Math. Struct. in Comp. Science (2009), *vol.* 19, *pp.* 1223–1263.
doi:10.1017/S0960129509990181

Deciding reachability problems in Turing-complete fragments of Mobile Ambients

NADIA BUSI and GIANLUIGI ZAVATTARO[†]

[†]*Department of Computer Science, Università di Bologna, Italy*
Email: zavattar@cs.unibo.it

Received 24 April 2009; revised 8 July 2009

In memory of Nadia Busi

The calculus of Mobile Ambients was proposed by Cardelli and Gordon as a foundational calculus for mobile computing. Since its introduction, the computational strength and the decidability of properties have been investigated for several fragments and variants of the standard calculus. We consider the problem of reachability and characterise a public (that is, restriction-free) fragment for which it is decidable. This fragment is obtained by removing the `open` capability and restricting the application of the replication operator to guarded processes only. This decidability result may appear surprising in combination with the fact that the same fragment was shown to be Turing complete by Maffeis and Phillips. Finally, we extend our decidability result in two ways: we first prove the decidability of a more general property called *target reachability* (according to which the target of interest for the reachability analysis consists of a possibly infinite set of processes) and then show that our decidability results also hold for a more general calculus, which includes the sophisticated communication mechanisms of Boxed Ambients, which is the most relevant variant of Mobile Ambients without the `open` capability.

1. Introduction

The calculus of Mobile Ambients (Cardelli and Gordon 2000a), MA for short, is a well-known formalism for describing distributed and mobile systems in terms of *ambients*. An ambient is a named collection of active processes and nested sub-ambients. In the pure (that is, without communication) version of MA, only three mobility primitives are used for ambient and process interaction: `in` and `out` for ambient movement, and `open` to dissolve ambient boundaries.

More precisely, a process performs an `in` m primitive to instruct its surrounding ambient to move inside a sibling ambient named m, `out` m to exit its parent ambient named m, and `open` m to dissolve the boundary of an ambient named m located in the same ambient as the process.

Since its introduction, the calculus of Mobile Ambients has attracted widespread interest, and it has been used as a starting point for investigating the foundations of a great variety of mobile computing models. Consider, for example: Mobile Safe Ambients (Levi and Sangiorgi 2003), which were used to investigate security issues in mobile systems; the Push and Pull Ambient Calculus (Phillips and Vigliotti 2002), which formalises objective rather

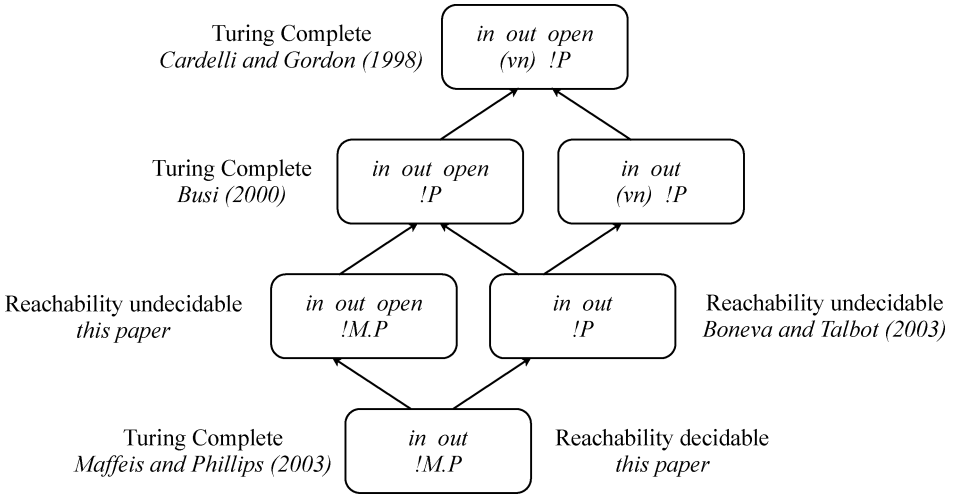

Fig. 1. Overview of the results for Turing completeness and the (un)decidability of reachability in pure Mobile Ambients (arrows represent the sublanguage relation).

than subjective mobility; Boxed Ambients (Bugliesi *et al.* 2004), which were used to model systems in which ambient boundaries cannot be dissolved and a direct communication between parent and child ambients is permitted; and BioAmbients (Regev *et al.* 2004), which were defined to model the behaviour of biological system.

Following in the tradition of process calculi, MA and its dialects have been equipped with a rich variety of formal tools for reasoning about and verifying properties of systems specified with these calculi. Just to mention a few of these tools, there are the behavioural semantics investigated in Merro and Hennessy (2002) or the type system (Cardelli *et al.* 2002) and logics (Cardelli and Gordon 2000b) used to reason about both the behaviour and spatial structure of ambients. Another line of research looks at the expressiveness of these calculi to distinguish between necessary and redundant features.

The computational strength of MA has been investigated in several papers. The most interesting result is that many of the MA operators are not required in the proof of Turing completeness for the calculus. Figure 1 shows a history of the main results on Turing completeness for fragments of MA. In their first paper on MA (Cardelli and Gordon 2000a), Cardelli and Gordon showed how to model Turing machines in MA. This encoding of Turing machines made use of all the capabilities of MA (`in`, `out` and `open`), as well as the restriction operator. Subsequently, Busi (2000) proved that restriction is unnecessary. This result was proved by showing how to model Random Access Machines (Shepherdson and Sturgis 1963) (which are a well-known register-based Turing-complete formalism) in MA without using the restriction operator. More recently, Maffeis and Phillips (2004) proved that the `open` capability is unnecessary by presenting an improved modelling of Random Access Machines without using `open`; moreover, the replication operator $!P$ is only applied to prefixed processes of the form $M.P$.

The proofs of Turing completeness mentioned above have as a direct consequence that termination is not decidable in the fragments of MA considered. Another property of processes, which is in some cases even more interesting than termination, is process

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

reachability: the reachability problem consists of verifying whether a target process can be reached from a source process. As an example of the relevance of reachability analysis, consider the system

$$intruder[P] \mid firewall[Q]$$

where an *intruder* running the program P attacks a $firewall$ executing the program Q. It would clearly be of interest if we could check whether the system

$$firewall[\ intruder[P'] \mid Q'],$$

where the intruder has succeeded in entering the firewall (for some residual program P' and Q'), can be reached.

To the best of our knowledge, the first work devoted to investigating reachability in MA was Boneva and Talbot (2003), which proved that reachability is undecidable even in a minimal fragment of pure MA where both the restriction operator and `open` capability are removed.

Figure 1 shows the lattice of fragments of MA considered in the papers mentioned above, and indicates the known results for their Turing completeness and the undecidability of reachability. These papers left as an open problem the (un)decidability of reachability for the fragment at the bottom of the lattice (the one considered by Maffeis and Phillips) and its extension having the `open` capability. In this paper we close these two open problems by showing that reachability is undecidable in the latter case, but turns out to be decidable in the former. In this way we completely characterise the fragments in Figure 1 as far as their Turing completeness and (un)decidability of reachability are concerned.

The decidability of reachability in the fragment at the bottom of the lattice in Figure 1 may appear surprising in light of the result on Turing completeness proved by Maffeis and Phillips. Intuitively, the decidability of reachability follows from the following monotonicity property deriving from the absence of the `open` capability (and from the impossibility of applying the replication operator to ambients): *during a computation, the number of active ambients cannot decrease*. According to this property, all possible computations from a given source process P to a given target process T traverse processes with a known bounded number of active ambients. The existence of this bound allows us to model all these possible computations as computations in finite Petri nets, a formalism in which the reachability problem is decidable.

An additional contribution of the current paper is a generalisation of the above decidability result to a more general problem, called *target reachability*, which we have defined to cope with a lack of expressiveness in traditional reachability in the context of MA. Returning to the example of the *intruder* and the $firewall$, traditional reachability allows us to check whether the source process

$$intruder[P] \mid firewall[Q]$$

has a computation leading to the target process

$$firewall[\ intruder[P'] \mid Q']$$

for some instantiated processes P' and Q'. However, we may only be interested in the structure of the target process (that is, the intruder is inside the firewall), so in order to

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

abstract away from the specific residual programs within those ambients (that is, abstract away from P' and Q'), we should universally quantify the reachability analysis on all possible processes P' and Q'.

These considerations led us introduce the *target reachability* problem. The idea is to consider an entire (possibly infinite) set of targets rather than just one specific target process. This set is specified by means of a finite expression that indicates a precise structure of ambient nesting that all the processes belonging to this set should have, and expresses constraints on the kind of processes residing in these ambients. Such constraints allow us to define lower and upper bounds on the number of instances of some given processes. In the above example of the *intruder* and the *firewall*, we can express the target processes of interest by

$$firewall[\ intruder[\texttt{any}] \mid \texttt{any}]$$

where `any` indicates the possibility of any process residing in the ambients under consideration, that is, every process P' (respectively, Q') could reside in the ambient *intruder* (respectively, *firewall*).

As another example of target reachability analysis, consider the following modelling of a drug delivery system:

$$TransportMolecule[P] \mid Blood[\ Muscular[Q] \mid Connective[R] \mid S\ ].$$

Here we assume that a transport molecule is prepared to be injected into the bloodstream of a patient for the delivery of some drug molecules to the muscle tissue, but not to the connective tissue. We could express this requirement by specifying that at least one of the target processes described by the following expressions should be reachable:

$$\begin{aligned} Blood[\ & Muscular[10 \leqslant Drug \leqslant 20] \mid Connective[0 \leqslant Drug \leqslant 0] \mid \\ & TransportMolecule[\texttt{any}] \mid \texttt{any}\ ]. \end{aligned}$$

According to this expression, between 10 and 20 drug molecules should be delivered to the muscle tissue but none to the connective tissue. We could also take into consideration the final contents of the transport molecule by considering the target processes associated with the expression

$$\begin{aligned} Blood[\ & Muscular[10 \leqslant Drug \leqslant 20] \mid Connective[0 \leqslant Drug \leqslant 0] \mid \\ & TransportMolecule[0 \leqslant BadResidual \leqslant 5 \mid 0 \leqslant GoodResidual \leqslant \infty] \mid \texttt{any}\ ], \end{aligned}$$

which additionally requires that only two kinds of residuals (that is, good and bad residuals) can be left inside the transport molecule, and also fixes a precise upper bound on the number of bad residuals.

As a final contribution, we show how to apply our technique for analysing target reachability to Boxed Ambients (Bugliesi *et al.* 2004), which is the most relevant `open`-free variant of MA for us. In Boxed Ambients the possibility of dissolving ambient boundaries is replaced by a sophisticated form of parent–child communication. We show how to limit this form of communication, and how to extend our technique so that we can also do reachability analysis in this non-pure (that is, with communication) dialect of MA.

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

Structure of the paper

In Section 2 we present the syntax and semantics of the calculi considered. In Section 3 we give the proof of the undecidability of reachability in the fragment of pure MA without restriction but with guarded replication, and in Section 4 we give the proof of the decidability of reachability in the fragment in which the `open` capability is also removed. In Section 5 we introduce the target reachability problem and prove that this more general problem is decidable too. In Section 6 we address the problem of applying the techniques we have developed to Boxed Ambients. Finally, we present some concluding remarks in Section 7.

Preliminary versions of this paper appeared in Busi and Zavattato (2005a; 2005b): in Busi and Zavattato (2005a) we discussed the (un)decidability of reachability for fragments of MA; and in Busi and Zavattaro (2005b) we introduced the target reachability problem and discussed its decidability for (a fragment of) Boxed Ambients. In the current paper we present these results in a uniform and incremental way, and give details of the proofs that were merely sketched in the previous papers.

2. Pure public Mobile Ambients

The pure public Mobile Ambients calculus, pMAfor short, corresponds to the restriction-free fragment of the version of Mobile Ambients without communication defined in Cardelli and Gordon (2000a).

Definition 2.1 (pMA). Let *Name*, ranged over by $n, m, \ldots$, be a denumerable set of ambient names. The terms of pMA are defined by the grammar

$$\begin{array}{lcl} P & ::= & \mathbf{0} \mid M.P \mid n[P] \mid P|P \mid !P \\ M & ::= & \mathtt{in}\, n \mid \mathtt{out}\, n \mid \mathtt{open}\, n. \end{array}$$

We use $\prod_k P$ to denote the parallel composition of k instances of the process P (if $k = 0$, then $\prod_k P = \mathbf{0}$), and $\prod_{i \in 1 \ldots k} P_i$ to denote the parallel composition of the indexed processes P_i.

The term $\mathbf{0}$ represents the inactive process (and is usually omitted). $M.P$ is a process guarded by one of the three mobility primitives (which were discussed in the Introduction): after the execution of the primitive, the process behaves like P. The processes $M.P$ are referred to as *guarded processes* in the rest of the paper. We use the term $n[P]$ to denote an ambient named n containing process P. A process may also be the parallel composition $P|P$ of two subprocesses. Finally, the replication operator $!P$ is used to create in parallel an unbounded number of instances of the process P.

The operational semantics of pMA is defined in terms of a structural congruence plus a reduction relation.

Definition 2.2 (Structural congruence). The structural congruence $\equiv$ is the smallest congruence relation satisfying:

$$\begin{array}{rcl} P \mid \mathbf{0} & \equiv & P \\ P \mid Q & \equiv & Q \mid P \\ P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R \\ !P & \equiv & P \mid !P\ . \end{array}$$

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

Definition 2.3 (Reduction relation). The reduction relation is the smallest relation $\rightarrow$ satisfying the following axioms and rules:

(1) $n[\texttt{in}\, m.P \mid Q] \mid m[R] \;\rightarrow\; m[n[P \mid Q] \mid R]$.

(2) $m[n[\texttt{out}\, m.P \mid Q] \mid R] \;\rightarrow\; n[P \mid Q] \mid m[R]$.

(3) $\texttt{open}\, n.P \mid n[Q] \;\rightarrow\; P \mid Q$.

(4) $\dfrac{P \;\rightarrow\; Q}{P \mid R \;\rightarrow\; Q \mid R}$.

(5) $\dfrac{P \;\rightarrow\; Q}{n[P] \;\rightarrow\; n[Q]}$.

(6) $\dfrac{P' \equiv P \quad P \;\rightarrow\; Q \quad Q' \equiv Q}{P' \;\rightarrow\; Q'}$.

As usual, we use $\rightarrow^*$ to denote the reflexive and transitive closure of $\rightarrow$. If $P \rightarrow^* Q$, we say that Q is a *derivative* of P. The reachability problem consists of checking, given two processes P and Q, whether Q is a derivative of P, that is, checking if $P \rightarrow^* Q$.

Axioms (1), (2) and (3) describe the semantics of the three primitives $\texttt{in}$, $\texttt{out}$ and $\texttt{open}$, respectively. A process inside an ambient n can perform an $\texttt{in}\, m$ operation in the presence of a sibling ambient m; if the operation is executed, the ambient n moves inside m. If inside an ambient m there is an ambient n with a process performing an $\texttt{out}\, m$ action, the effect is to move the ambient n outside the ambient m. Finally, a process performing an $\texttt{open}\, n$ operation can remove the boundary around an ambient $n[Q]$ composed in parallel with it.

Rules (4) and (5), respectively, are the contextual rules that indicate that a process can also move when it is in parallel with another process and when it is inside an ambient. Finally, rule (6) is used to ensure that two structurally congruent terms have the same reductions.

In this paper we consider three fragments of pMA, namely $\text{pMA}_{g!}$, $\text{pMA}^{-\texttt{open}}$ and $\text{pMA}_{g!}^{-\texttt{open}}$– we show that reachability is undecidable for $\text{pMA}_{g!}$ and $\text{pMA}^{-\texttt{open}}$, but decidable for $\text{pMA}_{g!}^{-\texttt{open}}$.

