# Behavioral and Spatial Observations in a Logic for the $\pi$-Calculus

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

**Abstract.** In addition to behavioral properties, spatial logics can talk about other key properties of concurrent systems such as secrecy, freshness, usage of resources, and distribution. We study an expressive spatial logic for systems specified in the synchronous $\pi$-calculus with recursion, based on a small set of behavioral and spatial observations. We give coinductive and equational characterizations of the equivalence induced on processes by the logic, and conclude that it strictly lies between structural congruence and strong bisimulation. We then show that model-checking is decidable for a useful class of processes that includes the finite-control fragment of the $\pi$-calculus.

## Introduction

Spatial logics support the specification not only of behavioral properties but also of structural properties of concurrent systems, in a fairly integrated way. Spatial properties arise naturally in the specification of distributed systems, for instance connectivity, stating that there is always an access route between two different sites, unique handling, stating that there is at most one server process listening on a given channel name, or resource availability, stating that a bound exists on the number of channels that can be allocated at a given location. Even secrecy can also be sometimes 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 [4, 3]. Essentially, spatial logics are modal logics that can talk about the internal structure of each world. The interpretation of each world as a structured space, and moreover as a space seen as a certain kind of resource [23], distinguishes spatial logics among modal logics. Spatial logics have been recently used in the definition of several core languages, calculi, and data models [2, 6, 18, 4, 5]. In this paper, we study a logic for systems modeled in the synchronous $\pi$-calculus with spatial and temporal operators, freshness quantifiers, and recursive formulas.

**Spatial and Behavioral Observations** In behavioral models of concurrency, a process is identified with its observable behavior, roughly, the sequence of interactions it can perform in the course of time. Modalities of a purely behavioral logic support the specification of processes by allowing us to talk about their actions; logics of this kind [21, 11, 12] are extensions of those introduced by Hennessy and Milner [19]. Although there is a traditional distinction between static and dynamic operations in process algebras [19], a purely behavioral semantics blurs the distinction between these two kinds of operations, to the extent that all process operators end up interpreted as purely behavioral, abstracting away from all structural information. The equivalence induced on the set of all processes by such logics is then expected to match some notion of behavioral equivalence (*e.g.*, strong bisimulation).

Spatial logics offer an enhanced power of observation, when compared with purely behavioral logics, because they can distinguish between systems that differ on their distributed structure, but not on their behavior. Spatial observations may then appear perhaps too much intensional. However, while certainly more intensional than purely behavioral observations, spatial observations are of a semantic nature, and should be actually extensional with respect to some well-defined model of space. Therefore, a spatial logic for concurrent processes should separate processes according to such well-defined spatial / behavioral semantic model.

A spatial logic may then add to a given set of behavioral modalities a set of spatial operators, closely related to the static operators of the process calculus, as in [2]. For nominal process calculi, the static operators are the composition $P \mid Q$, its identity element $\mathbf{0}$ (denoting the empty system), and the name restriction $(\nu n)P$. These process constructors give rise to the composition formula $A \mid B$, that holds of a process that can be separated 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 locally restricted channels.

Alternatively, a spatial logic can put a stronger emphasis on structure, and allow the observation of a process behavior in a more indirect way, using spatial adjuncts together with a minimal "next step" (*cf.*, the formula $\langle\tau\rangle A$) or "eventually" behavioral modality. The first proposal in this vein is the ambient logic of [6], also adopted in the $\pi$-calculus logic of [4, 3]. An advantage of this approach is its generality, moreover, it is easily adaptable to any process calculus whose operational semantics can be presented by means of a simple unlabeled reduction relation. Adjuncts are very expressive: composition adjunct $A \triangleright B$ supports an internal definition of validity, and makes it possible to express quite general context/system specifications. However, model-checking of logics with composition adjunct, and including either quantification over names [9] or revelation [15] turns out to be undecidable even for the simplest process languages.

**Overview and Contributions** In this work, we study a $\pi$-calculus logic which is based on purely structural spatial and behavioral observations. By "purely structural" we mean observations that can be determined by inspection of the local structure of the processes; therefore the logic does not include adjuncts operators. As a consequence, we obtain decidability of model-checking on interesting classes of processes, and preserve the ability to express context-dependent behavioral and spatial properties.

For the spatial fragment we consider the connectives of composition, void, and revelation. For the behavioral fragment we pick a few simple modalities, defined either from the label $\tau$, 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 $m$ on channel $n$, and the action of receiving name $m$ on channel $n$. To this basic set of connectives, we add propositional operators, first-order and freshness quantifiers, and recursive definitions, along the lines of [4].

To illustrate in an informal way the expressiveness of the logic, we go through a few examples. First, we show that by combining the fresh and hidden name quantifiers with the behavioral operators we can define modalities for name extrusion and intrusion (*cf.*, [21]).

$$n\langle\nu x\rangle.A \triangleq \mathsf{H}x.n\langle x\rangle.A \text{ (Bound Output)} \qquad n(\nu x).A \triangleq \text{И}x.n(x).A \text{ (Bound Input)}$$

The definition of bound output uses the hidden name quantifier [2, 4]. The hidden name quantifier is derivable [7] from the fresh name quantifier and the revelation operator: $\mathsf{H}x.A \triangleq \text{И}x.x®A$. Using these two operators we can define the following formula *Comm*.

$$\begin{aligned}
\mathit{Comm} &\triangleq m\langle \nu x\rangle.A \mid m(\nu x)B \Rightarrow \tau.\mathsf{H}x.(A \mid B)\\
\mathit{Pair} &\triangleq ((\nu n)m\langle n\rangle.n\langle m\rangle.\mathbf{0}) \mid m(q).q\langle q\rangle.\mathbf{0}
\end{aligned}$$

The formula *Comm* talks about name extrusion: it says that two separate parts of a system can become "connected" by a shared secret, after interacting. For example, the process *Pair* defined above satisfies the formula *Comm*. It also satisfies the formula $(\neg\mathbf{0} \mid \neg\mathbf{0}) \wedge \tau.\neg(\neg\mathbf{0} \mid \neg\mathbf{0})$: this formula says that the process has two separate threads initially, that become tied by a private shared channel after a reduction step. This illustrates the fact that the logic has the power to count resources (*e.g.*, threads, restricted channels). Combining spatial operators and recursive formulas we can define other useful operators, *e.g.*, $\mathsf{H}^*A \triangleq \mu X.(A \vee \mathsf{H}x.X)$; the formula $\mathsf{H}^*A$ means that $A$ holds under a (finite) number of restricted names [4]. Then, the formula $\neg\mathsf{H}^*\exists y.(\exists x.y(x).\mathbf{T} \mid \exists x.y(x).\mathbf{T})$ expresses a unique handling property [20], it is satisfied by systems that do no contain separate processes listening on the same (public or private) channel name.

The first contribution of this work is thus the proposal of the logic and the characterization of its expressive power, in terms of the equivalence relation (written $=_{\mathrm{L}}$) it induces on processes, aiming at a better understanding of its intended spatial model. We give coinductive and equational characterisation of $=_{\mathrm{L}}$, showing that it is a decidable congruence, even for the full process language with recursion. The equational presentation turns out to be the extension of the standard axiomatization of structural congruence with two natural principles: an axiom expressing absorption of identical choices (*cf.*, the axiom $P+P=P$ for bisimulation), and a coinduction principle, asserting uniqueness of solutions to equations. This shows that $=_{\mathrm{L}}$ lies strictly "in between" structural congruence and strong bisimulation, the gap towards strong bisimulation seems to be essentially justified by the failure of the expansion law in the spatial model. As a second contribution, we present a model-checker for the full logic and calculus, and show that model-checking is decidable for a class of bounded processes that includes the finite-control $\pi$-calculus. The algorithm builds on the decidable characterization of $=_{\mathrm{L}}$, and its presentation is surprisingly compact: we believe this to be a consequence of adopting a Pset-based semantic foundation [4], and permutation-based techniques [14, 22].

