The Decidability of Model Checking Mobile Ambients

Witold Charatonik[1,2] and Jean-Marc Talbot[3]

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

Abstract. The ambient calculus is a formalism for describing the mobility of both software and hardware. The ambient logic is a modal logic designed to specify properties of distributed and mobile computations programmed in the ambient calculus. In this paper we investigate the border between decidable and undecidable cases of model checking mobile ambients for some fragments of the ambient calculus and the ambient logic.
Recently, Cardelli and Gordon presented a model-checking algorithm for a fragment of the calculus (without name restriction and without replication) against a fragment of the logic (without composition adjunct) and asked the question, whether this algorithm could be extended to include either replication in the calculus or composition adjunct in the logic. Here we answer this question negatively: it is not possible to extend the algorithm, because each of these extensions leads to undecidability of the problem. On the other hand, we extend the algorithm to the calculus with name restriction and logic with new constructs for reasoning about restricted names.

1 Introduction

The ambient calculus [6, 4, 2] is a process calculus for modeling mobile computations and mobile devices; one can describe the mobility of both software and hardware in this formalism. An ambient is a named cluster of running processes and nested subambients. 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 [5] 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 over the time, the logic includes spatial modalities for describing the tree structure of processes. In a recent paper, Cardelli and Gordon extend the logic with the constructs for describing private names [7]. Other work on the ambient logic includes a study of the process equivalence induced by the satisfaction relation [11] and the use of spatial modalities to describe the tree structure of semistructured databases [1].

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 [5] give a model-checking algorithm for the fragment of the calculus in which the processes contain no replications and no dynamic name generation against a fragment of the logic in which formulas contain no composition adjunct. It was then proved in [8] that model checking of this fragment of the calculus against this fragment of the logic is PSPACE-complete.

Our results. Cardelli and Gordon raised in [5] the question whether their algorithm for model-checking could be extended to include either replication in the calculus or composition adjunct in the logic. Here we answer this question negatively: it is not possible to extend the algorithm, because each of these extensions leads to undecidability. On the other hand, we show that the restriction to public names was not necessary: model checking remains decidable for the replication-free fragment with dynamic name generation, even if we extend the logic with constructs from [7] for reasoning about restricted names. Moreover, this extension does not increase the complexity of the problem: one can obtain PSPACE algorithm by combining the abstract algorithm presented in Section 5 with the representation of processes proposed in [8].

We start by recalling the calculus and the logic in the next section. Then we prove the undecidability of the problem for the two mentioned extensions: in Section 3 for the case of the calculus with replication and in Section 4 for the case of the logic with composition adjunct. Finally, in Section 5 we present the model-checking algorithm extended to the case of the calculus with name restriction and the logic with constructs for reasoning about these names.

2 Review of the Ambient Calculus and Logic

In this section we recall the ambient calculus and logic from [6, 5, 7].

2.1 The Ambient Calculus

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

Processes and Capabilities:

$P ::=$	processes	$M ::=$	capabilities
$\mathbf{0}$	inactivity	n	name
$M[P]$	ambient	$\mathtt{in}\ M$	can enter M
$P \mid Q$	composition	$\mathtt{out}\ M$	can exit M
$M.P$	capability action	$\mathtt{open}\ M$	can open M
$(n).P$	input action	ϵ	null
$\langle M \rangle$	output action	$M.M'$	path
$!\,P$	replication		
$(\nu n)P$	name restriction		

The sets $bn(P)$ and $fn(P)$ of bound and free names of a given process P are defined in a usual way keeping in mind that (νn) and (n) are name binders. We identify processes up to renaming of bound names. We write $P\{n{\leftarrow}M\}$ for the substitution of the expression M for the name n in the process P.

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 $\rightarrow^*$ and $\downarrow^*$ for the reflexive and transitive closure of $\rightarrow$ and $\downarrow$, respectively.

Structural Congruence $P \equiv Q$:

$P \equiv P$	(Str Refl)	$(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$	(Str Res Res)
$P \equiv Q \Rightarrow Q \equiv P$	(Str Symm)	$(\nu n)\mathbf{0} \equiv \mathbf{0}$	(Str Res Zero)
$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$	(Str Trans)	$(\nu n)(P \mid Q) \equiv P \mid (\nu n)Q$ if $n \notin fn(P)$	(Str Res Par)
$P \equiv Q \Rightarrow (\nu n)P \equiv (\nu n)Q$	(Str Res)	$(\nu n)(m[P]) \equiv m[(\nu n)P]$ if $n \neq m$	(Str Res Amb)
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	(Str Par)	$P \mid \mathbf{0} \equiv P$	(Str Zero Par)
$P \equiv Q \Rightarrow\, !\,P \equiv\, !\,Q$	(Str Repl)	$P \mid Q \equiv Q \mid P$	(Str Par Comm)
$P \equiv Q \Rightarrow n[P] \equiv n[Q]$	(Str Amb)	$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(Str Par Assoc)
$P \equiv Q \Rightarrow M.P \equiv M.Q$	(Str Action)		
$P \equiv Q \Rightarrow (n).P \equiv (n).Q$	(Str Input)	$!\,\mathbf{0} \equiv \mathbf{0}$	(Str Repl Zero)
		$!\,(P \mid Q) \equiv\, !\,P \mid\, !\,Q$	(Str Repl Par)
$\epsilon.P \equiv P$	(Str ϵ)	$!\,P \equiv P \mid\, !\,P$	(Str Repl Copy)
$(M.M').P \equiv M.M'.P$	(Str .)	$!\,P \equiv\, !\,!\,P$	(Str Repl Repl)

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

$n[\mathtt{in}\ m.P \mid Q] \mid m[R] \rightarrow m[n[P \mid Q] \mid R]$	(Red In)
$m[n[\mathtt{out}\ m.P \mid Q] \mid R] \rightarrow n[P \mid Q] \mid m[R]$	(Red Out)
$\mathtt{open}\ n.P \mid n[Q] \rightarrow P \mid Q$	(Red Open)
$\langle M \rangle \mid (n).P \rightarrow P\{n{\leftarrow}M\}$	(Red I/O)
$P \rightarrow Q \Rightarrow (\nu n)P \rightarrow (\nu n)Q$	(Red Res)
$P \rightarrow Q \Rightarrow P \mid R \rightarrow Q \mid R$	(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)

When no confusion is possible, we omit the inactive process $\mathbf{0}$; for instance, we shorten $\mathtt{open}\ n.\mathbf{0}$ and $m[\mathbf{0}]$ as $\mathtt{open}\ n$ and $m[]$, respectively.

