The Complexity of Model Checking Mobile Ambients

Witold Charatonik[1,3], Silvano Dal Zilio[2], Andrew D. Gordon[2], Supratik Mukhopadhyay[1], and Jean-Marc Talbot[1,4]

[1] Max-Planck-Institut für Informatik, Germany.
[2] Microsoft Research, United Kingdom.
[3] University of Wrocław, Poland.
[4] Laboratoire d'Informatique Fondamentale de Lille, France.

Abstract. We settle the complexity bounds of the model checking problem for the replication-free ambient calculus with public names against the ambient logic without parallel adjunct. We show that the problem is PSPACE-complete. For the complexity upper-bound, we devise a new representation of processes that remains of polynomial size during process execution; this allows us to keep the model checking procedure in polynomial space. Moreover, we prove PSPACE-hardness of the problem for several quite simple fragments of the calculus and the logic; this suggests that there are no interesting fragments with polynomial-time model checking algorithms.

1 Introduction

The ambient calculus [1–3] is a formalism for describing the mobility of both software and hardware. An ambient is a named cluster of running processes and nested sub-ambients. Each computation state has a spatial structure, the tree induced by the nesting of ambients. Mobility is abstractly represented by re-arrangement of this tree: an ambient may move inside or outside other ambients.

The ambient logic [4] is a modal logic designed to specify properties of distributed and mobile computations programmed in the ambient calculus. As well as standard temporal modalities for describing the evolution of ambient processes, the logic includes novel spatial modalities for describing the tree structure of ambient processes. Serendipitously, these spatial modalities can also usefully describe the tree structure of semistructured databases [5]. Other work on the ambient logic includes a study of the process equivalence induced by the satisfaction relation [9] and a study of the logic extended with constructs for describing private names [6].

The *model checking* problem is to decide whether a given object (in our case, an ambient process) satisfies (that is, is a model of) a given formula. Cardelli and Gordon [4] show decidability of the model checking problem for a finite-state fragment of the ambient calculus against the fragment of the ambient logic without their parallel adjunct modality. This finite-state ambient calculus omits the constructs for unbounded replication and dynamic name generation of the full calculus. Cardelli and Gordon give no complexity analysis for their algorithm. Still, given the various possible applications of the logic, it is of interest to analyse the complexity of model checking mobile ambients. In fact, a naive analysis of the algorithm of Cardelli and Gordon gives only a doubly exponential bound on its use of time and space. A more sophisticated analysis based on results in this paper shows that their algorithm works in single-exponential time on single-exponential space.

In this paper we settle the complexity bounds of the model checking problem for the finite-state ambient calculus (that is, the full calculus apart from replication and name generation) against the logic without parallel adjunct. Our main result (embodied in Theorems 3.1 and 4.1) is that the problem is PSPACE-complete. Hence, this situates model checking the ambient logic in the same complexity class as model checking concurrent programs against CTL and CTL* [8].

As we discuss in Sect. 2, there are two reasons that the algorithm in [4] uses exponential space. One of them is that a process may grow exponentially during its execution; the other is that there may be exponentially many processes reachable from a given one. In Sect. 3, we present a new model checking algorithm that avoids these problems as follows. We avoid the first problem by devising a new representation of processes using a form of closure. The main feature of this representation is that substitutions that occur when communications take place within an ambient are not applied directly, but are kept explicit. These explicit substitutions prevent the representation blowing up exponentially in the size of the original process. The idea of using closures comes from DAG representations used in unification for avoiding exponential blow-up. A sequential substitution that we use here can be seen as a DAG representation of the substitution. To avoid the second problem, we first devise a non-deterministic algorithm for testing reachability that does not have to store all the reachable processes, but instead tests it on-the-fly, and then remove nondeterminism using Savitch's theorem [10]. Hence we prove Theorem 3.1, that the model checking problem is solvable in PSPACE.

We show this upper bound to be tight in Sect. 4; Theorem 4.1 asserts that the model checking problem is PSPACE-hard. Actually, we give PSPACE-hardness results for various fragments of the logic and of the calculus. For instance, by Theorem 4.2, even for a calculus of purely mobile ambients (that is, a calculus without communication or the capability to dissolve ambients) and the logic without quantifiers, the problem is PSPACE-hard. Moreover, by Theorem 4.3, for a calculus of purely communicative ambients (that is, a calculus without the capabilities to move or to dissolve ambients) and the logic without quantifiers, the problem is also PSPACE-hard. Often in the study of model checking fixing the model or the formula makes the problem easier. Here this is not the case. Even if we fix the process to be the constant $\mathbf{0}$, the model checking problem remains PSPACE-hard. Although we do not prove PSPACE-hardness for fixed arbitrary formulas, our result is not much weaker: Theorem 4.4 asserts that for any level of the polynomial-time hierarchy we can find a fixed formula such that the model checking problem is hard for that level.

A more complete presentation of the calculus and the logics, together with examples and omitted proofs, may be found in an extended version of this paper [7].

2 Review of the Ambient Calculus and Logic

We present a finite-state ambient calculus (that is, the full calculus [1] apart from replication and name generation) and the ambient logic without parallel adjunct. This is the same calculus and logic for which Cardelli and Gordon present a model checking algorithm [4].

2.1 The Ambient Calculus with Public Names

The following table describes the expressions and processes of our calculus.

Expressions and Processes:

$M, N ::=$	expressions	$P, Q, R ::=$	processes
n	name	$\mathbf{0}$	inactivity
$in\ M$	can enter M	$P \mid Q$	composition
$out\ M$	can exit M	$M[P]$	ambient
$open\ M$	can open M	$M.P$	action
ϵ	null	$(n).P$	input
$M.M'$	path	$\langle M \rangle$	output

A name n is said to be *bound* in a process P if it occurs within an input prefix (n). A name is said to be *free* in a process P if there is an occurrence of n outside the scope of any input (n). We write $bn(P)$ and $fn(P)$ for respectively the set of bound names and the set of free names in P. We say two processes are α-equivalent if they are identical apart from the choice of bound names. We write $M\{n{\leftarrow}N\}$ and $P\{n{\leftarrow}N\}$ for the outcomes of capture-avoiding substitutions of the expression N for the name n in the expression M and the process P, respectively.
The semantics of the calculus is given by the relations $P \equiv Q$ and $P \to Q$. The *reduction* relation, $P \to Q$, defines the evolution of processes over time. The *structural congruence* relation, $P \equiv Q$, is an auxiliary relation used in the definition of reduction. When we define the satisfaction relation of the modal logic in the next section, we use an auxiliary relation, the *sublocation* relation, $P \downarrow Q$, which holds when Q is the whole interior of a top-level ambient in P. We write $\to^*$ and $\downarrow^*$ for the reflexive and transitive closure of $\to$ and $\downarrow$, respectively.

Structural Congruence $P \equiv Q$:

P, Q are α-equivalent $\Rightarrow P \equiv Q$	(Str Refl)		
$Q \equiv P \Rightarrow P \equiv Q$	(Str Symm)		
$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$	(Str Trans)		
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	(Str Par)	$P \equiv Q \Rightarrow M[P] \equiv M[Q]$	(Str Amb)
$P \equiv Q \Rightarrow M.P \equiv M.Q$	(Str Action)	$P \equiv Q \Rightarrow (n).P \equiv (n).Q$	(Str Input)
$P \mid Q \equiv Q \mid P$	(Str Par Comm)	$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(Str Par Assoc)
$P \mid \mathbf{0} \equiv P$	(Str Zero Par)		
$\epsilon.P \equiv P$	(Str ϵ)	$(M.M').P \equiv M.M'.P$	(Str .)

