Deciding Reachability in Mobile Ambients

Nadia Busi and Gianluigi Zavattaro

Department of Computer Science, University of Bologna,
Mura A.Zamboni 7, 40127 Bologna (Italy)
{busi, zavattar}@cs.unibo.it

Abstract. Mobile Ambients has been proposed by Cardelli and Gordon as a foundational calculus for mobile computing. Since its introduction, the computational strength as well as the decidability of properties have been investigated for several fragments and variants of the standard calculus. We tackle the problem of reachability and we characterize a public (i.e., restriction free) fragment for which it is decidable. This fragment is obtained by removing the `open` capability and restricting the use of replication to guarded processes. Quite surprisingly, this fragment has been shown to be Turing complete by Maffeis and Phillips.

1 Introduction

Mobile Ambients (MA) [5] is a well known formalism exploited to describe distributed and mobile systems in terms of *ambients*. An ambient is a named collection of active processes and nested sub-ambients. In the pure (i.e., without communication) version of MA only three mobility primitives are used to permit ambient and process interaction: `in` and `out` for ambient movement, and `open` to dissolve ambient boundaries.

Following the tradition of process calculi, Mobile Ambients and its dialects have been equipped with a rich variety of formal tools useful for reasoning about and verifying properties of systems specified with these calculi (see, e.g., [9, 4, 6]). Another line of research regards the analysis of the expressiveness of these calculi in order to investigate the boundary between redundant and necessary features as well as the decidability of properties. For example, the Turing completeness of several variants and fragments of Mobile Ambients is investigated in [8], while the decidability of process termination (i.e. the existence of a finite computation) is investigated for fragments of the pure version of Mobile Ambients in [2].

Besides termination, an even more interesting property is process *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 is relevant to check whether the system

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

can be reached, where the intruder has succeeded.

M. Sagiv (Ed.): ESOP 2005, LNCS 3444, pp. 248–262, 2005.

The unique work, to the best of our knowledge, devoted to the investigation of reachability in Mobile Ambients is by Boneva and Talbot [1]. They prove that reachability is undecidable even in a minimal fragment of pure Mobile Ambients in which both the restriction operator (used to limit the scope of ambient names) and the `open` capability are removed.

Let us consider the above example of the *intruder* and the *firewall*. Traditional reachability consists of checking whether the target process is reachable for some instantiated processes P' and Q'. In general, one may be interested in concentrating only on the structure of the target process (i.e. the intruder is inside the firewall) abstracting away from the specific programs that run inside the ambients (i.e. abstracting away from P' and Q'). Exploiting classical reachability one should universally quantify on every possible processes P' and Q'.

To solve this problem we introduce *spatial reachability* permitting to specify a class of target processes. This class is characterized by a common structure of ambient nesting and a minimal number of processes that should be hosted inside those ambients. As an example of the use of spatial reachability consider the system

$$trojan[virus|P]|notebook[Q]$$

in which a *trojan* containing a *virus* program, and running program P, attacks a notebook running the program Q. One may be interested in checking whether the process

$$notebook[\ trojan[virus|P'] \mid Q']$$

can be reached for any possible P' and Q' not containing ambients. Observe that *virus* is a program for which it is necessary to check the actual presence inside the ambient *trojan* in the target process (a *trojan* that does not contain a *virus* is not dangerous).

We investigate the decidability of (spatial) reachability for fragments of the public, i.e. restriction free, version of the ambient calculus. We focus our analysis on calculi without restriction in order to concentrate on ambient nesting as the unique way for structuring processes. The relevance of restriction, as a mechanism for organizing processes inside name scopes, has been deeply investigated in the context of other process calculi such as the π–calculus [10].

The fragment that we characterize does not contain the `open` capability and limits the use of replication to guarded processes only (e.g., $!n[]$ is not a valid process for this fragment). This decidability result is proved by reducing reachability of processes to reachability in Petri nets (and spatial reachability to coverability). We prove the minimality of this fragment by showing that reachability becomes undecidable when relaxing at least one of the two restrictions imposed on the fragment. The undecidability for the `open`-free fragment has been proved by Boneva and Talbot [1]. For the fragment with guarded replication, we show how to reduce the halting problem for Random Access Machines [15] (a well known Turing powerful formalism) to the (spatial) reachability problem.

