



Model checking mobile ambients

Witold Charatonik^{a,*,1,2}, Silvano Dal Zilio^{b,3}, Andrew D. Gordon^c,
Supratik Mukhopadhyay^{d,2}, Jean-Marc Talbot^{e,2}

^aUniwersytet Wrocławski, Poland

^bLaboratoire d'Informatique Fondamentale de Marseille (LIF, UMR 6166), France

^cMicrosoft Research, UK

^dUniversity of Pennsylvania, USA

^eLaboratoire d'Informatique Fondamentale de Lille, France

Received 19 December 2001; received in revised form 26 September 2002; accepted 21 October 2002
Communicated by U. Montanari

Abstract

We settle the complexity bounds of the model checking problem for the ambient calculus with public names against the ambient logic. We show that if either the calculus contains replication or the logic contains the guarantee operator, the problem is undecidable. In the case of the replication-free calculus and guarantee-free logic we prove 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.

© 2002 Elsevier B.V. All rights reserved.

Keywords: Ambient calculus; Model checking; Ambient logic; Mobile computation; Verification

1. Introduction

The ambient calculus of Cardelli and Gordon [8,18,10] is a formalism for describing the mobility of both software and hardware. An ambient is a bounded place where

* Corresponding author. Instytut Informatyki Uniwersytetu Wrocławskiego, Przesmyckiego 20, 51-151 Wrocław, Poland.

E-mail address: wch@ii.uni.wroc.pl (W. Charatonik).

¹ Partially supported by Polish KBN grant 8T11C04319.

² Part of this work was done when the author was employed by Max-Planck-Institut für Informatik, Germany.

³ Part of this work was done when the author was employed by Microsoft Research, UK.

computation happens; it 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 names are used to control access to ambients.

The ambient calculus can be viewed as a language for programming the web. In order to ensure the correctness of programs written in this language, we need a language for formally describing their properties and reasoning about them. To this end, Cardelli and Gordon propose the ambient logic [9], 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 [6]. Other work on the ambient logic includes a study of the process equivalence induced by the satisfaction relation [26] and a study of the logic extended with constructs for describing private names [7].

Given a program written in the ambient calculus (an ambient process) and its properties specified as an ambient logic formula, one would like to determine automatically whether the process satisfies the specification. This problem is called *model checking*. In general, the model checking problem is to decide whether a given object satisfies (that is, is a model of) a given formula. This paper is concerned with model checking for mobile ambient processes against specifications described as formulas in the ambient logic.

Cardelli and Gordon [9] give a model checking algorithm for the fragment of the calculus in which processes contain neither replications nor dynamic name generations against a fragment of the logic in which formulas contain no guarantee operators. They raise the question whether their algorithm can be extended to include replication or guarantee. Both are sources of infinity: a replicated process is equivalent to an infinite array of replicas of a process; a guarantee formula is equivalent to a certain infinite quantification over processes. In Section 3 we answer this question negatively: it is not possible to extend the algorithm, because each of these extensions leads to undecidability.

Cardelli and Gordon do not give any complexity analysis for their algorithm. In fact, a naive analysis of the algorithm 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 Sections 4 and 5 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 guarantee. Our main result (embodied in Theorems 4.11 and 5.2) is that the problem is PSPACE-complete.

As we discuss in Section 4.1, there are two reasons why Cardelli and Gordon's algorithm 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 4, we present a new model checking algorithm that avoids these problems as follows.

- We avoid the first problem by devising a new representation of processes using a form of closure. The main feature of this representation is that substitutions that occur when communications take place within an ambient are not applied directly, but are kept explicit. These explicit substitutions prevent the representation blowing up exponentially in the size of the original process. The idea of using closures comes from DAG representations used in unification for avoiding exponential blow-up. A sequential substitution that we use here can be seen as a DAG representation of the substitution.
- To avoid the second problem, we first devise a non-deterministic algorithm for testing reachability that does not have to store all the reachable processes, but instead tests it on-the-fly, and then remove nondeterminism using Savitch's theorem [27].

Hence we prove Theorem 4.11, that the model checking problem is solvable in PSPACE. We show this upper bound to be tight in Section 5; Theorem 5.2 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 5.4, 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 5.6, 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.

Usually in model checking, the main bottleneck is the size of the process (which can be very large) since the size of the specification is typically small. Thus a more accurate measure of the complexity of model checking is its program complexity, where we fix the specification and only the program may vary. Often the combined complexity of model checking (that is, the case where both the model and the formula vary) is one exponential higher than the program complexity; this happens for example for finite transition systems against the logic LTL. Here this is not the case—even if we fix the process to be the constant $\mathbf{0}$, the model checking problem remains PSPACE-hard. Although we do not prove PSPACE-hardness for fixed arbitrary formulas, our result is not much weaker: Theorem 5.7 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.

We end the main part of the paper with conclusions in Section 6. Appendixes A and B contain missing details of the encodings and proofs for undecidability results from Section 3. Appendixes C and D contain proofs of properties concerning complexity results stated without proof in Sections 4 and 5, respectively.

Portions of this article appear in two conference papers [11,13].

2. Review of the ambient calculus and logic

We present the ambient calculus with public names (that is, the full calculus [10] apart from name restriction) and the ambient logic. This is the calculus and logic used by Cardelli and Gordon in [9].

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
$\text{in } M$	can enter M	$P \mid Q$	composition
$\text{out } M$	can exit M	$M[P]$	ambient
$\text{open } M$	can open M	$M.P$	action
ε	null	$(n).P$	input
MM'	path	$\langle M \rangle$	output
		$!P$	replication

In the following we will often refer to *replication-free* or *finite-state* fragment of this calculus, which we obtain by removing the replication symbol from the table above.

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, $P \downarrow Q$, which defines the spatial distribution of processes and 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, 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)
$\varepsilon.P \equiv P$	(Struct ε)
$(M.M').P \equiv M.M'.P$	(Struct \cdot)
$P \equiv Q \Rightarrow !P \equiv !Q$	(Struct Repl)
$!\mathbf{0} \equiv \mathbf{0}$	(Struct Repl Zero)
$!(P \mid Q) \equiv !P \mid !Q$	(Struct Repl Par)
$!P \equiv P \mid !P$	(Struct Repl Copy)
$!P \equiv !!P$	(Struct Repl Repl)

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

$n[\text{in } m.P \mid Q] \mid m[R] \rightarrow m[n[P \mid Q] \mid R]$	(Red In)
$m[n[\text{out } m.P \mid Q] \mid R] \rightarrow n[P \mid Q] \mid m[R]$	(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)
<hr/>	
$P \rightarrow Q \Rightarrow P \mid R \rightarrow Q \mid R$	(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)
<hr/>	
$P \equiv n[P'] \mid P'' \Rightarrow P \downarrow P'$	(Loc)

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	
T	true
$\neg \mathcal{A}$	negation
$\mathcal{A} \vee \mathcal{B}$	disjunction
0	void
$\eta[\mathcal{A}]$	location
$\mathcal{A} \mid \mathcal{B}$	composition
$\mathcal{A} @ \eta$	placement
$\exists x. \mathcal{A}$	existential quantification
$\diamond \mathcal{A}$	sometime modality
$\diamond \mathcal{A}$	somewhere modality
$\mathcal{A} \triangleright \mathcal{B}$	guarantee

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 **T**, $\neg \mathcal{A}$, and $\mathcal{A} \vee \mathcal{B}$ embed propositional logic. The formulas **0**, $\eta[\mathcal{A}]$, and $\mathcal{A} \mid \mathcal{B}$ are spatial modalities. A process satisfies **0** if it is structurally congruent to the empty process **0**. It satisfies $\eta[\mathcal{A}]$ 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 $\diamond \mathcal{A}$ (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 $\diamond\mathcal{A}$ 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} . Finally, a process P satisfies $\mathcal{A}\triangleright\mathcal{B}$ if for all P' , the process $P|P'$ guarantees \mathcal{B} assuming that P' satisfies \mathcal{A} .

The satisfaction relation $P \models \mathcal{A}$ formalizes these intuitions.

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

$$\begin{array}{l}
P \models \mathbf{T} \\
P \models \neg\mathcal{A} \quad \triangleq \neg(P \models \mathcal{A}) \\
P \models \mathcal{A} \vee \mathcal{B} \quad \triangleq P \models \mathcal{A} \vee P \models \mathcal{B} \\
P \models \mathbf{0} \quad \triangleq P \equiv \mathbf{0} \\
P \models n[\mathcal{A}] \quad \triangleq \exists P'. P \equiv n[P'] \wedge P' \models \mathcal{A} \\
P \models \mathcal{A} | \mathcal{B} \quad \triangleq \exists P', P''. P \equiv P' | P'' \wedge P' \models \mathcal{A} \wedge P'' \models \mathcal{B} \\
P \models \exists x.\mathcal{A} \quad \triangleq \exists m.P \models \mathcal{A}\{x \leftarrow m\} \\
P \models \diamond\mathcal{A} \quad \triangleq \exists P'. P \rightarrow^* P' \wedge P' \models \mathcal{A} \\
P \models \diamond\mathcal{A} \quad \triangleq \exists P'. P \downarrow^* P' \wedge P' \models \mathcal{A} \\
P \models \mathcal{A}@n \quad \triangleq n[P] \models \mathcal{A} \\
P \models \mathcal{A}\triangleright\mathcal{B} \quad \triangleq \forall P'. P' \models \mathcal{A} \Rightarrow P|P' \models \mathcal{B}
\end{array}$$

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

3. Undecidability results

In this section we show that if either the calculus contains replication or the logic contains guarantee, the problem is undecidable.

3.1. Calculus with replication

We start by showing that model checking for the ambient calculus with public names against the ambient logic without guarantee 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 such as *Concatenate*, *Compare* or *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 $\{(\alpha_1, \beta_1), \dots, (\alpha_n, \beta_n)\}$ 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 PCP is undecidable [24].

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

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

where $start_1$ and $start_2$ are two different ambient names and $Word_i(\gamma)$ is a process representing the word γ (we start with the empty word). Before we give the precise definition of the processes $Word_i(\gamma)$, $Concatenate$ and $Compare$, 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.

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

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

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

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

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

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.

$$\text{Compare} \triangleq \text{compare}[] \mid \text{Initialize}(!(\text{open compare.Consume}(a)) \mid \\ !(\text{opencompare.Consume}(b)))$$

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 $\alpha = \alpha'$ 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

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

where

$$\text{nonempty}(w_i) \triangleq \diamond_{w_i}[(a[\mathbf{T}] \vee b[\mathbf{T}]) \mid \mathbf{T}] \\ \text{empty}(w_i) \triangleq \neg \text{nonempty}(w_i).$$

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. \square

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.

Recently, in two independent works [1,19], it has been shown that the ambient calculus with replication but without private names and communication is actually Turing-complete.

3.2. Logic with guarantee

We investigate in this section the problem of model checking finite-state ambient calculus against formulas that may contain guarantee. First we observe that the model checking problem of such formulas subsumes the satisfiability problem of formulas without guarantee.

Observation 3.2. *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 guarantee) is an undecidable problem.⁴ Thus, it implies:

Theorem 3.3. *The model checking problem of ambient processes without replication and name restriction against formulas with guarantee is undecidable.*

Let us consider the set \mathcal{F} of first-order formulas defined over a countable set of variables x, y, z, \dots and some relational symbols $\{R_1, \dots, R_k\}$, 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 $R_i(x_1, \dots, x_l)$, 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 $R_i(a_1, \dots, a_l)$ where R_i is an l -ary relational symbol and a_1, \dots, 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

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

Theorem 3.4 (Trakhtenbrot [29]). *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 $\llbracket \varphi \rrbracket$ from the ambient logic inductively defined as follows:

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

We identify first-order variables in formulas from \mathcal{F} with variables of the ambient logic. Therefore, free variables of φ and $\llbracket \varphi \rrbracket$ coincide.

⁴ Actually we consider a very small fragment of the logic, in particular without temporal modalities.

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} .

A formula φ has a finite model if there exists a finite structure that is a model of φ .

Lemma 3.5. *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 $P \models \llbracket \varphi \rrbracket$.*

The proof of Lemma 3.5 can be found in the appendix. It is straightforward that Lemma 3.5 and Theorem 3.4 yield the undecidability of the satisfiability problem of the logic without guarantee over ambient processes without replication and name restriction. Hence, Theorem 3.3 follows.

4. A model checking algorithm

From now on we restrict ourselves to the finite-state fragment of the calculus and logic without guarantee. In this section we show that the model checking problem for this fragment of the calculus and logic can be decided in polynomial space.

We start by giving an example (Section 4.1) that requires exponential time and space in the Cardelli and Gordon's algorithm. Then we devise a new representation of processes (Section 4.2) that remains polynomial in the size of the initial process (Section 4.3). In Section 4.4 we present a new model checking algorithm based on this representation.

4.1. A motivating example

The following example shows that in the finite-state calculus the size of reachable processes may grow exponentially, and that there may be a reduction path of exponential length. The algorithm given in [9] uses exponential space to check properties of this example.

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

$$\begin{aligned} \langle \text{in } q.\text{out } q \rangle \mid P_0 &\rightarrow^1 p[\text{in } q.\text{out } q] \mid q[\mathbf{0}] \\ \langle \text{in } q.\text{out } q \rangle \mid P_1 &\rightarrow^2 p[(\text{in } q.\text{out } q)^2] \mid q[\mathbf{0}] \\ \langle \text{in } q.\text{out } q \rangle \mid P_2 &\rightarrow^3 p[(\text{in } q.\text{out } q)^4] \mid q[\mathbf{0}] \\ \langle \text{in } q.\text{out } q \rangle \mid P_k &\rightarrow^{k+1} p[(\text{in } q.\text{out } q)^{2^k}] \mid q[\mathbf{0}] \end{aligned}$$

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

This example points out two facts. First, using a simple representation of processes (such as the one proposed in [9]), 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 [9] 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 the input process. Second, we give a nondeterministic algorithm for testing reachability that uses only polynomial space in the combined size of the problem; then by an application of Savitch's theorem [27] we remove nondeterminism and obtain a deterministic version that itself uses only polynomial space.

4.2. 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 [9]). We also present basic operations on closures and prove that closures indeed simulate the processes they represent. All proofs not in this section (in particular, proofs of Propositions 4.1–4.4) can be found in the appendix.

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.

Annotated processes, substitutions, closures

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

In a sequential substitution $\{n_1 \leftarrow M_1\} \dots \{n_k \leftarrow M_k\}$, the expression M_i lies in the scope of the bindings for the remaining names n_{i+1}, \dots, n_k . We denote by ι the empty sequence of substitutions and treat it as the identity substitution. A sequential substitution σ is said to be *acyclic* if either $\sigma = \iota$ or $\sigma = \{x \leftarrow M\}\sigma'$, where x does not occur in σ' and σ' is an acyclic substitution.

For an annotated process \tilde{P} , we define free and bound names in the same way as for ambient processes. Let $names(\sigma)$ be the set of all names occurring in σ .

We define a partial mapping \mathcal{U} from closures to the set of ambient processes. Intuitively, it unfolds a closure to the process it represents by applying the substitution and

cutting off the prefix defined by the offset. Roughly speaking, the expression $\mathcal{U}(\tilde{P}, \sigma)$ is defined if the offsets within the annotated process do not exceed the length of the expression they are associated with. The unfolding $\mathcal{U}(\tilde{P}, \sigma)$ is defined as follows.

The unfolding $\mathcal{U}(\tilde{P}, \sigma)$ of a closure $\langle \tilde{P}; \sigma \rangle$

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

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

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

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

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

We are only interested in a particular kind of closure, which we refer to as *normal*. Let a closure $\langle \tilde{P}; \sigma \rangle$ be normal if $\mathcal{U}(\tilde{P}, \sigma)$ is defined and if it meets some technical conditions about free and bound names.

Definition 1. A closure $\langle \tilde{P}; \sigma \rangle$ is *normal* if:

- (1) $\mathcal{U}(\tilde{P}, \sigma)$ is defined,
- (2) $bn(\tilde{P}) \cap (fn(\tilde{P}) \cup names(\sigma)) = \emptyset$,
- (3) names occurring in inputs are pairwise different,
- (4) every offset o occurring in the scope of an input in \tilde{P} is equal to 0, and
- (5) σ is acyclic.

The next proposition says that our representation of ambient processes with normal closures preserves their basic properties.

Proposition 4.1 (Structural equivalences). *Let $\langle \prod_{i \in I} \pi_i; \sigma \rangle$ be a normal closure. Then*

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

Next, we present how the reduction and sublocation transitions \rightarrow, \downarrow can be defined on closures. Due to this particular representation and the fact that some part of the ambient process is contained in the sequential substitution, some auxiliary subroutines are needed.

One can see in the definition of \mathcal{U} that only expressions M in the annotated process are affected by the sequential substitution. For the sublocation transition, it is important

to extract the name represented by the expression M under the substitution σ . So, one of those subroutines, $nam(M, \sigma)$, consists in recovering from an expression M the name it effectively represents within the substitution σ .

The reduction transition for a closure $\langle \tilde{P}; \sigma \rangle$ requires some other auxiliary subroutines, which 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' . Intuitively, the outcome of applying the substitution σ to an expression M contained within \tilde{P} is a finite sequence of either capabilities of the form in M' , out M' , open M' , or names not bound by the substitution. We need a subroutine to compute the length of this sequence in terms of capabilities. To keep the algorithm in 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 the first capability to execute from a sequence of capabilities, M , a substitution, σ , and an offset, o . This is computed by $fst(M, o, \sigma)$.

The next 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 to be executed in $\langle \{M(o).\tilde{P}\}; \sigma \rangle$ (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 $split$

$$\begin{aligned}
 nam(n, \{m \leftarrow M\} \sigma) &= \begin{cases} nam(M, \sigma) & \text{if } n = m \\ nam(n, \sigma) & \text{otherwise} \end{cases} \\
 nam(n, \iota) &= n \\
 len(\varepsilon, \sigma) &= 0 \\
 len(M.N, \sigma) &= len(M, \sigma) + len(N, \sigma) \\
 len(M, \sigma) &= 1 \quad \text{if } M \in \{\text{in } N, \text{out } N, \text{open } N\} \\
 len(n, \{m \leftarrow M\} \sigma) &= \begin{cases} len(M, \sigma) & \text{if } n = m \\ len(n, \sigma) & \text{otherwise} \end{cases} \\
 len(n, \iota) &= 1 \\
 fst(M.N, o, \sigma) &= \begin{cases} fst(M, o, \sigma) & \text{if } len(M, \sigma) > o \\ fst(N, o - len(M, \sigma), \sigma) & \text{otherwise} \end{cases} \\
 fst(\text{cap } N, 0, \sigma) &= \text{cap } (nam(N, \sigma)) \quad \text{for cap in } \{\text{in}, \text{out}, \text{open}\} \\
 fst(n, o, \{m \leftarrow M\} \sigma) &= \begin{cases} fst(M, o, \sigma) & \text{if } n = m \\ fst(n, o, \sigma) & \text{otherwise} \end{cases} \\
 split(M(o).\tilde{P}, \sigma) &= \begin{cases} (fst(M, o, \sigma), \{M(o+1).\tilde{P}\}) & \text{if } len(M, \sigma) > o + 1 \\ (fst(M, o, \sigma), \tilde{P}) & \text{otherwise} \end{cases}
 \end{aligned}$$

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.⁵ 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.

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

The auxiliary function *norm*

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

Next, we define a transition relation, $\langle \tilde{P}; \sigma \rangle \rightarrow \langle \tilde{P}'; \sigma' \rangle$, and a sublocation relation, $\langle \tilde{P}; \sigma \rangle \downarrow \langle \tilde{P}'; \sigma' \rangle$, on closures. These relations simulate the reduction and the sublocation relations on processes defined in Section 2.1.

Transitions and sublocations of closures

(Trans In)

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

(Trans Out)

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

(Trans Open)

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

(Trans I/O)

$$\frac{\tilde{P}' = norm(\tilde{P}, \{n \leftarrow M\} \sigma)}{\langle \{(n), \tilde{P}, \langle M \rangle\}; \sigma \rangle \rightarrow \langle \tilde{P}'; \{n \leftarrow M\} \sigma \rangle}$$

(Trans Amb)

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

(Trans Par)

$$\frac{\langle \tilde{P}; \sigma \rangle \rightarrow \langle \tilde{P}'; \sigma' \rangle}{\langle \tilde{P} ++ \tilde{Q}; \sigma \rangle \rightarrow \langle \tilde{P}' ++ \tilde{Q}; \sigma' \rangle}$$

(Loc)

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

⁵ We are not concerned here with 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.

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

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

Proposition 4.2.

- (1) 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.
- (2) If $\langle \tilde{P}; \sigma \rangle$ is normal and $\langle \tilde{P}; \sigma \rangle \rightarrow^* \langle \tilde{P}'; \sigma' \rangle$ then $\langle \tilde{P}'; \sigma' \rangle$ is normal.

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

Proposition 4.3 (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 4.3. It refers to time in the same way as Proposition 4.3 refers to space.

Proposition 4.4 (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 4.1–4.4 are enough to prove that normal closures indeed simulate the processes they represent.

4.3. Size of the representation

We show that closures indeed give a polynomial representation of processes. To do this, we have to bound the size of offsets that occur in closures.

For a given object (a closure or a process) O , by $|O|$ we mean the length of its string representation and by $\|O\|$ the number of nodes in its tree representation. We assume that an offset is represented by a single node in the tree representation.

Lemma 4.5. *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 $((), \langle \rangle, \cdot)$ 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. \square

Proposition 4.6. *Assume $\langle \tilde{P}; \sigma \rangle$ is normal and $\langle \tilde{P}; \sigma \rangle \rightarrow \langle \tilde{P}'; \sigma' \rangle$. Then all offsets used in \tilde{P} and \tilde{P}' can be represented by the same number of bits, polynomial in $|\langle \tilde{P}; \sigma \rangle|$ and, with such a representation, $|\langle \tilde{P}'; \sigma' \rangle| \leq |\langle \tilde{P}; \sigma \rangle|$.*

Proof. A simple induction on the length of the substitution σ' proves that the offsets in \tilde{P}' are bounded by the value $\|\langle \tilde{P}'; \sigma' \rangle\|^{\|\langle \tilde{P}'; \sigma' \rangle\|}$. By Lemma 4.5, 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 $\|\langle \tilde{P}; \sigma \rangle\| \cdot (\lfloor \log(\|\langle \tilde{P}; \sigma \rangle\|) + 1)$ bits. With this representation of offsets, incrementing an offset does not increase the size of its string representation. Thus no transitions can increase the length of the string representations of closures. \square

The following proposition is a key fact in the proof that our model checking algorithm and also the algorithm of Cardelli and Gordon [9] 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. For example, the closure $\langle \{n[\text{in } n(0).\tilde{P}_0], \dots, n[\text{in } n(0).\tilde{P}_k]\}; \sigma \rangle$ has at most k^2 different successors. These two facts do not give, however, the exponential bound on the number of nodes in the tree, which is given by the following proposition.