## 1 The Process Model

In this section, we briefly introduce the syntax and operational semantics of the synchronous $\pi$-calculus. We adopt a version with guarded choice [20], but with recursion replacing replication.

**Definition 1.1 (Actions and Processes).** *Given infinite sets* $\Lambda$ *of* pure names $(m,n,p)$ *and* $\chi$ *of* process variables $(\mathcal{X},\mathcal{Y},\mathcal{Z})$*, the sets* $\mathcal{A}$ *of actions* $(\alpha,\beta)$*,* $\mathcal{N}$ *of* normal processes $(N,T,U)$*,* $\mathcal{P}$ *of* processes $(P,Q,R)$*, and* $\mathcal{A}$ *of* abstractions $(F,G)$ *are defined by*

$$\begin{array}{ll}
\alpha,\beta ::= m(n) \mid m\langle n\rangle & N,T ::= \alpha.P \mid N+T\\
F,G ::= (\overline{n})P & P,Q ::= \mathbf{0} \mid N \mid P \mid Q \mid (\nu n)P \mid \mathcal{X}[\overline{n}] \mid (\mathbf{rec}\,\mathcal{X}(\overline{n}).P)[\overline{p}]
\end{array}$$

Each component of a choice $N+T$ is a *guarded* process, that is, either an input process $m(n).P$ or an output process $m\langle n\rangle.P$. In restriction $(\nu n)P$ and input $m(n).P$ the distinguished occurrence of the name $n$ is binding, with scope the process $P$. The bound names $bn(\alpha)$ of an action $\alpha$ are given by $bn(m(n)) \triangleq \{n\}$ and $bn(m\langle n\rangle) \triangleq \emptyset$. In a recursive process $(\mathbf{rec}\, \mathcal{X}(\overline{n}).P)[\overline{p}]$, the distinguished occurrences of the variable $\mathcal{X}$ and names $\overline{n}$ are binding, with scope the process $P$. As usual, we require all free occurrences of $\mathcal{X}$ in $P$ to be *guarded*, that is, they may only occur inside the continuation part $Q$ of a guarded process $\alpha.Q$ in $P$. For any process $P$, we assume defined as usual the set $fn(P)$ of *free names* of $P$, and the set $fpv(P)$ of *free process variables* of $P$. A process is *closed* if it does not contain free occurrences of process variables, in general by "process" we mean "closed process".

Abstractions denote functions from names to processes, our basic use for abstractions is in the definition of substitutions for process variables. A *substitution* $\theta$ is a mapping assigning a name to each name in its finite domain $\mathfrak{D}(\theta)$, and an abstraction of the appropriate arity to each process variable in $\mathfrak{D}(\theta)$. We write $\{n\leftarrow m\}$ (respectively $\{\mathcal{X}\leftarrow F\}$) for the singleton substitution of domain $\{n\}$ (respectively $\{\mathcal{X}\}$) that assigns $m$ to $n$ (respectively $F$ to $\mathcal{X}$).

We assume defined the relation of $\alpha$-*congruence* $\equiv_\alpha$ that identifies processes up to the safe renaming of bound names and bound process variables. For any process $P$ and substitution $\theta$ we denote by $\theta(P)$ the result of the safe application of $\theta$ to $P$ (using $\alpha$-conversion as needed to avoid illegal capture of free variables). The action of substitutions on process variables is defined as expected, *e.g.*, if $\theta(\mathcal{X}) = (\overline{q})P$ then $\theta(\mathcal{X}[\overline{m}]) = P\{\overline{q}\leftarrow\overline{m}\}$. We abbreviate $\{\mathcal{X}\leftarrow(\overline{q})(\mathbf{rec}\, \mathcal{X}(\overline{n}).P)[\overline{q}]\}$ by $\{\mathcal{X}\leftarrow(\mathbf{rec}\, \mathcal{X}(\overline{n}).P)\}$, and write $P\theta$ or $P\theta$ for $\theta(P)$.

**Definition 1.2 (Observable Names).** *For every closed process P and $i \geq 0$ we define*

$$
\begin{array}{ll}
ofn_i(\mathbf{0}) = \emptyset & ofn_i(m(n).P) = \{m\} \cup (ofn_i(P) \setminus \{n\}) \\
ofn_i(P \mid Q) = ofn_i(P) \cup ofn_i(Q) & ofn_i(N+T) = ofn_i(N) \cup ofn_i(T) \\
ofn_i((\nu n)P) = ofn_i(P) \setminus \{n\} & ofn_0((\mathbf{rec}\, \mathcal{X}(\overline{n}).P)[\overline{p}]) = \emptyset \\
ofn_i(m\langle n\rangle.P) = \{m,n\} \cup ofn_i(P) & ofn_{i+1}((\mathbf{rec}\, \mathcal{X}(\overline{n}).P)[\overline{p}]) = \\
& \qquad ofn_i(P\{\overline{n}\leftarrow\overline{p}\}\{\mathcal{X}\leftarrow(\mathbf{rec}\, \mathcal{X}(\overline{n}).P)\})
\end{array}
$$

*The set $ofn(P)$ of* observable names *of P is defined by $ofn(P) \triangleq \bigcup_{i\geq 0} ofn_i(P)$.*

N.B. For any $P$, the set $ofn(P)$ is computable because the set of processes that, according to the definition of $ofn_i(-)$, are relevant to determine $ofn(P)$ is finite up to $\equiv_\alpha$ and renaming of revealed bound names (arising in the cases for $(\nu n)P$ and $m(n).P$).

The notion of "observable name" is less syntactical than the one of "free name", and more consistent with our intended structural model, where recursively defined processes are seen as certain infinite trees. For example, given $P \triangleq (\mathbf{rec}\, \mathcal{X}(n).a\langle a\rangle.\mathcal{X}[n])[p]$, the name $p$ is free in (the syntax of) $P$, but certainly not observable in the infinite process $a\langle a\rangle.a\langle a\rangle.\cdots$ that $P$ denotes. The set of observable names of a process is preserved by unfolding of recursive processes, and thus also by structural congruence. This point is important, because structural congruence plays a central role in the semantic of spatial formulas, and the logic should not distinguish processes that just differ on free but non-observable names. We can also verify that for all processes $P$, $ofn(P) \subseteq fn(P)$.

Structural congruence expresses basic identities on the structure of processes:

**Definition 1.3 (Structural congruence).** *Structural congruence $\equiv$ is the least congruence relation on processes such that*

$$
\begin{array}{ll}
P \equiv_\alpha Q \Rightarrow P \equiv Q & \text{(Struct Alpha)} \\
P \mid \mathbf{0} \equiv P & \text{(Struct Par Void)} \\
P \mid Q \equiv Q \mid P & \text{(Struct Par Comm)} \\
P \mid (Q \mid R) \equiv (P \mid Q) \mid R & \text{(Struct Par Assoc)} \\
N + T \equiv T + N & \text{(Struct Cho Comm)} \\
N + (T + U) \equiv (N + T) + U & \text{(Struct Cho Assoc)} \\
n \notin \mathit{ofn}(P) \Rightarrow P \mid (\nu n)Q \equiv (\nu n)(P \mid Q) & \text{(Struct Res Par)} \\
(\nu n)\mathbf{0} \equiv \mathbf{0} & \text{(Struct Res Void)} \\
(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P & \text{(Struct Res Comm)} \\
(\mathbf{rec}\ \mathcal{X}(\overline{n}).P)[\overline{p}] \equiv P\{\overline{n}\leftarrow\overline{p}\}\{\mathcal{X}\leftarrow(\mathbf{rec}\ \mathcal{X}(\overline{n}).P)\} & \text{(Struct Rec Unfold)}
\end{array}
$$

The behavior of processes is defined by a relation of reduction that captures the computations that a process may perform by itself. To observe the communication flow between a process and its environment, we then introduce a relation of commitment.

