(\vec{x}) & \text{(consume tokens)} \\
\ldots & = \ldots & \\
PRE_{t,m}(\vec{x}) & = x_{i_m}.TRANS_t(\vec{x}) & \\
 & & \\
TRANS_t(\vec{x}) & = STEP_{t,1}(\vec{x}) \oplus RESET_t(\vec{x}) & \text{(begin transfer)} \\
STEP_t(\vec{x}) & = x_k.\,(\overline{x_l} \mid TRANS_t(\vec{x})) & \text{(transfer one token)} \\
RESET_t(\vec{x}) & = \tau.(\nu x')POST_t(\vec{x'}) & \text{(flush remaining tokens)} \\
 & & \\
POST_t(\vec{x}) & = \tau.\,(\Pi_{i\in I} Post(t,p_i)\cdot\overline{x_i} \mid LOOP(\vec{x})) & \text{(put tokens)}
\end{array}
$$

Figure 1: Simulating a Petri net with transfer

where again $\vec{x}$ is a vector of $\sharp I$ distinct names x_i, for $i \in I$.

Note that in equation $RESET_t$, the parameter x_k is replaced in the continuation by the generated name x': this means that further transitions affecting p_k will seek to receive or send messages on x' rather than x_k, and since the name x' is fresh, this amounts to resetting place p_k.

Thus, starting from the configuration $\phi(M)$, our process nondeterministically chooses a transition t, consumes the required number of tokens/messages from the preplaces of t (thus blocking if t is not enabled with respect to M), transfers a number of tokens from the origin to the destination of the transfer arc, resets the origin of the transfer arc, and finally puts the correct number of tokens in the postplaces.

When firing an enabled transition t can yield the reduction $M \to M'$, our process, starting from the configuration $\phi(M)$, can take t as its option for the internal choice in $LOOP$, reach $TRANS$ without blocking, take the $STEP$ option exactly as many times as there are tokens in the origin of t's transfer arc, and eventually reach a configuration which only differs from $\phi(M')$ by messages on the channel that was dropped from the parameters when executing $RESET$.

On the other hand, when our process, starting from $\phi(M)$, simulates the firing of a transition t and thus reaches a configuration differing from $\phi(M')$, for some M', only by dead messages, we cannot be sure that all the messages $\overline{x_k}$ were transfered to $\overline{x_l}$ before the $RESET$ option was taken. However, we do know that t is enabled and allows the reduction $M \to M''$ for some M'' that differs from M' only in place p_l, and such that $M'(p_l) \leq M''(p_l)$.

Before we can give a more formal description of the sense in which our process simulates $\mathcal{P}$, we need to introduce the following definitions: a reduction chain $P_1 \to P_2 \to \cdots \to P_n$ will be called *internal* (and denoted $P_1 \to_i^* P_n$) whenever P_j does not contain the process identifier $LOOP$ for any $j \in \{2,\ldots,n-1\}$. This answers the need of talking about reduction chains that remain contained within the simulation of *one* reduction step of $\mathcal{P}$. Moreover, to formalize the sentence "P differs from Q only by dead messages", we introduce the notion of *minimum form*: given a configuration P, the minimum form $mf(P)$ associated to P is the configuration obtained by pruning from P its dead messages, *i.e.*, if

$$P \equiv (\nu\vec{d})\left(\Pi_{i\in I}\overline{c_i}\vec{d_i} \mid \Pi_{j\in J}A_j(\vec{a_j};\vec{b_j})\right)\ ,$$

then

$$mf(P) \equiv (\nu \vec{d}) \left(\Pi_{i \in I'} \overline{c_i} \vec{d_i} \mid \Pi_{j \in J} A_j(\vec{a_j}; \vec{b_j}) \right) \ ,$$

with $I' = \{i \in I \mid \exists j \in J \ a_i \in \{\vec{a_j}\}\}$.

