A Spatial Logic for Concurrency
(Part I)

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

and

Luca Cardelli
Microsoft Research, Cambridge, UK

Version: (June 9, 2003)

We present a logic that can express properties of freshness, secrecy, structure, and behavior of concurrent systems. In addition to standard logical and temporal operators, our logic includes spatial operations corresponding to composition, local name restriction, and a primitive fresh name quantifier. Properties can also be defined by recursion; a central aim of this paper is then the combination of a logical notion of freshness with inductive and coinductive definitions of properties.

1. INTRODUCTION

We present a logic for describing the behavior and spatial structure of concurrent systems. Logics for concurrent systems are certainly not new [22, 14, 29, 16], but the intent to describe spatial properties seems to have arisen only recently. The spatial properties that we consider here are essentially of two kinds: whether a system is composed of two or more identifiable subsystems, and whether a system restricts the use of certain resources to certain subsystems. Previous work [10] has considered also whether a system is composed of named locations; in that case, the notion of spatial structure is particularly natural.

The initial motivation for studying these logics was to be able to specify systems that deal with fresh or secret resources such as keys, nonces, channels, and locations. In previous papers [10, 6], we have found that the spatial properties of process composition and of location structures are fairly manageable. Instead, the properties of restriction are much more challenging, and are closely related to the study of logical notions of freshness [20, 19, 31].

The main aim of this paper is to advance the study of restriction started in [11, 6] and to build a closer correspondence with treatments of freshness [19]. For simplicity, we use a basic process calculus (the asynchronous π-calculus) that includes composition and restriction. We omit locations in this paper because they are easy to add along the lines of [10], and are comparatively easier to handle than composition or restriction. It will become clear that our general approach is fairly insensitive to the details of specific process calculi, and is largely insensitive to the "dynamics" (reduction behavior) of specific calculi. Therefore, it can be easily adapted to other calculi, and perhaps even generalized to process frameworks [21].

A formula in our logic describes a property of a particular part of a concurrent system at a particular time; therefore it is modal in space as well as in time. This dual modality can be captured by standard box and diamond operators, reflecting reachability in space and in time [10, 6]. As a further contribution of this paper, though, we wish to investigate a more general framework akin to the μ-calculus [25], where formulas can be recursive and can subsume box and diamond operators. Moreover, by combining spatial and temporal connectives with recursion, we can then define new and interesting modalities, such as "under an arbitrary number of restrictions". The most technically challenging part of the paper is then the interaction of recursive formulas, already present in [6, 9, 16], with logical notions of freshness, composition, and name restriction, already present in [6, 11].

We now give a brief overview of the constructs of the logic, before moving on to the formal treatment. Let $\mathcal{P}$ be the set of (asynchronous π-calculus) processes. A *property* is a set of processes; a subset of $\mathcal{P}$. A closed formula denotes a property, namely, it denotes the collection of processes satisfying that formula.

• The collection of all properties (which is not quite the powerset of $\mathcal{P}$, as we shall discuss) has the structure of a Boolean Algebra under set inclusion, so we naturally get boolean connectives(we take $\mathbf{F}$, $A \wedge B$ and $A \Rightarrow B$ as primitive).

• The above propositional fragment is extended to predicate logic via a universal quantifier $\forall x.A$. This quantifier has standard properties, but the bound variable x ranges always over the countable set of channel names of the process calculus.

• The collection of all properties has also the structure of a quantale, induced by the parallel composition operator over processes. In the logic, this is reflected by the operators $A|B$ (the tensor, or parallel composition, of two properties), $\mathbf{0}$ (the unit of the tensor, or collection of void processes), and $A \triangleright B$ (the linear implication associated with the tensor). This last operator corresponds to context-system specifications, which are the concurrency theory equivalent of pre/post conditions. In addition, $A \triangleright B$ is a first class formula that can be freely and usefully combined with other operators.

• In much the same way that parallel process composition induces the quantale structure, process restriction induces a pair of operators $n®A$ and $A \oslash n$, called revelation and hiding, that give us a basis for describing restricted processes at the logical level.

• The notion of "fresh name" is introduced by a quantifier $\text{И}x.A$; a process P satisfies $\text{И}x.A$ if P satisfies A for some name fresh in P and $\text{И}x.A$. This quantifier allows us to then derive a hidden name quantifier [11, 6]. $\text{И}x.A$ is defined along the lines of the freshness quantifier of Gabbay-Pitts [19]: the connections will be discussed. A similar $\text{И}x.A$ quantifier is studied in [11] (in absence of recursion), but is handled as a meta-level construct, and not as a proper formula that can be mixed with recursion.

• A logical operator $n\langle m\rangle$ allows us to assert that a message m is present over channel n, giving us some minimal power to observe the behavior of processes.

• A "next step" temporal operator, $\Diamond A$, allows us to talk about the behavior of a process after a single (unspecified) reduction step.

• Finally, a second-order quantifier $\forall X.A$ enables us to quantify over the collection of all properties. Combining $\forall X.A$ with other operators of our logic, we then define a maximal fixpoint operator $\boldsymbol{\nu} X.A$ (provided A is monotonic in X), and a dual minimal fixpoint operator $\boldsymbol{\mu} X.A$. From these recursive formulas, we can then define operators for temporal and spatial modalities, for instance $\Box A$ denoting that A holds anytime in the future, and $⌑ A$, meaning that A holds everywhere in space.

Related Work A logic for a process calculus including a tensor operator and a hidden name quantifier was developed in [6, 2], but a satisfactory semantic treatment for the latter connective not was achieved before the contributions of [11] and of the present paper. Initial versions of spatial logics for the Ambient Calculus were introduced in [10], which also investigated connections with linear logic. We now target the logic towards a more standard π-calculus.

Following the initial approach of Hennessy-Milner [22], modal logics for the π-calculus have been proposed in [29, 16, 17]. The main difference between our logic and these more standard logics of concurrency is the presence in our case of structural operations: namely, of a tensor operator that corresponds to process composition, and of a revelation operator that corresponds to name restriction. Usually, those other logics require formulas to denote processes up to bisimulation, which is difficult to reconcile with a tensor operator that can make distinctions between bisimilar processes (however, such an operator was anticipated in [15]). In our case, we only require formulas to denote processes up to structural equivalence, so that a tensor operator makes easy sense. Sangiorgi has shown, for a closely related logic, that the equivalence induced by the logic is then essentially structural equivalence [33].

The connections between name restriction and Gabbay-Pitts notions of freshness [20, 19, 31], first studied in [11], are further explored in this paper.

The work on Bunched Logics [30] and Separation Logic [32] is closely related to ours, at least in intent. Spatial logics for trees and graphs have also been investigated in [9, 7, 8].

Organization of the paper We start with a concise review of the asynchronous π-calculus. In Section 3 we give a detailed presentation of the syntax of the spatial logic. In Section 4, we introduce the central notion of property set, we define satisfaction, and we proceed with the analysis of semantical aspects of the logic. In Section 4.3 we then study an appropriate notion of logical validity. In Sections 5 and 6 we motivate and discuss fresh and hidden name quantification, and the recursive definition of properties. In Section 7 we discuss in more detail the arguments that lead to our choices.

2. PRELIMINARIES ON THE ASYNCHRONOUS π-CALCULUS

We review the syntax and operational semantics of the asynchronous π-calculus [1, 24], following the notations of [27]. We base our presentation of α-equivalence on the use of transpositions (simple name replacements), which become prominent later in the paper.

Definition 2.1 (Processes). *Given a countable set Λ of* names*, the set $\mathcal{P}$ of* processes *is given by the following abstract syntax*

$$
\begin{array}{lll}
m, n, p & \in & \Lambda \text{ (Names)} \\
P, Q, R & ::= & \\
 & \mathbf{0} & \text{(Void)} \\
 & P|Q & \text{(Par)} \\
 & (\boldsymbol{\nu} n)P & \text{(Restriction)} \\
 & m\langle n\rangle & \text{(Message)} \\
 & m(n).P & \text{(Input)} \\
 & !P & \text{(Replication)}
\end{array}
$$

We write $na(P)$ for the set of all names that textually occur in a process P (either bound or free).

DEFINITION 2.2 (Free names in Processes). *For any process P, the set of* free names *of P, written $fn(P)$, is inductively defined as follows.*

$$
\begin{aligned}
&fn(\mathbf{0}) \triangleq \mathbf{0} \\
&fn(P|Q) \triangleq fn(P) \cup fn(Q) \\
&fn(m\langle n\rangle) \triangleq \{m, n\} \\
&fn((\boldsymbol{\nu} n)P) \triangleq fn(P) \setminus \{n\} \\
&fn(m(n).P) \triangleq (fn(P) \setminus \{n\}) \cup \{m\} \\
&fn(!P) \triangleq fn(P)
\end{aligned}
$$

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. We write $bn(P)$ for the set of names which occur bound in the process P, and ©N for the set $\{P \mid N \subseteq fn(P)\}$ of processes that contain all names in N free. If N is a finite set of names, and N' is a any set of names, a substitution $\theta : N \to N'$ of *domain* N and *codomain* N' is a mapping assigning $\theta(n) \in N'$ to each $n \in N$, and n to each $n \notin N$. Thus, outside its domain, any substitution behaves like the identity. Given a substitution θ, we denote by $\mathfrak{D}(\theta)$ its domain. The *image* of a substitution θ, written $\mathfrak{I}(\theta)$, is the set $\{\theta(n) \mid n \in \mathfrak{D}(\theta)\}$. We write $\{n \leftarrow m\}$ for the singleton substitution of domain $\{n\}$ that assigns m to n.

Substitutions that just interchange a pair of names will play a special role in technical developments to follow. More precisely, the *transposition* of n and m, noted $\{n \leftrightarrow m\}$, denotes the substitution $\tau : \{m, n\} \to \{m, n\}$ such that $\tau(n) = m$ and $\tau(m) = n$. Note that $\{n \leftrightarrow m\} = \{m \leftrightarrow n\}$. Before defining safe substitution on processes, we first introduce transposition, and then define α-congruence in the style of [19].