**Definition 1.4 (Reduction).** Reduction *$(P \to Q$ ) is defined as follows:*

$$
\begin{array}{ll}
m\langle n\rangle.Q + N \mid m(p).P + T \to Q \mid P\{p\leftarrow n\} & \text{(Red React)} \\
Q \to Q' \Rightarrow P \mid Q \to P \mid Q' & \text{(Red Par)} \\
P \to Q \Rightarrow (\nu n)P \to (\nu n)Q & \text{(Red Res)} \\
P \equiv P',\, P' \to Q',\, Q' \equiv Q \Rightarrow P \to Q & \text{(Red Struct)}
\end{array}
$$

Commitment coincides with the standard relation of labeled transition for the $\pi$-calculus ([25]), except that "bound output" and "bound input" labels are omitted. It turns out that bound output and bound input can be expressed in the logic from more primitive observations. Thus, a labelling action is either $\tau$, an input $m(n)$, or an output $m\langle n\rangle$.

**Definition 1.5 (Commitment).** Commitment *$(P \stackrel{\alpha}{\to} Q)$ is defined as follows:*

$$
\begin{array}{ll}
P \to Q \Rightarrow P \stackrel{\tau}{\to} Q & \text{(Com Red)} \\
m,n \notin \overline{p} \Rightarrow (\nu\overline{p})(m\langle n\rangle.Q + N \mid P) \stackrel{m\langle n\rangle}{\to} (\nu\overline{p})(Q \mid P) & \text{(Com Output)} \\
m,n \notin \overline{p} \Rightarrow (\nu\overline{p})(m(q).Q + N \mid P) \stackrel{m(n)}{\to} (\nu\overline{p})(Q\{q\leftarrow n\} \mid P) & \text{(Com Input)} \\
P \equiv P',\, P' \stackrel{\alpha}{\to} Q',\, Q' \equiv Q \Rightarrow P \stackrel{\alpha}{\to} Q & \text{(Com Struct)}
\end{array}
$$

## 2 Logic

In this section, we present the syntax and semantics of the logic, following closely the scheme of [4]; essentially, adjuncts are removed, and behavioral modalities added. Formulas $(A,B,C)$ are built from pure names in $\Lambda$, name variables in $\mathcal{V}$ $(x,y,z)$, and propositional variables in $\mathcal{X}$ $(X,Y,Z)$ as defined in Fig. 1 (we use the metavariable $\eta$ to denote a name or name variable).

The set of logical operators includes propositional, spatial, and temporal operators, first-order quantification, freshness quantification, and recursive formulas. Boolean

$$
\begin{array}{lll|l}
\mathbf{T} & \text{(True)} & [\![\mathbf{T}]\!]_v & \triangleq \mathcal{P} \\
\eta = \eta' & \text{(Equality)} & [\![n = m]\!]_v & \triangleq \text{if } n = m \text{ then } \mathcal{P} \text{ else } \emptyset \\
\neg A & \text{(Negation)} & [\![\neg A]\!]_v & \triangleq \mathcal{P} \setminus [\![A]\!]_v \\
A \wedge B & \text{(Conjunction)} & [\![A \wedge B]\!]_v & \triangleq [\![A]\!]_v \cap [\![B]\!]_v \\
\mathbf{0} & \text{(Void)} & [\![\mathbf{0}]\!]_v & \triangleq \{P \mid P \equiv \mathbf{0}\} \\
A \mid B & \text{(Composition)} & [\![A \mid B]\!]_v & \triangleq \{P \mid \exists Q, R.\ P \equiv Q \mid R \\
 & & & \quad \text{and } Q \in [\![A]\!]_v \text{ and } R \in [\![B]\!]_v\} \\
\eta \circledR A & \text{(Revelation)} & [\![n \circledR A]\!]_v & \triangleq \{P \mid \exists Q.\ P \equiv (\nu n)Q \text{ and } Q \in [\![A]\!]_v\} \\
\forall x.A & \text{(Name quantification)} & [\![\forall x.A]\!]_v & \triangleq \bigcap_{n \in \Lambda} [\![A\{x \leftarrow n\}]\!]_v \\
\text{И}x.A & \text{(Fresh quantification)} & [\![\text{И}x.A]\!]_v & \triangleq \bigcup_{n \notin fn^v(A)} ([\![A\{x \leftarrow n\}]\!]_v \setminus \{P \mid n \in fn(P)\}) \\
\alpha.A & \text{(Action)} & [\![\alpha.A]\!]_v & \triangleq \{P \mid \exists Q.\ P \xrightarrow{\alpha} Q \text{ and } Q \in [\![A]\!]_v\} \\
X & \text{(Propositional variable)} & [\![X]\!]_v & \triangleq v(X) \\
\nu X.A & \text{(Greatest fixpoint)} & [\![\nu X.A]\!]_v & \triangleq \bigcup\{\Psi \in \mathbb{P} \mid \Psi \subseteq [\![A]\!]_{v[X \leftarrow \Psi]}\}
\end{array}
$$

**Fig. 1.** Syntax and Semantics of the Logic

connectives and name equality are interpreted in the standard way. The basic spatial connectives correspond to the static operators of the $\pi$-calculus. The formula $\forall x.A$ denotes quantification over all names in $\Lambda$. The formula $\text{И}x.A$ expresses fresh name quantification: a process satisfies $\text{И}x.A$ if for some fresh (in the process and formula) name $n$, it satisfies $A\{x \leftarrow n\}$. The formula $\alpha.A$ is satisfied by all processes that after performing action $\alpha$ can evolve to a process that satisfy $A$.

In the formulas $\forall x.A$, $\text{И}x.A$, and $\nu X.A$ the distinguished occurrences of $x$ and $X$ are binding, with scope the formula $A$. In a formula $\nu X.A$, we require $A$ to be *monotonic in* $X$, that is, every free occurrence of the propositional variable $X$ in $A$ occurs under an even number of negations. The connectives $\vee$, $\exists$, $\Rightarrow$, and $\mu X.A$ are definable as usual.

The relation of $\alpha$-congruence $\equiv_\alpha$ is defined on formulas in the standard way (safe renaming of bound variables). Given a formula $A$, the sets $fn(A)$ of free names of $A$, $fv(A)$ of free 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 given in a domain of Psets, following closely the approach of [4]. A Pset is a set of processes that is closed under $\equiv$ and has finite support. The support of a Pset is a finite set of names; intuitively, the set of names that are relevant for the property (*cf.*, the free names of a formula). So a Pset is closed under transposition of names out of its support. Recall that a *name permutation* ($\rho$) is a bijective name substitution. As a special case, we consider *name transpositions* ($\tau$), writing $\{m \leftrightarrow n\}$ for the transposition of $m$ and $n$, that is, for the substitution that assigns $m$ to $n$ and $n$ to $m$. For any finite set of names $N$, we say that a name permutation $\rho$ *fixes* $N$ if $\rho(n) = n$ for all $n \in N$. We denote by $\mathbb{R}_N$ the set of all name permutations that fix $N$.

**Definition 2.1 (PSet [4]).** *A* property set *is a set of processes* $\Psi$ *such that*

1. *For all processes* $Q$*, if* $P \in \Psi$ *and* $P \equiv Q$ *then* $Q \in \Psi$.
2. *Exists 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 collection of all Psets. The denotation of a formula $A$ is given by a Pset $[\![A]\!]_v$, with respect to a valuation $v$ that assigns to each propositional variable free in the formula $A$ a Pset in $\mathbb{P}$, defined in Fig. 1. Every Pset $\Phi \in \mathbb{P}$ has a least support [14, 4], denoted by $supp(\Phi)$. 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

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

Hence $fn^v(A)$ is almost $fn(A)$, except that we take $fn(X) = supp(v(X))$ for any $X \in fpv(A)$, so that $fn^v(A) = fn(A)$ for any closed formula $A$. $fn^v(A)$ is used in the semantic clause for the fresh name quantifier, where the selected quantification witness must be fresh for the property set denoted by a formula that may contain free occurrences of propositional variables.