2.2 The Ambient Logic

We describe the formulas and satisfaction relation of the logic. This is the logic defined in [5] extended with the two revelation constructs for handling name restriction [7].

Logical Formulas:

η	a name n or a variable x

$\mathcal{A}, \mathcal{B} ::=$	formula	$\eta[\mathcal{A}]$	location
$\mathbf{T}$	true	$\mathcal{A}@\eta$	location adjunct
$\neg\mathcal{A}$	negation	$\eta\circledR\mathcal{A}$	revelation
$\mathcal{A} \vee \mathcal{B}$	disjunction	$\mathcal{A}\oslash\eta$	revelation adjunct
$\mathbf{0}$	void	$\Diamond\mathcal{A}$	sometime modality
$\mathcal{A} \mid \mathcal{B}$	composition match	$\text{✧}\mathcal{A}$	somewhere modality
$\mathcal{A} \triangleright \mathcal{B}$	composition adjunct	$\exists x.\mathcal{A}$	existential quantification

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. The sets of bound and free variables of a given formula are defined in a usual way keeping in mind that $\exists$ is the only variable binder. There are no name binders for formulas, so all names occurring in a formula are free. 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 inactive 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 $\text{✧}\mathcal{A}$ (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 $\text{✧}\mathcal{A}$ if it has a spatial successor, that is, a sublocation, that satisfies $\mathcal{A}$. A process P satisfies the formula $\mathcal{A}@n$ if the ambient $n[P]$ satisfies $\mathcal{A}$. A process P satisfies $\mathcal{A} \triangleright \mathcal{B}$ if for all P', the process $P \mid P'$ guarantees $\mathcal{B}$ assuming that P' satisfies $\mathcal{A}$. Finally, we discuss shortly the two logical constructs for reasoning about restricted names that were not present in [5, 8]. Intuitively, a process P satisfies the formula $n\circledR\mathcal{A}$ (read "reveal n then $\mathcal{A}$") if it is possible to pull a restricted name from P to the top and rename it n and then strip off the restriction to leave a residual process that satisfies $\mathcal{A}$. The inverse of revelation is hiding: a process P satisfies $\mathcal{A}\oslash n$ (read "hide n then $\mathcal{A}$") if it is possible to hide n in P and then satisfy $\mathcal{A}$.

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

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

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

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

3 Calculus with Replication

In this section we show that model checking for the ambient calculus with replication (but still without name restriction, and in fact without communication) against the ambient logic (without composition adjunct) is undecidable. We use here α, β, γ for words in $\{a, b\}^*$, σ for letters in $\{a, b\}$ and ϵ for the empty word. Lower-case strings (possibly with subscripts) like $c_i, n_i, w_i, start_i, word_i, compare$ denote ambient names, while upper-case strings like $Concatenate$, $Compare$, $Word_i$ denote processes.

Encoding of PCP. The undecidability proof is done by a reduction of the Post correspondence problem (PCP). An instance of the problem is a set of pairs of words $\{\langle\alpha_1, \beta_1\rangle, \dots, \langle\alpha_n, \beta_n\rangle\}$ over the two-letter alphabet $\{a, b\}$ (that is, $\alpha_i, \beta_i \in \{a, b\}^*$). The question is whether there exists a sequence of numbers $1 \leq i_0, i_1, \dots, i_k \leq n$ such that $\alpha_{i_0} \cdot \dots \cdot \alpha_{i_k} = \beta_{i_0} \cdot \dots \cdot \beta_{i_k}$, where $\cdot$ denotes word concatenation. It is well-known that Post correspondence problem is undecidable [10].

The idea of the reduction is to construct for a given instance of PCP a process P whose reduction simulates all possible concatenations of pairs of words in the instance. Then we have to only check if a process representing two equal words is reachable.

The process P is defined as the parallel composition

$$P \triangleq start_1[] \mid start_2[] \mid Word_1(\epsilon) \mid Word_2(\epsilon) \mid Concatenate \mid Compare,$$

where $start_1$ and $start_2$ are two different ambient names (we write ambient names with lower-case letters and (meta-)names of processes in upper case) and $Word_i(w)$ is a process representing the word w (we start with the empty word). Before we give the precise definition of the processes $Word_i(w)$, $Concatenate$, $Compare$, we briefly describe the intuition behind them. $Concatenate$ is a process responsible for concatenating pairs of words from the given instance of PCP: it chooses nondeterministically a pair $\langle\alpha_i, \beta_i\rangle$ and rewrites $Word_1(\alpha) \mid Word_2(\beta)$ to $Word_1(\alpha_i \cdot \alpha) \mid Word_2(\beta_i \cdot \beta)$; this is done again and again. At some nondeterministically chosen point of time the process $Compare$ activates — it stops $Concatenate$ and starts comparing the two words represented by $Word_1$ and $Word_2$ by nondeterministically choosing the letter a or b and trying to delete it simultaneously from both words; this is repeated until both words are empty or they start with a different letter. Clearly, the instance of PCP has a solution if and only if there exists a (nonempty) execution of the process that ends with the representation of two empty words.

$$\begin{aligned} Concatenate \;\triangleq\;& !\,(\mathtt{open}\; start_1.\mathtt{open}\; start_2.\mathtt{open}\; pair) \mid \\ & !\, pair[Concatenate_1(\alpha_1) \mid Concatenate_2(\beta_1)] \mid \\ & \dots \mid \\ & !\, pair[Concatenate_1(\alpha_n) \mid Concatenate_2(\beta_n)] \end{aligned}$$

The two ambients $start_1[]$ and $start_2[]$ are used for synchronization — the only possible reduction is to open $start_1[]$ and then $start_2[]$; after this the two ambients disappear and they will appear again only after $Concatenate_1$ and $Concatenate_2$ finish their jobs. In this way we avoid processing two different pairs at the same time (and thus confusing different pairs during computation).

