The Complexity of Model Checking Mobile Ambients

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

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, 2, 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. The parallel adjunct modality is omitted because it is defined as an infinite quantification over processes. 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.

[1]Max-Planck-Institut für Informatik, Germany.
[2]University of Wrocław, Poland.
[3]Microsoft Research, United Kingdom.

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 1 and 2) 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 Section 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 Section 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 nondeterministic algorithm that does not have to store all the reachable processes, and then remove nondeterminism using Savitch's theorem [10]. Hence we prove Theorem 1, that the model-checking problem is solvable in PSPACE.

We show this upper bound to be tight in Section 4; Theorem 2 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 3, 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, 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 **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 5 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, as well as 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 $P \rightarrow Q$. The *reduction* relation, $P \rightarrow Q$, 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, $P \downarrow Q$, that defines the spatial distribution of processes. We write $\rightarrow^*$ and $\downarrow^*$ for the reflexive and transitive closure of $\rightarrow$ and $\downarrow$, respectively.

Structural congruence $P \equiv Q$

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

Reduction $P \rightarrow Q$ and sublocation $P \downarrow Q$

$n[in\ m.P \mid Q] \mid m[R] \rightarrow m[n[P \mid Q] \mid R]$	(Red In)
$m[n[out\ m.P \mid Q] \mid R] \rightarrow n[P \mid Q] \mid m[R]$	(Red Out)
$open\ n.P \mid n[Q] \rightarrow P \mid Q$	(Red Open)
$\langle M \rangle \mid (n).P \rightarrow P\{n{\leftarrow}M\}$	(Red I/O)
$P \rightarrow Q \Rightarrow P \mid R \rightarrow Q \mid R$	(Red Par)
$P \rightarrow Q \Rightarrow n[P] \rightarrow n[Q]$	(Red Amb)
$P' \equiv P, P \rightarrow Q, Q \equiv Q' \Rightarrow P' \rightarrow Q'$	(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. An innovation of this paper is a new polynomial-size representation of processes that allows us to solve the model-checking problem in polynomial space, in spite of this example.

Consider the family of processes $(P_n)_{n \geq 0}$, recursively defined by the equations $P_0 = (y).(p[y] \mid q[])$ and $P_{n+1} = (x_{n+1}).(\langle x_{n+1}.x_{n+1} \rangle \mid P_n)$. Intuitively, the process P_{n+1} reads the variable x_{n+1}, doubles it and sends the result to the process P_n. We have the following, where $M^1 = M$ and $M^{n+1} = M.M^n$.

$$\begin{array}{lll}
\langle in\ q.out\ q \rangle \mid P_0 & \rightarrow^1 & p[in\ q.out\ q] \mid q[] \\
\langle in\ q.out\ q \rangle \mid P_1 & \rightarrow^2 & p[(in\ q.out\ q)^2] \mid q[] \\
\langle in\ q.out\ q \rangle \mid P_2 & \rightarrow^3 & p[(in\ q.out\ q)^4] \mid q[] \\
\langle in\ q.out\ q \rangle \mid P_n & \rightarrow^{n+1} & p[(in\ q.out\ q)^{2^n}] \mid q[]
\end{array}$$

Since $(in\ q.out\ q)^{2^n}$ is a sequence of 2^n copies of $in\ q.out\ q$, the process $p[(in\ q.out\ q)^{2^n}] \mid q[]$ reduces in 2^{n+1} steps to $p[] \mid q[]$ and therefore, $\langle in\ q.out\ q \rangle \mid P_n \rightarrow^{(n+1)+2^{n+1}} p[] \mid q[]$.

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.

These remarks motivate the approach taken in this paper. First, we devise a new representation for ambient processes that remains of polynomial size with respect to to the input process. Second, we give a non-deterministic algorithm using only polynomial space in the combined size of the problem, which gives a deterministic version that itself uses only polynomial space.

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
$\mathbf{0}$	void
$\eta[\mathcal{A}]$	ambient match
$\mathcal{A} \mid \mathcal{B}$	composition match
$\exists x.\mathcal{A}$	existential quantification
$\Diamond\mathcal{A}$	sometime modality

$\diamondsuit\mathcal{A}$	somewhere modality
$\mathcal{A}@\eta$	location adjunct

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 $\mathbf{0}$. A process satisfies $n[\mathcal{A}]$ if it is structurally congruent to an ambient $n[P]$ where P satisfies $\mathcal{A}$. A process satisfies $A \mid \mathcal{B}$ if it is structurally congruent to a composition $P \mid Q$, where P satisfies $\mathcal{A}$, and Q 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 $P \models \mathcal{A}$ formalises this intuitive interpretation as follows.

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), $\boxdot\mathcal{A}$ (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 (Section 3.1) that remains polynomial in the size of the initial process (Section 3.2). In Section 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:

$\tilde{P} ::=$	annotated process
$\quad\prod_{i\in I}\pi_i$	multiset of primes
$\pi ::=$	prime
$\quad M[\tilde{P}]$	ambient
$\quad M(o).\tilde{P}$	action, with offset $o \geq 0$
$\quad (x).\tilde{P}$	input
$\quad \langle M\rangle$	output
$\sigma ::= \{x_1\leftarrow M_1\}\cdots\{x_k\leftarrow M_k\}$	sequential substitution, $k \geq 0$
$\langle\tilde{P};\sigma\rangle$	closure

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 and $names(\sigma)$ as 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 $(\tilde{P},\sigma)$:

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

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

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

$$\mathcal{U}((x).\tilde{P},\sigma) = (x).\mathcal{U}(\tilde{P},\sigma)$$

$$\mathcal{U}(\langle M\rangle,\sigma) = \langle M\sigma\rangle$$

We are actually 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 null.

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 1 (Structural Equivalences) *Let $\langle\prod_{i\in I}\pi_i;\sigma\rangle$ be a normal closure. Then*

- $\mathcal{U}(\prod_{i\in I}\pi_i,\sigma) \equiv \mathbf{0}$ *iff* $I = \varnothing$.
- $\mathcal{U}(\prod_{i\in I}\pi_i,\sigma) \equiv M[Q]$ *iff* $\exists M',\tilde{Q} :$ *I is a singleton* $\{i\}$, $\pi_i = M'[\tilde{Q}]$, $M'\sigma = M, \mathcal{U}(\tilde{Q},\sigma) \equiv Q$.
- $\mathcal{U}(\prod_{i\in I}\pi_i,\sigma) \equiv P' \mid P''$ *iff* $\exists J, K :\ J \cup K = I,\ J \cap K = \varnothing,\ P' \equiv \mathcal{U}(\prod_{j\in J}\pi_j,\sigma),\ P'' \equiv \mathcal{U}(\prod_{k\in K}\pi_k,\sigma)$.
- $\mathcal{U}(\prod_{i\in I}\pi_i,\sigma) \equiv \langle M\rangle$ *iff* $\exists M' :$ *I is a singleton* $\{i\}$, $\pi_i = \langle M'\rangle$ *and* $M'\sigma = M$.