Definition 2.4. We define the folowing fragments of pMA:

— $\text{pMA}_{g!}$ permits only guarded replication, that is, it restricts the application of the replication operator to guarded processes:

$$\begin{array}{rcl} P & ::= & \mathbf{0} \mid M.P \mid n[P] \mid P|P \mid !M.P \\ M & ::= & \texttt{in}\, n \mid \texttt{out}\, n \mid \texttt{open}\, n. \end{array}$$

— $\text{pMA}^{-\texttt{open}}$ removes the $\texttt{open}$ capability:

$$\begin{array}{rcl} P & ::= & \mathbf{0} \mid M.P \mid n[P] \mid P|P \mid !P \\ M & ::= & \texttt{in}\, n \mid \texttt{out}\, n. \end{array}$$

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

— $\text{pMA}_{g!}^{-\texttt{open}}$ combines the restrictions imposed by the previous fragments:

$$\begin{array}{rcl} P & ::= & \mathbf{0} \mid M.P \mid n[P] \mid P|P \mid !M.P \\ M & ::= & \texttt{in}\, n \mid \texttt{out}\, n. \end{array}$$

The structural congruence for $\text{pMA}_{g!}$ and $\text{pMA}_{g!}^{-\texttt{open}}$ is obtained by replacing the axiom for replication by $!M.P \equiv M.P \mid !M.P$ and the congruence rule for the replication operator $!P$ by the congruence rule for the operator of restricted replication $!M.P$.

3. Undecidability results

In this section we discuss the undecidability of reachability for the fragments $\text{pMA}^{-\texttt{open}}$ and $\text{pMA}_{g!}$.

As far as $\text{pMA}^{-\texttt{open}}$ is concerned, we resort to an equivalent result proved by Boneva and Talbot for a slightly different calculus (see Boneva and Talbot (2003, Theorem 3)). The only difference between that calculus, which was originally proposed in Cardelli and Gordon (2000b; 2005), and $\text{pMA}^{-\texttt{open}}$ is that it has three extra rules in the definition of the structural congruence relation:

$$\begin{array}{rcl} !\mathbf{0} & \equiv & \mathbf{0} \\ !!P & \equiv & !P \\ !(P \mid Q) & \equiv & !P \mid !Q. \end{array}$$

The undecidability of reachability was proved by Boneva and Talbot by showing how to encode two-counter machines (Minsky 1961), which are a well-known Turing powerful formalism. Even though the calculus in Boneva and Talbot (2003) is slightly different from $\text{pMA}^{-\texttt{open}}$, the encoding of two-counter machines presented in that paper can also be used in our calculus. This is because the encoding does not exploit the possibility of applying the replication operator to the empty process, to replicated processes or to parallel composition of processes (that is, the cases in which the three extra structural congruence rules come into play).

As far as $\text{pMA}_{g!}$ is concerned, we will present a modelling of Random Access Machines (RAMs) (Shepherdson and Sturgis 1963), which are similar to two-counter machines. The encoding we present here is an enhancement of the RAM modelling we used in Busi and Zavattaro (2004) for proving the undecidability of termination in the fragment of Mobile Ambients without restriction.

One of the main innovations of the new encoding is that it does not apply replication to ambients: this modification is necessary because replication is guarded in $\text{pMA}_{g!}$. A more significant difference concerns the production of garbage, that is, processes that do not play any further role in the computation. The encoding in Busi and Zavattaro (2004) produces garbage whose shape depends on the number and type of executed instructions. The production of garbage was not problematic in Busi and Zavattaro (2004) because the RAM modelling was used there to prove the undecidability of process termination, that is, the reachability of any deadlocked process independently of the garbage it

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

contains. Here, we need a more sophisticated RAM modelling that keeps control over the garbage produced so that we can prove the undecidability of reachability for a specific process.

In the new encoding, at the end of the RAM computation, an activity is started that formats the garbage into a predefined form. Thus, we can conclude that a RAM terminates if and only if the encoding of the final state of the RAM plus the formatted garbage is reachable from the encoding of the initial state of the RAM. This is enough for us to conclude that reachability is undecidable.

The rest of this section begins with a brief description of RAMs; we then discuss the modelling of RAMs.

3.1. *Random Access Machines*

RAMs are a computational model based on finite programs acting on a finite set of registers. More precisely, a RAM R is composed of the registers $r_1, \ldots, r_n$, which can hold arbitrarily large natural numbers, and by a sequence of indexed instructions $(1 : I_1), \ldots, (m : I_m)$. Minsky (1967) showed that the following two instructions are sufficient for modelling every recursive function:

— $(i : Succ(r_j))$: add 1 to the contents of register r_j and go to the next instruction;
— $(i : DecJump(r_j, s))$: if the contents of the register r_j is not zero, decrease it by 1 and go to the next instruction, otherwise jump to the instruction s.

The computation starts from the first instruction and continues by executing the other instructions in sequence, unless a jump instruction is encountered. The execution stops when an instruction number higher than the length of the program is reached. For our purposes, we can assume without loss of generality that the instruction number reached at the end of the computation is always $m + 1$, and that all the registers are empty when the computation starts and when it terminates.

3.2. *Modelling RAMs in $pMA_{g!}$*

We model instructions and registers independently. We model the program counter i using an ambient $pc_i[]$, and each instruction I_i is represented by a replicated process guarded by the capability $\texttt{open}\, pc_i$, which is able to open the corresponding program counter ambient $pc_i[]$. The processes modelling the instructions are replicated because each instruction could be performed an unbounded number of times during the computation.

The key idea underlying the modelling of the registers is to represent natural numbers with a corresponding nesting of ambients. We use an ambient named z_j to represent the register r_j when it is empty. When the register is incremented, we move the ambient z_j inside an ambient named s_j, and on register decrement, we dissolve the outer s_j ambient boundary. In this way, for instance, the register r_j with content 2 is modelled by the nesting $s_j[s_j[z_j[]]]$ (plus some other processes hosted by these ambients, which will be discussed in the following).

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

Definition 3.1 (RAM encoding). Given the RAM R with the instructions $(1 : I_1), \ldots, (m : I_m)$ and registers $r_1, \ldots, r_n$, we define $[\![R]\!]$ as the process

$$pc_1[] \mid \prod_{i \in 1 \ldots m} !\mathtt{open}\, pc_i.C_i \mid \prod_{j \in 1 \ldots n} R_j^0 \mid \\ \mathtt{open}\, pc_{m+1}.GC \mid !\mathtt{open}\, msg \mid garbage[\mathtt{open}\, gc]$$

where C_i (modelling the ith instruction), R_j^0 (modelling register r_j holding the value zero) and GC (the garbage collector, which is started at the end of the computation) are shorthand notations defined below.

Note the use of two extra processes: $!\mathtt{open}\, msg$, which is used to open ambients containing messages produced during the computation, and the ambient $garbage[\mathtt{open}\, gc]$, which is a container for the garbage produced. The process $\mathtt{open}\, gc$ is used at the end of the computation to allow the garbage collector to act inside the ambient $garbage$, as described below.

The register r_j, which initially has content 0, is represented by the process R_j^0 defined by

$$\begin{aligned} R_j^0 \;=\; z_j[\;\; & !\mathtt{open}\, inc_j. \\ & (\; msg[\;\mathtt{out}\, z_j.s_j[\; REG_j \;]\;] \mid \\ & \;\; \mathtt{in}\, s_j.acki_j[\;\mathtt{out}\, z_j.!\mathtt{out}\, s_j]\;) \mid \\ & !\mathtt{open}\, zero_j.ackz_j[\;\mathtt{out}\, z_j.\mathtt{in}\, dj_j\;] \mid \\ & \mathtt{open}\, gc\;\;] \end{aligned}$$

where REG_j is a shorthand notation defined by

$$REG_j \;=\; \mathtt{open}\, dec_j.ackd_j[\;\mathtt{out}\, s_j.\mathtt{in}\;\; dj_j\;] \mid !\mathtt{open}\, msg.$$

The process $\mathtt{open}\, gc$ is used again, in this case to allow the garbage collector to act inside the ambient z_j. We will discuss the behaviour of the term REG_j, and of the other processes inside the ambient z_j, after we have discussed the encoding of the instructions.

Before formalising the modelling of the instructions, we may mention that the names inc_j, $zero_j$ and dec_j will be used to model requests for increment, test for zero and decrement of register r_j, respectively, and the names $acki_j$, $ackz_j$ and $ackd_j$ will model the corresponding acknowledgements produced by the registers to notify the fact that a request has been managed.

The instructions are modelled as follows. If the ith instruction is $Succ(r_j)$, its encoding is

$$\begin{aligned} C_i \;=\; & increq_j[\;!\mathtt{in}\, s_j \mid \mathtt{in}\, z_j.inc_j[\mathtt{out}\, increq_j]\;] \mid \\ & \mathtt{open}\, acki_j.pc_{i+1}[]. \end{aligned}$$

In this case, C_i consists of two processes. The first is the ambient $increq_j$, which represents a request to increment the register r_j. The second process blocks, waiting for an acknowledgement that will be produced after the increment of the register has actually occurred. When the acknowledgement is received, the process increments the program counter by spawning $pc_{i+1}[]$.

The ambient $increq_j$ has the ability to enter the boundary of the ambient modelling the register r_j, to move through the nesting of ambients, and, finally, to enter the inner ambient z_j. After that, a new ambient inc_j exits the ambient $increq_j$ and is executed in

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

parallel with the processes of the ambient z_j. One of these processes (see the definition of R_j^0) detects the arrival of the new ambient and reacts by producing $s_j[REG_j]$. The ambient z_j then moves inside this new ambient. In this way, the nesting of ambients s_j is incremented by one. Afterwords, the acknowledgement is produced through an ambient named $acki_j$, which moves outside the register boundary.

If the ith instruction is $DecJump(r_j, s)$, the encoding is

$$C_i = zero_j[\texttt{in}\, z_j] \mid dec_j[\texttt{in}\, s_j] \mid dj_j[\, ACKZ_{js} \mid ACKD_{ji}\,]$$

where

$$ACKZ_{js} = \texttt{open}\, ackz_j.\texttt{in}\, garbage.\ msg[\ \texttt{out}\, dj_j.\texttt{out}\, garbage.\texttt{open}\, dec_j.pc_s[]\ ]$$
$$ACKD_{ji} = \texttt{open}\, ackd_j.\texttt{in}\, garbage.\ msg[\ \texttt{out}\, dj_j.\texttt{out}\, garbage.\texttt{open}\, zero_j.\texttt{open}\, s_j.pc_{i+1}[]\ ].$$

In this case, C_i consists of three processes: the first is an ambient named $zero_j$, which represents a request for a test for zero of the register r_j; the second is an ambient named dec_j, which represents a request for decrement of the register r_j; the third is an ambient named dj_j, which is in charge of managing the acknowledgement produced by the register r_j. The acknowledgement indicates whether the decrement or the test for zero request has succeeded.

We first consider the test for zero request. The ambient $zero_j[\texttt{in}\, z_j]$ has the ability to move inside the ambient z_j. This can occur only if the register r_j is currently empty. In fact, if r_j is not empty, the ambient z_j is not at the outer level. If the request enters the z_j ambient boundary, the processes inside the ambient z_j (see the definition of R_j^0) react by producing an acknowledgement modelled by an ambient named $ackz_j$, which moves inside the ambient dj_j.

Now consider the request for decrement. The ambient $dec_j[\texttt{in}\, s_j]$ has the ability to enter the boundary of the process modelling the register r_j. This can occur only if the register is not empty (otherwise there is no ambient s_j). Inside the ambient s_j, the process REG_j reacts by producing an acknowledgement modelled by an ambient named $ackd_j$, which moves inside the ambient dj_j.

The processes inside the ambient dj_j have the ability to detect which kind of acknowledgement has been produced, and react accordingly:

— If it is $ackz_j$, the reaction is to move the ambient dj_j inside the ambient $garbage$ and to dissolve the boundary of the outer ambient dec_j. This is required in order to remove the decrement request that has failed.

— If it is $ackd_j$, the process also dissolves one of the boundaries s_j so that it actually decrements the register.

In both cases, the program counter is finally updated: in the first case by jumping to instruction s and in the second case by activating the next instruction $i + 1$.

This way of modelling RAMs produces additional processes during the simulation of each instruction. We refer to these additional processes as *garbage processes*. More

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

precisely, the following garbage processes are produced:

— Each increment operation leaves an ambient $increq_j[!\mathtt{in}\, s_j]$ inside the ambient z_j, plus the process $!\mathtt{out}\, s_j$ at the outer level.
— Each decrement operation leaves an ambient dj_j inside the ambient *garbage*, plus the two processes $\mathtt{in}\, z_j$ and $!\mathtt{open}\, msg$ at the outer level.
— Each test for zero operation leaves an ambient dj_j inside the ambient *garbage*, plus the process $\mathtt{in}\, s_j$ at the outer level.

Summarising, the garbage processes can be classified into three categories: those left at the outer level; those left inside the ambient *garbage*; and those left inside the z_j ambients. We define these three sets of possible garbage processes formally as follows:

$$\begin{aligned} Garb_{outer} &= \{(\textstyle\prod_{j\in 1\ldots n}(\prod_{k_1} !\mathtt{out}\, s_j | \prod_{k_2} \mathtt{in}\, z_j | \prod_{k_3} \mathtt{in}\, s_j)) \mid \prod_{k_4} !\mathtt{open}\, msg \\ &\qquad \mid k_1, k_2, k_3, k_4 \in I\!N\} \\ Garb_{garbage} &= \{\textstyle\prod_{j\in 1\ldots n} \prod_{i\in 1\ldots m+1}(\prod_{k_1} dj_j[ACKZ_{ji}] | \prod_{k_2} dj_j[ACKD_{ji}]) \\ &\qquad \mid k_1, k_2 \in I\!N\} \\ Garb_{z_j} &= \{\textstyle\prod_k increq_j[!\mathtt{in}\, s_j] \mid k \in I\!N\} \end{aligned}$$

Because of the presence of the garbage processes, a given configuration of a RAM can be represented by more than one term of $\mathrm{pMA}_{g!}$, which differ from each other in the number of garbage processes they contain. Formally, given a RAM R with the instructions $(1 : I_1), \ldots, (m : I_m)$ and registers $r_1, \ldots, r_n$, we use $(i, c_1, \ldots, c_n)$ to denote the state of the computation of R in which the next instruction to be executed is $(i : I_i)$, and the contents of the registers $r_1, \ldots, r_n$ are $c_1, \ldots, c_n$, respectively. This state can be represented by a set of processes, which we denote using $[\![i, c_1, \ldots, c_n]\!]_R$, that differ only in the number of garbage processes.

Definition 3.2 (encoding of RAM states). Given a RAM R with the instructions $(1 : I_1), \ldots, (m : I_m)$ and registers $r_1, \ldots, r_n$, we use $[\![i, c_1, \ldots, c_n]\!]_R$ to denote the minimal set of processes closed under structural congruence and including at least the following set of processes of $\mathrm{pMA}_{g!}$:

$$\begin{aligned} \{\ & G_1 \mid pc_i[] \mid \textstyle\prod_{i\in 1\ldots m} !\mathtt{open}\, pc_i.C_i \mid \prod_{j\in 1\ldots n} Reg_j^{c_j} \mid \\ & \mathtt{open}\, pc_{m+1}.GC \mid !\mathtt{open}\, msg \mid garbage[\mathtt{open}\, gc \mid G_2] \\ & \qquad \mid G_1 \in Garb_{outer}, G_2 \in Garb_{garbage}, Reg_j^{c_j} \in Register_j^{c_j} \ \} \end{aligned}$$

where the processes C_i and GC are as previously discussed, and the set of processes $Register_j^{c_j}$ is defined inductively on c_j by

$$\begin{aligned} Register_j^0 &= \{\ z_j[\ G_3 \mid !\mathtt{open}\, inc_j. \\ &\qquad (\ msg[\ \mathtt{out}\, z_j.s_j[\ REG_j\ ]\ ] \mid \\ &\qquad\ \ \mathtt{in}\, s_j.acki_j[\ \mathtt{out}\, z_j.!\mathtt{out}\, s_j]\ ) \mid \\ &\qquad !\mathtt{open}\, zero_j.ackz_j[\ \mathtt{out}\, z_j.\mathtt{in}\, dj_j\ ] \mid \\ &\qquad \mathtt{open}\, gc\ \ ] \\ &\qquad\qquad \mid G_3 \in Garb_{z_j}\ \} \\ Register_j^{l+1} &= \{\ s_j[\ REG_j \mid Reg_j^l\ ] \mid Reg_j^l \in Register_j^l\ \} \end{aligned}$$

where REG_j is as previously defined.