When P is a minimum form and $P \to Q$, we shall write $P \rightsquigarrow mf(Q)$. Note that whenever P and Q are configurations such that $P \to Q$, we have $mf(P) \rightsquigarrow mf(Q)$, and that conversely, if $P \rightsquigarrow Q$, then for any P' such that $mf(P') = P$ there is Q' such that $P' \to Q'$ and $mf(Q') = Q$. We shall use the notation $\rightsquigarrow_i^*$ to denote the internal chains of this abstract reduction on minimum forms with the same meaning as for the standard reduction. Since, for any marking M, $\phi(M)$ is itself a minimum form, we can now state our main proposition as follows.

Proposition 6.9 *The process defined in 1 simulates $\mathcal{P}$ in the following sense:*

(1) *If $M \to M'$ then $\phi(M) \rightsquigarrow_i^+ \phi(M')$*

(2) *If $\phi(M) \rightsquigarrow_i^+ \phi(M')$, then there is $M'' \geq M'$ such that $M \to M''$.*

A (rather straightforward) proof of this proposition can be found in appendix B.

Corollary 6.10 *The coverability problem for Petri Nets with transfer reduces to the control reachability problem for parametric systems with name generation, bounded control, and no name mobility.*

Proof. Suppose we want to decide whether a marking M_f is coverable by $\mathcal{P}$, starting from the initial marking M_0. We add to the process described above the equations

$$\begin{aligned} FINAL_1(\vec{x}) &= x_{i_1}.FINAL_2(\vec{x}) \\ \ldots &= \ldots \\ FINAL_n(\vec{x}) &= x_{i_m}.ACCEPT(\vec{x}) \\ ACCEPT(\vec{x}) &= 0 \ , \end{aligned}$$

where $(i_j)_{j \in \{1,\ldots,m\}}$ is a sequence of indices in I such that $\sharp\{j \in \{1,\ldots,n\} \mid i_j = i\} = M_f(p_i)$ for all $i \in I$, and change the equation associated to $LOOP$ to

$$LOOP(\vec{x}) = (\oplus_{t \in T} PRE_t(\vec{x})) \oplus FINAL_1(\vec{x}) \ ,$$

thus ensuring that for any marking M, we have $M \geq M_f$ if and only if the process identifier $ACCEPT$ is reachable from $\phi(M)$.

Finally, we prove that M_f is coverable from M_0 if and only if $ACCEPT$ is reachable from $\phi(M_0)$. First, if $M_0 \to^* M$ with $M \geq M_f$, applying proposition 6.9(1) inductively yields $\phi(M_0) \rightsquigarrow^* \phi(M)$, and since $M \geq M_f$, we have that $ACCEPT$ is reachable from $\phi(M_0)$ – note that from the point of view of control reachability, the reductions $\to$ and $\rightsquigarrow$ are equivalent in the sense that they associate the same reachability set to any given configuration. Conversely, if $ACCEPT$ is reachable from $\phi(M_0)$, then there must be M such that $M \geq M_f$ and $\phi(M_0) \rightsquigarrow^* \phi(M)$. An induction on the length of this reduction chain, using proposition 6.9(2) and the monotonicity of reduction on Petri Nets with transfer (*i.e.*, the fact that if $M_0 \to M_1$ and $M_0' \geq M_0$ then there is M_1' such that $M_0' \to M_1'$ and $M_1' \geq M_1$), then suffices to conclude that there is $M' \geq M$ such that $M_0 \to^* M'$, and since we have $M' \geq M \geq M_f$, this concludes the proof. □

Remark 6.11 (on complexity) *Define* $2^n_0 = n$ *and* $2^n_{k+1} = 2^{2^n_k}$ *so that* 2^n_k *is a 'tower' of height* k *of exponentials. Let* n *be the size of a system of parametric equations with bounded control (one continuation or an internal choice) and without name generation obtained, say, by counting the number of symbols. It is clear that there is a* k *such that the size of the parameterless system and the related Petri Net described in section 3 is bounded by* 2^n_k*. Rackoff's upper bound [26] shows that the coverability problem for Petri Nets can be solved in exponential space in the size of the net. The number of problems that needs to be solved is also at worse exponential in* n*. Thus there is some* $k' \geq k$ *such that the control reachability problem for such systems can be solved in* $O(2^n_{k'})$*. A recent result by Ph. Schnoebelen [28] claims that the coverability problem for Petri Nets with transfer is* not *primitive recursive. Coupled with the encoding above this shows that if we add name generation to the systems just described then the complexity of deciding the control reachability problem goes beyond primitive recursion.*