- $\mathcal{U}(\prod_{i\in I}\pi_i,\sigma) \equiv (x).P$ *iff* $\exists\tilde{P}$: I *is a singleton* $\{i\}$, $\pi_i = (x).\tilde{P}$ *and* $\mathcal{U}(\tilde{P},\sigma) \equiv P$.

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 treatments are needed.

One can see in the definition of $\mathcal{U}$ that only expressions M in the annotated process are concerned 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 treatments, $nam(M,\sigma)$, consists in recovering from an expression M the name it effectively represents within the substitution σ.

The reduction transition for a closure $\langle\tilde{P};\sigma\rangle$ is more involved and requires some other auxiliary treatments. Those treatments are more specifically dedicated to the case where the substitution applied on the expression M leads to a sequence of capabilities $in M'$, $out M'$, $open M'$: we need first an algorithm to compute the length in terms of capabilities of an expression M with a substitution σ. To keep the algorithm polynomial space, we must simply be able to compute this length without applying explicitly σ on M; this is the role of $len(M,\sigma)$.

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 from a sequence of capabilities, M, a substitution, σ, and an offset, 0, the first capability to execute. This is computed by $fst(M,o,\sigma)$.

The last subroutine introduced here, $split(M(o).\tilde{P},\sigma)$, 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.

The Auxiliary Functions nam, len, fst and $split$:

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

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

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

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

Notice that $nam(M,\sigma)$ is undefined if M is of the form ϵ, $N.N'$, $in\ N$, $out\ N$, or $open\ N$. Therefore, the expression $nam(M,\sigma)$ is either undefined or is evaluated to a name. Moreover, we can compute the name returned by $nam(M,\sigma)$, or whether it is undefined, in linear time. The number returned by $len(M,\sigma)$ can be computed in polynomial space[1]. We can compute the capability returned by $fst(M,o,\sigma)$ and the pair returned by $split(M(o).\tilde{P},\sigma)$, or whether they are undefined, in polynomial space.

The following relations on closures, $\langle\tilde{P};\sigma\rangle \rightarrow \langle\tilde{P}';\sigma'\rangle$ and $\langle\tilde{P};\sigma\rangle \downarrow \langle\tilde{P}';\sigma\rangle$, simulates the reduction and the sublocation relations on processes defined in Section 2.1.

Transitions and Sublocations of Closures:

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

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

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

(Trans I/O)
$$\frac{}{\langle\{(x).\tilde{P},\langle M\rangle\};\sigma\rangle \rightarrow \langle\tilde{P};\{x\leftarrow M\}\sigma\rangle}$$

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

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

(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 unique normal forms. The two rules (Trans Par) and (Trans Amb) reflect the stability under contexts expressed in the definition of the reduction on processes by the rules (Red Par) and (Red Amb). They can also be viewed as a non-deterministic algorithm that guesses a position (or, a redex) where one of the rules (Trans In), (Trans Out), (Trans Open) or (Trans I/O) may be applied. This redex can be guessed using only polynomial space in the size of $\tilde{P}$. Once this redex is guessed, then it is easy to see that testing whether a rule (Trans In), (Trans Out), (Trans Open) or (Trans I/O) has been applied can be done in polynomial space. Therefore, one can decide for two closures $\langle\tilde{P};\sigma\rangle$

[1]We are not interested here in time complexity; a naive algorithm for computing $len(M,\sigma)$, as presented here, runs in exponential time in the worst case. However, it is quite easy to provide a version of this function that runs in polynomial time.

and $\langle \tilde{P}'; \sigma' \rangle$ whether $\langle \tilde{P}; \sigma \rangle \rightarrow \langle \tilde{P}'; \sigma' \rangle$ using only polynomial space in the size of $\langle \tilde{P}; \sigma \rangle$.

In the same way as for ambient processes, we define the relations $\rightarrow^*$ and $\downarrow^*$ (on closures) as the reflexive and transitive closures of $\rightarrow$ and $\downarrow$, respectively.

Proposition 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 $\langle \tilde{P}; \sigma \rangle \rightarrow^* \langle \tilde{Q}; \sigma \rangle$ then $\langle \tilde{Q}; \sigma \rangle$ is normal.*

The next proposition says that the representation of processes as closures preserves sublocations.

Proposition 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 $\mathcal{U}(\tilde{Q}, \sigma) \equiv Q$.*

The following proposition is a counterpart of Proposition 3. It refers to time in the same way as Proposition 3 refers to space.

Proposition 4 (Reduction Equivalences) *Assume $\langle \tilde{P}; \sigma \rangle$ is a normal closure. If $\langle \tilde{P}; \sigma \rangle \rightarrow \langle \tilde{P}'; \sigma' \rangle$ then $\mathcal{U}(\tilde{P}, \sigma) \rightarrow \mathcal{U}(\tilde{P}', \sigma')$. If $\mathcal{U}(\tilde{P}, \sigma) \rightarrow P'$ then there exists $\langle \tilde{P}'; \sigma' \rangle$ such that $\langle \tilde{P}; \sigma \rangle \rightarrow \langle \tilde{P}'; \sigma' \rangle$ and $\mathcal{U}(\tilde{P}', \sigma') \equiv P'$.*

Propositions 1–4 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 in a single node in the tree representation.