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

Now consider a RAM R that terminates its computation in the state $(m+1, 0, \ldots, 0)$. The precise shape of the garbage in the corresponding process reached by the encoding $[\![R]\!]$ is unpredictable because it depends on the exact number of instructions executed. However, we can use the garbage collector process GC, which is activated by the process $pc_{m+1}[]$ on program termination, to reshape the garbage into a predefined format.

The key idea underlying the garbage collection process is to exploit the structural congruence rule $!P \equiv P|!P$, which is used to unfold (and fold) replication. Consider an unpredictable number of processes P in parallel, that is, $\prod_n P$ with n unknown. If we add the process $!P$ in parallel, we have $\prod_n P \mid !P \equiv !P$, thus reshaping the process into a known format.

We can now define the garbage collector process formally:

$$\begin{array}{rl} GC = & !!\mathtt{out}\, s_j \mid !\mathtt{in}\, z_j \mid !!\mathtt{open}\, msg \mid !\mathtt{in}\, s_j \mid \\ & \prod_{j\in 1\ldots n}(gc[\ \mathtt{in}\, z_j.(!\mathtt{open}\, increq_j \mid !!\mathtt{in}\, s_j)\ ]) \mid \\ & gc[\ \mathtt{in}\, garbage \mid \\ & \quad \prod_{j\in 1\ldots n}(\ !\mathtt{open}\, dj_j \mid \prod_{i\in 1\ldots m} !ACKD_{ji} \mid \\ & \quad\quad \prod_{s\in 1\ldots m+1} !ACKZ_{js}\ )\ ]. \end{array}$$

The process GC has the ability to reshape any process in the set $[\![m+1, 0, \ldots, 0]\!]_R$ into the following *final state* $[\![R]\!]_{final}$.

Definition 3.3 (final state). Given a RAM R with the instructions $(1 : I_1), \ldots, (m : I_m)$ and registers $r_1, \ldots, r_n$, we use $[\![R]\!]_{final}$ to denote the following $\mathrm{pMA}_{g!}$ process:

$$\begin{array}{l} \prod_{i\in 1\ldots m} !\mathtt{open}\, pc_i.C_i \mid \\ \prod_{j\in 1\ldots n}(\ z_j[\ !\mathtt{open}\, inc_j. \\ \qquad\qquad (\ msg[\ \mathtt{out}\, z_j.s_j[\ REG_j\ ]\ ] \mid \\ \qquad\qquad\ \ \mathtt{in}\, s_j.acki_j[\ \mathtt{out}\, z_j.!\mathtt{out}\, s_j]\ ) \mid \\ \qquad\qquad !\mathtt{open}\, zero_j.ackz_j[\ \mathtt{out}\, z_j.\mathtt{in}\, dj_j\ ] \mid \\ \qquad\qquad !\mathtt{open}\, increq_j \mid !!\mathtt{in}\, s_j]\ ) \mid \\ !!\mathtt{out}\, s_j \mid !\mathtt{in}\, z_j \mid !!\mathtt{open}\, msg \mid !\mathtt{in}\, s_j \mid \\ garbage[\ \prod_{j\in 1\ldots n}(\ !\mathtt{open}\, dj_j \mid \prod_{i\in 1\ldots m} !ACKD_{ji} \mid \\ \qquad\qquad \prod_{s\in 1\ldots m+1} !ACKZ_{js}\ )\ ]. \end{array}$$

In the following proposition, which formalises the correctness of our encoding of RAMs into $\mathrm{pMA}_{g!}$, we use the following notation and terminology:

— $(i, c_1, \ldots, c_n) \rightarrow_R (i', c'_1, \ldots, c'_n)$ means that the state of the RAM R changes from $(i, c_1, \ldots, c_n)$ to $(i', c'_1, \ldots, c'_n)$ due to the execution of the ith instruction.

— A *deterministic computation* is a sequence of reduction steps $P_0 \rightarrow P_1 \rightarrow \ldots \rightarrow P_k$ such that, for each $0 \leqslant i < k$, if $P_i \rightarrow Q$, then $Q \equiv P_{i+1}$.

Proposition 3.4. Consider a RAM R with the instructions $(1 : I_1), \ldots, (m : I_m)$ and registers $r_1, \ldots, r_n$. Given the state $(i, c_1, \ldots, c_n)$ of R and a process $P \in [\![i, c_1, \ldots, c_n]\!]_R$, we have:

— If $i \leqslant m$, then $(i, c_1, \ldots, c_n) \rightarrow_R (i', c'_1, \ldots, c'_n)$ and there exists $k > 0$ such that $P \rightarrow P_1 \rightarrow \ldots \rightarrow P_k$ is a deterministic computation with $P_k \in [\![i', c'_1, \ldots, c'_n]\!]_R$ and, for every $1 \leqslant j < k$, we have $P_j \notin [\![i'', c''_1, \ldots, c''_n]\!]_R$, for every $i'', c''_1, \ldots, c''_n$.

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

— If $i = m + 1$, then there exists $k > 0$ such that $P \to P_1 \to \ldots \to P_k$ with $P_k = [\![R]\!]_{final}$.

Proof. Assuming that $i \leqslant m$, it is easy to see that the first item of the proposition holds by case analysis on the three possible instructions: increment, decrement or jump. In each of the three cases, we can show that all of the computations of P proceed deterministically, traversing processes that are not in the set $[\![i'', c_1'', \ldots, c_n'']\!]_R$ for every $i'', c_1'', \ldots, c_n''$, and leading to a process P_k such that $P_k \in [\![i', c_1', \ldots, c_n']\!]_R$.

If $i = m+1$, as we have restricted our interest to RAMs that terminate with all registers empty, we have $(i, c_1, \ldots, c_n) = (m + 1, 0, \ldots, 0)$. Given any $P \in [\![m + 1, 0, \ldots, 0]\!]_R$, we have already observed (see the paragraphs before Definition 3.3) that the garbage collector process GC, which can be activated in P due to the presence of $pc_{m+1}[]$, performs a computation whose effect is to reshape the process to the final state $[\![R]\!]_{final}$. □

We are now finally ready to prove that our encoding of RAMs reduces the termination problem for RAMs to the reachability problem for $\text{pMA}_{g!}$ processes.

Theorem 3.5. Given the RAM R with instructions $(1 : I_1), \ldots, (m : I_m)$ and registers $r_1, \ldots, r_n$, we have that R terminates if and only if $[\![R]\!]_{final}$ is reachable from $[\![R]\!]$.

Proof. Assume that the RAM R terminates. This means that the computation of R, starting from $(1, 0, \ldots, 0)$, leads to the state $(m + 1, 0, \ldots, 0)$. We now consider the process $[\![R]\!]$. It is easy to see that $[\![R]\!] \in [\![1, 0, \ldots, 0]\!]_R$. Applying the first item of Proposition 3.4, we can prove that $[\![R]\!]$ has a computation leading to a process $P \in [\![m + 1, 0, \ldots, 0]\!]_R$. Applying the second item of Proposition 3.4, we can conclude that P has a computation leading to $[\![R]\!]_{final}$.

Assume now that $[\![R]\!]$ has a computation $[\![R]\!] \to P_1 \to \ldots \to P_k = [\![R]\!]_{final}$. As $[\![R]\!] \in [\![1, 0, \ldots, 0]\!]_R$, we can apply the first item of Proposition 3.4 to conclude that some of the processes $P_1, \ldots, P_k$ belong to sets $[\![i, c_1, \ldots, c_n]\!]_R$, for some $i, c_1, \ldots, c_n$. Let P_l be the last of these processes, that is, the process of this kind with the greatest index l. Applying Proposition 3.4, we can observe that $P_l \in [\![m + 1, c_1, \ldots, c_n]\!]_R$. In fact, if, in order to show a contradiction, we assume that $P_l \in [\![i, c_1, \ldots, c_n]\!]_R$ with $i \leqslant m$, we can apply the first item of Proposition 3.4 to conclude that a computation starting from P_l cannot terminate until a new process $Q \in [\![i', c_1', \ldots, c_n']\!]_R$ is reached. But P_l has the terminating computation $P_l \to P_{l+1} \to \ldots \to [\![R]\!]_{final}$, which never traverses a process like Q. Thus $P_l \in [\![m + 1, c_1, \ldots, c_n]\!]_R$. We can conclude that $[\![R]\!] \to P_1 \to \ldots \to P_l \in [\![m + 1, c_1, \ldots, c_n]\!]_R$, hence, by applying the first item of Proposition 3.4, we also have that the RAM R has a terminating computation. □

The undecidability of the reachability problem for $\text{pMA}_{g!}$ is a trivial corollary of the above theorem, which reduces the termination problem for RAMs to the reachability problem for $\text{pMA}_{g!}$ processes.

Corollary 3.6. The reachability problem is undecidable in $\text{pMA}_{g!}$.

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

4. Deciding reachability in $\mathbf{pMA}_{g!}^{-\mathtt{open}}$

In this Section we show that reachability is decidable in $\mathrm{pMA}_{g!}^{-\mathtt{open}}$. We reduce reachability in $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ to reachability in Place/Transition Petri nets (P/T nets). As reachability is decidable on this class of Petri nets (Reutenauer 1988), we obtain the decidability result for reachability on $\mathrm{pMA}_{g!}^{-\mathtt{open}}$. We start by recalling some basic definitions for Petri nets.

4.1. *P/T nets*

We begin by recalling the notion of Place/Transition nets with unweighted flow arcs (see, for example, Reisig (1985)). Here we provide a characterisation of this model that is convenient for our purposes.

Definition 4.1. Given a set S, a *finite multiset* over S is a function $m : S \to \mathbb{N}$ such that the set $dom(m) = \{s \in S \mid m(s) \neq 0\}$ is finite. The *multiplicity* of an element s in m is given by the natural number $m(s)$. The set of all finite multisets over S, denoted by $\mathscr{M}_{fin}(S)$, is ranged over by m. A multiset m such that $dom(m) = \emptyset$ is said to be *empty*. The set of all finite sets over S is denoted by $\wp_{fin}(S)$.

Given the multisets m and m', we write $m \subseteq m'$ if $m(s) \leqslant m'(s)$ for all $s \in S$, and we use $\oplus$ to denote their *multiset union*:

$$m \oplus m'(s) = m(s) + m'(s).$$

We use the operator $\setminus$ to denote *multiset difference*:

$$(m \setminus m')(s) \;=\; \begin{cases} m(s) - m'(s) & \text{if } m(s) \geqslant m'(s) \\ 0 & \text{otherwise.} \end{cases}$$

The *scalar product*, $j \cdot m$, of a number j with a multiset m is $(j \cdot m)(s) = j \cdot (m(s))$.

To simplify the notation, we sometimes use the following abbreviations. If m is a multiset containing only one occurrence of an element s (that is, $dom(m) = \{s\}$ and $m(s) = 1$), we just use s to denote m. Multiset union is also represented by comma, that is, $m, m' = m \oplus m'$. Let m be a multiset over S and m' be a multiset over $S' \supseteq S$, such that $(m'(s') = 0)$ for each $s' \in S' \setminus S$; with abuse of notation, we sometimes use m in place of m', and *vice versa.*

Definition 4.2. A P/T net is a pair (S, T) where S is the set of *places* and $T \subseteq \mathscr{M}_{fin}(S) \times \mathscr{M}_{fin}(S)$ is the set of *transitions.*

Finite multisets over the set S of places are called *markings.* Given a marking m and a place s, we say that the place s contains $m(s)$ *tokens.*

A P/T net is finite if both S and T are finite.

A P/T system is a triple $N = (S, T, m_0)$ where (S, T) is a P/T net and m_0 is the *initial marking.*

A transition $t = (c, p)$ is usually written in the form $c \to p$. The marking c, usually denoted by ${}^\bullet t$, is called the *preset* of t and represents the tokens to be *consumed* by t; the marking p, usually denoted by $t^\bullet$, is called the *postset* of t and represents the tokens to be *produced* by t.

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

A transition t is *enabled* at a marking m if ${}^\bullet t \subseteq m$. The execution of a transition t enabled at m produces the marking $m' = (m \setminus {}^\bullet t) \oplus t^\bullet$. This is written as $m \xrightarrow{t} m'$, or simply $m \to m'$ when the transition t is not relevant. We use σ, τ to range over sequences of transitions. The empty sequence is denoted by ε. If $\sigma = t_1, \ldots, t_n$, we write $m \xrightarrow{\sigma} m'$ to mean the *firing sequence* $m \xrightarrow{t_1} \cdots \xrightarrow{t_n} m'$.

We say that m' is *reachable from* m if there exists some σ such that $m \xrightarrow{\sigma} m'$.

Definition 4.3. Let $N = (S, T, m_0)$ be a P/T system. The reachability problem for marking m consists of checking if $m_0 \to^* m$.

The reachability problem is known to be decidable for P/T systems (Mayr 1981).

4.2. *Reducing reachability on processes to reachability on Petri nets*

Given two processes P and R, we show how to construct a (finite) Petri system $Sys_{P,R}$ satisfying the following property: the check of $P \to^* R$ is equivalent to checking the reachability of a finite set of markings on $Sys_{P,R}$.

The intuition behind this result relies on a monotocity property of $\mathrm{pMA}_{g!}^{-\mathtt{open}}$: because it lacks the `open` capability, the number of 'active' ambients in a process (that is, ambients that are not guarded by a capability) cannot decrease during the computation. Moreover, as the applicability of replication is restricted to guarded processes, the number of 'active' ambients in a set of structurally equivalent processes is finite (this is not the case in, for example, the pMA process $!n[0]$). Thanks to the properties explained above, it is sufficient to take into account a subset of the derivatives of P: namely, those derivatives whose number of active ambients is no greater than the number of active ambients in R.

Unfortunately, this subset of derivatives is, in general, not finite, since the processes inside an ambient can grow without limited. Consider, for example, the process

$$P = m[!\mathtt{in}\, n.\mathtt{out}\, n.Q] \mid n[].$$

It is easy to see that, for each l, we have that

$$m[\prod_{l} Q \mid !\mathtt{in}\, n.\mathtt{out}\, n.Q] \mid n[]$$

is a derivative of P.

On the other hand, note that the set of 'sequential' subprocesses of (the derivatives of) a process P (namely, the subterms of kind $M.P$ or $!M.P$) is finite. In the light of this observation, we can borrow a traditional technique for mapping process algebras onto Petri nets. A process P is decomposed in the (finite) multiset of its sequential subprocesses that appear unguarded in P; this multiset is then considered to be the marking of a Place/Transition Petri net. The execution of a computational step in a process will correspond to the firing (execution) of a transition in the corresponding net. Thus, we reduce the reachability problem for $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ processes to reachability of a finite set of markings in a Place/Transition Petri net, which is a decidable problem.