7 Conclusion

We regard the decidability and undecidability results presented here as a step towards the systematic introduction of *approximated* decision methods for the asynchronous π-calculus and related formalisms. It seems natural to factorize such approximation methods through a translation to (possibly extended) Petri Nets.

Going from the π-calculus to Petri Nets, one general remark is that identifying names in the π-calculus produces a conservative approximation of the reduction relation – *i.e.*, the approximation can do *more* reductions; this is not true if a conditional like (7) is introduced, however such a conditional can be approximated by internal choice. This suggests exploring approximation methods where a finite space of names is allocated to every name generator. Then, for instance, starting from a π-calculus with a unique receiver condition we can *force* the bounded input condition and develop a translation into Petri Nets. Once the behaviour of a system is mapped to a Petri Net further standard approximation and acceleration techniques are readily available based, *e.g.*, on semi-linear sets (see, *e.g.*, [29], for an up to date survey).

In another direction, let us note that the unique receiver condition and the bounded input condition are instrumental to *name collection* and *name generation* operations, respectively. We have selected them because they are effective and simple, however more general conditions might be found that still allow for an effective implementation of the collection and generation operations in Petri Nets. For instance, one could try to go from the unique receiver condition to a weaker 'locality' condition requiring that received names can be used only in output.

Yet in another direction, we point out that the decision method for Petri Net with transfer is an instance of a general method that applies to so called *well structured systems* [1, 13]. Such systems include, *e.g.*, finite state systems with *lossy* channels. Thus a possible research direction is to look at variations over the π-calculus model that still admit a translation into well structured systems.

Finally we remark that in this paper we have focused on the control reachability problem. As mentioned in the introduction, other problems such as boundedness, deadlock, or liveness deserve to be investigated. We expect that for these problems too it will be fruitful to look at translations into Petri Nets. As a first concrete example, we can offer the analysis of the *message deliverability property* for the asynchronous π-calculus defined in [3]. This property requires that every emitted message has a chance of being received. It turns out that this property is quite close to the liveness property of Petri Nets and that it can be shown to be

decidable by similar techniques in the case *without* name generation.

A Proof of lemma 6.6

Suppose $P \mathcal{R} Q$, we show that:

(1) If $P \to P'$ then for some Q', $Q \to Q'$ and $P' \mathcal{R} Q'$.

(2) If $Q \to Q'$ then for some P', $P \to P'$ and $P' \mathcal{R} Q'$.

(1) We develop the analysis in a simplified case thus saving some notation: (i) there are just two threads (that can generate names), (ii) all channels are monadic and threads have just one output parameter, and (iii) all names in P are restricted. Then suppose:

$$P \equiv (\nu \vec{c}_I)(\nu \vec{d}_I)(\nu \vec{c}) \quad (\overline{c}c' \mid \Pi_{h \in H_Z} \overline{c}_h c'_h \mid \Pi_{h \in H_L} \overline{c}_h c'_h \mid A(\vec{c}_I; c_O) \mid A'(\vec{d}_I; d_O))$$

$$\text{where:} \quad c_h \in \begin{cases} \{\vec{c}\} & \text{if } h \in H_Z \quad \text{(dead names)} \\ \{\vec{c}_I, \vec{d}_I\} & \text{if } h \in H_L \quad \text{(live names)} \end{cases}$$

$$\text{and} \qquad A(\vec{a}_I; a_O) = a(a').(\nu \vec{c}'')(\Pi_{k \in K} \overline{b}_k b'_k \mid B(\vec{b}_I; b_O)) \ .$$

$$\text{Let us set:} \begin{cases} \sigma_I = [\vec{c}_I / \vec{a}_I], & \sigma_I(a) = c, \quad \vec{c}_I \equiv \vec{c}_{I,l}, \vec{c}_{I,d}, \\ \sigma_I(\vec{b}_I) \equiv \vec{c}_{I,l}, \vec{c}'', & D = \{\vec{c}_{I,d}\} \ . \end{cases}$$