Lemma 1 *Suppose that $\langle \tilde{P}; \sigma \rangle \rightarrow \langle \tilde{P}'; \sigma' \rangle$. Then $\|\langle \tilde{P}'; \sigma' \rangle\| \leq \|\langle \tilde{P}; \sigma \rangle\|$.*

Proof By a simple case analysis on the derivation of $\langle \tilde{P}; \sigma \rangle \rightarrow \langle \tilde{P}'; \sigma' \rangle$. In cases (Trans In), (Trans Out) and (Trans Open), the transition either does not change or decreases the representation's size. In case (Trans I/O), the three nodes representing input, output and process composition ($()$, $\langle\rangle$, $.$) together with the representation of x and M are replaced with two nodes representing assignment and substitution composition ($\leftarrow$, $\{\}$) together with the representation of x and M. Thus the tree decreases by one node. ■

Proposition 5 *Suppose that the closure $\langle \tilde{P}; \sigma \rangle$ 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 on the same number of bits, polynomial in $|\langle \tilde{P}; \sigma \rangle|$. Moreover, 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 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. ■

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 Section 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 Section 2 shows, this bound is tight) and that the number of successors for every node is at most polynomial[2]. 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 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 5 and the observation that there are only exponentially many strings of polynomial length. ■

3.3 A New Algorithm

We propose a new algorithm, $\mathit{Check}(\tilde{P}, \sigma, \mathcal{A})$, that check whether the ambient process simulated by $\langle \tilde{P}; \sigma \rangle$ satisfies the closed formula $\mathcal{A}$. We do not consider all the possible closures though and for each ambient process, P, we only consider the closure, $\mathcal{F}(P)$, obtained using the *folding* function defined as follows. We prove (Proposition 8), 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:

$\mathcal{F}(\mathbf{0}) = \{\}$
$\mathcal{F}(P \mid Q) = \mathcal{F}(P) +\!\!\!+ \mathcal{F}(Q)$
$\mathcal{F}(M.P) = \{M(0).\mathcal{F}(P)\}$ if P is not of the form $M'.P'$ for $M' \neq \epsilon$
$\mathcal{F}(M[P]) = \{M[\mathcal{F}(P)]\}$
$\mathcal{F}((n).P) = \{(n).\mathcal{F}(P)\}$
$\mathcal{F}(\langle M \rangle) = \{\langle M \rangle\}$

For any process P, the closure $\langle \mathcal{F}(P); \iota \rangle$ is normal and $\mathcal{U}(\mathcal{F}(P), \iota)$ is structurally equivalent 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 $\tilde{P}$. We denote by $\mathit{fn}(\tilde{P}, \sigma)$ the set $(\mathit{fn}(\tilde{P}) \cup \mathit{names}(\sigma)) \setminus \mathit{dom}(\sigma)$.

Computing whether a process satisfies a closed formula:

$\mathit{Check}(\tilde{P}, \sigma, \mathbf{T}) = \mathbf{T}$
$\mathit{Check}(\tilde{P}, \sigma, \neg\mathcal{A}) = \neg\mathit{Check}(\tilde{P}, \sigma, \mathcal{A})$
$\mathit{Check}(\tilde{P}, \sigma, \mathcal{A} \vee \mathcal{B}) = \mathit{Check}(\tilde{P}, \sigma, \mathcal{A}) \vee \mathit{Check}(\tilde{P}, \sigma, \mathcal{B})$

$$\mathit{Check}(\textstyle\prod_{i \in I} \pi_i, \sigma, \mathbf{0}) = \begin{cases} \mathbf{T} & \text{if } I = \varnothing \\ \mathbf{F} & \text{otherwise} \end{cases}$$

$$\mathit{Check}(\textstyle\prod_{i \in I} \pi_i, \sigma, n[\mathcal{A}]) = \begin{cases} \mathit{Check}(\tilde{Q}, \sigma, \mathcal{A}) & \text{if } I = \{i\},\ \pi_i = M[\tilde{Q}],\ \mathit{nam}(M, \sigma) = n \\ \mathbf{F} & \text{otherwise} \end{cases}$$

$$\mathit{Check}(\textstyle\prod_{i \in I} \pi_i, \sigma, \mathcal{A} \mid \mathcal{B}) = \begin{cases} \mathbf{T} & \text{if } \exists J, K. J \cup K = I \wedge J \cap K = \varnothing \wedge \\ & \quad \mathit{Check}(\prod_{j \in J} \pi_j, \sigma, \mathcal{A}) \wedge \mathit{Check}(\prod_{k \in K} \pi_k, \sigma, \mathcal{B}) \\ \mathbf{F} & \text{otherwise} \end{cases}$$

[2]Note that the process $n[\mathit{in}\ n.P_0] \mid \ldots \mid n[\mathit{in}\ n.P_k]$ has k^2 successors.

$Check(\tilde{P}, \sigma, \exists x.\mathcal{A}) =$
  let $\{m_1, \ldots, m_k\} = fn(\tilde{P}, \sigma) \cup fn(\mathcal{A})$ in
  let $m_0 \notin \{m_1, \ldots, m_k\} \cup bn(\tilde{P}) \cup dom(\sigma)$ be fresh in
  $\begin{cases} \mathbf{T} & \text{if } Check(\tilde{P}, \sigma, \mathcal{A}\{x \leftarrow m_i\}) \text{ for some } i \in 0..k \\ \mathbf{F} & \text{otherwise} \end{cases}$
$Check(\tilde{P}, \sigma, \Diamond\mathcal{A}) =$
  $Check(\tilde{P}, \sigma, \mathcal{A}) \vee \bigvee_{\langle \tilde{P};\sigma\rangle \to \langle \tilde{P}';\sigma'\rangle} Check(\tilde{P}', \sigma', \Diamond\mathcal{A})$
$Check(\tilde{P}, \sigma, \diamondsuit\mathcal{A}) =$
  $Check(\tilde{P}, \sigma, \mathcal{A}) \vee \bigvee_{\langle \tilde{P};\sigma\rangle \downarrow \langle \tilde{P}';\sigma\rangle} Check(\tilde{P}', \sigma, \diamondsuit\mathcal{A})$
$Check(\tilde{P}, \sigma, \mathcal{A}@n) = Check(n[\tilde{P}], \sigma, \mathcal{A})$

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

Proposition 8 *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 and 4 in the cases $\diamondsuit\mathcal{A}$ and $\Diamond\mathcal{A}$, respectively. ■

Theorem 1 *Model-checking for the ambient calculus and logic of this paper is decidable in PSPACE.*

Proof We only give a sketch of the proof. Deciding $Check(\tilde{P}, \sigma, \mathcal{A}) = \mathbf{T}$ can be done by means of a non-deterministic algorithm which uses only polynomial-space in the size of $\langle \tilde{P}; \sigma\rangle$ and $\mathcal{A}$. This holds in particular for the closure $\langle \mathcal{F}(P); \iota\rangle$.

