Model-Checking of Spatial Properties in the π-Calculus

Luís Caires
Departamento de Informática FCT/UNL, Lisboa, Portugal

December 2002

Abstract. In addition to behavioral properties, spatial logics can talk about other key properties of concurrent systems like secrecy, freshness, usage of resources, and distribution. We present a model-checker for a spatial logic suited to systems specified in the π-calculus. Model-checking is decidable for a useful class of processes that includes the processes with finite-control.

1 Introduction

A concurrent system is often seen as a whole whose behavior is distributed among an evolving collection of interacting separate parts, assembled together according to a certain spatial structure. Spatial logics support the specification not only of behavioral properties but also of structural properties of concurrent systems, in a fairly integrated way. In this paper, we study model-checking in a spatial logic for systems expressed in the π-calculus, that includes spatial and temporal operators, freshness quantifiers, and recursive formulas.

1.1 Spatial Properties in Concurrency

Spatial logics are modal logics in which one can not only express properties of the worlds, but also talk about the internal structure of each world, and describe how properties of its parts (also understood as worlds themselves) relate to the properties of the whole. The interpretation of each world as a structured space, and moreover as a space interpreted as a resource of a certain kind, distinguishes spatial logics among modal logics.

Spatial logics have been proposed for various programming languages and calculi, for instance, the process logic of [6, 2]; the ambient logics of [11, 12], where each world is a process of the ambient calculus; the logics of [28, 22], where each world is an heap state of an imperative program, and the spatial logic for concurrency of [3, 5, 4], where each world is a π-calculus process. Spatial logic has also proved to be useful in the definition of type systems and query languages for various data models [10, 9, 7, 8].

In spatial logics for concurrency, the truth of a property depends not only on time, as in more usual process logics, but also on the spatial location where the property is evaluated. The ability to talk about structural properties of systems, for instance, about the maximum number of components in a system with evolving structure, or about the amount of resources, seems to require some degree of spatial observation. Spatial properties arise frequently in the specification of distributed systems, *e.g.*, when describing a location-dependent policy to access resources. In fact, many key properties of concurrent systems are inherently spatial, for instance connectivity, stating that there is always an access route between two different sites, or unique handling, stating that there is at most one server process listening on a given channel name. Secrecy and non-interference properties can also be understood in spatial terms, since a secret is a piece of data whose knowledge of is restricted to some parts of a system, and unforgeable by other parts [5, 4].

1.2 Spatial and Behavioral Observations

In behavioral models of concurrency, a process is identified with its observable behavior, roughly, the sequence of actions it can perform in the course of time. The modalities of a "pure" behavioral process logic allow us to talk about the actions of processes, and thus support the specification of processes at the behavioral level; logics of this kind are either variants or extensions of those originally introduced by Hennessy and Milner [24].

In models for concurrency in the CCS tradition, including the π-calculus, an action by a process is always a communication, either between two distinct parts of the system, or between the system and its environment. Such actions can be expressed as labels of a transition system that defines the behavioral semantics of the process model. The modal operators of a behavioral logic then arise almost automatically from such a set of labels.

The definition of a logic along these lines thus comes after the definition of natural notions of labelled transition system and behavioral congruence. These notions may be hard to obtain for some process models (*e.g.*, the ambient calculus, the spi calculus), but are well established in the π-calculus case, in which communication just consists in sending or receiving (channel) names on channel names. Therefore, in the π-calculus case, for each label α a logical modality $\langle\alpha\rangle A$ can be defined. For instance, a process satisfies formula $\langle m(n)\rangle A$ if it can receive name n in channel m and after become a process that satisfies formula A. Several logics along these lines were first proposed and studied by Milner, Parrow and Walker [26]; and then also by Dam [16, 17]. The equivalence relation induced on the set of all processes by such logics is expected to match some notion of behavioral congruence (*e.g.*, early or late strong bisimulation).

However, with spatial logics we want to observe not just the behavior of processes, but also their internal structure. Spatial observations may appear intensional, because they can discriminate between systems that just differ on the internal structure, not in their behavior in the sense discussed above. However, following [5], we may take the view that spatial observations are actually extensional, but with respect to a richer semantic domain.

In process algebra, one usually distinguish between static and dynamic operators [24]: the static operators allow the assembly of systems from parts, the dynamic operators allow the definition of behaviors from behaviors. A behavioral semantics tends to blur the distinction between the two kinds of operations, to the extent that all process operators end up interpreted as purely behavioral, abstracting away from all structural information.

A spatial logic adds to a given set of behavioral modalities a set of spatial operators, related to the static operators of the process calculus, an early proposal along these lines is [6, 2]. In the case of nominal process calculi, the static operators are the parallel composition $P|Q$ and its neutral element $\mathbf{0}$ (denoting the empty system) and the name restriction $(\boldsymbol{\nu}n)P$. These process constructors give rise to the composition formula $A|B$, that holds of a process that can be separated (up to structural congruence) into a process that satisfies formula A and a process that satisfies formula B, to the void formula $\mathbf{0}$, that holds of the void process, and to the hidden name quantifier $\mathsf{H}x.A$, that allows us to quantify over restricted names.

Alternatively, a spatial logic can put a strong emphasis on structure, and just allow the observation of the behavior of processes in a rather indirect way, by looking how their spatial structure changes in reaction to the environment. The first proposal in this vein is the ambient logic of Cardelli and Gordon [11], it requires all observations of processes to be done essentially through the use of spatial modalities. The behavioral aspects are then handled by a minimal "next step" modality (corresponding to the $\langle\tau\rangle A$ formula) or a (more lax) "eventually true" modality. An advantage of this approach is its generality: it can be easily adapted to any process calculus whose operational semantics can be given by a reduction relation, in particular to those for which a natural labelled transition semantics may be hard to find. But reduction just captures the computational behavior of closed systems. Therefore, to enable the expression of contextual and security properties, the ambient logic includes adjuncts among its spatial operators.

Spatial adjuncts arise canonically given a set of basic spatial operators, and allow logical formulas to talk not just about the system, but also about properties the system can ensure when placed into an environment. For example, the adjunct of the composition connective is the guarantee operator $A \triangleright B$ [11]. A process P satisfies $A \triangleright B$ if for every process Q satisfying A, it holds that $P|Q$ satisfies B. The guarantee operator makes it possible to approximate behavioral observations, and express quite general context/system specifications. The spatial logic for concurrency of [5, 4], also includes adjuncts among its spatial operators. Adjuncts are very expressive. For instance, guarantee allows an internal definition of validity $!A \triangleq (\neg A) \triangleright \mathbf{F}$ [11]: a process satisfies $!A$ if and only if A is satisfied by all processes; this definition fully exploits the universal quantification over all processes involved in the semantical definition of the guarantee operator. Thus, even for finite processes, model-checking of spatial logics with guarantee and quantification over names is undecidable [14], and (heavily) intractable for a simple propositional fragment [7].

The logic we study in this paper is based on purely structural spatial and behavioral observations. By "purely structural observations" we mean observations that can be concluded just by inspecting the local structure of the process at hand; therefore it does not contain composition adjunct among its operators. Thus, this logic can talk about the behavior and spatial structure of the system under consideration, but not about the autonomous behavior and spatial structure of the environment. As a consequence, we recover decidability of model-checking on expressive classes of processes and formulas, while preserving some ability to express useful context-sensitive behavioral and spatial properties.

The logic is based on a rather small set of space-time observations. For the spatial fragment we consider the connectives of composition match, void and revelation. For the behavioral fragment we just consider the basic modalities:

			Observations	*Formulas*
P	$\equiv$	$\mathbf{0}$	the process is void	$\mathbf{0}$
P	$\equiv$	$Q\|R$	the process can be split into Q and R	$A\|B$
P	$\equiv$	$(\boldsymbol{\nu} n)Q$	the process can reveal Q under n	$n®A$
P	$\xrightarrow{\tau}$	Q	the process can reduce to Q	$\tau.A$
P	$\xrightarrow{n\langle m\rangle}$	Q	the process can send m on n, and proceed as Q	$n\langle m\rangle.A$
P	$\xrightarrow{n(m)}$	Q	the process can receive m on n, and proceed as Q	$n(m).A$

Actually, instead of the basic composition connective, we introduce in the present logic a (more general) form of dependent composition, written $\mathcal{X} : A|B$. This formula means the same as $A|B$ except that the bound process variable $\mathcal{X}$, whose scope is formula B, is interpreted as the part of the process that satisfies A; we write $A|B$ for $\mathcal{X} : A|B$ when $\mathcal{X}$

does not occur free in B. The process variable $\mathcal{X}$ can then be used inside B as a proposition, in particular in a formula $A \triangleleft \mathcal{X}$, that we call here "recomposition", but that means the same as $\mathcal{X} \triangleright A$. So, both $\mathcal{X} : A|B$ and $A \triangleleft \mathcal{X}$ are definable in the extension of the logic of [5] with process variables and quantification over processes (*cf.*, tree variables and quantification over trees [10]).

A behavioral modality is defined either from the label τ, that denotes an internal communication, or from one of the labels $n\langle m\rangle$ and $n(m)$, denoting respectively the action of sending name m of channel n, and the action of receiving name m on channel n. To this basic set of connectives, we then add propositional operators, first-order and freshness quantifiers, and recursive definitions, following closely the logic of [5].

To illustrate the expressiveness of the logic, we run through a sequence of examples. The input and output labels used in the primitive behavioral modalities are closely related to the input and output constructors of the process calculus; to cope with extrusion of private names, the set of labels used in standard labelled transition system for the π-calculus includes another couple of labels. We then have $(\boldsymbol{\nu} m)n\langle m\rangle$, that denotes the output on channel n of a new (freshly generated by the sender) name m. This action corresponds to the first half (scope opening) of the mechanism of scope extrusion in the π-calculus. A dual label $(\boldsymbol{\nu} m)n(m)$ corresponds to the second half (scope closing) of the mechanism of scope extrusion. Combining the freshness and hidden name quantifiers with the behavioral operator, we can define modalities for name extrusion and intrusion (*cf.*,[26]).

$$
\begin{array}{rcll}
n\langle \boldsymbol{\nu} x\rangle.A & \triangleq & \mathsf{H}x.n\langle x\rangle A & \text{(Bound Output)} \\
n(\boldsymbol{\nu} x).A & \triangleq & \mathsf{И}x.n(x)A & \text{(Bound Input)}
\end{array}
$$

The definition of bound output uses the hidden name quantifier [2, 5]. The hidden name quantifier can also be derived ([12]) from the fresh name quantifier and the revelation operator: $\mathsf{H}x.A \triangleq \mathsf{И}x.x\circledR A$. We can then see that a process P satisfies the formula $\mathsf{H}x.n\langle x\rangle A$ if there is a fresh name m and a process Q such that $P \equiv (\boldsymbol{\nu} m)Q$ and Q satisfies $n\langle m\rangle A$, that is, there is R such that $Q \overset{n\langle m\rangle}{\to} R$ and R satisfies A. From the standard labelled transition system of the π-calculus, we conclude that $P \overset{\alpha}{\to} R$, where α is the bound output action $(\boldsymbol{\nu} m)n\langle m\rangle$. For example using the two operators above we can define the formula

$$I \triangleq m\langle \boldsymbol{\nu} x\rangle.A|m(\boldsymbol{\nu} x)B \Rightarrow \tau.\mathsf{H}x.(A|B)$$

This formula talks about name extrusion: it says that after interacting, two separate parts of the system can become "connected" by a shared secret. The process

$$P \triangleq ((\boldsymbol{\nu} n)m\langle n\rangle.n\langle m\rangle.\mathbf{0})|m(q).q\langle q\rangle.\mathbf{0}$$

satisfies the formula I above. It also satisfies the formula $(\neg\mathbf{0}|\neg\mathbf{0}) \wedge \tau.\neg(\neg\mathbf{0}|\neg\mathbf{0})$: this formula says that the process has two separate threads initially, but becomes single-threaded after a reduction step. This simple example illustrates how a spatial logic can capture modifications of spatial structure induced by name extrusion. Using together spatial operators and recursive formulas we can define many useful operators, *e.g.*,

$$\mathsf{H}^* A \triangleq \boldsymbol{\mu} X.(A \vee \mathsf{H}x.X)$$

$\mathsf{H}^* A$ means that A holds under a (finite) number of restricted names [2, 5]. For example, the formula

$$\neg\mathsf{H}^*\exists y.(\exists x.y(x)\mathbf{T}|\exists x.y(x)\mathbf{T})$$

expresses a unique handling property [25]: it says that there are no separate processes listening on the same (public or private) channel name. A quite similar property, but related to resource usage, can then be expressed by the formula

$$\neg\mathsf{H}^{*}\exists y.(\exists x.y\langle x\rangle.\mathbf{T}|\exists x.y\langle x\rangle.\mathbf{T})$$

It says that there are no independent requests competing for access to the same resource in the process at hand (we think of a "resource" as a server process, that is, a process listening on a given channel name). Combining the action modalities with recursion we can easily express the anytime modality $\Box A$.

$$\begin{array}{lcl} [*]A & \triangleq & \forall xy([\tau]A \wedge [x\langle y\rangle]A \wedge [x(y)]A \wedge [x\langle \boldsymbol{\nu} z\rangle]A) \\ \Box A & \triangleq & \boldsymbol{\nu}X.(A \wedge [*]X) \end{array}$$

The formula $\Box A$ states that A will always hold of the system, even when it interacts with an arbitrary environment (*e.g.*, an attacker). Hence, the following formula expresses a secrecy property, namely it asserts that it will never be the case that the system contains a secret resource satisfying property $P(x)$ whose identity x is about to be sent to the environment.

$$\Box\neg\exists y.\mathsf{H}x.((P(x)|\mathbf{T}) \wedge y\langle x\rangle\mathbf{T})$$

As a final example, illustrating a natural use of dependent composition consider

$$\mathcal{X} : S|((\neg\tau.\mathbf{T}) \wedge \boldsymbol{\mu}Z.(\mathbf{0} \vee ((\tau.\mathbf{T} \triangleleft \mathcal{X})|Z)))$$

A process satisfying this formula is composed by a part satisfying property S (*e.g.,* a server), and by a finite number of parts (*e.g.,* the clients) that can individually interact with the server, but not among themselves. All the examples discussed here can be directly handled by our model checking algorithm.