Under these hypotheses, P consumes the message $\overline{c}c'$ and we have that $P \to P'$ with:

$$\begin{array}{lll} P' \equiv & (\nu \vec{c}_{I,l}, \vec{c}'')(\nu \vec{d}_I)(\nu \vec{c}, \vec{c}_{I,d}) & \\ & (\Pi_{h \in H_Z} \overline{c}_h c'_h \mid & (i) \\ & \Pi_{h \in H'_Z} \overline{c}_h c'_h \mid \Pi_{h \in H_L \setminus H'_Z} \overline{c}_h c'_h \mid & (ii) \\ & \sigma(\Pi_{k \in K_Z} (\overline{b}_k b'_k) \mid \Pi_{k \in K \setminus K_Z} (\overline{b}_k b'_k)) \mid & (iii) \\ & B(\vec{c}_{I,l}, \vec{c}''; \sigma b_O) \mid & (iv) \\ & A'(\vec{d}_I; d_O)), & (v) \end{array}$$

$$\text{where:} \begin{cases} \sigma & = [\vec{c}_I / \vec{a}_I, c_O / a_O, c' / a'] \\ H'_Z & = \{h \in H_L \mid c_h \in D\} \\ K_Z & = \{k \in K \mid \sigma(b_k) \in D \cup \{\vec{c}\}\} \ . \end{cases}$$

From $P \mathcal{R} Q$ we know that Q is obtained from P by application of a substitution τ and pruning of dead messages. Suppose the live names in P are renamed so that the substitution τ is the identity on live names:

$$\tau \equiv [\vec{c}_I / \vec{c}_I, \vec{d}_I / \vec{d}_I, \vec{g} / \vec{c}] \ .$$

Then we can write Q as follows:

$$Q \equiv (\overline{\langle c, \tau c' \rangle} \mid \Pi_{h \in H_Z} \overline{z} \mid \Pi_{h \in H_L} \overline{\langle c_h, \tau c'_h \rangle} \mid \langle A, \vec{c}_I, \tau c_O \rangle \mid \langle A', \vec{d}_I, \tau d_O \rangle) \ .$$

The parameterless system includes the equation:

$$\langle A, \vec{c}_I, \vec{c}_O \rangle = \Sigma_{d' \downarrow a'} \langle c, d' \rangle . Q_{d'}$$

and taking $d' = \tau c'$ we have:

$$Q_{\tau c'} \equiv Col(D).(\Pi_{k \in K_D} \overline{z} \mid \Pi_{k \in K \setminus K_D} \sigma_{\tau c'} \overline{\langle b_k, \sigma_D b'_k \rangle} \mid \langle B, \vec{c}_{I,l}, \vec{c}'', \sigma_{\tau c'}(\sigma_D b_O) \rangle)$$

We rename the generated names $\vec{c}''$ in P so that they coincide with the ones selected by the function *choose*. Then the substitution $\sigma_{\tau c'}$ considered here is

$$\sigma_{\tau c'} = [\vec{c}_I / \vec{a}_I, \tau c_O / a_O, \tau c' / a']$$

and $K_D = \{k \in K \mid \sigma_{\tau c'}(b_k) \in G \cup D\}$.

Now we observe that for a formal parameter $b \in \{\vec{a}_I, a_O, a', \vec{c}''\}$ we have:

$$\sigma_{\tau c'}(b) = \begin{cases} \sigma(b) & \text{if } b \in \{\vec{a}_I, \vec{c}''\} \\ \tau(\sigma b) & \text{if } b \in \{a_O, a'\} \ . \end{cases}$$

It follows that:

$$\sigma(b) \in D \cup \{\vec{c}\} \ \text{ iff } \ \sigma_{\tau c'}(b) \in D \cup G \tag{16}$$

and from this we derive that $K_Z = K_D$.