Intuitively, this non-deterministic algorithm either (in the case of $\mathbf{T}, \mathbf{0}, n[\mathcal{A}], \mathcal{A}@n$) directly checks whether the respective conditions hold, or (in the case of $\mathcal{A} \vee \mathcal{B}, \mathcal{A} \mid \mathcal{B}, \exists x.\mathcal{A}, \Diamond\mathcal{A}, \diamondsuit\mathcal{A}$) simply guesses which of the disjuncts in the definition of *Check* holds. The only more difficult case is that of negation; here we apply the Immerman's theorem (NSPACE(n) = co-NSPACE(n)). Then, using the general statement of Savitch's theorem (NPSPACE(n) $\subseteq$ PSPACE(n^2)), this non-deterministic algorithm can be turned into a deterministic one. Finally, the fact that $\mathcal{F}(P)$ is polynomial in the size of P and the statement of Proposition 8 complete the proof. ■

Discussion. The algorithm above relies on two non-trivial results from computational complexity theory, namely the theorems by Savitch and Immerman. A direct implementation of the algorithm as it is could be quite impractical. It is possible to improve this by introducing to the logic dual constructs: conjunction, universal quantification, everytime and everywhere modalities, constructs dual to names, parallel composition etc. Then, every formula of the logic can be transformed to a positive normal form that does not contain negation at all — in this way we eliminate the use of Immerman's theorem. It is then not difficult to modify the algorithm in such a way that the use of Savitch's theorem is limited to the reachability subproblem (more precisely, to the problem of enumerating for a given closure $\langle \tilde{P}; \sigma\rangle$ all closures $\langle \tilde{P}'; \sigma'\rangle$ such that $\langle \tilde{P}; \sigma\rangle \to^* \langle \tilde{P}'; \sigma'\rangle$). Due to the lack of space we do not present the details here.

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): we can assume without loss of generality that these Boolean formulas are in prenex and conjunctive normal form. 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 $\Sigma_1^P = NP$)

We will use the following formula as a running example of a valid closed QBF formula:

$$\forall v_1.\exists v_2.\exists v_3.(v_1 \vee \overline{v_2} \vee v_3) \wedge (\overline{v_1} \vee v_2 \vee v_3) \wedge \overline{v_3}$$

4.1 The Full Calculus and Logic

We define an encoding of QBF formulas into ambient formulas. This encoding is then used to prove (Theorem 2) the complexity of model-checking the ambient logic.

In our encoding, we assume that the truth values tt and ff used in the definition of QBF satisfaction are distinct ambient calculus names. We also use a derived operator for name equality. Recall an encoding of name equality in the ambient logic [4]:

$$\eta = \mu \quad \triangleq \quad \eta[\mathbf{T}]@\mu$$

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.

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

Encoding QBF Formulas as Ambient Logic Formulas:

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

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

Lemma 2 *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 2 *Expression complexity of checking the full logic (including name quantification) is PSPACE-hard.*

Proof Straightforward from Lemma 2 since for the fixed ambient process $\mathbf{0}$ solving the model-checking problem $\mathbf{0} \models \varphi$ 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. It might suggest that the complexity of the model-checking problem comes from the quantification in the logic. Below we show that it is not the case: the problem remains so 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

Encoding QBF Formulas as Ambient Processes and Formulas:

$$
\begin{aligned}
&\llbracket v \rrbracket = v[pos[\mathbf{0}] \mid v'[\mathbf{0}]] \mid \mathbf{T}\\
&\llbracket \overline{v} \rrbracket = v[neg[\mathbf{0}] \mid v'[\mathbf{0}]] \mid \mathbf{T}\\
&\llbracket \ell_1 \vee \cdots \vee \ell_k \rrbracket = \llbracket \ell_1 \rrbracket \vee \cdots \vee \llbracket \ell_k \rrbracket\\
&\\
&\llbracket C_1 \wedge \cdots \wedge C_k \rrbracket = (end[], \llbracket C_1 \rrbracket \wedge \cdots \wedge \llbracket C_k \rrbracket)\\
&\llbracket \forall v.\varphi \rrbracket = (v'[in\ v.n[out\ v'.out\ v.P]],\\
&\qquad\qquad \Box((n[\mathbf{T}] \mid \mathbf{T}) \Rightarrow \mathcal{A}))\\
&\qquad\qquad\quad \text{where } (n[P], \mathcal{A}) = \llbracket \varphi \rrbracket\\
&\llbracket \exists v.\varphi \rrbracket = (v'[in\ v.n[out\ v'.out\ v.P]],\\
&\qquad\qquad \Diamond((n[\mathbf{T}] \mid \mathbf{T}) \wedge \mathcal{A}))\\
&\qquad\qquad\quad \text{where } (n[P], \mathcal{A}) = \llbracket \varphi \rrbracket\\
&\\
&enc(\varphi) = (v_1[pos[]] \mid v_1[neg[]] \mid \cdots \mid v_n[pos[]] \mid v_n[neg[]] \mid P, \mathcal{A})\\
&\qquad\qquad \text{where } (P, \mathcal{A}) = \llbracket \varphi \rrbracket\\
&\qquad\qquad \text{and } \varphi = Q_1 v_1. \ldots .Q_n v_n.C_1 \wedge \cdots \wedge C_k\\
&\qquad\qquad \text{where each } Q_i \in \{\exists, \forall\}.
\end{aligned}
$$

For the formula φ given above as an example, one would obtain $enc(\varphi) = (P, \mathcal{A})$ where P is depicted in Figure 1(a).

We explain this encoding with reference to the ambient process depicted in Figure 1(a). The ambients whose names range over v_i describe an interpretation for the Boolean variables v_i whereas the ambients named v_i' are the "material" to extend this interpretation. In the initial ambient, the ambients v_i encode the empty interpretation and the material is in an ambient named v_1' marking the fact that v_1 is the first variable to treat. The first step of reduction will move the ambient v_1' non-deterministically either inside $v_1[pos[]]$ (the Boolean variable v_1 takes the value tt) or inside $v_1[neg[]]$ (the Boolean variable v_1 takes the value $f\!f$). The next two steps of reduction are deterministic. They aim to leave a mark in one of the ambients v_1 according to the first non-deterministic choice and to reach a situation in which the Boolean variable v_2 is considered. For instance, if the first choice was to instantiate v_1 with tt then, one would obtain a parallel composition of $v_1[pos[] \mid v_1'[]]$ and $v_1[neg[]]$. The ambients named v_2, v_3 are kept unchanged and the ambient containing the rest of the interpretation would be of the form $v_2'[in\ v_2.v_3'[Q]]$ where Q is the internal of v_3' in the initial process. This computation, consisting of one non-deterministic step followed by two deterministic ones, can be carried on for the variables v_2 and v_3. Then, when no more reduction step is possible, the resulting process is a parallel composition of the empty ambient $end[]$ and, for each i, of $v_i[n[] \mid v_i'[]]$ and $v_i[n'[]]$ where n, n' are distinct elements from $\{pos, neg\}$. For instance, the irreducible process given in Figure 1(b) represents the interpretation $v_1 \mapsto tt, v_2 \mapsto tt, v_3 \mapsto f\!f$.