Proposition 4.7. *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 4.6 and the observation that there are only exponentially many strings of polynomial length. \square

Proposition 4.8. *The reachability problem for normal closures is decidable in PSPACE.*

Proof. Take any instance $\langle \tilde{P}; \sigma \rangle, \langle \tilde{P}'; \sigma' \rangle$ of the reachability problem. To decide whether $\langle \tilde{P}; \sigma \rangle \rightarrow^* \langle \tilde{P}'; \sigma' \rangle$, we first define a nondeterministic algorithm that starting from $\langle \tilde{P}; \sigma \rangle$ guesses an immediate successor of the current closure until it reaches $\langle \tilde{P}'; \sigma' \rangle$ or there are no further successors. By Proposition 4.6 the algorithm requires only polynomial space (we have to store only the current closure and its one immediate successor); Proposition 4.7 implies termination. Finally, using the general statement of Savitch's theorem [27] ($\text{NPSPACE}(S(n)) \subseteq \text{PSPACE}(S(n)^2)$), this non-deterministic algorithm can be turned into a deterministic one. \square

4.4. A new algorithm

We propose a new algorithm, $\text{Check}(\tilde{P}, \sigma, \mathcal{A})$, to check whether the ambient process simulated by $\langle \tilde{P}; \sigma \rangle$ satisfies the closed formula \mathcal{A} . For each ambient process, P , we

only consider the closure, $\mathcal{F}(P)$, obtained using the *folding* function defined as follows. We prove (Proposition 4.10), that $P \models \mathcal{A}$ if and only if $\text{Check}(\mathcal{F}(P), \iota, \mathcal{A})$ returns the Boolean value **T**.

The folding $\mathcal{F}(P)$ of a process P

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

For any process P , the closure $\langle \mathcal{F}(P); \iota \rangle$ is normal and $\mathcal{U}(\mathcal{F}(P), \iota)$ is structurally congruent to P . Furthermore, $\mathcal{F}(P)$ can be computed in linear time in the size of P .

For the model checking problem, $P \models \mathcal{A}$, we may assume without loss of generality that the free names of \mathcal{A} are disjoint from the bound names of P . We denote by $fn(\tilde{P}, \sigma)$ the set $(fn(\tilde{P}) \cup \text{names}(\sigma)) \setminus \text{dom}(\sigma)$.

Computing whether a process satisfies a closed formula

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

An expression $\text{Check}(\tilde{P}, \sigma, \mathcal{A})$ 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 $fn(\mathcal{A}) \cap (\text{bn}(\tilde{P}) \cup \text{dom}(\sigma)) = \emptyset$. Hence, for the model checking problem $P \models \mathcal{A}$ where \mathcal{A} is a closed formula, the expression $\text{Check}(\mathcal{F}(P), \iota, \mathcal{A})$ is normal and moreover we have:

Proposition 4.9. *The model checking algorithm described above preserves the normality of $\text{Check}(\tilde{P}, \sigma, \mathcal{A})$.*

Proposition 4.10. *For all processes P and closed formulas \mathcal{A} , we have $P \models \mathcal{A}$ if and only if $\text{Check}(\mathcal{F}(P), \mathbf{1}, \mathcal{A}) = \mathbf{T}$.*

Theorem 4.11. *Model checking the finite-state ambient calculus against the guarantee-free ambient logic is decidable in PSPACE.*

Proof. To test for a given process P and formula \mathcal{A} whether $P \models \mathcal{A}$ we simply compute the value of $\text{Check}(\mathcal{F}(P), \mathbf{1}, \mathcal{A})$. The only problem is to implement *Check* in such a way that it works in polynomial space.

In the case of $\mathbf{T}, \mathbf{0}, n[\mathcal{A}], \mathcal{A}@n, \neg\mathcal{A}$, the algorithm can directly check whether the respective conditions hold. In the case of $\mathcal{A} \vee \mathcal{B}, \mathcal{A} \mid \mathcal{B}, \exists x.\mathcal{A}, \diamond\mathcal{A}, \heartsuit\mathcal{A}$, we have to be more careful about the space used to compute the value of disjunctions. In a loop we iteratively compute the value of each disjunct, reusing the same space in every iteration. In the case of $\heartsuit\mathcal{A}$ the subroutine computing $\bigvee_{\langle \tilde{P}; \sigma \rangle \rightarrow^* \langle \tilde{P}'; \sigma' \rangle} \text{Check}(\tilde{P}', \sigma', \mathcal{A})$ could look as follows:

```

result ← F
for all  $\langle \tilde{P}'; \sigma' \rangle$  such that  $\langle \tilde{P}; \sigma \rangle \rightarrow^* \langle \tilde{P}'; \sigma' \rangle$ 
  if  $\text{Check}(\tilde{P}', \sigma', \mathcal{A}) = \mathbf{T}$  then result ← T
return(result)

```

By Propositions 4.6 and 4.8, every iteration requires only polynomial space. The cases of $\mathcal{A} \vee \mathcal{B}, \mathcal{A} \mid \mathcal{B}, \exists x.\mathcal{A}, \heartsuit\mathcal{A}$ are similar. Thus, the space $S(k, |\tilde{P}| + |\sigma|)$ used by the algorithm to compute $\text{Check}(\tilde{P}, \sigma, \mathcal{A})$ for formulas \mathcal{A} of depth not exceeding k satisfies the inequality

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

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

Finally, the fact that $\mathcal{F}(P)$ is polynomial in the size of P and the statement of Proposition 4.10 complete the proof. \square

4.5. Extension to name restriction

Name restriction allows declarations of private (local) names for processes in the same way as in the π -calculus; in the process $(\nu n)P$, the name n is made local to P .

Recently, Cardelli and Gordon [7] have presented an extended version of the logic that allows reasoning about restricted names; intuitively, a process P satisfies the formula $n\textcircled{R}\mathcal{A}$ (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 $\mathcal{A}\textcircled{H}n$ (read “hide n then \mathcal{A} ”) if it is possible to hide n in P and then satisfy \mathcal{A} .

In [13] we adapt the algorithm from the previous section to the case with name restriction. To do this, we first bring the processes to a kind of prenex normal form by

using α -renaming of restricted names and the scope extrusion rules of the (extended) congruence relation [10]. In this way we obtain a process of the form $(vn_1)\dots(vn_k)P$ where P essentially does not contain name restriction. Then we extend the algorithm by adding two lines implementing directly the intuitive meaning of the operators \textcircled{R} and $\textcircled{\cup}$ described above.

Although the correctness proof in this case is not very difficult, it is much longer and much more tedious than the one in Section 4 and Appendix C and we do not include it in the present paper. The difficulty lies in the fact that to decide if $(vn)P \models \mathcal{A} \mid \mathcal{B}$ one has to compute all processes Q, R such that $(vn)P \equiv Q \mid R$ (and similarly for other logical constructs); computationally it is not a problem, but the correctness proof is very close to the proof of decidability of structural congruence, which goes beyond the scope of the current paper and can be found in [14].

5. Complexity lower bounds

Below we present lower bounds on the space complexity of model checking finite-state ambient calculus against our modal logic (without guarantee), and also for two significant fragments. Without further qualification, throughout this section, we only consider fragments of the ambient calculus without replication, and fragments of the ambient logic without guarantee.

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 \sum_k^P -complete [28], where \sum_k^P denotes the k th level of the polynomial-time hierarchy. In particular, $\sum_0^P = \text{P}$ and $\sum_1^P = \text{NP}$.

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

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

5.1. The full calculus and logic

We define an encoding of QBF formulas into ambient formulas. This encoding is then used to prove Theorem 5.2, that the complexity of model checking the (finite-state) ambient calculus against the guarantee-free ambient logic is PSPACE-hard.

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 in the ambient logic first defined by Cardelli and Gordon [9]:

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

Then $\mathbf{0} \models \eta = \mu$ if and only if the names η and μ are equal. We encode the \forall and \exists quantifiers over truth values as follows:

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

Encoding QBF formulas as ambient logic formulas

$$\begin{aligned}\llbracket v \rrbracket &\triangleq (v = tt) \\ \llbracket \bar{v} \rrbracket &\triangleq (v = ff) \\ \llbracket \ell_1 \vee \dots \vee \ell_k \rrbracket &\triangleq \llbracket \ell_1 \rrbracket \vee \dots \vee \llbracket \ell_k \rrbracket \\ \llbracket C_1 \wedge \dots \wedge C_k \rrbracket &\triangleq \llbracket C_1 \rrbracket \wedge \dots \wedge \llbracket C_k \rrbracket \\ \llbracket \forall v. \varphi \rrbracket &\triangleq \forall v \in \{ff, tt\}. \llbracket \varphi \rrbracket \\ \llbracket \exists v. \varphi \rrbracket &\triangleq \exists v \in \{ff, tt\}. \llbracket \varphi \rrbracket\end{aligned}$$

The following properties are proved in the appendix. The proof of Lemma 5.1 is by induction on the number of variables quantified in φ .

Lemma 5.1. *Consider a closed quantified Boolean formula φ and its encoding $\llbracket \varphi \rrbracket$ in the ambient logic. The formula φ is valid if and only if the model checking problem $\mathbf{0} \models \llbracket \varphi \rrbracket$ holds.*

Theorem 5.2. *The complexity of model checking the full logic (including name quantification) is PSPACE-hard.*

Proof. Straightforward from Lemma 5.1 since for the fixed ambient process $\mathbf{0}$ solving the model checking problem $\mathbf{0} \models \varphi$ is PSPACE-hard. So in fact the expression complexity, that is, the complexity of checking formulas against a fixed process, is PSPACE-hard. \square

The theorem above holds for any fragment of the logic including Boolean connectives, name quantification, and the location and placement modalities, and for any fragment of the calculus including ambients. This might suggest that the complexity of the model checking problem comes from the quantification in the logic. 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.

5.2. Mobile ambients without I/O, no quantifiers

In this section, we study the complexity of the model checking problem for the fragment of the finite-state ambient calculus without I/O and the fragment of the logic without quantification or guarantee.

For every QBF variable, v , we assume that v and v' are distinct ambient calculus names; η is a meta variable ranging over names.

Encoding QBF formulas as ambient processes and formulas

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

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

In the case where φ is the formula defined previously as an example, one would obtain $\text{enc}(\varphi) = (P, \mathcal{A})$, where P is the process depicted in Fig. 1(a) and where the formula \mathcal{A} is of the form

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

where \mathcal{B} is the formula given by $\llbracket v_1 \vee \bar{v}_2 \vee v_3 \rrbracket \wedge \llbracket \bar{v}_1 \vee v_2 \vee v_3 \rrbracket \wedge \llbracket \bar{v}_3 \rrbracket$.

More detailed explanation. We explain this encoding with reference to the ambient process depicted in Fig. 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' nondeterministically either inside $v_1[\text{pos}[\mathbf{0}]]$ (the Boolean variable v_1 takes the value *tt*) or inside $v_1[\text{neg}[\mathbf{0}]]$ (the Boolean variable v_1 takes the value *ff*). 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 nondeterministic 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 $v_1[\text{pos}[\mathbf{0}]] \mid v_1'[\mathbf{0}]$ and $v_1[\text{neg}[\mathbf{0}]]$.

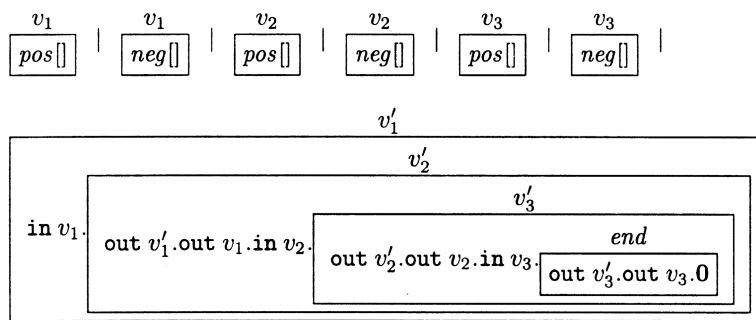
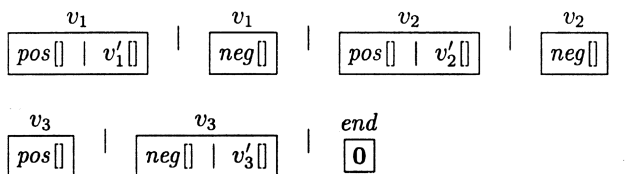
(a) The process P in $enc(\varphi) = (P, \mathcal{A})$ (b) The irreducible process for the interpretation $v_1 \mapsto tt, v_2 \mapsto tt, v_3 \mapsto ff$

Fig. 1. Encoding for mobile ambients without I/O, no quantifiers.

The ambients named v_2, v_3 are kept unchanged and the ambient containing the rest of the interpretation would be of the form $v_2'[\text{in } v_2.v_3'[Q]]$ where Q is the internal of v_3' in the initial process. This computation, consisting of one nondeterministic 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 $v_i[\eta[] | v_i'[]]$ and $v_i[\eta'[]]$ where η, η' are distinct elements from $\{pos, neg\}$. For instance, the irreducible process given in Fig. 1(b) represents the interpretation $v_1 \mapsto tt, v_2 \mapsto tt, v_3 \mapsto ff$.

We 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 $R | v_2'[R']$. 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 $R | v_2'[R']$ reachable from the initial process”. This rephrased statement can be expressed in the ambient logic as $\Box((v_2'[\mathbf{T}] | \mathbf{T}) \Rightarrow \dots)$.

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 $T | v_3'[T']$ 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 $S | v_3'[S']$. This statement can be expressed by means of the ambient logic by the formula $\Diamond((v_3'[\mathbf{T}] | \mathbf{T}) \wedge \dots)$. Finally, the quantification $\exists v_3$ is expressed by $\Diamond((\text{end}[\mathbf{T}] | \mathbf{T}) \wedge \dots)$.

Lemma 5.3. *Assume φ is a closed quantified Boolean formula, and $(P, \mathcal{A}) = \text{enc}(\varphi)$. Then $P \models \mathcal{A}$ if and only if φ is valid.*

Theorem 5.4. *The complexity of model checking mobile ambients without I/O against the quantifier-free logic is PSPACE-hard.*

Proof. Straightforward from the PSPACE-completeness of the validity for QBF and from Lemma 5.3, taking into account that for $\text{enc}(\varphi) = (P, \mathcal{A})$, both P and \mathcal{A} are of polynomial size with respect to φ . \square

5.3. Immobile ambients with I/O, no quantifiers

In this section, we study the complexity of the model checking problem for the fragment of the finite-state ambient calculus without action prefix and the fragment of the logic without quantification or guarantee.

We consider fixed names end , C , and D . For every QBF variable v , we assume that v, \bar{v}, v', v'' are distinct ambient names. For a name v_i' , let

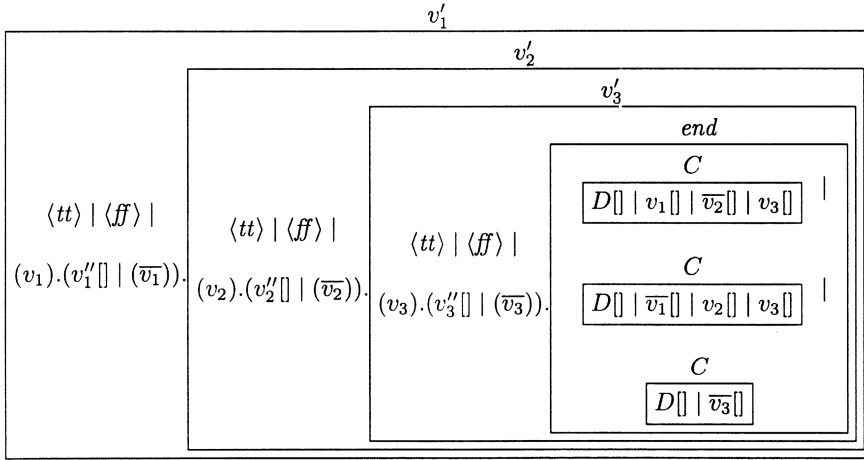
$$\text{Inst}(v_i') \triangleq v_i'[\mathbf{T}] | \mathbf{T} \quad \text{Inst}^+(v_i') \triangleq v_i'[v_i''[\mathbf{T}] | \mathbf{T}] | \mathbf{T}$$

and for the name end ,

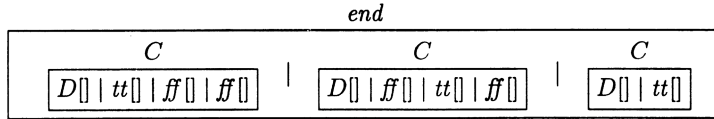
$$\text{Inst}(\text{end}) \triangleq \text{end}[\mathbf{T}] | \mathbf{T} \quad \text{Inst}^+(\text{end}) \triangleq \text{end}[\text{end}'[\mathbf{T}] | \mathbf{T}] | \mathbf{T}$$

Encoding QBF formulas as ambient processes and formulas

$$\begin{aligned} [v] &= v[\mathbf{0}] \\ [\bar{v}] &= \bar{v}[\mathbf{0}] \\ [\ell_1 \vee \dots \vee \ell_k] &= D[\mathbf{0}] | [\ell_1] | \dots | [\ell_k] \\ \text{enc}(C_1 \wedge \dots \wedge C_k) &= (\text{end}[C[[C_1]] | \dots | C[[C_k]]], \\ &\quad \Box((D[\mathbf{0}] | \mathbf{T}) \Rightarrow (t[\mathbf{0}] | \mathbf{T}))) \\ \text{enc}(\exists v.\varphi) &= (v'[\langle t \rangle | \langle ff \rangle] | (v).(v''[\Box | (\bar{v}).\eta[P]]], \\ &\quad \mathbf{T} | v'[\Diamond((\text{Inst}(\eta) \wedge \neg \text{Inst}^+(\eta)) \wedge \mathcal{A})]) \\ &\quad \text{where } \text{enc}(\varphi) = (\eta[P], \mathcal{A}) \\ \text{enc}(\forall v.\varphi) &= (v'[\langle t \rangle | \langle ff \rangle] | (v).(v''[\Box | (\bar{v}).\eta[P]]], \\ &\quad \mathbf{T} | v'[\Box((\text{Inst}(\eta) \wedge \neg \text{Inst}^+(\eta)) \Rightarrow \mathcal{A})]) \\ &\quad \text{where } \text{enc}(\varphi) = (\eta[P], \mathcal{A}) \end{aligned}$$



(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 $v_1 \mapsto tt, v_2 \mapsto tt, v_3 \mapsto ff$

Fig. 2. Encoding for immobile ambients with I/O, no quantifiers.

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

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

More detailed explanation. 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 $C_1 \wedge \dots \wedge C_k$ is encoded in the ambient process itself, inside an ambient named end . For instance, in Fig. 2(a) for our example, the ambient $end[C[D[] | v_1[] | \bar{v}_2[] | v_3[]] | C[D[] | \bar{v}_1[] | v_2[] | v_3[]] | C[D[] | \bar{v}_3[]]]$ 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 Fig. 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 nondeterministic 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 \bar{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 \bar{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 replace in the ambient end all the names corresponding to literals by Boolean values tt and ff . As an example, in Fig. 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 Fig. 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 Fig. 2(b), every ambient named C contains a sub-ambient $tt[\]$. This can be tested with the formula $\mathcal{B} = \square((D[\mathbf{0} \mid \mathbf{T}] \Rightarrow (tt[\mathbf{0} \mid \mathbf{T}])))$, which is exactly the formula given by $enc(C_1 \wedge C_2 \wedge C_3)$.

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

$$\begin{aligned} & \mathbf{T} \mid v_1' [\square((Inst(v_2') \wedge \neg Inst^+(v_2')) \\ & \Rightarrow (\mathbf{T} \mid v_2' [\diamond((Inst(v_3') \wedge \neg Inst^+(v_3')) \\ & \wedge (\mathbf{T} \mid v_3' [\diamond((Inst(end) \wedge \neg Inst^+(end) \wedge \mathcal{B}))]))))] \end{aligned}$$

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

$$\mathbf{T} \mid v_1' [\square((Inst(v_2') \wedge \neg Inst^+(v_2')) \Rightarrow \dots)]$$

This formula is model checked against the process P given in Fig. 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 (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 \bar{v}_1 have been replaced by Boolean values, against the formula

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

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 , \bar{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 , \bar{v}_1 , v_2 , \bar{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

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

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

Lemma 5.5. *Assume φ is a closed quantified Boolean formula, and $(P, \mathcal{A}) = enc(\varphi)$. Then $P \models \mathcal{A}$ if and only if φ is valid.*

Theorem 5.6. *The complexity of model checking immobile ambients with I/O against the quantifier-free logic is PSPACE-hard.*

Proof. This follows from the PSPACE-completeness of validity for QBF, from Lemma 5.5 taking into account that for $enc(\varphi) = (P, \mathcal{A})$, both P and \mathcal{A} are of polynomial size with respect to φ . \square

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 perform two successive tests for reachability; 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:

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

The revised encoding

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

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

$$enc(\forall v.\varphi, i) = (a_i[\langle tt \rangle \mid \langle ff \rangle] \mid (v).(a_i[] \mid (\bar{v})).P, \\ \mathbf{T} \mid a_i[\square(Inst(a_{i+1}) \wedge Inst^+(a_{i+1}) \Rightarrow \mathcal{A}))]) \\ \text{where } enc_{\forall}(\varphi, i) = (P, \mathcal{A})$$

$$enc(\exists v.\varphi, i) = (a_i[\langle tt \rangle \mid \langle ff \rangle] \mid (v).(a_i[] \mid (\bar{v})).P, \\ \mathbf{T} \mid a_i[\diamond(Inst(a_{i+1}) \wedge Inst^+(a_{i+1}) \wedge \mathcal{A}))]) \\ \text{where } enc_{\exists}(\varphi, i) = (P, \mathcal{A})$$

$$enc_{\forall}(\exists v.\varphi, i) = enc(\exists v.\varphi, i + 1)$$

$$enc_{\forall}(\forall v.\varphi, i) = (\langle tt \rangle \mid \langle ff \rangle \mid (v).(\bar{v}).P, \mathcal{A}) \quad \text{where } enc_{\forall}(\varphi, i) = (P, \mathcal{A})$$

$$enc_{\exists}(\forall v.\varphi) = enc(\forall v.\varphi, i + 1)$$

$$enc_{\exists}(\exists v.\varphi, i) = (\langle tt \rangle \mid \langle ff \rangle \mid (v).(\bar{v}).P, \mathcal{A}) \quad \text{where } enc_{\exists}(\varphi, i) = (P, \mathcal{A})$$

$$enc(C_1 \wedge \dots \wedge C_k, i) = (a_i[C[\llbracket C_1 \rrbracket] \mid \dots \mid C[\llbracket C_k \rrbracket]], \square((D[\mathbf{0}] \mid \mathbf{T}) \Rightarrow tt[\mathbf{0}] \mid \mathbf{T}))$$

$$\llbracket \ell_1 \vee \dots \vee \ell_k \rrbracket = D[\mathbf{0}] \mid \llbracket \ell_1 \rrbracket \mid \dots \mid \llbracket \ell_k \rrbracket$$

$$\llbracket v \rrbracket = v[]$$

$$\llbracket \bar{v} \rrbracket = \bar{v}[]$$

The statement of Lemma 5.5 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 $\text{enc}(\varphi) = (P, \mathcal{A})$ and $\text{enc}(\varphi') = (P', \mathcal{A}')$ then $\mathcal{A} = \mathcal{A}'$.