Let us now see how Q simulates P. By consuming $\overline{\langle c, \tau c' \rangle}$ and unfolding the equation above we obtain that $Q \to Q'$ with:

$$\begin{array}{rcll} Q' & \equiv & \Pi_{h \in H_Z} \overline{z} \mid & (i) \\ & & Col(D)(\Pi_{h \in H_L} \overline{\langle c_h, \tau c'_h \rangle}) \mid & (ii) \\ & & \Pi_{k \in K_D} \overline{z} \mid \Pi_{k \in K \setminus K_D} \sigma_{\tau c'} \overline{\langle b_k, \sigma_D b'_k \rangle} \mid & (iii) \\ & & \langle B, \vec{c}_{I,l}, \vec{c}'', \sigma_{\tau c'}(\sigma_D b'_O) \rangle \mid & (iv) \\ & & Col(D)(\langle A', \vec{d}_I, \tau d_O \rangle) & (v) \end{array}$$

where we use $Col(D)(R)$ as an abbreviation for the result of applying the transfer operations in $Col(D)$ to the process R. We define the substitution:

$$\tau' \equiv [\vec{c}_{I,l} / \vec{c}_{I,l}, \vec{d}_I / \vec{d}_I, \vec{c}'' / \vec{c}'', \vec{g} / \vec{c}, \vec{g} / \vec{c}_{I,d}]$$

and we apply it to show that $P' \ \mathcal{R} \ Q'$. First we note that τ' is injective on live names and maps dead names to generic names. Then we compare the parts $(i) - (v)$ of P' and Q'.

(i) By definition of $\mathcal{R}$ a message addressed to a dead name is mapped to $\overline{z}$.

(ii) If $h \in H'_Z$ then $c_h \in D$ and by definition of the collection operation $Col(D)\overline{\langle c_h, \tau c'_h \rangle} = \overline{z}$. On the other hand, if $h \in H_L \setminus H'_Z$ then

$$Col(D)\overline{\langle c_h, \tau c'_h \rangle} = \overline{\langle c_h, [\vec{g}/\vec{c}_{I,d}](\tau c_h) \rangle} = \overline{\langle c_h, \tau' c_h \rangle} \ .$$

(iii) We have remarked above that $K_Z = K_D$. If $k \in K_Z$ the same reasoning as in *(i)* applies. Otherwise, if $k \in K \setminus K_Z$ we show:

$$\tau'(\sigma \langle b_k, b'_k \rangle) = \sigma_{\tau c'} \langle b_k, \sigma_D b'_k \rangle \ .$$

Since σb_k is live we have that $\tau'(\sigma b_k) = \sigma b_k$. On the other hand, we have observed above that:

$$\sigma_{\tau c'}(b_k) = \begin{cases} \sigma(b_k) & \text{if } b_k \in \{\vec{a}_I, \vec{c}''\} \\ \tau(\sigma b_k) & \text{if } b_k \in \{a_O, a'\}, \end{cases}$$

and in the second case we observe that $\tau'(\sigma b_k) = \sigma b_k$ implies $\tau(\sigma b_k) = \sigma b_k$.

Next consider $\sigma_D b'_k$. By definition of σ_D, we have:

$$\sigma_D(b'_k) = \begin{cases} g & \text{if } \sigma_{\tau c'} b'_k \in D \cup G \\ b'_k & \text{otherwise.} \end{cases}$$

Then we apply the observation (16). In the first case, $\sigma_{\tau c'} b'_k \in D \cup G$ implies $\sigma b'_k \in D \cup \{\vec{c}\}$ which implies $\tau'(\sigma b'_k) = g$. Therefore $\tau'(\sigma b'_k) = g = \sigma_{\tau c'}(g)$. In the second case, $\sigma_{\tau c'} b'_k = \sigma b'_k = \tau'(\sigma b'_k)$.