For the encoding of φ, the resulting ambient formula $\mathcal{A}$ will be of the form

$$\Box((v_2'[\mathbf{T}] \mid \mathbf{T}) \Rightarrow \Diamond((v_3'[\mathbf{T}] \mid \mathbf{T}) \wedge \Diamond((end[\mathbf{T}] \mid \mathbf{T}) \wedge \mathcal{B})))$$

where $\mathcal{B} = \llbracket v_1 \vee \overline{v_2} \vee v_3 \rrbracket \wedge \llbracket \overline{v_1} \vee v_2 \vee v_3 \rrbracket \wedge \llbracket \overline{v_3} \rrbracket$.

We have said that the ambient processes encode interpretations. The Boolean formula itself is encoded in the ambient formula $\mathcal{A}$. Once no more reduction step is possible on the ambient process, this latter represents an interpretation whose domain is the set of all variables in φ: this interpretation is given by the places where the marks v_i' have been put. It is easy with an ambient formula to test whether this interpretation renders true the quantifier-free part of φ. This role is played by the ambient formula $\mathcal{B}$ whereas the remaining part of $\mathcal{A}$ aims to encode the quantifiers of φ.

Let us first consider the outermost quantifier $\forall v_1$ in φ: this quantification stands for "for all possible interpretations of the variable v_1". We have described above the mechanism for the instantiation of the Boolean variable v_1 in the ambient process. It consists of first a non-deterministic step, then two deterministic steps. Whatever the first step is, those three steps lead to a situation where the ambient process is of the form $R \mid v_2'[R']$. It should be noticed that those two processes (one for each possibility of the first step) are the only processes of this form reachable from the initial process. Therefore, the statement "for all possible interpretations of the variable v_1" can be translated as "for all processes of the form $R \mid v_2'[R']$ reachable from the initial process". This rephrased statement can be expressed in the ambient logic as $\Box((v_2'[\mathbf{T}] \mid \mathbf{T}) \Rightarrow \ldots)$.

A dual reasoning can be applied then for $\exists v_2$, the following quantification of the formula φ. In that case, the statement "there exists an interpretation for the variable v_2" is translated into "there exists an ambient process of the form $T \mid v_3'[T']$ reachable from the current process". This current process is one of the two processes after the instantiation of the variable v_2, that is of the form $S \mid v_3'[S']$. This statement can be expressed by means of the ambient logic by the formula $\Diamond((v_3'[\mathbf{T}] \mid \mathbf{T}) \wedge \ldots)$. Finally, the quantification $\exists v_3$ is expressed by $\Diamond((end[\mathbf{T}] \mid \mathbf{T}) \wedge \ldots)$.

Lemma 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 3 *Combined 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 3, taking into account that for $enc(\varphi) = (P, \mathcal{A})$, both P and $\mathcal{A}$ are of polynomial size with respect to φ. ■

4.3 Immobile Ambients With I/O, No Quantifiers

We let for any ambient name v_i',

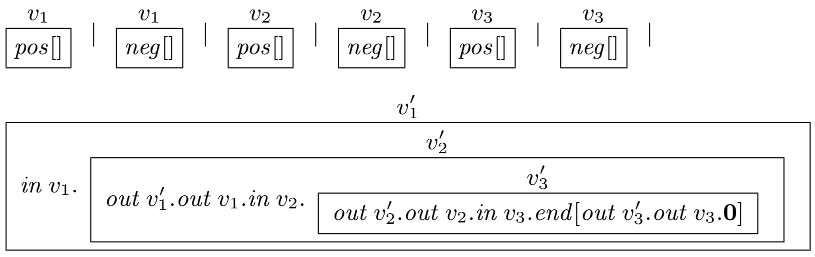
$$Inst(v_i') \triangleq v_i'[\mathbf{T}] \mid \mathbf{T} \quad Inst^+(v_i') \triangleq v_i'[v_i''[\mathbf{T}] \mid \mathbf{T}] \mid \mathbf{T}$$

and for the name end,

$$Inst(end) \triangleq end[\mathbf{T}] \mid \mathbf{T} \quad Inst^+(end) \triangleq end[end'[\mathbf{T}] \mid \mathbf{T}] \mid \mathbf{T}$$

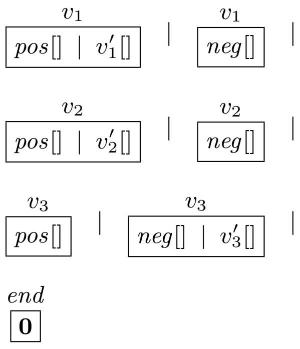
Encoding QBF Formulas as Ambient Processes and Formulas:

$$
\begin{aligned}
&\llbracket v \rrbracket = v[]\\
&\llbracket \overline{v} \rrbracket = \overline{v}[]\\
&\llbracket \ell_1 \vee \ldots \vee \ell_k \rrbracket = \llbracket \ell_1 \rrbracket \mid \ldots \mid \llbracket \ell_k \rrbracket\\
&\\
&enc(C_1 \wedge \ldots \wedge C_k) = (end[C[\,\llbracket C_1 \rrbracket\,] \mid \ldots \mid C[\,\llbracket C_k \rrbracket\,]],\\
&\qquad\qquad\qquad\qquad \text{❏}((C[\mathbf{T}] \mid \mathbf{T}) \Rightarrow C[tt[\mathbf{0}] \mid \mathbf{T}] \mid \mathbf{T}))\\
&enc(\exists v.\varphi) = (v'[\langle tt \rangle \mid \langle f\!f \rangle \mid (v).(v''[] \mid (\overline{v}).n[P])],\\
&\qquad\qquad \mathbf{T} \mid v'[\Diamond(\ (Inst(n) \wedge \neg Inst^+(n))\ \wedge \mathcal{A}\ )])\\
&\qquad\qquad \text{where } enc(\varphi) = (n[P], \mathcal{A})
\end{aligned}
$$



v_1 $pos[]$ | v_1 $neg[]$ | v_2 $pos[]$ | v_2 $neg[]$ | v_3 $pos[]$ | v_3 $neg[]$ |
v'_1
$in\ v_1.$
v'_2
$out\ v'_1.out\ v_1.in\ v_2.$
v'_3
$out\ v'_2.out\ v_2.in\ v_3.end[out\ v'_3.out\ v_3.\mathbf{0}]$