The denotation mapping $[\![-]\!]_v$ satisfies certain fundamental properties, collected in the next Proposition 2.2 and Proposition 2.3. In Proposition 2.2 we refer to transposition of Psets and valuations: if $\Phi$ is a Pset (supported by $N$), then $\tau(\Phi) \triangleq \{\tau(P) \mid P \in \Phi\}$ is also a Pset (supported by $\tau(N)$). We can also define the action of transpositions on valuations as follows: when $v$ is a valuation, $\tau(v)$ is the valuation with the same domain as $v$ and defined by $\tau(v)(X) \triangleq \tau(v(X))$, for all $X \in \mathcal{X}$ in the domain of $v$.

**Proposition 2.2.** *Let A be a closed formula, and v a valuation for A. Then*

1. $[\![A]\!]_v \in \mathbb{P}$ *with* $supp([\![A]\!]_v) \subseteq fn^v(A)$.
2. *For all transpositions* $\tau$, $\tau([\![A]\!]_v) = [\![\tau(A)]\!]_{\tau(v)}$.
3. *(Gabbay-Pitts) Let M be a finite set of names such that* $fn^v(A) \cup fn(P) \subseteq M$. *If* $P \in [\![A\{x \leftarrow p\}]\!]_v$ *for some* $p \notin M$, *then* $P \in [\![A\{x \leftarrow p\}]\!]_v$ *for all* $p \notin M$.

**Proposition 2.3.** *Let A be a formula monotonic in X and v a valuation for the formula* $\nu X.A$. *Let* $\phi$ *be the mapping* $\mathbb{P} \to \mathbb{P}$ *defined by* $\phi(s) \triangleq [\![A]\!]_{v[X \leftarrow s]}$.

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

Proposition 2.2 is proved as Theorem 4.2.1 in [4]. Proposition 2.3 collects some results about fixpoints that carry over to the domain of Psets; (3) is the "reduction lemma" [26].

## 3 Expressiveness

We have already discussed how spatial properties reflect an enhanced observational power when compared with behavioral properties. However, spatial properties are expected to be invariant under a natural notion of structural identity; in turn, structural identity is expected to be close to structural congruence [16, 24]. For example, the processes $m\langle n\rangle \mid p\langle n\rangle$ and $m\langle n\rangle.p\langle n\rangle + p\langle n\rangle.m\langle n\rangle$ are equivalent with respect to the standard strong bisimulation semantics, but are distinguished by the formula $\neg\mathbf{0} \mid \neg\mathbf{0}$, which holds of systems constructed from at least two separate non-void parallel components. Hence, these processes, although strongly bisimilar, are not logically equivalent: the logical equivalence relation $=_L$ induced by a logic on a set of processes is given by defining $P =_L Q$ whenever for any closed formula $A$, $P \in [\![A]\!]$ if and only if $Q \in [\![A]\!]$.

Conversely, the processes $(\mathbf{rec}\ \mathcal{X}.n\langle m\rangle.\mathcal{X})$ and $(\mathbf{rec}\ \mathcal{X}.n\langle m\rangle.n\langle m\rangle.\mathcal{X})$ are strongly bisimilar, and in fact cannot be distinguished by any formula of the logic: both processes denote the same single-threaded behavior. However, they are not structurally congruent. In this section, we discuss the relation between the equivalence induced by the logic and some process equivalences, and conclude that logical equivalence is strictly coarser than structural congruence, and strictly finer that strong bisimulation. Logical equivalence can be equationally characterized by modularly extending structural congruence as defined in Definition 1.3 with two natural principles: we call *extended structural congruence* to the resulting congruence. Extended structural congruence is decidable, and plays a useful role in the model-checker presented in Section 4.

**Definition 3.1.** Extended structural congruence $\equiv^e$ *is the least congruence relation on processes generated by the axioms of structural congruence in Definition 1.3 and the following two axioms*

$$\mathcal{X} \text{ guarded in } Q,\ P \equiv^e Q\{\mathcal{X}\leftarrow(\overline{q})P\} \Rightarrow P \equiv^e (\mathbf{rec}\ \mathcal{X}(\overline{q}).Q)[\overline{q}] \qquad \text{(Struct Rec Solve)}$$
$$\alpha.P + \alpha.P \equiv^e \alpha.P \qquad \text{(Struct Cho Abs)}$$

Our results about $\equiv^e$ build on a characterization of $\equiv^e$ in terms of *structural bisimulations*. Natural notions of structural bisimulation have been defined [24, 16], following the usual coinductive pattern of progressive observation of process commitments. For our purposes we find it more convenient to define structural bisimulations on representations of processes based on finite systems of equations. This choice supports a compact representation for structural bisimulations, and brings several technical simplifications.

**Definition 3.2.** *An* equation *(defining $\mathcal{X}$) has the form $\mathcal{X}[\overline{n}] \doteq P$ where $\mathcal{X}$ is a process variable, and $P$ is a process. A* system *is a pair $\mathcal{S} = (\mathcal{X}[\overline{m}], S)$ where $\mathcal{X}$ is a process variable (the* root *of $\mathcal{S}$), and $S$ is a finite set of equations, such that every process variable appearing in $\mathcal{S}$ is uniquely defined by some equation in $S$.*

The *domain* $\mathfrak{D}(\mathcal{S})$ of a system $\mathcal{S}$ is the set of all process variables defined in $\mathcal{S}$. We write $na(\mathcal{S})$ for the set of names that occur in the equations of $\mathcal{S}$. If $S$ is a set of equations and $\mathcal{X}[\overline{q}] \doteq Q \in S$, we denote by $S(\mathcal{X})[\overline{p}]$ the process $Q\{\overline{q}\leftarrow\overline{p}\}$. A system $\mathcal{S}$ is *expanded* if all of its equations have the general form

$$\mathcal{X}[\overline{m}] \doteq (\nu\overline{n})(\Sigma_{i_1}\alpha^1_{i_1}.\mathcal{X}^1_{i_1}[\overline{m}_{i_1}] \mid \cdots \mid \Sigma_{i_k}\alpha^k_{i_k}.\mathcal{X}^k_{i_k}[\overline{m}_{i_k}])$$

Since choice is associative and commutative, we denote by $\Sigma_i\alpha_i.P_i$ a choice $\alpha_1.P_1 + \cdots + \alpha^i_n.P^i_n$. We can now define:

**Definition 3.3.** *Let $\mathcal{S}_P = (\mathcal{X}_0, S_P)$ and $\mathcal{S}_Q = (\mathcal{Y}_0, S_Q)$ be two expanded systems, where $M \triangleq na(\mathcal{S}_P) \cup na(\mathcal{S}_Q)$. A* structural bisimulation *for $\mathcal{S}_P$ and $\mathcal{S}_Q$ is a relation $\approx$ such that*

1. *$\approx\ \subseteq \{(\mathcal{X}[\overline{n}], \mathcal{Y}[\overline{m}]) \mid \mathcal{X} \in \mathfrak{D}(\mathcal{S}_P), \mathcal{Y} \in \mathfrak{D}(\mathcal{S}_Q), \overline{m}, \overline{n}$ names $\}$ and $\mathcal{X}_0 \approx \mathcal{Y}_0$;*
2. *If $\mathcal{X}[\overline{p}] \approx \mathcal{Y}[\overline{q}]$ then there are $\overline{m}, \overline{N}, \overline{T}$ with $\overline{m} \cap M = \emptyset$ and $\#N = \#T$, such that $\mathcal{S}_P(\mathcal{X})[\overline{p}] \equiv (\nu\overline{m})\overline{N}$, $\mathcal{S}_Q(\mathcal{Y})[\overline{q}] \equiv (\nu\overline{m})\overline{T}$, and for all $i = 1, \ldots, \#N$ and $\alpha$ such that $bn(\alpha) \notin \overline{m} \cup M$:*
   *If $N_i \xrightarrow{\alpha} \mathcal{X}'[\overline{p}']$ for some $\mathcal{X}', \overline{p}'$ then exists $\mathcal{Y}'$ such that $T_i \xrightarrow{\alpha} \mathcal{Y}'[\overline{q}']$ and $\mathcal{X}'[\overline{p}'] \approx \mathcal{Y}'[\overline{q}']$;*
   *If $T_i \xrightarrow{\alpha} \mathcal{Y}'[\overline{q}']$ for some $\mathcal{Y}', \overline{q}'$ then exists $\mathcal{X}'$ such that $N_i \xrightarrow{\alpha} \mathcal{X}'[\overline{p}']$ and $\mathcal{X}'[\overline{p}'] \approx \mathcal{Y}'[\overline{q}']$.*