2 Pure Public Mobile Ambients

Pure public mobile ambients, that we denote with pMA, corresponds to the restriction-free fragment of the version of Mobile Ambients without communication defined by Cardelli and Gordon in [5].

Definition 1. – **pMA** – *Let Name, ranged over by n, m, ..., be a denumerable set of ambient names. The terms of pMA are defined by the following 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, while $\prod_{i \in 1 \ldots k} P_k$ denotes the parallel composition of the indexed processes P_i.

The term $\mathbf{0}$ represents the inactive process (and it is usually omitted). $M.P$ is a process guarded by one of the three mobility primitives (already 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 following. The term $n[P]$ denotes an ambient named n containing process P. A process may be also the parallel composition $P|P$ of two subprocesses. Finally, the replication operator $!P$ is used to put in parallel an unbounded number of instances of the process P.

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

Definition 2. – **Structural congruence** – *The structural congruence $\equiv$ is the smallest congruence relation satisfying:*

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

Definition 3. – **Reduction relation** – *The reduction relation is the smallest relation $\rightarrow$ satisfying the following axioms and rules:*

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

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

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

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

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

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

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 in checking, given two processes* P *and* Q*, whether* Q *is a derivative of* P*, i.e. checking if* $P \rightarrow^* Q$*.*

Axioms (1), (2) and (3) describe the semantics of the three primitives `in`, `out` and `open`, respectively. A process inside an ambient n can perform an $\texttt{in}\, m$ operation in presence of a sibling ambient m; if the operation is executed then 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, this results in moving the ambient n outside the ambient m. Finally, a process performing an $\texttt{open}\, n$ operation has the ability to remove the boundary of an ambient $n[Q]$ composed in parallel with it.

Rules (4) and (5) are the contextual rules that respectively indicate that a process can move also 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 the paper we consider three fragments of pMA; $\text{pMA}_{g!}$ and $\text{pMA}^{-\texttt{open}}$ for which we show that reachability is undecidable and $\text{pMA}_{g!}^{-\texttt{open}}$ for which it turns out to be decidable.

Definition 4.
$pMA_{g!}$ permits only guarded replication, i.e. it restricts the application of the replication operator to guarded processes:[1]

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

$pMA^{-\texttt{open}}$ removes the `open` *capability:*

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

$pMA_{g!}^{-\texttt{open}}$ combines the restrictions imposed by the previous fragments:

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

3 Deciding Reachability in $\text{pMA}_{g!}^{-\texttt{open}}$

In this Section we show that reachability is decidable in $\text{pMA}_{g!}^{-\texttt{open}}$. We reduce reachability on $\text{pMA}_{g!}^{-\texttt{open}}$ to reachability on Place/Transition Petri nets.

[1] The structural congruence for $\text{pMA}_{g!}$ is obtained by replacing the axiom for replication with $!M.P \equiv M.P \mid !M.P$, and the the congruence rule for the replication operator with the congruence rule for the operator of restricted replication.

As reachability is decidable on such class of Petri nets [13], we get the decidability result for reachability on $\text{pMA}_{g!}^{-\texttt{open}}$.

Another interesting property is spatial reachability. Given two processes, P and R, the spatial reachability problem roughly consists in checking if, starting from P, it is possible to reach a process R' "greater" than R, in the following sense: the ambients in R and R' have the same structure of ambient nesting, and the (sequential and replicated) active subprocesses inside an R ambient are a subset of the subprocesses inside the corresponding ambient in R'. The Petri net constructed for the solution of the reachability problem can be exploited to reduce the spatial reachability problem for $\text{pMA}_{g!}^{-\texttt{open}}$ processes to the coverability problem for Petri nets, which is a decidable problem [14].

We start recalling Place/Transition nets with unweigthed flow arcs (see, e.g., [14]).