Related Work. A large amount of work was done on behavioral logics and model-checking for process calculi [26, 1]; in this context, the case of the π-calculus was addressed mostly by Dam [16, 17]. However, the aim to address spatial properties in nominal calculi just appears more recently [11, 12, 2, 4, 5]. Concerning model-checking for spatial logics, most of the existing work focus on the ambient calculus and the ambient logic.

Model-checking for the replication-free (finite) ambient calculus against the ambient logic without the guarantee operator was proposed by Cardelli and Gordon already in [11]. Charatonik and Talbot have shown that the model-checking problem for the ambient logic with universal quantification and guarantee, and even without the temporal modality, is undecidable for the finite fragment of the ambient calculus [14]; in the same work, the extension of the basic algorithm in [11] to a logic with name restriction is also presented. Charatonik, Gordon and Talbot also studied model-checking for an ambient logic against (recursive) finite-control ambients [13], where replication is replaced by recursion in the ambient calculus; here we also use recursion instead of recursion in the π-calculus. More recently, Calcagno, Cardelli and Gordon proved decidability for a logic for labelled trees that includes the guarantee operator [7].

The present work seems to be a first proposal for model-checking a spatial logic for the π-calculus. The logic includes behavioral and spatial operators, freshness and hidden name quantifiers, recursive formulas and processes, subsumes standard behavioral logics for the

π-calculus (*cf.*, [16]), and can be seen as a fragment of the spatial logic of [5, 4] (in the sense that the action modalities can be expressed with the composition adjunct). The semantic foundation for the model-checker strongly builds on the approach developed by Cardelli and the author in [5], which is based on a domain of finitely supported sets of processes, and on the theory of freshness by Gabbay and Pitts [20, 27]. As a consequence, even if we restrict to the behavioral and recursive fragment of the logic, the treatment of freshness and recursion in our model-checker, essentially based on the use of name transpositions, differs considerably from existing proposals (*e.g.*, [16]), and seems to yield a simpler algorithm and simpler proofs.

2 The Process Model

In this section, we recall the syntax and operational semantics of the π-calculus.

Definition 2.1 (Processes) *Given infinite sets* Λ *of* names *and* χ *of process variables, the set* $\mathcal{P}$ *of* processes *is given by*

m, n, p	$\in$	Λ (Names)
$\mathcal{X}, \mathcal{Y}, \mathcal{Z}$	$\in$	χ (Process Variables)
P, Q, R	$::=$	(Processes)
	$\mathbf{0}$	(Void)
	$P\|Q$	(Par)
	$(\boldsymbol{\nu} n)P$	(Restriction)
	$m\langle n\rangle.P$	(Output)
	$m(n).P$	(Input)
	$\mathcal{X}$	(Process variable)
	$(\mathbf{rec}\ \mathcal{X})P$	(Recursive process)

In restriction $(\boldsymbol{\nu} n)P$ and input $m(n).P$, the distinguished occurrence of the name n is binding, with scope the process P. In a recursive process $(\mathbf{rec}\ \mathcal{X})P$, the distinguished occurrence of the process variable $\mathcal{X}$ is binding, with scope the process P. For any process P, we assume defined as usual the set $na(P)$ of *names* of a process P, the set $fn(P)$ of *free names* of P and the set $fpv(P)$ of *free process variables* of P.

In a process $(\mathbf{rec}\ \mathcal{X})P$ we require all free occurrences of $\mathcal{X}$ in P to be *guarded*, that is, can only occur inside the continuation part Q of an output process $m\langle n\rangle.Q$ or input process $m(n).Q$ in P. For example, processes $(\mathbf{rec}\ \mathcal{X})\mathcal{X}$ and $(\mathbf{rec}\ \mathcal{X})(P|\mathcal{X})$ are not well-formed, while processes $(\mathbf{rec}\ \mathcal{X})\alpha.(\mathcal{X}|P)$, $(\mathbf{rec}\ \mathcal{X})(\mathbf{rec}\ \mathcal{Y})(\alpha.\mathcal{X}|\beta.\mathcal{Y})$ and $(\mathbf{rec}\ \mathcal{X})(\alpha.\mathcal{X}|\beta.\mathcal{X})$ are well-formed. A process is *closed* if all process variables $\mathcal{X}$ in it occur under the scope of a $(\mathbf{rec}\ \mathcal{X})$ binder.

Let N be a finite set of names or process variables. A name substitution θ of domain N is a mapping assigning a name $\theta(n)$ to each $n \in N$, and a process $\theta(\mathcal{X})$ to each process variable $\mathcal{X} \in N$. Outside its domain, any substitution behaves like the identity. Given a substitution θ, we denote by $\mathfrak{D}(\theta)$ its domain. We write $\{n{\leftarrow}m\}$ (resp. $\{X{\leftarrow}P\}$) for the singleton substitution of domain $\{n\}$ (resp. $\{\mathcal{X}\}$) that assigns m to n (resp. P to $\mathcal{X}$). A name permutation ρ is a bijective name substitution. As a particular case, we consider name transpositions, and write $\{m{\leftrightarrow}n\}$ for the transposition of m and n, that is, for the substitution that assigns m to n and n to m. Given a finite set of names N, we say that a name permutation ρ *fixes* N if $\rho(n) = n$ for every name $n \in N$. We denote by $\mathbb{R}_N$ the set of all name permutations that fix N.

We assume defined the relation of α-*congruence* $\equiv_\alpha$ that identifies processes up to the safe renaming of bound names and bound process variables. For any process P and substitution θ we denote by $\theta(P)$ the result of the safe application of θ to P (using α-conversion as needed to avoid illegal capture of free variables). We write $P\theta$ for $\theta(P)$ when θ has the form $\{n{\leftarrow}m\}$ or $\{n{\leftrightarrow}m\}$. Structural congruence identifies processes that have the same spatial structure:

Definition 2.2 (Structural congruence) Structural congruence $\equiv$ *is the least congruence on processes such that*

$P \equiv_\alpha Q \Rightarrow P \equiv Q$	(Struct Alpha)
$P\|\mathbf{0} \equiv P$	(Struct Par Void)
$P\|Q \equiv Q\|P$	(Struct Par Comm)
$P\|(Q\|R) \equiv (P\|Q)\|R$	(Struct Par Assoc)
$n \notin \mathit{fn}(P) \Rightarrow P\|(\boldsymbol{\nu} n)Q \equiv (\boldsymbol{\nu} n)(P\|Q)$	(Struct Res Par)
$(\boldsymbol{\nu} n)\mathbf{0} \equiv \mathbf{0}$	(Struct Res Void)
$(\boldsymbol{\nu} n)(\boldsymbol{\nu} m)P \equiv (\boldsymbol{\nu} m)(\boldsymbol{\nu} n)P$	(Struct Res Comm)
$(\mathbf{rec}\ \mathcal{X})P \equiv P\{\mathcal{X}{\leftarrow}(\mathbf{rec}\ \mathcal{X})P\}$	(Struct Rec Unfold)

A process Q is a *subterm* of P if $P \equiv C[Q]$. Note that due to the presence of recursion and α-conversion, the set of subterms of a term is in general infinite.

We choose to use recursion instead of replication in order to ensure that the set of partitions of a process with respect to composition (up to structural congruence) is always finite, this seems important to ensure decidability of model-checking. Hence, we use the abbreviation

$$!p(n).P \triangleq (\mathbf{rec}\ \mathcal{X})p(n).(P|\mathcal{X})$$

(where $\mathcal{X}$ is fresh in P) to describe a useful form of guarded lazy replication. Since we have $!p(n).P \equiv p(n).(P|!p(n).P)$, rather than $!P \equiv\ !P|P$, $!p(n).P$ is always single-threaded.

Definition 2.3 *For every finite set of names M, we define the relation $\equiv_M$ in processes by*

$$P \equiv_M Q \textit{ if and only if there is } \rho \in \mathbb{R}_M \textit{ such that } \rho(P) \equiv Q$$

N.B.: $\equiv_M$ is a congruence and $\equiv \subseteq \equiv_M$, for every set of names M. We then have

Lemma 2.4 *For every process P,*

1. *For every process Q, it is decidable whether $P \equiv Q$.*
2. *For every finite set of names M and processes P and Q, it is decidable whether $P \equiv_M Q$.*

Proof. (1) By reduction to a canonic form, along the lines of [18]. In the case at hand, we don't have replication, so the proof is much simpler, although not immediate. (2) To test $P \equiv_M Q$ we just need to check if there is some permutation $\rho \in R$ such that $\rho(P) \equiv Q$, where R is the (finite) set of all permutations that only mention names in $(\mathit{fn}(Q) \cup \mathit{fn}(P)) \setminus M$. ■

The behavior of processes is given by reduction:

Definition 2.5 (Reduction) *Reduction is the least binary relation $\rightarrow$ on processes inductively defined as follows.*

$$m\langle n\rangle.Q|m(p).P \rightarrow Q|P\{p\leftarrow n\} \qquad \text{(Red React)}$$
$$Q \rightarrow Q' \Rightarrow P|Q \rightarrow P|Q' \qquad \text{(Red Par)}$$
$$P \rightarrow Q \Rightarrow (\boldsymbol{\nu} n)P \rightarrow (\boldsymbol{\nu} n)Q \qquad \text{(Red Res)}$$
$$P \equiv P',\, P' \rightarrow Q',\, Q' \equiv Q \Rightarrow P \rightarrow Q \qquad \text{(Red Struct)}$$

Reduction captures the computation a process may perform by itself in isolation. In order to observe some of the computation steps a process can perform in conjugation with an appropriate environment, we define a relation of commitment. Commitment is a subset of $\mathcal{P} \times \mathcal{A} \times \mathcal{P}$, where $\mathcal{A}$ is a set of *actions*. We write $P \xrightarrow{\alpha} Q$ to denote that (P, α, Q) belongs to the commitment relation. The actions we consider for commitment are: τ, denoting an autonomous action, $n(m)$, denoting that the process inputs name m on channel n, and $n\langle m\rangle$, denoting that the process sends name m on channel n. Hence, commitment is like the standard relation of labeled transition for the π-calculus as presented *e.g.*, in [30], except that we restrict here the set of labels, not including the "bound output" and the "bound input" actions. Our motivation for introducing commitment is not to provide a basis for a definition of some behavioral process equivalence (e.g., strong bisimulation), as is often the case for labeled transition systems, but rather to give us some additional power to observe the contextual behavior of processes.