(*iv*) We reason as in case *(iii)*.

(*v*) We reason as in case *(ii)*.

(2) Suppose now Q performs a reduction to Q'. We can take P, Q, Q', and τ as in case (1). Because the substitution τ is injective on i/o parameters we can trace back the redex from Q to P and perform a corresponding reduction to P'. Then the (reversible) computations we performed in the previous case show that $P' \mathcal{R} Q'$. □

B Proof of proposition 6.9

We begin by proving two lemmas.

Lemma B.1 *The transition t is enabled with respect to the marking M if and only if*

$$\begin{aligned} &(\nu \vec{x})\,(\Pi_{i \in I} M(p_i) \cdot \overline{x_i} \mid PRE_{t,1}(\vec{x})) \\ &\quad \rightsquigarrow_i^* (\nu \vec{x})\,(\Pi_{i \in I}\,(M(p_i) - Pre(p_i, t)) \cdot \overline{x_i} \mid TRANS_t(\vec{x}))\ . \end{aligned}$$

Proof. If t is enabled with respect to M, then for all $i \in I$, $M(p_i) \geq Pre(p_i, t)$, and by construction of the equations $PRE_{t,j}$ for $j \in \{1, \ldots, m\}$, we have the required reduction chain. Conversely, if the reduction chain given above holds, for all $i \in I$, we have $M(p_i) \geq Pre(p_i, t)$, and t is enabled. □

Lemma B.2 *For all markings M and M',*

$$\begin{aligned} Q \equiv\ &(\nu \vec{x})\,(\Pi_{i \in I} M(p_i) \cdot \overline{x_i} \mid TRANS_t(\vec{x})) \\ &\rightsquigarrow_i^* (\nu \vec{x})\,(\Pi_{i \in I} M'(p_i) \cdot \overline{x_i} \mid POST_t(\vec{x})) \end{aligned}$$

if and only if

$$\begin{cases} M'(p_k) = 0 \\ M(p_l) \leq M'(p_l) \leq M(p_l) + M(p_k) \\ M'(p) = M(p) \text{ for all } p \in P \setminus \{p_k, p_l\}\ . \end{cases}$$

Proof. ($\Rightarrow$) The configuration Q can only reduce to

$$(\nu \vec{x})\,(\cdots \mid RESET_t(\vec{x}))$$

after having performed a number of $TRANS_t \to STEP_t \to TRANS_t$ loops. Since each loop consumes one message $\overline{x_k}$ and outputs one $\overline{x_l}$, the reachable configurations are the

$$(\nu \vec{x})\,\left(\Pi_{i \in I} M''(p_i) \cdot \overline{x_i} \mid RESET_t(\vec{x})\right)\ ,$$

for all M'' satisfying

$$\begin{cases} M''(p_k) = M(p_k) - n \\ M''(p_l) = M(p_l) + n \\ M''(p) = M(p) \text{ for all } p \in P \setminus \{p_k, p_l\} \ , \end{cases}$$

for some $n \in \{0, \ldots, M(p_k)\}$.

From each of these configurations, the only possible reduction step is to

$$Q' \equiv (\nu \vec{x}, x') \left(\Pi_{i \in I} M''(p_i) \cdot \overline{x_i} \mid POST_t \left([x'/x_k] \, \vec{x} \right) \right) ,$$

where x' is a fresh name. By α-renaming, we have

$$Q' \equiv (\nu \vec{x}, x'') \left(M''(p_k) \cdot \overline{x''} \mid \Pi_{i \in I \setminus \{k\}} M''(p_i) \cdot \overline{x_i} \mid POST_t(\vec{x}) \right) \ ,$$

and

$$mf(Q') \equiv (\nu \vec{x}) \left(\Pi_{i \in I \setminus \{k\}} M''(p_i) \cdot \overline{x_i} \mid POST_t(\vec{x}) \right) \ .$$

Therefore, M' must satisfy $M'(p_k) = 0$ and $M'(p) = M''(p)$ for all $p \neq p_k$. But then M' meets the conditions stated in the lemma.