Theorem 5.7. *For every integer k there exists a formula \mathcal{A}_k^\exists such that the complexity of model checking processes against \mathcal{A}_k^\exists is \sum_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 , $\text{enc}(\varphi) = (P_\varphi, \mathcal{A}_k^\exists)$. Due to the remark above, we know that this formula exists and furthermore, is of size polynomial in k .

Now, by Lemma 5.5, 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 $\text{enc}(\varphi) = (P_\varphi, \mathcal{A}_k^\exists)$. Thus, since the size of P_φ is polynomial in the size of φ , the theorem follows. \square

6. Conclusion

We study in this paper the model checking problem for the ambient calculus with public names against the ambient logic. We show that if either the calculus contains replication or the logic contains guarantee then the problem is undecidable, which answers a question stated in [9]. In the decidable case of the replication-free ambient calculus with public names and the guarantee-free ambient logic we prove that the problem 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. We use this representation together with a new algorithm to prove the PSPACE upper bound.

We also show that there is little chance to find polynomial algorithms for interesting subproblems: model checking remains PSPACE-hard even for quite simple fragments of the calculus and the logic.

Possible directions for future work include investigations of the model checking problem for extensions of the logic and the calculus in the decidable case.

As discussed in Section 4.5, Cardelli and Gordon [7] present an extended version of the logic that allows reasoning about restricted names, and another paper [13] shows that the algorithm presented here can be directly extended to deal with name restriction. A subsequent paper [12] shows that model checking a finite-control version of the ambient calculus with restricted names is PSPACE-complete.

Cardelli and Gordon [9] discuss connections between the ambient logic and several substructural logics [30,17,16,23]. We are aware of no prior studies of model checking a logic with spatial operators against a process calculus. Dam [15] proposes such a logic for the CCS process calculus; he obtains various axiomatizations but does not consider model checking. His logic is more extensional than the ambient logic: satisfaction is preserved by bisimulation equivalence, which is not the case for the ambient logic.

The ambient logic is one of several *spatial logics* that have attracted attention recently. In a spatial logic, the truth of a formula depends on a spatial structure, such as a system of ambients. Apart from the ambient logic, other examples include logics for concurrent objects [4], heaps [25,20,22], trees [6], graphs [5], and the π -calculus [2,3]. Spatial logics are being applied as notations for specifying and verifying programs, and as query languages for semistructured data. The results of this paper apply directly only to the ambient logic, but we hope they may be instructive in the study of other spatial logics.

Acknowledgements

The anonymous referees made useful comments.

Appendix A. Encoding of PCP: concatenation and comparison of words

A.1. Concatenation

Here we show how to rewrite $Word_i(\gamma) \mid Concatenate_i(\gamma')$ to $Word_i(\gamma' \cdot \gamma) \mid start_i[\]$. For this, we need precise definition of $Word_i$ and $Concatenate_i$. For $i=1,2$ we introduce fresh ambient names $word_i, c_i, n_i, s_i, v_i, w_i$; 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

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

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[in\ v_i.in\ w_i \mid \sigma_{k'}[\dots \sigma'_1[P] \dots]] \\ MvIn_i(\sigma'_1 \dots \sigma_{k'}) &\triangleq n_i[in\ w_i.in\ s_i.in\ \sigma'_1 \dots in\ \sigma_{k'}] \\ MvOut(\sigma'_1 \dots \sigma_{k'}) &\triangleq out\ \sigma_{k'} \dots out\ \sigma'_1.out\ s_i \\ Continue_i &\triangleq open\ w_i.v_i[MvOut(\gamma').Tinue_i] \\ Tinue_i &\triangleq in\ w_i \mid w_i[open\ n_i \mid open\ s_i.out\ v_i.Nue_i] \\ Nue_i &\triangleq open\ v_i.start_i[out\ w_i.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

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

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

$$\begin{aligned} & \text{word}_i[!\text{open } c_i \mid \text{String}'(\gamma', \text{Continue}_i) \mid \\ & \quad w_i[\text{in } s_i.\text{in } \sigma_1' \dots \text{in } \sigma_k' \mid \text{String}(\gamma)]], \end{aligned}$$

$w_i[...]$ goes inside $\text{String}'(...)$

$$\begin{aligned} & \text{word}_i[!\text{open } c_i \mid \\ & \quad \text{String}'(\gamma', (w_i[\text{String}(\gamma)] \mid \text{Continue}_i))], \end{aligned}$$

Continue_i opens w_i and v_i moves out of $s_i[\gamma']$

$$\text{word}_i[!\text{open } c_i \mid \text{String}'(\gamma' \cdot \gamma, \mathbf{0}) \mid v_i[\text{Tinue}_i]],$$

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

$$\text{word}_i[!\text{open } c_i \mid v_i[\text{in } w_i \mid w_i[\text{open } n_i \mid \text{out } v_i.\text{Nue}_i \mid \text{String}(\gamma' \cdot \gamma)]]],$$

w_i gets out of v_i and v_i gets into w_i

$$\text{word}_i[!\text{open } c_i \mid w_i[\text{open } n_i \mid v_i[] \mid \text{Nue}_i \mid \text{String}(\gamma' \cdot \gamma)]],$$

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

$$\text{Word}_i(\gamma' \cdot \gamma) \mid \text{start}_i[]$$

which is the desired process. Note that since guarded processes cannot be reduced, this was the only possible execution of $\text{Word}_i(\gamma) \mid \text{Concatenate}_i(\gamma')$.

A.2. Comparing the words

First we recall the definition of *Compare* and define the two missing processes.

$$\begin{aligned} \text{Compare} \triangleq & \text{compare}[] \mid \text{Initialize}(!(\text{open } \text{compare}.\text{Consume}(a)) \mid \\ & \quad !(\text{open } \text{compare}.\text{Consume}(b))) \end{aligned}$$

$$\text{Initialize} \triangleq \text{open } \text{start}_1.\text{open } \text{start}_2.\text{open } \text{word}_1.\text{open } \text{word}_2$$

$$\text{Consume}(\sigma) \triangleq n_1[\text{in } w_1.\text{open } \sigma.(\text{open } n_1 \mid \text{Nsume}(\sigma))]$$

where

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

Then

$$\text{start}_1[] \mid \text{start}_2[] \mid \text{Word}_1(\alpha) \mid \text{Word}_2(\beta) \mid \text{Compare}$$

reduces to

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

The two processes $!\text{open } c_i$ 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

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

reduces to

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

This can be easily checked by the reader. The process $\text{Consume}(\sigma)$ 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 $\text{open } n_1$ for the next iteration, and as $\text{Nsume}(\sigma)$ goes out of w_1 ; then it repeats the same thing with w_2 and leaves the ambient $\text{compare}[]$ at the top level. Note that if the two words u and v start with two different letters a and b then the process $w_1[\text{open } n_1 \mid \text{String}(\alpha)] \mid w_2[\text{open } n_2 \mid \text{String}(\beta)] \mid \text{Consume}(\sigma)$ 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.

Appendix B. Satisfiability of the ambient logic

We give here the proof of Lemma 3.5. We consider the relation $\rightsquigarrow_{\text{Proc}}$ 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 $\mathcal{S} \rightsquigarrow_{\text{Proc}} P$ if:

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

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

We prove the following proposition which implies Lemma 3.5 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 $\mathcal{S} \rightsquigarrow_{\text{Proc}} P$ then $P \models \llbracket \varphi \rrbracket \alpha$,*
- (ii) *let P be an ambient process without replication and name restriction and α be a mapping from variables of φ to names. If $P \models \llbracket \varphi \rrbracket \alpha$ and $P \rightsquigarrow_{\text{Struct}} \mathcal{S}$ then $\mathcal{S}, \alpha \models \varphi$.*

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

- For $\varphi = R_i(x_1, \dots, x_l)$:
 - Case (i): $\mathcal{S}, \alpha \models R_i(x_1, \dots, x_l)$. Therefore, for the valuation α equal to $\{x_1 \leftarrow a_1, \dots, x_l \leftarrow a_l\}$, $R_i(a_1, \dots, a_l)$ belongs to \mathcal{S} . Therefore, by definition of $\mathcal{S} \rightsquigarrow_{Proc} P$, P is structurally congruent to $r_i[a_1[\dots[a_l[\mathbf{0}]]\dots]]|P'$ for some P' . So, P is a model of $(r_i[x_1[\dots[x_l[\mathbf{0}]]\dots]]|\mathbf{T})\alpha$, that is $P \models \llbracket R_i(x_1, \dots, x_l) \rrbracket \alpha$.
 - Case (ii): $P \models \llbracket R_i(x_1, \dots, x_l) \rrbracket \alpha$. Then, for $\alpha = \{x_1 \leftarrow a_1, \dots, x_l \leftarrow a_l\}$, $P \models r_i[a_1[\dots[a_l[\mathbf{0}]]\dots]]|\mathbf{T}$. Hence, by definition of the satisfaction relation, there exists P' such that $P \equiv r_i[a_1[\dots[a_l[\mathbf{0}]]\dots]]|P'$. Since $P \rightsquigarrow_{Struct} \mathcal{S}$, we have that a_1, \dots, a_l belong to \mathcal{D} and $R_i(a_1, \dots, a_l)$ belongs to \mathcal{S} . Thus, $\mathcal{S}, \alpha \models R_i(x_1, \dots, x_l)$.
- 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, $P \models \llbracket \varphi' \rrbracket \alpha$ and $P \models \llbracket \varphi'' \rrbracket \alpha$. Thus, $P \models \llbracket \varphi \rrbracket \alpha$.
 - Case (ii): dual to 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 $P \not\models \llbracket \varphi' \rrbracket \alpha$ 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, $P \not\models \llbracket \varphi' \rrbracket \alpha$ holds. Therefore, $P \models \neg(\llbracket \varphi' \rrbracket \alpha)$. Finally, $P \models \llbracket \varphi \rrbracket \alpha$.
 - Case (ii): $P \models \llbracket \neg \varphi' \rrbracket \alpha$. So, $P \not\models \llbracket \varphi' \rrbracket \alpha$. 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 $\mathcal{S} \rightsquigarrow_{Proc} P$, so there exists P' such that $P \equiv d[a[\mathbf{0}]]|P'$. Moreover, by induction hypothesis, $P \models \llbracket \varphi' \rrbracket \alpha\{x \leftarrow a\}$. Hence, $P \models (d[a[\mathbf{0}]]|\mathbf{T}) \wedge (\llbracket \varphi' \rrbracket \alpha\{x \leftarrow a\})$. Hence, $P \models ((d[x[\mathbf{0}]]|\mathbf{T}) \wedge \llbracket \varphi' \rrbracket \alpha\{x \leftarrow a\})$. So, $P \models \exists x.((d[x[\mathbf{0}]]|\mathbf{T}) \wedge \llbracket \varphi' \rrbracket \alpha)$.
 - Case (ii): $P \models \llbracket \exists x \varphi' \rrbracket \alpha$, that is by definition $P \models (\exists x.((d[x[\mathbf{0}]]|\mathbf{T}) \wedge \llbracket \varphi' \rrbracket \alpha))$. Therefore, by definition of satisfiability, there exists a name a such that $P \models ((d[x[\mathbf{0}]]|\mathbf{T}) \wedge \llbracket \varphi' \rrbracket \alpha\{x \leftarrow a\})$. This implies that
 - there exists P' such that $P \equiv d[a[\mathbf{0}]]|P'$,
 - $P \models \llbracket \varphi' \rrbracket \alpha\{x \leftarrow a\}$.
 As $P \rightsquigarrow_{Struct} \mathcal{S}$, 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, $\mathcal{S}, \alpha \models \exists x \varphi'$. \square

Appendix C. Correctness proofs

This appendix contains proofs of results stated in Section 4.

C.1. Proof of proposition 4.1

Proposition 4.1 concerns the relationship between normal closures and structural congruence. In this appendix we develop enough facts about closures and structural congruence to prove it.

We begin with a proposition that normality is preserved by decomposition with ambient or parallel composition.

Proposition C.1.

- $\langle \tilde{P}; \sigma \rangle$ and $\langle \tilde{Q}; \sigma \rangle$ are normal and $fn(\tilde{P}) \cap bn(\tilde{Q}) = bn(\tilde{P}) \cap fn(\tilde{Q}) = bn(\tilde{P}) \cap bn(\tilde{Q}) = \emptyset$ iff $\langle \tilde{P} ++ \tilde{Q}; \sigma \rangle$ is normal.
- for all expressions M such that M does not contain names from $bn(\tilde{P})$, $\langle \{M[\tilde{P}]\}; \sigma \rangle$ is normal iff $\langle \tilde{P}; \sigma \rangle$ is normal.

Proof. For the first point: from right to left, it is straightforward from the definition of \mathcal{U} that if $\mathcal{U}(\tilde{P} ++ \tilde{Q}, \sigma)$ is defined then both $\mathcal{U}(\tilde{P}, \sigma)$ and $\mathcal{U}(\tilde{Q}, \sigma)$ are so. As $fn(\tilde{P} ++ \tilde{Q}) = fn(\tilde{P}) \cup fn(\tilde{Q})$ and $bn(\tilde{P} ++ \tilde{Q}) = bn(\tilde{P}) \cup bn(\tilde{Q})$, if $bn(\tilde{P} ++ \tilde{Q}) \cap (fn(\tilde{P} ++ \tilde{Q}) \cup names(\sigma)) = \emptyset$ then $bn(\tilde{P}) \cap (fn(\tilde{P}) \cup names(\sigma)) = bn(\tilde{Q}) \cap (fn(\tilde{Q}) \cup names(\sigma)) = \emptyset$. If for $\tilde{P} ++ \tilde{Q}$ bound variables occur at most once within an input and offsets in the scope of an input are equal to 0, then it is so for \tilde{P} and \tilde{Q} . The last condition for normality on sequential substitution is obvious. The three other conditions follow directly from the normality of $\langle \tilde{P} ++ \tilde{Q}; \sigma \rangle$. From left to right, the definition of \mathcal{U} implies that if $\langle \tilde{P}; \sigma \rangle$ and $\langle \tilde{Q}; \sigma \rangle$ are defined then $\langle \tilde{P} ++ \tilde{Q}; \sigma \rangle$ is defined. Now, $fn(\tilde{P} ++ \tilde{Q}) \cap bn(\tilde{P} ++ \tilde{Q}) = (fn(\tilde{P}) \cup fn(\tilde{Q})) \cap (bn(\tilde{P}) \cup bn(\tilde{Q}))$. We have $fn(\tilde{P}) \cap bn(\tilde{Q}) = bn(\tilde{P}) \cap fn(\tilde{Q}) = \emptyset$ by assumption and $fn(\tilde{P}) \cap bn(\tilde{P}) = fn(\tilde{Q}) \cap bn(\tilde{Q}) = \emptyset$ as $\langle \tilde{P}; \sigma \rangle$ and $\langle \tilde{Q}; \sigma \rangle$ are normal. So, $fn(\tilde{P} ++ \tilde{Q}) \cap bn(\tilde{P} ++ \tilde{Q}) = \emptyset$. By normality of $\langle \tilde{P}; \sigma \rangle$ and $\langle \tilde{Q}; \sigma \rangle$, $names(\sigma) \cap bn(\tilde{R}) = \emptyset$ for $\tilde{R} \in \{\tilde{P}, \tilde{Q}\}$. So, $names(\sigma) \cap bn(\tilde{P} ++ \tilde{Q}) = \emptyset$. $\langle \tilde{P}; \sigma \rangle$ and $\langle \tilde{Q}; \sigma \rangle$ being normal and as by assumption $bn(\tilde{P}) \cap bn(\tilde{Q}) = \emptyset$, every input variable occurs at most once within an input in $\tilde{P} ++ \tilde{Q}$. The last conditions on offsets in the scope of an input and on sequential substitution is obvious.

For the second point: it is easy to see that $\mathcal{U}(\{M[\tilde{P}]\}, \sigma)$ is defined iff $\mathcal{U}(\tilde{P}, \sigma)$ is so. The set of names occurring free in M is exactly the set $fn(\{M[\mathbf{0}]\})$. Now, as $bn(\{M[\tilde{P}]\}) = bn(\tilde{P})$ and $fn(\{M[\tilde{P}]\}) = fn(\tilde{P}) \cup fn(\{M[\mathbf{0}]\})$, $fn(\{M[\tilde{P}]\}) \cap bn(\{M[\tilde{P}]\})$ is empty iff $fn(\tilde{P}) \cap bn(\tilde{P})$ is empty (taking into account the assumption that $bn(\tilde{P}) \cap fn(\{M[\mathbf{0}]\}) = \emptyset$) and $bn(\{M[\tilde{P}]\}) \cap names(\sigma) = bn(\tilde{P}) \cap names(\sigma) = \emptyset$. Finally, the last three statements are obvious to check. \square

In the proof of Proposition 4.1 we will have to show that some processes are equivalent if and only if some conditions hold. In particular, we will have to show that if these conditions do not hold, the processes are not equivalent. Although it is relatively easy to prove equivalence of processes, it is not so easy to prove their inequivalence (which requires showing that no equivalence proof exists). We use Theorem C.2 and Propositions C.3–C.5 below as tools for proving inequivalences needed in Proposition 4.1.

Let us consider Σ the signature used to build processes from the ambient calculus with public names. The signature Σ contains an infinite number of constants used as names. It contains moreover $\mathbf{0}$ and ε as constant symbols, the capabilities *in*, *out*, *open* and $\langle \rangle$ as unary function symbols. Finally, the binary function symbols $|$, $[],$ $.,$ $()$ belong to Σ .

Let us denote T_Σ the set of all terms over Σ . Any ambient process from the ambient calculus with public names can be written as a term over this vocabulary. And of course, some terms from T_Σ are not ambient process, as for instance, $\langle \mathbf{0} | \mathbf{0} \rangle$.

The set T_Σ induces a canonical algebra that we denote \mathcal{T}_Σ : the algebra \mathcal{T}_Σ has for carrier the set T_Σ and each function symbols from Σ is interpreted syntactically in \mathcal{T}_Σ .

The structural congruence relation \equiv defined in Section 2.1 over pairs of ambient processes can be viewed as a relation defined over $\mathcal{T}_\Sigma \times \mathcal{T}_\Sigma$. One should notice that the set of axioms defining \equiv is a set of definite Horn clauses, and thus, $(\mathcal{T}_\Sigma, \equiv)$ is a Herbrand model for this set of axioms. Moreover, as we consider the least relation satisfying these axioms, the structure $(\mathcal{T}_\Sigma, \equiv)$ is the least Herbrand model for this set of axioms. This implies that two processes P, Q are structurally equivalent if and only if $P \equiv Q$ belongs to the least Herbrand model of these axioms.

If \equiv is not assumed to be the least relation satisfying the axioms but for instance the greatest one, then one would have $P \equiv Q$ whatever P, Q are.

The following theorem is a direct consequence of two well known facts [21], that (1) every model of a set of Horn clauses can be translated to a Herbrand model, and (2) that every Herbrand model contains the least Herbrand model. Essentially, the theorem says that anything that does not belong to some model cannot belong to the least model.

Theorem C.2. *Let \mathcal{S} be a set of definite Horn clauses defining a relation symbol \equiv . Then for all algebras \mathbf{A} , for all structures \mathbf{R} defined over \mathbf{A} and giving an interpretation for \equiv such that $\mathbf{R} \models \mathcal{S}$,*

$$\mathbf{R} \models s \equiv t \text{ if } (\mathcal{T}_\Sigma, \equiv) \models s \equiv t$$

That is, if there exists a structure \mathbf{R} such that $\mathbf{R} \models \mathcal{S}$ and $\mathbf{R} \models s \not\equiv t$, then $(\mathcal{T}_\Sigma, \equiv) \models s \not\equiv t$.

Let us consider now the algebra $\hat{\mathbf{A}}$ defined over Σ ; the carrier $D_{\hat{\mathbf{A}}}$ is the least set such that

- the constants from Σ except ε and $\mathbf{0}$ belong to $D_{\hat{\mathbf{A}}}$,
- the empty string and the empty multiset belong to $D_{\hat{\mathbf{A}}}$,
- for any $d_1, d_2 \in D_{\hat{\mathbf{A}}}$, the items in d_1 , out d_1 , open d_1 , $\langle d_1 \rangle$, $(d_1)d_2$ and $d_1[d_2]$ belong to $D_{\hat{\mathbf{A}}}$,
- for any $d_1, \dots, d_n \in D_{\hat{\mathbf{A}}}$, the string $d_1 \dots d_n$ and the multiset $\{d_1, \dots, d_n\}$ belong to $D_{\hat{\mathbf{A}}}$.

The function symbols from Σ are interpreted in $\hat{\mathbf{A}}$ as follows.

- The constants from Σ except ε and $\mathbf{0}$ are interpreted syntactically.
- The constants ε and $\mathbf{0}$ are interpreted respectively as the empty string and as the empty multiset.
- The function symbols in, out, open, $\langle \rangle$, $()$ and $[]$ are interpreted syntactically.
- For the function symbol $..$: $d_1.d_2$ is the string obtained by concatenation of d_1 and d_2 if both d_1 and d_2 are strings. Otherwise, elements from $\{d_1, d_2\}$ that are not strings are transformed into a string of length one and then, the concatenation is performed.
- For the function symbol $|$: $d_1 | d_2$ is the multiset obtained by union of d_1 and d_2 if

both d_1 and d_2 are multisets. Otherwise, elements from $\{d_1, d_2\}$ that are not multisets are transformed into a singleton multiset and then, the union is performed.