(a) The process P in $enc(\varphi) = (P, \mathcal{A})$



v_1 $pos[] \mid v'_1[]$ | v_1 $neg[]$ |
v_2 $pos[] \mid v'_2[]$ | v_2 $neg[]$ |
v_3 $pos[]$ | v_3 $neg[] \mid v'_3[]$ |
end $\mathbf{0}$

(b) The irreducible process for the interpretation $v_1 \mapsto tt, v_2 \mapsto tt, v_3 \mapsto ff$

Figure 1: Encoding for mobile ambients without I/O, no quantifiers

$$\begin{aligned} enc(\forall v.\varphi) = (&v'[\langle tt\rangle \mid \langle ff\rangle \mid (v).(v''[] \mid (\overline{v}).n[P])], \\ &\mathbf{T} \mid v'[\Box(\ (Inst(n) \wedge \neg Inst^+(n))\ \Rightarrow \mathcal{A}\ )]) \\ &\text{where } enc(\varphi) = (n[P], \mathcal{A}) \end{aligned}$$

For the formula φ used in our example, one would have $enc(\varphi) = (P, \mathcal{A})$, where P is depicted in Figure 2(a).

The key idea of this encoding is to use (reductions of) communications for performing the instantiation of the quantifier-free part of φ with respect to some interpretation. Therefore, the quantifier-free formula $C_1 \wedge \ldots \wedge C_k$ is encoded in the ambient process itself, inside an ambient named end. For instance, in Figure 2(a) for our example, the ambient $end[C[v_1[] \mid \overline{v_2}[]v_3[]] \mid C[\overline{v_1}[] \mid v_2[]v_3[]] \mid C[\overline{v_3}[]]]$ encodes the quantifier-free part of φ: the ambient end contains a sub-ambient called C for each clause C_i in φ and the ambient corresponding to C_i contains an ambient $\ell_j\,[]$ for each literal ℓ_j from C_i.

Starting from P described in Figure 2(a), let us inspect the behaviour of processes through reductions. Two reductions can be performed on P: one establishes a communication between $\langle tt\rangle$ and (v_1) and the other one between $\langle ff\rangle$ and (v_1). Once this reduction step is performed the name v_1 has been replaced by either tt or ff uniformly at every position and in particular in the ambient named end. Hence, the first step of computation is non-deterministic and instantiates the literal v_1. It has also a side-effect: it reveals an ambient process $v''_1[]$ within the ambient v'_1; this process is a marker for the control of computations. Its precise role will be explained later on. The second step is deterministic: for each first step, only one second step is possible. This second step aims to instantiate the literal $\overline{v_1}$ according to the instantiation of v_1. Indeed, if the first communication has consumed the output $\langle tt\rangle$ then for the second one only the output $\langle ff\rangle$ remains and vice-versa. So, after the second step, the name $\overline{v_1}$ is globally replaced by a Boolean value. Moreover, at this point there are no more actions prefixing the ambient named v'_2 and so this ambient can be now reduced using the rules (Red Par) and (Red Amb). The next reduction steps are performed in a similar way: a non-deterministic step follows by a deterministic one. This leads finally to replacing in the ambient end all the names corresponding to literals by Boolean values tt and ff. As an example, in Figure 2(b), we have depicted the ambient end once the reductions corresponding to the interpretation $\mathcal{M} = v_1 \mapsto tt, v_2 \mapsto tt, v_3 \mapsto ff$ have been performed.

Now, using an ambient formula it is not difficult to test whether the interpretation induced from the process in Figure 2(b) is a model for $C_1 \wedge C_2 \wedge C_3$: as $C_1 \wedge C_2 \wedge C_3$ is in conjunctive normal form, $\mathcal{M}$ is a model for it if and only if $\mathcal{M}$ renders at least one literal true in every clause C_i. According to the way reductions are performed and correspond to instantiations, this is equivalent to the claim that in the process from Figure 2(b), every ambient named C contains a sub-ambient $tt[]$. This can be tested with the formula $\mathcal{B} = \Box((C[\mathbf{T}] \mid \mathbf{T}) \Rightarrow C[tt[\mathbf{0}] \mid \mathbf{T}] \mid \mathbf{T})$ and exactly this formula comes out in the encoding $enc(C_1 \wedge C_2 \wedge C_3)$.

In the encoding $enc(\varphi) = (P, \mathcal{A})$, one part of $\mathcal{A}$ aims to test whether the interpretation corresponding to the reductions is a model of φ. The other part of $\mathcal{A}$ is used to encode the quantification of φ. Let us illustrate on our example the ideas of this encoding: for the formula φ from our example, the formula $\mathcal{A}$ is equal to

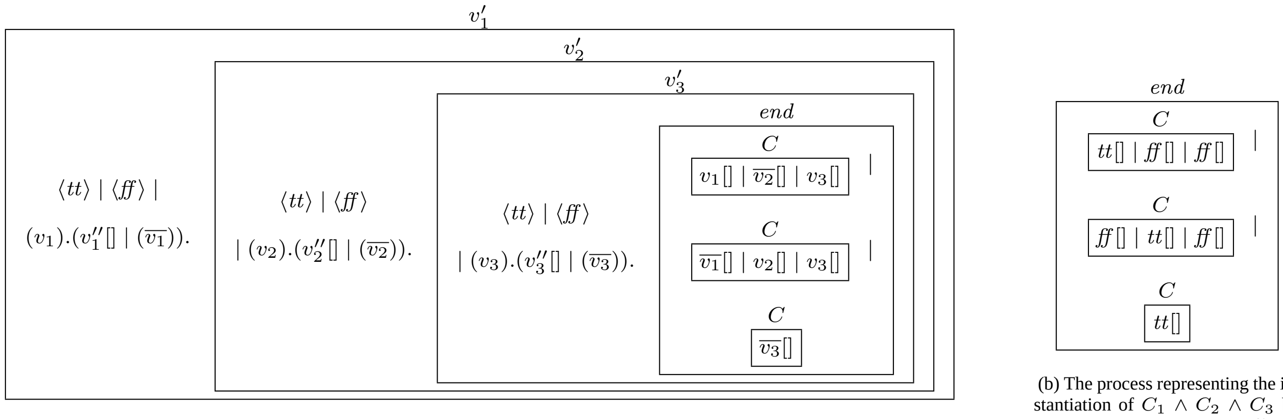
$$\begin{aligned} &\mathbf{T} \mid v'_1[\Box(\ (Inst(v'_2) \wedge \neg Inst^+(v'_2))\ \Rightarrow \\ &\quad (\mathbf{T} \mid v'_2[\Diamond(\ (Inst(v'_3) \wedge \neg Inst^+(v'_3)) \wedge \\ &\qquad (\mathbf{T} \mid v'_3[\Diamond(Inst(end) \wedge \neg Inst^+(end)\ \wedge \mathcal{B})])\ )])\ )] \end{aligned}$$