Definition 5. *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 set of all finite multisets over S, denoted by $\mathcal{M}_{fin}(S)$, is ranged over by m. We write $m \subseteq m'$ if $m(s) \leq m'(s)$ for all $s \in S$. With $\oplus$ and $\setminus$ we denote* multiset union *and* multiset difference*, respectively.*

Definition 6. *A P/T net is a pair (S, T) where S is the set of* places *and $T \subseteq \mathcal{M}_{fin}(S) \times \mathcal{M}_{fin}(S)$ 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 $c \to p$. A transition $t = (c, p)$ is enabled *at m if $c \subseteq m$. The execution of a transition t enabled at m produces the marking $m' = (m \setminus c) \oplus p$. This is written as $m \xrightarrow{t} m'$ or simply $m \to m'$ when the transition t is not relevant.*

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

We say that m' covers *m if $m \subseteq m'$.*

Definition 7. *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 coverability problem for marking m consists of checking if there exists m' such that $m_0 \to^* m'$ and m' covers m.

3.1 Reducing Reachability on Processes to Reachability on Petri Nets

Now we show that reachability on processes can be reduced to reachability on Petri nets; by decidability of reachability on Petri nets, we get the following:

Theorem 1. *Let P, R be $pMA_{g!}^{-\texttt{open}}$ processes. The reachability problem $P \to^* R$ is decidable.*

Given two processes P and R, we outline construction of a (finite) Petri system $Sys_{P,R}$ satisfying the following property: the check of $P \rightarrow^* R$ is equivalent to check reachability of a finite set of markings on $Sys_{P,R}$. Because of the lack of space, the technical details concerning the construction of the net, as well as the auxiliary results needed to prove Theorem 1 are omitted; they can be found in [3].

The intuition behind this result relies on a monotonicity property of $\text{pMA}_{g!}^{-\texttt{open}}$: because of the absence of the `open` capability, the number of "active" ambients in a process (i.e., ambients that are not guarded by any 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 (while this is not the case in , e.g., the pMA process $!n[0]$). Thanks to the property explained above, in order to check if R is reachable from P it is sufficient to take into account a subset of the derivatives of P: namely, the P-derivatives whose number of active ambients is not greater than the number of active ambients in R.

Unfortunately, this subset of P-derivatives is, in general, not finite, as the processes inside an ambient can grow unlimitedly. Consider, e.g., the process $P = m[!\texttt{in}\, n.\texttt{out}\, n.Q] \mid n[]$: it is easy to see that, for any k, $m[\prod_k Q \mid !\texttt{in}\, n.\texttt{out}\, n.Q] \mid n[]$ is a derivative of P.

On the other hand, we note that the set of *sequential* and *replicated* terms that can occur as subprocesses of (the derivatives of) a process P (namely, the subterms of kind $M.P$ or $!M.P$) is finite. The idea is to borrow a technique used to map (the public fragment of) a process algebra on Petri nets. A process P is decomposed in the (finite) multiset of its sequential subprocesses that appear at top-level (i.e., occur unguarded in P); this multiset is then considered as 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 $\text{pMA}_{g!}^{-\texttt{open}}$ processes to reachability of a finite set of markings in a Place/Transition Petri net, which is a decidable problem. However, differently from what happens in process algebras, where processes can be faithfully represented by a multiset of subprocesses, $\text{pMA}_{g!}^{-\texttt{open}}$ processes have a tree-like structure that hardly fits in a flat model such as a multiset.

The solution is to consider $\text{pMA}_{g!}^{-\texttt{open}}$ processes as composed of two kinds of components; the tree-like structure of ambients and the family of multisets of sequential subterms contained in each ambient. As an example, consider the process

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

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

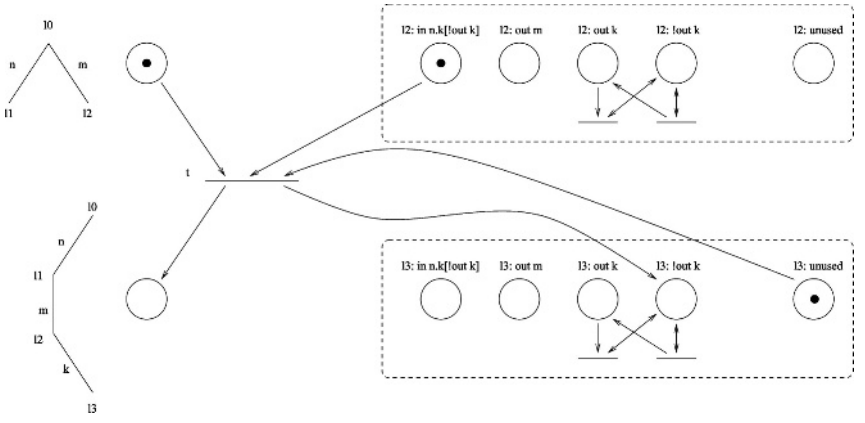

Fig. 1. A portion of the net corresponding to process $n[\texttt{out}\, m] \mid m[\texttt{in}\, n.k[!\texttt{out}\, k]]$