Reduction $P \to Q$ and Sublocation $P \downarrow Q$:

$n[in\ m.P \mid Q] \mid m[R] \to m[n[P \mid Q] \mid R]$	(Red In)
$m[n[out\ m.P \mid Q] \mid R] \to n[P \mid Q] \mid m[R]$	(Red Out)
$open\ n.P \mid n[Q] \to P \mid Q$	(Red Open)
$\langle M \rangle \mid (n).P \to P\{n{\leftarrow}M\}$	(Red I/O)
$P \to Q \Rightarrow P \mid R \to Q \mid R$	(Red Par)
$P \to Q \Rightarrow n[P] \to n[Q]$	(Red Amb)
$P' \equiv P, P \to Q, Q \equiv Q' \Rightarrow P' \to Q'$	(Red $\equiv$)
$P \equiv n[Q] \mid P' \Rightarrow P \downarrow Q$	(Loc)

The following example shows that the size of reachable processes may be exponential, and that there may be a reduction path of exponential length. The algorithm given in [4] may use exponential space to check properties of this example.

Consider the family of processes $(P_k)_{k\geq 0}$, recursively defined by the equations $P_0 = (n).(p[n] \mid q[\mathbf{0}])$ and $P_{k+1} = (n_{k+1}).(\langle n_{k+1}.n_{k+1}\rangle \mid P_k)$. Intuitively, the process P_{k+1} inputs a capability, calls it n_{k+1}, doubles it, and outputs the result to the process P_k. We have the following, where $M^1 = M$ and $M^{k+1} = M.M^k$.

$$\begin{array}{ll}
\langle in\ q.out\ q\rangle \mid P_0 \to^1 & p[in\ q.out\ q] \mid q[\mathbf{0}] \\
\langle in\ q.out\ q\rangle \mid P_1 \to^2 & p[(in\ q.out\ q)^2] \mid q[\mathbf{0}] \\
\langle in\ q.out\ q\rangle \mid P_k \to^{k+1} & p[(in\ q.out\ q)^{2^k}] \mid q[\mathbf{0}]
\end{array}$$

Since $(in\ q.out\ q)^{2^k}$ is a sequence of 2^k copies of $in\ q.out\ q$, the process $\langle in\ q.out\ q\rangle \mid P_k$ reduces in $(k+1)+2^{k+1}$ steps to $p[\mathbf{0}] \mid q[\mathbf{0}]$.

This example points out two facts. First, using a simple representation of processes (such as the one proposed in [4]), it may be that the size of a process considered during model checking grows exponentially bigger than the size of the initial process. Second, during the model checking procedure, there may be an exponential number of reachable processes to consider. Therefore, a direct implementation of the algorithm proposed in [4] may use space exponential in the size of the input process. In this paper, using a new representation of processes, we show that the model checking problem $P \models \mathcal{A}$ can be solved using only polynomial space in the sizes of P and $\mathcal{A}$, in spite of this example.

2.2 The Logic (for Public Names)

We describe the formulas and satisfaction relation of the logic.

Logical Formulas:

η	a name n or a variable x		
$\mathcal{A}, \mathcal{B} ::=$	formula		
$\mathbf{T}$	true	$\neg\mathcal{A}$	negation
$\mathcal{A} \vee \mathcal{B}$	disjunction	$\exists x.\mathcal{A}$	existential quantification
$\mathbf{0}$	void	$\mathcal{A} \mid \mathcal{B}$	composition match
$\eta[\mathcal{A}]$	ambient match	$\mathcal{A}@\eta$	location adjunct
$\diamond\!\!\!\cdot\,\mathcal{A}$	somewhere modality	$\Diamond\mathcal{A}$	sometime modality

We assume that names and variables belong to two disjoint vocabularies. We write $\mathcal{A}\{x{\leftarrow}m\}$ for the outcome of substituting each free occurrence of the variable x in the formula $\mathcal{A}$ with the name m. We say a formula $\mathcal{A}$ is closed if and only if it has no free variables (though it may contain free names).

Intuitively, we interpret closed formulas as follows. The formulas $\mathbf{T}$, $\neg\mathcal{A}$, and $\mathcal{A} \vee \mathcal{B}$ embed propositional logic. The formulas $\mathbf{0}$, $\eta[\mathcal{A}]$, and $\mathcal{A} \mid \mathcal{B}$ are spatial modalities. A process satisfies $\mathbf{0}$ if it is structurally congruent to the empty process. It satisfies $n[\mathcal{A}]$ if it is structurally congruent to an ambient $n[P]$ where P satisfies $\mathcal{A}$. A process P satisfies $\mathcal{A} \mid \mathcal{B}$ if it can be decomposed into two subprocesses, $P \equiv Q \mid R$, where Q satisfies $\mathcal{A}$, and R satisfies $\mathcal{B}$. The formula $\exists x.\mathcal{A}$ is an existential quantification over names. The formulas $\Diamond\mathcal{A}$ (sometime) and $\diamondsuit\mathcal{A}$ (somewhere) quantify over time and space, respectively. A process satisfies $\Diamond\mathcal{A}$ if it has a temporal successor, that is, a process into which it evolves, that satisfies $\mathcal{A}$. A process satisfies $\diamondsuit\mathcal{A}$ if it has a spatial successor, that is, a sublocation, that satisfies $\mathcal{A}$. Finally, a process P satisfies the formula $\mathcal{A}@n$ if the ambient $n[P]$ satisfies $\mathcal{A}$.

The satisfaction relation $P \models \mathcal{A}$ provides the semantics of our logic.

Satisfaction $P \models \mathcal{A}$ (for $\mathcal{A}$ Closed):

$P \models \mathbf{T}$	$P \models \neg\mathcal{A} \triangleq \neg(P \models \mathcal{A})$
$P \models \mathcal{A} \vee \mathcal{B} \triangleq P \models \mathcal{A} \vee P \models \mathcal{B}$	$P \models \mathbf{0} \triangleq P \equiv \mathbf{0}$
$P \models n[\mathcal{A}] \triangleq \exists P'.P \models n[P'] \wedge P' \models \mathcal{A}$	
$P \models \mathcal{A} \mid \mathcal{B} \triangleq \exists P', P''.P \equiv P' \mid P'' \wedge P' \models \mathcal{A} \wedge P'' \models \mathcal{B}$	
$P \models \exists x.\mathcal{A} \triangleq \exists m.P \models \mathcal{A}\{x{\leftarrow}m\}$	$P \models \Diamond\mathcal{A} \triangleq \exists P'.P \rightarrow^* P' \wedge P' \models \mathcal{A}$
$P \models \diamondsuit\mathcal{A} \triangleq \exists P'.P \downarrow^* P' \wedge P' \models \mathcal{A}$	$P \models \mathcal{A}@n \triangleq n[P] \models \mathcal{A}$

We use $\Box\mathcal{A}$ (everytime modality), ❑$\mathcal{A}$ (everywhere modality) and $\forall x.\mathcal{A}$ (universal quantification) as abbreviations for $\neg(\Diamond\neg\mathcal{A})$, $\neg(\diamondsuit\neg\mathcal{A})$ and $\neg(\exists x.\neg\mathcal{A})$, respectively.

3 A Model Checking Algorithm

We show that the model checking problem can be decided in polynomial space by devising a new representation of processes (Sect. 3.1) that remains polynomial in the size of the initial process (Sect. 3.2). In Sect. 3.3 we present a new model checking algorithm based on this representation.

Since the reduction relation is defined up to α-equivalence, we may assume for the purposes of computing reachable processes that the free and bound names of every ambient process are distinct, and moreover that the bound names are pairwise distinct.