DEFINITION 2.3 (Transposition). *Given a process P and a transposition τ, we denote by $\tau{\cdot}P$ the process inductively defined as follows.*

$$
\begin{array}{lcllcl}
\tau{\cdot}\mathbf{0} & \triangleq & \mathbf{0} & \tau{\cdot}((\boldsymbol{\nu} n)P) & \triangleq & (\boldsymbol{\nu}\tau(n))\tau{\cdot}P \\
\tau{\cdot}m\langle n\rangle & \triangleq & \tau(m)\langle\tau(n)\rangle & \tau{\cdot}(p(n).P) & \triangleq & \tau(p)(\tau(n)).(\tau{\cdot}P) \\
\tau{\cdot}(P|Q) & \triangleq & (\tau{\cdot}P)|(\tau{\cdot}Q) & \tau{\cdot}(!P) & \triangleq & !\tau{\cdot}P
\end{array}
$$

PROPOSITION 2.4. *For all processes P and Q, and transpositions τ,*

1. $\tau{\cdot}\tau{\cdot}P = P$
2. $\{m \leftrightarrow n\}{\cdot}\tau{\cdot}P = \tau{\cdot}\{\tau(m) \leftrightarrow \tau(n)\}{\cdot}P$

Proof. By induction on the structure of P. ∎

DEFINITION 2.5 (Congruence). *A binary relation $\cong$ on processes is a* congruence *whenever for all processes P, Q and R,*

$$
\begin{array}{ll}
P \cong P & \text{(Cong Refl)} \\
P \cong Q \Rightarrow Q \cong P & \text{(Cong Symm)} \\
P \cong Q, Q \cong R \Rightarrow P \cong R & \text{(Cong Trans)} \\
P \cong Q \Rightarrow P|R \cong Q|R & \text{(Cong Parl)} \\
P \cong Q \Rightarrow R|P \cong P|Q & \text{(Cong Parr)} \\
P \cong Q \Rightarrow (\boldsymbol{\nu} n)P \cong (\boldsymbol{\nu} n)Q & \text{(Cong Res)} \\
P \cong Q \Rightarrow m(n).P \cong m(n).Q & \text{(Cong In)} \\
P \cong Q \Rightarrow !P \cong !Q & \text{(Cong Repl)}
\end{array}
$$

In this paper we essentially make use of two congruences: α-congruence and structural congruence. As usual, *α-congruence* $\equiv_\alpha$ is the congruence that identifies processes up to the safe renaming of bound names.

DEFINITION 2.6 (α-congruence). α-congruence $\equiv_\alpha$ *is the least congruence on processes such that,*

$$
\begin{array}{lll}
(\boldsymbol{\nu} n)P \equiv_\alpha (\boldsymbol{\nu} p)\{n\leftrightarrow p\}\cdot P & \textit{where } p \notin na(P) & \text{(Alpha Res)} \\
m(n).P \equiv_\alpha m(p).\{n\leftrightarrow p\}\cdot P & \textit{where } p \notin na(P) & \text{(Alpha In)}
\end{array}
$$

DEFINITION 2.7 (Safe substitution). *For any process P and substitution θ we denote by $\theta(P)$ the process inductively defined as follows:*

$$
\begin{array}{lll}
\theta(\mathbf{0}) & \triangleq & \mathbf{0} \\
\theta(m\langle n\rangle) & \triangleq & \theta(m)\langle\theta(n)\rangle \\
\theta(P|Q) & \triangleq & \theta(P)|\theta(Q) \\
\theta((\boldsymbol{\nu} n)P) & \triangleq & (\boldsymbol{\nu} p)\theta(P\{n\leftrightarrow p\}) \quad \textit{where } p \notin \mathfrak{D}(\theta) \cup \mathfrak{I}(\theta) \cup fn(P) \\
\theta(m(n).P) & \triangleq & \theta(m)(p).\theta(P\{n\leftrightarrow p\}) \quad \textit{where } p \notin \mathfrak{D}(\theta) \cup \mathfrak{I}(\theta) \cup fn(P) \\
\theta(!P) & \triangleq & !\theta(P)
\end{array}
$$

The name p in the clauses for restriction and input is chosen fresh, hence safe substitution is well-defined mapping on α-equivalence classes of processes, as usual. We write $P\theta$ for $\theta(P)$ when θ has the form $\{n\leftarrow m\}$ or $\{n\leftrightarrow m\}$. We have

LEMMA 2.8. *Let P be a process. Then*

1. $\tau\cdot P \equiv_\alpha \tau(P)$ *where τ is any transposition*
2. $P\{n\leftarrow p\} \equiv_\alpha \{n\leftrightarrow p\}\cdot P$ *where $p \notin fn(P)$*

Proof. By induction on the structure of P. ∎

From Lemma 2.8 the usual characterization of α-congruence follows:

PROPOSITION 2.9. *α-congruence is the least congruence on processes such that*

$$
\begin{array}{ll}
(\boldsymbol{\nu} n)P \equiv_\alpha (\boldsymbol{\nu} p)P\{n\leftarrow p\} & \textit{where } p \notin fn(P) \\
m(n).P \equiv_\alpha m(p).P\{n\leftarrow p\} & \textit{where } p \notin fn(P)
\end{array}
$$

As expected, safe substitution preserves α-congruence:

PROPOSITION 2.10. *If $P \equiv_\alpha Q$ then $\theta(P) \equiv_\alpha \theta(Q)$.*

Proof. Standard. ∎

DEFINITION 2.11 (Structural congruence). Structural congruence $\equiv$ *is the least congruence on processes such that*

$$
\begin{array}{ll}
P \equiv_\alpha Q \Rightarrow P \equiv Q & \text{(Struct Alpha)} \\
P|\mathbf{0} \equiv P & \text{(Struct Par Void)} \\
P|Q \equiv Q|P & \text{(Struct Par Comm)} \\
P|(Q|R) \equiv (P|Q)|R & \text{(Struct Par Assoc)} \\
n \notin fn(P) \Rightarrow P|(\boldsymbol{\nu} n)Q \equiv (\boldsymbol{\nu} n)(P|Q) & \text{(Struct Res Par)} \\
n \neq p, n \neq m \Rightarrow (\boldsymbol{\nu} n)p(m).P \equiv p(m).(\boldsymbol{\nu} n)P & \text{(Struct Res Inp)} \\
(\boldsymbol{\nu} n)\mathbf{0} \equiv \mathbf{0} & \text{(Struct Res Void)} \\
(\boldsymbol{\nu} n)(\boldsymbol{\nu} m)P \equiv (\boldsymbol{\nu} m)(\boldsymbol{\nu} n)P & \text{(Struct Res Comm)} \\
!\mathbf{0} \equiv \mathbf{0} & \text{(Struct Repl Void)} \\
!P \equiv !P|P & \text{(Struct Repl Copy)} \\
!(P|Q) \equiv !P|!Q & \text{(Struct Repl Par)} \\
!!P \equiv !P & \text{(Struct Repl Repl)}
\end{array}
$$

Although the axiom (Struct Res Inp) is absent from standard presentations of π-calculi, the general consensus seems to be that such an axiom is quite sensible as a structural congruence. (Struct Res Inp) is implicit in early work of Boudol on the chemical abstract machine, and is harmless as far as extensional properties of processes (*e.g.*, behavioral equivalence) are concerned. On the other hand, it has some convenient consequences in the setting of a more intensional logic like ours. Moreover, Engelfriet and Gelsema have shown the decidability of structural congruence in the presence of the (Struct Repl Void/Par/Repl) and (Struct Res Inp) axioms [18].

PROPOSITION 2.12 (Basic properties of $\equiv$). *For all processes P and Q,*

1. *If $P \equiv Q$ then $fn(P) = fn(Q)$.*
2. *If $n \notin fn(P)$ then $(\boldsymbol{\nu} n)P \equiv P$.*
3. *For all transpositions τ, $P \equiv Q$ if and only if $\tau(P) \equiv \tau(Q)$.*
4. *For all substitutions θ, if $P \equiv Q$, then $\theta(P) \equiv \theta(Q)$.*

Proof. Standard. ∎

PROPOSITION 2.13 (Inversion). *For all processes P and Q,*

1. *If $(\boldsymbol{\nu} n)P \equiv \mathbf{0}$ then $P \equiv \mathbf{0}$.*
2. *If $(\boldsymbol{\nu} n)P \equiv R|Q$ then there are R' and Q' such that $P \equiv R'|Q'$ and either $R \equiv (\boldsymbol{\nu} n)R'$ and $Q \equiv Q'$ and $n \notin fn(Q)$, or $R \equiv R'$ and $Q \equiv (\boldsymbol{\nu} n)Q'$ and $n \notin fn(R)$.*
3. *If $(\boldsymbol{\nu} n)P \equiv (\boldsymbol{\nu} m)Q$ then either $P \equiv \{n \leftrightarrow m\} \cdot Q$ or there are P' and Q' such that $P \equiv (\boldsymbol{\nu} m)P'$, $Q \equiv (\boldsymbol{\nu} n)Q'$ and $P' \equiv Q'$.*

Versions of Proposition 2.13(1–2) for the Ambient Calculus have been proved in [13] and [12]. Proposition 2.13(3) has a simple proof based on results in [18], as suggested by J. Engelfriet.

The dynamics of processes is captured by reduction:

DEFINITION 2.14 (Reduction). *Reduction is the least binary relation $\rightarrow$ on processes inductively defined as follows.*

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

PROPOSITION 2.15 (Basic properties of $\rightarrow$). *For all processes P and Q,*

1. *If $P \rightarrow Q$ then $fn(Q) \subseteq fn(P)$.*
2. *For all substitutions θ, if $P \rightarrow Q$, then $\theta(P) \rightarrow \theta(Q)$.*
3. *If $P \rightarrow Q$ and $Q \equiv (\boldsymbol{\nu} n)Q'$ for some Q', then there is P' such that $P \equiv (\boldsymbol{\nu} n)P'$ and $P' \rightarrow Q'$.*

x, y, z	$\in$	$\mathcal{V}$ (Name variables)
X, Y, Z	$\in$	$\mathcal{X}$ (Propositional variables)
η	$\in$	$\Lambda \cup \mathcal{V}$ (Names or name variables)
A, B, C	$::=$	
	$\mathbf{F}$	(False)
	$A \wedge B$	(Conjunction)
	$A \Rightarrow B$	(Implication)
	$\mathbf{0}$	(Void)
	$A \mid B$	(Composition)
	$A \triangleright B$	(Guarantee)
	$\eta \circledR A$	(Revelation)
	$A \oslash \eta$	(Hiding)
	$\eta\langle\eta'\rangle$	(Message)
	$\forall x.A$	(First-order universal quantification)
	$\text{И}x.A$	(Fresh name quantification)
	$\Diamond A$	(Next step)
	X	(Propositional variable)
	$\forall X.A$	(Second-order universal quantification)

FIG. 1 Formulas.

Proof. (1—2) Standard. (3) By induction on the derivation of $P \to Q$ (see Remark 2.16 below). ∎

REMARK 2.16. Proposition 2.15(3) is a consequence of Proposition 2.13(3) and does not hold for versions of π-calculi where $\equiv$ does not satisfy (Struct Res Inp). *E.g.*, consider $P \triangleq p\langle p\rangle | p(m).(\boldsymbol{\nu} n)m\langle n\rangle$: we have $P \to (\boldsymbol{\nu} n)Q'$ where $Q' = p\langle n\rangle$. Now, pick any P' such that $P \equiv (\boldsymbol{\nu} n)P'$. Then, we can show that for all R such that $P' \to R$ we have $R \equiv (\boldsymbol{\nu} n)Q' \not\equiv Q'$. However, if we admit (Struct Res Inp), we have $P \equiv (\boldsymbol{\nu} n)(p\langle p\rangle | p(m).m\langle n\rangle)$, so we can take $P' = p\langle p\rangle | p(m).m\langle n\rangle$.