The Petri net we construct is composed of the following two parts: the first part is basically a finite state automaton, where the marked place represents the current tree-like structure of the process; the second part is a set of identical subnets: the marking of each subnet represents the multiset associated to a particular node of the tree. To keep the correspondence between the nodes of the tree and the multiset associated to that node, we make use of labels. A distinct label is associated to each subnet; this label will be used in the tree-like structure to label the node whose contents (i.e., 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 of all, the set of ambient names in a process is finite. Moreover, to verify reachability we need to take into account only 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 (at most one for each active ambient). In general, the number of active ambients grows during the computation; hence, we need a mechanism to remember which subnets are currently in use and which ones are not used. When a new ambient is created, a correspondence between the node representing such a new 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 aim, each subnet is equipped with a place called *unused*, that contains a token as long as the subnet does not correspond to any node in the tree-like structure.

For example, consider the process $n[\texttt{out}\, m] \mid m[\texttt{in}\, n.k[!\texttt{out}\, k]]$. The relevant part of the net is depicted in Figure 1: a subset of the places, representing the tree-like structure, is depicted in the left-hand part of the figure, while the subnets are depicted in the right-hand part. We only report the subnets labelled

with l_2 and l_3, and omit the two subnets labelled with l_0 (with empty marking) and with l_1 (whose marking consists of a token in place $l_1 : \mathtt{out}\, m$). The computation step $n[\mathtt{out}\, m] \mid m[\mathtt{in}\, n.k[!\mathtt{out}\, k]] \rightarrow n[\mathtt{out}\, m \mid m[k[!\mathtt{out}\, k]]]$ corresponds to the firing of transition t in the net.

A last remark is concerned with structural congruence: because of the structural congruence rule (6), the reachability of a process R actually correspond to decide if it is possible to reach a process that is structurally congruent to R. As we are reducing the reachability in $\mathrm{pMA}_{g!}^{-\mathtt{open}}$ to marking reachability in Petri nets, it is necessary that the set of markings, corresponding to the set of processes structurally congruent to R, is finite. We concentrate on the markings of the subnets. The top-level applications of the monoidal laws for parallel composition are automatically dealt with, as processes that are structurally congruent because of such laws are mapped on the same marking. Unfortunately, the application of the replication law permits to produce an infinite set of markings corresponding to structurally congruent processes. Take, e.g., $!\mathtt{in}\, n.P \equiv \mathtt{in}\, n.P \mid !\mathtt{in}\, n.P \equiv \mathtt{in}\, n.P \mid \mathtt{in}\, n.P \mid !\mathtt{in}\, n.P \equiv \ldots$ and the corresponding set of markings $\{!\mathtt{in}\, n.P\}, \{\mathtt{in}\, n.P, !\mathtt{in}\, n.P\}, \{\mathtt{in}\, n.P, \mathtt{in}\, n.P, !\mathtt{in}\, n.P\} \ldots$.

To solve this problem, we make use of the following two techniques.

The top-level application of the law for replication can be easily dealt with by adding 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 permitting to spawn a new copy of a replicated process and to absorbe a process that also appears in a replicated form in the marking. An instance of such transitions is depicted in the subnet $l2$ of Figure 1.

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

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

To this aim, we introduce canonical representations of the equivalence classes of structural congruence, roughly consisting in nested multisets where the presence of a replicated version of a sequential term forbids the presence of any occurrence of the nonreplicated 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 $!\mathtt{in}\, n.(!\mathtt{out}\, m.\mathbf{0}) \mid n[\mathtt{in}\, n.\mathbf{0}]$.