3.1 A Polynomial-Space Representation

We give in this section a new representation for ambient processes based on *normal closures*. (It is different from the *normal form* of processes introduced in [4].) We also present basic operations on closures and prove that closures indeed simulate the processes they represent.

Annotated Processes, Substitutions, Closures:

$\tilde{P} ::= \prod_{i \in I} \pi_i$	annotated process, multiset of primes, I finite indexing set		
$\pi ::=$	prime		
$M[\tilde{P}]$	ambient	$M(o).\tilde{P}$	action, with offset $o \geq 0$
$(n).\tilde{P}$	input	$\langle M \rangle$	output
$\sigma ::= \{n_1 \leftarrow M_1\} \cdots \{n_k \leftarrow M_k\}$	sequential substitution, $k \geq 0$		
$\langle \tilde{P}; \sigma \rangle$	closure		

In a sequential substitution $\{n_1 \leftarrow M_1\} \cdots \{n_k \leftarrow M_k\}$, the expression M_i bound to n_i lies in the scope of the bindings for the remaining names $n_{i+1}, \ldots, n_k$. We denote by ι the empty sequence of substitutions and treat it as the identity substitution. For an annotated process $\tilde{P}$, we define free and bound names in the same way as for ambient processes. Let $names(\sigma)$ be the set of all names occurring in σ.
We define a partial mapping $\mathcal{U}$ from closures to the set of ambient processes. Intuitively, it unfolds a closure to the process it represents by applying the substitution and cutting off the prefix defined by the offset. Roughly speaking, the expression $\mathcal{U}(\tilde{P}, \sigma)$ is defined if the offsets within the annotated process do not exceed the length of the expression they are associated with. The unfolding $\mathcal{U}(\tilde{P}, \sigma)$ is defined as follows.

The Unfolding $\mathcal{U}(\tilde{P}, \sigma)$ of a Closure $\langle \tilde{P}; \sigma \rangle$:

$$\mathcal{U}(\textstyle\prod_{i \in I} \pi_i, \sigma) = \begin{cases} \mathcal{U}(\pi_1, \sigma) \mid \ldots \mid \mathcal{U}(\pi_n, \sigma) & \text{if } I = \{1, \ldots, n\} \neq \varnothing \\ \mathbf{0} & \text{otherwise} \end{cases}$$

$$\mathcal{U}(M[\tilde{P}], \sigma) = M\sigma[\mathcal{U}(\tilde{P}, \sigma)]$$

$$\mathcal{U}(M(o).\tilde{P}, \sigma) = \begin{cases} N_{o+1}. \cdots .N_l.\mathcal{U}(\tilde{P}, \sigma) & \text{if } M\sigma = N_1. \cdots .N_l, o < l \text{ and } N_i \\ & \text{being either a name or of the form} \\ & cap\ N \text{ with } cap \in \{in, out, open\} \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\mathcal{U}((n).\tilde{P}, \sigma) = (n).\mathcal{U}(\tilde{P}, \sigma) \qquad \mathcal{U}(\langle M \rangle, \sigma) = \langle M\sigma \rangle$$

We are only interested in a particular kind of closure, which we refer to as *normal*. Let a closure $\langle \tilde{P}; \sigma \rangle$ be normal if each of the following conditions hold: (1) $\mathcal{U}(\tilde{P}, \sigma)$ is defined; (2) the bound names of $\tilde{P}$ are pairwise distinct and such that $bn(\tilde{P}) \cap (fn(\tilde{P}) \cup names(\sigma)) = \varnothing$; (3) every offset occurring in the scope of an input in $\tilde{P}$ is zero.
The next proposition says that our representation of ambient processes with normal closures preserves their basic properties. We write $\{\}$ and $+\!\!+$ for the empty multiset and the multiset union operation, respectively.

Proposition 3.1 (Structural Congruences). *Let $\tilde{P} = \prod_{i \in I} \pi_i$ and $\langle \tilde{P}; \sigma \rangle$ be a normal closure. Then*

(1) $\mathcal{U}(\tilde{P}, \sigma) \equiv \mathbf{0}$ *iff* $I = \varnothing$.

(2) $\mathcal{U}(\tilde{P},\sigma) \equiv M[Q]$ *iff* $\exists M', \tilde{Q} :$ *I is a singleton* $\{i\}$, $\pi_i = M'[\tilde{Q}]$, $M'\sigma = M$, *and* $\mathcal{U}(\tilde{Q},\sigma) \equiv Q$.
(3) $\mathcal{U}(\tilde{P},\sigma) \equiv P' \mid P''$ *iff* $\exists J, K : J \cup K = I$, $J \cap K = \varnothing$, $P' \equiv \mathcal{U}(\prod_{j\in J} \pi_j, \sigma)$, *and* $P'' \equiv \mathcal{U}(\prod_{k\in K} \pi_k, \sigma)$.
(4) $\mathcal{U}(\tilde{P},\sigma) \equiv \langle M \rangle$ *iff* $\exists M' :$ *I is a singleton* $\{i\}$, $\pi_i = \langle M' \rangle$ *and* $M'\sigma = M$.
(5) $\mathcal{U}(\tilde{P},\sigma) \equiv (n).Q$ *iff* $\exists \tilde{Q} :$ *I is a singleton* $\{i\}$, $\pi_i = (n).\tilde{Q}$ *and* $\mathcal{U}(\tilde{Q},\sigma) \equiv Q$.

Next, we present how the reduction and sublocation transitions can be defined on closures. Due to this particular representation and the fact that some part of the ambient process is contained in the sequential substitution, some auxiliary subroutines are needed.
One can see in the definition of $\mathcal{U}$ that only expressions M in the annotated process are affected by the sequential substitution. For the sublocation transition, it is important to extract the name represented by the expression M under the substitution σ. So, one of those subroutines, $nam(M,\sigma)$, consists in recovering from an expression M the name it effectively represents within the substitution σ and is inductively defined over sequential substitution as follows: for the empty substitution, $nam(n,\iota) = n$ and for an non-empty one,

$$nam(n, \{m \leftarrow M\}\sigma) = \begin{cases} nam(M,\sigma) & \text{if } n = m \\ nam(n,\sigma) & \text{otherwise} \end{cases}$$

The reduction transition for a closure $\langle \tilde{P}; \sigma \rangle$ requires some other auxiliary subroutines. Intuitively, the outcome of applying the substitution σ to an expression M contained within $\tilde{P}$ is a finite sequence of either capabilities of the form $in\, M'$, $out\, M'$, $open\, M'$, or names not bound by the substitution. We need a subroutine to compute the length of the sequence, and to do so in polynomial space, even though the length may be exponential in the size of the closure. Therefore, the subroutine $len(M,\sigma)$ cannot explicitly apply σ to M. It is defined by the equations:

$$\begin{aligned}
&len(\epsilon,\sigma) = 0 \\
&len(cap\ N,\sigma) = 1 \text{ if } cap \in \{in, out, open\} \\
&len(n,\iota) = 1 \\
&len(n, \{m \leftarrow N\}\sigma) = \begin{cases} len(N,\sigma) & \text{if } n = m \\ len(n,\sigma) & \text{otherwise} \end{cases} \\
&len(M.N,\sigma) = len(M,\sigma) + len(N,\sigma)
\end{aligned}$$

Now, from the definition of the reduction on ambient processes, one can see that the reduction consumes one capability: once the reduction is done, the involved capability disappears from the resulting process. This is slightly different for the representation we have proposed: a sequence of capabilities can be partially contained in a sequential substitution σ. This substitution remains fixed during the execution of capabilities and the offset attached to this sequence plays the role of a program counter. Therefore, to perform a reduction step one has to extract the first capability to execute from a sequence of capabilities, M, a substitution, σ, and an offset, o. This is computed by