$$
\begin{array}{ll}
|\mathbf{0}|^N_{\overline{q}} & \triangleq (\mathcal{X}[\overline{q}], \{\mathcal{X}[\overline{q}] \doteq \mathbf{0}\}) \\
|(\nu n)P|^N_{\overline{q}} & \triangleq (\mathcal{X}[\overline{q}], \{\mathcal{X}[\overline{q}] \doteq (\nu n\overline{m})\overline{R}\} \cup S_P) \\
& \quad |P|^N_{\overline{q}n} = (\mathcal{Y}[\overline{q}n], S_P) \\
& \quad S_P(\mathcal{Y})[\overline{q}n] = (\nu\overline{m})\overline{R},\, n \in ofn(P) \\
|(\nu n)P|^N_{\overline{q}} & \triangleq (\mathcal{Y}[\overline{q}], S_P),\ |P|^N_{\overline{q}} = (\mathcal{Y}[\overline{q}], S_P),\, n \notin ofn(P) \\
|P \mid Q|^N_{\overline{q}} & \triangleq (\mathcal{X}[\overline{q}], \{\mathcal{X}[\overline{q}] \doteq (\nu\overline{nm})(\overline{R} \mid \overline{S})\} \cup S_P \cup S_Q) \\
& \quad |P|^N_{\overline{q}} = (\mathcal{Y}[\overline{q}], S_P),\ S_P(\mathcal{Y})[\overline{q}] = (\nu\overline{n})\overline{R} \\
& \quad |Q|^N_{\overline{q}} = (\mathcal{Z}[\overline{q}], S_Q),\ S_Q(\mathcal{Z})[\overline{q}] = (\nu\overline{m})\overline{S} \\
|\Sigma\alpha_i.P_i|^N_{\overline{q}} & \triangleq (\mathcal{X}[\overline{q}], \{\mathcal{X}[\overline{q}] \doteq \Sigma_i\alpha_i.\mathcal{Y}_i[\overline{qp_i}]\} \cup \bigcup_i S_{P_i}) \\
& \quad |P_i|^N_{\overline{qp_i}} = (\mathcal{Y}_i[\overline{qp_i}], S_{P_i}), \overline{p}_i = bn(\alpha_i) \\
|(\mathbf{rec}\,\mathcal{Y}(\overline{q}).P)[\overline{p}]|^N_{\overline{r}} & \triangleq (\mathcal{X}[\overline{r}], \{\mathcal{X}[\overline{r}] \doteq S(\mathcal{Y})[\overline{rp}]\} \cup S \downarrow \mathcal{Y}) \\
& \quad |P\{\mathcal{Y} \leftarrow (\overline{q})\mathcal{X}[\overline{rq}]\}|^N_{\overline{rq}} = (\mathcal{Y}[\overline{rq}], S) \\
|\mathcal{Y}[\overline{p}]|^N_{\overline{q}} & \triangleq (\mathcal{Y}[\overline{p}], \emptyset)
\end{array}
$$

**Fig. 2.** Systems from processes.

In clause 2, $\overline{N}$ and $\overline{T}$ denote sequences of *guarded* processes, so that each $N_i$ and $T_i$ denotes a (possibly singleton) choice process, and we write $\#\overline{N}$ for the length of the sequence $\overline{N}$. We write $\mathcal{S} \approx \mathcal{S}'$ to state that there is a structural bisimulation for $\mathcal{S}$ and $\mathcal{S}'$.

**Definition 3.4.** *For any process $P$ such that $fn(P) \subseteq \overline{q} \cup N$ we define a system $|P|^N_{\overline{q}}$ as specified in Fig. 2.*

We denote by $|P|$ the system $|P|^{fn(P)}_{\emptyset} = (\mathcal{Z}, S)$. When constructing $|P|^N_{\overline{q}}$ we require $N \cap \overline{q} = \emptyset$ and, more generally, that the bound names introduced in the cases for restriction and input are distinct, and different from free names, using $\equiv_\alpha$ on $P$ if needed. In the case for the recursive process, by $S \downarrow \mathcal{Y}$ we denote the set of equations obtained from $S$ by applying the substitution $\{\mathcal{Y} \leftarrow (\overline{rq})S(\mathcal{Y})[\overline{rq}]\}$ to every equation where $\mathcal{Y}$ appears unguarded. We can then verify that, for any $P$, the system $|P|$ is expanded, and unique up to the choice of bound names and process variables.

*Example 3.5.* Let $P \triangleq (\mathbf{rec}\,\mathcal{Y}.a(m).(\mathcal{Y} \mid b\langle m\rangle.\mathbf{0}))$. Then $fn(P) = \{a, b\}$, and:
(1) $|\mathcal{Z}' \mid b\langle m\rangle.\mathbf{0}|_m = (\mathcal{Z}_1[m], \{\mathcal{Z}_1[m] \doteq \mathcal{Z}' \mid b\langle m\rangle.\mathcal{Z}_2, \mathcal{Z}_2[m] \doteq \mathbf{0}\}$;
(2) $|a(m).(\mathcal{X} \mid b\langle m\rangle.\mathbf{0})|_\emptyset = (\mathcal{Z}', \{\mathcal{Z}' \doteq a(m).\mathcal{Z}_1[m], \mathcal{Z}_1[m] \doteq \mathcal{Z}' \mid b\langle m\rangle.\mathcal{Z}_2, \mathcal{Z}_2[m] \doteq \mathbf{0}\})$;
(3) $|P| = (\mathcal{Z}_0, \{\mathcal{Z}_0 \doteq a(m).\mathcal{Z}_1[m], \mathcal{Z}_1[m] \doteq a(m').\mathcal{Z}_1[m'] \mid b\langle m\rangle.\mathcal{Z}_2[m], \mathcal{Z}_2[m] \doteq \mathbf{0})$.

A *solution* for the system $\mathcal{S}$ is an assignment of an abstraction $(\overline{q}_i)Q_i$ of appropriate arity to each process variable $\mathcal{X}_i$ in $\mathfrak{D}(\mathcal{S})$ such that $Q_i \equiv^e P_i\{\mathcal{X}_1 \leftarrow (\overline{q_1})Q_1\} \ldots \{\mathcal{X}_n \leftarrow (\overline{q_n})Q_n\}$ for every equation $\mathcal{X}_i[\overline{q_i}] \doteq P_i$ of $\mathcal{S}$. We can prove

**Lemma 3.6.** *There is a solution $s$ for $|P|^N_{\overline{q}} = (\mathcal{X}[\overline{q}], S)$ with $s(\mathcal{X}) \equiv^e (\overline{q})P$.*

**Lemma 3.7.** *For all processes $P$ and $Q$, if $P \equiv^e Q$ then $|P| \approx |Q|$.*

*Proof.* By induction on the derivation of $P \equiv^e Q$, and construction of appropriate structural bisimulations.