Now we are ready to describe the net that will be used to decide reachability of a process R starting from a process P.

The set of places of the net is constructed as follows. The part of the net representing the tree-like structure contains a place for each tree of size not 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 named "unused", that remains filled as long as the subnet does not correspond to any node in the tree-like structure. Moreover, we associate a distinct label to 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 set permits to model the execution of the `in` and `out` capabilities, while the second set is used to cope with the structural congruence rule for replication.

We concentrate on the first set of transitions. A capability, say, e.g., $\texttt{in}\, n$, 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 right 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 exist 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.

Besides deciding reachability, the net system described above can be used to check the weaker property of spatial reachability.

3.2 Spatial Reachability

The spatial reachability problem for processes P and R roughly consists in checking if, starting from P, it is possible to reach a process R' "greater than" R, in the following sense:

- R' has the same spatial ambient structure of R, and
- the sequential and replicated active subprocesses contained in each ambient of R are also present in the corresponding ambient of R'.

The $\preceq_s$ relation is a formalization of the "greater than" concept:

Definition 8. *Let P and Q be $pMA^{-\texttt{open}}_{g!}$ processes.*
$P \preceq_s Q$ iff

- *either $Q \equiv P \mid \prod_i M_i.P_i \mid \prod_j !M'_j.P'_j$,*
- *or $P \equiv P_1 \mid n[P_2]$, $Q \equiv Q_1 \mid n[Q_2]$ and $P_i \preceq_s Q_i$ for $i = 1, 2$*

The spatial reachability problem for processes P and R consists in checking if there exists R' such that $P \rightarrow^* R'$ and $R \preceq_s R'$.

The mapping of processes to Petri nets markings outlined above satisfies the following property: if $P_1 \preceq_s P_2$ then the marking corresponding to P_2 covers the marking corresponding to P_1. Hence, the Petri net constructed in the previous section permits to reduce the spatial reachability problem for processes P and R to a coverability problem. As coverability is decidable in P/T nets, we obtain the following:

Theorem 2. *Let P, R be $pMA_{g!}^{-\mathtt{open}}$ processes. The spatial reachability problem for P and R is decidable.*

4 Undecidability Results

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

As far as $\mathrm{pMA}^{-\mathtt{open}}$ is concerned, we resort to an equivalent result proved by Boneva and Talbot in [1] for a slightly different calculus. That calculus, proposed in [6, 7], differs from $\mathrm{pMA}^{-\mathtt{open}}$ only for three extra rules in the definition of the structural congruence relation: $\mathbf{0} \equiv \mathbf{0}$, $!!P \equiv !P$, $!(P \mid Q) \equiv !P \mid !Q$. These rules are added by Boneva and Talbot to guarantee that the congruence relation is confluent, thus decidable.

The undecidability of reachability is proved by Boneva and Talbot showing how to encode two-counters machines [11], a well known Turing powerful formalism. The encoding preserves the *one-step property*: if the two-counters machine $2CM$ moves in one step to $2CM'$ then $[\![2CM]\!] \rightarrow^* [\![2CM']\!]$, where $[\![\,]\!]$ is the considered encoding. Even if the calculus in [1] is slightly different from $\mathrm{pMA}^{-\mathtt{open}}$, the encoding of two-counters presented in that paper applies also to our calculus; this because the encoding does not apply the replication operator to the empty process, to replicated processes and to parallel composition of processes (i.e. the cases in which the three extra structural congruence rules come into play, respectively).

As far as $\mathrm{pMA}_{g!}$ is concerned, we present a modeling of Random Access Machines (RAMs) [15], a formalism similar to two-counters machines. The encoding that we present is an enhancement of the RAM modeling in [2]: the main novelties are concerned with a more restricted use of replication and a reshaping of the garbage processes.

4.1 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 $r_1, \ldots, r_n$, that can hold arbitrary large natural numbers, and by a sequence of indexed instructions $(1 : I_1), \ldots, (m : I_m)$. In [12] it is shown that the following two instructions are sufficient to model every recursive function:

- $(i : \mathit{Succ}(r_j))$: adds 1 to the contents of register r_j and goes to the next instruction;
- $(i : \mathit{DecJump}(r_j, s))$: if the contents of the register r_j is not zero, then decreases it by 1 and goes to the next instruction, otherwise jumps to the instruction s.