The algebra $\hat{\mathbf{A}}$ is extended into a structure $\hat{\mathbf{R}}$ in which \equiv is interpreted as the binary relation \doteq over $D_{\hat{\mathbf{A}}} \times D_{\hat{\mathbf{A}}}$. The relation \doteq is defined recursively as follows: $d \doteq d'$ iff

- d and d' are both the empty string.
- d and d' are both composed strings such that d_h and d'_h , the first two elements of d, d' satisfy $d_h \doteq d'_h$ and d_t and d'_t the two strings obtained by removing the first element in respectively d and d' satisfy $d_t \doteq d'_t$.
- d and d' are both the empty multiset.
- d and d' are both nonempty multisets and there exists d_e and d'_e respectively in d and d' such that $d_e \doteq d'_e$ and $d \setminus d_e \doteq d' \setminus d'_e$.
- d and d' are respectively of the form $\langle d_1 \rangle$ and $\langle d'_1 \rangle$ and $d_1 \doteq d'_1$.
- d and d' are respectively of the form $\text{cap } d_1$ and $\text{cap } d'_1$ and $d_1 \doteq d'_1$ where cap belongs to $\{\text{in}, \text{out}, \text{open}\}$.
- d and d' are respectively of the form $d_1[d_2]$ and $d'_1[d'_2]$ and $d_1 \doteq d'_1$, $d_2 \doteq d'_2$.
- d and d' are respectively α -equivalent to $(d_1)d_2$ and $(d_1)d'_2$ where d_1 is a constant and $d_2 \doteq d'_2$.

Proposition C.3. $\hat{\mathbf{R}}$ is a model of the axioms for \equiv .

Proof. By case inspection. \square

Proposition C.4. For any process P , for any M , for any name n , for any $\text{cap} \in \{\text{in}, \text{out}, \text{open}\}$:

- For any process Q , we have $\mathbf{0} \not\equiv M[P]$, $\mathbf{0} \not\equiv (n).P$, $\mathbf{0} \not\equiv \langle M \rangle$, $\mathbf{0} \not\equiv \text{cap } M.P$ and $\mathbf{0} \not\equiv P|Q$ if $P \not\equiv \mathbf{0}$.
- If $P \not\equiv \mathbf{0}$, then for any processes Q, P' such that $Q \not\equiv \mathbf{0}$, we have $P|Q \not\equiv M[P']$, $P|Q \not\equiv (n).P'$, $P|Q \not\equiv \langle M \rangle$, $P|Q \not\equiv \text{cap } M.P'$.
- For any processes Q, P' and for any M' , we have $M[P] \not\equiv (n).Q$, $M[P] \not\equiv \langle M' \rangle$, $M[P] \not\equiv \text{cap } M'.P'$ and $M[P] \not\equiv M'[P']$ if M, M' are two different sequences or if $P \not\equiv P'$.
- For any M' , we have $\langle M \rangle \not\equiv \text{cap } M'.P$, $\langle M \rangle \not\equiv (n).P$ and $\langle M \rangle \not\equiv \langle M' \rangle$ if M, M' are two different sequences.
- For any process Q we have $(n).P \not\equiv \text{cap } M.Q$.
- For any process Q , for any M' and for any capability $\text{cap}' \in \{\text{in}, \text{out}, \text{open}\}$, we have $\text{cap } M.P \not\equiv \text{cap}' M'.Q$ if either $\text{cap} \neq \text{cap}'$ or M, M' are two different sequences or if $P \not\equiv Q$.

Proof. It is easy to check that all the statements above holds for $\hat{\mathbf{R}}$. Using Proposition C.3 with Theorem C.2, those statements hold for ambient processes and \equiv . \square

Proposition C.5. For any sequential substitution σ , for any prime π such that $\langle \{\pi\}; \sigma \rangle$ is normal, $\mathcal{U}(\pi, \sigma) \not\equiv \mathbf{0}$.

Proof. Straightforward from the definition of \mathcal{U} and Proposition C.3. \square

Restatement of Proposition 4.1. Let $\langle \prod_{i \in I} \pi_i; \sigma \rangle$ be a normal closure. Then

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

Proof. For the first point, if $I = \emptyset$ then $\tilde{P} = \{\}$; so, by definition for \mathcal{U} , $\mathcal{U}(\tilde{P}, \sigma) \equiv \mathbf{0}$. Now for the other direction, the closure $\langle \tilde{P}; \sigma \rangle$ being normal, if I is not empty, then by Proposition C.4 and the definition for \mathcal{U} , $\mathcal{U}(\prod_{i \in I} \pi_i, \sigma) \neq \mathbf{0}$.

For the second point, for the direction from right to left: $\mathcal{U}(\prod_{i \in I} \pi_i, \sigma) \equiv \mathcal{U}(\{\pi_i\}, \sigma) \equiv \mathcal{U}(\{M'[\tilde{Q}]\}, \sigma)$ since I is a singleton $\{i\}$ and $\pi_i = M'[\tilde{Q}]$. Now, by definition for \mathcal{U} , $\mathcal{U}(\prod_{i \in I} \pi_i, \sigma) \equiv M'\sigma[\mathcal{U}(\tilde{Q}, \sigma)] \equiv M[\mathcal{U}(\tilde{Q}, \sigma)]$ since $M'\sigma = M$. So, $\mathcal{U}(\prod_{i \in I} \pi_i, \sigma) \equiv M[Q]$. From left to right: let us assume that I is not a singleton. For $I = \emptyset$, according to the first point, $\mathcal{U}(\prod_{i \in I} \pi_i, \sigma) \equiv \mathbf{0}$ and thus, by Proposition C.4, $\mathcal{U}(\prod_{i \in I} \pi_i, \sigma) \neq M[Q]$ for any M, Q . Now, the closure $\langle \tilde{P}; \sigma \rangle$ being normal, if I contains at least two elements then by definition of \mathcal{U} , $\mathcal{U}(\tilde{P}, \sigma) \equiv R' | R''$ for some $R, R' \neq \mathbf{0}$ by Propositions C.5 and C.4. Thus, still by Proposition C.4, $\mathcal{U}(\tilde{P}, \sigma) \neq M[Q]$ whatever M, Q are. So, I is a singleton. Now, if $\pi_i \neq M'[\tilde{Q}]$ or $M'\sigma, M$ are two different sequences, once again from the definition of \mathcal{U} and Proposition C.4, $\mathcal{U}(\tilde{P}, \sigma) \neq M[Q]$. Finally, since $\mathcal{U}(\prod_{i \in I} \pi_i, \sigma) = M[\mathcal{U}(\tilde{Q}, \sigma)]$, we have $\mathcal{U}(\tilde{Q}, \sigma) \equiv Q$.

For the third point, from right to left: we have $P' | P'' \equiv \mathcal{U}(\prod_{j \in J} \pi_j, \sigma) | \mathcal{U}(\prod_{k \in K} \pi_k, \sigma)$. By definition of \mathcal{U} , since J, K are disjoint and $J \cup K = I$, $P' | P'' \equiv \mathcal{U}(\prod_{i \in I} \pi_i, \sigma)$. From left to right: by definition, $\mathcal{U}(\prod_{i \in I} \pi_i, \sigma) = \mathcal{U}(\pi_1, \sigma) | \dots | \mathcal{U}(\pi_k, \sigma)$ where I is assumed to be $\{1, \dots, k\}$ and the π_i 's are primes. Since $\mathcal{U}(\prod_{i \in I} \pi_i, \sigma) = P' | P''$, there must exist I, J two disjoint sets of indices such that $I \cup J = 1..k$, $P' \equiv \mathcal{U}(\prod_{i \in I} \pi_i, \sigma)$ and $P'' \equiv \mathcal{U}(\prod_{j \in J} \pi_j, \sigma)$.

For the fourth point, from right to left: from the definition of \mathcal{U} , we have $\mathcal{U}(\prod_{i \in I} \pi_i, \sigma) = \mathcal{U}(\pi_i, \sigma) = \langle M'\sigma \rangle$. So, using the hypothesis, $\mathcal{U}(\prod_{i \in I} \pi_i, \sigma) \equiv \langle M \rangle$. From left to right: similar to the second point.

For the fifth point, from right to left: from the definition of \mathcal{U} , we have $\mathcal{U}(\prod_{i \in I} \pi_i, \sigma) = \mathcal{U}(\pi_i, \sigma) = (n).\mathcal{U}(\tilde{P}, \sigma)$. Using the hypothesis, $\mathcal{U}(\prod_{i \in I} \pi_i, \sigma) \equiv (n).P$. From left to right: similar to the second point. \square

C.2. Properties of the auxiliary functions

Here, we state and prove correctness properties needed in subsequent sections of the auxiliary functions *nam*, *len*, *fst*, and *split*.

First, the function *nam* is correct in the following sense.

Proposition C.6. $nam(M, \sigma) = n$ iff $M\sigma = n$.

Proof. Straightforward by induction over the length of the sequential substitution σ . \square

Second, the function len has the following property.

Proposition C.7. $len(M, \sigma) = l$ iff $M\sigma = N_1 \cdot \dots \cdot N_l$ with N_i being either a name or of the form $\text{cap } N'$ with $\text{cap} \in \{\text{in}, \text{out}, \text{open}\}$.

Proof. The proof goes by induction on the length of the sequential substitution σ .

For σ being the empty sequence $\iota: M\iota = M = N_1 \cdot \dots \cdot N_l$. By definition, $len(N_1 \cdot \dots \cdot N_l, \iota) = \sum_{i=1}^l len(N_i, \iota)$. Since each N_i is either a name n or of the form $\text{in } N'$, $\text{out } N'$ or $\text{open } N'$, we have $len(N_i, \iota) = 1$. This is equivalent to $len(N_1, \dots \cdot N_l, \iota) = l$.

For σ being the sequence $\{x \leftarrow M'\}\sigma'$ of length at least 1:

let $M = N'_1 \cdot \dots \cdot N'_k$. By induction over k :

- $k = 0$: in this case, $M = \varepsilon$ and $M\{x \leftarrow M'\}\sigma = \varepsilon$. So, $l = 0$ and by definition $len(M, \sigma) = 0$.
- $k = 1$: in this case $M = N'_1$ and we have three cases:
 - N'_1 is of the form $\text{cap } N'$ for some $\text{cap} \in \{\text{in}, \text{out}, \text{open}\}$: in this case, $M\{x \leftarrow M'\}\sigma$ is of the form $\text{cap } N''$ and by definition, $len(M, \sigma\{x \leftarrow M'\}) = 1$.
 - N'_1 is a name different from x : in this case, $M\{x \leftarrow M'\}\sigma = M\sigma$ and $len(M, \{x \leftarrow M'\}\sigma) = len(M, \sigma)$. Using the induction hypothesis, $M\sigma = N''_1 \cdot \dots \cdot N''_l$ iff $len(M, \sigma) = l$, therefore $M\{x \leftarrow M'\}\sigma = N''_1 \cdot \dots \cdot N''_l$ iff $len(M, \{x \leftarrow M'\}\sigma) = l$.
 - $N'_1 = x$: in this case, $M\{x \leftarrow M'\}\sigma = M'\sigma$ and $len(M, \{x \leftarrow M'\}\sigma) = len(M', \sigma)$. By induction hypothesis $M'\sigma = N''_1 \cdot \dots \cdot N''_{l'}$ iff $len(M', \sigma) = l'$, so $M\{x \leftarrow M'\}\sigma = N''_1 \cdot \dots \cdot N''_{l'}$ iff $len(M, \{x \leftarrow M'\}\sigma) = l'$.
- $k > 1$: using the induction hypothesis, $len(N'_1 \cdot \dots \cdot N'_{k-1}, \{x \leftarrow M'\}\sigma) = l'$ iff $N'_1 \{x \leftarrow M'\}\sigma \cdot \dots \cdot N'_{k-1} \{x \leftarrow M'\}\sigma = N''_1 \cdot \dots \cdot N''_{l'}$, and for the expression N'_k , $len(N'_k, \{x \leftarrow M'\}\sigma) = l''$ iff $N'_k \{x \leftarrow M'\}\sigma = N''_{l'+1} \cdot \dots \cdot N''_{l'+l''}$. By definition, $len(M, \{x \leftarrow M'\}\sigma)$ is the sum of $len(N'_1 \cdot \dots \cdot N'_{k-1}, \{x \leftarrow M'\}\sigma)$ and of $len(N'_k, \{x \leftarrow M'\}\sigma)$. So, we can conclude that $M\{x \leftarrow M'\}\sigma = N''_1 \cdot \dots \cdot N''_{l'+l''}$ iff $len(M, \{x \leftarrow M'\}\sigma) = l' + l''$. \square

Third, we state the correctness of fst in Proposition C.9. To prove it, we need the following lemma.

Lemma C.8. Let $\langle \tilde{P}; \{x \leftarrow N\}\sigma \rangle$ be a normal closure. Then $\langle \tilde{P}\{x \leftarrow N\}; \sigma \rangle$ is normal and $\mathcal{U}(\tilde{P}, \{x \leftarrow N\}\sigma) \equiv \mathcal{U}(\tilde{P}\{x \leftarrow N\}, \sigma)$.

Proof. For the normality of $\langle \tilde{P}\{x \leftarrow N\}; \sigma \rangle$: we can show that $\mathcal{U}(\tilde{P}\{x \leftarrow N\}, \sigma)$ is defined by induction over the structure of processes and primes. The only nontrivial case is for $\tilde{P} = M(o).\tilde{P}'$: then, $\tilde{P}\{x \leftarrow N\} = M\{x \leftarrow N\}(o).\tilde{P}'\{x \leftarrow N\}$. Since $\mathcal{U}(\tilde{P}, \{x \leftarrow N\}\sigma)$ by assumption and $\mathcal{U}(\tilde{P}'\{x \leftarrow N\}, \sigma)$ by induction hypothesis are defined and $(M\{x \leftarrow N\})\sigma = M(\{x \leftarrow N\}\sigma)$, $\mathcal{U}(\tilde{P}\{x \leftarrow N\}, \sigma)$ is defined. For the second statement, since $\langle \tilde{P}; \{x \leftarrow N\}\sigma \rangle$ is normal, x and names from N are not bound in \tilde{P} , so $bn(\tilde{P}\{x \leftarrow N\}) = bn(\tilde{P})$ and $fn(\tilde{P}\{x \leftarrow N\})$ contains $fn(\tilde{P})$ and some possibly other names that do not belong to $bn(\tilde{P})$. So, $fn(\tilde{P}\{x \leftarrow N\}) \cap bn(\tilde{P}\{x \leftarrow N\}) = \emptyset$. Moreover,

as the bound names from \tilde{P} do not occur in $\{x \leftarrow N\}\sigma$ and $bn(\tilde{P}\{x \leftarrow N\}) = bn(\tilde{P})$, $bn(\tilde{P}\{x \leftarrow N\}) \cap names(\sigma) = \emptyset$. Since x is not bound in \tilde{P} , occurrences of bound variables in \tilde{P} are not affected by the substitution $\{x \leftarrow N\}$. The requirement on offsets is trivially preserved and finally, as $\{x \leftarrow N\}\sigma$ is acyclic, σ is so.

We show that $\mathcal{U}(\tilde{P}, \{x \leftarrow N\}\sigma) \equiv \mathcal{U}(\tilde{P}\{x \leftarrow N\}, \sigma)$ by induction over the structures of processes and primes taking into account that x is not a bound variable in \tilde{P} . \square

Proposition C.9. *Let N be a capability of the form $\text{in } n$, $\text{out } n$ or $\text{open } n$. Then for all normal closures $\langle \tilde{Q}; \sigma \rangle$, there exists Q such that $\mathcal{U}(M(o).\tilde{Q}, \sigma) \equiv N.Q$ iff $fst(M, o, \sigma) = N$.*

Proof. Let us assume that $M = N_1 \cdot \dots \cdot N_l$ and that $N = \text{cap } n$ where cap ranges over in , out , open . The proof goes by induction over the offset o .

Case where $o = 0$: we have $fst(M, 0, \sigma) = \text{cap } n$. We follow by induction over the length of the sequential substitution σ .

- Case where the length of σ is 0: $\sigma = \iota$ and $fst(M, 0, \iota) = \text{cap } n$. By definition of fst , this is equivalent to $fst(N_1, 0, \iota) = \text{cap } n$ and to $N_1 = \text{cap } n$. Furthermore, as $\mathcal{U}(M(0).\tilde{Q}, \iota) = N_1 \cdot \dots \cdot N_l.\mathcal{U}(\tilde{Q}, \iota)$, this is equivalent to $\mathcal{U}(M(0).\tilde{Q}, \iota) \equiv \text{cap } n.Q$ for some Q .
- Case where σ is of the form $\{x \leftarrow M'\}\sigma'$ and the proposition holds for σ' : by definition of fst , $fst(M, 0, \sigma) = fst(N_1, 0, \sigma) = \text{cap } n$. Now, according to the value of N_1 :
 - N_1 is of the form $\text{cap } L$: so, $nam(L, \sigma) = n$ which is equivalent due to Proposition C.6, to $L\sigma = n$. As $\mathcal{U}(M(0).\tilde{Q}, \sigma) = N_1\sigma \cdot \dots \cdot N_l\sigma.\mathcal{U}(\tilde{Q}, \sigma)$, $\mathcal{U}(M(0).\tilde{Q}, \sigma) = \text{cap } n.N_2\sigma \cdot \dots \cdot N_l\sigma.\mathcal{U}(\tilde{Q}, \sigma)$. Therefore, this is equivalent to that $\mathcal{U}(M(0).\tilde{Q}, \sigma) \equiv \text{cap } n.Q$ for some Q .
 - N_1 is a name m : for each of the two cases in the definition of fst .
 - Case where $m = x$: we have $fst(N_1, 0, \sigma) = fst(m, 0, \{x \leftarrow M'\}\sigma') = fst(M', 0, \sigma') = \text{cap } n$. By induction hypothesis, it is equivalent to that for any \tilde{Q} , $\mathcal{U}(M'(0).\tilde{Q}, \sigma') \equiv \text{cap } n.Q$ for some Q . In particular for some P , $\text{cap } n.P \equiv \mathcal{U}(M'(0).N_2\{x \leftarrow M'\} \cdot \dots \cdot N_l\{x \leftarrow M'\}(0).\tilde{P}\{x \leftarrow N'\}, \sigma')$, that is $\text{cap } n.P \equiv M'\sigma'.N_2\{x \leftarrow M'\}\sigma' \cdot \dots \cdot N_l\{x \leftarrow M'\}\sigma'.$ So $\text{cap } n.P \equiv m\{x \leftarrow M'\}\sigma'.N_2\sigma' \cdot \dots \cdot N_l\sigma'.$ $\mathcal{U}(\tilde{P}, \{x \leftarrow N'\}\sigma')$ by Lemma C.8. And thus, by definition of \mathcal{U} , this is equivalent to that for some P , $\text{cap } n.P \equiv \mathcal{U}(M(0).\tilde{P}, \sigma)$.
 - Case where $m \neq x$: in this case, $fst(M, 0, \sigma) = fst(m, 0, \sigma') = \text{cap } n$. By induction hypothesis, this is equivalent to that for any \tilde{Q} , $\mathcal{U}(m(0).\tilde{Q}, \sigma') \equiv \text{cap } n.Q$ for some Q . The rest of the proof is similar to the previous case, using the fact that $m\sigma' = m\{x \leftarrow M'\}\sigma'$ since $m \neq x$.

Case where the proposition holds for any $o' < o$: we have $fst(M, o, \sigma) = \text{cap } n$. By induction over the length of the sequential substitution σ .

- Case where the length of σ is 0: $\sigma = \iota$ and $fst(M, o, \iota) = \text{cap } n$. Since $len(N_1 \cdot \dots \cdot N_o, \iota) = o$, $\text{cap } n = fst(N_{o+1} \cdot \dots \cdot N_l, 0, \iota)$. Using the base case, this latter is equivalent to that for any \tilde{P} , $\mathcal{U}(N_{o+1} \cdot \dots \cdot N_l(0).\tilde{P}, \iota) \equiv \text{cap } n.P$ for some P . Now, this is equivalent to $\text{cap } n.P \equiv N_{o+1} \cdot \dots \cdot N_l.\mathcal{U}(\tilde{P}, \iota)$ by definition of \mathcal{U} . Finally, as $M\iota = N_1 \cdot \dots \cdot N_l$, by definition of \mathcal{U} , it is equivalent to that $\text{cap } n.P \equiv \mathcal{U}(M(o).\tilde{P}, \iota)$ for some P .