Definition 2.6 (Actions, and Commitment) *The set $\mathcal{A}$ of actions is given by*

$$\begin{array}{rll} \alpha & ::= & \text{(Actions)} \\ & \tau & \text{(Silent)} \\ & m\langle n\rangle & \text{(Output)} \\ & m(n) & \text{(Input)} \end{array}$$

Commitment is the least relation $P \xrightarrow{\alpha} Q$ defined as follows:

$$P \rightarrow Q \Rightarrow P \xrightarrow{\tau} Q \qquad \text{(Lab Red Red)}$$
$$m, n \notin \vec{p} \Rightarrow (\boldsymbol{\nu}\vec{p})(m\langle n\rangle.Q|P) \xrightarrow{m\langle n\rangle} (\boldsymbol{\nu}\vec{p})(Q|P) \qquad \text{(Lab Red Output)}$$
$$m, n \notin \vec{p} \Rightarrow (\boldsymbol{\nu}\vec{p})(m(q).Q|P) \xrightarrow{m(n)} (\boldsymbol{\nu}\vec{p})(Q\{q\leftarrow n\}|P) \qquad \text{(Lab Red Input)}$$
$$P \equiv P',\, P' \xrightarrow{\alpha} Q',\, Q' \equiv Q \Rightarrow P \xrightarrow{\alpha} Q \qquad \text{(Lab Red Struct)}$$

Proposition 2.7 (Basic properties of $\xrightarrow{\alpha}$) *For all processes P and Q*

1. *If $P \xrightarrow{\tau} Q$ then $P \rightarrow Q$.*
2. *If $P \rightarrow Q$ then there are processes P', P'', Q', Q'' and names m, q such that $P \equiv (\boldsymbol{\nu}\vec{n})(P'|P'')$, $P' \xrightarrow{m\langle q\rangle} Q'$, $P'' \xrightarrow{m(q)} Q''$, and $Q \equiv (\boldsymbol{\nu}\vec{n})(Q'|Q'')$.*
3. *For every action α, if $P \xrightarrow{\alpha} Q$ then $fn(Q) \subseteq fn(P) \cup na(\alpha)$.*
4. *For all actions α and substitutions θ, if $P \xrightarrow{\alpha} Q$, then $\theta(P) \xrightarrow{\theta(\alpha)} \theta(Q)$.*

x, y, z	$\in$	$\mathcal{V}$ (Name variables)
X, Y, Z	$\in$	$\mathcal{X}$ (Propositional variables)
$\mathcal{X}, \mathcal{Y}, \mathcal{Z}$	$\in$	$\mathcal{X}$ (Process variables)
η	$\in$	$\Lambda \cup \mathcal{V}$ (Names or name variables)
A, B, C	$::=$	
	$\mathbf{T}$	(True)
	$\eta = \eta'$	(Equality)
	$\mathcal{X}$	(Process Variable)
	$\neg A$	(Negation)
	$A \wedge B$	(Conjunction)
	$\mathbf{0}$	(Void)
	$\mathcal{X} : A \vert B$	(Decomposition)
	$A \triangleleft \mathcal{X}$	(Recomposition)
	η®A	(Revelation)
	$A \oslash \eta$	(Hiding)
	$\forall x.A$	(First-order universal quantification)
	$\text{И}x.A$	(Fresh name quantification)
	$\alpha.A$	(Action)
	X	(Propositional variable)
	$\boldsymbol{\nu} X.A$	(Greatest fixpoint)

Figure 1: Logical Formulas.

3 Logic

Formulas, defined in Fig. 1, are built from a supply of names, name variables, and propositional variables. The set of logical operators include propositional, spatial, and temporal operators, first-order quantification, freshness quantification, and recursive formulas [5]. Boolean connectives and name equality have the expected meaning.

Basic spatial connectives correspond to the static operators of the π-calculus. The formula **0** is satisfied by any process structurally congruent to the process **0**. The formula $\mathcal{X} : A|B$ is satisfied by any process that can be split up to structural congruence into a process P satisfying formula A and a process Q satisfying formula B, where all free occurrences of the process variable $\mathcal{X}$ in A denote the process P. Hence, the distinguished occurrence of $\mathcal{X}$ in $\mathcal{X} : A|B$ is binding with scope the formula B. If $\mathcal{X}$ does not occur free in B, we just write $A|B$ for $\mathcal{X} : A|B$.

The formula n®A is satisfied by all processes structurally congruent with a process $(\boldsymbol{\nu} n)P$ where P satisfies A. The formula $A \oslash n$ is satisfied by all processes that satisfy A after hiding name n.

The quantifier $\forall x.A$ denotes quantification over all names in Λ. The formula $\text{И}x.A$ denotes fresh name quantification; a process satisfies $\text{И}x.A$ if for some fresh (in the process and in the formula) name n, it satisfies $A\{x \leftarrow n\}$. A general property of freshness [19, 12, 5] is that if a process satisfies $\text{И}x.A$ it satisfies $A\{x \leftarrow n\}$ for all fresh names n.

The behavioral formula $\alpha.A$ is satisfied by all processes that after committing to action α can become a process that satisfies A, it gives the logic some power to talk about the behavior a process can engage into when placed in an arbitrary environment.

In the formulas $\forall x.A$, $\text{И}x.A$, $\mathcal{X} : B|A$, and $\boldsymbol{\nu} X.A$ the distinguished occurrences of x, $\mathcal{X}$, and X are binding, with scope the formula A. In every formula $\boldsymbol{\nu} X.A$, we require A to be *monotonic in* X, that is, that every free occurrence of the propositional variable X in the formula A is under an even number of negations. The connectives $\vee$ (Disjunction), $\exists$ (Existential Quantification), $\Rightarrow$ (Implication), ⓒ (Occurrence), and $\boldsymbol{\mu} X.A$ (Least fixpoint) are definable as usual (see [5]).

We define on formulas the relation $\equiv_\alpha$ of α-congruence in the standard way, that is, as the least congruence that identifies formulas that just differ on their bound (name, process and propositional) variables. Given a formula A, the sets $fn(A)$ of free names of A, $fv(A)$ of free variables of A, $fprv(A)$ of free process variables of A, and $fpv(A)$ of free propositional variables of A are defined also as expected. We assume defined on formulas the capture avoiding substitution of names/variables for names/variables, and of propositional variables for formulas (written as usual, *e.g.*, $A\{x \leftarrow n\}$, $\theta(A)$, $A\{X \leftarrow B\}$).

The semantics of formulas is defined following quite strictly the approach in [5]: the denotation of any formula is a set of processes that satisfy certain natural closure conditions, namely closure under $\equiv$ and finite support. Intuitively, we expect satisfaction to be closed under structural congruence (so that processes with the same spatial structure satisfy the same formulas). Moreover, any property should just depend on a finite set of relevant names (*cf.*, the free names of a formula); such a set of names is called the *support* of the property.

Definition 3.1 (PSet [5]) *A* property set *is a set of processes* Ψ *such that*

1. *For all Q, if $P \in \Psi$ and $P \equiv Q$ then $Q \in \Psi$.*
2. *There is a finite set of names N such that, for all $n, m \notin N$, if $P \in \Phi$ then $P\{n \leftrightarrow m\} \in \Phi$.*

We denote by $\mathbb{P}$ the set of all Psets. It turns out that every Pset $\Phi \in \mathbb{P}$ has a least support [27, 5], that we denote by $supp(\Phi)$. If Φ is a Pset (supported by N), then $\tau(\Phi) \triangleq \{\tau(P) \mid P \in \Phi\}$ is also a Pset (supported by $\tau(N)$). For any set of processes S and finite set of names N, we can define a Pset $Close(S, N) \in \mathbb{P}$ by $Close(S, N) \triangleq \{\tau(P) \mid \tau \in \mathbb{R}_N \text{ and } P \in S\}$. We can verify that $Close(S, N)$ contains S and is supported by N.

Hence, the denotation of a formula A is given by a Pset $[\![A]\!] \in \mathbb{P}$. Since a formula A may contain free occurrences of propositional and process variables, its denotation depends on the denotation of such variables, which is given by a valuation. A valuation v is a finite mapping assigning each propositional variable in its domain a Pset in $\mathbb{P}$, and each process variable in its domain a process in $\mathcal{P}$. We use the abbreviation $P \models_v A$ for $P \in [\![A]\!]_v$.

We extend the application of transpositions to valuations as follows: when v is a valuation, let $\tau(v)$ be the valuation with the same domain as v and defined by $\tau(v)(\xi) \triangleq \tau(v(\xi))$, for all $\xi \in \chi \cup \mathcal{X}$ in the domain of v.

We also need a semantic characterization for the "free" names of a formula A under a valuation as follows [5]: if A is a formula, and v a valuation for A, we define the set $fn^v(A)$ of *free names of A under v* by

$$fn^v(A) \triangleq fn(A) \cup \bigcup\{supp(v(X)) \mid X \in fpv(A)\} \cup \bigcup\{fn(v(\mathcal{X})) \mid \mathcal{X} \in fprv(A)\}$$

Intuitively, $fn^v(A)$ is basically $fn(A)$ except that we take $fn(X) = supp(v(X))$ for any $X \in fpv(A)$, and $fn(\mathcal{X}) = fn(v(\mathcal{X}))$ for any $\mathcal{X} \in fprv(A)$, hence $fn^v(A) = fn(A)$ for

$$
\begin{array}{lcl}
[\![\mathbf{T}]\!]_v & \triangleq & \mathcal{P} \\
[\![n = m]\!]_v & \triangleq & \textit{if } n = m \textit{ then } \mathcal{P} \textit{ else } \emptyset \\
[\![x]\!]_v & \triangleq & \{v(x)\} \\
[\![\neg A]\!]_v & \triangleq & \mathcal{P} \setminus [\![A]\!]_v \\
[\![A \wedge B]\!]_v & \triangleq & [\![A]\!]_v \cap [\![B]\!]_v \\
[\![\mathbf{0}]\!]_v & \triangleq & \{P \mid P \equiv \mathbf{0}\} \\
[\![x : A|B]\!]_v & \triangleq & \{P \mid \textit{Exists } Q, R.\ P \equiv Q|R \textit{ and } Q \in [\![A]\!]_v \textit{ and } R \in [\![B]\!]_{v[x \leftarrow Q]}\} \\
[\![A \triangleleft x]\!]_v & \triangleq & \{P \mid P|v(x) \in [\![A]\!]_v\} \\
[\![n \circledR A]\!]_v & \triangleq & \{P \mid \textit{Exists } Q.\ P \equiv (\boldsymbol{\nu} n)Q \textit{ and } Q \in [\![A]\!]_v\} \\
[\![A \oslash n]\!]_v & \triangleq & \{P \mid (\boldsymbol{\nu} n)P \in [\![A]\!]_v\} \\
[\![\forall x.A]\!]_v & \triangleq & \bigcap_{n \in \Lambda} [\![A\{x \leftarrow n\}]\!]_v \\
[\![\text{И} x.A]\!]_v & \triangleq & \bigcup_{n \notin \mathit{fn}^v(A)} ([\![A\{x \leftarrow n\}]\!]_v \setminus \{P \mid n \in \mathit{fn}(P)\}) \\
[\![\alpha.A]\!]_v & \triangleq & \{P \mid \textit{Exists } Q.\ P \xrightarrow{\alpha} Q \textit{ and } Q \in [\![A]\!]_v\} \\
[\![X]\!]_v & \triangleq & v(X) \\
[\![\boldsymbol{\nu} X.A]\!]_v & \triangleq & \bigcup\{\Psi \in \mathbb{P} \mid \Psi \subseteq [\![A]\!]_{v[X \leftarrow \Psi]}\}
\end{array}
$$

Figure 2: Denotation of formulas.

closed A. The set $\mathit{fn}^v(A)$ is useful in the definition of the semantics of the fresh name quantifier, where the quantification witness must be fresh with respect to the property set denoted by a formula that contain in general free occurrences of propositional (and process) variables.

The semantics of formulas is inductively defined as shown in Fig. 2. The denotation mapping $[\![-]\!]_v$ satisfies certain fundamental properties, collected in the next theorem.

Theorem 3.2 *For all formulas A and B, and valuations v*

1. *If $A \equiv_\alpha B$ then $[\![A]\!]_v = [\![B]\!]_v$.*
2. *$[\![A]\!]_v \in \mathbb{P}$ with $\mathit{supp}([\![A]\!]_v) \subseteq \mathit{fn}^v(A)$.*
3. (Equivariance) *For all transpositions τ, $\tau([\![A]\!]_v) = [\![\tau(A)]\!]_{\tau(v)}$.*
4. (Gabbay-Pitts Property) *Let $M = \mathit{fn}^v(A) \cup \mathit{fn}(P)$.*
   If $P \in [\![A\{x \leftarrow p\}]\!]_v$ for some $p \notin M$, then $P \in [\![A\{x \leftarrow p\}]\!]_v$ for all $p \notin M$.