$\mathit{fst}(M, o, \sigma)$ defined by:

$$\mathit{fst}(M.N, o, \sigma) = \begin{cases} \mathit{fst}(M, o, \sigma) & \text{if } \mathit{len}(M, \sigma) > o \\ \mathit{fst}(N, o - \mathit{len}(M, \sigma), \sigma) & \text{otherwise} \end{cases}$$
$$\mathit{fst}(\mathit{cap}\ N, 0, \sigma) = \mathit{cap}\ (\mathit{nam}(N, \sigma)) \text{ for } \mathit{cap} \in \{\mathit{in}, \mathit{out}, \mathit{open}\}$$
$$\mathit{fst}(n, o, \{m{\leftarrow}M\}\sigma) = \begin{cases} \mathit{fst}(M, o, \sigma) & \text{if } n = m \\ \mathit{fst}(n, o, \sigma) & \text{otherwise} \end{cases}$$

The next subroutine introduced here, $\mathit{split}(M(o).\tilde{P}, \sigma)$, computes a pair from a prime, $M(o).\tilde{P}$, and a sequential substitution, σ. The first component of this result is the first capability of $\langle\{M(o).\tilde{P}\}; \sigma\rangle$ (the one in head position). The second component is the remaining annotated process once this first capability has been executed.

$$\mathit{split}(M(o).\tilde{P}, \sigma) = \begin{cases} (\mathit{fst}(M, o, \sigma), \{M(o+1).\tilde{P}\}) & \text{if } \mathit{len}(M, \sigma) > o + 1 \\ (\mathit{fst}(M, o, \sigma), \tilde{P}) & \text{otherwise} \end{cases}$$

Suppose $\langle\tilde{P}; \sigma\rangle$ is a normal closure containing an action $M(o).\tilde{Q}$. From the definition of a normal closure, $\mathit{len}(M, \sigma) > o$, and if the action occurs under an input variable n, then the offset $o = 0$. If n occurs in M and gets bound to ϵ by an I/O step, it may be that $\mathit{len}(M, \{n{\leftarrow}\epsilon\}\sigma) = 0$. So, in the transition rule for I/O, we need to re-normalise the closure representing the outcome of the transition. We do so using the following subroutines $\mathit{norm}(\tilde{P}, \sigma)$ and $\mathit{norm}(\pi, \sigma)$ that return the annotated process obtained by removing from $\tilde{P}$ and π, respectively, any prefix $M(o)$ such that $\mathit{len}(M, \sigma) = 0$.

$$\mathit{norm}(\textstyle\prod_{i \in 1..k} \pi_i, \sigma) = \begin{cases} \{\} & \text{if } k = 0 \\ \mathit{norm}(\pi_1, \sigma) +\!\!+ \cdots +\!\!+ \mathit{norm}(\pi_k, \sigma) & \text{otherwise} \end{cases}$$
$$\mathit{norm}(M[\tilde{P}], \sigma) = \{M[\mathit{norm}(\tilde{P}, \sigma)]\}$$
$$\mathit{norm}(M(o).\tilde{P}, \sigma) = \begin{cases} \{M(o).\mathit{norm}(\tilde{P}, \sigma)\} & \text{if } \mathit{len}(M, \sigma) = 0 \\ \mathit{norm}(\tilde{P}, \sigma) & \text{otherwise} \end{cases}$$
$$\mathit{norm}((n).\tilde{P}, \sigma) = \{(n).\mathit{norm}(\tilde{P}, \sigma)\} \qquad \mathit{norm}(\langle M\rangle, \sigma) = \{\langle M\rangle\}$$

Notice that $\mathit{nam}(M, \sigma)$ is undefined if M is of the form ϵ, $N.N'$, $\mathit{in}\ N$, $\mathit{out}\ N$, or $\mathit{open}\ N$. Therefore, the expression $\mathit{nam}(M, \sigma)$ is either undefined or is evaluated to a name. Moreover, we can compute the name returned by $\mathit{nam}(M, \sigma)$, or whether it is undefined, in linear time. The number returned by $\mathit{len}(M, \sigma)$ can be computed in polynomial space. We can compute the capability returned by $\mathit{fst}(M, o, \sigma)$ and the pair returned by $\mathit{split}(M(o).\tilde{P}, \sigma)$, or whether they are undefined, in polynomial space. We can compute the outcome of $\mathit{norm}(\tilde{P}, \sigma)$ and $\mathit{norm}(\pi, \sigma)$ in polynomial space.
Next, we define transition and the sublocation relations on closures.

Transitions $\langle\tilde{P}; \sigma\rangle \to \langle\tilde{P}'; \sigma'\rangle$ and Sublocations $\langle\tilde{P}; \sigma\rangle \downarrow \langle\tilde{P}'; \sigma'\rangle$ of Closures:

(Trans In)
$$\frac{\mathit{split}(\pi, \sigma) = (\mathit{in}\ m, \tilde{P}) \quad \mathit{nam}(M, \sigma) = m \quad \mathit{nam}(N, \sigma) = n}{\langle\{N[\{\pi\} +\!\!+ \tilde{Q}], M[\tilde{R}]\}; \sigma\rangle \to \langle\{M[\{N[\tilde{P} +\!\!+ \tilde{Q}]\} +\!\!+ \tilde{R}]\}; \sigma\rangle}$$

(Trans Out)
$$\frac{\mathit{split}(\pi, \sigma) = (\mathit{out}\ m, \tilde{P}) \quad \mathit{nam}(M, \sigma) = m \quad \mathit{nam}(N, \sigma) = n}{\langle\{M[\{N[\{\pi\} +\!\!+ \tilde{Q}]\} +\!\!+ \tilde{R}]\}; \sigma\rangle \to \langle\{N[\tilde{P} +\!\!+ \tilde{Q}], M[\tilde{R}]\}; \sigma\rangle}$$

(Trans Open)
$$\frac{split(\pi,\sigma) = (open\ n, \tilde{P}) \quad nam(M,\sigma) = n}{\langle \pi, \{M[\tilde{Q}]\}; \sigma\rangle \to \langle \tilde{P} +\!\!+\ \tilde{Q}; \sigma\rangle}$$

(Trans I/O)
$$\frac{\tilde{P}' = norm(\tilde{P}, \{n \leftarrow M\}\sigma)}{\langle \{(n).\tilde{P}, \langle M\rangle\}; \sigma\rangle \to \langle \tilde{P}'; \{n \leftarrow M\}\sigma\rangle}$$

(Trans Par)
$$\frac{\langle \tilde{P}; \sigma\rangle \to \langle \tilde{P}'; \sigma'\rangle}{\langle \tilde{P} +\!\!+\ \tilde{Q}; \sigma\rangle \to \langle \tilde{P}' +\!\!+\ \tilde{Q}; \sigma'\rangle}$$

(Trans Amb)
$$\frac{\langle \tilde{P}; \sigma\rangle \to \langle \tilde{P}'; \sigma'\rangle \quad nam(M,\sigma) = n}{\langle \{M[\tilde{P}]\}; \sigma\rangle \to \langle \{M[\tilde{P}']\}; \sigma'\rangle}$$

(Loc)
$$\frac{nam(M,\sigma) = m}{\langle \tilde{Q} +\!\!+\ \{M[\tilde{P}]\}; \sigma\rangle \downarrow \langle \tilde{P}; \sigma\rangle}$$

The condition for (Loc) ensures simply that the expression M together with σ is a name. For two normal closures $\langle P; \sigma\rangle$, $\langle P'; \sigma'\rangle$, deciding whether $\langle P; \sigma\rangle \downarrow \langle P'; \sigma'\rangle$ can be achieved in polynomial space. There is no rule corresponding to (Red $\equiv$) since we always keep closures in normal form. The two rules (Trans Par) and (Trans Amb) correspond to the congruence rules (Red Par) and (Red Amb) for reduction.
In the same way as for ambient processes, we define the relations $\to^*$ and $\downarrow^*$ (on closures) as the reflexive and transitive closures of $\to$ and $\downarrow$, respectively.