- Case where σ is of the form $\{x \leftarrow M'\}\sigma'$ and the proposition holds for σ' : since $\text{fst}(M, o, \sigma)$ is defined, $o < \text{len}(M, \sigma)$. Let i be the unique integer such that $\text{len}(N_1 \cdot \dots \cdot N_{i-1}, \sigma) \leq o$ and $\text{len}(N_1 \cdot \dots \cdot N_i, \sigma) > o$ and p be $o - \text{len}(N_1 \cdot \dots \cdot N_{i-1}, \sigma)$. Then we have $\text{cap } n = \text{fst}(M, o, \sigma) = \text{fst}(N_i \cdot \dots \cdot N_i, p, \sigma)$. Now, according to the value of N_i :

- N_i is of the form $\text{cap } L$: so, $\text{nam}(L, \sigma) = n$ which is equivalent due to Proposition C.6, to $L\sigma = n$. Furthermore, since $\text{len}(N_i, \sigma) = 1$, we have $o = \text{len}(N_1 \cdot \dots \cdot N_{i-1}, \sigma)$ and thus, $p = 0$. Hence, $\text{cap } n = \text{fst}(N_i \cdot \dots \cdot N_i, 0, \sigma)$. According to the base case, this is equivalent to that for any \tilde{P} , $\mathcal{U}(N_i \cdot \dots \cdot N_i(0), \tilde{P}, \sigma) \equiv \text{cap } n.P$ for some P . Let $M\sigma$ be $N'_1 \cdot \dots \cdot N'_k$. So by definition of \mathcal{U} , $\mathcal{U}(M(o), \tilde{P}, \sigma) = N'_{o+1} \cdot \dots \cdot N'_k \cdot \mathcal{U}(\tilde{P}, \sigma)$. Now, as $o = \text{len}(N_1 \cdot \dots \cdot N_{i-1}, \sigma)$, $N_i\sigma \cdot \dots \cdot N_i\sigma = N'_{o+1} \cdot \dots \cdot N'_k$. Hence, $\mathcal{U}(M(o), \tilde{P}, \sigma) = N_i\sigma \cdot \dots \cdot N_i\sigma \cdot \mathcal{U}(\tilde{P}, \sigma)$. Equivalently, $\mathcal{U}(M(o), \tilde{P}, \sigma) = \mathcal{U}(N_i \cdot \dots \cdot N_i(0), \tilde{P}, \sigma)$ and so, $\mathcal{U}(M(o), \tilde{P}, \sigma) \equiv \text{cap } n.P$ for some P .
- N_i is a name m : in this case, we have $\text{len}(N_i, \sigma) > p$. Hence, by definition of fst , $\text{cap } n = \text{fst}(M, o, \sigma) = \text{fst}(N_i, p, \{x \leftarrow M'\}\sigma')$. For each of the two cases in the definition of fst :

Case where $m = x$: we have $\text{cap } n = \text{fst}(M', p, \sigma')$. By induction hypothesis, this is equivalent to that for any \tilde{Q} , $\mathcal{U}(M'(p), \tilde{Q}, \sigma') \equiv \text{cap } n.Q$ for some Q . As a particular case, this latter holds for $Q = P$ and for $\tilde{Q} = N_{i+1}\{x \leftarrow M'\} \cdot \dots \cdot N_i\{x \leftarrow M'\}(0). \tilde{P}\{x \leftarrow M'\}$. Now, from the definition of \mathcal{U} and using that $M' = N_i\{x \leftarrow M'\}$, this is equivalent to that $\mathcal{U}(N_i\{x \leftarrow M'\} \cdot \dots \cdot N_i\{x \leftarrow M'\}(p). \tilde{P}\{x \leftarrow M'\}, \sigma') = \text{cap } n.P$ for some P . Let $N'_1 \cdot \dots \cdot N'_k$ be $N_i\sigma$. Then, still by definition of \mathcal{U} , it is equivalent to that $N'_{p+1} \cdot \dots \cdot N'_k \cdot N_{i+1}\sigma \cdot \dots \cdot N_i\sigma \cdot \mathcal{U}(P\{x \leftarrow M'\}, \sigma') = \text{cap } n.P$. By Lemma C.8, it is equivalent to $N'_{p+1} \cdot \dots \cdot N'_k \cdot N_{i+1}\sigma \cdot \dots \cdot N_i\sigma \cdot \mathcal{U}(P, \sigma) = \text{cap } n.P$. Once again, by definition of \mathcal{U} , we have $\mathcal{U}(N_i \cdot \dots \cdot N_i(p), \tilde{P}, \sigma) = \text{cap } n.P$. Let p' be $\text{len}(N_1 \cdot \dots \cdot N_{i-1}, \sigma)$. By definition of \mathcal{U} , we have $\mathcal{U}(N_1 \cdot \dots \cdot N_{i-1}(p'). N_i \cdot \dots \cdot N_i(p). \tilde{P}, \sigma) = \text{cap } n.P$. By definition of \mathcal{U} , $\mathcal{U}(N_1 \cdot \dots \cdot N_{i-1}. N_i \cdot \dots \cdot N_i(p + p'). \tilde{P}, \sigma) = \text{cap } n.P$. Finally, as $p + p' = o$, this latter is equivalent to that $\mathcal{U}(M(o), \tilde{P}, \sigma) = \text{cap } n.P$ for some P .

Case where $m \neq x$: by definition of fst , $\text{cap } n = \text{fst}(m, p, \{x \leftarrow M'\}\sigma') = \text{fst}(m, p, \sigma')$. By induction hypothesis, this is equivalent to that for all \tilde{Q} , there exists \tilde{Q} such that $\mathcal{U}(m(p), \tilde{Q}, \sigma) \equiv \text{cap } n.\tilde{Q}$. The rest of the proof is similar to the previous case, using the fact that $m\sigma' = m\{x \leftarrow M'\}\sigma'$ since $m \neq x$. \square

Fourth, we prove that *split* is correct in the following sense.

Proposition C.10. *Let $\langle \prod_{i \in I} \pi_i, \sigma \rangle$ be a normal closure, and let L be of the form in n , out n or open n . Then $\mathcal{U}(\prod_{i \in I} \pi_i, \sigma) \equiv L.P$ iff $\exists L', o, \tilde{P}, \tilde{P}' : I$ is a singleton $\{i\}$, $\pi_i = L'(o). \tilde{P}'$, $\text{split}(\pi_i, \sigma) = (L, \tilde{P})$ and $\mathcal{U}(\tilde{P}, \sigma) \equiv P$.*

Proof. From right to left: we have $\mathcal{U}(\prod_{i \in I} \pi_i, \sigma) = \mathcal{U}(\pi_i, \sigma)$, $\pi_i = L'(o). \tilde{P}'$, $\text{split}(\pi_i, \sigma) = (L, \tilde{P})$. By Proposition C.9, $\mathcal{U}(\pi_i, \sigma) \equiv L.P$ for some P . Moreover, for $L'\sigma$ being of the form $L'_1 \cdot \dots \cdot L'_l$, $\mathcal{U}(\pi_i, \sigma) = L'_{o+1} \cdot \dots \cdot L'_l \cdot \mathcal{U}(\tilde{P}, \sigma)$ and $L'_{o+1} = L$. Note that $\mathcal{U}(\pi_i, \sigma)$ being defined, we have $o < \text{len}(L', \sigma) = l$. Now, by the definition of *split*, according to

the values of o and $len(L', \sigma)$:

- $len(L', \sigma) > o + 1$: in this case, $\tilde{P} = \{L'(o+1).\tilde{P}'\}$. So, by definition of \mathcal{U} , $\mathcal{U}(\{L'(o+1).\tilde{P}'\}, \sigma) = L'_{o+2} \cdot \dots \cdot L'_1 \cdot \mathcal{U}(\tilde{P}', \sigma)$ and thus, $\mathcal{U}(\prod_{i \in I} \pi_i, \sigma) \equiv L'_{o+1} \cdot \mathcal{U}(\{L'(o+1).\tilde{P}'\}, \sigma) \equiv L.P$ for $P \equiv \mathcal{U}(\{L'(o+1).\tilde{P}'\}, \sigma) \equiv \mathcal{U}(\tilde{P}, \sigma)$.
- $len(L', \sigma) = o + 1$: in this case, $\tilde{P} = \tilde{P}'$. Therefore, $\mathcal{U}(\{L'(o+1).\tilde{P}'\}, \sigma) = L'_1 \cdot \mathcal{U}(\tilde{P}', \sigma) = L'_{o+1} \cdot \mathcal{U}(\tilde{P}', \sigma) = L \cdot \mathcal{U}(\tilde{P}', \sigma)$. Thus, $\mathcal{U}(\prod_{i \in I} \pi_i, \sigma) \equiv L.P$ for $P \equiv \mathcal{U}(\tilde{P}', \sigma) \equiv \mathcal{U}(\tilde{P}, \sigma)$.

From left to right: let us assume that $\mathcal{U}(\prod_{i \in I} \pi_i, \sigma) \equiv L.P$. Using Proposition C.4, the set I has to be a singleton and π_i has to be of the form $L'(o).\tilde{P}'$. Now, by Proposition C.9, we know that $fst(L', o, \sigma) = L$. Thus, it is sufficient to prove that $P \equiv \mathcal{U}(\tilde{P}, \sigma)$ for $split(\pi_i, \sigma) = (L, \tilde{P})$. From the definitions of \mathcal{U} and $split$ and from Proposition C.4, it is straightforward to see that $P \not\equiv \mathcal{U}(\tilde{P}, \sigma)$ implies $\mathcal{U}(\prod_{i \in I} \pi_i, \sigma) \not\equiv L.P$. \square

C.3. Proof of Proposition 4.2

Using Lemma C.11 below, we show Proposition 4.2(1), that \downarrow^* , the reflexive and transitive closure of the sublocation relation \downarrow , preserves normality of closures.

Lemma C.11. *If $\langle \tilde{P}; \sigma \rangle$ is normal, then for any $\langle \tilde{P}'; \sigma \rangle$ such that $\langle \tilde{P}; \sigma \rangle \downarrow \langle \tilde{P}'; \sigma \rangle$, the closure $\langle \tilde{P}'; \sigma \rangle$ is normal.*

Proof. From the definition of \downarrow , we have $\tilde{P} = \tilde{Q} ++ \{M[\tilde{P}']\}$ for some \tilde{Q} , M . Thus, by the first point of Proposition C.1, the closure $\langle \{M[\tilde{P}']\}; \sigma \rangle$ is normal. Now, the names from M occur freely in $\{M[\tilde{P}']\}$. So, $\langle \{M[\tilde{P}']\}; \sigma \rangle$ being normal, none of the names from M is in $bn(\{M[\tilde{P}']\})$ and thus, in $bn(\tilde{P}')$. Therefore, by the second point of Proposition C.1, $\langle \tilde{P}'; \sigma \rangle$ is normal. \square

Restatement of Proposition 4.2(1). *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.*

Proof. A simple induction using Lemma C.11. \square

Using Lemmas C.12 and C.13 below, we show Proposition 4.2(2), that \rightarrow^* , the reflexive and transitive closure of the reduction relation \rightarrow , preserves normality of closures.

Lemma C.12. *If $\langle \{\pi\}; \sigma \rangle$ is normal and $split(\pi, \sigma) = (N, \tilde{S})$ then $\langle \tilde{S}; \sigma \rangle$ is normal.*

Proof. Since $split(\pi, \sigma) = (N, \tilde{S})$, $\pi = M(o).\tilde{S}'$ for some expression M and some annotated process \tilde{S}' . Furthermore, $\mathcal{U}(\{\pi\}, \sigma)$ being defined, $\mathcal{U}(\tilde{S}', \sigma)$ is defined. Now, according to the value of \tilde{S} : if $\tilde{S} = M(o+1).\tilde{S}'$ then, from the definition of $split$, $o+1 < len(M, \sigma)$. So, from the definition of \mathcal{U} , $\mathcal{U}(\tilde{S}', \sigma)$ being defined, $\mathcal{U}(M(o+1).\tilde{S}', \sigma) = \mathcal{U}(\tilde{S}, \sigma)$ is defined. If $\tilde{S} = \tilde{S}'$ then $\mathcal{U}(\tilde{S}, \sigma)$ is defined.

Let us first notice that $bn(\{\pi\}) = bn(\{M(o+1).\tilde{S}'\}) = bn(\tilde{S}')$ and that $fn(\{\pi\}) = fn(\{M(o+1).\tilde{S}'\}) \supseteq fn(\tilde{S}')$. Therefore, since by normality $bn(\{\pi\}) \cap (fn(\{\pi\}) \cup names(\sigma)) = \emptyset$, we have $bn(\tilde{S}) \cap (fn(\tilde{S}) \cup names(\sigma)) = \emptyset$.

The last three statements are obvious to check. \square

Lemma C.13. *If $\langle \tilde{P}; \sigma \rangle$ is normal, then for any $\langle \tilde{P}'; \sigma \rangle$ such that $\langle \tilde{P}; \sigma \rangle \rightarrow \langle \tilde{P}'; \sigma \rangle$, the closure $\langle \tilde{P}'; \sigma \rangle$ is normal, and moreover*

- either $\sigma' = \sigma$, $bn(\tilde{P}) = bn(\tilde{P}')$ and $fn(\tilde{P}') \subseteq fn(\tilde{P})$,
- or for some x, M , $\sigma' = \{x \leftarrow M\}\sigma$, $bn(\tilde{P}) = bn(\tilde{P}') \cup \{x\}$ and $fn(\tilde{P}') \subseteq fn(\tilde{P}) \cup \{x\}$.

Proof. The proof goes by induction over the structure of the context under which the reduction takes place.

If the context is empty, then the applied reduction corresponds to one of the rules (Trans In), (Trans Out), (Trans Open) and (Trans I/O). For (Trans In), (Trans Out) and (Trans Open) respectively, $\langle \{N[\tilde{Q} ++ \{\pi\}], M[\tilde{R}]\}; \sigma \rangle$, $\langle \{M[\{N[\tilde{Q} ++ \pi]\} ++ \tilde{R}]\}; \sigma \rangle$ and $\langle \{M[\tilde{P}], \pi\}; \sigma \rangle$ are normal by assumption.

Concerning the second claim of the lemma: obviously, $\sigma' = \sigma$, $bn(\tilde{P}) = bn(\tilde{P}')$. For the rules (Trans In) and (Trans Out), $fn(\tilde{P}) = fn(\tilde{P}')$ and for (Trans Open) $fn(\tilde{P}') \subseteq fn(\tilde{P})$ (the execution of open may let an ambient name disappeared).

Now for the first claim, by using Proposition C.1, $\langle \pi; \sigma \rangle$ is normal. Then, from Lemma C.12 together with the transition rules on closures, $\langle \tilde{P}; \sigma \rangle$ is normal (where $split(\pi, \sigma) = (N, \tilde{P})$ and N being respectively in m , out m and open m). Finally, using the fact that $bn(\{\pi\}) = bn(\tilde{P})$ and that $fn(\{\pi\}) \subseteq fn(\tilde{P})$ and by applying once more Proposition C.1, the closures $\langle \{M[\{N[\tilde{Q} ++ \pi]\} ++ \tilde{R}]\}; \sigma \rangle$, $\langle \{N[\tilde{Q} ++ \{\pi\}], M[\tilde{R}]\}; \sigma \rangle$ and $\langle \tilde{P} ++ \tilde{Q}; \sigma \rangle$ are normal.

For (Trans I/O), $\langle \{\langle M \rangle, (x).\tilde{P}\}; \sigma \rangle$ is normal by assumption. Let us start with the second claim of the lemma. We have $\sigma' = \{x \leftarrow M\}\sigma$. Due to the assumption of normality, x occurs at most once within an input in \tilde{P} and bound and free names are disjoint in \tilde{P} . So $bn(\{\langle M \rangle, (x).\tilde{P}\}) = bn(\tilde{P}) \cup \{x\}$ and $fn(\tilde{P}) \subseteq fn(\{\langle M \rangle, (x).\tilde{P}\}) \cup \{x\}$. Now, for the first claim, let us first prove that $\mathcal{U}(norm(\tilde{P}, \{x \leftarrow M\}\sigma), \{x \leftarrow M\}\sigma)$ is defined by induction over the structure of \tilde{P} : this is obvious for \tilde{P} being the empty multiset or the singleton $\{\langle M' \rangle\}$. For the induction step, this is also straightforward for \tilde{P} being a multiset of primes or a singleton $\{(x').\tilde{Q}\}$ or $\{M'[\tilde{Q}]\}$. Now, for $\tilde{P} = \{M'(o).\tilde{Q}\}$. By hypothesis, $\mathcal{U}(M'(o).\tilde{Q}, \sigma)$ is defined and $o = 0$. So, $0 < len(M', \sigma)$. If $len(M', \{x \leftarrow M\}\sigma) = 0$, then $norm(\tilde{P}, \{x \leftarrow M\}\sigma) = norm(\tilde{Q}, \{x \leftarrow M\}\sigma)$ and so $\mathcal{U}(norm(\tilde{P}, \{x \leftarrow M\}\sigma), \{x \leftarrow M\}\sigma)$ is defined by induction hypothesis. Otherwise, $len(M', \{x \leftarrow M\}\sigma) > 0$. So $\mathcal{U}(norm(\tilde{P}, \{x \leftarrow M\}\sigma), \{x \leftarrow M\}\sigma) = \mathcal{U}(M'(0).norm(\tilde{Q}, \{x \leftarrow M\}\sigma), \{x \leftarrow M\}\sigma)$ is defined. Since every variable occurs at most once within an input in the annotated process of a normal closure, $bn(\tilde{P}) = bn(\{(x).\tilde{P}, \langle M \rangle\}) \setminus \{x\}$; moreover, since $fn(\tilde{P}) \subseteq fn(\{\langle M \rangle, (x).\tilde{P}\}) \cup \{x\}$, $bn(\{\langle M \rangle, (x).\tilde{P}\}) \cap fn(\{\langle M \rangle, (x).\tilde{P}\}) = \emptyset$. Let us show that names from $bn(\tilde{P})$ do not occur in $\sigma' = \{x \leftarrow M\}\sigma$. As $bn(\tilde{P}) \subseteq bn(\{\langle M \rangle, (x).\tilde{P}\})$, because of the hypothesis of normality, names from $bn(\tilde{P})$ do not occur in σ . Moreover, we know that $x \notin bn(\tilde{P})$ and names occurring in M are free in $\{\langle M \rangle, (x).\tilde{P}\}$ and so, in \tilde{P} . It is straightforward that the property of uniqueness of variable within an input and the fact that offsets are equal to 0 in the scope of an input are preserved. Finally, since $\langle \{\langle M \rangle, (x).\tilde{P}\}; \sigma \rangle$ is normal, σ is acyclic and as x is bound, x does not occur in σ ; so the last point holds for $\langle \tilde{P}; \{x \leftarrow M\}\sigma \rangle$.

Now, we investigate the case where the context of reduction is nonempty, that is the rule used for reduction is either (Trans Par) or (Trans Amb). We show in this case that the second claim of the lemma holds and then that normality is preserved.

For (Trans Amb): we assume the closure $\langle M[\tilde{P}]; \sigma \rangle$ to be normal. For any \tilde{S} , we have $bn(M[\tilde{S}]) = bn(\tilde{S})$, $fn(M[\tilde{S}]) = fn(\tilde{S}) \cup fn(M[\mathbf{0}])$. Let us first consider the case where $\sigma = \sigma'$: by induction hypothesis $bn(\tilde{P}) = bn(\tilde{P}')$, $fn(\tilde{P}') \subseteq fn(\tilde{P})$. So, $bn(M[\tilde{P}]) = bn(M[\tilde{P}'])$ and $fn(M[\tilde{P}']) \subseteq fn(M[\tilde{P}])$. Now, for the case where $\sigma' = \{x \leftarrow M\}\sigma$: By induction hypothesis, $bn(\tilde{P}) = bn(\tilde{P}') \cup \{x\}$, $fn(\tilde{P}') = fn(\tilde{P}) \cup \{x\}$. So, $bn(M[\tilde{P}]) = bn(M[\tilde{P}']) \cup \{x\}$ and $fn(M[\tilde{P}']) = fn(M[\tilde{P}]) \cup \{x\}$.

Let us show now that $\langle M[\tilde{P}']; \sigma' \rangle$ is normal: since $\langle M[\tilde{P}]; \sigma \rangle$ is normal, by Proposition C.1, $\langle \tilde{P}; \sigma \rangle$ is normal. Then, since $\langle \tilde{P}; \sigma \rangle \rightarrow \langle \tilde{P}'; \sigma' \rangle$, by induction hypothesis, $\langle \tilde{P}'; \sigma' \rangle$ is normal. So, as $bn(\tilde{P}') \subseteq bn(\tilde{P})$, by Proposition C.1, $\langle M[\tilde{P}']; \sigma' \rangle$ is normal.

For (Trans Par): we assume the closure $\langle \tilde{P} ++ \tilde{Q}; \sigma \rangle$ to be normal. For any \tilde{S}, \tilde{S}' , we have $bn(\tilde{S} ++ \tilde{S}') = bn(\tilde{S}) \cup bn(\tilde{S}')$ and $fn(\tilde{S} ++ \tilde{S}') = fn(\tilde{S}) \cup fn(\tilde{S}')$. Let us first consider the case where $\sigma = \sigma'$: as by induction hypothesis $bn(\tilde{P}) = bn(\tilde{P}')$ and $fn(\tilde{P}') \subseteq fn(\tilde{P})$, we have $bn(\tilde{P} ++ \tilde{Q}) = bn(\tilde{P}' ++ \tilde{Q})$ and $fn(\tilde{P}' ++ \tilde{Q}) \subseteq fn(\tilde{P} ++ \tilde{Q})$. Now, for the case where $\sigma' = \{x \leftarrow M\}\sigma$: as by induction hypothesis $bn(\tilde{P}) = bn(\tilde{P}') \cup \{x\}$ and $fn(\tilde{P}') \subseteq fn(\tilde{P}) \cup \{x\}$, we have $bn(\tilde{P} ++ \tilde{Q}) = bn(\tilde{P}' ++ \tilde{Q}) \cup \{x\}$ and $fn(\tilde{P}' ++ \tilde{Q}) \subseteq fn(\tilde{P} ++ \tilde{Q}) \cup \{x\}$.

Let us show now that $\langle \tilde{P} ++ \tilde{Q}; \sigma' \rangle$ is normal: $\langle \tilde{P} ++ \tilde{Q}; \sigma' \rangle$ being normal, by Proposition C.1, both $\langle \tilde{P}; \sigma \rangle$ and $\langle \tilde{Q}; \sigma \rangle$ are normal. Now, since $\langle \tilde{P}; \sigma \rangle \rightarrow \langle \tilde{P}'; \sigma' \rangle$, by induction hypothesis, $\langle \tilde{P}'; \sigma' \rangle$ is normal. Let us now prove that $\langle \tilde{Q}; \sigma' \rangle$ is normal: we know that $x \in bn(\tilde{P})$; so, by normality of $\langle \tilde{P} ++ \tilde{Q}; \sigma \rangle$, x does not occur in \tilde{Q} , so $\mathcal{U}(\tilde{Q}, \sigma') \equiv \mathcal{U}(\tilde{Q}, \sigma)$ and thus, $\mathcal{U}(\tilde{Q}, \sigma')$ is defined. The other points are obviously implied by the normality of $\langle \tilde{Q}; \sigma \rangle$ and $\langle \tilde{P}'; \sigma' \rangle$. Finally, the fact that $\langle \tilde{P} ++ \tilde{Q}; \sigma \rangle$ and $\langle \tilde{Q}; \sigma' \rangle$ are normal together with Proposition C.1 implies that $\langle \tilde{P}' ++ \tilde{Q}; \sigma' \rangle$ is normal. \square

Restatement of Proposition 4.2(2). *If $\langle \tilde{P}; \sigma \rangle$ is normal and $\langle \tilde{P}; \sigma \rangle \rightarrow^* \langle \tilde{P}'; \sigma' \rangle$ then $\langle \tilde{P}'; \sigma' \rangle$ is normal.*

Proof. An induction with appeal to Lemma C.13. \square

C.4. Proof of Proposition 4.3

We prove now that the sublocation relation defined on closures simulates the sublocation relation defined on processes.

Restatement of Proposition 4.3. *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 $\mathcal{U}(\tilde{Q}, \sigma) \equiv Q$.*

Proof. For the first point, by definition for \downarrow on closures, we have $\tilde{P} = \tilde{Q} ++ \{M[\tilde{P}']\}$ for some \tilde{Q}, M, n such that $nam(M, \sigma) = n$. Therefore, by definition of \mathcal{U} , $\mathcal{U}(\tilde{P}, \sigma) = \mathcal{U}(\tilde{Q}, \sigma) \mid M\sigma[\mathcal{U}(\tilde{P}', \sigma)]$. Note that $\langle \tilde{P}; \sigma \rangle$ being normal, both $\langle \tilde{Q}; \sigma \rangle$, $\langle \tilde{P}'; \sigma \rangle$ are defined and thus, processes. Now, for the two processes $\mathcal{U}(\tilde{P}, \sigma)$, $\mathcal{U}(\tilde{P}', \sigma)$, there exists a process Q (namely $\mathcal{U}(\tilde{Q}, \sigma)$) and a name n ($n = M\sigma$ by Proposition C.6) such that $\mathcal{U}(\tilde{P}, \sigma) \equiv Q \mid n[\mathcal{U}(\tilde{P}', \sigma)]$. So, $\mathcal{U}(\tilde{P}, \sigma) \downarrow \mathcal{U}(\tilde{P}', \sigma)$.