Thus, in every iteration of $Concatenate$, we rewrite in several steps a process of the form $start_1[] \mid start_2[] \mid Word_1(\alpha) \mid Word_2(\beta) \mid Concatenate \mid Compare$ to

$$\begin{aligned} &Word_1(\alpha) \mid Word_2(\beta) \mid Concatenate_1(\alpha') \mid Concatenate_2(\beta') \\ &\qquad \mid Concatenate \mid Compare \end{aligned}$$

for some words α, β and some pair $\langle \alpha', \beta' \rangle$ from the instance of PCP. Intuitively, two words $\gamma = \sigma_1 \dots \sigma_k$ and $\gamma' = \sigma'_1 \dots \sigma'_{k'}$ in $\{a, b\}^*$ are represented by ambients $\sigma_1[\sigma_2[\dots \sigma_k[]] \dots]$ and $\sigma'_1[\sigma'_2[\dots \sigma'_{k'}[]] \dots]$ and the process $Concatenate_i(\gamma')$ leads the process $Word_i(\gamma)$ inside $\sigma'_{k'}[]$ and generates an ambient $start_i[]$ so that

$$Word_i(\gamma) \mid Concatenate_i(\gamma') \quad \rightarrow^* \quad Word_i(\gamma' \cdot \gamma) \mid start_i[].$$

The details are quite technical and are presented in the appendix. Then the initial process rewrites to

$$start_1[] \mid start_2[] \mid Word_1(\alpha' \cdot \alpha) \mid Word_2(\beta' \cdot \beta) \mid Concatenate \mid Compare$$

The process $Compare$ works in a similar way.

$$\begin{aligned} Compare \;\triangleq\; & compare[] \mid Initialize.(!\,(\texttt{open}\; compare.Consume(a)) \mid \\ & \qquad !\,(\texttt{open}\; compare.Consume(b))) \end{aligned}$$

The initialization essentially opens $start_1$ and $start_2$ so that $Concatenate$ is blocked. The process $Consume(a)$ replaces the representation of the two words α, β by α', β' if $\alpha = a\alpha'$ and $\beta = a\beta'$ by simply opening the leading ambients $a[\dots]$ in the representation of both words, similarly $Consume(b)$ opens the leading $b[\dots]$ if both words start with b. The ambient $compare[]$ is used for synchronization to avoid deleting different letters from the two words. The details are presented in the appendix.

We have the following theorem.

Theorem 3.1. *The model checking problem for the ambient calculus with replication against the ambient logic is undecidable.*

Proof. Let P be the process defined above (note that the definition of P depends on the instance of PCP). We have already seen that the instance has a solution if and only if there exists an execution of P starting with the concatenation of at least one pair and ending in a configuration representing the pair of empty words. This can be expressed by the formula

$$\mathcal{A} \triangleq \Diamond(nonempty(w_1) \wedge \Diamond(empty(w_1) \wedge empty(w_2)))$$

where

$$\begin{aligned} nonempty(w_i) &\triangleq \diamond w_i[(a[\mathbf{T}] \vee b[\mathbf{T}]) \mid \mathbf{T}] \\ empty(w_i) &\triangleq \neg nonempty(w_i). \end{aligned}$$

Here, w_i is an ambient name used in the encoding of the process $Word_i(\gamma)$ (see the appendix for details), and the formula $a[\mathbf{T}] \vee b[\mathbf{T}]$ is matched by (the encoding of) the first letter in the word γ. Then $P \models \mathcal{A}$ if and only if the instance of PCP has a solution. □

It should be noticed that our proof of undecidability of model-checking the ambient calculus with replication but without private names implies that reachability via reduction for ambient processes with public names and with replication is undecidable.

4 Logic with Composition Adjunct

We investigate in this section the problem of model checking ambients against formulas that may contain composition adjunct. Let us first show that the model checking problem of formulas with composition adjunct subsumes the satisfiability problem of formulas without composition adjunct.

Proposition 4.1. *The process* $\mathbf{0}$ *satisfies the formula* $\neg(\mathbf{T} \triangleright \neg\mathcal{A})$ *if and only if the formula* $\mathcal{A}$ *is satisfiable.*

Proof. By definition, $\mathbf{0} \models \mathbf{T} \triangleright \neg\mathcal{A}$ if and only if for all processes P that satisfy $\mathbf{T}$, the process $P \mid \mathbf{0}$ satisfies $\neg\mathcal{A}$. Since all processes satisfy $\mathbf{T}$ and $P \mid \mathbf{0}$ is equivalent to P, by the definition of satisfaction for negation we have that $\mathbf{0} \models \neg(\mathbf{T} \triangleright \neg\mathcal{A})$ if and only if there exists P that satisfies $\mathcal{A}$. $\square$

We show now that the satisfiability problem for ambient formulas (even without composition adjunct) is an undecidable problem [1]. Thus, it implies

Theorem 4.1. *The model checking problem of ambient processes without replication and name restriction against formulas with composition adjunct is undecidable.*

Let us consider the set $\mathcal{F}$ of first-order formulas defined over a countable set of variables $x, y, z, \ldots$ and some relational symbols $\{R_1, \ldots, R_k\}$, each of those symbols having strictly positive arity. The set of formulas $\mathcal{F}$ is the least set such that (i) for any R_i with arity l, $\mathcal{F}$ contains $R_i(x_1, \ldots, x_l)$, and (ii) for all φ and φ' in $\mathcal{F}$, $\varphi \wedge \varphi'$, $\neg\varphi$ and $\exists x\varphi$ belong to $\mathcal{F}$.

Formulas from $\mathcal{F}$ are interpreted over structures; a structure $\mathcal{S}$ over some domain $\mathcal{D}$ is simply a set of objects of the form $R_i(a_1, \ldots, a_l)$ where R_i is an l-ary relational symbol and $a_1, \ldots, a_l$ are elements of $\mathcal{D}$. We say that a structure S is finite whenever its domain $\mathcal{D}$ is finite.

A formula is said to be closed if it has no free variables. We assume wlog. that in formulas bound variables are pairwise distinct. For a formula φ and a structure S with domain $\mathcal{D}$, a valuation σ is a mapping from the free variables of φ to $\mathcal{D}$. A structure S is a model of a formula φ under a valuation σ (written $S, \sigma \models \varphi$) if

- $R_i(x_1, \ldots, x_l)\sigma \in S$ for $\varphi = R_i(x_1, \ldots, x_l)$,
- $S, \sigma \models \varphi'$ and $S, \sigma \models \varphi''$ for $\varphi = \varphi' \wedge \varphi''$,
- $S, \sigma \not\models \varphi'$ for $\varphi = \neg\varphi'$,
- there exists a in $\mathcal{D}$ such that $S, \sigma\{x{\leftarrow}a\} \models \varphi'$ for $\varphi = \exists x\varphi'$.

[1] Actually we consider a very small fragment of the logic, in particular without temporal modalities.