The computation starts from the first instruction and it 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. It is not restrictive to assume that the instruction number

reached at the end of the computation is always $m+1$, and to assume that the computation starts and terminates with all the registers empty.

4.2 Modelling RAMs in pMA$_{g!}$

We model instructions and registers independently. As far as the instructions and the program counter are concerned, we model the program counter i with an ambient $pc_i[]$. Each instruction I_i is represented with a replicated process guarded by the capability $\texttt{open}\, pc_i$ able to open the corresponding program counter ambient $pc_i[]$. The processes modeling the instructions are replicated because each instruction could be performed an unbounded amount of times during the computation.

The key idea underlying the modeling 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, while on register decrement we dissolve the outer s_j ambient boundary. In this way, for instance, the resister r_j with content 2 is modeled by the nesting $s_j[s_j[z_j[]]]$ (plus other processes hosted in these ambients that are discussed in the following).

Definition 9. *Given the RAM R with instructions $(1 : I_1), \ldots, (m : I_m)$ and registers $r_1, \ldots, r_n$ we define $[\![R]\!]$ as the following process*

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

where C_i (modeling the $i-th$ instruction), R_j^0 (modeling the empty register r_j) and GC (the garbage collector which is started at the end of the computation) are shorthand notations defined in the following.

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

The register r_j with content l is represented by the process R_j^l defined inductively as follows

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

where REG_j is a shorthand notation defined as follows

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

Also in this case, the process $\texttt{open}\, gc$ is used 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 having discussed the encoding for the instructions.

Before formalizing the modeling of the instructions we anticipate that the names inc_j, $zero_j$ and dec_j are used to model requests for increment, test for zero and decrement of register r_j, respectively; the names $acki_j$, $ackz_j$ and $ackd_j$ models the corresponding acknowledgements produced by the registers to notify that a request has been managed.

The instructions are modeled as follows. If the i-th instruction is $Succ(r_j)$, its encoding is

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

This modeling is based on two processes. The first one is the ambient $increq_j$ that represents a request for increment of the register r_j. The second process blocks waiting for an acknowledgement that will be produced after the actual increment of the register; when the acknowledgement is received, this process increments the program counter 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 $increq_j$ becoming in 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. After, the acknowledgement is produced in terms of an ambient named $acki_j$ that moves outside the register boundary.

If the i-th instruction is $DecJump(r_j, s)$ the encoding is as follows

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

where

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

This modeling is based on three processes. The first process is an ambient named $zero_j$ which represents a request for a test for zero of the register r_j; the second process is an ambient named dec_j representing a request for decrement of the register r_j; the third process is an ambient named dj_j which is in charge to manage the acknowledgement produced by the register r_j. The acknowledgement indicates whether the decrement, or the test for zero request, has succeeded.

Let us 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. In case of $ackz_j$, 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 necessary to remove the decrement request that has failed. In case of $ackd_j$, the process also dissolves one of the boundaries s_j, in order to actually decrement the register. In both cases, the program counter is finally updated by either jumping to instruction s, or by activating the next instruction $i+1$, respectively.

This way of modeling RAMs does not guarantee the one-step preservation property because of the production of garbage, that is processes that are no more involved in the subsequent computation. More precisely, the following garbage is produced:

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

Clearly, the exact shape of the garbage at the end of the modeling of the RAM computation is unpredictable because it depends on the exact number of instructions that are executed. Nevertheless, we use the garbage collector process GC, activated on program termination, in order to reshape the garbage in a predefined format.

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

We are now in place for defining the garbage collector process formally

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

The undecidability of reachability and spatial reachability is a trivial corollary of the following theorem. It is worth noting that in the statement of the theorem the register r_j (which is assumed to be empty at the end of the computation) is represented by a process which is the same as R_j^0 with the difference that the process $\texttt{open}\, gc$, initially available in the ambient z_j (see the definition of R_j^0), is replaced by the two processes $!\texttt{open}\, increq \mid !!\texttt{in}\, s_j$ left by the garbage collector.

Theorem 3. *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*

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

is reachable from the process $[\![R]\!]$ (as defined in Definition 9).