REMARK 2.17. Reduction as defined in Definition 2.14 is "almost the same" as the usual one in the following sense. Let $\equiv_c$ be the the standard structural congruence of [27] restricted to the asynchronous π-calculus. Thus $\equiv_c \subseteq \equiv$. Likewise, let $\to_c$ be the standard reduction of the asynchronous π-calculus. It is clear that $\to_c \subseteq \to$. We also have that for all processes P and Q, if $P \to Q$ then there is a process R such that $P \to_c R$ and $Q \equiv R$.

3. SYNTAX OF THE SPATIAL LOGIC

Basic constructs of our spatial logic include propositional, spatial, and temporal operators, first-order and second-order quantifications, and freshness quantification (*cf.* [19]). As shown later, from this basic set of connectives we can define a quite expressive set of properties, including fixpoint combinators (supporting inductive and coinductive definition of properties) and an internal satisfiability modality [10].

DEFINITION 3.1 (Formulas). *Given an infinite set $\mathcal{V}$ of name variables, and an infinite set $\mathcal{X}$ of propositional variables (mutually disjoint from Λ), formulas are defined as shown in Fig. 1.*

C	$fn(C)$	$fv(C)$	$fpv(C)$
$\mathbf{F}$ $\mathbf{0}$	$\emptyset$	$\emptyset$	$\emptyset$
$A \wedge B$ $A \Rightarrow B$ $A\|B$ $A \triangleright B$	$fn(A) \cup fn(B)$	$fv(A) \cup fv(B)$	$fpv(A) \cup fpv(B)$
$\eta\langle\eta'\rangle$	$\{\eta, \eta'\} \cap \Lambda$	$\{\eta, \eta'\} \cap \mathcal{V}$	$\emptyset$
$\eta ® A$ $A \oslash \eta$	$fn(A) \cup (\{\eta\} \cap \Lambda)$	$fv(A) \cup (\{\eta\} \cap \mathcal{V})$	$fpv(A)$
$\forall x.A$ $И x.A$	$fn(A)$	$fv(A) \setminus \{x\}$	$fpv(A)$
$\Diamond A$	$fn(A)$	$fv(A)$	$fpv(A)$
X	$\emptyset$	$\emptyset$	$\{X\}$
$\forall X.A$	$fn(A)$	$fv(A)$	$fpv(A) \setminus \{X\}$

FIG. 2 Free names in formulas.

The meaning of these formulas was briefly discussed in the introduction; their semantics is given later in Definition 4.17. We highlight here some of the more unusual operators. The formula $\mathbf{0}$ is satisfied by any process in the structural congruence class of $\mathbf{0}$. The formula $A|B$ is satisfied by any process that can be decomposed into processes that satisfy respectively A and B. Guarantee is the logical adjunct of composition: $A \triangleright B$ is satisfied by those processes whose composition with any process satisfying A results in a process satisfying B. The formula $n®A$ is satisfied by all processes congruent with some process $(\boldsymbol{\nu} n)P$, where P satisfies A. The formula $A \oslash n$ is satisfied by any process P such that $(\boldsymbol{\nu} n)P$ satisfies A; *i.e.*, by a process that satisfies A after hiding n. Message $m\langle n\rangle$ holds of processes structurally congruent to a message $m\langle n\rangle$. The formula $И x.A$ denotes fresh name quantification; a process satisfies $И x.A$ if for (some/all) fresh names n (fresh in the process and in the formula), it satisfies $A\{x \leftarrow n\}$. This quantifier exhibits the universal/existential ambivalence typical of freshness: a property holding of some fresh names should also hold of any other fresh name. As we shall see, combining the fresh name quantifier with revelation will enable us to define a hidden name quantifier, that is a quantifier over names that are locally restricted in the process at hand.

In formulas of the form $\forall x.A$, $И x.A$, and $\forall X.A$ the distinguished occurrences of x and X are binding, with scope the formula A. We define on formulas the relation $\equiv_\alpha$ of α-congruence in the standard way, that is, as the least congruence identifying formulas modulo safe renaming of bound (name and propositional) variables. We consider formulas always modulo α-congruence.

DEFINITION 3.2 (Free names and variables in formulas). *For any formula C, we introduce the following sets, inductively defined in Fig. 2.*

$fn(C)$	*free names in C*
$fv(C)$	*free name variables in C*
$fpv(C)$	*free propositional variables in C*

By $fnv(A)$ we mean the set $fv(A) \cup fn(A)$. A formula is *name-closed* if it has no free name variables, and *closed* if it has no free variables whatsoever.

We extend the previously given notion of substitution to name variables and formulas as follows. When S is a finite set of either variables and names, and N is a set of names, $\theta : S \to N$ means that θ is a substitution assigning a name in N to each variable or name in S. If $\theta : S \to N$ is a substitution then θ_{-x} denotes the substitution of domain $S \setminus \{x\}$ and codomain N defined by $\theta_{-x}(y) = \theta(y)$, for all $y \in S \setminus \{x\}$.

DEFINITION 3.3 (Safe substitution). *For any formula A and substitution θ we denote by $\theta(A)$ the formula inductively defined as follows.*

$$
\begin{array}{lcllcl}
\theta(\mathbf{F}) & \triangleq & \mathbf{F} & \theta(\eta \circledR A) & \triangleq & \theta(\eta)\circledR\theta(A) \\
\theta(\mathbf{0}) & \triangleq & \mathbf{0} & \theta(A \oslash \eta) & \triangleq & \theta(A)\oslash\theta(\eta) \\
\theta(A \wedge B) & \triangleq & \theta(A) \wedge \theta(B) & \theta(\text{И}x.A) & \triangleq & \text{И}x.\theta_{-x}(A) \\
\theta(A \Rightarrow B) & \triangleq & \theta(A) \Rightarrow \theta(B) & \theta(\forall x.A) & \triangleq & \forall x.\theta_{-x}(A) \\
\theta(A|B) & \triangleq & \theta(A)|\theta(B) & \theta(\Diamond A) & \triangleq & \Diamond\theta(A) \\
\theta(A \triangleright B) & \triangleq & \theta(A) \triangleright \theta(B) & \theta(X) & \triangleq & X \\
\theta(\eta\langle\eta'\rangle) & \triangleq & \theta(\eta)\langle\theta(\eta')\rangle & \theta(\forall X.A) & \triangleq & \forall X.\theta(A)
\end{array}
$$

When A and B are formulas, we denote by $A\{X \leftarrow B\}$ the capture avoiding substitution of all free occurrences of X in A by B, defined in the expected way. By $F[-]$ we denote a formula context with possibly multiple occurrences of the hole $-$. Then, whenever A is a formula, we denote by $F[A]$ the formula obtained by textually replacing every occurrence of the hole $-$ in the context $F[-]$ by A. Note that free (name or propositional) variables in A will be captured by binders present in $F[-]$; *cf.*, the standard notion of context substitution.

4. SEMANTICS

The semantics of formulas is defined by assigning to each formula A a set of processes $[\![A]\!]$, namely the set of all processes that satisfy the property denoted by formula A.

However not any set of processes can denote a property in a proper way. For instance, it is sensible to require $[\![A]\!]$ to be closed under structural congruence. That is, if a process P satisfies some property A, then any process Q such that $Q \equiv P$ must also satisfy A. We also want to be able to express freshness of names with relation to $[\![A]\!]$. Suppose we have $P \in [\![A]\!]$, $n \notin fn(A)$ but $n \in fn(P)$. Since $n \notin fn(A)$, the free occurrences of n in P are *fresh* for the formula A. Now, the particular choice of the name n should not depend on A itself, since it is natural to consider that all fresh names for A are to be treated uniformly. Therefore, it is natural to require that also $P\{n \leftarrow m\} \in [\![A]\!]$, where m is any other fresh name for A and P, that is $m \notin fn(P) \cup fn(A)$.

Hence, we say that a set of processes Φ is *supported* by the set of names N if, for all m, n not in the support N, if P belongs to Φ then $P\{n \leftrightarrow m\}$ is also in Φ. We then take as properties only those sets of processes that have a finite support. Intuitively, the support of a property is the semantic counterpart of the set of free names of a formula; the least support of the denotation of a formula is included in the set of free names of the formula. Sets with infinite support could only correspond to formulas that have an infinite set of free names, and are therefore excluded.

Moreover, the notion of finite support seems crucial for the semantics of the fresh name quantifier, $\text{И}x.A$, and consequently for the semantics of the derived hidden name quantifier $\mathsf{H}x.A$. The semantics of the spatial logics of [11, 10, 6] is given in terms of sets of processes that are closed only under structural congruence, but if we try to extend that semantics to recursive formulas, we run into a problem: $\text{И}x.A$ is not a monotonic operator, and could not be used together with recursion. This discussion is continued in more detail in Section 6.

4.1. Property Sets

The above observations motivate the following notion of *property set*. A property set is a set of processes closed under structural congruence and finitely supported.

DEFINITION 4.1 (Property Set). *A* property set *(Pset) is a set of processes Ψ such that*

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

DEFINITION 4.2 (Collections of Property Sets).

1. *$\mathbb{P}_N$ is the set of all Psets supported by the finite set of names N.*
2. *$\mathbb{P}_-$ is the set of all Psets.*

The finite set N mentioned in Definition 4.1(2) is referred to as a *support* of the Pset. We use Ψ and Φ to range over property sets. A support N plays for a Pset a role similar to (a bound on) the set of free names of a formula, and enables the definition of a notion of name freshness with respect to a possibly infinite set of processes. We use the notation $S^{\equiv}$ to denote the closure under structural congruence of an arbitrary set of processes S.

LEMMA 4.3 (Operations on Psets). *For all finite $N \subseteq \Lambda$,*

1. *If $N \subseteq N'$ then $\mathbb{P}_N \subseteq \mathbb{P}_{N'}$.*
2. *(Bottom and Top) $\emptyset \in \mathbb{P}_N$ and $\mathcal{P} \in \mathbb{P}_N$.*
3. *(Meet and Join) If $S \subseteq \mathbb{P}_N$ then $\bigcap S \in \mathbb{P}_N$ and $\bigcup S \in \mathbb{P}_N$.*
4. *(Inverse) If $\Psi \in \mathbb{P}_N$ then $\overline{\Psi} = \mathcal{P} \setminus \Psi \in \mathbb{P}_N$.*

Proof. See appendix. ∎

We can also extend the application of transpositions (not of arbitrary substitutions!) to Psets as follows: if τ is a transposition and Ψ is a Pset, define $\tau(\Psi) \triangleq \{\tau(P) \mid P \in \Psi\}$.