**Lemma 3.8.** *For all processes $P$ and $Q$, if $|P| \approx |Q|$ then $P \equiv^e Q$.*

$$
\begin{array}{lll}
[P]_0 & \triangleq \mathbf{T} & \\
[\mathbf{0}]_{k+1} & \triangleq \mathbf{0} & \\
[P \mid Q]_{k+1} & \triangleq [P]_{k+1} \mid [Q]_{k+1} & \\
[(\nu q)P]_{k+1} & \triangleq \text{И}x.(\copyright x \wedge [P]_{k+1}\{q \leftarrow x\}) & \text{if } q \in ofn(P) \\
[(\nu q)P]_{k+1} & \triangleq [P]_{k+1} & \text{if } q \notin ofn(P) \\
[\Sigma \overline{G}]_{k+1} & \triangleq GG \wedge Out^k_{\overline{G}} \wedge In^k_{\overline{G}} \wedge ActI_{\overline{G}} \wedge ActO_{\overline{G}} & \\
[(\mathbf{rec}\ X(\overline{q}).P)[\overline{p}]]_k & \triangleq [P\{\overline{q} \leftarrow \overline{p}\}\{X \leftarrow (\mathbf{rec}\ X(\overline{q}).P)\}]_k &
\end{array}
\qquad
\begin{array}{ll}
\mathbf{1} & \triangleq \neg(\neg\mathbf{0} \mid \neg\mathbf{0}) \wedge \neg\mathbf{0} \\
NR & \triangleq \neg \text{И}x.\copyright x \\
GG & \triangleq \mathbf{1} \wedge NR \\
\copyright\eta & \triangleq \neg\eta \circledR \mathbf{T} \\
[\alpha].A & \triangleq \neg\alpha.\neg A
\end{array}
$$

$$
\begin{array}{ll}
Out^k_{\overline{G}} & \triangleq \bigwedge_{m\langle n\rangle.Q \in \overline{G}} (m\langle n\rangle.|Q|_k \wedge [m\langle n\rangle].Out^k_{m\langle n\rangle,\overline{G}}) \\
Out^k_{m\langle n\rangle,\overline{G}} & \triangleq \bigvee_{m\langle n\rangle.Q \in \overline{G}} |Q|_k \\
In^k_{\overline{G}} & \triangleq \bigwedge_{m(q).Q \in \overline{G}} (\text{И}x.m(x).(|Q|_k\{q \leftarrow x\}) \wedge \text{И}x.[m(x)].(In^k_{m(q),\overline{G}}\{q \leftarrow x\})) \\
In^k_{m(q),\overline{G}} & \triangleq \bigvee_{m(q).Q \in \overline{G}} |Q|_k \\
ActI_{\overline{G}} & \triangleq \forall x.\forall y.[x(y)].\bigvee_{n(q).Q \in \overline{G}} x = n \\
ActO_{\overline{G}} & \triangleq \forall x.\forall y.[x\langle y\rangle].\bigvee_{n\langle m\rangle.Q \in \overline{G}} (x = n \wedge y = m)
\end{array}
$$

**Fig. 3.** Construction of Bounded Characteristic Formulas

*Proof.* From $|P| \approx |Q|$ we build another system $\mathcal{Z} = (Z_0, Z)$ such that $|P| \approx \mathcal{Z} \approx |Q|$. Then we show that any solution $s$ for $\mathcal{Z}$ gives rise to solutions for $|P|$ and $|Q|$, such that $P \equiv^e s(Z_0) \equiv^e Q$, and conclude by transitivity of $\equiv^e$. The proof follows the pattern of completeness proofs for equational characterizations of "rational trees" (*e.g.* [1]); but the need to cope with binding operators (with scope extrusion) and structural congruence raise some additional challenges.

We thus conclude:

**Proposition 3.9.** *For all processes $P$ and $Q$, $|P| \approx |Q|$ if and only if $P \equiv^e Q$.*

Moreover, since the existence of a structural bisimulation for $|P|$ and $|Q|$ just depends on the inspection of a number of pairs that is finite up to name permutations fixing $na(|P|) \cup na(|Q|)$, we have:

**Lemma 3.10.** *For all processes $P$ and $Q$, it is decidable to check $P \equiv^e Q$.*

A main result of this section is then the following property.

**Proposition 3.11.** *For all processes $P$ and $Q$, $P =_{\mathrm{L}} Q$ if and only if $P \equiv^e Q$.*

The proof makes essential use of the characterization of extended structural congruence in terms of structural bisimulation, and requires some build up, namely, the definition of (bounded) characteristic formulas. These formulas characterize processes up to a certain "depth", modulo extended structural congruence.

**Definition 3.12 (Characteristic Formulas).** *Given a process $P$ and $k \geq 0$, we define a formula $[P]_k$ as specified in Fig. 3.*

*N.B. When $\overline{G}$ is a multiset of guarded processes, we use the notation $\Sigma\overline{G}$ to denote the choice of the elements of $\overline{G}$,* e.g., *$\Sigma\{a(p).P, a(q).Q, b\langle r\rangle.P\}$ denotes the process $a(p).P + a(q).Q + b\langle r\rangle.P$.*

Notice that $[-]_k$ is well-defined by induction on the pairs $(k, s(P))$ (ordered lexicographically), where $s(P)$ is the number of process operators in $P$ that do not occur behind a prefix. Intuitively, the formula $\mathbf{1}$ is satisfied precisely by non-void processes that cannot be split in two non-void parts, that is $P$ satisfies $\mathbf{1}$ if and only if $P$ is single-threaded. The formula $NR$ is satisfied by those processes that do not contain a "true" restricted name at the toplevel, that is $P$ satisfies $NR$ if and only if for all $n$ and $P'$ such that $P \equiv (\nu n)P'$ it is always the case that $n \notin ofn(P')$. Recall that $P$ satisfies ©$n$ if and only if $n \in ofn(P)$. So, a process $P$ satisfies $GG$ if and only if $P$ is structurally congruent to a choice process. The intent of the formula $ActO_{\overline{G}}$ (respectively $ActI_{\overline{G}}$) is to characterize what output (respectively input) actions a choice process offers, while $Out^k_{\overline{G}}$ and $Out^k_{\alpha,\overline{G}}$ (resp. $In^k_{\overline{G}}$ and $In^k_{\alpha,\overline{G}}$) characterize the effects of output (resp. input) actions.

We can also define a notion of finite approximations to structural bisimulations, along standard lines, and write $\mathcal{S} \approx_k \mathcal{S}'$ if there is a structural bisimulation of depth $k$ for $\mathcal{S}$ and $\mathcal{S}'$. We then have

**Lemma 3.13.** *If $\mathcal{S} \approx_k \mathcal{S}'$ for all $k \geq 0$, then $\mathcal{S} \approx \mathcal{S}'$.*

We can then show that our definition of bounded characteristic formulas is correct, in the sense of the following Lemma:

**Lemma 3.14.** *For all $k \geq 0$ and processes $P, Q$ we have*

1. *$P \in [\![ [P]_k ]\!]$.*
2. *If $Q \in [\![ [P]_k ]\!]$ then $|P| \approx_k |Q|$.*

*Proof.* Induction on $k$.

**Lemma 3.15.** *For all processes P and Q, if $P =_{\mathrm{L}} Q$ then $P \equiv^e Q$.*

*Proof.* Consider the formulas $[P]_k$ for all $k \geq 0$. By Lemma 3.14(1), we have $P \in [\![ [P]_k ]\!]$, for all $k \geq 0$. Since $P =_{\mathrm{L}} Q$, we have $Q \in [\![ [P]_k ]\!]$, for all $k \geq 0$. By Lemma 3.14(2), we have $|P| \approx_k |Q|$, for all $k \geq 0$. By Lemma 3.13, $|P| \approx |Q|$. By Lemma 3.8, $P \equiv^e Q$.

**Lemma 3.16.** *For all processes P and Q, if $P \equiv^e Q$ then $P =_{\mathrm{L}} Q$.*

*Proof.* We first prove, by induction on the structure of formulas, that satisfaction is closed under $\equiv_e$ (*cf.*, Proposition 2.2(1)). The statement then follows.