Moreover, we have that the RAM R terminates if and only if the process

$$pc_{m+1}[] \mid \prod_{j\in 1\ldots n} z_j[] \mid garbage[]$$

is spatially reachable from the process $[\![R]\!]$.

5 Conclusion

We have discussed the decidability of reachability in Mobile Ambients. We have characterized a fragment of the pure and public Mobile Ambients, namely the $\texttt{open}$-free fragment with guarded replication, for which reachability is decidable. We call this fragment $\mathrm{pMA}_{g!}^{-\texttt{open}}$. Our decidability result also holds for a variant of reachability, called spatial reachability, that permits to specify a class of target processes characterized by a common structure of ambient nesting.

The fragment $\mathrm{pMA}_{g!}^{-\texttt{open}}$ has been already investigated by Maffeis and Phillips in [8] (called L_{io} in that paper). They show that such a small fragment is indeed Turing complete, by providing an encoding of RAMs. The encoding they present permits to conclude that the existence of a terminating computation is an undecidable problem, while the decidability of reachability is raised as an open problem. Our decidability result provides a positive answer to this problem.

In order to prove the minimality of $\mathrm{pMA}_{g!}^{-\texttt{open}}$ we make use of (a slight adaptation of) the undecidability result by Boneva and Talbot [1]. They prove that reachability is undecidable for the $\texttt{open}$-free fragment, equipped with a structural congruence slightly different from the standard one (see the discussion in Section 4). Instead of getting decidability by imposing syntactical restrictions

(as we do for $\mathrm{pMA}_{g!}^{-\mathtt{open}}$), they move to a weaker version of the operational semantics. In particular, they show that reachability becomes decidable when the structural congruence law $!P \equiv P \mid !P$ is replaced by the reduction axiom $!P \rightarrow P \mid !P$.

Acknowledgements. We thank Jean-Marc Talbot and Iain Phillips for their insightful comments on a preliminary version of this paper.

References

1. I. Boneva and J.-M. Talbot. When Ambients Cannot be Opened. In *Proc. FOSSACS'03*, volume 2620 of *Lecture Notes in Computer Science*, pages 169-184. Springer-Verlag, Berlin, 2003. Full version to appear in Theoretical Computer Science, Elsevier.
2. N. Busi and G. Zavattaro. On the Expressive Power of Movement and Restriction in Pure Mobile Ambients. *in Theoretical Computer Science*, 322:477–515, Elsevier, 2004.
3. N. Busi and G. Zavattaro, Deciding Reachability in Mobile Ambients - Extended version. Available at `http://www.cs.unibo.it/ busi/papersaMA05.pdf/`
4. L. Cardelli, G. Ghelli, and A.D. Gordon. Types for the ambient calculus Information and Computation, 177(2):160–194, 2002.
5. L. Cardelli and A.D. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.
6. L. Cardelli and A.D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In Proc. of POPL'00, pages 365–377. ACM Press, 2000.
7. L. Cardelli and A.D. Gordon. Ambient Logic. Mathematical Structures in Computer Science, to appear.
8. S. Maffeis and I. Phillips. On the Computational Strength of Pure Mobile Ambients. To appear in Theoretical Computer Science, Elsevier, 2004.
9. M. Merro and M. Hennessy. Bisimulation congruences in safe ambients. In Proc. of POPL'02, pages 71–80, ACM Press, 2002.
10. R. Milner, J. Parrow, D. Walker. A calculus of mobile processes. *Journal of Information and Computation*, 100:1–77. Academic Press, 1992.
11. M.L. Minsky. *Recursive Unsolvability of Post's Problem of "Tag" and others Topics in the Theory of Turing Machines. Annals of Math.*, 74:437–455, 1961.
12. M.L. Minsky. *Computation: finite and infinite machines.* Prentice-Hall, 1967.
13. C. Reutenauer. *The Mathematics of Petri Nets.* Masson, 1988.
14. W. Reisig. *Petri nets: An Introduction.* EATCS Monographs in Computer Science, Springer, 1985.
15. J.C. Shepherdson and J.E. Sturgis. Computability of recursive functions. *Journal of the ACM*, 10:217–255, 1963.