Theorem 4.2 (Trakhtenbrot [12]). *Given a closed first-order formula φ, it is undecidable to know whether φ admits a finite model.*

With a formula φ from $\mathcal{F}$ we associate a formula $[\![\varphi]\!]$ from the ambient logic inductively defined as follows:

- $[\![R_i(x_1, \ldots, x_l)]\!] = r_i[x_1[x_2[\ldots[x_l[\mathbf{0}]]\ldots]]] \mid \mathbf{T}$,
- $[\![\varphi \wedge \varphi']\!] = [\![\varphi]\!] \wedge [\![\varphi']\!]$
- $[\![\neg\varphi]\!] = \neg[\![\varphi]\!]$,
- $[\![\exists x\varphi]\!] = \exists x((d[x[\mathbf{0}]] \mid \mathbf{T}) \ \wedge \ [\![\varphi]\!])$.

Note that we identify first-order variables in formulas from $\mathcal{F}$ with variables of the ambient logic. Therefore, free variables of φ and $[\![\varphi]\!]$ coincide.

The key idea of this encoding is to consider the parallel operator of the ambient calculus as a (multi-)set constructor. Then, the finite domain $\mathcal{D}$ as well as the structure $\mathcal{S}$ are encoded in a straightforward way using simply ambient name d for elements from $\mathcal{D}$ and ambient names r_i for the relational symbols R_i in $\mathcal{S}$.

Lemma 4.1. *A closed formula φ from $\mathcal{F}$ admits a finite model iff there exists an ambient process P without replication and without name restriction such that $P \models [\![\varphi]\!]$.*

The proof of Lemma 4.1 can be found in the appendix. It is straightforward that Lemma 4.1 and Theorem 4.2 yield the undecidability of the satisfiability problem of the logic without composition adjunct over ambient processes without replication and name restriction. Hence, Theorem 4.1 follows.

5 Calculus with Name Restriction

In this section we show that the model-checking algorithm from [5] can be extended to the calculus with name restriction. Moreover, the additional logical operators introduced in [7] do not influence the decidability. We recall that, as in [5], the logic does not contain the composition adjunct and the calculus does not contain replication.

First, we fix the representation of the processes: Using α-renaming of restricted names and the rules (Str Res Par) and (Str Res Amb) of the congruence relation, we group together all name-restriction operators by transforming every process to one of the form $(\nu n_1)\ldots(\nu n_k)P$ where every occurrence of name restriction is guarded by an action, that is every name restriction occurring in P occurs in a subprocess of the form $M.P'$ or $(n).P'$ with $M \not\equiv \epsilon$. Formally, a process is guarded if it is of the form

$$\texttt{in}\, M.P,\ \texttt{out}\, M.P,\ \texttt{open}\, M.P,\ n.P,\ (n).P,\ \text{or } \langle M\rangle$$

for some process P and expression M. Note that neither a guarded process nor any of its subprocesses can be reduced (a guarded process can be reduced only if it occurs in a parallel composition with other processes) and that the sublocation relation $\downarrow^*$ does not look inside a guarded process. We separate bounded names from the unguarded part of a process using the following function $separate$. Recall that we assume that all bounded names are renamed apart so that they are different.

Separating Bounded Names from a Process

$\mathit{separate}(P) \triangleq \langle \varnothing, P\rangle$	if P is guarded
$\mathit{separate}((\nu n)P) \triangleq \langle N \cup \{n\}, P'\rangle$	if $\mathit{separate}(P) = \langle N, P'\rangle$
$\mathit{separate}(n[P]) \triangleq \langle N, n[P']\rangle$	if $\mathit{separate}(P) = \langle N, P'\rangle$
$\mathit{separate}(P \mid Q) \triangleq \langle N \cup N', P' \mid Q'\rangle$	if $\mathit{separate}(P) = \langle N, P'\rangle$ and $\mathit{separate}(Q) = \langle N', Q'\rangle$

The definition of the $\rightarrow$ and $\downarrow$ relations extends in a straightforward way to the representation with bounded names separated from the process. We say that $\langle N, P\rangle \rightarrow \langle N', P'\rangle$ if there exists P'' such that $P \rightarrow P''$, $\mathit{separate}(P'') = \langle N'', P'\rangle$ and $N' = N \cup N''$. Similarly, $\langle N, P\rangle \downarrow \langle N, P'\rangle$ if there exist n, P'' such that $P \equiv n[P'] \mid P''$ and $n \notin N$. As usually, $\rightarrow^*$ and $\downarrow^*$ are reflexive and transitive closures of $\rightarrow$ and $\downarrow$ respectively. We define

$$\mathit{Reachable}(N, P) \triangleq \{\langle N', P'\rangle \mid \langle N, P\rangle \rightarrow^* \langle N', P'\rangle\}$$
$$\mathit{Sublocations}(N, P) \triangleq \{\langle N, P'\rangle \mid \langle N, P\rangle \downarrow^* \langle N, P'\rangle\}$$

Example 5.1. Consider the processes $P_1 = (m).(\nu n)(m[\mathtt{in}\ n] \mid n[]) \mid \langle a\rangle$, $P_2 = (\nu n)(m).(m[\mathtt{in}\ n] \mid n[]) \mid \langle a\rangle$ and the formulas $\mathcal{A}_1 = \Diamond(n®n[a[\mathbf{T}]])$ and $\mathcal{A}_2 = n®(\Diamond n[a[\mathbf{T}]])$.

The process P_1 is guarded, so $\mathit{separate}(P_1) = \langle \varnothing, P_1\rangle$; the process P_2 is not guarded and $\mathit{separate}(P_2) = \langle \{n\}, (m).(m[\mathtt{in}\, n] \mid n[]) \mid \langle a\rangle\rangle$. In both cases, that is for $i = 1, 2$, using the reduction (Red I/O) followed by (Red In), we reduce $\mathit{separate}(P_i) \rightarrow \langle \{n\}, a[\mathtt{in}\, n] \mid n[]\rangle \rightarrow \langle \{n\}, \mid n[a[]]\rangle$.

The reader can check that $P_1 \models \mathcal{A}_1$ and $P_2 \models \mathcal{A}_2$. Moreover, $P_2 \models \mathcal{A}_1$, but $P_1 \not\models \mathcal{A}_2$.

Now we are ready to define our model-checking algorithm. It is an extension of the algorithms from [5, 8] to the calculus and logic with name restriction. We write $\dot\cup$ for disjoint union, that is, $A = B \dot\cup C$ if $A = B \cup C \wedge B \cap C = \varnothing$. We recall the assumption that all bound names in the process are renamed apart so that they are all different from each other and different from all free names occurring in the process and the formula.