This concludes the proof of Proposition 3.11. Since the modalities introduced for early strong bisimulation in [21] are expressible in the logic, we also have

**Proposition 3.17.** *The equivalence relation induced by the logic on the set of all processes is strictly included in early strong bisimulation.*

## 4 Model Checking

In this section, we present a model-checking algorithm for the logic of Section 2. It is interesting to notice that the choice of a small set of logical primitives and the adoption

$$
\begin{array}{ll}
C(P,v,\mathbf{T}) & \triangleq \mathbf{true} \\
C(P,v,n=m) & \triangleq Test(m=n) \\
C(P,v,\neg A) & \triangleq \mathbf{not}\; C(P,v,A) \\
C(P,v,A \wedge B) & \triangleq C(P,v,A)\; \mathbf{and}\; C(P,v,B) \\
C(P,v,\mathbf{0}) & \triangleq Test(P \equiv \mathbf{0}) \\
C(P,v,A \mid B) & \triangleq \mathbf{Exists}\; Q,R.(Q,R) \in Comp(P)\; \mathbf{and}\; C(Q,v,A)\; \mathbf{and}\; C(R,v,B) \\
C(P,v,n\text{®}A) & \triangleq \mathbf{Exists}\; Q.\; Q \in Res(n,P)\; \mathbf{and}\; C(Q,v,A) \\
C(P,v,\alpha.A) & \triangleq \mathbf{Exists}\; Q.\; Q \in Red(\alpha,P)\; \mathbf{and}\; C(Q,v,A) \\
C(P,v,\forall x.A) & \triangleq C(P,v,A\{x \leftarrow new(fn(P) \cup fs^v(A))\})\; \mathbf{and} \\
 & \quad \mathbf{All}\; n \in fn(P) \cup fs^v(A).C(P,v,A\{x \leftarrow n\}) \\
C(P,v,\text{И}x.A) & \triangleq C(P,v,A\{x \leftarrow new(fn(P) \cup fs^v(A))\}) \\
C(P,v,X) & \triangleq \mathbf{let}\; (S,\nu X.A) = v(X)\; \mathbf{in\; if}\; In(P,v,X)\; \mathbf{then\; true\;\; else}\; C(P,v(X+P),A) \\
C(P,v,\nu X.A) & \triangleq C(P,v[X \leftarrow (\{P\},\nu X.A)],A) \\
In(P,v,X) & \triangleq \mathbf{let}\; (S,A) = v(X)\; \mathbf{in\; Exists}\; Q \in S\;\; \mathbf{and}\; Test(P \equiv^e_{fs^v(A)} Q)
\end{array}
$$

**Fig. 4.** Model-checking algorithm

of the Pset-based semantic foundation allows us to present in a rather succinct way a complete model-checker for a quite expressive $\pi$-calculus and logic.

The algorithm is specified by the boolean-valued procedure $C(P,v,A)$ defined in Figure 4. In every procedure call $C(P,v,A)$, $P$ is a process, $A$ is a formula, and $v$ is a *syntactic valuation*, whose role is fully explained below. The boolean connectives are handled by the model-checker as expected. Spatial and behavioral connectives are handled by the set of auxiliary procedures $Comp(-)$, $Res(-,-)$ and $Red(-,-)$ introduced in Lemma 4.1. The purpose of these algorithms is to decompose processes up to structural congruence, and compute the set of commitments a given process may present.

**Lemma 4.1.** *For any process $P$ we have*

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

N.B. We have $Res(n,P) = \emptyset$ if and only if $n \in ofn(P)$. Similar results for calculi with replication (the $\pi$-calculus and the ambient calculus) have been presented in [13, 10, 8]. However, the property stated in Lemma 4.1(1a) does not hold for process calculi with replication where the principle $!P \equiv P \mid\, !P$ holds (*cf.*, [13]).

The cases for the freshness quantifier and the universal quantifier requires the generation of fresh names. Instead of attempting to determine in advance 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 [11]), we rely on Proposition 2.2(3), and in each case pick an *arbitrary* name out of the support (in the sense of Definition 2.1) of the denotation of the formula to be checked. By Proposition 2.2(1), we know that such support can be approximated by the set of free names of the formula to be checked, 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). Generation of fresh names can then be implemented by a choice function that assigns to every finite set of names $M$ a name $new(M) \notin M$: any choice function meeting this specification is acceptable. In fact, no property of the model-checker (*e.g.*, termination) requires fresh names to be generated according to some fixed strategy.

Syntactic valuations are finitary counterparts to the (semantic) valuations defined in Section 2. 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 recursive formula. Intuitively, if $v$ is a syntactic valuation and $v(X) = (S,A)$ then $S$ is a finite approximation to the denotation of the recursive formula $A$.

**Definition 4.2.** *A* syntactic valuation *$v$ is a mapping from a finite sequence $\overline{X}_n$ of propositional variables such that $v(X_i) = (S_i, \nu X_i.A_i)$ for all $i = 1,\dots,n$, where each $S_i$ is a finite set of processes, and each $\nu X_i.A_i$ is a formula with $fpv(A_i) \subseteq \{X_1,\dots,X_{i-1}\}$.*

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

$$fs^v(A) \triangleq fn(A) \cup \bigcup\{fs^v(B) \mid X \in fpv(A) \text{ and } v(X) = (S,B)\}$$

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 $fpv(A) \subseteq \mathfrak{D}(v)$ we write $v[X \leftarrow (S,A)]$ for the extension (not the update) of $v$ with the additional binding $[X \leftarrow (S,A)]$. We use the notation $v(X+P)$ to denote the valuation that results from $v$ by adding the process $P$ to the set-valued component of $v(X)$, *e.g.*, if $v$ is the valuation $w[X \leftarrow (S,A)]w'$ then $v(X+P)$ is the valuation $w[X \leftarrow (S \cup \{P\},A)]w'$.

The algorithm handles fixpoint formulas by appealing to Winskel-Kozen's reduction lemma (Proposition 2.3(3)). The reduction lemma suggests a progressive unfolding strategy for recursive formulas used in many model-checkers for $\mu$-calculus based process logics. However, a main technical difference between the treatment of fixpoints in our algorithm and other proposals concerns the interaction of spatial decomposition of restricted processes, fresh name generation, and recursion. Here, we compute an approximation to the finite support of the denotation of a fixpoint formula (given by $fs^v(A)$), and use this information to stop unfolding it, relying on the fact that if a process $P$ belongs to a Pset $\Psi$ with $supp(\Psi) \subseteq M$, then $\rho(P) \in \Psi$ for all permutations $\rho$ that fix $M$. This approach seem conceptually simpler than other proposals for coping with fresh name generation in model-checkers for $\pi$-calculus logics (*e.g.* [11]), and allows us to keep the description of the algorithm more abstract, and the correctness proofs simpler.

**Definition 4.3.** *Given a finite set of names $M \subset \Lambda$, we define the relation $\equiv^e_M$ on processes by letting $P \equiv^e_M Q$ if and only if there is $\rho \in \mathbb{R}_M$ such that $\rho(P) \equiv^e Q$.*

Since for given $P$ and $Q$, the number of permutations to test is finite, by Lemma 3.10 we conclude that checking $P \equiv^e_M Q$ is decidable. The purpose of the boolean procedure $In(P,v,X)$ at the bottom of Fig. 4 is then to check for the presence of a representative of the equivalence class of $P$ in $\mathcal{P}/\equiv^e_M$ in the current approximation to the denotation of the fixpoint formula that introduced the propositional variable $X$. By Propositions 2.2(2) and 3.9 and our characterisation of $=_{\mathrm{L}}$ in terms of $\equiv^e$ (Proposition 3.11), we know that $Q \equiv^e_M P$ implies that $Q \in [\![\nu X.A]\!]$ if and only if $P \in [\![\nu X.A]\!]$. Notice also that $fs^v(A)$ is only used in the procedure $In(-,-,-)$, in the test for $P \equiv^e_{fs^v(A)} Q$.