We need to consider a finite set of possible reachable markings, rather than just one, because the technique we use to represent processes with P/T nets associates with each

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

process a finite set of corresponding markings. This follows from the fact that (unlike the case for classical process algebras where processes can be faithfully represented by a multiset of subprocesses) $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ processes have a tree-like structure, which is a poor fit to a flat model such as a multiset. The solution is to consider $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ processes as composed of two kinds of component: the tree-like structure of ambients and the family of multisets of sequential subterms contained in each ambient. As an example, consider the process

$$\mathtt{in}\, n.P \mid m[\mathtt{in}\, n.P \mid \mathtt{out}\, n.Q \mid n[\mathbf{0}] \mid k[\mathbf{0}] \mid \mathtt{in}\, n.P] \mid n[\mathtt{in}\, n.P].$$

Its tree-like structure is $m[n[] \mid k[]] \mid n[]$. Moreover, there is a multiset corresponding to each 'node' of the tree: the multiset $\{\mathtt{in}\, n.P\}$ is associated with the root; the same multiset is associated with the n-labelled son of the root; the multiset $\{\mathtt{in}\, n.P, \mathtt{in}\, n.P, \mathtt{out}\, n.Q\}$ is associated with the m-labelled son of the root; and so on.

The Petri net we construct is composed of the following parts. The first part is basically a finite state automaton, where the marked place represents the current tree-like structure of the process, and a set of identical subnets: the marking of each subnet represents the multiset associated with a particular node of the tree. In order to keep the correspondence between the nodes of the tree and the multiset associated with that node, we make use of labels. A distinct label is associated with each subnet; this label will be used in the tree-like structure to label the node whose contents (that is, the set of sequential subprocesses contained in the ambient corresponding to the node) is represented by the subnet.

The set of possible tree-like structures we need to consider is finite for the following reasons. First, the set of ambient names in a process is finite. Moreover, to verify reachability, we only need to take into account those processes whose number of active ambients is limited by the number of ambients in the process we want to reach.

The upper bound on the number of nodes in the tree-like structures also provides an upper bound to the number of identical subnets we need to decide reachability.

In general, the number of active ambients grows during the computation, so we need a mechanism for remembering which subnets are currently in use and which are not. When a new ambient is created, a correspondence between the node corresponding to such an ambient in the tree-like structure and a not yet used subnet is established, and the places of the 'fresh' subnet are filled with the marking corresponding to the sequential subprocesses contained in the newly created ambient. To this end, each subnet is equipped with a place called *unused*, which contains a token for as long as the subnet does not correspond to any node in the tree-like structure.

Because of the structural congruence rule (6), determining the reachability of a process R actually corresponds to deciding whether it is possible to reach a process that is structurally congruent to R. Since we are reducing the reachability in $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ to marking reachability in Petri nets, the set of markings, corresponding to the set of processes structurally congruent to R, must be finite. Let us concentrate on the markings of the subnets. The top-level applications of the monoidal laws for parallel composition are automatically dealt with since processes that are structurally congruent because of such laws are mapped to the same marking. Unfortunately, the application of the replication law allows the generation of an infinite set of markings corresponding to structurally

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

congruent processes. Take, for example,

$$!\mathtt{in}\, n.P \equiv \mathtt{in}\, n.P \mid !\mathtt{in}\, n.P \equiv \mathtt{in}\, n.P \mid \mathtt{in}\, n.P \mid !\mathtt{in}\, n.P \equiv \ldots$$

and the corresponding set of markings

$$\{!\mathtt{in}\, n.P\}, \{\mathtt{in}\, n.P, !\mathtt{in}\, n.P\}, \{\mathtt{in}\, n.P, \mathtt{in}\, n.P, !\mathtt{in}\, n.P\} \ldots.$$

The top-level application of the law for replication can be dealt with easily by adding, for example, the transitions $!\mathtt{in}\, n.P \rightarrow !\mathtt{in}\, n.P \mid \mathtt{in}\, n.P$ and $!\mathtt{in}\, n.P \mid \mathtt{in}\, n.P \rightarrow !\mathtt{in}\, n.P$, respectively, allowing us to spawn a new copy of a replicated process and to absorb a process that also appears in a replicated form in the marking.

The final problem to be dealt with is the application of the laws in combination with the congruence law for prefix and ambient. Consider, for example, the reachability of process $R = m[\mathtt{in}\, n.!\mathtt{in}\, m.\mathbf{0}]$. For the subnet corresponding to the m-labelled son of the root, we must check reachability of an infinite set of markings, namely, $\{\mathtt{in}\, n.!\mathtt{in}\, m.\mathbf{0}\}$, $\{\mathtt{in}\, n.(\mathtt{in}\, m.\mathbf{0} \mid !\mathtt{in}\, m.\mathbf{0})\}$, $\{\mathtt{in}\, n.(\mathtt{in}\, m.\mathbf{0} \mid \mathtt{in}\, m.\mathbf{0} \mid !\mathtt{in}\, m.\mathbf{0})\}$, ...

To this end, we introduce canonical representations of the equivalence classes of structural congruence, consisting roughly of nested multisets where the presence of a replicated version of a sequential term forbids the presence of any occurrence of the non-replicated version of the same term. For example, the normal form of process

$$\mathtt{in}\, n.(!\mathtt{out}\, m.\mathbf{0}) \mid !\mathtt{in}\, n.(\mathtt{out}\, m.\mathbf{0} \mid !\mathtt{out}\, m.\mathbf{0}) \mid n[\mathtt{in}\, n.\mathbf{0}]$$

is the nested multiset

$$!\mathtt{in}\, n.(!\mathtt{out}\, m.\mathbf{0}) \mid n[\mathtt{in}\, n.\mathbf{0}].$$

We will now start the technical part by providing a definition of ambient multisets – which are the canonical representations of the equivalence classes of the structural congruence relation – and of the function α that maps a process to its canonical representation. The function α behaves as a homomorphism for all process operations apart from parallel composition, where some care has to be taken to avoid the presence of both the replicated and the unreplicated versions of a guarded process.

Definition 4.4. An *index set* is a set $I \subseteq \mathbb{N}$ such that $I = \{1, 2, \ldots, k\}$ for some natural number k.

The set $\mathscr{A}$ of *ambient multisets* is the least set closed with respect to the equation

$$a = \bigoplus_{i \in I} M_i.a_i \oplus \bigoplus_{j \in J} !M'_j.a'_j \oplus \bigoplus_{k \in K} n_k[a''_k]$$

where I, J, K are index sets, $a_i, a'_j, a''_k \in \mathscr{A}$ and $M_i = M'_j$ implies $a_i \neq a'_j$ for all $i \in I$, $j \in J$ and $k \in K$.

The function $\alpha : \mathrm{pMA}_{g!}^{-\mathtt{open}} \rightarrow \mathscr{A}$ maps a process to the corresponding ambient multiset and is defined inductively by

$$\begin{aligned} \alpha(\mathbf{0}) &= \varnothing \\ \alpha(M.P) &= M.\alpha(P) \\ \alpha(!M.P) &= !M.\alpha(P) \\ \alpha(n[P]) &= n[\alpha(P)]. \end{aligned}$$

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

Let

$$\alpha(P_h) = \bigoplus_{i\in I_h} M_{ih}.a_{ih} \oplus \bigoplus_{j\in J_h} !M'_{jh}.a'_{jh} \oplus \bigoplus_{k\in K_h} n_{kh}[a''_{kh}]$$

for $h = 1, 2$. We define

$$\alpha(P_1 \mid P_2) = \bigoplus_{h=1,2} (\bigoplus_{i\in I_h} \mu_{ih} \oplus \bigoplus_{j\in J_h} !M'_{jh}.a'_{jh} \oplus \bigoplus_{k\in K_h} n_{kh}[a''_{kh}])$$

where

$$\mu_{i1} = \begin{cases} M_{i1}.a_{i1} & \text{if } \forall j \in J_2 : M_{i1} = M'_{j2} \Rightarrow a_{i1} \neq a'_{j2} \\ \varnothing & \text{otherwise,} \end{cases}$$

and define the μ_{i2} symmetrically.

The tree-like structure of the ambients of a process is represented by an ambient tree, which is basically a tree with edges labelled by ambient names and nodes decorated with labels. We also define the set of ambient trees whose number of ambients is bounded by an upper limit.

Definition 4.5. Let $\mathscr{L}$ be a denumerable set of labels (that is, $\mathscr{L} = l_0, l_1, l_2, \ldots$), with $\mathscr{L}$ ranged over by $l, l', l'_1, \ldots$. Sequences of labels, that is, elements of $\mathscr{L}^*$, are ranged over by $\lambda, \lambda', \ldots$.

The set $\mathscr{T}$ of *ambient trees* is the least set closed with respect to the equation

$$t = l \cdot \bigoplus_{i\in I} n_i[t_i]$$

where I is an index set and $t_i \in \mathscr{T}$ for all $i \in I$.

The number of ambients in an ambient tree $t = l \cdot \bigoplus_{i\in I} n_i[t_i]$ is defined by

$$\#amb(t) = |I| + \sum_{i\in I} \#amb(t_i).$$

The set of labels and the set of ambient names in an ambient tree $t = l \cdot \bigoplus_{i\in I} n_i[t_i]$ is defined by

$$labels(t) = \{l\} \cup \bigcup_{i\in I} labels(t_i)$$
$$names(t) = \{n_i \mid i \in I\} \cup \bigcup_{i\in I} names(t_i).$$

The set of ambient trees of size at most h defined on the set of ambient names $\mathscr{N}$ is

$$\mathscr{T}_h^{\mathscr{N}} = \{t \in \mathscr{T} \mid \#amb(t) \leqslant h \ \wedge \ labels(t) \in \{l_0, \ldots, l_h\} \ \wedge \ names(t) \subseteq \mathscr{N}\}.$$

In the following, we will consider ambient trees containing distinct labels.

Now we are almost ready to construct the net that will enable us to decide the reachability of a process R starting from a process P. Recall that the markings of the net can represent the processes whose number of active ambients is no greater than the number of active ambients in R. We consider a set of labels equal to the number of active ambients of R plus one.

The part of the net representing the tree-like structure contains a place for each ambient tree of size no greater than the number of active ambients in R. Each of the subnets contains a place for each sequential and replicated subprocess of process P, and a place

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

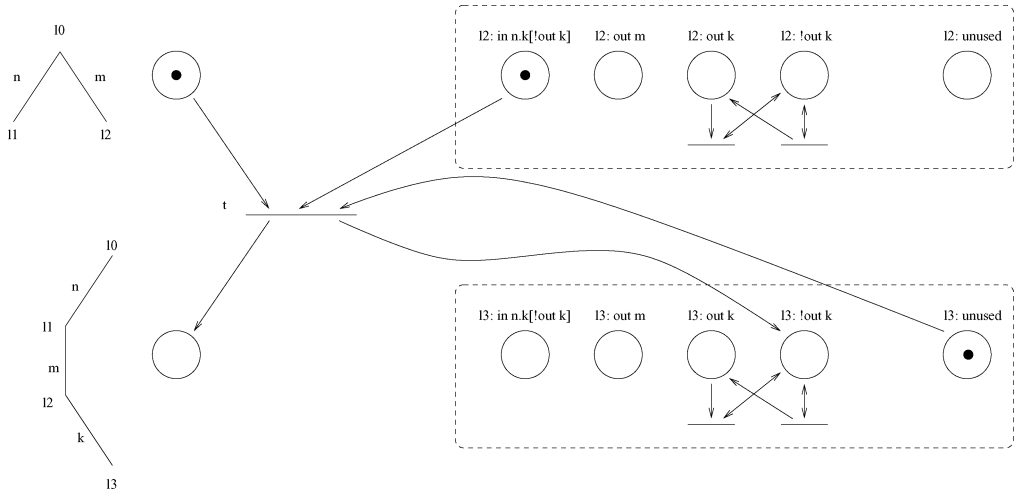

Fig. 2. A portion of the net corresponding to process $n[\texttt{out}\, m] \mid m[\texttt{in}\, n.k[\,!\texttt{out}\, k]]$.

named 'unused', which remains filled until the subnet does not correspond to any node in the tree-like structure. Moreover, we associate a distinct label with each subnet, and all the places of the subnet will be decorated with such a label.

The net has two sets of transitions: the first enables us to model the execution of the `in` and `out` capabilities, and the second is used to handle the structural congruence rule for replication.

Let us concentrate on the first set of transitions. A capability $\texttt{in}\, n$ can be executed when the following conditions are fulfilled: the tree-like structure must have a specific structure, and a place corresponding to a sequential subprocess $\texttt{in}\, n.Q$ is marked in a subnet whose label appears in the correct position in the tree-like structure. Moreover, the number of active ambients created by the execution of the capability, added to the number of currently active ambients, must not exceed the number of active ambients in the process R we want to reach. This condition is checked by requiring that there are a sufficient number of 'unused' places that are currently marked. The execution of the capability causes the following changes to the marking of the net: the place corresponding to the new tree-like structure is now filled and the marking of the subnet performing the $\texttt{in}\, n$ operation is updated (by adding the tokens in the places corresponding to the active sequential and replicated subprocesses in the continuation Q). Moreover, a number of subnets equal to the number of active ambients in the continuation Q become active: their places will be filled with the tokens corresponding to the active sequential and replicated subprocesses contained in the corresponding ambient, and the tree-like structure is updated accordingly.

For example, consider the process $n[\texttt{out}\, m] \mid m[\texttt{in}\, n.k[\,!\texttt{out}\, k]]$. The relevant part of the corresponding net is shown in Figure 2: a subset of the places, representing the tree-like structure, is shown in the left-hand part of the figure, while the subnets are shown in the right-hand part. We only consider the subnets labelled l_2 and l_3, and omit the two subnets labelled l_0 (with empty marking) and l_1 (whose marking consists of a token in

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

place l_1 : out m). The computation step

$$n[\texttt{out}\, m] \mid m[\texttt{in}\, n.k[!\texttt{out}\, k]] \rightarrow n[\texttt{out}\, m \mid m[k[!\texttt{out}\, k]]]$$

corresponds to the firing of transition t in the net.

The transitions for the execution of a capability out n are dealt with similarly by checking a different tree-like structure.

To define the transitions formally, we need some auxiliary notation.

Ambient tree contexts will be used to model the requirement that the tree-like structure has a specific form, and to update this structure. An ambient tree context is essentially an ambient tree with a hole, which can be fulfilled using a set of trees, each labelled with an ambient name. The set of ambient tree contexts is generated by the grammar

$$\begin{aligned} C[] \;=\; & l \cdot [] \oplus \textstyle\bigoplus_{i\in I} n_i[t_i] \quad \mid \\ & l \cdot n[l' \cdot C[] \oplus \textstyle\bigoplus_{j\in J} n'_j[t'_j]] \oplus \textstyle\bigoplus_{i\in I} n_i[t_i]. \end{aligned}$$

We now introduce some notions related to the features of ambient multisets.

Definition 4.6. Let

$$a = \bigoplus_{i\in I} M_i.a_i \oplus \bigoplus_{j\in J} !M'_j.a'_j \oplus \bigoplus_{k\in K} n_k[a''_k]$$

be an ambient multiset.

The set of sequential and replicated subprocesses of a is defined by

$$\begin{aligned} sub(a) = \; & \{M_i.a_i \mid i \in I\} \cup \\ & \{M'_j.a'_j, !M'_j.a'_j \mid j \in J\} \cup \\ & \textstyle\bigcup_{i\in I} sub(a_i) \cup \bigcup_{j\in j} sub(a'_j) \cup \bigcup_{k\in K} sub(a''_k). \end{aligned}$$

The number of active ambients in a is defined by

$$\#amb(a) = |K| + \sum_{k\in K} \#amb(a''_k).$$

The number of active ambients in a process P is defined by

$$\#amb(P) = \#amb(\alpha(P)).$$

To define the set of transitions, we need some preliminary definitions, which will enable us to construct the new part of the ambient tree (generated by the active ambients in the continuation) and the marking of the newly activated subnets.

Definition 4.7. Let

$$a = \bigoplus_{i\in I} M_i.a_i \oplus \bigoplus_{j\in J} !M'_j.a'_j \oplus \bigoplus_{k\in K} n_k[a''_k]$$

be an ambient multiset[†].