Model-checking Algorithm

$\mathit{Check}(N, P, \mathbf{T}) \triangleq \mathbf{T}$

$\mathit{Check}(N, P, \neg\mathcal{A}) \triangleq \neg\,\mathit{Check}(N, P, \mathcal{A})$

$\mathit{Check}(N, P, \mathcal{A} \vee \mathcal{B}) \triangleq \mathit{Check}(N, P, \mathcal{A}) \vee \mathit{Check}(N, P, \mathcal{B})$

$\mathit{Check}(N, P, \mathbf{0}) \triangleq \begin{cases} \mathbf{T} \text{ if } P \equiv \mathbf{0} \\ \mathbf{F} \text{ otherwise} \end{cases}$

$\mathit{Check}(N, P, \mathcal{A} \mid \mathcal{B}) \triangleq \bigvee_{N_1 \dot\cup N_2 = N} \bigvee_{P_1 \mid P_2 \equiv P} \mathit{Check}(N_1, P_1, \mathcal{A}) \wedge \mathit{Check}(N_2, P_2, \mathcal{B}) \wedge$
$\qquad \mathit{fn}(P_1) \cap N_2 = \varnothing \wedge \mathit{fn}(P_2) \cap N_1 = \varnothing$

$\mathit{Check}(N, P, n[\mathcal{A}]) \triangleq P \equiv n[Q] \wedge n \notin N \wedge \mathit{Check}(N, Q, \mathcal{A})$

$\mathit{Check}(N, P, \mathcal{A}@n) \triangleq \mathit{Check}(N, n[P], \mathcal{A})$

$\mathit{Check}(N, P, n®\mathcal{A}) \triangleq \bigvee_{m \in N} \mathit{Check}(N - \{m\}, P\{m \leftarrow n\}, \mathcal{A})$
$\qquad \vee (n \notin \mathit{fn}(P) \wedge \mathit{Check}(N, P, \mathcal{A}))$

$\mathit{Check}(N, P, \mathcal{A}⊘n) \triangleq \mathit{Check}(N \cup \{n\}, P, \mathcal{A})$

$$Check(N,P,\Diamond\mathcal{A}) \triangleq \bigvee_{\langle N',P'\rangle \in Reachable(N,P)} Check(N',P',\mathcal{A})$$
$$Check(N,P,\diamond\mathcal{A}) \triangleq \bigvee_{\langle N,P'\rangle \in Sublocations(N,P)} Check(N,P',\mathcal{A})$$
$$Check(N,P,\exists x.\mathcal{A}) \triangleq \text{let } n_0 \notin N \cup fn(P) \cup bn(P) \text{ be a fresh name in}$$
$$\bigvee_{n \in fn(N,P) \cup fn(\mathcal{A})} Check(N,P,\mathcal{A}\{x \leftarrow n\}) \vee Check(N,P,\mathcal{A}\{x \leftarrow n_0\})$$

The correctness of the algorithm (Theorem 5.1) is based on the following lemmas and propositions.

Proposition 5.1.

(1) $(\nu n)P \equiv \mathbf{0}$ *if and only if* $P \equiv \mathbf{0}$.
(2) *If* n *and* m *are different names, then* $(\nu n)P \equiv m[Q]$ *if and only if there exists a process* R *such that* $P \equiv m[R]$ *and* $Q \equiv (\nu n)R$.
(3) $(\nu n)P \equiv Q \mid Q'$ *if and only if there exist processes* R, R' *such that either* $Q \equiv (\nu n)R$ *and* $Q' \equiv R'$ *and* $n \notin fn(Q')$ *or* $Q \equiv R$ *and* $Q' \equiv (\nu n)R'$ *and* $n \notin fn(Q)$.

Sketch of proof. The "if" implications follow from (Str Res Zero), (Str Res Amb) and (Str Res Par) congruence rules. The first two of the equivalences above are stated as Inversion Lemmas (Lemmas 2-2) in [7] with a reference to [9] for a proof. However, the proof from [9] cannot be applied directly, since spatial congruence from [9] is not the same as structural congruence. In particular, it does not distinguish between the processes P_1 and P_2 from Example 5.1. On the other hand, the methods of [9] can be easily extended to the case of structural congruence.

The third equivalence is a stronger version of the third Inversion Lemma (the Inversion Lemma does not mention the $n \notin fn(Q)$ condition), again proved in the case of spatial congruence in [9]. Again, the same methods can be used to show our stronger version in the case of structural congruence (in the general case of processes with replication the proof is not very easy, but in the absence of replication, as we have it here, it is enough to use the Inversion Lemma together with the observation that equivalent processes have the same numbers of occurrences of free names). $\square$

Lemma 5.1. *If* $n \notin fn(P)$ *then* $(\nu n)P \equiv P$.

Proof. By the rules (Str Res) and (Str Zero Par), $(\nu n)P$ is equivalent to $(\nu n)(P \mid \mathbf{0})$, which by (Str Res Par), (Str Res Zero) and (Str Zero Par) is equivalent to P. $\square$

Proposition 5.2. *Consider any process* P *and a closed* $\triangleright$*-free formula* $\mathcal{A}$. *Let* $fn(P) \cup fn(\mathcal{A}) = \{n_1, \ldots, n_k\}$, *and suppose* $n_0 \notin \{n_1, \ldots, n_k\}$. *Then* $P \models \exists x.\mathcal{A}$ *if and only if* $P \models \mathcal{A}\{x \leftarrow n_i\}$ *for some* $i \in \{0, \ldots, k\}$.

The proof follows the lines of the proof of Proposition 4.11 in [3]. $\square$

Lemma 5.2. *For all processes* P, P'

(1) $P \downarrow P'$ *if and only if* $separate(P) \downarrow separate(P')$
(2) $P \to P'$ *if and only if* $separate(P) \to separate(P')$
(3) *the sets* $Reachable(separate(P))$ *and* $Sublocations(separate(P))$ *are finite and effectively computable.*

Sketch of proof. The first two equivalences follow from Proposition 5.1 and the definitions of $\downarrow$ and $\rightarrow$ relations. The third one follows from the finiteness of the analogous sets for the replication-free fragment of the ambient calculus with public names [5, 8] and a simple observation that $P \downarrow P'$ and $P \rightarrow P'$ imply $\mathit{public}(P) \downarrow \mathit{public}(P')$ and $\mathit{public}(P) \rightarrow \mathit{public}(P')$, respectively, where $\mathit{public}(P)$ is the process obtained from P by removing all ν quantifiers. □

Theorem 5.1. *For all replication-free processes P and closed $\triangleright$-free formulas $\mathcal{A}$, we have $P \models \mathcal{A}$ if and only if $\mathit{Check}(\mathit{separate}(P), \mathcal{A}) = \mathbf{T}$.*