Note that Lemma 4.3(2-4) implies

PROPOSITION 4.4 (Lattice). *For all finite $N \subseteq \Lambda$, we have*

1. *$\langle \mathbb{P}_N, \subseteq, \cup, \cap \rangle$ is a complete lattice.*
2. *$\langle \mathbb{P}_N, \cup, \cap, \overline{\cdot} \rangle$ is a Boolean algebra.*

Remark 4.5. Note that $\mathbb{P}_{-}$ is neither closed under arbitrary unions nor closed under arbitrary intersections. For instance, let $\langle m_1, m_2, \dots \rangle$ be a linear ordering of Λ, let $P_0 \triangleq \mathbf{0}$ and for any $i > 0$, $P_i \triangleq m_i(n).P_{i-1}$. Then $\{P_i\}^{\equiv}$ is finitely supported (with support $\{m_1, \dots, m_i\}$) for any $i \geq 0$, but $\bigcup_{i \geq 0}\{P_i\}^{\equiv}$ is not. Thus the collection of all Psets $\mathbb{P}_{-}$ is not a complete lattice.

However, we can recover closure under all basic set-theoretic operations, by restricting to a cumulative hierarchy of finitely supported sets [19].

Definition 4.6. *For any finite set of names N, a collection S of Psets is finitely supported by N if for all $m, n \notin N$ and $\Phi \in S$ we have $\{m \leftrightarrow n\}(\Phi) \in S$.*

Definition 4.7. *$\mathbb{P}^2_N$ is the set of all collections of Psets supported by the finite set of names N.*

Lemma 4.8. *If $S \in \mathbb{P}^2_N$ then $\bigcup S \in \mathbb{P}_N$ and $\bigcap S \in \mathbb{P}_N$.*

Proof. If $P \in \bigcup S$ then $P \in \Phi$ for some $\Phi \in S \subseteq \bigcup S$. If $Q \equiv P$ then $Q \in \Phi \in S \subseteq \bigcup S$. Let $\tau = \{m \leftrightarrow n\}$ with $m, n \notin N$. Since $\tau(\Phi) \in S$, we also have $\tau(P) \in \bigcup S$. The case for $\bigcap S$ is similar. ∎

Definition 4.9 (Tensor and Unit). *For every $\mathbb{P}_N$, define operations*

$$\otimes : \mathbb{P}_N \times \mathbb{P}_N \to \mathbb{P}_N \qquad \mathbf{1} : \mathbb{P}_N$$

by letting, for all $\Psi, \Phi \in \mathbb{P}_N$

$$\begin{array}{ll} \Psi \otimes \Phi & \triangleq \{P \mid \textit{Exists } Q, R.\ P \equiv Q|R \textit{ and } Q \in \Psi \textit{ and } R \in \Phi\} \\ \mathbf{1} & \triangleq \{P \mid P \equiv \mathbf{0}\} \end{array}$$

In [10] it is shown that the set of all $\equiv$-closed subsets of $\mathcal{P}$ gives rise to a commutative quantale. The same result still holds for domains of Psets.

Proposition 4.10 (Quantale). *For all finite $N \subseteq \Lambda$, $\langle \mathbb{P}_N, \subseteq, \bigcup, \otimes, \mathbf{1} \rangle$ is a commutative quantale, that is:*

1. *$\langle \mathbb{P}_N, \subseteq, \bigcup \rangle$ is a complete join semilattice.*
2. *$\langle \mathbb{P}_N, \otimes, \mathbf{1} \rangle$ is a commutative monoid.*
3. *$\Phi \otimes \bigcup S = \bigcup\{\Phi \otimes \Psi \mid \Psi \in S\}$, for all $\Phi \in \mathbb{P}_N$ and $S \subseteq \mathbb{P}_N$.*

Proof. See appendix. ∎

Lemma 4.11 (Transposing Psets). *We have*

1. *For any process P and Pset Ψ, $P \in \tau(\Psi)$ if and only if $\tau(P) \in \Psi$.*
2. *$\Psi \in \mathbb{P}_N$ if and only if $\tau(\Psi) \in \mathbb{P}_{\tau(N)}$.*
3. *If $m, n \notin N$ and $\Psi \in \mathbb{P}_N$ then $\{m \leftrightarrow n\}(\Psi) = \Psi$.*

Proof. See appendix. ∎

Definition 4.12 (Support). *Let N be a set of names. A Pset Ψ is* supported *by N whenever every permutation that fixes N also fixes Ψ. Moreover Ψ is* finitely supported *by N if supported by N and N is finite.*

We also have

Proposition 4.13 (Least Support). *Let $\Phi \in \mathbb{P}_N$. Then*

1. *There is a least set of names $supp(\Phi)$ such that $\Phi \in \mathbb{P}_{supp(\Phi)}$.*

2. *For any transposition τ, $supp(\tau(\Phi)) = \tau(supp(\Phi))$.*

Proof. See appendix. ∎

Intuitively, the set of names $supp(\Phi)$ represents the set of "free names" of the Pset Φ (in the sense of Lemma 4.11(3)), hence $supp(-)$ is the semantic counterpart of the set $fn(-)$ of free names of a formula.

Remark 4.14. We can verify that a Pset Ψ supported by N is finitely supported by N in the precise sense of [19]:

A name permutation π over Λ is an injective name substitution such that $\mathfrak{D}(\pi) = \mathfrak{I}(\pi)$. Let S_Λ be the group of all name permutations; recall that any permutation can be expressed as a composition of transpositions. For any Pset Ψ, $\pi(\Psi) \in \mathbb{P}_-$, by Lemma 4.11(2). Hence $\mathbb{P}_-$ is an S_Λ−set.

Now, let $\Psi \in \mathbb{P}_N$. Pick any $\pi \in S_\Lambda$ and assume that π is not the identity permutation. This implies that there is some permutation π', such that $\pi'(m) = \pi(m)$ for all $m \in \Lambda$ and $\pi'(m) \neq m$, for all $m \in \mathfrak{D}(\pi')$. Assume that for all $n \in N$, $\pi(n) = n$. Then, for all $n \in N$, $\pi'(n) = n$. We can see that N is disjoint from $\mathfrak{D}(\pi') = \mathfrak{I}(\pi')$. Hence, π' can be written as a composition of transpositions $\tau_1 \cdots \tau_k$ such that $\tau_i = \{p_i \leftrightarrow q_i\}$ and $p_i, q_i \notin N$, for all $i = 1, \cdots, k$. Therefore $\pi'(\Psi) = \pi(\Psi) = \Psi$. This means that N (finitely) supports Ψ. We conclude that $\mathbb{P}_-$ is a $perm(\Lambda)$-set with the finite support property.

4.2. Satisfaction

We define the denotation of a formula A by a Pset $[\![A]\!] \in \mathbb{P}_{fn(A)}$. However, since A may contain free occurrences of propositional variables, its denotation depends on the denotation of such variables, which is given by a valuation.

Definition 4.15 (Valuation). *A valuation v is a mapping from a finite subset of $\mathcal{X}$ (the propositional variables), assigning to each propositional variable X in its domain $\mathfrak{D}(v)$ a Pset Ψ. Given a formula A, a valuation for A is any valuation v such that $fpv(A) \subseteq \mathfrak{D}(v)$.*

Thus, the role of valuations is to interpret free propositional variables occurring in the formula A. When v is a valuation, we write $v[X \leftarrow \Psi]$ to denote the valuation of domain $\mathfrak{D}(v) \cup \{X\}$ that assigns Ψ to the propositional variable X, and $v(Z)$ to any other propositional variable $Z \neq X$. For any valuation v, we let

$$fn(v) \triangleq \bigcup\{supp(v(X)) \mid X \in \mathfrak{D}(v)\}$$

Taking into account the extra information yielded by a valuation, we now give a refined characterization of the free names of a formula A as follows

Definition 4.16 (Free names under a valuation). *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*

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

$$
\begin{array}{lcl}
[\![\mathbf{F}]\!]_v & \triangleq & \emptyset \\
[\![A \wedge B]\!]_v & \triangleq & [\![A]\!]_v \cap [\![B]\!]_v \\
[\![A \Rightarrow B]\!]_v & \triangleq & \{P \mid \textit{if } P \in [\![A]\!]_v \textit{ then } P \in [\![B]\!]_v\} \\
[\![\mathbf{0}]\!]_v & \triangleq & \mathbf{1} \\
[\![A|B]\!]_v & \triangleq & [\![A]\!]_v \otimes [\![B]\!]_v \\
[\![A \triangleright B]\!]_v & \triangleq & \{P \mid \textit{Forall } Q.\ \textit{if } Q \in [\![A]\!]_v \textit{ then } P|Q \in [\![B]\!]_v\} \\
[\![n \circledR A]\!]_v & \triangleq & \{P \mid \textit{Exists } Q.\ P \equiv (\boldsymbol{\nu} n)Q \textit{ and } Q \in [\![A]\!]_v\} \\
[\![A \oslash n]\!]_v & \triangleq & \{P \mid (\boldsymbol{\nu} n)P \in [\![A]\!]_v\} \\
[\![m\langle n\rangle]\!]_v & \triangleq & \{P \mid P \equiv m\langle n\rangle\} \\
[\![\forall x.A]\!]_v & \triangleq & \bigcap_{n \in \Lambda} [\![A\{x \leftarrow n\}]\!]_v \\
[\![\text{И} x.A]\!]_v & \triangleq & \bigcup_{n \notin fn^v(A)} ([\![A\{x \leftarrow n\}]\!]_v \setminus \{P \mid n \in fn(P)\}) \\
[\![\Diamond A]\!]_v & \triangleq & \{P \mid \textit{Exists } Q.\ P \rightarrow Q \textit{ and } Q \in [\![A]\!]_v\} \\
[\![X]\!]_v & \triangleq & v(X) \\
[\![\forall X.A]\!]_v & \triangleq & \bigcap_{\Psi \in \mathbb{P}_-} [\![A]\!]_{v[X \leftarrow \Psi]}
\end{array}
$$

FIG. 3 Denotation of formulas.

The set $fn^v(A)$ is used in an essential way in the definition of the semantics of the fresh name quantifier, where the quantification witness is tested for freshness with respect to the property set denoted by the formula A, where the formula A may contain free occurrences of propositional variables.

DEFINITION 4.17 (Denotation and Satisfaction). *The* denotation map $[\![-]\!]_v$, *inductively defined in Fig. 3, is the function that assigns a set of processes* $[\![A]\!]_v$ *to each name-closed formula* A *and valuation (for* A*)* v. *We write* $P \models_v A$ *whenever* $P \in [\![A]\!]_v$*: this means that* P satisfies *formula* A *under valuation* v.