Proof. (1) Induction on the structure of A. (2-3) By (mutual) induction on the structure of A. Similar to [5]. In the appendix, we show the cases not present in [5]. (4) A direct consequence of (2) and (3). ∎

We also have:

Proposition 3.3 *Let A be a formula monotonic in X and v a valuation for the formula $\boldsymbol{\nu} X.A$. Let ϕ be the mapping $\mathbb{P} \to \mathbb{P}$ defined by $\phi(s) \triangleq [\![A]\!]_{v[X \leftarrow s]}$.*

1. *ϕ is monotonic in $\mathbb{P}$.*
2. *ϕ has a greatest fixpoint (written $\boldsymbol{\nu} s.\phi(s)$ or $Gfix(\phi)$) and $[\![\nu X.A]\!]_v = GFix(\phi)$.*
3. *For every $\Phi \in \mathbb{P}$, $\Phi \subseteq \boldsymbol{\nu} s.\phi(s)$ if and only if $\Phi \subseteq \phi(\boldsymbol{\nu} s.(\Phi \cup \phi(s)))$.*

Proof. (1) Induction on the structure of A (see [5]). (2) Along standard lines. (3) This is a standard result (the "reduction lemma") [23, 31]. ■

4 Model Checking

Our model-checking algorithm is specified by the boolean-valued procedure $MC(P, v, A)$ presented in Figure 3. In every procedure call $MC(P, v, A)$, P is a process, A is a formula, and v is a *syntactic valuation*. The algorithm relies on structural congruence tests at several points (namely, in the case for **0**, for a process variable $\mathcal{X}$, and for a propositional variable X). The boolean connectives are handled as expected. Spatial and behavioral connectives are handled by the set of auxiliary procedures $Comp(-)$, $Res(-,-)$ and $Red(-,-)$. The purpose of these procedures is to decompose processes up to structural congruence and compute the set of labeled reductions that may arise from a given process. We can show

Lemma 4.1 *For any process P we have*

1. *A finite set $Comp(P) \subseteq \mathcal{P} \times \mathcal{P}$ can be effectively constructed such that:*
   (a) *For all Q, R such that $P \equiv Q|R$, there is $(Q', P') \in Comp(P)$ such that $Q \equiv Q'$ and $P \equiv P'$.*
   (b) *For all $(Q', R') \in Comp(P)$ we have $P \equiv Q'|R'$.*
2. *For any name n a finite set $Res(n, P) \subseteq \Lambda \times \mathcal{P}$ can be effectively constructed such that:*
   (a) *For all Q such that $P \equiv (\boldsymbol{\nu} n)Q$, there is $Q' \in Res(n, P)$ such that $Q \equiv Q'$.*
   (b) *If $Q' \in Res(n, P)$ then $P \equiv (\boldsymbol{\nu} n)Q'$.*
3. *For any action α, a finite set $Red(\alpha, P) \subseteq \mathcal{P}$ can be effectively constructed such that:*
   (a) *For all Q such that $P \xrightarrow{\alpha} Q$, there is $Q' \in Red(\alpha, P)$ such that $Q \equiv Q'$.*
   (b) *If $Q \in Red(\alpha, P)$ then $P \xrightarrow{\alpha} Q$.*

Similar results for calculi with replication (the π-calculus and the ambient calculus) have been developed in [18, 15]. However, the property in Lemma 4.1(1a) does not hold in the form stated here in process calculi with replication and the structural congruence axiom $!P \equiv P|!P$ (*cf.* [18]).

Remark 4.2 We have $Res(n, P) = \emptyset$ if and only if $n \in fn(P)$.

The model-checker cases for the freshness quantifier and the universal quantifier requires the generation of freshness witnesses (fresh names). Instead of attempting to pre-compute a bound to the set of freshness witnesses for every process and formula to submit to the model-checker (*cf.,* the bound output modality in the model-checker of [16]), we rely on the Gabbay-Pitts property (Theorem 3.2(4)), and in each case pick an arbitrary name out of the support of the denotation of the formula to be checked. By Theorem 3.2(2), we know that such support can be approximated by the free names of the formula, where we consider for the free names of a propositional variable the free names of the recursive formula that introduces its binding occurrence. To that end, we introduce the auxiliary function $fs^v(A)$, that computes (an approximation to) a support given a formula A and a syntactic valuation v (defined below). We then assume given a choice function that assigns to every finite set of names M a name $fresh(M) \notin M$: we can use any function meeting this specification. In fact, no property of the model-checker (*e.g.*, termination) requires fresh names to be generated according to some fixed procedure: this means that the essential use of $\mathit{fs}^v(A)$ appears in the auxiliary function $In(-,-,-)$.

Syntactic valuations are finite counterparts to the (semantic) valuations defined in Section 3. A syntactic valuation is essentially a mapping that assigns to each propositional variable in its domain a pair (S, A), where S is a finite set of processes and A is a fixpoint formula, and to each process variable in its domain a process. Intuitively, if v is a syntactic valuation and $v(X) = (S, A)$ then S is a finite approximation to the denotation of the formula A. More precisely,

Definition 4.3 (Syntactic Valuation) *A syntactic valuation v is a mapping from a finite set $\{X_1, \ldots, X_n, \mathcal{X}_1, \ldots, \mathcal{X}_m\}$ of propositional and process variables such that:*

$$\begin{array}{ll} v(X_i) = (S_i, \boldsymbol{\nu} X_i.A_i) & \textit{for all } i = 1, \ldots, n \\ v(\mathcal{X}_i) \in \mathcal{P} & \textit{for all } i = 1, \ldots, m \end{array}$$

where each S_i is a finite sets of processes, and each $\boldsymbol{\nu} X_i.A_i$ is a formula with $\mathit{fpv}(A_i) \subseteq \{X_1, \ldots, X_{i-1}\}$ and $\mathit{fprv}(A_i) \subseteq \{\mathcal{X}_1, \ldots, \mathcal{X}_m\}$ for all $i = 1, \ldots, n$.

We say that *v is a syntactic valuation for A* if v is a syntactic valuation and $\mathfrak{D}(v) \subseteq \mathit{fpv}(A) \cup \mathit{fprv}(A)$. We define for any syntactic valuation v for A the set

$$\begin{array}{rcl} \mathit{fs}^v(A) & \triangleq & \mathit{fn}(A) \cup \\ & & \bigcup\{\mathit{fs}^v(B) \mid X \in \mathit{fpv}(A) \text{ and } v(X) = (S, B)\} \cup \\ & & \bigcup\{\mathit{fn}(v(\mathcal{X})) \mid \mathcal{X} \in \mathit{fprv}(A)\} \end{array}$$

of *free names of A under v*.

When v is a valuation, $X \notin \mathfrak{D}(v)$, S is a finite set of processes, and $\mathit{fpv}(A) \subseteq \mathfrak{D}(v)$ we write $v[X \leftarrow (S, A)]$ for the extension (not the update) of v with the additional binding $[X \leftarrow (S, A)]$.

We also use the notation $v(X + P)$ to denote the valuation that results from v by adding the process P to the set component of $v(X)$, *e.g.*, if v is the valuation $w[X \leftarrow (S, A)]w'$ then $v(X + P)$ is the valuation $w[X \leftarrow (S \cup \{P\}, A)]w'$.

Fixpoint formulas are handled by appealing to the standard reduction lemma (Proposition 3.3) (attributed to Kozen and Winskel). The reduction lemma suggests a progressive unfolding strategy for recursive formulas that used in most model-checkers for μ-calculus based process logics (*cf.*, the tagged-fixpoints of [1]). The main technical difference between the treatment of fixpoints in the present algorithm and the one adopted in alternative proposals concerns the interaction of freshness, fresh name generation, and recursion. Here, we compute an approximation to the finite support of the denotation of a fixpoint formula (given by $\mathit{fs}^v(A)$), and use later on that information to detect termination of an unfolding sequence, relying on the fact that if a process P belongs to a Pset Ψ with $\mathit{supp}(\Psi) \subseteq M$, then $\rho(P) \in \Psi$ for all permutations ρ that fix M. The role of the boolean-valued procedure $In(-,-,-)$ shown at the bottom of Figure 3 is precisely to check this condition. This approach seems conceptually simpler than related proposals handling recursion and freshness when model-checking the π-calculus (for instance, [16] requires the introduction of higher-order propositional variables, and recursive formulas to be name-closed) and

$$
\begin{array}{lcl}
MC(P,v,\mathbf{T}) & \triangleq & \textbf{true} \\[1ex]
MC(P,v,n=m) & \triangleq & Test(m=n) \\[1ex]
MC(P,v,\mathcal{X}) & \triangleq & Test(P \equiv v(\mathcal{X})) \\[1ex]
MC(P,v,\neg A) & \triangleq & \textbf{not } MC(P,v,A) \\[1ex]
MC(P,v,A \wedge B) & \triangleq & MC(P,v,A) \textbf{ and } MC(P,v,B) \\[1ex]
MC(P,v,\mathbf{0}) & \triangleq & Test(P \equiv \mathbf{0}) \\[1ex]
MC(P,v,\mathcal{X} : A|B) & \triangleq & \textbf{Exists } Q, R.(Q,R) \in Comp(P) \\
 & & \quad \textbf{and } MC(Q,v,A) \textbf{ and } MC(R, v[\mathcal{X} \leftarrow Q], B) \\[1ex]
MC(P,v,A \triangleleft \mathcal{X}) & \triangleq & MC(v(\mathcal{X})|P,v,A) \\[1ex]
MC(P,v,n \circledR A) & \triangleq & \textbf{Exists } Q.\ Q \in Res(n,P) \textbf{ and } MC(Q,v,A) \\[1ex]
MC(P,v,A \oslash n) & \triangleq & MC((\boldsymbol{\nu} n)P,v,A) \\[1ex]
MC(P,v,\alpha.A) & \triangleq & \textbf{Exists } Q.\ Q \in Red(\alpha,P) \textbf{ and } MC(Q,v,A) \\[1ex]
MC(P,v,\forall x.A) & \triangleq & \textbf{All } n \in \mathit{fn}(P) \cup \mathit{fs}^v(A).MC(P,v,A\{x \leftarrow n\}) \\
 & & \quad \textbf{and } MC(P,v,A\{x \leftarrow fresh(\mathit{fn}(P) \cup \mathit{fs}^v(A))\}) \\[1ex]
MC(P,v,\text{И} x.A) & \triangleq & MC(P,v,A\{x \leftarrow fresh(\mathit{fn}(P) \cup \mathit{fs}^v(A))\}) \\[1ex]
MC(P,v,X) & \triangleq & \textbf{let } (S, \boldsymbol{\nu} X.A) = v(X) \textbf{ in} \\
 & & \quad \textbf{if } In(P,v,X) \textbf{ then true} \\
 & & \quad\ \textbf{else } MC(P, v(X+P), A) \\[1ex]
MC(P,v,\boldsymbol{\nu} X.A) & \triangleq & MC(P, v[X \leftarrow (\{P\}, \boldsymbol{\nu} X.A)], A) \\[1ex]
In(P,v,X) & \triangleq & \textbf{let } (S,A) = v(X) \\
 & & \quad \textbf{in Exists } Q \in S \textbf{ and } Test(P \equiv_{\mathit{fs}^v(A)} Q)
\end{array}
$$

Figure 3: Model-checking algorithm

extends smoothly to cope with the spatial connectives, in particular those related to name restriction. In our setting, the extension of the logic and model-checker to allow for name parameters in recursive formulas can be accomplished in an orthogonal way, as a natural development of the basic techniques.

We now look at a simple example of a run of the model-checking algorithm.

Example 4.4 Consider the formula

$$A \triangleq \boldsymbol{\nu} X.\mathsf{H}y.\exists x.(p\langle x, y\rangle.\mathbf{0}|[p(x, y)]X)$$

and the processes

$$\begin{array}{rcl} R & \triangleq & !p(a, b).(\boldsymbol{\nu} q)p\langle b, q\rangle \\ P & \triangleq & (\boldsymbol{\nu} n)p\langle m, n\rangle|R \end{array}$$

Intuitively, a process satisfies A if it can be split into a process that can send $p\langle x, y\rangle$ and then vanish; and a process that after consuming a message $p\langle x, y\rangle$ is bound to evolve to a process that (again) satisfies A. The name y in the message is private to every such pair of processes, while x is some unspecified name (existentially quantified in the scope of the restriction, so x may be y or may be not).

We verify $P \in [\![A]\!]$ by model-checking, exemplifying the termination of the unfolding sequence for a recursive formula using a permutation. We have in succession

$$\begin{array}{l} MC(P, \emptyset, A) = \\ MC((\boldsymbol{\nu} n)p\langle m, n\rangle|R, v, \mathsf{H}y.\exists x.(p\langle x, y\rangle.\mathbf{0}|[p(x, y)]X)) = \\ MC(p\langle m, f_1\rangle|R, v, \exists x.(p\langle x, f_1\rangle.\mathbf{0}|[p(x, f_1)]X)) = \\ MC(p\langle m, f_1\rangle|R, v, p\langle m, f_1\rangle.\mathbf{0}|[p(m, f_1)]X) = \\ MC(p\langle m, f_1\rangle, v, p\langle m, f_1\rangle.\mathbf{0}) = \textbf{true and} \\ MC(R, v, [p(m, f_1)]X) = \\ MC((\boldsymbol{\nu} q)p\langle f_1, q\rangle|R, v, X) = \textbf{true} \\ \\ \text{where } v = [X \leftarrow (\{(\boldsymbol{\nu} n)p\langle m, n\rangle|R\}, A)] \end{array}$$

The last step is justified by $fs^v(B) = \{p\}$ and $(\boldsymbol{\nu} q)p\langle f_1, q\rangle|R \equiv_{\{p\}} (\boldsymbol{\nu} n)p\langle m, n\rangle$ (just consider the transposition $\{f_1 \leftrightarrow m\}$).

The remainder of this section is devoted to the statement and proof of correctness results for our model-checker. We start by introducing some auxiliary concepts. For every syntactic valuation v, we can define a corresponding (semantic) valuation $v^\star$.