In the remainder of this section we establish correctness results for our model-checker. We start by introducing some auxiliary concepts. For any set of processes $S$ and finite set of names $N$, we can define a Pset $Close(S,N) \in \mathbb{P}$ by $Close(S,N) \triangleq \{Q \mid Q \equiv^e \tau(P),\, \tau \in \mathbb{R}_N \text{ and } P \in S\}$. Notice that $Close(S,N)$ contains $S$ and is supported by $N$. Now, for every syntactic valuation $v$, we define a (semantic) valuation $v^\star$ as follows:

**Definition 4.4.** *Given a syntactic valuation $v$, we define a valuation $v^\star$ as follows:*

$$\begin{array}{lll} \emptyset^\star \triangleq \emptyset & & \Phi \triangleq Gfix(\lambda s.S^\star \cup [\![A]\!]_{w^\star[X\leftarrow s]}) \\ w[X\leftarrow(S,\nu X.A)]^\star \triangleq w^\star[X\leftarrow\Phi] & \textit{where} & S^\star \triangleq Close(S,fs^v(\nu X.A)) \end{array}$$

**Proposition 4.5 (Soundness).** *For every $P$, formula $A$ and syntactic valuation $v$ for $A$ we have: (a) If $C(P,v,A) =$ **true** then $P \in [\![A]\!]_{v^\star}$. (b) If $C(P,v,A) =$ **false** then $P \notin [\![A]\!]_{v^\star}$.*

We now show completeness of the model-checking algorithm. To obtain decidability we need to impose some finiteness conditions: we restrict model-checking to a class of bounded processes. 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. Completeness then results from the fact that our model-checker always terminates on bounded processes. We first define reachability:

**Definition 4.6 (Reachability).** *For every $P$ we define the set $Reach(P)$ as follows:*

$$\begin{array}{l} P \in Reach(P) \\ P' \in Reach(P),\, (Q,R) \in Comp(P') \Rightarrow Q \in Reach(P),\, R \in Reach(P) \\ P' \in Reach(P),\, Exists\ n.\ Q \in Res(n,P') \Rightarrow Q \in Reach(P) \\ P' \in Reach(P),\, Exists\ \alpha.\ Q \in Red(\alpha,P') \Rightarrow Q \in Reach(P) \end{array}$$

**Definition 4.7 (Bounded process).** *A process $P$ is* bounded *if for every finite set of names $M$ the set (of equivalence classes) $Reach(P)/\equiv_M$ is finite.*

**Proposition 4.8 (Completeness).** *If $Q$ is bounded and $Q \in [\![A]\!]$ then $C(Q,\emptyset,A) =$ **true**.*

Therefore, after noticing that all tests in the model-checking procedure 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 conclude

**Corollary 4.9.** *Model-checking of bounded processes is decidable.*

Due to spatial reachability, the fact that a process always terminates is not enough to ensure its boundedness: a deadlocked process may contain components which are not bounded when considered in isolation, *e.g.*, the process $(\nu n)(\mathbf{rec}\, \mathcal{X}.n\langle n\rangle.(\mathcal{X} \mid \mathcal{X}))$ is not bounded in the sense of Definition 4.7. However, we can verify that the class of bounded processes includes the class of finite-control processes as defined in [11].

**Proposition 4.10.** *Any finite-control process is bounded.*

## 5 Related Work and Conclusions

We have proposed and studied a logic for the synchronous π-calculus, organized around a small set of spatial and behavioral observations, and including freshness quantifiers and recursive formulas. This logic subsumes existing behavioral logics for π-calculi [21, 12], and can be seen as a fragment of the spatial logic of [4, 3] (in the sense that action modalities can be expressed with the composition adjunct [17]). The semantic foundation for the logic and model-checker presented here builds on the approach developed by Cardelli and the present author in [4], which is in turn based on domains of finitely supported sets of processes and the theory of freshness by Gabbay and Pitts [14].

We have investigated the separation power of the logic, providing sound and complete characterizations of the equivalence (actually the congruence) induced by the logic on processes. These results build on the definition of bounded characteristic formulas for processes, and on some technical results about solutions of equations on π-calculus processes up to extended structural congruence. Expressiveness and separation results for spatial logics for the public ambient calculus have already been investigated by Sangiorgi, Lozes and Hirshckoff [24, 16].

We have also presented a model-checker for the logic, and have shown that model-checking is decidable on a class of bounded processes, that includes the finite-control fragment of the π-calculus. Model-checking the π-calculus against behavioral logics was studied extensively by Dam [11, 12]. Most of the existing work on model-checking for spatial logics focus on the ambient logic, after the first proposal of [6]. The work of Charatonik, Gordon and Talbot on model-checking the Ambient logic against finite-control mobile ambients [8] (where, like done here for the π-calculus, replication is replaced by recursion) seems to be the most related to ours, although it does not address a spatial logic with recursive formulas and with freshness quantifiers.

*Acknowledgements* This research builds on previous joint work with Luca Cardelli. Thanks also to Luís Monteiro, Etienne Lozes, and Daniel Hirshckoff for related discussions. This work was partially funded by FET IST-2001-33100 Profundis.

## References

1. R. Amadio and L. Cardelli. Subtyping recursive types. *ACM Transactions on Programming Languages and Systems*, 15(4):575–631, 1993.
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 II). In *CONCUR 2002 (13th International Conference)*, Lecture Notes in Computer Science. Springer-Verlag, 2002.

4. L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part I). *Information and Computation*, 186(2):194–235, 2003.
5. L. Cardelli, P. Gardner, and G. Ghelli. Manipulating Trees with Hidden Labels. In A. D. Gordon, editor, *Proc. of Foundations of Software Science and Computation Structures (FoSSaCS '03)*, Lecture Notes in Computer Science. Springer-Verlag, 2003.
6. 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.
7. 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.
8. 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.
9. W. Charatonik and J.-M. Talbot. The decidability of model-checking mobile ambients. In D. Metayer, editor, *11th European Symposium on Programming (ESOP 2002)*, number 2305 in Lecture Notes in Computer Science. Springer-Verlag, 2001.
10. S. Dal-Zilio. Spatial Congruence for Ambients is Decidable. In *6th ACS Conference*, number 1961 in Lecture Notes in Computer Science, pages 365–377. Springer-Verlag, 2000.
11. M. Dam. Model checking mobile processes. *Information and Computation*, 129(1):35–51, 1996.
12. 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.
13. J. Engelfriet and T. Gelsema. Multisets and Structural Congruence of the π-calculus with Replication. *Theoretical Computer Science*, (211):311–337, 1999.
14. M. Gabbay and A. Pitts. A New Approach to Abstract Syntax with Variable Binding. *Formal Aspects of Computing*, 13(3–5):341–363, 2002.
15. G. Ghelli and G. Conforti. Decidability of freshness, undecidability of revelation. Technical Report 03–11, Dipartimento di Informatica, Universita di Pisa, 2003.
16. D. Hirschkoff, E. Lozes, and D. Sangiorgi. Separability, Expressiveness and Decidability in the Ambient Logic. In *Proc. LICS*, Copenhagen, Denmark, 2002. IEEE Computer Society.
17. D. Hirschkoff, E. Lozes, and D. Sangiorgi. Minimality results for the spatial logics. In *Proc. of FSTTCS'2003*, number 2914 in Lecture Notes in Computer Science. Springer Verlag, 2003.
18. S. Ishtiaq and P. O'Hearn. BI as an Assertion Language for Mutable Data Structures. In *28th ACM Symp. on Principles of Programming Languages*, 2001.
19. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
20. R. Milner. *Communicating and Mobile Systems: the π-calculus*. CUP, 1999.
21. R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 114:149–171, 1993.
22. U. Montanari and M. Pistore. Pi-Calculus, Structured Coalgebras, and Minimal HD-Automata. In *MFCS: Symp. on Mathematical Foundations of Computer Science*, pages 569–578, 2000.
23. P. O'Hearn and D. Pym. The Logic of Bunched Implications. *The Bulletin of Symbolic Logic*, 5(2):215–243, 1999.
24. D. Sangiorgi. Extensionality and Intensionality of the Ambient Logics. In *28th Annual Symposium on Principles of Programming Languages*, pages 4–13. ACM, 2001.
25. D. Sangiorgi and D. Walker. *The π-calculus: A Theory of Mobile Processes*. CUP, 2001.
26. G. Winskel. A note on model checking the modal ν-calculus. In G. Ausiello, M. Dezani-Ciancaglini, and S. R. D. Rocca, editors, *(ICALP 1989)*, volume 372 of *Lecture Notes in Computer Science*, pages 761–772. Springer-Verlag, 1989.