($\Leftarrow$) For the converse, if M' satisfies the conditions, then after performing $n = M'(p_l) - M(p_l)$ iterations of the *STEP* loop,

$$Q \rightarrow_i^* (\nu \vec{x}) \left(\Pi_{i \in I} M''(p_i) \cdot \overline{x_i} \mid RESET_t(\vec{x}) \right) \ ,$$

with

$$\begin{cases} M''(p_k) = M(p_k) - n \\ M''(p_l) = M(p_l) + n = M'(p_l) \\ M''(p) = M(p) \text{ for all } p \in P \setminus \{p_k, p_l\} \ , \end{cases}$$

and by suitably α-renaming the configuration reached after the *RESET* reduction step, we have

$$Q \rightsquigarrow_i^* (\nu \vec{x}) \left(\Pi_{i \in I} M'(p_i) \cdot \overline{x_i} \mid POST_t(\vec{x}) \right) \ .$$

□

Proof of (1) Let t be the transition fired for the reduction $M \rightarrow M'$. We have

$$\phi(M) \rightsquigarrow (\nu \vec{x}) \left(\Pi_{i \in I} M(p_i) \cdot \overline{x_i} \mid PRE_{t,1}(\vec{x}) \right) \ ,$$

and since t is enabled with respect to M, we can apply lemma B.1 to get

$$\phi(M) \rightsquigarrow_i^* (\nu \vec{x}) \left(\Pi_{i \in I} \left(M(p_i) - Pre(p_i, t) \right) \cdot \overline{x_i} \mid TRANS_t(\vec{x}) \right) \ .$$

Now, let M_0 be the marking defined by $M_0(p) = M(p) - Pre(p, t)$, and let M_1 be defined by

$$\begin{cases} M_1(p_k) = 0 \\ M_1(p_l) = M_0(p_l) + M_0(p_k) \\ M_1(p) = M_0(p) \text{ for all } p \in P \setminus \{p_k, p_l\} \ . \end{cases}$$

Lemma B.2 then yields

$$\phi(M) \rightsquigarrow_i^* (\nu \vec{x}) \left(\Pi_{i \in I} M_1(p_i) \cdot \overline{x_i} \mid POST_t(\vec{x}) \right) \ ,$$

so that we eventually get

$$\phi(M) \rightsquigarrow_i^* (\nu\vec{x})\,(\Pi_{i\in I}\,(M_1(p_i) + Post(t,p_i)) \cdot \overline{x_i} \mid LOOP(\vec{x}))\ .$$

All that remains to be checked is that $M' = M_1 + Post(t,\_)$. To this end, let $p \in P$: if p is the origin of t's transfer arc, we have

$$M'(p_k) = 0 = M_1(p_k) + Post(t,p_k)\ .$$

If p is the destination of t's transfer arc, we have

$$M'(p_l) = M(p_k) + M(p_l) = M_0(p_k) + M_0(p_l) = M_1(p_l) = M_1(p_l) + Post(t,p_l)$$

(we remind the reader that $Pre(p_l,t) = Pre(p_k,t) = Post(t,p_l) = Post(t,p_k) = 0$). In all other cases, we have

$$M'(p) = M(p) - Pre(p,t) + Post(t,p) = M_0(p) + Post(t,p) = M_1(p) + Post(t,p)\ .$$

□

Proof of (2) The first step of the reduction has to be

$$\phi(M) \rightsquigarrow (\nu\vec{x})\,(\Pi_{i\in I} M(p_i) \cdot \overline{x_i} \mid PRE_{t,1}(\vec{x}))\ ,$$

for some t. Then, the only possible reduction path leads to

$$\phi(M) \rightsquigarrow_i^* (\nu\vec{x})\,(\Pi_{i\in I}\,(M(p_i) - Pre(p_i,t)) \cdot \overline{x_i} \mid TRANS_t(\vec{x}))\ .$$

At this point, we can use lemma B.1 to ensure that t is enabled with respect to M. Let M'' be the marking obtained by firing t from M. We must simply prove $M'' \geq M'$.