Proposition 3.2. *If $\langle \tilde{P}; \sigma\rangle$ is normal and $\langle \tilde{P}; \sigma\rangle \downarrow^* \langle \tilde{P}'; \sigma\rangle$ then $\langle \tilde{P}'; \sigma\rangle$ is normal. If $\langle \tilde{P}; \sigma\rangle$ is normal and $\langle \tilde{P}; \sigma\rangle \to^* \langle \tilde{P}'; \sigma'\rangle$ then $\langle \tilde{P}'; \sigma'\rangle$ is normal.*

The next propositions says that the representation of processes as closures preserves sublocations and reductions.

Proposition 3.3 (Sublocation Equivalences). *Assume $\langle \tilde{P}; \sigma\rangle$ is a normal closure. If $\langle \tilde{P}; \sigma\rangle \downarrow \langle \tilde{Q}; \sigma\rangle$ then $\mathcal{U}(\tilde{P}, \sigma) \downarrow \mathcal{U}(\tilde{Q}, \sigma)$. If $\mathcal{U}(\tilde{P}, \sigma) \downarrow Q$ then there exists $\tilde{Q}$ such that $\langle \tilde{P}; \sigma\rangle \downarrow \langle \tilde{Q}; \sigma\rangle$ and $\mathcal{U}(\tilde{Q}, \sigma) \equiv Q$.*

Proposition 3.4 (Reduction Equivalences). *Assume $\langle \tilde{P}; \sigma\rangle$ is a normal closure. If $\langle \tilde{P}; \sigma\rangle \to \langle \tilde{P}'; \sigma'\rangle$ then $\mathcal{U}(\tilde{P}, \sigma) \to \mathcal{U}(\tilde{P}', \sigma')$. If $\mathcal{U}(\tilde{P}, \sigma) \to P'$ then there exists $\langle \tilde{P}'; \sigma'\rangle$ such that $\langle \tilde{P}; \sigma\rangle \to \langle \tilde{P}'; \sigma'\rangle$ and $\mathcal{U}(\tilde{P}', \sigma') \equiv P'$.*

Propositions 3.1–3.4 are enough to prove that normal closures indeed simulate the processes they represent.

3.2 Size of the Representation

We show that closures indeed give a polynomial representation of processes. To do this, we have to bound the size of offsets that occur in closures.
For a given object (a closure or a process) O, by $|O|$ we mean the length of its string representation and by $||O||$ the number of nodes in its tree representation. We assume that an offset is represented by a single node in the tree representation.

Lemma 3.1. *Suppose that $\langle \tilde{P}; \sigma\rangle \to \langle \tilde{P}'; \sigma'\rangle$. Then $||\langle \tilde{P}'; \sigma'\rangle|| \leq ||\langle \tilde{P}; \sigma\rangle||$.*

Proof. A simple case analysis on the derivation of $\langle \tilde{P}; \sigma\rangle \to \langle \tilde{P}'; \sigma'\rangle$. □

Proposition 3.5. *Assume $\langle \tilde{P}; \sigma \rangle$ is normal and $\langle \tilde{P}; \sigma \rangle \rightarrow \langle \tilde{P}'; \sigma' \rangle$. Then all offsets used in $\tilde{P}$ and $\tilde{P}'$ can be represented by the same number of bits, polynomial in $|\langle \tilde{P}; \sigma \rangle|$ and, with such a representation, $|\langle \tilde{P}'; \sigma' \rangle| \leq |\langle \tilde{P}; \sigma \rangle|$.*

Proof. A simple induction on the length of the substitution σ' proves that the offsets in $\tilde{P}'$ are bounded by the value $\|\langle \tilde{P}'; \sigma' \rangle\|^{\|\langle \tilde{P}'; \sigma' \rangle\|}$. By Lemma 3.1, they are also bounded by $\|\langle \tilde{P}; \sigma \rangle\|^{\|\langle \tilde{P}; \sigma \rangle\|}$ and then all offsets used in $\tilde{P}$ and $\tilde{P}'$ are bounded by this value, which can be represented on $\|\langle \tilde{P}; \sigma \rangle\| \cdot (\lfloor \log(\|\langle \tilde{P}; \sigma \rangle\|) \rfloor + 1)$ bits. With this representation of offsets, incrementing an offset does not increase the size of its string representation. Thus no transitions can increase the length of the string representations of closures. $\square$

The following proposition is a key fact in the proof that our model checking algorithm and also the algorithm in [4] terminate in exponential time. It implies that the computation tree of a given process might be very deep and very narrow (as in our example in Sect. 2) or not so deep and wider; in any case the number of nodes in the tree remains exponentially bounded. A naive argument (without using closures) gives only a doubly exponential bound on the number of reachable processes: one can prove that the computation tree of a given process is at most exponentially deep (as our example in Sect. 2 shows, this bound is tight) and that the number of successors for every node is at most polynomial. For example, the closure $\langle \{n[inn(0).\tilde{P}_0], \ldots, n[inn(0).\tilde{P}_k]\}; \sigma \rangle$ has at most k^2 different successors. These two facts do not give, however, the exponential bound on the number of nodes in the tree, which is given by the following proposition.

Proposition 3.6. *Let $\langle \tilde{P}; \sigma \rangle$ be a normal closure. Then there exist at most exponentially many $\langle \tilde{P}'; \sigma' \rangle$ such that $\langle \tilde{P}; \sigma \rangle \rightarrow^* \langle \tilde{P}'; \sigma' \rangle$.*

Proof. This is a direct consequence of Proposition 3.5 and the observation that there are only exponentially many strings of polynomial length. $\square$

Proposition 3.7. *The reachability problem for normal closures is decidable in PSPACE.*

Proof. Take any instance $\langle \tilde{P}; \sigma \rangle$, $\langle \tilde{P}'; \sigma' \rangle$ of the reachability problem. To decide whether $\langle \tilde{P}; \sigma \rangle \rightarrow^* \langle \tilde{P}'; \sigma' \rangle$, we first define a nondeterministic algorithm that starting from $\langle \tilde{P}; \sigma \rangle$ guesses an immediate successor of the current closure until it reaches $\langle \tilde{P}'; \sigma' \rangle$ or there are no further successors. By Proposition 3.5 the algorithm requires only polynomial space (we have to store only the current closure and its one immediate successor); Proposition 3.6 implies termination. Finally, using the general statement of Savitch's theorem [10] (NPSPACE$(S(n)) \subseteq$ PSPACE$(S(n)^2)$), this non-deterministic algorithm can be turned into a deterministic one. $\square$

3.3 A New Algorithm

We propose a new algorithm, $\mathit{Check}(\tilde{P}, \sigma, \mathcal{A})$, to check whether the ambient process simulated by $\langle \tilde{P}; \sigma \rangle$ satisfies the closed formula $\mathcal{A}$. For each ambient process, P, we only consider the closure, $\mathcal{F}(P)$, obtained using the *folding* function defined as follows. We prove (Proposition 3.9), that $P \models \mathcal{A}$ if and only if $\mathit{Check}(\mathcal{F}(P), \iota, \mathcal{A})$ returns the Boolean value $\mathbf{T}$.