Sketch of proof. The proof goes by induction on the formula $\mathcal{A}$. In the cases of $\mathbf{T}$, $\neg\mathcal{A}$, $\mathcal{A} \vee \mathcal{B}$, $\mathcal{A}@n$, $\mathcal{A} \oslash n$ it follows directly from the definition of the satisfaction relation (the side condition $n_0 \notin N \cup \mathit{fn}(P)$ in the case of $\mathcal{A} \oslash n$ reflects only our convention that all bounded names are renamed apart). In the case of $\mathbf{0}$ and $n[\mathcal{A}]$ it follows from Proposition 5.1(1) and (2), in the case of $\mathcal{A} \mid \mathcal{B}$ from Proposition 5.1(3) by induction on the number of names in N. The cases of $\diamond\mathcal{A}$ and $\Diamond\mathcal{A}$ follow from Lemma 5.2 and the definitions of $\mathit{Reachable}$ and $\mathit{Sublocations}$; $\exists x.\mathcal{A}$ from Proposition 5.2. Finally, the case of $n®\mathcal{A}$ reflects the two possibilities that either n is one of the bounded names occurring in the process or it does not occur there (in the latter case we appeal to Lemma 5.1). □

Theorem 5.2. *The model-checking problem for replication-free processes against $\triangleright$-free ambient logic is decidable. Moreover, it is PSPACE-complete.*

Sketch of proof. Decidability follows from Theorem 5.1. One obtains the PSPACE upper bound by combining the above algorithm with the polynomial-size representation of processes from [8] and implementing disjunction in polynomial space, as it is done in [8]. The PSPACE lower bound is proved in [8]. □

6 Conclusion

We investigate in this paper borders of the decidability for model-checking mobile ambient against the ambient logic. We have started from the fragments of mobile calculus without name restriction and without replication against the logic without composition adjunct for which decidability of model-checking has been showed in [5]. We have showed that adding either replication in the calculus or composition adjunct in the logic leads to undecidability for the model-checking problem. On the other hand, we have considered the extension of the calculus with private names and the adequate operators in the logic to manipulate those names. We have proved this extension to preserve decidability of model-checking as well as the complexity of the original fragments.

A Encoding of PCP: Concatenation and Comparison of Words

A.1 Concatenation

Here we show how to rewrite $\mathit{Word}_i(\gamma) \mid \mathit{Concatenate}_i(\gamma')$ to $\mathit{Word}_i(\gamma' \cdot \gamma) \mid \mathit{start}_i[]$. For this, we need precise definition of Word_i and $\mathit{Concatenate}_i$. For $i = 1, 2$ we introduce fresh ambient names $\mathit{word}_i, c_i, n_i, v_i, w_i$; similarly, we introduce fresh names a, b

corresponding to the two letters of the alphabet. Let $\gamma = \sigma_1 \dots \sigma_k$ and $\gamma' = \sigma'_1 \dots \sigma'_{k'}$ be two words in $\{a,b\}^*$. We define

$$\begin{aligned} Word_i(\gamma) &\triangleq word_i[!\,\texttt{open}\, c_i \mid \\ &\qquad w_i[\texttt{open}\, n_i \mid String(\gamma)]] \\ Concatenate_i(\gamma') &\triangleq c_i[\texttt{in}\, word_i.MvIn_i(\gamma') \mid String'(\gamma', Continue_i)] \end{aligned}$$

where

$$\begin{aligned} String(\sigma_1 \dots \sigma_k) &\triangleq \sigma_1[\sigma_2[\dots \sigma_k[\,] \dots]] \\ String'(\sigma'_1 \dots \sigma'_{k'}, P) &\triangleq s_i[\texttt{in}\, v_i.\texttt{in}\, w_i \mid \sigma'_1[\dots \sigma'_k[P] \dots]] \\ MvIn_i(\sigma'_1 \dots \sigma'_k) &\triangleq n_i[\texttt{in}\, w_i.\texttt{in}\, s_i.\texttt{in}\, \sigma'_1 \dots \texttt{in}\, \sigma'_k] \\ MvOut(\sigma'_1 \dots \sigma'_k) &\triangleq \texttt{out}\, \sigma'_k \dots \texttt{out}\, \sigma'_1 \\ Continue_i &\triangleq \texttt{open}\, w_i.v_i[MvOut(\gamma').Tinue_i] \\ Tinue_i &\triangleq \texttt{in}\, w_i \mid w_i[\texttt{open}\, n_i \mid \texttt{open}\, s.\texttt{out}\, v_i.Nue_i] \\ Nue_i &\triangleq \texttt{open}\, v_i.start[\texttt{out}\, w_i.\texttt{out}\, word_i]. \end{aligned}$$

Then $Word_i(\gamma) \mid Concatenate_i(\gamma')$ reduces by moving the ambient $c_i[\dots]$ inside $word_i[\dots]$ and opening it to

$$\begin{aligned} &word_i[!\,\texttt{open}\, c_i \mid MvIn_i(\gamma') \mid String'(\gamma', Continue_i) \\ &\qquad w_i[\texttt{open}\, n_i \mid String(\gamma)]], \end{aligned}$$

the ambient $n_i[\dots]$ goes inside $w_i[\dots]$ and gets opened there

$$\begin{aligned} &word_i[!\,\texttt{open}\, c_i \mid String'(\gamma', Continue_i) \\ &\qquad w_i[\texttt{in}\, s_i.\texttt{in}\, \sigma'_1 \dots \texttt{in}\, \sigma'_k \mid String(\gamma)]], \end{aligned}$$

$w_i[\dots]$ goes inside $String'(\dots)$

$$\begin{aligned} &word_i[!\,\texttt{open}\, c_i \mid \\ &\qquad String'(\gamma', (w_i[String(\gamma)] \mid Continue_i))], \end{aligned}$$

$Continue_i$ opens w_i and v_i moves out of γ'

$$\begin{aligned} &word_i[!\,\texttt{open}\, c_i \mid \\ &\qquad String'(\gamma' w, \mathbf{0}) \mid v_i[Tinue_i]], \end{aligned}$$

$s_i[\dots]$ goes inside v_i, then inside w_i and gets opened there

$$\begin{aligned} &word_i[!\,\texttt{open}\, c_i \mid \\ &\qquad v_i[\texttt{in}\, w_i \mid w_i[\texttt{open}\, n_i \mid \texttt{out}\, v_i.Nue_i \mid String(\gamma'\gamma)]]]], \end{aligned}$$

w_i gets out of v_i and v_i gets into w_i

$$\begin{aligned} &word_i[!\,\texttt{open}\, c_i \mid \\ &\qquad w_i[\texttt{open}\, n_i \mid v_i[\,] \mid Nue_i \mid String(\gamma'\gamma)]], \end{aligned}$$

Nue_i opens v_i; $start_i$ goes out of w_i and out of $word_i$

$$Word_i(\gamma'\gamma) \mid start_i[]$$

which is the desired process. Note that since guarded processes cannot be reduced, this was the only possible execution of the process $Word_i(\gamma) \mid Concatenate_i(\gamma')$.