Before reaching $\phi(M')$, our process must reach a configuration containing $POST_t$, so let M_1 be a marking such that

$$\begin{array}{rcl} \phi(M) \rightarrow_i^* & (\nu\vec{x})\,(\Pi_{i\in I} M_1(p_i) \cdot \overline{x_i} \mid POST_t(\vec{x})) \\ \rightarrow & \phi(M')\ . \end{array}$$

Note that we must have $M'(p) = M_1(p) + Post(t,p)$ for all $p \in P$.

By applying lemma B.2, we get

$$\left\{ \begin{array}{l} M_1(p_k) = 0 \\ M_0(p_l) \leq M_1(p_l) \leq M_0(p_l) + M_0(p_k) \\ M_1(p) = M_0(p) \text{ for all } p \in P \setminus \{p_k, p_l\}\ , \end{array} \right.$$

where $M_0(p) = M(p) - Pre(p,t)$ for all $p \in P$. We conclude that $M'' \geq M'$, since

- $M''(p_k) = M'(p_k) = 0$,
- $M''(p_l) = M(p_l) + M(p_k) = M_0(p_l) + M_0(p_k) \geq M_1(p_l) = M'(p_l)$,
- and for all other p,

$$\begin{array}{rl} M''(p) & = M(p) - Pre(p,t) + Post(t,p) \\ & = M_0(p) + Post(t,p) \\ & = M_1(p) + Post(t,p) \\ & = M'(p)\ . \end{array}$$

□

References

[1] P. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *Proc. IEEE-LICS*, 1996.

[2] R. Amadio. On modeling mobility. *Theoretical Computer Science*, 240:147–176, 2000.

[3] R. Amadio, G. Boudol, and C. Lhoussaine. On message deliverability and non-uniform receptivity. http://www.cmi.univ-mrs.fr/~amadio, January 2002.

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

[5] R. Amadio and W. Charatonik. On name generation and set-based analysis in Dolev-Yao model. Technical report, 2002. RR-4379, INRIA.

[6] R. Amadio and C. Meyssonnier. On the decidability of fragments of the asynchronous π-calculus. In *Proc. EXPRESS01, Electronic Notes in Theoretical Computer Science*, volume 52.1, 2001. Also appeared as RR-INRIA 4241.

[7] T. Araki and T. Kasami. Some decision problems related to the reachability problem for Petri nets. *Theoretical computer science*, 3(1):85–104, 1977.

[8] G. Boudol. Asynchrony and the π-calculus. Technical report, *RR 1702, INRIA, Sophia-Antipolis*, 1992.

[9] M. Dam. Model checking mobile processes (full version). *Information and Computation*, 1996. Also appeared in Proc. Concur'93.

[10] C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In *Proc. ICALP 98*. Springer Lect. Notes in Comp. Sci. 1443, 1998.

[11] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. Formal methods and security protocols, FLOC Workshop, Trento*, 1999.

[12] J. Esparza. Decidability and complexity of Petri net problems - an introduction. In *Lectures on Petri Nets I: Basic Models, Springer Lect. Notes in Comp. Sci. 1491*. Springer, 1998.

[13] A. Finkel and Ph. Schnoebelen. Well structured systems everywhere. *Theoretical Computer Science*, 256(1-2):63–92, 2001.

[14] C. Fournet and G. Gonthier. The reflexive CHAM and the join-calculus. *Proc. ACM Principles of Prog. Lang.*, 1996.

[15] 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.

[16] 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.

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

[18] 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.

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

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

[21] R. Karp and R. Miller. Parallel program schemata. *J. Comp. Sys. Sciences*, 3:147–195, 1969.

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

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

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

[25] B. Pierce and D. Turner. Pict: a programming language based on the π-calculus. University of Cambridge, 1996.

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

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

[28] Ph. Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Information Processing Letters*, 2002. To appear, draft available at http://www.lsv.ens-cachan.fr/~phs.

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

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