The Folding $\mathcal{F}(P)$ of a Process P:

$$\mathcal{F}(\mathbf{0}) = \{\} \qquad \mathcal{F}(P \mid Q) = \mathcal{F}(P) +\!\!+\ \mathcal{F}(Q)$$
$$\mathcal{F}(M[P]) = \{M[\mathcal{F}(P)]\} \qquad \mathcal{F}((n).P) = \{(n).\mathcal{F}(P)\}$$
$$\mathcal{F}(\langle M \rangle) = \{\langle M \rangle\}$$
$$\mathcal{F}(M.P) = \begin{cases} \mathcal{F}(P) & \text{if } len(M, \iota) = 0 \\ \{M(0).\mathcal{F}(P)\} & \text{otherwise} \end{cases}$$

For any process P, the closure $\langle \mathcal{F}(P); \iota \rangle$ is normal and $\mathcal{U}(\mathcal{F}(P), \iota)$ is structurally congruent to P. Furthermore, $\mathcal{F}(P)$ can be computed in linear time in the size of P.
For the model checking problem, $P \models \mathcal{A}$, we may assume without loss of generality that the free names of $\mathcal{A}$ are disjoint from the bound names of P. We denote by $fn(\tilde{P}, \sigma)$ the set $(fn(\tilde{P}) \cup names(\sigma)) \setminus dom(\sigma)$.

Computing whether a Process Satisfies a Closed Formula:

$$Check(\tilde{P}, \sigma, \mathbf{T}) = \mathbf{T}$$
$$Check(\tilde{P}, \sigma, \neg\mathcal{A}) = \neg\, Check(\tilde{P}, \sigma, \mathcal{A})$$
$$Check(\tilde{P}, \sigma, \mathcal{A} \vee \mathcal{B}) = Check(\tilde{P}, \sigma, \mathcal{A}) \vee Check(\tilde{P}, \sigma, \mathcal{B})$$
$$Check(\textstyle\prod_{i \in I} \pi_i, \sigma, \mathbf{0}) = \begin{cases} \mathbf{T} \text{ if } I = \varnothing \\ \mathbf{F} \text{ otherwise} \end{cases}$$
$$Check(\textstyle\prod_{i \in I} \pi_i, \sigma, n[\mathcal{A}]) = \begin{cases} Check(\tilde{Q}, \sigma, \mathcal{A}) & \text{if } I = \{i\},\ \pi_i = M[\tilde{Q}], \\ & \quad nam(M, \sigma) = n \\ \mathbf{F} & \text{otherwise} \end{cases}$$
$$Check(\textstyle\prod_{i \in I} \pi_i, \sigma, \mathcal{A} \mid \mathcal{B}) = \bigvee_{J \subseteq I} Check(\prod_{j \in J} \pi_j, \sigma, \mathcal{A}) \wedge Check(\prod_{k \in I-J} \pi_k, \sigma, \mathcal{B})$$
$$\begin{aligned} Check(\tilde{P}, \sigma, \exists x.\mathcal{A}) = {} & \text{let } \{m_1, \ldots, m_k\} = fn(\tilde{P}, \sigma) \cup fn(\mathcal{A}) \text{ in} \\ & \text{let } m_0 \notin \{m_1, \ldots, m_k\} \cup bn(\tilde{P}) \cup dom(\sigma) \text{ be fresh in} \\ & \textstyle\bigvee_{i \in 0..k} Check(\tilde{P}, \sigma, \mathcal{A}\{x \leftarrow m_i\}) \end{aligned}$$
$$Check(\tilde{P}, \sigma, \Diamond\mathcal{A}) = \textstyle\bigvee_{\langle \tilde{P}; \sigma \rangle \to^* \langle \tilde{P}'; \sigma' \rangle} Check(\tilde{P}', \sigma', \mathcal{A})$$
$$Check(\tilde{P}, \sigma, \diamond\mathcal{A}) = \textstyle\bigvee_{\langle \tilde{P}; \sigma \rangle \downarrow^* \langle \tilde{P}'; \sigma \rangle} Check(\tilde{P}', \sigma, \mathcal{A})$$
$$Check(\tilde{P}, \sigma, \mathcal{A}@n) = Check(n[\tilde{P}], \sigma, \mathcal{A})$$

An expression $Check(\tilde{P}, \sigma, \mathcal{A})$ is said to be *normal* if and only if the closure $\langle \tilde{P}; \sigma \rangle$ is normal, $\mathcal{A}$ is a closed formula, and $fn(\mathcal{A}) \cap (bn(\tilde{P}) \cup dom(\sigma)) = \varnothing$. Hence, for the model checking problem $P \models \mathcal{A}$ where $\mathcal{A}$ is a closed formula, the expression $Check(\mathcal{F}(P), \iota, \mathcal{A})$ is normal and moreover we have:

Proposition 3.8. *The model checking algorithm described above preserves the normality of $Check(\tilde{P}, \sigma, \mathcal{A})$.*

Proposition 3.9. *For all processes P and closed formulas $\mathcal{A}$, we have $P \models \mathcal{A}$ if and only if $Check(\mathcal{F}(P), \iota, \mathcal{A}) = \mathbf{T}$.*

Proof. By induction on the structure of the ambient formula $\mathcal{A}$ with appeal to Propositions 3.3 and 3.4 in the cases $\diamond\mathcal{A}$ and $\Diamond\mathcal{A}$, respectively. $\square$

Theorem 3.1. *Model checking the ambient calculus and logic of this paper is decidable in PSPACE.*

Proof. To test for a given process P and formula $\mathcal{A}$ whether $P \models \mathcal{A}$ we simply compute the value of $\mathit{Check}(\mathcal{F}(P), \iota, \mathcal{A})$. The only problem is to implement Check in such a way that it works in polynomial space.
In the case of $\mathbf{T}, \mathbf{0}, n[\mathcal{A}], \mathcal{A}@n, \neg\mathcal{A}$, the algorithm can directly check whether the respective conditions hold. In the case of $\mathcal{A} \vee \mathcal{B}, \mathcal{A} \mid \mathcal{B}, \exists x.\mathcal{A}, \Diamond\mathcal{A}, \diamondsuit\mathcal{A}$, we have to be more careful about the space used to compute the value of disjunctions. In a loop we iteratively compute the value of each disjunct, reusing the same space in every iteration. In the case of $\Diamond\mathcal{A}$ the subroutine computing $\bigvee_{\langle\tilde{P};\sigma\rangle \to^* \langle\tilde{P}';\sigma'\rangle} \mathit{Check}(\tilde{P}', \sigma', \mathcal{A})$ could look as follows.

```
result ← F
for all ⟨P̃′; σ′⟩ such that ⟨P̃; σ⟩ →* ⟨P̃′; σ′⟩
    if Check(P̃′, σ′, A) = T then result ← T
return(result)
```

By Propositions 3.5 and 3.7, every iteration requires only polynomial space. The cases of $\mathcal{A} \vee \mathcal{B}, \mathcal{A} \mid \mathcal{B}, \exists x.\mathcal{A}, \diamondsuit\mathcal{A}$ are similar. Thus, the space $S(k, |\tilde{P}| + |\sigma|)$ used by the algorithm to compute $\mathit{Check}(\tilde{P}, \sigma, \mathcal{A})$ for formulas $\mathcal{A}$ of depth not exceeding k satisfies the inequality

$$S(k+1, |\tilde{P}| + |\sigma|) \leq S(k, |\tilde{P}| + c + |\sigma|) + p(|\tilde{P}| + |\sigma|)$$

for some constant c and some polynomial p (the constant c comes from the fact that in the case of $\mathcal{A} = \mathcal{B}@n$ the size of $n[\tilde{P}]$ is greater than the size of $\tilde{P}$; the polynomial p estimates the space needed for testing reachability etc). Therefore, $S(k, |\tilde{P}| + |\sigma|) \leq k \cdot p(|\tilde{P}| + k \cdot c + |\sigma|)$.
Finally, the fact that $\mathcal{F}(P)$ is polynomial in the size of P and the statement of Proposition 3.9 complete the proof. ◻

4 Complexity Lower Bounds

Below we present lower bounds on the space complexity of model checking our process calculus against our modal logic, and also for two significant fragments.
The results given here are based on known results about the complexity of decision problems for Quantified Boolean Formulas (QBF). The alternation depth of a formula is the number of alternations between existential and universal quantifiers in its prenex quantification.
Those known results are: (1) deciding the validity problem for a closed quantified Boolean formula φ is PSPACE-complete; (2) deciding the validity problem for a closed quantified Boolean formula φ of alternation depth k whose outermost quantifier is $\exists$ is Σ_k^P-complete [11], where Σ_k^P denotes the k-th level of the polynomial-time hierarchy. In particular, $\Sigma_0^P = P$ and $\Sigma_1^P = NP$.