For the second point, by definition of \downarrow on processes, $\mathcal{U}(\tilde{P}, \sigma) \downarrow P'$ iff there exists Q , n such that $\mathcal{U}(\tilde{P}, \sigma) \equiv Q \mid n[P']$. The annotated process \tilde{P} being of the form $\prod_{k \in K} \pi_k$, by Proposition 4.1, there exists I, J such that $I \cup J = K, I \cap J = \emptyset$ and $\mathcal{U}(\prod_{i \in I} \pi_i, \sigma) \equiv Q$, $\mathcal{U}(\prod_{j \in J} \pi_j, \sigma) \equiv n[P']$. From $\mathcal{U}(\prod_{j \in J} \pi_j, \sigma) \equiv n[P']$, by Proposition 4.1, there exists M', \tilde{P}' such that J is a singleton $\{j\}$, $\pi_j = M'[\tilde{P}']$, $M'\sigma = n$ and $\mathcal{U}(\tilde{P}', \sigma) \equiv P'$. Since $M'\sigma = n$, by Proposition C.6, $\text{nam}(M', \sigma) = n$. Furthermore, \tilde{P} is equal to $\prod_{i \in I} \pi_i \mid \{M'[\tilde{P}']\}$. So, $\langle \tilde{P}; \sigma \rangle \downarrow \langle \tilde{P}'; \sigma \rangle$ and $\mathcal{U}(\tilde{P}', \sigma) \equiv P'$. \square

C.5. Proof of Proposition 4.4

Given Lemmas C.14, C.15, and C.16 below, we prove Proposition 4.4, that the reduction relation defined on closures simulates the reduction relation defined on processes.

Lemma C.14. *Let $\langle \tilde{P}; \sigma \{x \leftarrow M\} \rangle$ be a normal closure such that all the offsets o occurring in P are set to 0. Then $\mathcal{U}(\tilde{P}, \sigma \{x \leftarrow M\}) \equiv \mathcal{U}(\tilde{P}, \sigma) \{x \leftarrow M\}$.*

Proof. The proof goes by induction over the structures of processes and primes. Most of the cases simply uses the definition of \mathcal{U} and the application of a substitution. We detail here the only two cases that are not straightforward.

For primes π :

- Case where $\pi = (y).\tilde{P}'$:

$$\begin{aligned} \mathcal{U}((y).\tilde{P}', \sigma) \{x \leftarrow M\} &\equiv ((y).\mathcal{U}(\tilde{P}', \sigma)) \{x \leftarrow M\} \\ &\equiv ((y) \{x \leftarrow M\}).(\mathcal{U}(\tilde{P}', \sigma) \{x \leftarrow M\}) \\ &\equiv (y).(\mathcal{U}(\tilde{P}', \sigma) \{x \leftarrow M\}) \\ &\equiv (y).(\mathcal{U}(\tilde{P}', \sigma \{x \leftarrow M\})) \\ &\equiv \mathcal{U}((y).\tilde{P}', \sigma \{x \leftarrow M\}) \end{aligned}$$

The first and last equivalences follow from the definition of \mathcal{U} ; the second one corresponds simply to the application of the substitution $\{x \leftarrow M\}$. For the third one, the closure $\langle \tilde{P}; \sigma \{x \leftarrow M\} \rangle$ being normal, by Proposition C.1, the closure $\langle \{\pi\}; \sigma \{x \leftarrow M\} \rangle$ is normal too. Therefore, as y is a bound variable and $\text{bn}(\tilde{P}) \cap \text{dom}(\sigma \{x \leftarrow M\}) = \emptyset$, x and y are different. So, $y \{x \leftarrow M\} = y$. The fourth equivalence appeals to induction hypothesis.

- Case where $\pi = M'(o).\tilde{P}'$:

$$\begin{aligned} \mathcal{U}(M'(o).\tilde{P}', \sigma) \{x \leftarrow M\} &\equiv (M'\sigma.\mathcal{U}(\tilde{P}', \sigma)) \{x \leftarrow M\} \\ &\equiv M'\sigma \{x \leftarrow M\}. \mathcal{U}(\tilde{P}', \sigma) \{x \leftarrow M\} \\ &\equiv M'\sigma \{x \leftarrow M\}. \mathcal{U}(\tilde{P}', \sigma \{x \leftarrow M\}) \\ &\equiv \mathcal{U}(M'(o).\tilde{P}', \sigma \{x \leftarrow M\}) \end{aligned}$$

The first equivalence uses the definition of \mathcal{U} and the fact that by hypothesis, o is equal to 0; the second one is simply the application of the substitution $\{x \leftarrow M\}$. The third equivalence is due to the induction hypothesis. Finally, the last equivalence is a direct consequence of the definition of \mathcal{U} and of $o = 0$. \square

Lemma C.15. *Let $\langle \tilde{P}; \{x \leftarrow M\}\sigma \rangle$ be a normal closure such that all the offsets o occurring in P are set to 0. Then $\mathcal{U}(\tilde{P}, \{x \leftarrow M\}\sigma) \equiv \mathcal{U}(\tilde{P}, \sigma)\{x \leftarrow M\sigma\}$.*

Proof. The proof goes by induction on the length of the sequential substitution of σ .

For σ being the empty substitution ι : $\mathcal{U}(\tilde{P}, \{x \leftarrow M\}\iota) \equiv \mathcal{U}(\tilde{P}, \iota\{x \leftarrow M\})$ since ι corresponds to the identity. So, by Lemma C.14, $\mathcal{U}(\tilde{P}, \{x \leftarrow M\}\iota) \equiv \mathcal{U}(\tilde{P}, \iota)\{x \leftarrow M\}$.

For σ being of the form $\sigma'\{y \leftarrow M'\}$:

$$\begin{aligned} \mathcal{U}(\tilde{P}, \{x \leftarrow M\}\sigma'\{y \leftarrow M'\}) &\equiv \mathcal{U}(\tilde{P}, \{x \leftarrow M\}\sigma')\{y \leftarrow M'\} \\ &\equiv (\mathcal{U}(\tilde{P}, \sigma')\{x \leftarrow M\sigma'\})\{y \leftarrow M'\} \end{aligned}$$

The first equivalence follows from Lemma C.14 and the second one from the induction hypothesis.

Now, the fact that $\langle \tilde{P}; \{x \leftarrow M\}\sigma'\{y \leftarrow M'\} \rangle$ is normal implies that $x \neq y$ and that x does not occur in M' . Let us consider now the process $\mathcal{U}(\tilde{P}, \sigma')\{x \leftarrow M\sigma'\}$. As $x \neq y$, the occurrences of y in $\mathcal{U}(\tilde{P}, \sigma')$ are preserved in $\mathcal{U}(\tilde{P}, \sigma')\{x \leftarrow M\sigma'\}$ and some new occurrences of y may appear in this latter, due to the possible occurrences of y in $M\sigma'$. As x does not occur in M' , we can first replace in $\mathcal{U}(\tilde{P}, \sigma')$ the occurrences of y with M' and then, replace the occurrences of x with an expression L ; this expression L is the expression $M\sigma$ in which the occurrences of y are replaced by M' . Hence

$$(\mathcal{U}(\tilde{P}, \sigma')\{x \leftarrow M\sigma'\})\{y \leftarrow M'\} \equiv (\mathcal{U}(\tilde{P}, \sigma')\{y \leftarrow M'\})\{x \leftarrow M\sigma'\{y \leftarrow M'\}\}$$

By Lemma C.14, this latter is equivalent to $\mathcal{U}(\tilde{P}, \sigma'\{y \leftarrow M'\})\{x \leftarrow M\sigma'\{y \leftarrow M'\}\}$ and so, to $\mathcal{U}(\tilde{P}, \sigma)\{x \leftarrow M\sigma\}$. \square

Lemma C.16. *Suppose $\langle \tilde{P}; \sigma \rangle$ is a normal closure such that all the offsets o occurring in P are set to 0 and x occurs neither in σ nor in $\text{bn}(\tilde{P})$. Then $\mathcal{U}(\text{norm}(\tilde{P}, \{x \leftarrow M\}\sigma), \{x \leftarrow M\}\sigma) \equiv \mathcal{U}(\tilde{P}, \sigma)\{x \leftarrow M\sigma\}$.*

Proof. First, observe that normality of $\langle \tilde{P}; \sigma \rangle$ and the assumption about x imply normality of $\langle \text{norm}(\tilde{P}, \{x \leftarrow M\}\sigma); \{x \leftarrow M\}\sigma \rangle$. Therefore, by Lemma C.15, $\mathcal{U}(\text{norm}(\tilde{P}, \{x \leftarrow M\}\sigma), \{x \leftarrow M\}\sigma) \equiv \mathcal{U}(\text{norm}(\tilde{P}, \{x \leftarrow M\}\sigma), \sigma)\{x \leftarrow M\sigma\}$. So, it is enough to prove that

$$\mathcal{U}(\text{norm}(\tilde{P}, \{x \leftarrow M\}\sigma), \sigma)\{x \leftarrow M\sigma\} \equiv \mathcal{U}(\tilde{P}, \sigma)\{x \leftarrow M\sigma\}.$$

Let us consider the two cases: $\text{len}(M, \sigma) \neq 0$ and $\text{len}(M, \sigma) = 0$. In the first case, $\text{norm}(\tilde{P}, \{x \leftarrow M\}\sigma) = \tilde{P}$ and there is nothing to prove. In the second case, normality of $\langle \tilde{P}; \sigma \rangle$ implies that $\text{norm}(\tilde{P}, \{x \leftarrow M\}\sigma)$ differs from \tilde{P} only by some occurrences of $x(0)$. The equivalence $\mathcal{U}(\text{norm}(\tilde{P}, \{x \leftarrow M\}\sigma), \sigma)\{x \leftarrow M\sigma\} \equiv \mathcal{U}(\tilde{P}, \sigma)\{x \leftarrow M\sigma\}$ follows then by induction on the structure of $M\sigma$ using the congruence rule (Struct ε). \square

Restatement of Proposition 4.4. 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 $\mathcal{U}(\tilde{P}', \sigma') \equiv P'$.

Proof. The proof goes by induction over the structure of the context under which the reduction takes place.

If the context is empty, then for the first point, the reduction applied corresponds to one of the rules (Trans In), (Trans Out), (Trans Open) and (Trans I/O).

For the first point and the rule (Trans In):

$$\begin{aligned} \mathcal{U}(\{N[\tilde{Q}++\{\pi\}], M[\tilde{R}]\}, \sigma) &\equiv N\sigma[\mathcal{U}(\tilde{Q}, \sigma) \mid \mathcal{U}(\{\pi\}, \sigma)] \mid M\sigma[\mathcal{U}(\tilde{R}, \sigma)] \\ &\equiv n[\mathcal{U}(\tilde{Q}, \sigma) \mid \mathcal{U}(\{\pi\}, \sigma)] \mid m[\mathcal{U}(\tilde{R}, \sigma)] \\ &\equiv n[\mathcal{U}(\tilde{Q}, \sigma) \mid \text{in } m.\mathcal{U}(\tilde{P}, \sigma)] \mid m[\mathcal{U}(\tilde{R}, \sigma)] \end{aligned}$$

The first equivalence follows from the definition of \mathcal{U} . The second one is a consequence of the conditions of the rule (Trans In) and of Proposition C.6. The third equivalence follows from the conditions of the rule (Trans In) and from Proposition C.10.

On the other hand

$$\begin{aligned} \mathcal{U}(M[N[\tilde{Q}++\tilde{P}]]+\tilde{R}), \sigma) &\equiv M\sigma[N\sigma[\mathcal{U}(\tilde{Q}, \sigma) \mid \mathcal{U}(\tilde{P}, \sigma)] \mid \mathcal{U}(\tilde{R}, \sigma)] \\ &\equiv m[n[\mathcal{U}(\tilde{Q}, \sigma) \mid \mathcal{U}(\tilde{P}, \sigma)] \mid \mathcal{U}(\tilde{R}, \sigma)] \end{aligned}$$

The first equivalence follows from the definition of \mathcal{U} and the second one from the conditions of the rule (Trans In) and from Proposition C.6. Therefore, $\mathcal{U}(N[\tilde{Q}++\{\pi\}]+M[\tilde{R}], \sigma) \rightarrow \mathcal{U}(M[N[\tilde{Q}++\tilde{P}]]+\tilde{R}), \sigma)$.

The proof is similar for the rules (Trans Out) and (Trans Open). Now, for the first point and the rule (Trans I/O): by the definition of \mathcal{U} , we have $\mathcal{U}(\{\langle M \rangle, (x).\tilde{P}\}, \sigma) \equiv \langle M\sigma \mid (x).\mathcal{U}(\tilde{P}, \sigma) \rangle$. Let \tilde{P}' be $\text{norm}(\tilde{P}, \{x \leftarrow M\}\sigma)$. By Lemma C.15, the closure $\langle \{\langle M \rangle, (x).\tilde{P}\}; \sigma \rangle$ being normal, $\mathcal{U}(\tilde{P}', \{x \leftarrow M\}\sigma) \equiv \mathcal{U}(\tilde{P}', \sigma)\{x \leftarrow M\sigma\}$. Therefore, $\mathcal{U}(\{\langle M \rangle, (x).\tilde{P}\}, \sigma) \rightarrow \mathcal{U}(\tilde{P}', \{x \leftarrow M\}\sigma)$.

Let us consider now the second point with the assumption that the context is empty, that is the reduction is made by (Red In), (Red Out), (Red Open) or (Red I/O).

For the second point and the rule (Red In): let us assume that $\mathcal{U}(\tilde{S}, \sigma) \rightarrow S'$ by the rule (Red In). Therefore, $S' \equiv m[n[Q|P]|R]$ for some m, n, P, Q, R and $\mathcal{U}(\tilde{S}, \sigma) \equiv n[Q \mid \text{in } m.P] \mid m[R]$. So, by Proposition 4.1 and Proposition C.10, there exists $N, M, L', \tilde{P}, \tilde{P}', \tilde{Q}, \tilde{R}$ such that $\tilde{S} = \{N[\tilde{Q}++\{L'(o).\tilde{P}'\}], M[\tilde{R}]\}, N\sigma = n, M\sigma = m, \mathcal{U}(\tilde{Q}, \sigma) \equiv Q, \mathcal{U}(\tilde{R}, \sigma) \equiv R, \text{split}(L'(o).\tilde{P}') = (\text{in } m, \tilde{P})$ and $\mathcal{U}(\tilde{P}, \sigma) \equiv P$. Using Proposition C.6, we have $\text{nam}(M, \sigma) = m$ and $\text{nam}(N, \sigma) = n$. So, by definition for (Red In),

$$\langle \tilde{S}; \sigma \rangle \rightarrow \langle \{M[\{N[\tilde{P}++\tilde{Q}]]+\tilde{R}\}]; \sigma \rangle$$

and furthermore,

$$\begin{aligned} \mathcal{U}(M[N[\tilde{Q}++\tilde{P}]]+\tilde{R}), \sigma) &\equiv m[n[\mathcal{U}(\tilde{Q}, \sigma) \mid \mathcal{U}(\tilde{P}, \sigma)] \mid \mathcal{U}(\tilde{R}, \sigma)] \\ &\equiv m[n[Q|P]|R] \equiv S' \end{aligned}$$

The proof is similar for the rules (Red Out) and (Red Open). Now, for the second point and the rule (Red I/O): let us assume that $\mathcal{U}(\tilde{S}, \sigma) \rightarrow S'$ by the rule (Red I/O). Therefore, $S' \equiv P\{x \leftarrow M\}$ and $\mathcal{U}(\tilde{S}, \sigma) \equiv (x).P \mid \langle M \rangle$. So, by Proposition 4.1, there exists M', \tilde{P} such that $\tilde{S} = \{\langle M' \rangle, (x).\tilde{P}\}$, $M'\sigma = M$ and $\mathcal{U}(\tilde{P}, \sigma) \equiv P$. Therefore, $\langle \tilde{S}; \sigma \rangle \rightarrow \langle \tilde{P}' ; \{x \leftarrow M'\}\sigma \rangle$ where $\tilde{P}' = \text{norm}(\tilde{P}, \{x \leftarrow M'\}\sigma)$. Furthermore, $\langle \{\langle M' \rangle, (x).\tilde{P}\}; \sigma \rangle$ being normal, by Lemma C.16

$$\begin{aligned} \mathcal{U}(\tilde{P}', \{x \leftarrow M'\}\sigma) &\equiv \mathcal{U}(\tilde{P}, \sigma)\{x \leftarrow M'\sigma\} \\ &\equiv P\{x \leftarrow M\}. \end{aligned}$$

Now, we investigate the case where the context of reduction is nonempty: for the first point, the rule used for reduction is either (Trans Par) or (Trans Amb).

For the rule (Trans Amb): if $\langle \tilde{P}; \sigma \rangle \rightarrow \langle \tilde{P}'; \sigma' \rangle$ then $\langle M[\tilde{P}]; \sigma \rangle \rightarrow \langle M[\tilde{P}']; \sigma' \rangle$. In this case, $\mathcal{U}(M[\tilde{P}], \sigma) = M\sigma[\mathcal{U}(\tilde{P}, \sigma)]$ and $\mathcal{U}(M[\tilde{P}'], \sigma') = M\sigma'[\mathcal{U}(\tilde{P}', \sigma')]$. By C.13, either $\sigma' = \sigma$ or $\sigma' = \{x \leftarrow L\}\sigma$. In this last case, x is bound in \tilde{P} and thus, by normality, x does not occur in M . So in both cases, $M\sigma' = M\sigma$. Moreover, by the rule (Red Amb), $M\sigma[\mathcal{U}(\tilde{P}, \sigma)] \rightarrow M\sigma[\mathcal{U}(\tilde{P}', \sigma')]$. So, $\mathcal{U}(M[\tilde{P}], \sigma) \rightarrow \mathcal{U}(M[\tilde{P}'], \sigma')$.

For the rule (Trans Par): if $\langle \tilde{P}; \sigma \rangle \rightarrow \langle \tilde{P}'; \sigma' \rangle$ then $\langle \tilde{P} \text{ ++ } \tilde{Q}; \sigma \rangle \rightarrow \langle \tilde{P}' \text{ ++ } \tilde{Q}; \sigma' \rangle$. In this case, $\mathcal{U}(\tilde{P} \text{ ++ } \tilde{Q}; \sigma) \equiv \mathcal{U}(\tilde{P}, \sigma) \mid \mathcal{U}(\tilde{Q}, \sigma)$ and $\mathcal{U}(\tilde{P}' \text{ ++ } \tilde{Q}; \sigma') \equiv \mathcal{U}(\tilde{P}', \sigma') \mid \mathcal{U}(\tilde{Q}, \sigma')$. By C.13, either $\sigma' = \sigma$ or $\sigma' = \{x \leftarrow M\}\sigma$. In this last case, x is bound in \tilde{P} and thus, by normality does not occur in \tilde{Q} . So, in both cases, we have $\mathcal{U}(\tilde{Q}, \sigma') \equiv \mathcal{U}(\tilde{Q}, \sigma)$. Moreover, by the rule (Red Par), $\mathcal{U}(\tilde{P}, \sigma) \mid \mathcal{U}(\tilde{Q}, \sigma) \rightarrow \mathcal{U}(\tilde{P}', \sigma') \mid \mathcal{U}(\tilde{Q}, \sigma)$. So, $\mathcal{U}(\tilde{P} \text{ ++ } \tilde{Q}, \sigma) \rightarrow \mathcal{U}(\tilde{P}' \text{ ++ } \tilde{Q}, \sigma)$.

For the second point, the rule used for reduction is either (Red Par) or (Red Amb).

For (Red Amb): let us assume that $\mathcal{U}(\tilde{S}, \sigma) \rightarrow S'$ by (Red Amb). We have $S' = n[P']$ and $\mathcal{U}(\tilde{S}, \sigma) \equiv n[P]$. So, by Proposition 4.1, there exists N, σ such that \tilde{S} is a singleton $\{\pi\}$, $\pi = N[\tilde{P}]$, $N\sigma = n$ and $\mathcal{U}(\tilde{P}, \sigma) \equiv P$. By hypothesis $P \rightarrow P'$, so $\mathcal{U}(\tilde{P}, \sigma) \rightarrow P'$. By induction hypothesis, there exists \tilde{P}', σ' such that $\langle \tilde{P}; \sigma \rangle \rightarrow \langle \tilde{P}'; \sigma' \rangle$ and $\mathcal{U}(\tilde{P}', \sigma') \equiv P'$. Then by the rule (Trans Amb), $\langle \{N[\tilde{P}]\}; \sigma \rangle \rightarrow \langle \{N[\tilde{P}']\}; \sigma' \rangle$; so, $\langle \tilde{S}; \sigma \rangle \rightarrow \langle \{N[\tilde{P}']\}; \sigma' \rangle$. Finally, $\mathcal{U}(\{N[\tilde{P}']\}, \sigma') \equiv N\sigma'[\mathcal{U}(\tilde{P}', \sigma')]$. By Lemma C.13, either $\sigma = \sigma'$ or $\sigma' = \{x \leftarrow M\}\sigma$ with x a bound variable in \tilde{P} . By normality x does not belong to N , so $N\sigma' = N\sigma = n$. Therefore, $N\sigma'[\mathcal{U}(\tilde{P}', \sigma')] \equiv n[\mathcal{U}(\tilde{P}', \sigma')] \equiv n[P'] \equiv S'$.