Definition 4.5 (Interpretation of syntactic valuations) *For every syntactic valuation v, we inductively define a valuation $v^\star$ as follows:*

$$\begin{array}{lcl} \emptyset^\star & \triangleq & \emptyset \\ w[X \leftarrow P] & \triangleq & w^\star[X \leftarrow P] \\ w[X \leftarrow (S, \boldsymbol{\nu} X.A)]^\star & \triangleq & w^\star[X \leftarrow \Psi] \end{array}$$

where $\Psi \triangleq Gfix(\lambda\Psi.S^\star \cup [\![A]\!]_{w^\star[X \leftarrow \Psi]})$ *and* $S^\star \triangleq Close(S, fs^v(\boldsymbol{\nu} X.A))$.

We can then verify the following correspondence properties.

Lemma 4.6 *For every syntactic valuation v, we have*

1. *If* $v(X) = (S, A)$*, then* $\mathit{supp}(v^\star(X)) \subseteq \mathit{fs}^v(A)$.
2. *For every formula* A*, we have* $\mathit{fn}^{v^\star}(A) \subseteq \mathit{fs}^v(A)$.

Proof. (1) By induction on the index i of the propositional variable $X = X_i$ in the valuation v. Pick $P \in v^\star(X) = Gfix(\phi)$ where $\phi(s) \triangleq S^\star \cup [\![A]\!]_{w^\star[X \leftarrow s]}$. Then there is $\Psi \in \mathbb{P}$ such that $\Psi \subseteq \phi(\Psi)$ and $P \in \Psi$. Pick τ fixing $\mathit{fs}^v(A)$. Then $\tau(\Psi) \subseteq \tau(\phi(\Psi))$. By Theorem 3.2(3), we have $\tau(\phi(\Psi)) = S^\star \cup [\![\tau(A)]\!]_{\tau(w^\star)[X \leftarrow \tau(\Psi)]} = S^\star \cup [\![A]\!]_{\tau(w^\star)[X \leftarrow \tau(\Psi)]}$ (note that $\tau(S^\star) = S^\star$). Write u for the restriction of w to the free propositional variables in A. Then $[\![A]\!]_{\tau(w^\star)[X \leftarrow \tau(\Psi)]} = [\![A]\!]_{\tau(u^\star)[X \leftarrow \tau(\Psi)]}$. By induction hypothesis, we have that $\mathit{supp}(v^\star(X)) \subseteq \mathit{fs}^v(A_i) \subseteq \mathit{fs}^v(A)$, for all $X_i \in \mathit{fpv}(A)$ such that $X \in \mathfrak{D}(u)$. Hence $\tau(u^\star(X)) = u^\star(X)$, for all $X \in \mathfrak{D}(u)$. We then conclude $[\![A]\!]_{\tau(u^\star)[X \leftarrow \tau(\Psi)]} = [\![A]\!]_{u^\star[X \leftarrow \tau(\Psi)]} = [\![A]\!]_{w^\star[X \leftarrow \tau(\Psi)]}$.

We obtain $\tau(\Psi) \subseteq S^\star \cup [\![A]\!]_{w^\star[X \leftarrow \tau(\Psi)]}$, that is $\tau(\Psi) \subseteq \phi(\tau(\Psi))$. Hence $\tau(\Psi) \subseteq Gfix(\phi)$, and therefore $\tau(P) \in v^\star(X)$.

(2) A direct consequence of (1). Let $X \in \mathit{fpv}(A)$, and $v(X) = (S, B)$. Then $\mathit{fn}^{v^\star}(X) = \mathit{supp}(v^\star(X)) \subseteq \mathit{fs}^v(B)$. But $\mathit{fs}^v(B) = \mathit{fs}^v(X)$, hence $\mathit{fn}^{v^\star}(X) \subseteq \mathit{fs}^v(X)$, for all $X \in \mathit{fpv}(A)$. We conclude $\mathit{fn}^{v^\star}(A) \subseteq \mathit{fs}^v(A)$. ∎

We can now state and proof

Proposition 4.7 (Soundness) *For every process* P*, formula* A *and syntactic valuation* v *for* A *we have:*

1. *If* $MC(P, v, A) =$ **true** *then* $P \in [\![A]\!]_{v^\star}$.
2. *If* $MC(P, v, A) =$ **false** *then* $P \notin [\![A]\!]_{v^\star}$.

Proof. See appendix. ∎

We now show completeness of the model-checking algorithm: to obtain decidability we need to impose some finiteness conditions. Namely, we restrict model-checking to the class of bounded processes and linear formulas. Intuitively, a process is bounded if the set of processes reachable after an arbitrary sequence of spatial or behavioral observations if finite up to finitely supported name permutations. As we shall see, the class of bounded processes subsumes the class of finite-control processes [16]. We first define reachability:

Definition 4.8 (Reachability) *For every process* P *we define the set of processes* $Reach(P)$ *as follows:*

$P \in Reach(P)$
$P' \in Reach(P),\ (Q, R) \in Comp(P') \Rightarrow Q \in Reach(P)$
$P' \in Reach(P),\ (Q, R) \in Comp(P') \Rightarrow R \in Reach(P)$
$P' \in Reach(P),\ (Q, R) \in Comp(P'),\ R' \in Reach(R) \Rightarrow Q|R' \in Reach(P)$
$P' \in Reach(P),\ \mathit{Exists}\ n.\ Q \in Res(n, P') \Rightarrow Q \in Reach(P)$
$P' \in Reach(P) \Rightarrow (\boldsymbol{\nu} n)P' \in Reach(P)$
$P' \in Reach(P),\ \mathit{Exists}\ \alpha.\ Q \in Red(\alpha, P') \Rightarrow Q \in Reach(P)$

Definition 4.9 (Bounded) *A process* P *is* bounded *if for every finite set of names* M *the set (of equivalence classes)* $Reach(P)/{\equiv_M}$ *is finite.*

A formula is linear if for all subformulas of the form $A \triangleleft \mathcal{X}$, the process variable $\mathcal{X}$ does not occur free in A. For instance, all uses of $\triangleleft$ discussed in Section 1.2 are linear. We then have:

Proposition 4.10 (Completeness) *Let Q be a bounded process and F a linear formula such that $Q \in [\![F]\!]$. Then $MC(Q, \emptyset, F) =$* **true**.

Proof. We just need to show that the model-checking algorithm always terminates with a result, that such result is correct results from Proposition 4.7.

Since Q is bounded, we have that for every finite set of names M the number of equivalence classes $Reach(Q)/\equiv_M$ is finite.

We first verify that in every recursive call $MC(P, v, A)$ of the algorithm with initial call $MC(Q, \emptyset, F)$, we always have $P \in Reach(Q)$. To check this claim, it suffices to note that for every such recursive call we have $v(\mathcal{X}_1)|\cdots|v(\mathcal{X}_n)|P \in Reach(Q)$ where $\mathcal{X}_1, \ldots, \mathcal{X}_n$ are all process variables in the domain of v that occur free in A.

The proof then proceeds by induction on a measure $|MC(P, v, A)|_c$ defined on the set of calls of the model-checking algorithm. The measure depends on a auxiliary mapping c that assigns a natural number $c(X)$ to each propositional variable X in $\mathfrak{D}(v)$, where we require the invariant $c(X) = \#(Reach(Q)/\equiv_{fs^v(A)}) - \#S$ whenever $v(X) = (S, A)$, for every $X \in \mathfrak{D}(v)$. We denote by $c(X-)$ the valuation that results from c by decrementing the value assigned by c to X. The measure $|MC(P, v, A)|_c$ is defined as follows:

$$
\begin{array}{lcll}
|MC(P, v, \mathbf{T})|_c & \triangleq & 1 & \\
|MC(P, v, n = m)|_c & \triangleq & 1 & \\
|MC(P, v, \mathcal{X})|_c & \triangleq & 1 & \\
|MC(P, v, A \dagger B|_c & \triangleq & 1 + |MC(P, v, A)|_c + |MC(P, v, B)|_c & \\
|MC(P, v, \ddagger A)|_c & \triangleq & 1 + |MC(P, v, A)|_c & \\
|MC(P, v, \nu X.A)|_c & \triangleq & 1 + |MC(P, v[X \leftarrow (\emptyset, A)]|_{c[X \leftarrow \#(Reach(Q)/\equiv_{fs^v(A)})]} & \\
|MC(P, v, X)|_c & \triangleq & 1 & (\textit{if } c(X) = 0) \\
|MC(P, v, X)|_c & \triangleq & 1 + |MC(P, v(X + P), A)|_{c(X-)} & (\textit{if } c(X) > 0)
\end{array}
$$

where we denote by † any binary connective and by ‡ any unary connective. Note that $|MC(P, v, A)|_c$ is well-defined, since each propositional variable X can only be unfolded into its defining formula A the finite number of times $\#(Reach(Q)/\equiv_{fs^v(A)})$.

To conclude, it is enough to verify that the measure strictly decreases for every recursive call of the algorithm. We show in detail the case of the propositional variable, the case of the fixpoint is handled in a similar way, and the remaining cases are easy to verify.

So, consider the case $MC(P, v, X)$ that reduces to $MC(P, v(X + P), A)$, assuming $v = v'[X \leftarrow (S, A)]v''$. Then $w = v(X + P) = v'[X \leftarrow (S \cup \{P\}, A)]v''$. Note that we must have $X \in fpv(A)$, otherwise the call $MC(P, v, X)$ would have not been reached. Note that $c(X) > 0$, since one one hand $P \in Reach(Q)$, and on the other hand there is no representative in the set S for the equivalence class (in $Reach(Q)/\equiv_{fs^v(A)}$) of Q (since we must have $In(P, v, X) =$ **false**). Now, we can verify that $|MC(P, v, X)|_c = 1 + |MC(P, v(X + P), A)|_{c(X-)} > |MC(P, v(X + P), A)|_{c(X-)}$. ■

A consequence of Proposition 4.10, after noting that all tests in the model-checking algorithm are decidable, and that the number and structure of recursive calls associated to each call of the model-checking algorithm is finite and decidable in all cases, we have

Corollary 4.11 (Decidability) *The model-checking problem for bounded processes and linear formulas is decidable.*

For the record, we also state the following

Proposition 4.12 *Model-checking is decidable on recursion-free formulas.*

In what follows, we define a useful class of bounded processes, that corresponds to the standard finite-control subset of the π-calculus [16]; other classes of bounded processes can possibly be characterized, by considering appropriate syntactic or type-driven conditions. Note that in our setting, due to spatial reachability, the condition that all computations of process P terminate is not enough to ensure boundedness: a deadlocked process can in general have spatial components which many not be bounded when considered in isolation, *e.g.*, the process $(\boldsymbol{\nu} n)(\mathbf{rec}\ \mathcal{X})n\langle n\rangle.(\mathcal{X}|\mathcal{X})$ is not bounded in the sense of Definition 4.9.

Definition 4.13 *The set $\mathcal{F}$ of finite-control processes is given by the following abstract syntax*

m, n, p	$\in$	Λ (Names)
T, U	$::=$	(Threads)
	$\mathbf{0}$	(Void)
	$(\boldsymbol{\nu} n)T$	(Restriction)
	$m\langle n\rangle.T$	(Output)
	$m(n).T$	(Input)
	$\mathcal{X}$	(Process variable)
	$(\mathbf{rec}\ \mathcal{X})T$	(Recursive process)
P, Q, R	$::=$	(Finite-Control Processes)
	T	(Thread)
	$T\|U$	(Composition)
	$(\boldsymbol{\nu} n)T$	(Restriction)

Proposition 4.14 *Let P be a finite-control process. Then, P is bounded.*

Proof. Any finite-control process P can be written (up to structural congruence)

$$P \equiv (\boldsymbol{\nu} n_1)\cdots(\boldsymbol{\nu} n_p)(T_1|\ldots|T_m)$$

where each T_i is a thread. We define

$$Threads(P) \triangleq \{T \mid T \text{ is a thread and } T = C[P] \text{ for some process context } C[]\}$$

Due to α-congruence, $Threads(P)$ is in general infinite, however $Threads(P)/\equiv_M$ is finite for every finite set of names M. Moreover, note that if P is finite-control and $Q \in Reach(P)$ then Q is also finite-control. We also have that if P is finite-control, then there is a fixed number N_T such that for all $Q \in Reach(P)$ the number of parallel threads of Q is smaller by N_T. More precisely, by induction on the definition of reachability, we can verify that for every process $R \in Reach(P)$ we can write $R \equiv (\boldsymbol{\nu} n_1)\cdots(\boldsymbol{\nu} n_p)(T_1|\ldots|T_m)$, with $m < N_T$, where any of the restricted names n_i has a free occurrence in some of the threads T_j; this also implies that the number p of relevant restricted names in any such a process must also be bonded by some fixed N_R. We can also verify that any of such T_i must result from some thread initially present in the process P by the action of some name substitution. That is, for all T_i there is some P_j in $Threads(P)$ and substitution σ such that $T_i \equiv \sigma(P_j)$. Since $Threads(P)/\equiv_M$ is finite for any set of names M, we have that $Reach(P)/\equiv_M$ is also finite for any finite M. ∎

5 Remarks on Expressiveness

Spatial properties reflect an enhanced observational power when compared with behavioral properties expressible in Hennessy-Milner-like process logics. However, we expect spatial properties to be invariant under a notion of structural identity, typically structural identity should be close to structural congruence [21, 29]. In this section, we discuss the relation between some fragments of this logic and some process equivalences.