The boolean connectives ($\mathbf{F}$, $\wedge$ and $\Rightarrow$) are interpreted as expected, while the spatial operations related to composition ($\mathbf{0}$, $|$) are interpreted in terms of the quantale operations in Definition 4.9. Then $\triangleright$ is given the expected semantics for the adjunct operator of the tensor. The spatial operations related to name hiding (revelation and hiding) are defined along similar lines.

In the semantics of name quantification the quantified name variable is ranged over the set Λ of all names.

The semantics given to the freshness quantifier is such that a process P satisfies $[\![\text{И} x.A]\!]_v$ if and only if P satisfies $[\![A\{x \leftarrow n\}]\!]_v$ for some name n fresh in A and P: this is the reason for subtracting $\{P \mid n \in fn(P)\}$, as further discussed in Section 5. Since in general A may contain free occurrences of propositional variables, freshness with relation to formula A must be defined in terms of $fn^v(A)$, as already mentioned; this is justified in more detail in Section 5.2, where alternative definitions for the semantics of $\text{И} x.A$ are also discussed.

The denotation of second order quantification is also defined as expected, except that the quantified propositional variable ranges over all property sets (rather than all sets of processes).

We now show that the denotation map is well-defined. Since we are considering formulas up to α-congruence, we start by verifying that the denotation map is well-defined on the corresponding equivalence classes.

LEMMA 4.18. *For all formulas A, B and valuations v for A and B, if $A \equiv_\alpha B$, then $[\![A]\!]_v = [\![B]\!]_v$.*

Proof. Induction on the structure of A. ∎

Note that assignments to propositional variables that do not occur free in the interpreted formula do not affect its denotation. Therefore, valuations can always be weakened and thinned whenever appropriate.

REMARK 4.19. For any formula A, Pset Φ and valuation v for A, if $X \notin \mathit{fpv}(A)$ then $[\![A]\!]_v = [\![A]\!]_{v[X \leftarrow \Phi]}$.

We now extend the application of transpositions to valuations, this is done in the expected way: when v is a valuation, let $\tau(v)$ be the valuation with same domain as v and defined by $\tau(v)(X) \triangleq \tau(v(X))$, for all $X \in \mathfrak{D}(v)$.

LEMMA 4.20. *For any formula A, valuation v and transposition τ, if $\tau = \{m \leftrightarrow n\}$ and $m, n \notin \mathit{fn}(v)$ then $[\![A]\!]_v = [\![A]\!]_{\tau(v)}$.*

Proof. For any $X \in \mathfrak{D}(v)$ we have $m, n \notin \mathit{supp}(v(X))$ and $v(X) \in \mathbb{P}_{\mathit{supp}(v(X))}$. Thus $\tau(\mathit{supp}(X)) = \mathit{supp}(X)$ by Lemma 4.11(3). Hence $\tau(v) = v$. ∎

Fundamental properties of the denotation mapping are stated in the following main theorem, from which all correctness properties of the semantics follow.

THEOREM 4.21. *For all formulas A and appropriate valuations v*

1. $[\![A]\!]_v \in \mathbb{P}_{\mathit{fn}^v(A)}$.
2. *For all transpositions τ, $\tau([\![A]\!]_v) = [\![\tau(A)]\!]_{\tau(v)}$.*

Proof. See Appendix. ∎

The property expressed in Theorem 4.21(2) corresponds to the *equivariance* property of [19], and essentially means that the denotation of a formula depends on the distinctions between the names that occur on it, rather than on the particular identities of such names.

LEMMA 4.22. *For any formula A and valuation v for A we have*

$$\mathit{supp}([\![A]\!]_v) \subseteq \mathit{fn}^v(A)$$

Proof. By Theorem 4.21(1) $[\![A]\!]_v \in \mathbb{P}_{\mathit{fn}^v(A)}$; hence by Proposition 4.13 there is a least set $N = \mathit{supp}([\![A]\!]_v)$ such that $[\![A]\!]_v \in \mathbb{P}_N$. So $\mathit{supp}([\![A]\!]_v) \subseteq \mathit{fn}^v(A)$. ∎

REMARK 4.23. By inspection of the proof of Theorem 4.21 we can verify

- Assume $[\![A]\!]_v \in \mathbb{P}_N$ and $[\![B]\!]_v \in \mathbb{P}_M$. Then

$$\begin{array}{lll}
[\![\mathbf{F}]\!]_v \in \mathbb{P}_\emptyset & [\![A \Rightarrow B]\!]_v \in \mathbb{P}_{N\cup M} & [\![A \oslash n]\!]_v \in \mathbb{P}_{N\cup\{n\}} \\
[\![\mathbf{0}]\!]_v \in \mathbb{P}_\emptyset & [\![A | B]\!]_v \in \mathbb{P}_{N\cup M} & [\![\Diamond A]\!]_v \in \mathbb{P}_N \\
[\![p\langle q\rangle]\!]_v \in \mathbb{P}_{\{p,q\}} & [\![A \triangleright B]\!]_v \in \mathbb{P}_{N\cup M} & [\![X]\!]_v \in \mathbb{P}_{\mathit{supp}(v(X))} \\
[\![A \wedge B]\!]_v \in \mathbb{P}_{N\cup M} & [\![n \circledR A]\!]_v \in \mathbb{P}_{N\cup\{n\}} &
\end{array}$$

- If $[\![A\{x \leftarrow n\}]\!]_v \in \mathbb{P}_{N\cup\{n\}}$ for all $n \in \Lambda$, then $[\![\forall x.A]\!]_v \in \mathbb{P}_N$
- If $[\![A\{x \leftarrow n\}]\!]_v \in \mathbb{P}_{N\cup\{n\}}$ for all $n \notin \mathit{fn}^v(A)$, then $[\![\text{И} x.A]\!]_v \in \mathbb{P}_N$

- If $\{\llbracket A \rrbracket_{v[X \leftarrow \Psi]} \mid \Psi \in \mathbb{P}_{-}\} \in \mathbb{P}^2_N$ then $\llbracket \forall X.A \rrbracket_v \in \mathbb{P}_N$

LEMMA 4.24. *Let A be any formula, $v[X \leftarrow \Psi]$ a valuation for A and B, and B any formula in which X does not occur free. Then*

$$\llbracket A\{X \leftarrow B\} \rrbracket_v = \llbracket A \rrbracket_{v[X \leftarrow \llbracket B \rrbracket_v]}$$

Proof. Induction on the structure of formula A. ∎

Another consequence of the closure property stated in Theorem 4.21(2) is that the relation of satisfaction between processes and formulas is closed under fresh name renaming.

LEMMA 4.25 (Fresh renaming). *Let P be a process and A a closed formula such that $P \models A$. If $m \notin fn(A) \cup fn(P)$ then $P\{n \leftarrow m\} \models A\{n \leftarrow m\}$.*

Proof. Since $m \notin fn(P) \cup fn(A)$, by Lemma 2.8(2) we have $P\{n \leftarrow m\} \equiv P\{n \leftrightarrow m\}$, and $A\{n \leftarrow m\} = A\{n \leftrightarrow m\}$. We conclude by Theorem 4.21(2). ∎

It should be stressed that the use of transpositions, as suggested to us by A. Pitts, together with the notion of support, yields for Lemma 4.25 a proof that is much simpler than direct ones (*e.g.*, [2, 10]). Further motivation and alternatives for the present semantics will be discussed in the next sections.

4.2.1. Basic Derived Connectives

Some derived connectives of basic interest are defined as shown next.

$\neg A$	$\triangleq$	$A \Rightarrow \mathbf{F}$	(Negation)
$\mathbf{T}$	$\triangleq$	$\neg \mathbf{F}$	(True)
$A \vee B$	$\triangleq$	$(\neg A) \Rightarrow B$	(Disjunction)
$A \Vert B$	$\triangleq$	$\neg(\neg A \mid \neg B)$	(Decomposition)
$\exists x.A$	$\triangleq$	$\neg \forall x.\neg A$	(Existential quantification)
$\exists X.A$	$\triangleq$	$\neg \forall X.\neg A$	(Second order existential quantification)
$\sim A$	$\triangleq$	$A \triangleright \mathbf{F}$	(Unsatisfiability)
$!A$	$\triangleq$	$\sim \neg A$	(Validity)
©η	$\triangleq$	$\neg \eta$®$\mathbf{T}$	(Free name)
$\eta \# \eta'$	$\triangleq$	©$\eta \triangleright$ (©$\eta \oslash \eta'$)	(Inequality)
$\eta = \eta'$	$\triangleq$	$\neg(\eta \# \eta')$	(Equality)
$\boxdot A$	$\triangleq$	$\neg \Diamond \neg A$	(All next)

Standard operations of the classical predicate calculus, namely $\neg A$ (Negation), $\exists x.A$ (Existential quantification), $A \vee B$ (Disjunction) and $\mathbf{T}$ (True) are defined as expected. Another interesting connective is $A \Vert B$, the DeMorgan dual of composition $A|B$, which supports the definition of a form of spatial quantification. A process satisfies $A \Vert B$ if and only if every component of P with respect to composition, satisfies either A or B. A process P satisfies $\sim A$ if there does not exists any process Q that satisfies A. Hence A is valid if some process satisfies $!A$ [10]. A process satisfies ©η if the name denoted by η is free in it [11]. Then, any process satisfies $\eta \# \eta'$ if, in presence of a process containing η free, if we hide η' we still have a process that contains η free: this can only hold true if η' and η' denote distinct names. We also have the modality $\boxdot$, which is the dual of $\Diamond$: a process satisfies $\boxdot A$ if and only if all processes to which it reduces in one step satisfy A.

PROPOSITION 4.26. *For every process P and names n, p we have*

1. $P \in [\![©n]\!]_v$ *if and only if* $n \in fn(P)$.

2. $P \in [\![n = p]\!]_v$ *if and only if* $n = p$.

Proof. 1. See [11]. 2. We verify that $P \in [\![©n \triangleright (©n \oslash p)]\!]_v$ if and only if $n \neq p$. Suppose $P \in [\![©n \triangleright (©n \oslash p)]\!]_v$. Then, for every Q such that $n \in fn(Q)$ we have $Q|P \in [\![©n \oslash p]\!]_v$. This implies $n \in fn((\boldsymbol{\nu} p)(Q|P))$, and thus $n \neq p$. Conversely, if $n \neq p$ and $n \in fn(Q)$ then $n \in fn((\boldsymbol{\nu} p)(Q|P))$ and $n \in fn(Q|P)$, for all P. Thus, for every Q such that $n \in fn(Q)$, we have $Q|P \in [\![©n \oslash p]\!]_v$ for all P. ∎

4.3. Validity

We now introduce a notion of logical validity. A formula A is valid if all of its ground instances, under all valuations, are satisfied by all processes.