For (Red Par): let us assume that $\mathcal{U}(\tilde{S}, \sigma) \rightarrow S'$ by (Red Par). We have $S' = P' \mid Q$ and $\mathcal{U}(\tilde{S}, \sigma) \equiv P \mid Q$. So, by Proposition 4.1, there exists \tilde{P}, \tilde{Q} such that $\tilde{S} = \tilde{P} \text{ ++ } \tilde{Q}$, $\mathcal{U}(\tilde{P}, \sigma) \equiv P$ and $\mathcal{U}(\tilde{Q}, \sigma) \equiv Q$. By hypothesis, $P \rightarrow P'$, so $\mathcal{U}(\tilde{P}, \sigma) \rightarrow P'$. By induction hypothesis, there exists \tilde{P}', σ' such that $\langle \tilde{P}; \sigma \rangle \rightarrow \langle \tilde{P}'; \sigma' \rangle$ and $\mathcal{U}(\tilde{P}', \sigma') \equiv P'$. Then by the rule (Trans Par), $\langle \tilde{P} \text{ ++ } \tilde{Q}; \sigma \rangle \rightarrow \langle \tilde{P}' \text{ ++ } \tilde{Q}; \sigma' \rangle$; so, $\langle \tilde{S}; \sigma \rangle \rightarrow \langle \tilde{P}' \text{ ++ } \tilde{Q}; \sigma' \rangle$. Finally, $\mathcal{U}(\tilde{P}' \text{ ++ } \tilde{Q}; \sigma') \equiv \mathcal{U}(\tilde{P}', \sigma') \mid \mathcal{U}(\tilde{Q}, \sigma')$. Now, by Lemma C.13, either $\sigma = \sigma'$ or $\sigma' = \{x \leftarrow M\}\sigma$ with x a bound variable in \tilde{P} . By normality x does not occur in \tilde{Q} ; so, $\mathcal{U}(\tilde{Q}, \sigma') \equiv \mathcal{U}(\tilde{Q}, \sigma)$. Therefore, $\mathcal{U}(\tilde{P}' \text{ ++ } \tilde{Q}; \sigma') \equiv P' \mid Q \equiv S'$. \square

C.6. Proof of Proposition 4.9

Restatement of Proposition 4.9. *The model checking algorithm described in Section 4.4 preserves the normality of $\text{Check}(\tilde{P}, \sigma, \mathcal{A})$.*

Proof. By case inspection of the algorithm, we show that if $Check(\tilde{P}, \sigma, \mathcal{A})$ is normal in the left-hand side of equality then any expression $Check(\tilde{P}', \sigma', \mathcal{A}')$ occurring in the right-hand side is also normal.

- For the Boolean connectives \neg, \vee : since in any case, $\tilde{P}' = \tilde{P}$ and $\sigma = \sigma'$ and \mathcal{A}' is a closed formula such that $fn(\mathcal{A}') \subseteq fn(\mathcal{A})$, this is straightforward.
- For location $\mathcal{A} = n[\mathcal{A}']$: in this case, $\tilde{P} = \{n[\tilde{Q}]\}$ and $\sigma = \sigma'$. By Proposition C.1 the closure $\langle \tilde{Q}; \sigma \rangle$ is normal. The remaining conditions are fulfilled since $bn(P') = bn(P)$, $\sigma' = \sigma$ and for the closed formula \mathcal{A}' , $fn(\mathcal{A}') \subseteq fn(\mathcal{A})$.
- For composition $\mathcal{A} = \mathcal{A}' | \mathcal{A}''$: this proof is similar to the previous case.
- For existential quantification $\exists x.\mathcal{A}$: in this case, $\tilde{P}' = \tilde{P}$ and $\sigma = \sigma'$ and the fact that $\mathcal{A}\{x \leftarrow m_i\}$ is closed is straightforward. So, it is sufficient to show that whatever the ambient name m_i is, $fn(\mathcal{A}\{x \leftarrow m_i\}) \cap (bn(\tilde{P}) \cup dom(\sigma)) = \emptyset$. By noticing that $fn(\mathcal{A}\{x \leftarrow m_i\})$ is either equal to $fn(\exists x.\mathcal{A})$ or to $fn(\exists x.\mathcal{A}) \cup \{m_i\}$ and using the normality for $Check(\tilde{P}, \sigma, \exists x.\mathcal{A})$, this amounts to prove that $m_i \notin bn(\tilde{P}) \cup dom(\sigma)$. According to the value of m_i :
 - For $m_i = m_0$: straightforward.
 - $m_i \in fn(\tilde{P}, \sigma) \cup fn(\mathcal{A})$: let us assume that $m_i \in fn(\mathcal{A})$. Then, $m_i \in fn(\exists x.\mathcal{A})$. So, by normality of $Check(\tilde{P}, \sigma, \exists x.\mathcal{A})$, $m_i \notin bn(\tilde{P}) \cup dom(\sigma)$. Let us assume now that $m_i \in fn(\tilde{P}, \sigma)$: by definition, $m_i \notin dom(\sigma)$. Now, by normality of $\langle \tilde{P}; \sigma \rangle$, since $m_i \in fn(\tilde{P})$ or $m_i \in names(\sigma)$, $m_i \notin bn(\tilde{P})$.
- For sometime $\diamond \mathcal{A}$: we have to prove that for any closure $\langle \tilde{P}'; \sigma' \rangle$ such that $\langle \tilde{P}; \sigma \rangle \rightarrow^* \langle \tilde{P}'; \sigma' \rangle$, $Check(\tilde{P}', \sigma', \mathcal{A})$ is normal. This follows directly from Proposition 4.2(2) and Lemma C.13 by induction on the length of the derivation.
- For somewhere $\diamond \mathcal{A}$: we have to prove that for any closure $\langle \tilde{P}'; \sigma' \rangle$ such that $\langle \tilde{P}; \sigma \rangle \downarrow^* \langle \tilde{P}'; \sigma' \rangle$, $Check(\tilde{P}', \sigma', \mathcal{A})$ is normal. This follows directly by induction on the length of the derivation using Proposition 4.2(1) and the fact that $\sigma' = \sigma$ and $fn(\tilde{P}') \subseteq fn(\tilde{P})$.
- For placement $\mathcal{A}@n$: from the hypothesis of normality for $Check(\tilde{P}, \sigma, \mathcal{A}@n)$, since $n \in fn(\mathcal{A})$, $n \notin bn(\tilde{P})$. Therefore, by Proposition C.1, $\langle n[P]; \sigma \rangle$ is normal. Moreover, \mathcal{A} is a closed formula. Finally, by hypothesis, $fn(\mathcal{A}@n) \cap (bn(\tilde{P}) \cup dom(\sigma)) = \emptyset$, and $bn(\tilde{P}) = bn(n[\tilde{P}])$, $fn(\mathcal{A}) \subseteq fn(\mathcal{A}@n)$. So, $fn(\mathcal{A}) \cap (bn(n[\tilde{P}]) \cup dom(\sigma)) = \emptyset$. \square

C.7. Proof of Proposition 4.10

The correctness of our algorithm, Proposition 4.10, is a corollary of Lemma C.18 below, which itself depends on the following fact.

Lemma C.17 (Cardelli and Gordon [9]). *For any ambient process P and any ambient formula \mathcal{A} , let $\{m_1, \dots, m_k\} = fn(P) \cup fn(\mathcal{A})$ and suppose $m_0 \notin \{m_1, \dots, m_k\}$. Then $P \models \exists x.\mathcal{A}$ iff $P \models \mathcal{A}\{x \leftarrow m_i\}$ for some i in $0 \dots k$.*

Lemma C.18. *For any normal closure $\langle \tilde{P}; \sigma \rangle$, $\mathcal{U}(\tilde{P}, \sigma) \models \mathcal{A}$ if and only if $Check(\tilde{P}, \sigma, \mathcal{A}) = \mathbf{T}$.*

Proof. The proof is by induction on the structure of the formula \mathcal{A} .

- The base case $\mathcal{A} = \mathbf{T}$ is trivial. The other base case $\mathcal{A} = \mathbf{0}$ is a consequence of Proposition 4.1.
- For Boolean connectives \neg, \wedge , this is obvious from the induction hypothesis and the algorithm.
- For location $\mathcal{A} = n[\mathcal{A}']$: according to the algorithm, we have $Check(\prod_{i \in 1 \dots k} \pi_i, \sigma, n[\mathcal{A}']) = \mathbf{T}$ iff there exists \tilde{Q} and M with $k = 1, \pi_1 = M[\tilde{Q}], nam(M, \sigma) = n$ and $Check(\tilde{Q}, \sigma, \mathcal{A}') = \mathbf{T}$. Then, by Proposition 4.1, $\mathcal{U}(\prod_{i \in 1 \dots k} \pi_i, \sigma) \equiv n[\mathcal{U}(\tilde{Q}, \sigma)]$. By induction hypothesis, $Check(\tilde{Q}, \sigma, \mathcal{A}') = \mathbf{T}$ is equivalent to $\mathcal{U}(\tilde{Q}, \sigma) \models \mathcal{A}'$. So, it is equivalent to $\mathcal{U}(\prod_{i \in 1 \dots k} \pi_i, \sigma) \models n[\mathcal{A}']$.
- For composition $\mathcal{A} = \mathcal{A}' | \mathcal{A}''$: according to the algorithm, we have that $Check(\prod_{i \in 1 \dots k} \pi_i, \sigma, \mathcal{A}' | \mathcal{A}'') = \mathbf{T}$ iff there exists I, J such that $I \cup J = 1 \dots k, I \cap J = \emptyset, Check(\prod_{i \in I} \pi_i, \sigma, \mathcal{A}') = \mathbf{T}$ and $Check(\prod_{j \in J} \pi_j, \sigma, \mathcal{A}'') = \mathbf{T}$. Now, using the induction hypothesis, $Check(\prod_{i \in I} \pi_i, \sigma, \mathcal{A}') = \mathbf{T}$ and $Check(\prod_{j \in J} \pi_j, \sigma, \mathcal{A}'') = \mathbf{T}$ are equivalent respectively to $\mathcal{U}(\prod_{i \in I} \pi_i, \sigma) \models \mathcal{A}'$ and to $\mathcal{U}(\prod_{j \in J} \pi_j, \sigma) \models \mathcal{A}''$. Finally, by Proposition 4.1, it is equivalent to $\mathcal{U}(\prod_{i \in 1 \dots k} \pi_i, \sigma) \models \mathcal{A}' | \mathcal{A}''$.
- For existential quantification $\exists x. \mathcal{A}$: let us assume $Check(\tilde{P}, \sigma, \exists x. \mathcal{A}) = \mathbf{T}$. Let $\{m_1, \dots, m_k\} = fn(\tilde{P}, \sigma) \cup fn(\mathcal{A})$ and m_0 , an ambient name such that $m_0 \notin \{m_1, \dots, m_k\} \cup bn(\tilde{P}) \cup dom(\sigma)$. From the algorithm, this implies that there exists i such that $Check(\tilde{P}, \sigma, \mathcal{A}\{x \leftarrow m_i\}) = \mathbf{T}$. So, by the induction hypothesis, $\mathcal{U}(\tilde{P}, \sigma) \models \mathcal{A}\{x \leftarrow m_i\}$. Now, according to the value of m_i :
 - $m_i \in \{m_1, \dots, m_k\} \cap (fn(\mathcal{A}) \cup fn(\mathcal{U}(\tilde{P}, \sigma)))$: by Lemma C.17, we have $\mathcal{U}(\tilde{P}, \sigma) \models \exists x. \mathcal{A}$.
 - $m_i \in \{m_1, \dots, m_k\}$ and $m_i \notin (fn(\mathcal{A}) \cup fn(\mathcal{U}(\tilde{P}, \sigma)))$: by Lemma C.17, we have $\mathcal{U}(\tilde{P}, \sigma) \models \exists x. \mathcal{A}$.
 - $m_i \notin \{m_1, \dots, m_k\}$: it is obvious then that $m_i \notin fn(\mathcal{A}) \cup fn(\mathcal{U}(\tilde{P}, \sigma))$. So, by Lemma C.17, we have $\mathcal{U}(\tilde{P}, \sigma) \models \exists x. \mathcal{A}$.
 Conversely, let us assume that $\mathcal{U}(\tilde{P}, \sigma) \models \exists x. \mathcal{A}$. From Lemma C.17, this is equivalent to that for $\{m_1, \dots, m_k\} = fn(\mathcal{U}(\tilde{P}, \sigma)) \cup fn(\mathcal{A})$ and for any arbitrary m_0 such that $m_0 \notin \{m_1, \dots, m_k\}$, there exists i such that $\mathcal{U}(\tilde{P}, \sigma) \models \mathcal{A}\{x \leftarrow m_i\}$. This latter is equivalent to that $Check(\tilde{P}, \sigma, \mathcal{A}\{x \leftarrow m_i\}) = \mathbf{T}$ by induction hypothesis. Now according to the value of m_i :
 - $m_i \in fn(\mathcal{U}(\tilde{P}, \sigma)) \cup fn(\mathcal{A})$: in this case $m_i \in fn(\tilde{P}, \sigma) \cup fn(\mathcal{A})$. So, by the algorithm, $Check(\tilde{P}, \sigma, \exists x. \mathcal{A}) = \mathbf{T}$.
 - $m_i \notin fn(\mathcal{U}(\tilde{P}, \sigma)) \cup fn(\mathcal{A})$ and $m_i \in fn(\tilde{P}, \sigma) \cup fn(\mathcal{A})$: once again, by the algorithm, $Check(\tilde{P}, \sigma, \exists x. \mathcal{A}) = \mathbf{T}$.
 - $m_i \notin fn(\tilde{P}, \sigma) \cup fn(\mathcal{A})$: so, $m_i = m_0$. Since m_0 can be chosen arbitrarily, one can assume moreover that $m_i \notin bn(\tilde{P}) \cup dom(\sigma)$. So, by the algorithm, $Check(\tilde{P}, \sigma, \exists x. \mathcal{A}) = \mathbf{T}$.
- For sometime $\diamond \mathcal{A}$: $\mathcal{U}(\tilde{P}, \sigma) \models \diamond \mathcal{A}$ is by definition equivalent to the fact that there exists P' such that $\mathcal{U}(\tilde{P}, \sigma) \rightarrow^* P'$ and $P' \models \mathcal{A}$.

By Proposition 4.4, this latter is equivalent to that there exists \tilde{P}', σ' such that $\mathcal{U}(\tilde{P}, \sigma) \rightarrow^* \mathcal{U}(\tilde{P}', \sigma')$ and $\mathcal{U}(\tilde{P}', \sigma') \equiv P'$ and thus, $\mathcal{U}(\tilde{P}', \sigma') \models \mathcal{A}$. Therefore, by induction hypothesis, $Check(\tilde{P}', \sigma', \mathcal{A}) = \mathbf{T}$.

- For somewhere $\diamond \mathcal{A}$: the proof is similar to the previous case using Proposition 4.3 instead of Proposition 4.4.
- For placement $\mathcal{A}@n$: by definition, $\mathcal{U}(\tilde{P}, \sigma) \models \mathcal{A}@n$ iff $n[\mathcal{U}(\tilde{P}, \sigma)] \models \mathcal{A}$. By assumption n does not belong to $\text{dom}(\sigma)$. So, from the definition for \mathcal{U} , $n[\mathcal{U}(\tilde{P}, \sigma)] = \mathcal{U}(n[\tilde{P}], \sigma)$. Hence, $n[\mathcal{U}(\tilde{P}, \sigma)] \models \mathcal{A}$ is equivalent to that $\mathcal{U}(n[\tilde{P}], \sigma) \models \mathcal{A}$. Using the induction hypothesis, this latter is equivalent to $\text{Check}(n[\tilde{P}], \sigma, \mathcal{A}) = \mathbf{T}$, and thus to $\text{Check}(\tilde{P}, \sigma, \mathcal{A}@n) = \mathbf{T}$. \square

Restatement of Proposition 4.10. *For all processes P and closed formulas \mathcal{A} , we have $P \models \mathcal{A}$ if and only if $\text{Check}(\mathcal{F}(P), \iota, \mathcal{A}) = \mathbf{T}$.*

Proof. As the closure $\langle \mathcal{F}(P); \iota \rangle$ is normal, this follows from Lemma C.18. \square

Appendix D. Hardness proofs

This appendix contains proofs of results stated in Section 5.

D.1. Proof of Lemma 5.1

Lemma 5.1 is the crux of correctness for the encoding from Section 5.1 of QBF satisfaction in the full calculus and logic.

Restatement of Lemma 5.1. *Consider a closed quantified Boolean formula φ and its encoding $\llbracket \varphi \rrbracket$ in the ambient logic. The formula φ is valid if and only if the model checking problem $\mathbf{0} \models \llbracket \varphi \rrbracket$ holds.*

Proof. Let us denote $C_1 \wedge \dots \wedge C_k$ by ψ . We consider a closed QBF formula $Q_1 v_1 \dots Q_n v_n \psi$. We are going to show that for any $0 \leq m \leq n$, denoting φ' the formula $Q_{m+1} v_{m+1} \dots Q_n v_n \psi$,

$$v_1 \mapsto t_1, \dots, v_m \mapsto t_m \models \varphi' \quad \text{iff} \quad \mathbf{0} \models \llbracket \varphi' \rrbracket \{v_1 \leftarrow t_1, \dots, v_m \leftarrow t_m\}$$

Note that this statement obviously implies Lemma 5.1.

The proof of this statement goes by induction on the number l of variables that are quantified in φ' .

For the base case $l=0$: $v_1 \mapsto t_1, \dots, v_n \mapsto t_n \models \psi$ iff for each C_i , there exists ℓ_j in C_i such that $t_j = tt$ iff $\ell_j = v_j$ and $t_j = ff$ iff $\ell_j = \bar{v}_j$. This is equivalent to saying that for each C_i , there exists ℓ_j in C_i such that $\mathbf{0} \models \llbracket C_i \rrbracket \{v_1 \leftarrow t_1, \dots, v_n \leftarrow t_n\}$, which is equivalent to $\mathbf{0} \models \psi \{v_1 \leftarrow t_1, \dots, v_n \leftarrow t_n\}$.

For the induction step $0 < l \leq n$: let us denote \mathcal{M} the interpretation $v_1 \mapsto t_1, \dots, v_{n-l} \mapsto t_{n-l}$, σ the corresponding substitution $\{v_1 \leftarrow t_1, \dots, v_{n-l} \leftarrow t_{n-l}\}$ and φ' the formula $Q_{n-l+2} v_{n-l+2} \dots Q_n v_n \psi$. Assuming that the statement holds for $l-1$, let us consider $\mathcal{M} \models Q_{n-l+1} v_{n-l+1} \varphi'$.

By case distinction over Q_{n-l+1} :

Case where $Q_{n-l+1} = \exists$: in this case, either $\mathcal{M}, v_{n-l+1} \mapsto tt \models \varphi'$ or $\mathcal{M}, v_{n-l+1} \mapsto ff \models \varphi'$. By induction hypothesis, this is equivalent to that either $\mathbf{0} \models \llbracket \varphi' \rrbracket \sigma \{v_{n-l+1} \leftarrow tt\}$ or $\mathbf{0} \models \llbracket \varphi' \rrbracket \sigma \{v_{n-l+1} \leftarrow ff\}$. This latter is equivalent to $\mathbf{0} \models \exists v_{n-l+1} \in \{tt, ff\} \llbracket \varphi' \rrbracket \sigma$ which is equivalent by definition of the encoding to $\mathbf{0} \models \llbracket Q_{n-l+1} v_{n-l+1} \varphi' \rrbracket \sigma$.

Case where $Q_{n-l+1} = \forall$: this case is similar to the previous one. \square

D.2. Proof of Lemma 5.3

Lemma 5.3 is the crux of correctness for the encoding from Section 5.2 of QBF satisfaction in the calculus of mobile ambients without I/O.

To prove Lemma 5.3, let us first fix some notations and prove some auxiliary lemmas.

For a given closed QBF formula $\varphi = Q_1 v_1 \dots Q_n v_n \psi$ in prenex and conjunctive normal form, we denote ψ by $C_1 \wedge \dots \wedge C_k$ and define for all $0 \leq i \leq n$

$$V_i \triangleq v_i[pos[]] \mid v_i[neg[]]$$

$$V_i^{tt} \triangleq v_i[pos[]] \mid v'_i[] \mid v_i[neg[]]$$

$$V_i^{ff} \triangleq v_i[pos[]] \mid v_i[neg[]] \mid v'_i[]$$

For all $0 \leq m \leq n$, \mathcal{M} being equal to $v_1 \mapsto t_1, \dots, v_m \mapsto t_m$,

$$\varphi_m \triangleq Q_{m+1} v_{m+1} \dots Q_n v_n \psi$$

$$P_{\mathcal{M}} \triangleq V_1^{t_1} \mid \dots \mid V_m^{t_m} \mid V_{m+1} \mid \dots \mid V_n \mid P^{\varphi_m}$$

assuming that $\llbracket \varphi_m \rrbracket = (P^{\varphi_m}, \mathcal{A}^{\varphi_m})$.

It should be noticed that due to the definition of $\llbracket \cdot \rrbracket$, for all $0 \leq m < n$, $P^{\varphi_m} \models v'_{m+1}[\mathbf{T}]$ and $P^{\varphi_n} \models \text{end}[\mathbf{T}]$.

Lemma D.1. For all $0 \leq m < n$, $P_{\mathcal{M}} \rightarrow^3 P_{\mathcal{M}, v_{m+1} \mapsto tt}$ and $P_{\mathcal{M}} \rightarrow^3 P_{\mathcal{M}, v_{m+1} \mapsto ff}$. Moreover, there does not exist P' such that $P' \not\equiv P_{\mathcal{M}, v_{m+1} \mapsto tt}$, $P' \not\equiv P_{\mathcal{M}, v_{m+1} \mapsto ff}$ and $P_{\mathcal{M}} \rightarrow^3 P'$.

Proof. For $m < n - 1$, we consider \mathcal{M} to be $v_1 \mapsto t_1, \dots, v_m \mapsto t_m$ and we have $\varphi_m = Q_{m+1} v_{m+1} \dots Q_n v_n \psi$. Whatever Q_{m+1} is, by definition of *enc*,

$$P^{\varphi_m} = v'_{m+1}[\text{in } v_{m+1}.v_{m+2}[\text{out } v'_{m+1}.\text{out } v_{m+1}.R^{\varphi_{m+1}}]]$$

for $P^{\varphi_{m+1}} = v'_{m+1}[R^{\varphi_{m+1}}]$. Now the only thing the process $P_{\mathcal{M}}$ can do is to nondeterministically choose one of the occurrences of v_{m+1} and to go inside it. In the further two deterministic steps the whole process reduces to either $P_{\mathcal{M}, v_{m+1} \mapsto tt}$ or $P_{\mathcal{M}, v_{m+1} \mapsto ff}$. The case where $m = n - 1$ is similar. \square

Lemma D.2. For all m in $\{0, \dots, n-1\}$, \mathcal{M} being the interpretation $v_1 \mapsto t_1, \dots, v_m \mapsto t_m$, we have

- for $0 \leq m < n-1$, $P_{\mathcal{M}, v_{m+1} \mapsto tt}$ and $P_{\mathcal{M}, v_{m+1} \mapsto ff}$ are the two unique processes reachable from $P_{\mathcal{M}}$ that satisfy the ambient formula $v'_{m+2}[\mathbf{T}]|\mathbf{T}$.
- for $m = n-1$, $P_{\mathcal{M}, v_{m+1} \mapsto tt}$ and $P_{\mathcal{M}, v_{m+1} \mapsto ff}$ are the two unique processes reachable from $P_{\mathcal{M}}$ that satisfy the ambient formula $end[\mathbf{T}]|\mathbf{T}$.