As usual, the equivalence relation $\stackrel{L}{=}$ induced by the logic on a set of processes is given by letting $P \stackrel{L}{=} Q$ if and only if for all closed formulas A, $P \models A$ if and only if $Q \models A$. We start by introducing

Definition 5.1 (Characteristic Formulas) *For each process P we define a formula $|P|$ as follows:*

$$\begin{array}{lcl}
\mathbf{1} & \triangleq & \neg(\neg\mathbf{0}|\neg\mathbf{0}) \wedge \neg\mathbf{0} \\
NoRes & \triangleq & \neg\mathsf{H}x \copyright x \\
|\mathbf{0}| & \triangleq & \mathbf{0} \\
|P|Q| & \triangleq & |P|||Q| \\
|(\boldsymbol{\nu} n)P| & \triangleq & \mathsf{H}x.(|P|\{n\leftarrow x\}) \\
|m\langle n\rangle.P| & \triangleq & \mathbf{1} \wedge NoRes \wedge m\langle n\rangle|P| \\
|m(n).P| & \triangleq & \mathbf{1} \wedge NoRes \wedge \forall x.m(x)|P|\{n\leftarrow x\} \\
|X| & \triangleq & X \\
|(\mathbf{rec}\ X)P| & \triangleq & (\boldsymbol{\nu} X)|P|
\end{array}$$

N.B.: We have $fn(|P|) = fn(P)$, and $|P|\{\eta\leftarrow\eta'\} \equiv_\alpha |P\{\eta\leftarrow\eta'\}|$.

Proposition 5.2 *We have*

1. *For all processes P, $P \in [\![|P|]\!]$.*
2. *For all recursion-free processes P and Q, if $Q \in [\![|P|]\!]$ then $Q \equiv P$.*

Proof. By induction on the structure of P (see Appendix). ∎

We can then show

Proposition 5.3 *The equivalence relation induced by the logic on the set of all recursion-free processes coincides with structural congruence.*

Proof. Assume $P \stackrel{L}{=} Q$. Then for all closed formulas A, we have that $P \models A$ if and only if $Q \models A$. Let $A = |P|$. Since by Proposition 5.2(1) $P \models |P|$, we also have $Q \models |P|$. But then $Q \equiv P$, by Proposition 5.2(2). The inverse inclusion follows directly from the closure of satisfaction under structural congruence Theorem 3.2(2). ∎

We have

Proposition 5.4 *The equivalence relation induced by the logic on the set of all processes strictly includes structural congruence.*

Proof. That $\equiv$ is included in $P \stackrel{L}{=} Q$, follows from the fact that satisfaction is closed under $\equiv$ (Theorem 3.2(2)). To verify that the inclusion is strict, we may prove (for example) that there is no formula distinguishing between the processes $R_1 \triangleq (\mathbf{rec}\ X)n\langle n\rangle.X$, $R_2 \triangleq (\mathbf{rec}\ X)n\langle n\rangle.n\langle n\rangle.X$ and $R_3 \triangleq n\langle n\rangle.(\mathbf{rec}\ X)n\langle n\rangle.n\langle n\rangle.X$ (note that $R_i \not\equiv R_j$ if $i \neq j$, for all $i, j = 1, 2, 3$). Let V_R be the set of all valuations v such that for all X, if $R_i \in v(X)$ for some $i = 1, 2, 3$ then $R_j \in v(X)$ for all $j = 1, 2, 3$ We show that for all formulas A and valuations $v \in V_R$, if $R_i \models_v A$ for some $i = 1, 2, 3$ then $R_j \models_v A$ for all $j = 1, 2, 3$. The proof is the concluded by induction on the structure of the formula A. ■

We also have

Proposition 5.5 *The equivalence relation induced by the logic on the set of all processes is strictly included in early strong bisimulation.*

Proof. The inclusion follows from the fact that the modalities introduced in [26] are expressible in the logic as follows:

$$\begin{array}{lll} \langle\alpha\rangle A & \triangleq & \alpha.A \qquad \text{(for } \alpha = \tau, x\langle y\rangle, x(y)) \\ \langle\overline{x}(y)\rangle A & \triangleq & \mathsf{H}y.x\langle y\rangle.A \\ \langle x(y)\rangle^{\mathrm{E}} A & \triangleq & \forall y.x(y).A \end{array}$$

That the inclusion is strict is immediate. ■

Remark 5.6 Intuitively, the processes R_1, R_2 and R_3 mentioned in the proof of Proposition 5.4 all have the same observable behavior (all of them generate an infinite sequence of $n\langle n\rangle$ actions) and the same observable structure (all of them are single-threaded, with no hidden names, and have the same set of free names). However, we cannot show *e.g.*, $R_1 \equiv R_2$, from the axioms of structural congruence presented in Definition 2.2. To get an equational characterisation of structural identity as given by $\stackrel{L}{=}$, we need to strengthen structural congruence with additional principles. In the case at hand, it can be shown that if one adds the following principle

$$P \equiv C[P] \Rightarrow P \equiv (\mathbf{rec}\ X)C[X] \qquad \text{(Struct Rec Solve)}$$

(where $C[]$ is a syntactic process context such that the hole $[]$ occurs guarded in $C[]$), an exact match between $\equiv$ and $\stackrel{L}{=}$ is then obtained.

6 Conclusion

We studied model-checking in a spatial logic for the π-calculus. The logic can be seen as a fragment of the logic introduced in [5], and is based on a small set of space-time observation modalities. The model-checking problem is shown to be decidable on the class of bounded processes, we leave for future work the determination of a complexity bound. The class of bounded processes include the finite-control fragment of the π-calculus, and is defined by a notion of space-time reachability on sets of processes modulo finitely supported name permutations. Our model-checking algorithm, even when restricted to the behavioral and recursive fragment of the logic, seems to be structurally simpler than alternative proposals. We believe that these simplifications result from the modularization of the semantic development one obtains when following the general techniques of [5].

Acknowledgements Much of this paper builds on previous work with Luca Cardelli, to whom the author is thankful for so many illuminating discussions and suggestions. Luis Monteiro and Etienne Lozes are also acknowledged for their helpful comments. This research was partially funded by FET Profundis. Part of this work was developed during a visit to Microsoft Research Cambridge.

References

[1] Henrik Reif Andersen, Colin Stirling, and Glynn Winskel. A compositional proof system for the modal μ-calculus. In *Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 144–153, Paris, France, 4–7 July 1994. IEEE Computer Society Press.

[2] L. Caires. *A Model for Declarative Programming and Specification with Concurrency and Mobility*. PhD thesis, Dept. de Informática, FCT, Universidade Nova de Lisboa, 1999.

[3] L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part I). In N. Kobayashi and B.C. Pierce, editors, *10th Symposium on Theoretical Aspects of Computer Science (TACS 2001)*, volume 2215 of *Lecture Notes in Computer Science*, pages 1–30. Springer-Verlag, 2001.

[4] L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part II). In *CONCUR 2002: Concurrency Theory (13th International Conference)*, Lecture Notes in Computer Science. Springer-Verlag, 2002.

[5] L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part I). *Information and Computation*, accepted for publication, to appear in the special issue dedicated to TACS 2001.

[6] L. Caires and L. Monteiro. Verifiable and Executable Specifications of Concurrent Objects in $\mathcal{L}_\pi$. In C. Hankin, editor, *7th European Symp. on Programming (ESOP 1998)*, number 1381 in Lecture Notes in Computer Science, pages 42–56. Springer-Verlag, 1998.

[7] C. Calcagno, Luca Cardelli, and Andrew Gordon. Deciding Validity in a Spatial Logic of Trees. To appear, 2002.

[8] L. Cardelli, P. Gardner, and G. Ghelli. Manipulating trees with hidden labels. Submitted, 2002.

[9] L. Cardelli, P. Gardner, and G. Ghelli. A spatial logic for querying graphs. In *29td Colloquium on Automata, Languages and Programming (ICALP 2002) (Malaga, Spain)*, Lecture Notes in Computer Science. Springer-Verlag, 2002.

[10] L. Cardelli and G. Ghelli. A Query Language Based on the Ambient Logic. In D. Sands, editor, *10th European Symposium on Programming (ESOP 2001)*, volume 2028 of *Lecture Notes in Computer Science*, pages 1–22. Springer-Verlag, 2001.

[11] L. Cardelli and A. D. Gordon. Anytime, Anywhere. Modal Logics for Mobile Ambients. In *27th ACM Symp. on Principles of Programming Languages*, pages 365–377. ACM, 2000.

[12] L. Cardelli and A. D. Gordon. Logical Properties of Name Restriction. In S. Abramsky, editor, *Typed Lambda Calculi and Applications*, number 2044 in Lecture Notes in Computer Science. Springer-Verlag, 2001.

[13] W. Charatonik, A. D. Gordon, and J.-M. Talbot. Finite-control mobile ambients. In D. Metayer, editor, *11th European Symposium on Programming (ESOP 2002)*, number 2305 in Lecture Notes in Computer Science. Springer-Verlag, 2002.

[14] W. Charatonik and J.-M. Talbot. The decidability of model-checing mobile ambients. In D. Metayer, editor, *11th European Symposium on Programming (ESOP 2002)*, number 2305 in Lecture Notes in Computer Science. Springer-Verlag, 2001.

[15] S. Dal-Zilio. Spatial Congruence for Ambients is Decidable. In *ASIAN'00 — 6th Asian Computing Science Conference*, number 1961 in Lecture Notes in Computer Science, pages 365–377. Springer-Verlag, 2000.

[16] M. Dam. Model checking mobile processes. *Information and Computation*, 129(1):35–51, 1996.

[17] M. Dam. Proof systems for π-calculus logics. In de Queiroz, editor, *Logic for Concurrency and Synchronisation*, Studies in Logic and Computation. Oxford University Press, To appear.

[18] J. Engelfriet and Tj. Gelsema. Multisets and Structural Congruence of the π-calculus with Replication. *Theoretical Computer Science*, (211):311–337, 1999.

[19] M. Gabbay and A. Pitts. A New Approach to Abstract Syntax Involving Binders. *To appear in Formal Aspects of Computing*.

[20] M. Gabbay and A. Pitts. A New Approach to Abstract Syntax Involving Binders. In *14th Annual Symposium on Logic in Computer Science*, pages 214–224. IEEE Computer Society Press, Washington, 1999.

[21] D. Hirschkoff, E. Lozes, and D. Sangiorgi. Separability, Expressiveness and Decidability in the Ambient Logic. In *Third Annual Symposium on Logic in Computer Science*, Copenhagen, Denmark, 2002. IEEE Computer Society.

[22] S. Ishtiaq and P. O'Hearn. BI as an Assertion Language for Mutable Data Structures. In *28th ACM Symp. on Principles of Programming Languages*. ACM, 2001.

[23] D. Kozen. Results on the Propositional μ-Calculus. *Theoretical Computer Science*, 27(3):333–354, 1983.

[24] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[25] R. Milner. *Communicating and Mobile Systems: the π-calculus*. Cambridge University Press, 1999.

[26] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 114:149–171, 1993.

[27] A. Pitts. Nominal Logic: A First Order Theory of Names and Binding. In B.C. Pierce N. Kobayashi, editor, *10th Symposium on Theoretical Aspects of Computer Science (TACS 2001)*, volume 2215 of *Lecture Notes in Computer Science*, pages 219–235. Springer-Verlag, 2001.

[28] J. C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *Third Annual Symposium on Logic in Computer Science*, Copenhagen, Denmark, 2002. IEEE Computer Society.

[29] D. Sangiorgi. Extensionality and Intensionality of the Ambient Logics. In *28th Annual Symposium on Principles of Programming Languages*, pages 4–13. ACM, 2001.

[30] D. Sangiorgi and D. Walker. *The π-calculus; A Theory of Mobile Processes*. Cambridge University Press, 2001.

[31] Glynn Winskel. A note on model checking the modal ν-calculus. In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors, *12th Colloquium on Automata, Languages and Programming (ICALP 1989)*, volume 372 of *Lecture Notes in Computer Science*, pages 761–772. Springer-Verlag, 1989.

Appendix

Theorem 3.2

For all formulas A and B, and valuations v

1. If $A \equiv_\alpha B$ then $[\![A]\!]_v = [\![B]\!]_v$.
2. $[\![A]\!]_v \in \mathbb{P}$ with *supp*$([\![A]\!]_v) \subseteq fn^v(A)$.
3. (Equivariance) For all transpositions τ, $\tau([\![A]\!]_v) = [\![\tau(A)]\!]_{\tau(v)}$.
4. (Gabbay-Pitts Property) Let $M = fn^v(A) \cup fn(P)$.
   If $P \in [\![A\{x{\leftarrow}p\}]\!]_v$ for some $p \notin M$, then $P \in [\![A\{x{\leftarrow}p\}]\!]_v$ for all $p \notin M$.

Proof. By induction on the structure of the formula A.