A.2 Comparing the Words

First we define the two missing processes.

$$\begin{aligned} Initialize &\triangleq \mathtt{open}\ start_1.\mathtt{open}\ start_2.\mathtt{open}\ word_1.\mathtt{open}\ word_2 \\ Consume(\sigma) &\triangleq n_1[\mathtt{in}\ w_1.\mathtt{open}\ \sigma.(\mathtt{open}\ n_1 \mid Nsume(\sigma))] \end{aligned}$$

where

$$Nsume(\sigma) \triangleq n_2[\mathtt{out}\ w_1.\mathtt{in}\ w_2.\mathtt{open}\ \sigma.(\mathtt{open}\ n_2 \mid compare[\mathtt{out}\ w_2])].$$

Then

$$start_1[] \mid start_2[] \mid Word_1(\alpha) \mid Word_2(\beta) \mid Compare$$

reduces to

$$\begin{aligned} &!\,\mathtt{open}\ c_1 \mid !\,\mathtt{open}\ c_2 \mid \\ &w_1[\mathtt{open}\ n_1 \mid String(\alpha)] \mid w_2[\mathtt{open}\ n_2 \mid String(\beta)] \mid \\ &compare[] \mid !\,(\mathtt{open}\ compare.Consume(a)) \mid !\,(\mathtt{open}\ compare.Consume(b))). \end{aligned}$$

The two processes $!\,\mathtt{open}\ c_i$ remain inactive, since the names c_i will never occur again. The only possibility of executing the process is to choose one of the two subprocesses consuming a or b; each of them opens $compare$, so the desired property now is that

$$w_1[\mathtt{open}\ n_1 \mid String(\sigma\alpha)] \mid w_2[\mathtt{open}\ n_2 \mid String(\sigma\beta)] \mid Consume(\sigma)$$

reduces to

$$w_1[\mathtt{open}\ n_1 \mid String(\alpha)] \mid w_2[\mathtt{open}\ n_2 \mid String(\beta)] \mid compare[].$$

This can be easily checked by the reader: The process $Consume(\sigma)$ is an ambient named n_1, it goes inside w_1, gets opened there, opens σ thus deleting the leading letter from σu, leaves the capability $\mathtt{open}\ n_1$ for the next iteration, and as $Nsume(\sigma)$ goes out of w_1; then it repeats the same thing with w_2 and leaves the ambient $compare[]$ at the top level. Note that if the two words u and v start with two different letters a and b then the process $w_1[\mathtt{open} n_1 \mid String(\alpha)] \mid w_2[\mathtt{open} n_2 \mid String(\beta)] \mid Consume(\sigma)$ deadlocks after reaching a configuration where it tries to open σ but there is no ambient named σ at the respective place. If this happens, no further reduction of the whole process is possible.

B Satisfiability of the Ambient Logic

We give here the proof of Lemma 4.1. We consider the relation $\rightsquigarrow_{Proc}$ between finite structures and ambient processes without replication and name restriction. For a process P and a structure $\mathcal{S}$ whose domain is $\mathcal{D}$, we have $\mathcal{S} \rightsquigarrow_{Proc} P$ if:

- there exists P' such that $P \equiv d[a[\mathbf{0}]] \mid P'$ iff a belongs to $\mathcal{D}$,
- whenever $a_1, \ldots, a_l$ belong to $\mathcal{D}$, there exists P'' such that P is structurally congruent to $r_i[a_1[a_2[\ldots[a_l[\mathbf{0}]]\ldots]]] \mid P''$ iff $R_i(a_1, \ldots, a_l)$ belongs to $\mathcal{S}$.

We denote $\rightsquigarrow_{Struct}$ the symmetric relation of $\rightsquigarrow_{Proc}$. Notice that $\rightsquigarrow_{Proc} \circ \rightsquigarrow_{Struct}$ is the identity relation over structures and $\rightsquigarrow_{Struct} \circ \rightsquigarrow_{Proc}$ simply contains the identity relation.

We prove the following proposition which implies Lemma 4.1 in case where the formula φ is closed.

Proposition B.1. *Let φ be a formula from $\mathcal{F}$. Then,*

(i) let $\mathcal{S}$ be a finite structure over a domain $\mathcal{D}$ and α be a valuation for the free variables of φ. If $\mathcal{S}, \alpha \models \varphi$ and $\mathcal{S} \rightsquigarrow_{Proc} P$ then $P \models [\![\varphi]\!]\alpha$,

(ii) let P be an ambient process without replication and name restriction and α be a mapping from variables of φ to names. If $P \models [\![\varphi]\!]\alpha$ and $P \rightsquigarrow_{Struct} \mathcal{S}$ then $\mathcal{S}, \alpha \models \varphi$.

Proof. The proof goes by induction over the structure of φ.