[†] To be precise, at this point we have to fix an order on the elements of the multiset a, that is, instead of a, we must consider the sequence $\overline{a} = M_1.a_1 \ldots M_{|I|}.a_{|I|} !M'_1.a'_1 \ldots !M_{|J|}.a'_{|J|} n_1[\overline{a''_1}] \ldots n_{|K|}[\overline{a''_{|K|}}]$. We need to fix the ordering of the elements to obtain the correct correspondence between the labels in the ambient tree and the labels of the active nets.

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

$$\texttt{(in)}\quad \frac{C[m[l^m \cdot \mu^m] \oplus n[l^n \cdot \mu^n]],\quad l^m : \texttt{in}\, n.a,\quad \bigcup_{l\in\lambda} l : unused}{C[n[l^n \cdot \mu^n \oplus m[l^m \cdot \mu^m \oplus tree(a,\lambda)]]],\quad l^m : actproc(a),\quad proc(a,\lambda)}$$

$$\texttt{(out)}\quad \frac{C[n[l^n \cdot \mu^n \oplus m[l^m \cdot \mu^m]],\quad l^m : \texttt{out}\, n.a,\quad \bigcup_{l\in\lambda} l : unused}{C[m[l^m \cdot \mu^m \oplus tree(a,\lambda)] \oplus n[l^n \cdot \mu^n]],\quad l^m : actproc(a),\quad proc(a,\lambda)}$$

$$\texttt{(fold)}\quad \frac{l : M.a,\quad l :!M.a}{l :!M.a} \qquad\qquad \texttt{(unfold)}\quad \frac{l :!M.a}{l : M.a,\quad l :!M.a}$$

Table 1. *The transition schemata. For axioms* `(in)` *and* `(out)`*, we assume that* λ *is a sequence of distinct labels such that* $|\lambda| = \#amb(a)$.

Take a sequence of labels $\lambda = l'_1 \dots l'_{|K|}\lambda_1 \dots \lambda_{|K|}$ such that $|\lambda_i| = \#amb(a''_i)$ for all $i \in K$.

The function $tree(a,\lambda)$ constructs a portion of the ambient tree representing the active ambients in a, where nodes are labelled with the elements of λ taken in breadth first, left-to-right order:

$$tree(a,\lambda) = \bigoplus_{k\in K} n_k[l'_k \cdot tree(a''_k, \lambda_k)].$$

The function $actproc(a)$ gives the portion of a corresponding to the active (unguarded) sequential and replicated subprocesses:

$$actproc(a) = \bigoplus_{i\in I} M_i.a_i \oplus \bigoplus_{j\in J} !M'_j.a'_j.$$

For each active ambient in a, the function $proc(a,\lambda)$ constructs the marking for the places of the corresponding subnet:

$$proc(a,\lambda) = \bigoplus_{k\in K} l'_k : actproc(a''_k) \oplus \bigoplus_{k\in K} proc(a''_k, \lambda_k).$$

The set $Trans$ contains all the instances of the transition schemata listed in Table 1: axioms `(in)` and `(out)` deal with the execution of a capability, and axioms `(fold)` and `(unfold)` enable us to apply the structural congruence law for replication to unguarded processes.

The P/T net used to decide rechability is constructed as follows (where the number n is used to represent the maximal number of active ambients to be considered in the P/T net).

Definition 4.8. Let P be a $\mathrm{pMA}^{-\texttt{open}}_{g!}$ process, let $\mathcal{N}$ denote the set of ambient names occurring in P and let n be a natural number such that $\#amb(P) \leqslant n$. We define the net

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

$Net(P,n) = (S,T)$, where

$$\begin{array}{rcl} S & = & \bigcup_{i=0}^{n}(\{l_i : Q \mid Q \in sub(\alpha(P))\} \cup \{l_i : unused\}) \ \cup \ \mathscr{T}_n^{\mathscr{N}} \\ T & = & \{(c,p) \in Trans \mid c,p \subseteq S\}. \end{array}$$

Note that $Net(P,n)$ is a finite P/T net.

The following definition describes how to map a derivative of P to a marking of the net.

Definition 4.9. Let P be a $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ process and Q be a process such that $P \rightarrow^* Q$. Consider a natural number n such that $\#amb(Q) \leqslant n$.

Let λ be a sequence of distinct labels in $\{l_1,\ldots,l_n\}$ such that $|\lambda| = \#amb(Q)$, and let the set of labels not in λ be $C_\lambda = \{l_i \mid i = 1,\ldots,n \wedge l_i \notin \lambda\}$.

The *decomposition* of Q with respect to λ is defined as[†]

$$\begin{array}{rl} dec(Q, l_0\lambda) = & l_0 \cdot tree(\alpha(Q),\lambda), \\ & l_0 : actproc(\alpha(Q)), \\ & proc(\alpha(Q),\lambda), \\ & \bigcup_{l \in C_\lambda} l : unused. \end{array}$$

The decomposition of Q turns out to be a marking of $Net(P,n)$ because the following property holds.

Proposition 4.10. Let P,Q be a $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ process. If $P \rightarrow^* Q$, then $sub(\alpha(Q)) \subseteq sub(\alpha(P))$

Proof. We use induction on the length of the computation $P \rightarrow^* Q$. If the length is 0, the proposition holds trivially. If we consider $P \rightarrow^* Q' \rightarrow Q$, by the induction hypothesis, $sub(\alpha(Q')) \subseteq sub(\alpha(P))$. Proceeding by induction on the length of the proof of $Q' \rightarrow Q$, it is easy to prove that $sub(\alpha(Q)) \subseteq sub(\alpha(Q'))$. □

We now prove two Propositions: the first relates the P/T net semantics to the structural congruence $\equiv$, and the second is concerned with the reduction relation $\rightarrow$ for processes.

Proposition 4.11. Let P be a $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ process and Q be a process such that $P \rightarrow^* Q$. Consider a natural number n such that $\#amb(Q) \leqslant n$. Given a $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ process Q', we have $Q \equiv Q'$ if and only if there exists a sequence λ of distinct labels in $\{l_0,\ldots,l_n\}$ such that $|\lambda| = \#amb(Q) = \#amb(Q')$ and $dec(Q,\lambda) \xrightarrow{\sigma} dec(Q',\lambda)$ in $Net(P,n)$ where σ is a sequence that includes an arbitrary number (possibly 0) of transitions that are instances of `(fold)` or `(unfold)`.

Proof. The proof is by induction on the length of the proof of $Q \equiv Q'$. □

Proposition 4.12. Let P be a $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ process and Q be a process such that $P \rightarrow^* Q$. Consider a natural number n such that $\#amb(Q) \leqslant n$. Given a $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ process Q' such that $\#amb(Q') \leqslant n$, we have $Q \rightarrow Q'$ if and only if there exist two sequences λ

[†] To be precise, in this case we again have to fix an order on the elements of the ambient multiset $\alpha(Q)$, as in Definition 4.7.

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

and λ' of distinct labels in $\{l_0, \ldots, l_n\}$ such that $|\lambda| = \#amb(Q)$, $\lambda' = \#amb(Q')$, and $dec(Q, \lambda) \xrightarrow{\sigma} dec(Q', \lambda')$ in $Net(P, n)$ where σ is a sequence of transitions that includes an arbitrary number (possibly 0) of transitions that are instances of `(fold)` or `(unfold)` and exactly one transition which is an instance of `(in)` or `(out)`.

Proof. The proof is by induction on the length of the proof of $Q \to Q'$, where Proposition 4.11 is used for the case where the last rule applied to prove $Q \to Q'$ is rule (6) of Definition 2.3. □

We conclude this section by defining the P/T system that can be used to solve the reachability problem for $\text{pMA}_{g!}^{-\texttt{open}}$ processes.

Definition 4.13. Let P be a $\text{pMA}_{g!}^{-\texttt{open}}$ process and n be a natural number such that $\#amb(P) \leqslant n$. We define $Sys(P, n) = (S, T, m_0)$, where $(S, T) = Net(P, n)$ and the initial marking is

$$m_0 = dec(P, l_0 \ldots l_{\#amb(P)}).$$

In order to prove that we can actually use a P/T system to check the reachability of process R from process P, we need another preliminary result.

Proposition 4.14. Let P, R be $\text{pMA}_{g!}^{-\texttt{open}}$ processes such that $P \to^* R$. If Q is a process traversed by the sequence of reductions $P \to^* R$, that is, $P \to^* Q \to^* R$, then $\#amb(Q) \leqslant \#amb(R)$.

Proof. The proof is by induction on the length of $Q \to^* R$. If the length is 0, the proposition holds trivially. If $Q \to Q' \to^* R$, by the induction hypothesis, $\#amb(Q') \leqslant \#amb(R)$. Proceeding by induction on the length of the proof of $Q \to Q'$, it is easy to prove that $\#amb(Q) \leqslant \#amb(Q')$ (due to the absence of the `open` capability in $\text{pMA}_{g!}^{-\texttt{open}}$). □

Theorem 4.15. Let P, R be $\text{pMA}_{g!}^{-\texttt{open}}$ processes such that $\#amb(P) \leqslant \#amb(R)$. Then $P \to^* R$ if and only if there exists a sequence λ of distinct labels in $\{l_0, \ldots, l_{\#amb(R)}\}$ such that $|\lambda| = \#amb(R) + 1$ and $dec(R, \lambda)$ is a marking of $Sys(P, \#amb(R))$ that is reachable.

Proof. By Proposition 4.14, we know that each computation $P \to^* R$ traverses processes with a number of active ambients smaller than or equal to $\#amb(R)$. The theorem follows from the fact that, by Proposition 4.12, the P/T system $Sys(P, \#amb(R))$ faithfully reproduces these computations. □

We have the following trivial corollary.

Corollary 4.16. The reachability problem is decidable in $\text{pMA}_{g!}^{-\texttt{open}}$.

Proof. By Theorem 4.15, we have that in order to solve the reachability problem for the $\text{pMA}_{g!}^{-\texttt{open}}$ processes P and R, we can check the reachability of one of the markings in the set $\{dec(R, \lambda) \mid \lambda \text{ is a sequence of distinct labels in } \{l_0, \ldots, l_{\#amb(R)}\}\}$ in the P/T system $Sys(P, \#amb(R))$. As this set of markings is finite, the decidability of this problem follows directly from the decidability of reachability for P/T systems. □

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

5. Target reachability in $\mathrm{pMA}_{g!}^{-\mathtt{open}}$

In this section we prove that for the calculus $\mathrm{pMA}_{g!}^{-\mathtt{open}}$, an extension of the reachability problem, which we call *target reachability*, also turns out to be decidable. Target reachability generalises classical reachability by allowing a generic description of the target process. More precisely, we can impose constraints on the number of occurrences of guarded processes inside each active ambient in the target process. Such constraints are both lower bounds (for example, there must be at least one instance of the guarded process $M.P$ in a given ambient) and upper bounds (for example, there can be at most two occurrences of the guarded process $M.P$ in a given ambient).

We need to introduce some additional notation for describing target processes.

We introduce a notion of normal form for processes that forbids the presence of both the unreplicated and replicated versions of a guarded term in a parallel composition. Each process can be transformed in a structurally congruent process into the normal form by using the monoidal axioms for parallel composition and applying the axiom for replication from right to left (that is, $M.P|!M.P$ can be rewritten as $!M.P$).

Definition 5.1 (normal form). A $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ process P is in normal form if

$$P \;=\; \prod_i M_i.P_i \;\mid\; \prod_j !M'_j.P'_j \;\mid\; \prod_k n_k[P''_k]$$

and the following conditions hold:

— P_i, P'_j, P''_k are in normal form for all i, j, k;
— if $M_i = M'_j$, then $P_i \neq P'_j$.

Proposition 5.2. Let P be a $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ process. Then there exists a process Q in normal form such that $P \equiv Q$.

Proof. The proof is by induction on the structure of P. □

Definition 5.3 (target). The set of *targets* is defined by the grammar

$$T \;::=\; \mathbf{0} \;\mid\; \mathtt{any} \;\mid\; q \leqslant M.P \leqslant q' \;\mid\; T|T \;\mid\; !M.P \;\mid\; n[T]$$

where $q \in \mathbb{N}$ and $q' \in \mathbb{N} \cup \{\infty\}$[†].

The target any requires the presence of zero or more occurrences of any process. The target $q \leqslant M.P \leqslant q'$ requires the presence of k occurrences of process $M.P$, with $q \leqslant k \leqslant q'$ (if $q' = \infty$ there is no upper bound to the number of occurrences). And the target $!M.P$ requires the presence of one or more occurrences of process $!M.P$. Since the behaviour of processes $\prod_k !M.P$ is the same for any $k \geqslant 1$, we prefer just to require the presence – or absence – of a replicated process rather than to provide upper and lower bounds to the number of its occurrences. Targets can be composed in parallel, and can be nested in ambients.

[†] We assume that $q \leqslant \infty$ for all $q \in \mathbb{N}$.

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

As an example, consider the target

$$n[1 \leqslant \mathtt{in}\, m.P \leqslant 2] \;\mid\; m[!\mathtt{in}\, n.Q \mid k[\mathtt{any} \mid 3 \leqslant \mathtt{out}\, m.R \leqslant \infty]].$$

This target requires that ambient n contains one or two occurrences of process $\mathtt{in}\, m.P$, ambient m contains only occurrences of process $!\mathtt{in}\, n.Q$ (at least one occurrence is required) plus an ambient k that contains at least three occurrences of the process $\mathtt{out}\, m.R$. Moreover, this target also requires that there is no process at top level.

We will only consider a proper subset of *well-formed* targets defined as follows.

Basically, a target is well formed if the upper and lower bounds on guarded terms are satisfiable (that is, target $3 \leqslant M.P \leqslant 2$ is not well formed) and if the presence of a replicated version of a guarded process prevents the occurrence of the non-replicated version of the same process in a parallel composition (that is, target $M.P \mid !M.P$ is not well formed). We also require that at most one occurrence of a replicated process is present in a parallel composition (that is, target $!M.P \mid !M.P$ is not well formed).

In the formal definition of a well-formed target we make use of a structural congruence $\equiv_T$ for targets, which is defined as the least congruence satisfying the following axioms:

$$\begin{aligned} T|\mathbf{0} &\equiv_T T \\ T_1|T_2 &\equiv_T T_2|T_1 \\ T_1|(T_2|T_3) &\equiv_T (T_1|T_2)|T_3. \end{aligned}$$

Definition 5.4 (well-formed target). A *target* T is well formed if there exists a target $S = \prod_i q_i \leqslant M_i.P_i \leqslant q_i' \mid \prod_j !M_j'.P_j' \mid \prod_k n_k[T_k'']$ such that the following conditions hold:

— processes P_i, P_j' are in normal form for all i, j;
— either $T \equiv_T S$ or $T \equiv_T S \mid \mathtt{any}$;
— $q_i \leqslant q_i'$ for all i;
— if $M_i = M_j'$ then $P_i \neq P_j'$;
— if $M_i = M_j$ and $P_i = P_j$, then $i = j$;
— if $M_i' = M_j'$ and $P_i' = P_j'$, then $i = j$;
— T_k'' is well formed for all k.

We now define the set of processes $set(T)$ that satisfy the constraints imposed by a target T. Basically, we require the presence of the required number of occurrences of a prefixed process in each ambient; if the upper bound is ∞, the presence of a replicated version of the process satisfies the target (that is, process $n[!\mathtt{in}\, n.\mathbf{0}]$ satisfies the target $n[3 \leqslant \mathtt{in}\, n.\mathbf{0} \leqslant \infty]$). If the target any is present, other (different) processes may be present also. As previously discussed, if there is a replicated process in the target, we just require the presence of at least one occurrence of the replicated process.