4.1 The Full Calculus and Logic

We define an encoding of QBF formulas into ambient formulas. (We can assume without loss of generality that these Boolean formulas are in prenex and conjunctive normal

form.) This encoding is then used to prove Theorem 4.1, that the complexity of model checking the ambient logic is PSPACE-hard.
In our encoding, we assume that the truth values tt and ff used in the definition of QBF satisfaction [7] are distinct ambient calculus names.
We also use a derived operator for name equality in the ambient logic [4], $\eta = \mu$, encoded as $\eta[\mathbf{T}]@\mu$.Then $\mathbf{0} \models m = n$ if and only if the names m and n are equal.
We encode the $\forall$ and $\exists$ quantifiers over truth values as follows.

$$\forall x \in \{\mathit{ff}, tt\}.\mathcal{A} \triangleq \forall x.(x = \mathit{ff} \vee x = tt) \Rightarrow \mathcal{A}$$
$$\exists x \in \{\mathit{ff}, tt\}.\mathcal{A} \triangleq \exists x.(x = \mathit{ff} \vee x = tt) \wedge \mathcal{A}$$

Encoding QBF Formulas as Ambient Logic Formulas:

$[\![v]\!] \triangleq (v = tt)$	$[\![\overline{v}]\!] \triangleq (v = \mathit{ff})$
$[\![\ell_1 \vee \cdots \vee \ell_k]\!] \triangleq [\![\ell_1]\!] \vee \cdots \vee [\![\ell_k]\!]$	$[\![C_1 \wedge \cdots \wedge C_k]\!] \triangleq [\![C_1]\!] \wedge \cdots \wedge [\![C_k]\!]$
$[\![\forall v.\varphi]\!] \triangleq \forall v \in \{\mathit{ff}, tt\}.[\![\varphi]\!]$	$[\![\exists v.\varphi]\!] \triangleq \exists v \in \{\mathit{ff}, tt\}.[\![\varphi]\!]$

The following properties are proved in the extended version of this paper [7].

Lemma 4.1. *Consider a closed quantified boolean formula φ and its encoding $[\![\varphi]\!]$ in the ambient logic. The formula φ is valid if and only if the model checking problem $\mathbf{0} \models [\![\varphi]\!]$ holds.*

Theorem 4.1. *The complexity of model checking the full logic (including name quantification) is PSPACE-hard.*

Proof. Straightforward from Lemma 4.1 since for the fixed ambient process $\mathbf{0}$ solving the model checking problem $\mathbf{0} \models \varphi$ is PSPACE-hard. So in fact the expression complexity, that is, the complexity of checking formulas against a fixed process, is PSPACE-hard. □

The theorem above holds for any fragment of the logic including boolean connectives, name quantification, and the location and location adjunct modalities, and for any fragment of the calculus including ambients. This might suggest that the complexity of the model checking problem comes from the quantification in the logic. It is not the case: the problem remains as complex even if we remove quantification from the logic and communication or mobility from the calculus. This suggests there is little chance of finding interesting fragments of the calculus and the logic that would admit a faster model checking algorithm.

4.2 Mobile Ambients Without I/O, No Quantifiers

In this section, we study the complexity of the model checking problem for the fragment of the ambient calculus without I/O and the fragment of the logic without quantification. For every QBF variable, v, we assume that v, v' and v'' are distinct ambient calculus names.

Encoding QBF Formulas as Ambient Processes and Formulas:

$[\![v]\!] = v[pos[\mathbf{0}] \mid v'[\mathbf{0}]] \mid \mathbf{T}$
$[\![\overline{v}]\!] = v[neg[\mathbf{0}] \mid v'[\mathbf{0}]] \mid \mathbf{T}$
$[\![\ell_1 \vee \cdots \vee \ell_k]\!] = [\![\ell_1]\!] \vee \cdots \vee [\![\ell_k]\!]$

$[\![C_1 \wedge \cdots \wedge C_k]\!] = (end[\mathbf{0}], [\![C_1]\!] \wedge \cdots \wedge [\![C_k]\!])$

$[\![\forall v.\varphi]\!] = (v'[in\ v.n[out\ v'.out\ v.P]], \Box((n[\mathbf{T}] \mid \mathbf{T}) \Rightarrow \mathcal{A}))$ where $(n[P], \mathcal{A}) = [\![\varphi]\!]$
$[\![\exists v.\varphi]\!] = (v'[in\ v.n[out\ v'.out\ v.P]], \Diamond((n[\mathbf{T}] \mid \mathbf{T}) \wedge \mathcal{A}))$ where $(n[P], \mathcal{A}) = [\![\varphi]\!]$

$enc(\varphi) = (v_1[pos[\mathbf{0}]] \mid v_1[neg[\mathbf{0}]] \mid \cdots \mid v_n[pos[\mathbf{0}]] \mid v_n[neg[\mathbf{0}]] \mid P, \mathcal{A})$
where $(P, \mathcal{A}) = [\![\varphi]\!]$ and $\varphi = Q_1 v_1 . \ldots . Q_n v_n . C_1 \wedge \cdots \wedge C_k$
where each $Q_i \in \{\exists, \forall\}$.

In the encoding $enc(\varphi)$ above, the parallel composition $v_1[pos[\mathbf{0}]] \mid \ldots \mid v_n[neg[\mathbf{0}]]$ represents the sequence $v_1, \ldots v_n$ of (uninstantiated) boolean variables and P is a process that instantiates them. An instantiated variable v_i is represented by a subprocess $v_i[pos[\mathbf{0}] \mid v_i'[\mathbf{0}]] \mid v_i[neg[\mathbf{0}]]$ (if its value is tt) or $v_i[pos[\mathbf{0}]] \mid v_i[neg[\mathbf{0}] \mid v_i'[\mathbf{0}]]$ (if its value is *ff*). The process P first instantiates v_1 by choosing one of the ambients $v_1[pos[\mathbf{0}]]$ or $v_1[neg[\mathbf{0}]]$ nondeterministically, going inside it, leaving the token $v_1'[\mathbf{0}]$ inside the chosen ambient and then returning to the top level. It then iteratively instantiates the variables $v_2, \ldots, v_n$ in the same way. The formula $n[\mathbf{T}] \mid \mathbf{T}$ in the context of the encoding for a quantified variable v_i above (where n is v_{i+1} or end for $i = n$) expresses that instantiation of v_i has finished but instantiation of n has yet to start; thus $\Box(n[\mathbf{T}] \mid \mathbf{T} \ldots)$ and $\Diamond(n[\mathbf{T}] \mid \mathbf{T} \ldots)$ express, respectively, universal and existential quantifications over instantiations of v_i. For more details of the encoding, including examples and the proof of the lemma below, we refer to the full version of this paper [7].