DEFINITION 4.27 (Valid Formula). *A formula A is* valid *if for all substitutions θ with $fv(A) \subseteq \mathfrak{D}(\theta)$, and for all valuations v such that $fpv(A) \subseteq \mathfrak{D}(v)$, we have $[\![\theta(A)]\!]_v = \mathcal{P}$.*

We use the meta-level statement $valid(A)$ to assert validity of formula A. Logical validity satisfies the following general principles.

PROPOSITION 4.28 (Instantiation). *Let $F[-]$ be any formula context. We have*

1. *For any η and formula A, $valid(A) \Rightarrow valid(A\{x \leftarrow \eta\})$.*

2. *For any formula A, $valid(F[X]) \Rightarrow valid(F[A])$*

Proof. 1. Assume $valid(A)$. Then for all substitutions θ where $fv(A) \subseteq \mathfrak{D}(\theta)$, for all valuations v such that $fpv(A) \subseteq \mathfrak{D}(v)$, we have $\mathcal{P} = [\![\theta(A)]\!]_v$. Let θ' be any substitution with $fv(A\{x \leftarrow \eta\}) \subseteq \mathfrak{D}(\theta')$ and define $\sigma = \theta'_{-x} \circ \{x \leftarrow \theta'(\eta)\}$. Now, $fv(A) \subseteq \mathfrak{D}(\sigma)$. Thus $\mathcal{P} = [\![\sigma(A)]\!]_v$ for any appropriate valuation v. Since $\sigma(A) = \theta'(A\{x \leftarrow \eta\})$, we are done.
2. Similar to proof of Lemma 4.29 (induction in the size of $F[-]$). ∎

LEMMA 4.29 (Substitutivity). *Let $[\![\theta(A)]\!]_v = [\![\theta(B)]\!]_v$ for all substitutions θ and valuations v, and let $F[-]$ be a formula context. Then, for all substitutions σ and valuations w we have $[\![\sigma(F[A])]\!]_w = [\![\sigma(F[B])]\!]_w$.*

Proof. See appendix. ∎

A direct consequence of substitutivity is

PROPOSITION 4.30 (Replacement of equivalents). *Let $F[-]$ be any formula context. We have $valid(A \Leftrightarrow B) \Rightarrow valid(F[A] \Leftrightarrow F[B])$.*

Proof. Assume $valid(A \Leftrightarrow B)$. Then $[\![\theta(A)]\!]_v = [\![\theta(B)]\!]_v$ for any valuation v for $A \Leftrightarrow B$ and substitution θ. Let w be any valuation for $F[A] \Leftrightarrow F[B]$; we must show that $[\![\sigma(F[A])]\!]_w = [\![\sigma(F[B])]\!]_w$, for any substitution σ. But this follows directly from Lemma 4.29. ∎

5. FRESH AND HIDDEN NAME QUANTIFICATION

In this section the semantics of Section 4 is used to investigate basic properties of the fresh name quantifier and of the derived hidden name quantifier.

5.1. The Fresh Name Quantifier

As we have seen, freshness plays a central role in the spatial logic, but uses of the freshness quantifier $\text{И}x.A$ can be rather subtle. Consider, as an example, the formula $\text{И}x.x\langle m\rangle$, satisfied by any process P such that, for any n fresh in P and different from m, P satisfies $n\langle m\rangle$. But if P satisfies $n\langle m\rangle$, it must be congruent to $n\langle m\rangle$, and hence it must contain n. Therefore, n is not fresh in P, a contradiction. In fact, the denotation of $\text{И}x.x\langle m\rangle$ is empty. This shows that many simple uses of И are vacuous, when the fresh name maps directly to a free name of the process.

There are, however, two basic ways of making good use of the fresh quantifier. The first way is to use И in conjunction with ®, so that the fresh name is used to reveal a restricted name of the process (then the fresh name does not map to a free name of the original process). In this situation, we definitely do not want the name used to reveal a restricted name to clash with some other name of the process. This is one of the reasons that motivates the use of $\setminus\{P \mid n \in fn(P)\}$ in the semantics of $\text{И}x.A$ (Fig. 3), to eliminate such a possibility. The combination of И and ® is discussed further in Section 5.3.

The second way is to use И in conjunction with $\triangleright$, so that the fresh name maps to a free name of the context, but not of the process. For example, consider the formula

$$\text{И}x.\forall y.(x\langle y\rangle \triangleright \boxdot(x\langle y\rangle|\mathbf{T}))$$

This formula holds of all processes P that verify the following: if a message on a fresh channel x is composed in parallel with P, then no reduction from the resulting process consumes such a message.

Intuitively, we expect such a property to hold of every process. In fact, let P be any process, n some name not free in P, and m any name. Pick any process Q such that $Q \models_v n\langle m\rangle$. So, $Q \equiv n\langle m\rangle$. Now, we verify that if $Q|P \to R$, then $R \equiv n\langle m\rangle|P'$, where $P \to P'$, because $P \not\equiv n(q).R'|R''$. Thus $P \models_v n\langle m\rangle \triangleright \boxdot(n\langle m\rangle|\mathbf{T})$. Since m is arbitrary, $P \models_v \forall y.n\langle y\rangle \triangleright \boxdot(n\langle y\rangle|\mathbf{T})$. Since n is neither free in P nor belongs to $fn^v(\forall y.x\langle y\rangle \triangleright \boxdot(x\langle y\rangle|\mathbf{T}))$, we conclude $P \models_v \text{И}x.\forall y.x\langle y\rangle \triangleright \boxdot(x\langle y\rangle|\mathbf{T})$.

A fundamental consequence of closure of satisfaction under fresh renaming (Lemma 4.25) is the following characterisation of fresh name quantification, that makes clear the universal/existential ambivalence of freshness: if some property holds of a fresh name, it holds of all fresh names.

PROPOSITION 5.1 (Gabbay-Pitts Property). *Let $\text{И}x.A$ be a name-closed formula, P a process, and v a valuation for $\text{И}x.A$. Then, the following statements are equivalent*

1. $P \models_v \text{И}x.A$.
2. *There is* $n \notin fn(P) \cup fn^v(A)$ *such that* $P \models_v A\{x\leftarrow n\}$.
3. *For all* $n \notin fn(P) \cup fn^v(A)$ *we have* $P \models_v A\{x\leftarrow n\}$.

Proof. ($1 \Rightarrow 2$) By definition. ($2 \Rightarrow 3$) By Remark 4.19, there is n such that $n \notin fn(P) \cup fn^{v_A}(A)$ and $P \models_{v_A} A\{x\leftarrow n\}$, where v_A is the restriction of v to the free propositional variables of A. Now, pick $m \notin fn(P) \cup fn^{v_A}(A)$, and let $\tau = \{m\leftrightarrow n\}$. By Theorem 4.21(2), we conclude $\tau(P) \models_{\tau(v_A)} \tau(A\{x\leftarrow n\})$, that is, $P \models_{\tau(v_A)} A\{x\leftarrow m\}$. Note that $m, n \notin fn(v_A)$; hence $P \models_{v_A} A\{x\leftarrow m\}$, by Lemma 4.20(1). Hence $P \models_v A\{x\leftarrow m\}$, by Remark 4.19. ($3 \Rightarrow 1$) Immediate. ∎

A corollary of the previous proposition is

PROPOSITION 5.2. *Let A be a name-closed formula and v a valuation for A and B. We have*

1. $[\![\forall x.A]\!]_v \subseteq [\![\reflectbox{N} x.A]\!]_v \subseteq [\![\exists x.A]\!]_v$
2. $[\![\reflectbox{N} x.(A \Rightarrow B)]\!]_v = [\![\reflectbox{N} x.A \Rightarrow \reflectbox{N} x.B]\!]_v$

Proof. 2. (Left to right) Assume $P \in [\![\reflectbox{N} x.(A \Rightarrow B)]\!]_v$ and $P \in [\![\reflectbox{N} x.A]\!]_v$. Then $P \in [\![A\{x\leftarrow n\}]\!]_v$ for some $n \notin fn(P) \cup fn^v(A)$, and $P \in [\![A\{x\leftarrow m\} \Rightarrow B\{x\leftarrow m\}]\!]_v$ for some $m \notin fn(P) \cup fn^v(A \Rightarrow B)$. By Proposition 5.1(3), for all $n \notin fn(P) \cup fn^v(A)$ we have $P \in [\![A\{x\leftarrow n\}]\!]_v$. In particular, $P \in [\![A\{x\leftarrow m\}]\!]_v$, thus $P \in [\![B\{x\leftarrow m\}]\!]_v$. We conclude $P \in [\![\reflectbox{N} x.B]\!]_v$. (Right to left) Assume $P \in [\![\reflectbox{N} x.A \Rightarrow \reflectbox{N} x.B]\!]_v$. Pick $m \notin fn(P) \cup fn^v(A \Rightarrow B)$ and assume $P \in [\![A\{x\leftarrow m\}]\!]_v$. Then $P \in [\![\reflectbox{N} x.A]\!]_v$, this implies $P \in [\![\reflectbox{N} x.B]\!]_v$. By Proposition 5.1(3), $P \in [\![B\{x\leftarrow m\}]\!]_v$. Hence $P \in [\![\reflectbox{N} x.(A \Rightarrow B)]\!]_v$. ∎

Fresh quantification distributes over all boolean connectives, not only implication (*cf.*, Proposition 5.2(2), it suffices to note that (trivially) $[\![\reflectbox{N} x.\mathbf{F}]\!]_v = [\![\mathbf{F}]\!]_v$). In the next lemma, we list some other distribution properties of freshness quantification.

LEMMA 5.3 (Distribution properties of $\reflectbox{N}$). *We have*

1. $[\![\reflectbox{N} x.(A|B)]\!]_v = [\![\reflectbox{N} x.A|\reflectbox{N} x.B]\!]_v$
2. $[\![\reflectbox{N} x.(A \triangleright B)]\!]_v \subseteq [\![\reflectbox{N} x.A \triangleright \reflectbox{N} x.B]\!]_v$
3. $[\![\reflectbox{N} x.\Diamond A]\!]_v = [\![\Diamond \reflectbox{N} x.A]\!]_v$
4. $[\![\reflectbox{N} x.n \circledR A]\!]_v = [\![n \circledR \reflectbox{N} x.A]\!]_v$
5. $[\![\reflectbox{N} x.\forall y.A]\!]_v \subseteq [\![\forall y.\reflectbox{N} x.A]\!]_v$
6. $[\![\reflectbox{N} x.\forall X.A]\!]_v \subseteq [\![\forall X.\reflectbox{N} x.A]\!]_v$

Proof. See Appendix. ∎

It is not hard to see that properties 5. and 6. above are not strict equalities: for a counterexample to $\forall y.\reflectbox{N} x.A \subseteq \reflectbox{N} x.\forall y.A$ consider, *e.g.*, $A \triangleq x\#y$.