Definition 5.5 (set(T)). Let T be a well-formed target. A $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ process P is in $set(T)$ if

$$P \equiv \prod_h L_h.P_h \mid \prod_g !L_g'.P_g' \mid \prod_k n_k[P_k'']$$

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

and there exists a target

$$S = \prod_i q_i \leqslant M_i.Q_i \leqslant q_i' \mid \prod_j !M_j'.Q_j' \mid \prod_k n_k[T_k'']$$

such that the following conditions hold:

— either $T \equiv_T S$ or $T \equiv_T S \mid \texttt{any}$;
— for all i, either $q_i \leqslant |\{h \mid L_h.P_h = M_i.Q_i\}| \leqslant q_i'$ or $q_i' = \infty$ and there exists g such that $L_g'.P_g' = M_i.Q_i$;
— for all j there exist g such that $L_g'.P_g' = M_j'.Q_j'$;
— if $T \equiv_T S$, then for every h there exists i such that either $L_h.P_h = M_i.Q_i$ or $L_h.P_h = M_i'.Q_i'$ and for every g there exists j such that $L_g'.P_g' = M_j'.Q_j'$;
— for every k, we have $P_k'' \in set(T_k'')$.

It is worth noting that $set(T)$ is compatible with the structural congruence relation as formalised by the following Proposition.

Proposition 5.6. Let T be a target and P and Q two $\text{pMA}_{g!}^{-\texttt{open}}$ processes such that $P \equiv Q$. Then $P \in set(T)$ if and only if $Q \in set(T)$.

We are now ready to formalise the notion of *target reachability*.

Definition 5.7. Let P be a $\text{pMA}_{g!}^{-\texttt{open}}$ process and T be a target. We say that T is a target reachable from P (denoted by $TReach(P, T)$) if there exists a process Q such that $P \rightarrow^* Q$ and $Q \in set(T)$.

5.1. *Target marking reachability on P/T nets*

The proof of decidability of target reachability in $\text{pMA}_{g!}^{-\texttt{open}}$ follows from the encoding of $\text{pMA}_{g!}^{-\texttt{open}}$ into P/T nets discussed in the previous section. However, in this case the P/T net semantics does not reduce the target reachability problem for processes to some problem that is already known to be decidable for P/T nets. In fact, target reachability is not reduced to classical reachability due, for example, to the presence of `any`, which allows for the presence of any additional process in a given ambient.

The *coverability* problems for P/T nets allows for the presence of any additional token with respect to the given target.

Definition 5.8. Let $N = (S, T, m_0)$ be a P/T system. The coverability problem for marking m consists of checking if there exists m' such that $m_0 \rightarrow^* m'$ and m' *covers* m, that is, $m \subseteq m'$.

Coverability is known to be a decidable problem for P/T nets (Karp and Miller 1969). However, once again, target reachability for processes is not mapped to coverability on P/T nets by the P/T net semantics presented in the previous section. Consider, for example, the target $2 \leqslant M.P \leqslant 2$, which requires the presence of exactly 2 instances of the process $M.P$ (and no other additional processes).

For the reasons discussed above, we need to introduce a new problem for P/T nets, which we call *target marking reachability*, that allows us to specify both a lower and upper

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

bound to the number of tokens in each place of the net, and consists of checking if it is possible to reach a marking that satisfies these constraints.

Definition 5.9 (target marking). Let $N = (S, T)$ be a P/T net. A target marking of N is a pair of functions $(inf, sup) \in (S \to \mathbb{N}) \times (S \to \mathbb{N} \cup \infty)$ such that, for all $s \in S$, $inf(s) \leqslant sup(s)$.

Definition 5.10 (target marking satisfiablity). Let $N = (S, T)$ be a P/T net. A marking m of N satisfies a target marking (inf, sup) of N if, for all $s \in S$, $inf(s) \leqslant m(s) \leqslant sup(s)$.

Definition 5.11 (target marking reachability). Let $N = (S, T, m_0)$ be a P/T system. A target marking (inf, sup) is reachable if there exists a marking m such that $m_0 \to^* m$ and m satisfies (inf, sup).

Note that reachability and coverability are special cases of target marking reachability. Checking the reachability of marking m is equivalent to checking reachability of the target marking (m, m), while checking the coverability of m is equivalent to checking the reachability of the target marking $(m, \{(s, \infty) \mid s \in S\})$.

As the target marking reachability problem is more general than both reachability and coverability and, to the best of our knowledge, it is not included in any other decidable problem for P/T systems, we have to prove that it is indeed decidable. The proof shows how to reduce this problem to the verification of the reachability of at least one marking in a finite set of markings in a transformed P/T system. Formally, given a P/T system N and a target marking (inf, sup), we define the P/T system $TMSys(N, (inf, sup))$ as follows.

Definition 5.12. Let $N = (S, T, m_0)$ be a P/T system and (inf, sup) be a target marking of N. The P/T system $TMSys(N, (inf, sup)) = (S', T', m'_0)$ is defined as follows. Let $normal, ending \notin S$.

$$\begin{aligned} S' &= S \cup \{normal, ending\} \\ T' &= \{(c \cup normal, p \cup normal) \mid (c, p) \in T\} \cup \\ &\quad \{(normal, ending)\} \cup \\ &\quad \{(s \cup ending, ending) \mid sup(s) = \infty\} \\ m'_0 &= m_0 \cup normal. \end{aligned}$$

The P/T system $TMSys(N, (inf, sup))$ extends the P/T system N with the two places *normal* and *ending*. These places are used to divide the computation of the net into two phases. During the first phase the place *normal* holds one token and each computation of the P/T system N can be executed. The second phase is started non-deterministically by moving the token in the place *normal* to the place *ending*. During this second phase, tokens can be removed from the places having the ∞ upper bound.

We now characterise a finite set of markings for $TMSys(N, (inf, sup))$, which we call $TMMark(N, (inf, sup))$, such that the target marking (inf, sup) is reachable in N if and only if one of the markings in $TMMark(N, (inf, sup))$ is reachable in the P/T system $TMSys(N, (inf, sup))$.

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

Definition 5.13. Let $N = (S, T, m_0)$ be a P/T system and (inf, sup) be a target marking of N. We use $TMMark(N, (inf, sup))$ to denote the following set of markings of $TMSys(N, (inf, sup))$:

$$TMMark(N, (inf, sup)) = \{m \mid \forall s \in S : \big((sup(s) = \infty \Rightarrow m(s) = inf(s)) \wedge (sup(s) \neq \infty \Rightarrow inf(s) \leqslant m(s) \leqslant sup(s))\big) \wedge m(normal) = 0 \ \wedge \ m(ending) = 1 \ \}.$$

Proposition 5.14. Let $N = (S, T, m_0)$ be a P/T system and (inf, sup) be a target marking of N. The set of markings $TMMark(N, (inf, sup))$ is finite.

Proof. Consider a place p in a marking $m \in TMMark(N, (inf, sup))$. If $p \in S$ with $sup(p) = \infty$ or $p \in \{normal, ending\}$, we have that $m(p)$ is uniquely defined by $TMMark(N, (inf, sup))$. The only case in which we could have multiple possible values for $m(p)$ is if $p \in S$ with $sup(p) \neq \infty$. But in these cases there is a finite number of possible values for $m(p)$, that is, the values in the finite interval $[inf(p), sup(p)]$. □

Proposition 5.15. Let $N = (S, T, m_0)$ be a P/T system and (inf, sup) be a target marking of N. The target marking (inf, sup) is reachable in N if and only if one of the markings in the set $TMMark(N, (inf, sup))$ is reachable in $TMSys(N, (inf, sup))$.

Proof. Assume that the target (inf, sup) is reachable in $N = (S, T, m_0)$. This means that there exists a sequence of transitions $m_0 \rightarrow^* m$ in N such that $inf(s) \leqslant m(s) \leqslant sup(s)$. The same sequence of transitions leads to the computation $m_0 \oplus normal \rightarrow^* m \oplus normal$ in $TMSys(N, (inf, sup))$. This computation can be extended in such a way that a marking $m' \in TMMark(N, (inf, sup))$ is reached. Namely, consider $m \oplus normal \rightarrow m \oplus ending \rightarrow^* m'$ such that in the last part of the computation $m \oplus ending \rightarrow^* m'$ tokens are removed from each place s such that $sup(s) = \infty$, until the number of tokens become equal to $inf(s)$.

Assume now that any marking $m' \in TMMark(N, (inf, sup))$ is reachable in the P/T system $TMSys(N, (inf, sup))$. This means that there exists a sequence of transitions $m_0 \oplus normal \rightarrow^* m'$ in $TMSys(N, (inf, sup))$ such that $m' \in TMMark(N, (inf, sup))$. By the definition of $TMMark(N, (inf, sup))$, we have $m'(ending) = 1$ and $m'(normal) = 0$. We now separate the sequence of transitions $m_0 \oplus normal \rightarrow^* m'$ into two parts, $m_0 \oplus normal \rightarrow^* m \rightarrow^* m'$, where m is the last traversed marking having one token in the place *normal* (and no token in *ending*). Now consider the projections $proj_m$ and $proj_{m'}$ of the markings m and m' on places in S (the places in the initial P/T system $N = (S, T, m_0)$). We first observe that the sequence of transitions $m_0 \oplus normal \rightarrow^* m$ can also be fired in N, that is, $m_0 \rightarrow^* proj_m$. We conclude the proof by observing that for each $s \in S$ we have $inf(s) \leqslant proj_m(s) \leqslant sup(s)$. In fact, it is easy to see that $proj_m$ and $proj_{m'}$ differ only in the fact that $proj_{m'}$ has fewer tokens in those places $s \in S$ such that $sup(s) = \infty$, and, moreover, $m' \in TMMark(N, (inf, sup))$ guarantees that $inf(s) \leqslant proj_{m'}(s) \leqslant sup(s)$ for each $s \in S$. □

As a consequence of the two propositions above and of the decidability of reachability on P/T systems, we get the following theorem.

Theorem 5.16. Target marking reachability is decidable for P/T systems.

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

Proof. By Proposition 5.15, in order to verify the reachability of a target (inf, sup) in the P/T system N, it is sufficient to check the reachability of one of the markings in the set $TMMark(N,(inf,sup))$ in the P/T system $TMSys(N,(inf,sup))$. The theorem then follows from the decidability of reachability in P/T nets and the finiteness of $TMMark(N,(inf,sup))$, which was proved in Proposition 5.14. □

5.2. *Reducing target reachability on processes to target marking reachability on P/T nets*

We complete this section by showing how to use the P/T net semantics for $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ defined in Section 4 to reduce the target reachability problem for $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ processes to target marking reachability in the corresponding P/T system.

Consider a $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ process P and a well-formed target T. We proceed as follows. We first define how to extract the number of active ambients in T, which we denote with $\#amb(T)$. Then we consider the P/T system $Sys(P,\#amb(T))$. Finally, we show how to define a finite set of target markings $TargMark(P,T,\lambda)$, parameterised by a sequence of labels λ, having the following property: the target T is reachable from the process P if and only if there exists a sequence of labels λ such that there exists a target marking in $TargMark(P,T,\lambda)$ that is reachable in the P/T system $Sys(P,\#amb(T))$.

Definition 5.17. Consider the well-formed target T and let

$$S = \prod_{i\in I} q_i \leqslant M_i.P_i \leqslant q_i' \mid \prod_{j\in J} !M_j'.P_j' \mid \prod_{k\in K} n_k[T_k'']$$

be a corresponding target such that either $T \equiv_T S$ or $T \equiv_T S|\mathtt{any}^{\dagger}$.

The number of active ambients in T is defined by

$$\#amb(T) = |K| + \sum_{k\in K} \#amb(T_k'').$$

It is easy to see that all the processes belonging to $set(T)$ have a number of active ambients corresponding to $\#amb(T)$.

We now define the tree associated with a target T for a given sequence of labels λ.

Definition 5.18. Consider the well-formed target T and let

$$S = \prod_{i\in I} q_i \leqslant M_i.P_i \leqslant q_i' \mid \prod_{j\in J} !M_j'.P_j' \mid \prod_{k\in K} n_k[T_k''].$$

Take a sequence of labels $\lambda = l_1' \ldots l_{|K|}' \lambda_1 \ldots \lambda_{|K|}$ such that $|\lambda_i| = \#amb(T_i'')$ for all $i \in K$. The function $tree(T,\lambda)$ is defined by

$$tree(T,\lambda) = \bigoplus_{k\in K} n_k[l_k' \cdot tree(T_k'', \lambda_k)].$$

[†] The existence of such a target S is guaranteed by the definition of well formedness (see Definition 5.4)

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

In the following definition of $TargMark(P, T, \lambda)$, we overload the operator $\oplus$ by applying it to target markings also: given two target markings (inf_1, sup_1) and (inf_2, sup_2) defined on disjoint sets of places, we use $(inf_1, sup_1) \oplus (inf_2, sup_2)$ to denote the target marking (inf, sup) where inf is the disjoint union of the functions inf_1 and inf_2, and sup is the disjoint union of the functions sup_1 and sup_2.

Definition 5.19 (TargMark(P,T,λ)). Let P be a $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ process and T be a well-formed target reachable from P, that is, $TReach(P, T)$. Let

$$S = \prod_{i \in I} q_i \leqslant M_i.P_i \leqslant q_i' \mid \prod_{j \in J} !M_j'.P_j' \mid \prod_{k \in K} n_k[T_k'']$$

be a target such that either $T \equiv_T S$ or $T \equiv_T S|\mathtt{any}^{\dagger}$.

Take a sequence of labels $\lambda = l_0'\lambda_1 \ldots \lambda_{|K|}$ such that $|\lambda_k| = \#amb(T_k'')$ for all $k \in K$. We define $TargMarkSub(P, T, \lambda)$ as the following set of target markings:

$$\begin{aligned} TargMarkSub(P, T, \lambda) \;=\; \{(inf_T, sup_T) \oplus \textstyle\bigoplus_{k \in K} (inf_{T_k''}, sup_{T_k''}) \;\mid\; \\ (inf_{T_k''}, sup_{T_k''}) \in TargMarkSub(P, T_k'', \lambda_k) \wedge \\ (inf_T, sup_T) \text{ satisfes the property below}\} \end{aligned}$$

where (inf_T, sup_T) is a target marking defined on the set of places $\{l_0' : Q \mid Q \in sub(\alpha(P))\}$ such that:

— for all $i \in I$, one of the following holds:
  - $inf(l_0' : M_i.P_i) = q_i$ and $sup(l_0' : M_i.P_i) = q_i'$;
  - $q_i' = \infty$, $inf(l_0' : M_i.P_i) = 0$, $sup(l_0' : M_i.P_i) = \infty$, $inf(l_0' :!M_i.P_i) = 1$, and $sup(l_0' : !M_j.P_j) = \infty$;

— for all $j \in J$, we have $inf(l_0' :!M_j'.P_j') = 1$ and $sup(l_0' :!M_j'.P_j') = \infty$;

— for all other places $l_0' : Q$ not considered in the previous items, $inf(l_0' : Q) = 0$ and either $sup(l_0' : Q) = 0$ if $T \equiv_T S$, or $sup(l_0' : Q) = \infty$ if $T \equiv_T S|\mathtt{any}$.

We define the set of target markings associated with the source process P, the target T and the sequence of labels $l_0\lambda$ as follows:

$$\begin{aligned} TargMark(P, T, l_0\lambda) \;=\; \{(infTree, supTree) \oplus (inf, sup) \mid \\ (inf, sup) \in TargMarkSub(P, T, l_0\lambda) \wedge \\ (infTree, supTree) \text{ satisfes the property below}\} \end{aligned}$$

where $(infTree, supTree)$ is a target marking defined on the set of places $\mathcal{T}^{\mathcal{N}}_{\#amb(T)} \cup \bigcup_{l \in l_0\lambda} l : unused$ (where $\mathcal{N}$ is the set of ambient names occurring in P) such that:

— $infTree(l_0 \cdot tree(T, \lambda)) = 1$ and $supTree(l_0 \cdot tree(T, \lambda)) = 1$;

— if $p \neq l_0 \cdot tree(T, \lambda)$, then $infTree(p) = 0$ and $supTree(p) = 0$.

Note that, given a process P, a target T and a sequence of labels λ, the set of target markings $TargMark(P, T, \lambda)$ is finite.

We are now ready to formalise the correspondence between the satisfiability of a target for processes and the satisfiability of a target marking for P/T nets.