- for $\varphi = R_i(x_1, \ldots, x_l)$:
  Case (i): $\mathcal{S}, \alpha \models R_i(x_1, \ldots, x_l)$. Therefore, for α equal to $\{x_1 \leftarrow a_1, \ldots, x_l \leftarrow a_l\}$, $R_i(a_1, \ldots, a_l)$ belongs to $\mathcal{S}$. Therefore, by definition of $\mathcal{S} \rightsquigarrow_{Proc} P$, P is structurally congruent to $r_i[a_1[\ldots[a_l[\mathbf{0}]]\ldots]] \mid P'$ for some P'. So, P is a model of $(r_i[x_1[\ldots[x_l[\mathbf{0}]]\ldots]] \mid \mathbf{T})\alpha$, that is $P \models [\![R_i(x_1, \ldots, x_l)]\!]\alpha$.
  Case (ii): $P \models [\![R_i(x_1, \ldots, x_l)]\!]\alpha$. Therefore, for $\alpha = \{x_1 \leftarrow a_1, \ldots, x_l \leftarrow a_l\}$, $P \models r_i[a_1[\ldots[a_l[\mathbf{0}]]\ldots]] \mid \mathbf{T}$. Hence, by definition of the satisfaction relation, there exists P' such that $P \equiv r_i[a_1[\ldots[a_l[\mathbf{0}]]\ldots]] \mid P'$. Since $P \rightsquigarrow_{Struct} \mathcal{S}$, we have that $a_1, \ldots, a_l$ belong to $\mathcal{D}$ and $R_i(a_1, \ldots, a_l)$ belongs to $\mathcal{S}$. Thus, $\mathcal{S}, \alpha \models R_i(x_1, \ldots, x_l)$.
- for $\varphi = \varphi' \wedge \varphi''$:
  Case (i): as $\mathcal{S}, \alpha \models \varphi$, $\mathcal{S}, \alpha \models \varphi'$ and $\mathcal{S}, \alpha \models \varphi''$. So, by induction hypothesis, $P \models \varphi'\alpha$ and $P \models \varphi''\alpha$. Thus, $P \models \varphi$.
  Case (ii): dual from the previous case.
- for $\varphi = \neg\varphi'$:
  Case (i): $\mathcal{S}, \alpha \models \neg\varphi'$. So, $\mathcal{S}, \alpha \not\models \varphi'$. Hence, by induction hypothesis for Case (ii), either $P \not\models [\![\varphi']\!]\alpha$ or $P \not\rightsquigarrow_{Struct} \mathcal{S}$. Furthermore, we know by assumption that $\mathcal{S} \rightsquigarrow_{Proc} P$, and so, $P \rightsquigarrow_{Struct} \mathcal{S}$. Hence, $P \not\models [\![\varphi']\!]\alpha$ holds. Therefore, $P \models \neg([\![\varphi']\!]\alpha)$. Finally, $P \models [\![\varphi]\!]\alpha$.
  Case (ii): $P \models [\![\neg\varphi']\!]\alpha$. So, $P \not\models [\![\varphi']\!]\alpha$. Hence, by induction hypothesis for Case (i), either $\mathcal{S}, \alpha \not\models \varphi'$ or $\mathcal{S} \not\rightsquigarrow_{Proc} P$. As by assumption, $P \rightsquigarrow_{Struct} \mathcal{S}$, we have $\mathcal{S} \rightsquigarrow_{Proc} P$. So, $\mathcal{S}, \alpha \not\models \varphi'$ holds. Hence, $\mathcal{S}, \alpha \models \neg\varphi'$.

– $\varphi = \exists x \varphi'$:
Case (i): $\mathcal{S}, \alpha \models \exists x \varphi'$. By definition, there exists $a \in \mathcal{D}$ such that $\mathcal{S}, \alpha\{x \leftarrow a\} \models \varphi'$. By assumption $\mathcal{S} \leadsto_{Proc} P$, so there exists P' such that $P \equiv d[a[\mathbf{0}]] \mid P'$. Moreover, by induction hypothesis, $P \models [\![\varphi']\!]\alpha\{x \leftarrow a\}$. Hence, $P \models (d[a[\mathbf{0}]] \mid \mathbf{T}) \wedge ([\![\varphi']\!]\alpha\{x \leftarrow a\})$. Hence, $P \models ((d[x[\mathbf{0}]] \mid \mathbf{T}) \wedge [\![\varphi']\!])\alpha\{x \leftarrow a\}$. So, $P \models \exists x.((d[x[\mathbf{0}]] \mid \mathbf{T}) \wedge [\![\varphi']\!])\alpha$.
Case (ii): $P \models [\![\exists x \varphi']\!]\alpha$, that is by definition $P \models (\exists x.((d[x[\mathbf{0}]] \mid \mathbf{T}) \wedge [\![\varphi']\!]))\alpha$. Therefore, by definition of satisfiability, there exists a name a such that $P \models ((d[x[\mathbf{0}]] \mid \mathbf{T}) \wedge [\![\varphi']\!])\alpha\{x \leftarrow a\}$. This implies that
   - there exists P' such that $P \equiv d[a[\mathbf{0}]] \mid P'$,
   - $P \models [\![\varphi']\!]\alpha\{x \leftarrow a\}$.

  As $P \leadsto_{Struct} \mathcal{S}$, the first point implies that $a \in \mathcal{D}$. This latter together with the second point and the induction hypothesis implies that $\mathcal{S}, \alpha\{x \leftarrow a\} \models \varphi'$. So, $\mathcal{S}, \alpha \models \exists x \varphi'$.

References

1. L. Cardelli and G. Ghelli. A query language based on the ambient logic. In *Proceedings of the 9th European Symposium on Programming ESOP'01*, volume 2028 of *LNCS*, pages 1–22. Springer, 2001.
2. L. Cardelli and A. D. Gordon. Equational properties of mobile ambients. In *Proceedings FoSSaCS'99*, volume 1578 of *LNCS*, pages 212–226. Springer, 1999. An extended version appears as Microsoft Research Technical Report MSR–TR–99–11, April 1999.
3. L. Cardelli and A. D. Gordon. Modal logics for mobile ambients: Semantic reasoning. Unpublished annex to [5], 1999.
4. L. Cardelli and A. D. Gordon. Types for mobile ambients. In *Proceedings POPL'99*, pages 79–92. ACM, Jan. 1999.
5. L. Cardelli and A. D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *Proceedings POPL'00*, pages 365–377. ACM, Jan. 2000.
6. L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.
7. L. Cardelli and A. D. Gordon. Logical properties of name restriction. In *Proceedings of the 5th International Conference on Typed Lambda Calculi and Applications (TLCA'01)*, volume 2044 of *LNCS*, pages 46–60. Springer, 2001.
8. W. Charatonik, S. Dal Zilio, A. D. Gordon, S. Mukhopadhyay, and J.-M. Talbot. The complexity of model checking mobile ambients. In *Proceedings FoSSaCS'01*, volume 2030 of *LNCS*, pages 152–167. Springer, 2001.
9. S. Dal Zilio. Spatial congruence for ambients is decidable. In *Proceedings of the 6th Asian Computing Science Conference (ASIAN'00)*, volume 1961 of *LNCS*, pages 88–103. Springer, 2000. A full version *Technical Report MSR-TR-2000-57, Microsoft Research*.
10. E. L. Post. Recursively Enumerable Sets of Positive Integers and their Decision Problems. *Bulletion of the American Mathematical Society*, 50:284–316, 1944.
11. D. Sangiorgi. Extensionality and intensionality of the ambient logics. In *Proceedings POPL'01*, pages 4–13. ACM, Jan. 2001.
12. B. A. Trakhtenbrot. The impossibility of an algorithm for the decision problem for finite models. *Doklady Akademii Nauk SSR*, 70:569–572, 1950.