5.2. Discussion

In [19] a $\reflectbox{N}$-quantifier is defined, such that

$$\reflectbox{N} x.A \Leftrightarrow \{n \mid A\{x\leftarrow n\}\} \text{ is cofinite}$$

There is a quite close connection between this $\reflectbox{N}$-quantifier and ours, superficial differences being related to the fact that we are working in a modal logic. In our case, we have

PROPOSITION 5.4. $P \models_v \reflectbox{N} x.A$ *if and only if* $\{n \mid P \models_v A\{x\leftarrow n\}\}$ *is cofinite.*

Proof. (Left to right) Pick $P \models_v \reflectbox{N} x.A$. Thus there is $n \notin fn^v(A) \cup fv(P)$ such that $P \models_v A\{x\leftarrow n\}$. By Gabbay-Pitts (Proposition 5.1(3)) we have that for all n if $n \notin fn^v(A) \cup fv(P)$ then $P \models_v A\{x\leftarrow n\}$. Hence $\{n \mid P \models_v A\{x\leftarrow n\}\}$ is cofinite. (Right to left) Assume $S = \{n \mid P \models_v A\{x\leftarrow n\}\}$ is cofinite. Then, there is a finite set $M (= \Lambda \setminus S)$ such that for all n, if $n \notin M$ then $P \models_v A\{x\leftarrow n\}$. Pick $m \notin fn^v(A) \cup fn(P) \cup M$. Then $P \models_v A\{x\leftarrow m\}$, hence $P \models_v \reflectbox{N} x.A$. ∎

Now, let us define the following (meta-level) quantifier

$$\reflectbox{N}^* x.\mathcal{B} \triangleq \{n \in \Lambda \mid \mathcal{B}\{x\leftarrow n\}\} \text{ is cofinite}$$

where $\mathcal{B}$ is a meta-level statement of the (informal) theory of Psets of Section 4. Note that $\text{И}^*x.\mathcal{B}$ is defined exactly as the И-quantifier of Gabbay-Pitts. Then, we can read the statement of the previous proposition as:

$$P \in [\![\text{И}x.A]\!]_v \text{ if and only if } \text{И}^*n.(P \in [\![A\{x\leftarrow n\}]\!]_v)$$

It is interesting to discuss alternative freshness quantifiers. Our semantics of $\text{И}x.A$ is such that $P \models_v \text{И}x.A$ holds if and only if there is a name n, fresh both in A and P, such that $P \models A\{x\leftarrow n\}$ (*cf.*, Proposition 5.1). It is natural then to ask what happens if n only is required to be fresh in A. Let us define for this propose a different quantifier $\mathsf{F}x.A$ where

$$P \in [\![\mathsf{F}x.A]\!] \text{ if and only if } \exists n \notin fn(A) \text{ such that } P \in [\![A\{x\leftarrow n\}]\!]$$

One could then attempt to define $\text{И}x.A$ as $\mathsf{F}x.(A\wedge\neg©x)$. Although $[\![\mathsf{F}x.A]\!]$ is a Pset, the main problems with $\mathsf{F}x.A$, with respect to $\text{И}x.A$, are a failure of monotonicity (Proposition 6.5), a failure of the substitutivity property (Lemma 4.29), and a failure of the Gabbay-Pitts property (Proposition 5.1) relating to a proper notion of "freshness".

For substitutivity, we have that $[\![n\langle n\rangle \vee \neg n\langle n\rangle]\!] = [\![\mathbf{T}]\!]$. So, we would expect that $[\![\mathsf{F}x.((n\langle n\rangle \vee \neg n\langle n\rangle) \wedge x\langle x\rangle)]\!] = [\![\mathsf{F}x.(\mathbf{T} \wedge x\langle x\rangle)]\!]$. But $n\langle n\rangle \in [\![\mathsf{F}x.(\mathbf{T} \wedge x\langle x\rangle)]\!]$, while $n\langle n\rangle \notin [\![\mathsf{F}x.((n\langle n\rangle\vee\neg n\langle n\rangle)\wedge x\langle x\rangle)]\!]$. So, $\mathsf{F}x.A$ is not a proper "compositional" logical operator. While, rather amazingly, $\text{И}x.A$ is.

For monotonicity, consider

$$\psi = \{q\langle q\rangle\}^{\equiv} \subseteq \{p\langle p\rangle, q\langle q\rangle\}^{\equiv} = \phi$$

Note that $\psi, \phi \in \mathbb{P}_{\{p,q\}}$, $fn^{v[X\leftarrow\psi]}(X|x\langle x\rangle) = \{q\}$ and $fn^{v[X\leftarrow\phi]}(X|x\langle x\rangle) = \{p, q\}$. On the one hand, $q\langle q\rangle|p\langle p\rangle \in [\![\mathsf{F}x.X|x\langle x\rangle]\!]_{v[X\leftarrow\psi]}$, because there is $n \notin \{q\}$ (namely p) such that $q\langle q\rangle|p\langle p\rangle \in [\![X|n\langle n\rangle]\!]_{v[X\leftarrow\psi]}$. On the other hand, we have $q\langle q\rangle|p\langle p\rangle \notin [\![\mathsf{F}x.(X|x\langle x\rangle)]\!]_{v[X\leftarrow\Phi]}$, because there is no n out of $\{p, q\}$ such that $q\langle q\rangle|p\langle p\rangle \in [\![X|n\langle n\rangle]\!]_{v[X\leftarrow\Phi]}$. So $[\![\mathsf{F}x.(X|x\langle x\rangle)]\!]_{v[X\leftarrow\psi]} \not\subseteq [\![\mathsf{F}x.(X|x\langle x\rangle)]\!]_{v[X\leftarrow\phi]}$. We conclude that $\mathsf{F}x.A$ cannot be used with recursive formulas.

For the Gabbay-Pitts property, consider whether $p\langle p\rangle \in [\![\mathsf{F}x.\neg x\langle x\rangle]\!]$. This means, by definition: there is a name n such that $p\langle p\rangle \in [\![\neg n\langle n\rangle]\!]$. This is true, take any $n \neq p$. If we had a Gabbay-Pitts property for $\mathsf{F}x.A$ we would obtain that for all names n, $p\langle p\rangle \in [\![n\langle n\rangle]\!]$. But this is false: take $n = p$. So, by the interpretation of the Gabbay-Pitts property, the candidate $\mathsf{F}x.A$ is not a proper "freshness" quantifier.

5.3. The Hidden Name Quantifier

When combined with revelation, the fresh name quantifier gives rise to a natural operation of quantification over hidden (restricted) names in a process. Intuitively, a hidden name is revealed under a fresh identity, and then a property is asserted of the process where the name is hidden.

$$\mathsf{H}x.A \triangleq \text{И}x.x®A$$

A formula $\mathsf{H}x.A$ reads "there is a restricted name x such that A holds for the process under the restriction". From the above definition, we get the following direct semantic characterization of the name-closed formula $\mathsf{H}x.A$

$$[\![\mathsf{H}x.A]\!]_v = \{Q \mid Q \equiv (\boldsymbol{\nu}n)P \text{ and } n \notin fn(Q) \cup fn^v(A) \text{ and } P \in [\![A\{x\leftarrow n\}]\!]_v\}$$

The hidden name quantifier makes it possible to express properties of processes that depend on (or need to mention) some secret name. For a quite simple example, consider the closed formula

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

We can verify that a process satisfies this formula if there is some name n such that P satisfies the formula $\mathsf{H}x.(n\langle x\rangle|\mathbf{T})$. But this means that there is some name m, fresh with respect to P and $\mathsf{H}x.(n\langle x\rangle|\mathbf{T})$, such that $P \equiv (\boldsymbol{\nu}n)Q$ and $Q \equiv n\langle m\rangle|R$ for some Q and R. In summary, P satisfies $\exists y.\mathsf{H}x.(y\langle x\rangle|\mathbf{T})$ if and only if $P \equiv (\boldsymbol{\nu}m)(n\langle m\rangle|R)$ for some m and $n \neq m$ (hence n is public). We conclude that the formula $\exists y.\mathsf{H}x.(y\langle x\rangle|\mathbf{T})$ is satisfied by those processes that are ready to send a *secret* name over a *public* channel.

As a further example, let $A\{x\}$ be some formula with a free occurrence of the name variable x, and consider

$$\mathit{Keeps}(A\{x\}) \triangleq \mathsf{H}x.A\{x\} \wedge (\mathbf{T} \triangleright \Diamond \mathsf{H}x.A\{x\}|\mathbf{T})$$

A process that satisfies $\mathit{Keeps}(A\{x\})$ is always able to guarantee, until the next step, persistence of property A with respect to some secret x it owns, even when attacked by some other arbitrary process. Let Q be the process

$$(\boldsymbol{\nu}m)(m\langle n\rangle|!a(p).m(q).m\langle p\rangle)$$

We have $\mathit{fn}(Q) = \{a, n\}$. Now define $\mathit{Msg}(x,y) \triangleq (x\langle y\rangle|\mathbf{T})$. We can verify that Q satisfies $\mathit{Keeps}(\exists y.\mathit{Msg}(x,y))$.

As a further example, consider the formula

$$\mathit{NoRes} \triangleq \neg\mathsf{H}x.©x$$

A process P satisfies *NoRes* if and only if it is not the case that there is a process Q and a name n such that $P \equiv (\boldsymbol{\nu}n)Q$ and $n \in \mathit{fn}(Q)$. In other words, P satisfies *NoRes* if and only if for all processes Q and names n such that $P \equiv (\boldsymbol{\nu}n)Q$ we have $n \notin \mathit{fn}(Q)$. Intuitively, this means that P has no "genuine" restricted hidden name at the outermost level, because if $P \equiv (\boldsymbol{\nu}n)Q$ for some Q and n, then $P \equiv Q$ ($n \notin \mathit{fn}(Q)$ implies $(\boldsymbol{\nu}n)Q \equiv Q$). So, we will call *restriction-free* any process that satisfies *NoRes*. For instance $n\langle m\rangle$ satisfies *NoRes*, and so does $(\boldsymbol{\nu}n)m\langle m\rangle$ if $m \neq n$, but $(\boldsymbol{\nu}n)n\langle n\rangle$ does not.