- (Case A is $\mathcal{X}$) (2) If $P \in [\![\mathcal{X}]\!]_v$ then $P \equiv v(\mathcal{X})$. Hence for every $Q \equiv P$ we have $Q \in [\![\mathcal{X}]\!]_v$. If $m, n \notin fn^v(\mathcal{X})$ then $m, n \notin fn(v(\mathcal{X}))$, hence $\{m{\leftrightarrow}n\}(v(\mathcal{X})) \equiv v(\mathcal{X})$. So $[\![\mathcal{X}]\!]_v \in \mathbb{P}$ and *supp*$([\![\mathcal{X}]\!]_v) \subseteq fs^v(\mathcal{X})$.

  (3) If $P \in [\![\mathcal{X}]\!]_v$ then $P \equiv v(\mathcal{X})$, so $\tau(P) \equiv \tau(v(\mathcal{X}))$. Then $\tau(P) \in [\![\tau(\mathcal{X})]\!]_{\tau(v)}$.

- (Case A is $\mathcal{X} : B|C$) (2) If $P \in [\![\mathcal{X} : B|C]\!]_v$ then there are Q, R such that $P \equiv Q|R$ and $Q \in [\![B]\!]_v$ and $R \in [\![C]\!]_{v[\mathcal{X}\leftarrow Q]}$. It is immediate that if $P' \equiv P$ then $P' \in [\![\mathcal{X} : B|C]\!]_v$. So pick $m, n \notin fn^v(\mathcal{X} : B|C) = fn^v(B) \cup fn^{v'}(C)$ (where $v' \triangleq v[\mathcal{X}{\leftarrow}\mathbf{0}]$). Let $\tau \triangleq \{m{\leftrightarrow}n\}$. By induction hypothesis(2), we have $\tau(Q) \in [\![B]\!]_v$. By induction hypothesis(3), $\tau(R) \in [\![C]\!]_{\tau(v)[\mathcal{X}\leftarrow\tau(Q)]} = [\![C]\!]_{v[\mathcal{X}\leftarrow\tau(Q)]}$. So, $\tau(Q)|\tau(R) \equiv \tau(P) \in [\![\mathcal{X} : B|C]\!]_v$. Hence $[\![\mathcal{X} : B|C]\!]_v \in \mathbb{P}$ and *supp*$([\![\mathcal{X} : B|C]\!]_v) \subseteq fs^v(\mathcal{X} : B|C)$.

  (3) If $P \in [\![\mathcal{X} : B|C]\!]_v$ then there are Q, R such that $P \equiv Q|R$ and $Q \in [\![B]\!]_v$ and $R \in [\![C]\!]_{v[\mathcal{X}\leftarrow Q]}$. By induction hypothesis(3), we conclude $\tau(Q) \in [\![\tau(B)]\!]_{\tau(v)}$ and $\tau(R) \in [\![\tau(C)]\!]_{\tau(v)[\mathcal{X}\leftarrow\tau(Q)]}$. Hence $\tau(P) \in [\![\tau(\mathcal{X} : B|C)]\!]_{\tau(v)}$.

- (Case A is $A \triangleleft \mathcal{X}$) (2) If $P \in [\![A \triangleleft \mathcal{X}]\!]_v$ then $v(\mathcal{X})|P \in [\![A]\!]_v$. By induction hypothesis(2), we have $[\![A]\!]_v \in \mathbb{P}$ and *supp*$([\![A]\!]_v) \subseteq fn^v(A)$. Let $P \equiv Q$, then $v(\mathcal{X})|P \equiv v(\mathcal{X})|Q$ and then $Q \in [\![A \triangleleft \mathcal{X}]\!]_v$. Let $\tau = \{m{\leftrightarrow}n\}$ with $m, n \notin fs^v(A \triangleleft \mathcal{X}) = fs^v(A) \cup fn(v(\mathcal{X}))$. Then $m, n \notin fs^v(A)$, thus we have $\tau(P) \in [\![A]\!]_v$. Then

$v(\mathcal{X})|\tau(P) \in [\![A \triangleleft \mathcal{X}]\!]_v$. Since $m, n \notin \mathit{fn}(v(\mathcal{X}))$, we conclude $\tau(v(\mathcal{X})|P) \in [\![A \triangleleft \mathcal{X}]\!]_v$. Hence $[\![A \triangleleft \mathcal{X}]\!]_v \in \mathbb{P}$ and $\mathit{supp}([\![A \triangleleft \mathcal{X}]\!]_v) \subseteq \mathit{fs}^v(A \triangleleft \mathcal{X})$.

(3) If $P \in [\![A \triangleleft \mathcal{X}]\!]_v$ then $v(\mathcal{X})|P \in [\![A]\!]_v$. By induction hypothesis(3), we conclude $\tau(v(\mathcal{X}))|\tau(P) \in [\![\tau(A)]\!]_{\tau(v)}$. Hence $\tau(P) \in [\![\tau(A \triangleleft \mathcal{X})]\!]_{\tau(v)}$.

- (Case A is $\alpha.A$) (2) If $P \in [\![\alpha.A]\!]_v$ then there is Q such that $P \xrightarrow{\alpha} Q$ and $Q \in [\![A]\!]_v$. By induction hypothesis(2), we have $[\![A]\!]_v \in \mathbb{P}$ and $\mathit{supp}([\![A]\!]_v) \subseteq \mathit{fn}^v(A)$. Let $P \equiv P'$, then we have $P' \xrightarrow{\alpha} Q$, hence $P' \in [\![\alpha.A]\!]_v$.

  Let $\tau = \{m{\leftrightarrow}n\}$ with $m, n \notin \mathit{fs}^v(\alpha.A) = \mathit{fn}(\alpha) \cup \mathit{fs}^v(A)$. Then $m, n \notin \mathit{fs}^v(A)$, thus we have $\tau(Q) \in [\![A]\!]_v$. By Lemma 2.7(4), we have $\tau(P) \xrightarrow{\tau(\alpha)} \tau(Q)$. Since $m, n \notin \mathit{fn}(\alpha)$, we have $\tau(P) \xrightarrow{\alpha} \tau(Q)$. But then $\tau(P) \in [\![\alpha.A]\!]_v$. Hence $[\![\alpha.A]\!]_v \in \mathbb{P}$ and $\mathit{supp}([\![\alpha.A]\!]_v) \subseteq \mathit{fs}^v(\alpha.A)$.

  (3) If $P \in [\![\alpha.A]\!]_v$ then there is Q such that $P \xrightarrow{\alpha} Q$ and $Q \in [\![A]\!]_v$. By induction hypothesis(3), we have $\tau(Q) \in [\![\tau(A)]\!]_{\tau(v)}$. By Lemma 2.7(4), we have $\tau(P) \xrightarrow{\tau(\alpha)} \tau(Q)$. Thus, $\tau(P) \in [\![\tau(\alpha.A)]\!]_{\tau(v)}$.

- (Case A is $\boldsymbol{\nu}X.A$) (2) If $P \in [\![\boldsymbol{\nu}X.A]\!]_v$ then there is $\Psi \in \mathbb{P}$ such that $P \in \Psi$ and $\Psi \subseteq [\![A]\!]_{v[X \leftarrow \Psi]}$. If $Q \equiv P$ then $Q \in \Psi$, since $\Psi \in \mathbb{P}$.

  So pick $m, n \notin \mathit{fs}^v(\boldsymbol{\nu}X.A)$ and $\tau = \{m{\leftrightarrow}n\}$. We have $\tau(\Psi) \subseteq \tau([\![A]\!]_{v[X \leftarrow \Psi]})$. By induction hypothesis(3) we conclude that $\tau([\![A]\!]_{v[X \leftarrow \Psi]}) = [\![\tau(A)]\!]_{\tau(v)[X \leftarrow \tau(\Psi)]} = [\![A]\!]_{\tau(v)[X \leftarrow \tau(\Psi)]}$. Now, since $m, n \notin \mathit{fs}^v(\boldsymbol{\nu}X.A)$, we have $m, n \notin \mathit{supp}(v(X))$ for all $X \in \mathit{fpv}(A)$. Therefore we have $[\![A]\!]_{\tau(v)[X \leftarrow \tau(\Psi)]} = [\![A]\!]_{v[X \leftarrow \tau(\Psi)]}$. Hence $\tau(\Psi) \subseteq [\![A]\!]_{v[X \leftarrow \tau(\Psi)]}$. Since $\tau(P) \in \tau(\Psi)$, we conclude $\tau(P) \in [\![\boldsymbol{\nu}X.A]\!]_v$. Hence $[\![\boldsymbol{\nu}X.A]\!]_v \in \mathbb{P}$ and $\mathit{supp}([\![\boldsymbol{\nu}X.A]\!]_v) \subseteq \mathit{fs}^v(\boldsymbol{\nu}X.A)$.

  (3) If $P \in [\![\boldsymbol{\nu}X.A]\!]_v$ then there is $\Psi \in \mathbb{P}$ such that $P \in \Psi$ and $\Psi \subseteq [\![A]\!]_{v[X \leftarrow \Psi]}$. As in (2) we conclude $\tau(\Psi) \subseteq [\![\tau(A)]\!]_{\tau(v)[X \leftarrow \tau(\Psi)]}$, by induction hypothesis(3). Since $\tau(P) \in \tau(\Psi)$ we have $\tau(P) \in [\![\tau(\boldsymbol{\nu}X.A)]\!]_{\tau(v)}$. ∎

Proposition 4.7

For every process P, formula A and syntactic valuation v for A we have:

1. If $MC(P, v, A) = \textbf{true}$ then $P \in [\![A]\!]_{v^\star}$.
2. If $MC(P, v, A) = \textbf{false}$ then $P \notin [\![A]\!]_{v^\star}$.

Proof. By (mutual) induction on the derivation of $MC(P, v, A)$. We consider the several possible cases for A.

- (Case A is $\mathbf{T}$, $n = m$ or $\mathcal{X}$) Immediate.
- (Case A is $\neg B$) By induction hypothesis.
- (Case A is $B \wedge C$) By induction hypothesis.
- (Case A is $\mathbf{0}$) We have that if $MC(P, v, \mathbf{0}) = \textbf{true}$ then $P \equiv \mathbf{0}$ and thus $P \in [\![\mathbf{0}]\!]_{v^\star}$, and if $MC(P, v, \mathbf{0}) = \textbf{false}$ then $P \not\equiv \mathbf{0}$, and so $P \notin [\![\mathbf{0}]\!]_{v^\star}$.

- (Case A is $\mathcal{X} : A|B$)

  By the induction hypothesis, $MC(P, v, \mathcal{X} : A|B) =$ **true** implies that there is a pair of processes $(Q, R) \in Comp(P)$ such that $Q \in [\![A]\!]_{v^\star}$. and $R \in [\![B]\!]_{v[\mathcal{X} \leftarrow Q]^\star}$. By Lemma 4.1(1b) we have $P \equiv Q|R$; thus $P \in [\![\mathcal{X} : A|B]\!]_{v^\star}$.

  If $MC(P, v, \mathcal{X} : A|B) =$ **false** then, by induction hypothesis, for all processes Q and R such that $(Q, R) \in Comp(P)$ we either have $Q \notin [\![A]\!]_{v^\star}$, or $R \notin [\![B]\!]_{v[\mathcal{X} \leftarrow Q]^\star}$.

  By Lemma 4.1(1a), this implies that for all processes Q and R such that $Q|R \equiv P$ we either have $Q \notin [\![A]\!]_{v^\star}$ or $R \notin [\![B]\!]_{v[\mathcal{X} \leftarrow Q]^\star}$. Hence $P \notin [\![\mathcal{X} : A|B]\!]_{v^\star}$.

- (Case A is $B \triangleleft \mathcal{X}$)

  By the induction hypothesis, $MC(P, v, B \triangleleft \mathcal{X}) =$ **true** implies that $v(\mathcal{X})|R \in [\![C]\!]_{v^\star}$. Hence $P \in [\![B \triangleleft \mathcal{X}]\!]_{v^\star}$. Identical for $MC(P, v, B \triangleleft \mathcal{X}) =$ **false**.

- (Case A is n®B, $B \oslash n$, or $\alpha.B$)

  Similar to the case (A is $\mathcal{X} : A|B$) above, using Lemma 4.1(2,3).

- (Case A is $\text{И}x.B$)

  If $MC(P, v, \text{И}x.B) =$ **true** then we have $MC(P, v, B\{x \leftarrow n\}) =$ **true** where $f = fresh(fn(P) \cup fs^v(B))$. By induction hypothesis, we have $P \in [\![B\{x \leftarrow f\}]\!]_{v^\star}$. Thus $P \in [\![\text{И}x.B]\!]_{v^\star}$.

  If $MC(P, v, \text{И}x.B) =$ **false** then we have $MC(P, v, B\{x \leftarrow f\}) =$ **false** where $f = fresh(fn(P) \cup fs^v(B))$. By induction hypothesis, we have $P \notin [\![B\{x \leftarrow f\}]\!]_{v^\star}$. Thus $P \notin [\![\text{И}x.B]\!]_{v^\star}$ by Theorem 3.2(5), since $f \notin fn(P) \cup fn^{v^\star}(B)$ by Lemma 4.6(2).