where $\mathcal{B}$ is the result of the encoding of the quantifier-free part of φ. For the variable v_i, the intuitive reading of $Inst(v'_i)$ is "the next variable to consider is v_i", that is, the instantiation of the variable v_{i-1} has been completed. The reading of $Inst^+(v'_i)$ is "the variable v_i has been partially treated", that is, the instantiation has been performed for the positive literal v_i. For the ambient name end, $Inst(end)$ refers to the completion of the instantiation of the variable v_n.

The first quantification $\forall v_1$ stands for "for all possible interpretations of the variable v_1" and the part of φ related with this quantification is

$$\mathbf{T} \mid v'_1[\Box(\ (Inst(v'_2) \wedge \neg Inst^+(v'_2))\ \Rightarrow \ldots)]$$

This formula is model-checked against the process P given in Figure 2(a). As $P \equiv \mathbf{0} \mid P$, the model-checking problem is reduced to checking the interior of v'_1 against the sub-formula of the form $\Box\mathcal{A}_1$: all processes reachable from the interior of v'_1 must satisfy $\mathcal{A}_1$. Let us have a look at the form of those reachable processes: the interior of v'_1 is itself reachable as well as the two processes corresponding to the instantiation of the literal v_1


v_1'
v_2'
v_3'
end
$\langle tt\rangle \mid \langle f\!f\rangle \mid$
$(v_1).(v_1''[] \mid (\overline{v_1})).$
$\langle tt\rangle \mid \langle f\!f\rangle$
$\mid (v_2).(v_2''[] \mid (\overline{v_2})).$
$\langle tt\rangle \mid \langle f\!f\rangle$
$\mid (v_3).(v_3''[] \mid (\overline{v_3})).$
C
$v_1[] \mid \overline{v_2}[] \mid v_3[]$
$\mid$
C
$\overline{v_1}[] \mid v_2[] \mid v_3[]$
$\mid$
C
$\overline{v_3}[]$
end
C
$tt[] \mid f\!f[] \mid f\!f[]$
$\mid$
C
$f\!f[] \mid tt[] \mid f\!f[]$
$\mid$
C
$tt[]$

(a) The process P in $enc(\varphi) = (P, \mathcal{A})$

(b) The process representing the instantiation of $C_1 \wedge C_2 \wedge C_3$ by $v_1 \mapsto tt, v_2 \mapsto tt, v_3 \mapsto f\!f$

Figure 2: Encoding for immobile ambients with I/O, no quantifiers

(reachable in one step). In those processes v_1 has been replaced by a Boolean value but none of them satisfies $v_2'[\mathbf{T}] \mid \mathbf{T}$, that is, $Inst(v_2')$. Now, the processes reachable in two steps or more indeed satisfy the formula $Inst(v_2')$; but the ones reachable in exactly two steps can be distinguished from the others since these former are the only ones which do not satisfy $v_2'[v_2''[\mathbf{T}] \mid \mathbf{T}] \mid \mathbf{T}$, that is, $Inst^+(v_2')$. Indeed, steps beyond the second one reveal the marker $v_2''[]$ inside the ambient v_2'. We have already mentioned the fact that the two steps of computation correspond exactly to the complete treatment of the variable v_1 which is the intended meaning of $Inst(v_2') \wedge \neg Inst^+(v_2')$. Therefore, model-checking continues by checking the two processes (the second step of computation being deterministic), defined as the interior of v_1' in which the literals v_1 and $\overline{v_1}$ have been replaced by Boolean values, against the formula

$$\mathbf{T} \mid v_2'[\Diamond(\ (Inst(v_3') \wedge \neg Inst^+(v_3'))\ \wedge ...)]$$

from the encoding of the quantification $\exists v_2$. It stands for "there exists an interpretation for v_2". The process that is checked against this formula is of the form $v_1''[] \mid v_2'[R]$. Therefore, it amounts to check whether the process R, which is the interior of v_2' in which names $v_1, \overline{v_1}$ have been replaced with Boolean values, is a model for the sub-formula of the form $\Diamond\mathcal{A}_2$. Equivalently, there must exist a process reachable from R which satisfies $\mathcal{A}_2$. Let us inspect the processes reachable from R. Of course, R itself is reachable as well as the two processes reachable in one step of computation performing the instantiation for the literal v_2. None of these processes satisfies the formula $v_3'[\mathbf{T}] \mid \mathbf{T}$, that is, $Inst(v_3')$. Processes that are obtained with two steps or more from R do satisfy $Inst(v_3')$ but only those obtained by strictly more than two steps reveal the marker $v_3''[]$ inside v_3' and thus, satisfy $v_3'[v_3''[\mathbf{T}] \mid \mathbf{T}] \mid \mathbf{T}$, that is $Inst^+(v_3')$. Those computations from R of exactly two steps correspond to the complete treatment of the variable v_2 and satisfy $Inst(v_3') \wedge \neg(Inst^+(v_3'))$. So, model-checking carries on by checking that one of these two processes reachable from R in two steps and defined as the interior of v_2 in which the literals v_1, $\overline{v_1}$, v_2, $\overline{v_2}$ have been replaced by Boolean values, is a model for the remaining part of the encoding of the formula.

Finally, the quantification $\exists v_3$ is encoded as

$$\mathbf{T} \mid v_3'[\Diamond(\ ((\mathbf{T} \mid end[\mathbf{T}]) \wedge \neg(\mathbf{T} \mid end[end'[\mathbf{T}] \mid \mathbf{T}]))\ \wedge ...)]$$

and its treatment is similar to that of $\exists v_2$. It leads to model-checking the process named end given in Figure 2(b) against the formula $\mathcal{B}$.