Lemma 4.2. *Assume φ is a closed quantified Boolean formula, and that $(P, \mathcal{A}) = enc(\varphi)$. Then $P \models \mathcal{A}$ if and only if φ is valid.*

Theorem 4.2. *The complexity of model checking mobile ambients without I/O against the quantifier-free logic is PSPACE-hard.*

Proof. Straightforward from the PSPACE-completeness of the validity for QBF and from Lemma 4.2, taking into account that for $enc(\varphi) = (P, \mathcal{A})$, both P and $\mathcal{A}$ are of polynomial size with respect to φ. $\Box$

4.3 Immobile Ambients With I/O, No Quantifiers

In this section, we study the complexity of the model checking problem for the fragment of the ambient calculus without action prefix. For full details, see the full version.
We consider fixed names end, C, and D. For any QBF variable ambient name v_i', let $Inst(v_i') \triangleq v_i'[\mathbf{T}] \mid \mathbf{T}$ and $Inst^+(v_i') \triangleq v_i'[v_i''[\mathbf{T}] \mid \mathbf{T}] \mid \mathbf{T}$ and for the name end, $Inst(end) \triangleq end[\mathbf{T}] \mid \mathbf{T}$ and $Inst^+(end) \triangleq end[end'[\mathbf{T}] \mid \mathbf{T}] \mid \mathbf{T}$.

Encoding QBF Formulas as Ambient Processes and Formulas:

$$[\![v]\!] = v[]$$
$$[\![\overline{v}]\!] = \overline{v}[]$$
$$[\![\ell_1 \vee \ldots \vee \ell_k]\!] = D[\mathbf{0}] \mid [\![\ell_1]\!] \mid \ldots \mid [\![\ell_k]\!]$$

$$\begin{aligned} enc(C_1 \wedge \ldots \wedge C_k) &= (end[C[\,[\![C_1]\!]\,] \mid \ldots \mid C[\,[\![C_k]\!]\,]], \\ &\qquad ❏((D[\mathbf{0}] \mid \mathbf{T}) \Rightarrow (tt[\mathbf{0}] \mid \mathbf{T}))) \\ enc(\exists v.\varphi) &= (v'[\langle tt\rangle \mid \langle ff\rangle \mid (v).(v''[] \mid (\overline{v}).n[P])], \\ &\qquad \mathbf{T} \mid v'[\Diamond(\, (Inst(n) \wedge \neg Inst^{+}(n)) \, \wedge \mathcal{A}\,)]) \quad \text{where } enc(\varphi) = (n[P], \mathcal{A}) \\ enc(\forall v.\varphi) &= (v'[\langle tt\rangle \mid \langle ff\rangle \mid (v).(v''[] \mid (\overline{v}).n[P])], \\ &\qquad \mathbf{T} \mid v'[\Box(\, (Inst(n) \wedge \neg Inst^{+}(n)) \, \Rightarrow \mathcal{A}\,)]) \quad \text{where } enc(\varphi) = (n[P], \mathcal{A}) \end{aligned}$$

The idea of the encoding here is quite similar to that from the previous section. A boolean variable v is represented here by two ambients $v[]$ and $\overline{v}[]$, which after the instantiation are named $tt[]$ and $ff[]$. We exploit here the nondeterminism of communication: the variable v reads either the message $\langle tt\rangle$ or $\langle ff\rangle$; then its dual $\overline{v}$ has to read the other one. The names v_i' and v_i'' (similar to v_i' in the previous section) are used for distinguishing the moment when the variable v_i is already instantiated but v_{i+1} is not. The formula $❏((D[\mathbf{0}] \mid \mathbf{T}) \Rightarrow (tt[\mathbf{0}] \mid \mathbf{T}))$ requires that in the final state, each ambient representing a clause (that is, an ambient containing $D[\mathbf{0}]$) contains at least one true literal (that is, an ambient $tt[\mathbf{0}]$).

Lemma 4.3. *Assume φ is a closed quantified Boolean formula, and that $(P, \mathcal{A}) = enc(\varphi)$. Then $P \models \mathcal{A}$ if and only if φ is valid.*

Theorem 4.3. *The complexity of model checking immobile ambients with I/O against the quantifier-free logic is PSPACE-hard.*

Proof. This follows from the PSPACE-completeness of validity for QBF, from Lemma 4.3 taking into account that for $enc(\varphi) = (P, \mathcal{A})$, both P and $\mathcal{A}$ are of polynomial size with respect to φ. □

We can strengthen this result by slightly modifying our encoding to obtain a lower bound on the program complexity of the problem, that is, the complexity of model checking processes against a fixed formula.

Theorem 4.4. *For every integer k there exists a formula $\mathcal{A}_k^{\exists}$ such that the complexity of model checking processes against $\mathcal{A}_k^{\exists}$ is Σ_k^P-hard.*

5 Conclusion

We show in this paper that the model checking problem of the replication-free ambient calculus with public names against the ambient logic without composition-adjunct is PSPACE-complete. In order to prove this complexity bound, we have proposed a new representation for processes, called closures, that prevents the exponential blow-up of the size of the problem. We use this representation together with a new algorithm to prove the PSPACE upper bound.

An important property obtained using closures is that there exist at most exponentially many (up to structural congruence) processes reachable from a given ambient process. This result is interesting because it seems very difficult to give a precise estimation of the number of reachable processes without using this new representation. This bound can be used, for instance, to prove that a model checking algorithm previously proposed by Cardelli and Gordon [4] works in single-exponential time on single-exponential space.

Another result given in this paper is an analysis of the complexity of the model checking problem for different interesting subsets of the calculus and the logic. We show that there is little chance to find polynomial algorithms for interesting subproblems. Indeed, model checking remains PSPACE-hard even for quite simple fragments of the calculus (for instance, without communication) and the logic (for instance, without quantification on names).

Possible directions for future work include investigations of the model checking problem for extensions of the logic and the calculus. Recently, Cardelli and Gordon [6] presented an extended version of the logic that allows reasoning about restricted names; it seems that there is no difficulty in extending our algorithm to deal with name restriction.

References

1. L. Cardelli and A.D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240:177–213, 2000.
2. L. Cardelli and A.D. Gordon. Types for mobile ambients. In *Proceedings POPL'99*, pages 79–92. ACM, 1999.
3. L. Cardelli and A.D. Gordon. Equational properties of mobile ambients. In *Proceedings FoSSaCS'99*, volume 1578 of *Lecture Notes in Computer Science*, pages 212–226. Springer, 1999. An extended version appears as Technical Report MSR–TR–99–11, Microsoft Research, 1999.
4. L. Cardelli and A.D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *Proceedings POPL'00*, pages 365–377. ACM, 2000.
5. L. Cardelli and G. Ghelli. A query language for semistructured data based on the ambient logic. To appear in one of the proceedings volumes of ETAPS'01, to accompany an invited talk. Springer, 2001.
6. L. Cardelli and A.D. Gordon. Logical properties of name restriction. In *Proceedings TLCA'01*. Springer, 2001. To appear.
7. W. Charatonik, S. Dal Zilio, A.D. Gordon, S. Mukhopadhyay, and J.-M. Talbot. The complexity of model checking mobile ambients. Technical Report MSR–TR–2001–03, Microsoft Research, 2001.
8. O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.
9. D. Sangiorgi. Extensionality and intensionality of the ambient logics. In *Proceedings POPL'01*. ACM, 2001. To appear.
10. W.J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.
11. L.J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.