- (Case A is $\forall x.B$)

  Let $M = fn(P) \cup fs^v(A) \cup \{f\}$ where $f = fresh(fn(P) \cup fs^v(A))$.

  Assume $MC(P, v, \forall x.B) =$ **true**.

  Then, for every $n \in M$ we have $MC(P, v, B\{x \leftarrow n\}) =$ **true**, and thus $P \in [\![B\{x \leftarrow n\}]\!]_{v^\star}$, by induction hypothesis.

  On the other hand, for every $n \notin M$, by Theorem 3.2(4), we have $\{n \leftrightarrow f\}P \equiv P \in [\![\{n \leftrightarrow f\}(B\{x \leftarrow f\})]\!]_{\{n \leftrightarrow f\}v^\star} = [\![B\{x \leftarrow n\}]\!]_{\{n \leftrightarrow f\}v^\star}$. Since $n, f \notin fn^{v^\star}(B)$ by Lemma 4.6(2), we conclude $[\![B\{x \leftarrow n\}]\!]_{v^\star}$. Hence $P \in [\![\forall x.B]\!]_{v^\star}$.a

  Assume $MC(P, v, \forall x.B) =$ **false**.

  Then for some $n \in M$ we have $MC(P, v, B\{x \leftarrow n\}) =$ **false**. By induction hypothesis, we have $P \notin [\![B\{x \leftarrow n\}]\!]_{v^\star}$. Thus $P \notin [\![\forall x.B]\!]_{v^\star}$.

- (Case A is X) Assume $v = v'[X \leftarrow (S, \boldsymbol{\nu}X.B)]v''$ for some v', v'', S and B. Let $M = fs^v(\boldsymbol{\nu}X.B)$.

  If $MC(P, v, X) =$ **true** then we either have:

  (1) $P = \rho(Q)$ for some $Q \in S$ and permutation ρ that fixes M, or

  (2) $MC(P, v(X + P)), B) =$ **true**.

  Recall that by construction, we have $v^\star(X) = Gfix(\lambda\Psi.S^\star \cup \phi(\Psi))$, where $\phi(\Psi) = [\![B]\!]_{v^\star[X \leftarrow \Psi]}$ and $S^\star \triangleq Close(S, M)$.

(1) Since $Q \in S$, we also have $Q \in S^\star$, and thus $Q \in v^\star(X)$. Since $supp(v^\star(X)) \subseteq M$ we also have $\rho(Q) \equiv P \in v^\star(X)$.

(2) Let $w \triangleq v(X + P)$, thus $w = v'[X\leftarrow(\{P\} \cup S, B)]v''$.

By induction hypothesis we have $P \in [\![B]\!]_{w^\star}$. By construction, we have $w^\star(X) = Gfix(\lambda\Psi.S^\star \cup \{P\}^\star \cup \phi(\Psi))$ where $\{P\}^\star \triangleq Close(\{P\}, M)$. Then $P \in [\![B]\!]_{w^\star} = \phi(Gfix(\lambda\Psi.(S^\star \cup \{P\}^\star \cup \phi(\Psi))))$.

Since $supp([\![B]\!]_{w^\star}) \subseteq fn^{w^\star}(B)$ by Theorem 3.2(3) and $fn^{w^\star}(B) \subseteq fs^w(B) = M$ by Lemma 4.6(2), we have $supp([\![B]\!]_{w^\star}) \subseteq M$. Hence $\{P^\star\} \subseteq [\![B]\!]_{w^\star}$. Then $\{P\}^\star \subseteq S^\star \cup \phi(Gfix(\lambda\Psi.(S^\star \cup \{P\}^\star \cup \phi(\Psi))))$.

By Proposition 3.3(3), we conclude $P \in Gfix(\lambda\Psi.S^\star \cup \phi(\Psi)) = v^\star(X)$.

If $MC(P, v, X) =$ **false** then $MC(P, v(X + P)), B) =$ **false**, and for all $Q \in S$ we have $P \not\equiv_M Q$.

By induction hypothesis, we conclude that $P \notin [\![B]\!]_{w^\star}$, where w is defined as in (2) above. For the sake of getting a contradiction, assume $P \in v^\star(X)$. This implies $P \in Gfix(\lambda\Psi.S^\star \cup \phi(\Psi))$, in fact $\{P\}^\star \subseteq Gfix(\lambda\Psi.S^\star \cup \phi(\Psi))$.

By Proposition 3.3(3), we conclude $\{P\}^\star \subseteq S^\star \cup \phi(Gfix(\lambda\Psi.(S^\star \cup \{P\}^\star \cup \phi(\Psi)))) = S^\star \cup [\![B]\!]_{w^\star}$. Note that we cannot have $P \in S^\star$, because otherwise we would have $P \equiv_M Q$ for some $Q \in S$. So, $P \in [\![B]\!]_{w^\star}$, contradicting the induction hypothesis. Hence $P \notin v^\star(X)$.

- (Case A is $\boldsymbol{\nu}X.B$)

  Let $\phi(\Psi) \triangleq [\![B]\!]_{v^\star[X\leftarrow\Psi]}$.

  $MC(P, v, \boldsymbol{\nu}X.B) =$ **true** comes from $MC(P, v[X\leftarrow(\{P\}, B)], \boldsymbol{\nu}X.B) =$ **true**. Let $\Phi = Gfix(\lambda\Psi.\{P\} \cup \phi(\Psi))$. By induction hypothesis, we have $P \in \phi(\Phi)$.

  By Proposition 3.3(3) we have $P \in Gfix(\lambda\Psi.\phi(\Psi)) = [\![\boldsymbol{\nu}X.B]\!]_{v^\star}$.

  For $MC(P, v, \boldsymbol{\nu}X.B) =$ **false** then $MC(P, v[X\leftarrow(\{P\}, \boldsymbol{\nu}X.B)], B) =$ **true**, and $P \notin \phi(\Phi)$, by induction hypothesis.

  By Proposition 3.3(3) we have $P \notin Gfix(\lambda\Psi.\phi(\Psi)) = [\![\boldsymbol{\nu}X.B]\!]_{v^\star}$. ∎

Proposition 5.2

Proof. (1) By induction on the structure of P, we show the more general statement:

Let $R_1 \triangleq$ (**rec** $\mathcal{X}_1)Q_1, \ldots, R_n \triangleq$ (**rec** $\mathcal{X}_n)Q_n$ be any sequence of processes where $fpv(Q_i) \subseteq \{\mathcal{X}_1, \ldots, \mathcal{X}_i\}$.

Let P be any process such that $fpv(P) = \{\mathcal{X}_1, \ldots, \mathcal{X}_n\}$. Write θ_n for the substitution $\{X\leftarrow R_1\}\cdots\{X_n\leftarrow R_n\}$ and v_n for the valuation $[X_1\leftarrow\{R_1\}, \cdots, X_n\leftarrow\{R_n\}]$.

Then we have $\theta_n(P) \models_{v_n} |P|$.

(Case P is $\mathbf{0}$) Immediate.

(Case P is $P'|P''$) By induction hypothesis, we have $\theta_n(P') \models_{v_n} |P'|$ and $\theta_n(P'') \models_{v_n} |P''|$. Then $\theta_n(P'|P'') \models_{v_n} |P'|P''|$.

(Case P is $(\boldsymbol{\nu}m)P'$) We have $\theta_n((\boldsymbol{\nu}m)P') = (\boldsymbol{\nu}m')\theta_n(P'\{m\leftrightarrow m'\})$, where m' is not free in any of the Q_i nor in P'. By induction hypothesis, we have $\theta_n(P'\{m\leftrightarrow m'\}) \models_{v_n} |P'\{m\leftrightarrow m'\}|$.

Then $(\boldsymbol{\nu}m')\theta_n(P'\{m\leftrightarrow m'\}) \models_{v_n} m'®|P'\{m\leftrightarrow m'\}|$.

Hence, $\theta_n((\boldsymbol{\nu} m')P'\{m\leftrightarrow m'\}) \models_{v_n} (x\circledR(|P'|\{m\leftarrow x\}))\{x\leftarrow m'\}$.

Since $m' \notin \mathit{fn}(\theta_n((\boldsymbol{\nu} m')P'\{m\leftrightarrow m'\}))$ and $m' \notin \mathit{fn}^{v_n}(\mathsf{H}x.|P'|\{m\leftarrow x\})$, we conclude $\theta_n((\boldsymbol{\nu} m')P'\{m\leftrightarrow m'\}) \models_{v_n} \mathsf{H}x.(|P'|\{m\leftarrow x\})$.

Moreover, since $\theta_n((\boldsymbol{\nu} m')P'\{m\leftrightarrow m'\}) \equiv \theta_n((\boldsymbol{\nu} m)P')$, we have $\theta_n((\boldsymbol{\nu} m)P') \models_{v_n} \mathsf{H}x.(|P'|\{m\leftarrow x\})$.

(Case P is $\alpha.P'$) Since $P \xrightarrow{\alpha} P'$, we also have $\theta_n(P) \xrightarrow{\alpha} \theta_n(P')$. Note that $\theta_n(P) \models_{v_n} \mathbf{1} \wedge NoRes$. By induction hypothesis, $\theta_n(P') \models_{v_n} |P'|$. Hence $\theta_n(P) \models_{v_n} \alpha.|P'|$ and we conclude the result.

(Case P is $\mathcal{X}$) We have $\theta_n(\mathcal{X}) \in v_n(X)$ by construction of v_n and θ_n.

(Case P is $(\mathbf{rec}\ \mathcal{Y})P'$) We assume $\mathcal{Y}$ fresh.

We need to show $\theta_n((\mathbf{rec}\ \mathcal{Y})P') \models_{v_n} \boldsymbol{\nu} Y.|P'|$. Let $R_{n+1} \triangleq (\mathbf{rec}\ \mathcal{Y})\theta_n(P')$, $\theta_{n+1} = \theta_n; \{Y\leftarrow R_{n+1}\}$ and $v_{n+1} = v_n; [X\leftarrow\{R_{n+1}\}]$. To conclude by coinduction we must show $(\mathbf{rec}\ \mathcal{Y})\theta_n(P') \models_{v_{n+1}} |P'|$. By closure of satisfaction under structural congruence, this is equivalent to $\theta_{n+1}(P') \models_{v_{n+1}} |P'|$. But this follows from the induction hypothesis.

(2) By induction on the structure of the process P.

(Case P is $\mathbf{0}$) If $Q \models |\mathbf{0}|$ then $Q \equiv \mathbf{0}$.

(Case P is $P'|P''$) We have that $Q \models |P'|||P''|$ implies that $Q \equiv Q'|Q''$ such that $Q' \models |P'|$ and $Q'' \models |P''|$. By induction hypothesis, $Q' \equiv P'$ and $Q'' \equiv P''$. Thus $Q \equiv P'|P''$.

(Case P is $(\boldsymbol{\nu} m)P'$) We have that $Q \models \mathsf{H}x.(|P'|\{m\leftarrow x\})$. This implies that there is $p \notin \mathit{fn}(Q) \cup \mathit{fn}(|P|)$ and Q' such that $Q \equiv (\boldsymbol{\nu} p)Q'$ and $Q' \models |P'|\{m\leftarrow p\}$. By induction hypothesis, we have $Q' \equiv P'\{m\leftarrow p\}$. Hence $Q \equiv (\boldsymbol{\nu} p)(P'\{m\leftarrow p\})$. But $(\boldsymbol{\nu} p)(P'\{m\leftarrow p\}) \equiv (\boldsymbol{\nu} m)P'$, and we obtain the result.

(Case P is $m\langle n\rangle.P'$) We have that $Q \models \mathbf{1} \wedge NoRes \wedge m\langle n\rangle.|P'|$. Since $Q \models \mathbf{1}$, and $Q \models NoRes$, Q must be either an input or output guarded process. Moreover, since $Q \models m\langle n\rangle.|P'|$ there is a process Q' such that $Q \xrightarrow{m\langle n\rangle} Q'$ and $Q' \models |P'|$. So we must have $Q \equiv m\langle n\rangle.Q'$. By induction hypothesis, $Q' \equiv P'$. Hence $Q \equiv m\langle n\rangle.P'$.

(Case P is $n(m).P'$) We have that $Q \models \mathbf{1} \wedge NoRes \wedge \forall x.n(x).|P'\{m\leftarrow x\}|$. Since $Q \models \mathbf{1}$, and $Q \models NoRes$, Q must be either an input or output guarded process. Moreover, for all names p there is a process Q' such that $Q \xrightarrow{n(p)} Q'$ and $Q' \models |P'|\{m\leftarrow p\}$. In particular, there is Q' such that $Q \xrightarrow{n(m)} Q'$ and $Q' \models |P'|$. So we must have $Q \equiv n(m).Q'$. By induction hypothesis, $Q' \equiv P'$. Hence $Q \equiv n(m).P'$. ■