† In this case also, the existence of S is guaranteed by Definition 5.4.

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

Proposition 5.20. Let P, R be $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ processes such that $P \rightarrow^* R$. Let T be a well-formed target reachable from P, that is, $TReach(P, T)$. Then $R \in set(T)$ if and only if for every sequence λ of distinct labels in $\{l_1, \ldots, l_{\#amb(T)}\}$ such that $|\lambda| = \#amb(R)$, we have $dec(R, l_0\lambda)$ satisfies at least one target marking in $TargMark(P, T, l_0\lambda)$.

Proof. The proof is by induction on the maximal depth of the nesting of ambients in the target T, and is based on the simple observation that the definition of $set(T)$ (see Definition 5.5) imposes on processes the same constraints as are imposed on markings in the definition of $TargMarkSub(P, T, \lambda)$ (see Definition 5.19). □

We can now complete the formalisation of the reduction of target reachability of processes to target marking reachability problem on P/T nets.

Theorem 5.21. Let P be a $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ process and T be a well-formed target such that $\#amb(P) \leqslant \#amb(T)$.

Then T is reachable from P, that is, $TReach(P, T)$, if and only if there exists a sequence λ of distinct labels in $\{l_0, \ldots, l_{\#amb(T)}\}$ such that $|lambda| = \#amb(T) + 1$ and at least one of the target markings in $TargMark(P, T, \lambda)$ is reachable in $Sys(P, \#amb(T))$.

Proof. By definition, the target T is reachable by the process P if and only if there exists a process R such that:

(i) $R \in set(T)$ and
(ii) R is reachable from the process P.

By Proposition 5.20, part (i) holds if and only if for every sequence of labels $\lambda = l_0\lambda'$ we have that $dec(R, \lambda)$ satisfies at least one of the target markings in $TargMark(P, T, \lambda)$.

By Theorem 4.15, part (ii) holds if and only if there exists a sequence of labels λ such that $dec(R, \lambda)$ is reachable in $Sys(P, \#amb(T))$. □

We have the following as a trivial corollary.

Corollary 5.22. The target reachability problem is decidable in $\mathrm{pMA}_{g!}^{-\mathtt{open}}$.

Proof. By Theorem 5.21, the target reachability problem for the $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ process P and the target T can be solved by checking the reachability of one of the target markings in the set $\oplus_\lambda TargMark(P, T, l_0\lambda)$, where λ ranges over the set of sequences of distinct labels in $\{l_1, \ldots, l_{\#amb(T)}\}$, in the P/T system $Sys(P, \#amb(T))$.

As the above set of sequences of labels is finite, and for each sequence λ we have that $TargMark(P, T, l_0\lambda)$ is finite (see the observation after Definition 5.19), we can conclude that the set of target markings $\oplus_\lambda TargMark(P, T, l_0\lambda)$ to be considered is finite also. Hence, the decidability result follows directly from the decidability of target marking reachability for P/T systems proved in the Theorem 5.16. □

6. Boxed Ambients

In this section we show how the target reachability analysis presented in the previous section for $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ can also be applied to richer non-pure versions of MA that

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

include communication. As a testbed, we decided to consider Boxed Ambients (Bugliesi *et al.* 2004), which is the most relevant dialect of Mobile Ambients that omits the `open` capability (which is the operator we had to remove from Mobile Ambients to prove the decidability of reachability) and introduces a fairly rich set of communication capabilities that allow communication both between processes residing in the same ambient and between processes residing in parent–child ambients.

The Boxed Ambients model was developed for the formalisation and analysis of security aspects related to *access control* policies. In this particular context, we consider target reachability to be an interesting tool for the analysis of the correctness of a specified system with respect to policies described in terms of capabilities for ambients to enter other ambients (for instance, see the example in the Introduction where we discussed the case of a *virus* entering a *notebook* using a *trojan* ambient as a means of transport).

We will need to simplify the calculus if we are going to be able to apply target reachability to Boxed Ambients. The simplified calculus we consider is called BA^{-}.

Definition 6.1 (BA^{-}). Let *Name*, ranged over by $n, m, \ldots$, be a denumerable set of ambient names and *Var*, ranged over by $x, y, \ldots$, be a denumerable set of variables, such that $Name \cap Var = \emptyset$. The set of sequences of capabilities is defined by

$$C \ ::= \ \mathtt{in}\, n \mid \mathtt{out}\, n \mid \mathtt{in}\, x \mid \mathtt{out}\, x \mid C.C.$$

The set of expressions is defined by the grammar

$$e \ ::= \ n \mid x \mid C.$$

The set of locations, ranged over by η, is $Name \cup Var \cup \{\uparrow, \star\}$. The set of processes is defined by the grammar

$$\begin{array}{lll} P & ::= & \mathbf{0} \mid M.P \mid P|P \mid !M.P \mid n[P] \mid x[P] \\ M & ::= & C \mid x \mid (x)^{\eta} \mid \langle e \rangle^{\eta}. \end{array}$$

The main differences between Boxed Ambients and pure Mobile Ambients is the elimination of the `open` capability, which is compensated for by the introduction of a fairly rich communication mechanism. Processes residing in the same ambient or in two different ambients that are in a parent–child relation, can communicate with each other. The expressions that can be communicated can be variables, ambient names or sequences of capabilities.

The capabilities in a sequence are either on a fixed ambient name or on a variable, which will be subsequently instantiated by a communication. Expressions, representing the contents of messages, may be ambient names, variables or sequences of capabilities.

More precisely, communication primitives make use of locations: location n denotes communication with a process in a child ambient with name n; location x will be instantiated with an ambient name by a previous communication; location $\uparrow$ denotes communication with a process in the parent ambient; and location $\star$ (which is often omitted) denotes local communication. The input process $(x)^{\eta}.P$ and the output process $\langle e \rangle^{\eta}.P$ allow us to model communication, where η denotes the location. Both input and output processes are guarded processes, and $(x)^{\eta}$ acts as a binder for the occurrences of

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

variable x in P. The notions of free and bound variables (denoted by $fv(P)$ and $bv(P)$), and of closed processes are defined as usual. In the following we assume that BA$^-$ is restricted to the set of closed processes.

We will now discuss the differences between the original Boxed Ambients proposal (Bugliesi *et al.* 2004) and the fragment BA$^-$ that we consider in this paper. In particular, BA$^-$ imposes the following restrictions on Mobile Ambients:

— Replication is guarded, that is, it can only be applied to prefixed processes having the form $!M.P$.

— There is a constrained use of variables in sequences of capabilities. Namely, variables in Boxed Ambients can be included in a sequence of capabilities. As the following example shows, this allows the production of sequences of capabilities of unbounded length:

$$\langle \mathtt{in}\, n \rangle \mid !(x).\langle \mathtt{in}\, n.x \rangle.$$

In fact, after i synchronisations, the above process becomes

$$\underbrace{\langle \mathtt{in}\, n.\mathtt{in}\, n.\cdots.\mathtt{in}\, n \rangle}_{\text{sequence of length } i+1} \mid !(x).\langle \mathtt{in}\, n.x \rangle.$$

To ensure that the number of sequential subprocesses that can be generated from an initial BA$^-$ process remains finite (which is necessary if we are going to represent the behaviour of the initial process using a finite Petri net), we require that a variable cannot be a proper subsequence of a sequence of capabilities, thus excluding from the calculus the above process.

Moreover, for simplicity, we will consider a monadic version of the calculus.

We will now present the operational semantics, defined in terms of a structural congruence plus a reduction relation.

Definition 6.2 (structural congruence). The structural congruence $\equiv$ is the smallest congruence relation satisfying:

$$\begin{aligned} P \mid \mathbf{0} &\equiv P \\ P \mid Q &\equiv Q \mid P \\ P \mid (Q \mid R) &\equiv (P \mid Q) \mid R \\ !M.P &\equiv M.P \mid !M.P. \end{aligned}$$

Definition 6.3 (reduction relation). The reduction relation is the smallest relation $\rightarrow$ satisfying the following axioms and rules:

(1) $n[\mathtt{in}\, m.P \mid Q] \mid m[R] \rightarrow m[n[P \mid Q] \mid R]$.

(2) $m[n[\mathtt{out}\, m.P \mid Q] \mid R] \rightarrow n[P \mid Q] \mid m[R]$.

(4) $$\frac{P \rightarrow Q}{P \mid R \rightarrow Q \mid R}.$$

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

$$(5)\ \frac{P \ \rightarrow\ Q}{n[P] \ \rightarrow\ n[Q]}.$$

$$(6)\ \frac{P' \equiv P \quad P \ \rightarrow\ Q \quad Q' \equiv Q}{P' \ \rightarrow\ Q'}.$$

$$(7)\ (x).P \mid \langle e \rangle.Q \ \rightarrow\ P\{e/x\} \mid Q.$$

$$(8)\ (x)^n.P \mid n[\langle e \rangle.Q \mid R] \ \rightarrow\ P\{e/x\} \mid n[Q \mid R].$$

$$(9)\ n[(x)^{\uparrow}.P \mid Q] \mid \langle e \rangle.R \ \rightarrow\ n[P\{e/x\} \mid Q] \mid R.$$

$$(10)\ n[(x).P \mid Q] \mid \langle e \rangle^n.R \ \rightarrow\ n[P\{e/x\} \mid Q] \mid R.$$

$$(11)\ (x).P \mid n[\langle e \rangle^{\uparrow}.Q \mid R] \ \rightarrow\ P\{e/x\} \mid n[Q \mid R].$$

We use $P\{e/x\}$ to denote the process obtained by substituting each free occurrence of x in P with e. Note that, as we restricted ourselves to closed processes, there is no need to use alpha-conversion to avoid name captures when a variable is replaced with an expression.

Axioms (1) and (2) and rules (4)–(6) are the same as those considered in Definition 2.1. The new axioms, (7)–(11), define the communication mechanisms. Axiom (7) describes local communication. The input prefix $(x)^n.P$ in axiom (8) represents a request to read a datum sent by a process located in one of the child ambients n. In axiom (9), $(x)^{\uparrow}.P$ is a request to read a datum sent by a process located in the parent ambient. Dually, $\langle e \rangle^n.P$ in axiom (10) (respectively, $\langle e \rangle^{\uparrow}.P$ in axiom (11)) is a request to send e to a process located in the child ambient n (respectively, the parent ambient). Note that a direct remote communication between sibling ambients is not possible: either mobility or the intervention of the sibling ambients' parent is required.

Note that variables can be used with two different meanings: either as (sequence of) capabilities or as ambient names. Expressions can also have these two different meanings. For this reason, the substitution of one variable with an expression could generate a wrong process if the variable and expression have two different meanings. Consider, for instance, the process

$$(x).x[P] \mid \langle \mathtt{in}\, n \rangle.$$

If there is a communication, the substitution of $\mathtt{in}\, n$ for x, could generate the wrong process $\mathtt{in}\, n[P]$. In this paper we assume that only reductions that generate correct processes can be executed, thus avoiding the above reduction. Bugliesi *et al.* (2004) presented a typed version of Boxed Ambients that allows the absence of wrong reductions to be verifed statically.

The target reachability problem is defined for BA^{-} in the same way as we did for $\mathrm{pMA}^{-\mathtt{open}}_{g!}$ in Section 5: the only difference is that the syntax of targets is enriched with the new capabilities and the possibility of using variables as names for ambients.

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

Definition 6.4 (Target). The set of *targets* is defined by the grammar

$$\begin{array}{rcl} T & ::= & \mathbf{0} \mid \mathtt{any} \mid q \leqslant M.P \leqslant q' \mid T|T \mid !M.P \mid n[T] \mid x[T] \\ M & ::= & C \mid x \mid (x)^{\eta} \mid \langle e \rangle^{\eta} \end{array}$$

where $q \in \mathbb{N}$ and $q' \in \mathbb{N} \cup \{\infty\}$.

We conclude this section by showing how to use target reachability to analyse a simple example of an access control system. Consider two agents a and b and a resource r:

$$P \;=\; a[P_a] \mid b[P_b] \mid r[R \mid \langle M \rangle].$$

The processes P_a and P_b control the behaviour of agents a and b, respectively. Process R is a monitor controlling access to the resource r, which contains a message $\langle M \rangle$. Assume that agent a has the right to consume the messages inside the resource r, while b does not.

Formally, this means that it is possible that the configuration

$$Q' \;=\; b[P_b] \mid r[R' \mid a[P'_a]]$$

can be reached for any processes P'_a and R' such that R' does not contain any message $\langle M \rangle$. Formally, we accept $TReach(P, T')$ with

$$T' \;=\; b[P_b] \mid r[\mathtt{any} \mid 0 \leqslant \langle M \rangle \leqslant 0 \mid a[\mathtt{any}]].$$

Note that we assume that the agent b is not involved in the computation.

Moreover, we want to avoid the possibility of starting from P and reaching a configuration of the form

$$Q'' \;=\; a[P''_a] \mid r[R'' \mid b[P''_b]]$$

for any processes P''_a, P''_b and R'' such that R'' does not contain any message $\langle M \rangle$. Formally, we assume that $TReach(P, T'')$ with

$$T'' \;=\; a[\mathtt{any}] \mid r[\mathtt{any} \mid 0 \leqslant \langle M \rangle \leqslant 0 \mid b[\mathtt{any}]]$$

cannot hold.

Note that we formalise the absence of the message $\langle M \rangle$ by using the upper bound $\langle M \rangle \leqslant 0$, and that we add any in the specification of the contents of those ambients for which we do not impose any limitation besides those explicitly indicated. The ambient b in the target T' is the only ambient that does not make use of any, since we assume that the agent a should be able to access the resource r without any intervention from the agent b.

6.1. *Deciding target reachability in BA*$^{-}$

We now show how to enhance the technique presented in Section 5 to decide target reachability for $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ processes, so that we can also apply it to the more expressive calculus BA^{-}.

The first and main question we need to address is how to deal with variables and expressions. In fact, by substituting variables with expressions we can generate new sequential and replicated subprocesses, or ambients with a new name, that were not

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

present in the initial process. Consider, for instance, the process

$$(x).(x.P \mid !x.Q) \mid (y).y[R] \mid \langle \mathtt{in}\, n \rangle \mid \langle m \rangle,$$

which, after two reduction steps, can be transformed into

$$\mathtt{in}\, n.P \mid !\mathtt{in}\, n.Q \mid m[R],$$

which includes the new sequential subprocess $\mathtt{in}\, n.P$, the new replicated subprocess $!\mathtt{in}\, n.Q$ and the new named ambient $m[R]$.

Nevertheless, the limitations we have imposed on the calculus (namely, the impossibility of using a variable as a proper subsequence of a sequence of capabilities) allows us to define a finite set of possibly reachable sequential and replicated subprocesses, as well as a finite set of reachable named ambients.

Formally, we first have to extend the Definitions 4.4, 4.5, 4.6 and 4.7 by also considering the ambients $x[P]$ with variables as their names. This is achieved by simply assuming that in those definitions, n_k and n_i are used to range over variables also, and not just over ambient names.

Hence, we introduce the set of expressions of an ambient multiset.

Definition 6.5. Let a be an ambient multiset. The set of expressions of a, denoted with $exp(a)$, is defined as the least set satisfying the following properties:

— $exp(a)$ includes the expressions $\langle e \rangle$ occurring in a.
— $exp(a)$ is closed with respect to variable substitution, that is, given the expressions $\langle e \rangle, \langle e' \rangle \in exp(a)$, where e is an expression including the variable x, then $\langle e\{e'/x\} \rangle \in exp(a)$.

Note that for every ambient multiset a, the set $exp(a)$ is finite. This follows from the fact that the expressions in $exp(a)$ are defined on a finite set of variables and ambient names, and the length of expressions representing sequences of capabilities is smaller than the maximal length of a sequence occurring in a (this follows from the fact that variable substitution cannot increase the length of a sequence of capabilities).