Lemma 4 *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 *Combined complexity of model-checking immobile ambients with I/O against the quantifier-free logic is PSPACE-hard.*

Proof Straightforward from the PSPACE-completeness of the validity for QBF, from Lemma 4 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. Our previous encoding is based on an individual treatment for the variables in the quantification. The improved encoding will be based on the alternation of quantifiers: roughly, $\exists v_2 \exists v_3$ can be grouped together by saying that "there exists an interpretation for v_2 and v_3". As far as the previous encoding is concerned, the ambient formula resulting from the encoding of $\exists v_2 \exists v_3$ will test for two successive reachabilities; this can be modified in such a way that only one test of reachability is performed. This will imply for the new encoding that the markers used to control the model-checking (namely, the ambients v') will no longer be associated with the variables but with the alternation of quantifiers. Those ambient names will range over a_i where i is an integer. We define for those a_i's:

$$Inst(a_i) \triangleq a_i[\mathbf{T}] \mid \mathbf{T} \qquad\qquad Inst^+(a_i) \triangleq a_i[a_i[] \mid \mathbf{T}] \mid \mathbf{T}$$

The Revised Encoding:

$$enc(\forall v.\varphi) = enc(\forall v.\varphi, 1)$$
$$enc(\exists v.\varphi) = enc(\exists v.\varphi, 1)$$

$$\begin{aligned} enc(\forall v.\varphi, i) = (&a_i[\langle tt\rangle \mid \langle f\!f\rangle \mid (v).(a_i[] \mid (\overline{v})).P,\\ &\mathbf{T} \mid a_i[\Box(\ Inst(a_{i+1}) \wedge Inst^+(a_{i+1})\ \Rightarrow \mathcal{A}\ )])\\ &\quad \text{where } enc_\forall(\varphi, i) = (P, \mathcal{A})\\ enc(\exists v.\varphi, i) = (&a_i[\langle tt\rangle \mid \langle f\!f\rangle \mid (v).(a_i[] \mid (\overline{v})).P,\\ &\mathbf{T} \mid a_i[\Diamond(\ Inst(a_{i+1}) \wedge Inst^+(a_{i+1})\ \wedge \mathcal{A}\ )])\\ &\quad \text{where } enc_\exists(\varphi, i) = (P, \mathcal{A}) \end{aligned}$$

$$\begin{aligned} enc_\forall(\exists v.\varphi, i) &= enc(\exists v.\varphi, i+1)\\ enc_\forall(\forall v.\varphi, i) &= (\langle tt\rangle \mid \langle f\!f\rangle \mid (v).(\overline{v}).P, \mathcal{A})\\ &\quad \text{where } enc_\forall(\varphi, i) = (P, A)\\ enc_\exists(\forall v.\varphi) &= enc(\forall v.\varphi, i+1)\\ enc_\exists(\forall v.\varphi, i) &= (\langle tt\rangle \mid \langle f\!f\rangle \mid (v).(\overline{v}).P, \mathcal{A})\\ &\quad \text{where } enc_\exists(\varphi, i) = (P, A) \end{aligned}$$

$$\begin{aligned}&enc(C_1 \wedge \ldots \wedge C_k, i) = (a_i[C[\,[\![C_1]\!]\,] \mid \ldots \mid C[\,[\![C_k]\!]\,]],\\ &\qquad\qquad\qquad\qquad \Box((C[\mathbf{T}] \mid \mathbf{T}) \Rightarrow C[tt[\mathbf{0}] \mid \mathbf{T}] \mid \mathbf{T}))\\ &[\![\ell_1 \vee \ldots \vee \ell_k]\!] = [\![\ell_1]\!] \mid \ldots \mid [\![\ell_k]\!]\\ &[\![v]\!] = v[]\\ &[\![\overline{v}]\!] = \overline{v}[]\end{aligned}$$

The statement of Lemma 4 still holds for this new encoding. Furthermore, in the encoding $(P, \mathcal{A})$ of the Boolean formula φ, the ambient logic formula $\mathcal{A}$ depends only on the alternation depth and the outermost quantifier of φ; for any two Boolean formulas φ, φ' having the same alternation depth k and the same outermost quantifier Q, if $enc(\varphi) = (P, \mathcal{A})$ and $enc(\varphi') = (P', \mathcal{A}')$ then $\mathcal{A} = \mathcal{A}'$.

Theorem 5 *For every integer k there exists a formula $\mathcal{A}_k^{\exists}$ such that the program complexity of model checking for $\mathcal{A}_k^{\exists}$ is Σ_k^P-hard.*

Proof Let $\mathcal{A}_k^{\exists}$ be the formula such that for any closed quantified Boolean formula φ of alternation depth k whose outermost quantifier is $\exists$, $enc(\varphi) = (P_\varphi, \mathcal{A}_k^{\exists})$. Due to the remark above, we know that this formula exists and furthermore, is of size polynomial in k.

Now, by Lemma 4, every instance of the validity problem for a closed quantified Boolean formula φ of alternation depth k whose outermost quantifier is $\exists$ can be reduced to the model-checking problem $P_\varphi \models \mathcal{A}_k^{\exists}$ for $enc(\varphi) = (P_\varphi, \mathcal{A}_k^{\exists})$. Thus, since the size of P_φ is polynomial in the size of φ, the theorem follows. ■

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 non-deterministic algorithm to prove the PSPACE upper bound. We also briefly discuss how to adapt this algorithm to obtain a more practical implementation.

An important property obtained using closures is that there exist at most exponentially many (non structurally equivalent) 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/or 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. In *Proceedings FoSSaCS'98*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer, 1998. Accepted for publication in *Theoretical Computer Science*.

[2] L. Cardelli and A.D. Gordon. Types for mobile ambients. In *Proceedings POPL'99*, pages 79–92. ACM, January 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 Microsoft Research Technical Report MSR–TR–99–11, April 1999.

[4] L. Cardelli and A.D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *Proceedings POPL'00*, pages 365–377. ACM, January 2000.

[5] L. Cardelli and G. Ghelli. A query language for semistructured data based on the ambient logic. Submitted for publication, 2000.

[6] L. Cardelli and A.D. Gordon. Logical properties of name restriction. Submitted for publication, 2000.

[7] W. Charatonik, S. Dal Zilio, Gordon A. D., S. Mukhopadhyay, and J.-M. Talbot. The complexity of model checking mobile ambients. Extended version, available at `http://research.microsoft.com/~sdal/`, 2000.

[8] O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of ACM*, 47(2):312–360, 2000.

[9] D. Sangiorgi. Extensionality and intensionality of the ambient logics. In *Proceedings POPL'01*. ACM, January 2001. To appear.

[10] W.J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, April 1970.

[11] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, October 1976.