Proof. For $0 \leq m < n-1$, we know from the proof of Lemma D.1 that both $P_{\mathcal{M}, v_{m+1} \mapsto tt}$ and $P_{\mathcal{M}, v_{m+1} \mapsto ff}$ satisfy the ambient formula $v'_{m+2}[\mathbf{T}]|\mathbf{T}$ and do not satisfy formulas $v'[\mathbf{T}]|\mathbf{T}$ where v' is a primed ambient name different from v'_{m+2} . Now, still from the proof of Lemma D.1, we know that any reachable process from $P_{\mathcal{M}}$ is either $P_{\mathcal{M}'}$ for some extension \mathcal{M}' of \mathcal{M} or an “intermediate” process reachable from $P_{\mathcal{M}'}$ in one or two steps. It is easy to see that none of these “intermediate” processes satisfies an ambient formula $v'[\mathbf{T}]|\mathbf{T}$ whatever the primed name v' is. Finally, as \mathcal{M}' is different from \mathcal{M} , $P_{\mathcal{M}'}$ will satisfy a formula $v'[\mathbf{T}]|\mathbf{T}$ for some $v' \neq v'_{m+2}$, but not the formula $v'_{m+2}[\mathbf{T}]|\mathbf{T}$.

The proof goes in a similar way for the case where $m = n-1$. \square

Restatement of Lemma 5.3. 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.

Proof. We are going to show for any $0 \leq m \leq n$ that for the interpretation \mathcal{M} equal to $v_1 \mapsto t_1, \dots, v_m \mapsto t_m$

$$\mathcal{M} \models \varphi_m \quad \text{iff} \quad P_{\mathcal{M}} \models \mathcal{A}^{\varphi_m}$$

Note that for $m=0$, \mathcal{M} is the empty interpretation, $\varphi_m = \varphi$, $P_{\mathcal{M}} = P$ and $\mathcal{A}^{\varphi_m} = \mathcal{A}$, so this statement obviously implies Lemma 5.3. The proof of this statement goes by induction on the number $l = n - m$ of quantifiers in φ_m .

For the base case $l=0$: $\varphi_m = C_1 \wedge \dots \wedge C_k$ is an unquantified formula and $\mathcal{M} = v_1 \mapsto t_1, \dots, v_n \mapsto t_n$. The interpretation \mathcal{M} is a model for the formula φ_m if and only if \mathcal{M} renders true at least one literal ℓ_i in each of the clauses C_i . Now, depending on whether ℓ_i occurs positively or negatively in C_i , we have two cases:

- $\ell_i = v_i$: by the encoding and the definition of $P_{\mathcal{M}}$, this is equivalent to that $\llbracket \ell_i \rrbracket = v_i[pos[\mathbf{0}] | v'_i[\mathbf{0}]]|\mathbf{T}$ and $P_{\mathcal{M}} = v_i[pos[\mathbf{0}] | v'_i[\mathbf{0}]]|P'$ for some ambient process P' which does not contain the ambient name v'_i . Therefore, it is equivalent to that $P_{\mathcal{M}} \models \llbracket \ell_i \rrbracket$.
- $\ell_i = \bar{v}_i$: this case is dual to the previous one.

Now, in both cases we have $P_{\mathcal{M}} \models \llbracket \ell_i \rrbracket$, which means that $P_{\mathcal{M}}$ is a model for at least one literal in each of the $\llbracket C_i \rrbracket$'s, and thus it is equivalent to that $P_{\mathcal{M}} \models \mathcal{A}^{\varphi_m}$.

For the induction step $1 < l \leq n$ (the particular base case where $l=1$ differs only in the use of the ambient name end instead of v'_{n+1} and can be proved in the same way) we assume that the statement holds for $l-1$ (that is, it holds for $m+1$). The formula φ_m has the form $Q_{m+1}v_{m+1}\varphi_{m+1}$, so we have to consider two cases depending on whether Q_{m+1} is \exists or \forall .

In the case of \exists , we have $\mathcal{M} \models \varphi_m$ is equivalent to the disjunction $\mathcal{M}, v_{m+1} \mapsto tt \models \varphi_{m+1}$ or $\mathcal{M}, v_{m+1} \mapsto ff \models \varphi_{m+1}$. By induction hypothesis, this is equivalent to that either

$P_{\mathcal{M}, v_{m+1} \mapsto tt} \models \mathcal{A}^{\varphi_{m+1}}$ or $P_{\mathcal{M}, v_{m+1} \mapsto ff} \models \mathcal{A}^{\varphi_{m+1}}$. By Lemma D.2, we know that $P_{\mathcal{M}, v_{m+1} \mapsto tt}$ and $P_{\mathcal{M}, v_{m+1} \mapsto ff}$ are the two unique processes reachable from $P_{\mathcal{M}}$ satisfying the ambient formula $v'_{m+2}[\mathbf{T} | \mathbf{T}]$. Therefore, the last statement is equivalent to that

$$P_{\mathcal{M}} \models \diamond(v'_{m+2}[\mathbf{T} | \mathbf{T}]) \wedge \mathcal{A}^{\varphi_{m+1}}.$$

The case where $Q_{n-l+1} = \forall$ is dual to the previous one and leads to the equivalence with

$$P_{\mathcal{M}} \models \square(v_{m+2}[\mathbf{T} | \mathbf{T}]) \Rightarrow \mathcal{A}^{\varphi_{m+1}}.$$

In both cases, by definition of *enc*, we have the equivalence with $P_{\mathcal{M}} \models \mathcal{A}^{\varphi_m}$. \square

D.3. Proof of Lemma 5.5

Lemma 5.5 is the crux of correctness for the encoding from Section 5.3 of QBF satisfaction in the calculus of immobile ambients with I/O. To prove it, let us first fix some notations and then prove some auxiliary lemmas.

We use notations similar to the previous section. For a given closed QBF formula $\varphi = Q_1 v_1 \dots Q_n v_n \psi$ in prenex and conjunctive normal form, we denote ψ by $C_1 \wedge \dots \wedge C_k$. Let \mathcal{M} be an interpretation $v_1 \mapsto t_1, \dots, v_m \mapsto t_m$. We denote $\sigma_{\mathcal{M}}$ the substitution $\{v_1 \leftarrow t_1, \bar{v}_1 \leftarrow \bar{t}_1, \dots, v_m \leftarrow t_m, \bar{v}_m \leftarrow \bar{t}_m\}$ where \bar{t}_i is the negated value of t_i . If \mathcal{M} is the empty interpretation, we let $\sigma_{\mathcal{M}}$ to be the identity.

For $0 \leq m \leq n$, let φ_m be the formula $Q_{m+1} v_{m+1} \dots Q_n v_n \psi$ and $enc(\varphi_m) = (P^{\varphi_m}, \mathcal{A}^{\varphi_m})$. For $\mathcal{M} = v_1 \mapsto t_1, \dots, v_m \mapsto t_m$, let us denote $P_{\mathcal{M}}$ the process $Q^{\varphi_m} \sigma_{\mathcal{M}}$ such that $P^{\varphi_m} \equiv v'_{m+1}[Q^{\varphi_m}]$. Note that in this notation $P^{\varphi_m} \sigma_{\mathcal{M}} = v'_{m+1}[P_{\mathcal{M}}]$. By \mathcal{M}^+ and \mathcal{M}^- we denote respectively $\mathcal{M}, v_{m+1} \leftarrow tt, \bar{v}_{m+1} \leftarrow ff$ and $\mathcal{M}, v_{m+1} \leftarrow ff, \bar{v}_{m+1} \leftarrow tt$.

Lemma D.3. For all $0 \leq m < n$,

$$P_{\mathcal{M}} \rightarrow (\langle ff \rangle | v''_{m+1}[\square | (\bar{v}_{m+1})] \cdot P^{\varphi_{m+1}}) \sigma_{\mathcal{M}, v_{m+1} \mapsto tt}$$

and

$$P_{\mathcal{M}} \rightarrow (\langle tt \rangle | v''_{m+1}[\square | (\bar{v}_{m+1})] \cdot P^{\varphi_{m+1}}) \sigma_{\mathcal{M}, v_{m+1} \leftarrow ff}$$

and there is no other P' such that $P \rightarrow P'$.

Proof. Straightforward from the encoding. \square

Lemma D.4. For all $0 \leq m < n$, $P_{\mathcal{M}} \rightarrow^2 (v''_{m+1}[\square | P^{\varphi_{m+1}}] \sigma_{\mathcal{M}^+}$ and $P_{\mathcal{M}} \rightarrow^2 (v''_{m+1}[\square | P^{\varphi_{m+1}}] \sigma_{\mathcal{M}^-}$ and there is no other P' such that $P \rightarrow^2 P'$.

Proof. Straightforward from the encoding, Lemma D.3 and the definition of $P_{\mathcal{M}}$. \square

Restatement of Lemma 5.5. 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.

Proof. Let $V_0 = \mathbf{0}$ and for all $1 \leq m \leq n$ let $V_m = v_m''[\]$. We are going to show for any $0 \leq m \leq n$ that for the interpretation \mathcal{M} equal to $v_1 \mapsto t_1, \dots, v_m \mapsto t_m$,

$$\mathcal{M} \models \varphi_m \text{ iff } V_m | P^{\varphi_m} \sigma_{\mathcal{M}} \models \mathcal{A}^{\varphi_m}.$$

The particular case of this statement with $m=0$ is equivalent to Lemma 5.5. Its proof goes by induction over the number $l = n - m$ of quantified variables in φ_m .

Case where $l=0$: the formula φ_m is equal to $C_1 \wedge \dots \wedge C_k$, \mathcal{M} has the form $v_1 \mapsto t_1, \dots, v_n \mapsto t_n$ and $\mathcal{M} \models C_1 \wedge \dots \wedge C_k$. As $C_1 \wedge \dots \wedge C_k$ is in conjunctive normal form, for at least one literal ℓ_i in each C_i , $\mathcal{M}(\ell_i) = tt$. This is equivalent to that for each C_i , there exists at least one literal ℓ_i in C_i such that

- $v_j \leftarrow tt, \bar{v}_j \leftarrow ff$ belongs to $\sigma_{\mathcal{M}}$ if $\ell_i = v_j$ and
- $v_j \leftarrow ff, \bar{v}_j \leftarrow tt$ belongs to $\sigma_{\mathcal{M}}$ if $\ell_i = \bar{v}_j$.

By the definition of $enc(C_1 \wedge \dots \wedge C_k)$, this is equivalent to that the interior of each C ambient (each marked by a D ambient) in the process $P^{\varphi_m} \sigma_{\mathcal{M}}$ contains a tt subambient. This again is equivalent to $P^{\varphi_m} \sigma_{\mathcal{M}} \models \square((D[\mathbf{0}] | \mathbf{T}) \Rightarrow (tt[\mathbf{0}] | \mathbf{T}))$ that is, to $P^{\varphi_m} \sigma_{\mathcal{M}} \models \mathcal{A}^{\varphi_m}$. Since V_m does not contain any subambient $D[\mathbf{0}]$, the statement follows.

Case where $l=1$ (that is, $m = n - 1$): the formula φ_m is equal to $Q_n v_n \psi$, \mathcal{M} is the form $v_1 \mapsto t_1, \dots, v_{n-1} \mapsto t_{n-1}$. We follow according to the value of Q_n :

- Case where $Q_n = \exists$: $\mathcal{M} \models \varphi_m$ is equivalent to either $\mathcal{M}, v_n \leftarrow tt \models \psi$ or $\mathcal{M}, v_n \leftarrow ff \models \psi$. Using the case where $l=0$, this is equivalent to that either $P^{\varphi_n} \sigma_{\mathcal{M}^+} \models \mathcal{A}^{\varphi_n}$ or $P^{\varphi_n} \sigma_{\mathcal{M}^-} \models \mathcal{A}^{\varphi_n}$.

By Lemma D.4, the processes $v_n''[\] | P^{\varphi_n} \sigma_{\mathcal{M}^+}$ and $v_n''[\] | P^{\varphi_n} \sigma_{\mathcal{M}^-}$ are the two unique ones reachable from $P_{\mathcal{M}}$ in two steps. Moreover, as P^{φ_n} cannot be reduced, there is no process reachable from $P_{\mathcal{M}}$ in strictly more than two steps. It should be noticed that $P^{\varphi_n} \sigma_{\mathcal{M}^+}$ and $P^{\varphi_n} \sigma_{\mathcal{M}^-}$ both satisfy the formula $Inst(end) \wedge \neg Inst^+(end)$ whereas by Lemma D.3 the two unique successors of $P_{\mathcal{M}}$ as well as $P_{\mathcal{M}}$ itself do not satisfy the formula $Inst(end)$. Therefore, $P^{\varphi_n} \sigma_{\mathcal{M}^+} \models \mathcal{A}^{\varphi_n}$ or $P^{\varphi_n} \sigma_{\mathcal{M}^-} \models \mathcal{A}^{\varphi_n}$ holds iff $P_{\mathcal{M}} \models \diamond((Inst(end) \wedge \neg Inst^+(end)) \wedge \mathcal{A}^{\varphi_n})$. And thus, this is equivalent to $v_{n-1}''[\] | v_n[P_{\mathcal{M}}] \models \mathbf{T} | v_n[\diamond((Inst(end) \wedge \neg Inst^+(end)) \wedge \mathcal{A}^{\varphi_n})]$, that is $v_{n-1}''[\] | P^{\varphi_{n-1}} \models \mathcal{A}^{\varphi_{n-1}}$.

- Case where $Q_n = \forall$: this case is dual to the previous one.

Case where $1 < l \leq n$: the formula φ_m is equal to $Q_{m+1} v_{m+1} \varphi_{m+1}$, \mathcal{M} has the form $v_1 \mapsto t_1, \dots, v_m \mapsto t_m$ and we assume that the statement holds for $l-1$ (that is, it holds for $m+1$). We follow according to the value of Q_{m+1} :

- Case where $Q_{m+1} = \exists$: $\mathcal{M} \models \varphi_m$ is equivalent to either $\mathcal{M}, v_{m+1} \leftarrow tt \models \varphi_{m+1}$ or $\mathcal{M}, v_{m+1} \leftarrow ff \models \varphi_{m+1}$. By induction hypothesis, this is equivalent to that either $v_{m+1}''[\] | P^{\varphi_{m+1}} \sigma_{\mathcal{M}^+} \models \mathcal{A}^{\varphi_{m+1}}$ or $v_{m+1}''[\] | P^{\varphi_{m+1}} \sigma_{\mathcal{M}^-} \models \mathcal{A}^{\varphi_{m+1}}$.

Let us have a look now at processes reachable from $P_{\mathcal{M}}$: of course, $P_{\mathcal{M}}$ itself is reachable, but by construction it does not satisfy the formula $Inst(v_{m+2}')$. By Lemma D.3, two processes are reachable in one step from $P_{\mathcal{M}}$, but they do not satisfy the formula $Inst(v_{m+2}')$. By Lemma D.4, two processes are reachable from $P_{\mathcal{M}}$ in two steps, namely $(v_{m+1}''[\] | P^{\varphi_{m+1}}) \sigma_{\mathcal{M}^+}$ and $(v_{m+1}''[\] | P^{\varphi_{m+1}}) \sigma_{\mathcal{M}^-}$ and they both satisfy the formulas $Inst(v_{m+2}')$ and $\neg Inst^+(v_{m+2}')$ (by construction). Now, by using once again

Lemma D.3 for the internal of v'_{m+2} in $P^{\varphi_{m+1}}\sigma_{\mathcal{M}^+}$ and $P^{\varphi_{m+1}}\sigma_{\mathcal{M}^-}$, all the processes reachable from one of those latter satisfy $Inst^+(v'_{m+2})$.

Therefore, the last statement is equivalent to that $P_{\mathcal{M}} \models \diamond(Inst(v'_{m+2}) \wedge \neg Inst^+(v'_{m+2})) \wedge \mathcal{A}^{\varphi_{m+1}}$. Thus, it is equivalent to $V_m[] | v'_{m+1}[P_{\mathcal{M}}] \models \mathbf{T} | v'_{m+1}[\diamond(Inst(v'_{m+2}) \wedge \neg Inst^+(v'_{m+2})) \wedge \mathcal{A}^{\varphi_{m+1}}]$, that is $V_m[] | P^{\varphi_m} \models \mathcal{A}^{\varphi_m}$.

- The case where $Q_{m+1} = \forall$ is dual to the previous one. \square

References

- [1] N. Busi, G. Zavattaro, On the expressiveness of movement in pure mobile ambients, in: Foundations of Wide Area Network Computing, 2002, Electron. Notes in Theoret. Comput. Sci. 66 (3), Elsevier, 2002.
- [2] L. Caires, L. Cardelli, A spatial logic for concurrency (part I), in: Theoretical Aspects of Computer Software (TACS 2001), Lecture Notes in Computer Science, Vol. 2215, Springer, Berlin, 2001, pp. 1–37.
- [3] L. Caires, L. Cardelli, A spatial logic for concurrency (part II), in: CONCUR 2002—Concurrency Theory, Lecture Notes in Computer Science, Vol. 2421, Springer, Berlin, 2002, pp. 209–225.
- [4] L. Caires, L. Monteiro, Verifiable and executable logic specifications of concurrent objects in L_{π} , in: European Symp. on Programming (ESOP'99), Lecture Notes in Computer Science, Vol. 1381, Springer, Berlin, 1998, pp. 42–56.
- [5] L. Cardelli, P. Gardner, G. Ghelli, A spatial logic for querying graphs, in: Automata, Languages and Programming (ICALP'02), Lecture Notes in Computer Science, Vol. 2380, Springer, Berlin, 2002, pp. 597–610.
- [6] L. Cardelli, G. Ghelli, A query language based on the ambient logic, in: European Symposium on Programming (ESOP'01), Lecture Notes in Computer Science, Vol. 2028, Springer, Berlin, 2001, pp. 1–22.
- [7] L. Cardelli, A. Gordon, Logical properties of name restriction, in: Typed Lambda Calculi and Applications (TLCA'01), Lecture Notes in Computer Science, Vol. 2044, Springer, Berlin, 2001, pp. 46–60.
- [8] L. Cardelli, A.D. Gordon, Types for mobile ambients, in: 26th ACM Symp. on Principles of Programming Languages (POPL'99), 1999, pp. 79–92.
- [9] L. Cardelli, A.D. Gordon, Anytime, anywhere: Modal logics for mobile ambients, in: 27th ACM Symp. on Principles of Programming Languages (POPL'00), 2000, pp. 365–377.
- [10] L. Cardelli, A.D. Gordon, Mobile ambients, Theoret. Comput. Sci. 240 (2000) 177–213.
- [11] W. Charatonik, S. Dal Zilio, A.D. Gordon, S. Mukhopadhyay, J.-M. Talbot, The complexity of model checking mobile ambients, in: Foundations of Software Science and Computation Structures (FoSSaCS'01), Lecture Notes in Computer Science, Vol. 2030, Springer, Berlin, 2001, pp. 152–167.
- [12] W. Charatonik, A.D. Gordon, J.-M. Talbot, Finite-control mobile ambients, in: European Symposium on Programming (ESOP'02), Lecture Notes in Computer Science, Vol. 2305, Springer, Berlin, 2002, pp. 295–313.
- [13] W. Charatonik, J.-M. Talbot, The decidability of model checking mobile ambients, in: Computer Science Logic (CSL'01), Lecture Notes in Computer Science, Vol. 2142, Springer, Berlin, 2000, pp. 339–354.
- [14] S. Dal Zilio, Spatial congruence for ambients is decidable, in: 6th Asian Computing Sci. Conf. (ASIAN'00), Lecture Notes in Computer Science, Vol. 1961, Springer, Berlin, 2000, pp. 88–103.
- [15] M. Dam, Relevance logic and concurrent composition, in: Logic in Computer Science (LICS'88), IEEE, New York, 1988, pp. 178–185.
- [16] K. Došen, A historical introduction to substructural logics, in: P. Schroeder-Heistler, K. Došen (Eds.), Substructural Logics, Studies in Logic and Computation, Vol. 2, Clarendon Press, Oxford, 1994, pp. 1–30.
- [17] G.-Y. Girard, Y. Lafont, Linear logic and lazy computation, in: TAP-SOFT 87, Lecture Notes in Computer Science, Vol. 250 (2), Springer, Berlin, 1987, pp. 53–66.

- [18] A.D. Gordon, L. Cardelli, Equational properties of mobile ambients, *Math. Structures Comput. Sci.* 12 (2002) 1–38.
- [19] D. Hirschhoff, E. Lozes, D. Sangiorgi, Separability, expressiveness and decidability in the ambient logic, in: *Logic in Computer Science (LICS'02)*, IEEE, New York, 2002, pp. 423–432.
- [20] S. Ishtiaq, P.W. O'Hearn, BI as an assertion language for mutable data structures, in: *28th ACM Symp. on Principles of Programming Languages (POPL'01)*, 2001, pp. 14–26.
- [21] J.W. Lloyd, *Foundations of Logic Programming*, in: *Symbolic Computation*, 2nd extended Edition, Springer, Berlin, 1987.
- [22] P. O'Hearn, J. Reynolds, H. Yang, Local reasoning about programs that alter data structures, in: *Computer Science Logic (CSL'01)*, *Lecture Notes in Computer Science*, Vol. 2142, Springer, Berlin, 2001, pp. 1–19.
- [23] P.W. O'Hearn, D.J. Pym, The logic of bunched implications, *Bull. Symbolic Logic* 5 (2) (1999) 215–244.
- [24] E.L. Post, Recursively enumerable sets of positive integers and their decision problems, *Bull. Amer. Math. Soc.* 50 (1944) 284–316.
- [25] J.C. Reynolds, *Intuitionistic reasoning about shared mutable data structure*, in: *Millennial Perspectives in Computer Science*, Palgrave, Basingstoke, 2000.
- [26] D. Sangiorgi, Extensionality and intensionality of the ambient logics, in: *28th ACM Symp. on Principles of Programming Languages (POPL'01)*, 2001, pp. 4–13.
- [27] W. Savitch, Relationships between nondeterministic and deterministic tape complexities, *J. Comput. System Sci.* 4 (2) (1970) 177–192.
- [28] L.J. Stockmeyer, The polynomial-time hierarchy, *Theoret. Comput. Sci.* 3 (1) (1976) 1–22.
- [29] B.A. Trakhtenbrot, The impossibility of an algorithm for the decision problem for finite models, *Dokl. Akad. Nauk SSR* 70 (1950) 569–572.
- [30] A. Urquhart, Semantics for relevant logics, *J. Symbolic Logic* 37 (1) (1972) 159–169.