We are now ready to associate with an ambient multiset a richer finite set of sequential and replicated subprocesses, which also includes processes that can be produced dynamically.

Definition 6.6. Let a be an ambient multiset. We define the set of its potentially reachable sequential and replicated subprocesses, denoted $reachSub(a)$, as the least set satisfying the following properties:

— $reachSub(a)$ includes the sequential and replicated subprocesses in $sub(a)$ (as defined in Definition 4.6).
— $reachSub(a)$ is closed with respect to variable substitution: that is, given the expression $\langle e \rangle \in exp(a)$ and the sequential or replicated subprocess $P \in reachSub(a)$, where x occurs free in P, we have $P\{e/x\} \in reachSub(a)$.

Note that because of the finiteness of the sets $exp(a)$ and $sub(a)$, the set $reachSub(a)$ is finite also.

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

$$\text{(local)}\quad \begin{array}{c} D[l\cdot\mu],\quad l:\langle e\rangle.a,\quad l:(x).a',\quad \bigcup_{l\in\lambda\lambda'} l:unused \\ \downarrow \\ D[l\cdot\mu\oplus tree(a,\lambda)\oplus tree(a',\lambda')],\quad l:actproc(a),\quad l:actproc(a'\{e/x\}), \\ proc(a,\lambda),\quad proc(a',\lambda') \end{array}$$

$$\text{(rp2c)}\quad \begin{array}{c} D[l\cdot\mu\oplus n[l^n\cdot\mu^n]],\quad l:(x)^n.a,\quad l^n:\langle e\rangle.a',\quad \bigcup_{l\in\lambda\lambda'} l:unused \\ \downarrow \\ D[l\cdot\mu\oplus tree(a,\lambda)\oplus n[l^n\cdot\mu^n\oplus tree(a',\lambda')]],\quad l:actproc(a\{e/x\}),\quad l^n:actproc(a'), \\ proc(a,\lambda),\quad proc(a',\lambda') \end{array}$$

$$\text{(rc2p)}\quad \begin{array}{c} D[l\cdot\mu\oplus n[l^n\cdot\mu^n]],\quad l^n:(x)^{\uparrow}.a,\quad l:\langle e\rangle.a',\quad \bigcup_{l\in\lambda\lambda'} l:unused \\ \downarrow \\ D[l\cdot\mu\oplus tree(a',\lambda')\oplus n[l^n\cdot\mu^n\oplus tree(a,\lambda)]],\quad l^n:actproc(a\{e/x\}),\quad l:actproc(a'), \\ proc(a,\lambda),\quad proc(a',\lambda') \end{array}$$

$$\text{(wp2c)}\quad \begin{array}{c} D[l\cdot\mu\oplus n[l^n\cdot\mu^n]],\quad l^n:(x).a,\quad l:\langle e\rangle^n.a',\quad \bigcup_{l\in\lambda\lambda'} l:unused \\ \downarrow \\ D[l\cdot\mu\oplus tree(a',\lambda')\oplus n[l^n\cdot\mu^n\oplus tree(a,\lambda)]],\quad l^n:actproc(a\{e/x\}),\quad l:actproc(a'), \\ proc(a,\lambda),\quad proc(a',\lambda') \end{array}$$

$$\text{(wc2p)}\quad \begin{array}{c} D[l\cdot\mu\oplus n[l^n\cdot\mu^n]],\quad l:(x).a,\quad l^n:\langle e\rangle^{\uparrow}.a',\quad \bigcup_{l\in\lambda\lambda'} l:unused \\ \downarrow \\ D[l\cdot\mu\oplus tree(a,\lambda)\oplus n[l^n\cdot\mu^n\oplus tree(a',\lambda')]],\quad l:actproc(a\{e/x\}),\quad l^n:actproc(a'), \\ proc(a,\lambda),\quad proc(a',\lambda') \end{array}$$

Table 2. *The transition schemata for communication. We assume that $\lambda\lambda'$ is a sequence of distinct labels such that $|\lambda| = \#amb(a)$ and $|\lambda'| = \#amb(a')$.*

We now consider the net transitions. For $\text{pMA}_{g!}^{-\texttt{open}}$, we defined the transitions *Trans* as the set of all the instances of the transition schemata in Table 1. These transitions are still valid for BA^-, but we need to add new transitions to deal with communication. In the definition of these additional transition schemata, we make use of ambient tree contexts $D[]$ that are different from those used in Table 1:

$$D[] \quad = \quad [] \quad | \quad l\cdot n[D[]]\oplus\bigoplus_{i\in I} n_i[t_i]$$

where n and n_i also range over variables since variables can be used as ambient names in BA^-.

The set *TransBA* contains all the transitions obtained as instances of the transition schemata in Tables 1 and 2.

We are now ready to define the P/T net that we can use to decide target reachability for processes in BA^-.

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

Definition 6.7. Let P be a BA$^-$ process, let $\mathcal{N}$ denote the set of ambient names and variables occurring in P, and let n be a natural number such that $\#amb(P) \leqslant n$. We define the net $NetBA(P,n) = (S,T)$, where

$$\begin{aligned} S &= \bigcup_{i=0}^{n}(\{l_i : Q \mid Q \in reachSub(\alpha(P))\} \cup \{l_i : unused\}) \cup \mathcal{T}_n^{\mathcal{N}} \\ T &= \{(c,p) \in TransBA \mid c,p \subseteq S\}. \end{aligned}$$

We define $SysBA(P,n) = (S,T,m_0)$, where $(S,T) = NetBA(P,n)$ and the initial marking is $m_0 = decBA(P, l_0 \ldots l_{\#amb(P)})$, where $decBA(\_)$ is defined as the function $dec(\_)$ in Definition 4.9 assuming that ambients can also be named using variables.

Note that $NetBA(P,n)$ is a finite P/T net.

We conclude this section by observing that the decidability of the target reachability problem for BA$^-$ processes can be proved in the same way as we did for pMA$_{g!}^{-\texttt{open}}$ by considering the new P/T net semantics defined above. In particular, we need to consider the function $TargMarkBA(P,T,\lambda)$, which is defined in the same way as the function $TargMark(P,T,\lambda)$ in Definition 5.19, but assuming that ambients can also be named using variables. Finally, it is sufficient to observe that, given a target T and a BA$^-$ process P, we have that T is reachable from P if and only if there exists a sequence λ of distinct labels in $\{l_0,\ldots,l_{\#amb(T)}\}$ such that $|\lambda| = \#amb(T)+1$ and at least one of the target markings in $TargMarkBA(P,T,\lambda)$ is reachable in $SysBA(P,\#amb(T))$.

7. Conclusion

We have discussed the decidability of reachability in Mobile Ambients. We have characterised a fragment of the pure and public Mobile Ambients, namely the `open`-free fragment with guarded replication, for which reachability is decidable. We call this fragment pMA$_{g!}^{-\texttt{open}}$. Our decidability result also holds for a variant of reachability, called target reachability, which enables us to specify a class of target processes characterised by a common structure of ambient nesting. We have also extended our results to cope with (local and parent–child) communication primitives, as provided by Boxed Ambients (Bugliesi *et al.* 2004).

The fragment pMA$_{g!}^{-\texttt{open}}$ has already been investigated in Maffeis and Phillips (2004), where it was called L_{io}. They showed that such a small fragment is Turing complete by providing an encoding of Random Access Machines. The encoding they presented enabled them to conclude that the existence of a terminating computation is an undecidable problem, while the decidability of reachability was raised as an open problem. Our decidability result provides a positive answer to this problem.

In order to prove the minimality of pMA$_{g!}^{-\texttt{open}}$, we have made use of (a slight adaptation of) Boneva and Talbot's undecidability result (Boneva and Talbot 2003). They proved that reachability is undecidable for the `open`-free fragment equipped with a structural congruence that is slightly different from the standard one (see the discussion in Section 3). Instead of getting decidability by imposing syntactical restrictions (as we do for pMA$_{g!}^{-\texttt{open}}$), they moved to a weaker version of the operational semantics. In particular, they showed that reachability becomes decidable when the structural congruence law $!P \equiv P \mid !P$ is replaced by the reduction axiom $!P \rightarrow P \mid !P$.

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

An additional contribution of this paper is contained in Section 5.1, where we introduced a new problem for Petri-nets, called *target marking reachability*, and showed how to reduce it to the reachability problem (thus proving it is decidable). Target marking reachability is a generalisation of the well-known reachability and coverability problems for Petri nets. This new problem, even though it was defined and analysed specifically to prove the decidability of target reachability for $\mathrm{pMA}_{g!}^{-\mathtt{open}}$, may also be of interest in other contexts. For instance, Delzanno *et al.* (2009) used the decidability of target marking reachability in Petri nets to prove the decidability of a reachability problem (called the *simple coverability problem*) that is of interest for the κ-calculus (Danos and Laneve 2004) – the κ-calculus is a model for systems biology based on graphs for the representation of the structure of bio-molecules, and graph rewriting for the formalisation of their evolution.

Delzanno and Montagna (2006) and Zavattaro (2008) applied the techniques presented in (preliminary versions of) this paper to carry out a form of reachability analysis in BioAmbients (Regev *et al.* 2004), which is a dialect of Mobile Ambients used for the modelling of biological systems. The main differences between the calculi considered in this paper and the fragment of BioAmbients considered in Delzanno and Montagna (2006) and Zavattaro (2008) are:

(i) Monotonicity is obtained in the current paper by removing the *open* capability, but in Delzanno and Montagna (2006) and Zavattaro (2008) it was obtained for BioAmbients by removing the *merge* capability used to fuse the boundaries of two sibling ambients.

(ii) Unlike the case for Mobile Ambients, where a single process can execute its capability to move its hosting ambient, in BioAmbients two distinct processes located at different ambients must synchronise to allow their hosting ambients to change the nesting structure.

(iii) Communication in BioAmbients also includes communication between processes in sibling ambients, but in Boxed Ambients communication is just local or parent–child.

(iv) BioAmbients also includes a choice operator (which is guarded on either communication actions or capabilities), but there is no choice operator in the calculi considered in this paper.

As future work, we plan to investigate the relationship between our reachability analysis results and model-checking techniques for the ambient logic. A first result along these lines can be achieved with a slight modification of Theorem 3.5, which shows the undecidability of reachability for $\mathrm{pMA}_{g!}$, which is the fragment obtained by limiting replication to guarded processes. Using this approach, we can prove the undecidability of name convergence (Gordon and Cardelli 2002) for $\mathrm{pMA}_{g!}$: a process P converges to a name n if $P \rightarrow^* n[Q_1] \mid Q_2$ for some Q_1 and Q_2. As discussed in Boneva and Talbot (2003), name convergence can be reduced to an instance of the model checking problem. This provides a proof of the undecidability of model checking for $\mathrm{pMA}_{g!}$.

Another area that deserves further investigation is a complexity analysis of the problems we have proved to be decidable in this paper. In particular, we leave as an open problem the definition of lower bounds for the *target marking reachability* problem that we have defined for P/T nets as a generalisation of the well-known reachability and coverability

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press

problems. For instance, it might be interesting to characterise lower bounds that are strictly greater than those that are already known for reachability and coverability.

Acknowledgements

We would like to thank Jean-Marc Talbot and Iain Phillips for their insightful comments on preliminary versions of this paper, and the anonymous referees for their helpful suggestions.

References

Boneva, I. and Talbot, J.-M. (2003) When Ambients Cannot be Opened. *Theoretical Computer Science* **333** 127–169.

Bugliesi, M., Castagna, G. and Crafa, S. (2004) Access Control for Mobile Agents: The Calculus of Boxed Ambients. *ACM Transactions on Programming Languages and Systems* **26** (1) 57–124.

Busi, N. (2000) Personal communication including the proof of Turing completeness of the `open`-free fragment of pure MA. The extended version of this proof appears in Busi and Zavattaro (2004).

Busi, N. and Zavattaro, G. (2004) On the Expressive Power of Movement and Restriction in Pure Mobile Ambients. *Theoretical Computer Science* **322** 477–515.

Busi, N. and Zavattaro, G. (2005a) Deciding Reachability in Mobile Ambients. In: Proc. of ESOP. *Springer-Verlag Lecture Notes in Computer Science* **3444** 248–262.

Busi, N. and Zavattaro, G. (2005b) Reachability Analysis in Boxed Ambients. In Proc. of ICTCS. *Springer-Verlag Lecture Notes in Computer Science* **3701** 143–159.

Cardelli, L., Ghelli, G. and Gordon, A. D. (2002) Types for the ambient calculus. *Information and Computation* **177** (2) 160–194.

Cardelli, L. and Gordon, A. D. (2000a) Mobile Ambients. *Theoretical Computer Science* **240** (1) 177–213.

Cardelli, L. and Gordon, A. D. (2000b) Anytime, anywhere: Modal logics for mobile ambients. In: *Proc. of POPL'00*, ACM Press 365–377.

Cardelli, L. and Gordon, A. D. (2005) Ambient Logic. Accepted for publication in *Mathematical Structures in Computer Science*.

Delzanno, G. and Montagna, R. (2006) On Reachability and Spatial Reachability in Fragments of BioAmbients, In: Proc. MeCBIC'06. *Electronic Notes in Theoretical Computer Science* **171** (2) 69–79.

Delzanno, G., Giusto, C. D., Gabbrielli, M., Laneve, C. and Zavattaro, G. (2009) The Kappa-Lattice: Decidability Boundaries for Qualitative Analysis in Biological Languages. In: Proc. of CMSB'09. *Springer-Verlag Lecture Notes in Computer Science* **5688** 158–172.

Danos, V. and Laneve, C. (2004) Formal molecular biology. *Theoretical Computer Science* **325** (1) 69–110.

Gordon, A. D. and Cardelli, L. (2002) Equational Properties of Mobile Ambients. *Mathematical Structures in Computer Science* **12** 1–38.

Karp, R. M. and Miller, R. E. (1969) Parallel Program Scemata. *Journal of Computer and System Sciences* **3** (2) 147–195.

Levi, F. and Sangiorgi, D. (2003) Mobile Safe Ambients. *ACM Transactions on Programming Languages and Systems* **25** (1) 1–69.

Maffeis, S. and Phillips, I. (2004) On the Computational Strength of Pure Mobile Ambients. *Theoretical Computer Science* **330** 501–551.

Mayr, E. W. (1981) Persistence of Vector Replacement Systems is Decidable. *Acta Informatica* **15** 309–318.

Merro, M. and Hennessy, M. (2002) Bisimulation congruences in safe ambients. In: *Proc. of POPL'02*, ACM Press 71–80.

Milner, R., Parrow, J. and Walker, D. (1992) A calculus of mobile processes. *Information and Computation* **100** 1–77.

Minsky, M. L. (1961) Recursive Unsolvability of Post's Problem of "Tag" and others Topics in the Theory of Turing Machines. *Annals of Math.* **74** :437–455.

Minsky, M. L. (1967) *Computation: finite and infinite machines*, Prentice-Hall.

Phillips, I. and Vigliotti, M. (2002) On reduction semantics for the push and pull ambient calculus. In: *Proc. of IFIP International Conference on Theoretical Computer Science (TCS'02)*, Kluwer 550–562.

Regev, A., Panina, E. M., Silverman, W., Cardelli, L. and Shapiro, E. Y. (2004) BioAmbients: an abstraction for biological compartments. *Theoretical Computer Science* **325** (1) 141–167.

Reisig, W. (1985) *Petri nets: An Introduction*, EATCS Monographs in Computer Science, Springer-Verlag.

Reutenauer, C. (1988) *The Mathematics of Petri Nets*, Masson.

Shepherdson, J. C. and Sturgis, J. E. (1963) Computability of recursive functions. *Journal of the ACM* **10** 217–255.

Zavattaro, G. (2008) Reachability Analysis in BioAmbients. In: Proc. MeCBIC'08. *Electronic Notes in Theoretical Computer Science* **227** 179–193.

https://doi.org/10.1017/S0960129509990181 Published online by Cambridge University Press