LEMMA 5.5 (Some properties of H). *We have*

1. $[\![\mathsf{H}x.(A \wedge \neg©x)]\!]_v = [\![\text{И}x.A]\!]_v$
2. *If* $x \notin \mathit{fn}(B)$ *then* $[\![\mathsf{H}x.(A|(B \wedge \neg©x))]\!]_v = [\![(\mathsf{H}x.A)|B]\!]_v$
3. $[\![\mathsf{H}x.A|\text{И}x.B]\!]_v \subseteq [\![\mathsf{H}x.(A|B)]\!]_v$
4. $[\![\mathsf{H}x.\Diamond A]\!]_v = [\![\Diamond\mathsf{H}x.A]\!]_v$

Proof. 1. (Left to right inclusion) Pick some process $P \in [\![\mathsf{H}x.(A \wedge \neg©x)]\!]_v$. By the characterization given above, this means that $P \equiv (\boldsymbol{\nu}n)Q$ for some Q and n such that $n \notin \mathit{fn}(P) \cup \mathit{fn}^v(A)$ and $Q \in [\![A\{x\leftarrow n\} \wedge ©n]\!]_v$. But then, $n \notin \mathit{fn}(Q)$ and $Q \in [\![A\{x\leftarrow n\}]\!]_v$. We conclude $P \equiv (\boldsymbol{\nu}n)Q \equiv Q$, by Proposition 2.12(2). Therefore, $P \in [\![\text{И}x.A]\!]_v$. In the other direction the proof is similar.

2. (Left to right inclusion) Let $P \in [\![\mathsf{H}x.(A|(B \wedge \neg ©x))]\!]_v$. Then $P \equiv (\boldsymbol{\nu} n)(Q|R)$, here $n \notin fn(P) \cup fn^v(\mathsf{H}x.(A|(B \wedge \neg ©x)))$, $Q \models_v A$, $R \models_v B$ and $n \notin fn(R)$. Then $P \equiv (\boldsymbol{\nu} n)Q|R$, and we conclude $P \models_v \mathsf{H}x.A|B$. The converse inclusion is also immediate, using Proposition 5.1.

3. (Left to right inclusion) Let $P \in [\![\mathsf{H}x.A|\text{И}x.B]\!]_v$. Then $P \equiv Q|R$ where for all $n \notin fn(Q) \cup fn^v(A)$ there is Q' such that $Q \equiv (\boldsymbol{\nu} n)Q'$ and $Q' \models_v A\{x \leftarrow n\}$, and $R \models B\{x \leftarrow p\}$ for all $p \notin fn(R) \cup fn^v(B)$, by Proposition 5.1. Pick $m \notin fn(Q) \cup fn^v(A) \cup fn^v(B) \cup fn(R)$. Hence $R \models_v B\{x \leftarrow m\}$. Moreover, there is Q'' such that $Q \equiv (\boldsymbol{\nu} m)Q''$ and $Q'' \models_v A\{x \leftarrow p\}$. So $Q''|R \models (A|B)\{x \leftarrow m\}$, and thus $(\boldsymbol{\nu} m)(Q''|R) \models \mathsf{H}x.(A|B)$. To conclude, note that $P \equiv (\boldsymbol{\nu} m)(Q''|R)$.

4. (Left to right inclusion) Pick some process $P \in [\![\mathsf{H}x.\Diamond A]\!]_v$. So, $P \equiv (\boldsymbol{\nu} n)Q$ for some Q and n such that $Q \to Q'$ and $Q' \in [\![A\{x \leftarrow n\}]\!]_v$, where n is fresh with respect to P and A. But then $P \to (\boldsymbol{\nu} n)Q'$. Since n is also fresh w.r.t. $(\boldsymbol{\nu} n)Q'$, we conclude $(\boldsymbol{\nu} n)Q' \in [\![\mathsf{H}x.A]\!]_v$. Hence $P \in [\![\Diamond \mathsf{H}x.A]\!]_v$. (Right to left inclusion) Take some process $P \in [\![\Diamond \mathsf{H}x.A]\!]_v$. Then there is Q such that $P \to Q$ and $Q \in [\![\mathsf{H}x.A]\!]_v$. Then $Q \equiv (\boldsymbol{\nu} n)R$ where $R \in [\![A\{x \leftarrow n\}]\!]_v$ and $n \notin fn(Q) \cup fn^v(A)$. Now, since $P \to Q$, by Proposition 2.15(3) there are P' and R' such that $P \equiv (\boldsymbol{\nu} n)P'$, $P' \to R'$ and $R' \equiv R$. This means that $P \in [\![\mathsf{H}x.\Diamond A]\!]$. ∎

In the next section, we give further examples using the hidden name quantifier together with recursion.

6. RECURSIVE DEFINITIONS

The possibility of defining properties by induction and coinduction is a major source of expressiveness of our spatial logic. Of particular interest is the combination of properties involving recursion and freshness.

6.1. Encoding Recursion

We show that recursive definitions can be expressed in the logic via second order quantification and the guarantee operator. We begin by encoding validity of a formula A by $\sim\neg A$ (see Section 4.2.1):

$$!A \triangleq (A \Rightarrow \mathbf{F}) \triangleright \mathbf{F} \qquad \text{(Validity)}$$

We compute:

$$\begin{aligned}
[\![!A]\!]_v &= [\![(A \Rightarrow \mathbf{F}) \triangleright \mathbf{F}]\!]_v \\
&= \{P \mid \textit{Forall } R.R \in [\![A \Rightarrow \mathbf{F}]\!]_v \Rightarrow P|R \in \emptyset\} \\
&= \{P \mid \textit{Forall } R.\neg R \in [\![A \Rightarrow \mathbf{F}]\!]_v\} \\
&= \{P \mid \textit{Forall } R.R \in [\![A]\!]_v\} \\
&= \textit{if } [\![A]\!]_v = \mathcal{P} \textit{ then } \mathcal{P} \textit{ else } \emptyset
\end{aligned}$$

Next, we can use $[\![!(A \Rightarrow B)]\!]_v$ to say that $[\![A]\!]_v$ is included in $[\![B]\!]_v$:

$$A \mapsto B \triangleq !(A \Rightarrow B) \qquad \text{(Entailment)}$$

Then

$$\begin{aligned}
[\![A \mapsto B]\!]_v &= [\![!(A \Rightarrow B)]\!]_v \\
&= \textit{if } [\![A \Rightarrow B]\!]_v = \mathcal{P} \textit{ then } \mathcal{P} \textit{ else } \emptyset \\
&= \textit{if } \{P \mid P \in [\![A]\!]_v \Rightarrow P \in [\![B]\!]_v\} = \mathcal{P} \textit{ then } \mathcal{P} \textit{ else } \emptyset \\
&= \textit{if } [\![A]\!]_v \subseteq [\![B]\!]_v \textit{ then } \mathcal{P} \textit{ else } \emptyset
\end{aligned}$$

Finally, we can use the following formula to define the greatest property Y such that $Y \Leftrightarrow A$, provided A is monotonic in Y:

$$\boldsymbol{\nu} Y.A \triangleq \exists Y.Y \wedge (Y \mapsto A) \qquad \text{(Greatest fixpoint)}$$

We can verify

$$\begin{array}{l} [\![\boldsymbol{\nu} Y.A]\!]_v = [\![\exists Y.Y \wedge (Y \mapsto A)]\!]_v = \\ \bigcup_{\Phi \in \mathbb{P}_-} \Phi \cap (\mathit{if}\ \Phi \subseteq [\![A]\!]_{v[Y \leftarrow \Phi]}\ \mathit{then}\ \mathcal{P}\ \mathit{else}\ \emptyset) = \\ \bigcup \{\Phi \mid \Phi \in \mathbb{P}_-\ \mathit{and}\ \Phi \subseteq [\![A]\!]_{v[Y \leftarrow \Phi]}\} \end{array}$$

We now show that the last line above defines a greatest fixpoint.

LEMMA 6.1. *For any formula A and valuation v with $fpv(A) \subseteq \mathfrak{D}(v) \cup \{X\}$ the mapping $\mathfrak{F}^v_{A(X)}$ given by*

$$\mathfrak{F}^v_{A(X)}(\Psi) \triangleq [\![A]\!]_{v[X \leftarrow \Psi]}$$

is a mapping $\mathbb{P}_- \to \mathbb{P}_-$.

Proof. Let $\Psi \in \mathbb{P}_-$. Then $\Psi \in \mathbb{P}_M$ for some finite set of names M. Let $M' = M \cup fn^v(A)$. Since $fn^{v[X \leftarrow \Psi]}(A) \subseteq M \cup M'$ by Theorem 4.21(1) we conclude $[\![A]\!]_{v[X \leftarrow \Psi]} \in \mathbb{P}_{M \cup M'} \subseteq \mathbb{P}_-$. ∎

A mapping $f : \mathbb{P}_- \to \mathbb{P}_-$ is *monotonic* if $\Psi \subseteq \Phi$ implies $f(\Psi) \subseteq f(\Phi)$ for all Psets Ψ and Φ. For any mapping $f : \mathbb{P}_- \to \mathbb{P}_-$, a *fixpoint* of f is a Pset $\Psi \in \mathbb{P}_-$ such that $f(\Psi) = \Psi$. The $\subseteq$-greatest (respectively $\subseteq$-smallest) fixpoint of f, if it exists, is denoted by $\mathsf{gfix}(f)$ (respectively $\mathsf{lfix}(f)$). We say that a formula A is monotonic in X if for all valuations v the mapping $\mathfrak{F}^v_{A(X)}$ is monotonic. We have

LEMMA 6.2. *Let A be a formula monotonic in X. Then $\mathfrak{F}^v_{A(X)}$ has a unique greatest fixpoint given by*

$$\mathsf{gfix}(\mathfrak{F}^v_{A(X)}) = [\![\boldsymbol{\nu} X.A]\!]_v$$

Proof. First, note that although $\mathbb{P}_-$ is not a complete lattice we still have $[\![\boldsymbol{\nu} X.A]\!]_v \in \mathbb{P}_-$ by Theorem 4.21 (1), since $\boldsymbol{\nu} X.A$ is definable in the logic. Let $\mathfrak{G} \triangleq [\![\boldsymbol{\nu} X.A]\!]_v$ and $\Phi \triangleq \mathfrak{F}^v_{A(X)}(\mathfrak{G}) = [\![A]\!]_{v[X \leftarrow \mathfrak{G}]}$. We verify that $\mathfrak{G} = \Phi$. We first check that $\mathfrak{G} \subseteq \Phi$. If $P \in \mathfrak{G}$ then $P \in \Psi$ for some $\Psi \in \mathbb{P}_-$ such that $\Psi \subseteq [\![A]\!]_{v[X \leftarrow \Psi]}$. Since $\Psi \subseteq \mathfrak{G}$, by monotonicity, we have $\Psi \subseteq [\![A]\!]_{v[X \leftarrow \mathfrak{G}]} = \Phi$ and thus $P \in \Phi$. On the other hand, since $\mathfrak{G} \subseteq \Phi$, by monotonicity, we have $[\![A]\!]_{v[X \leftarrow \mathfrak{G}]} = \Phi \subseteq [\![A]\!]_{v[X \leftarrow \Phi]}$. Then $\Phi \subseteq [\![\boldsymbol{\nu} X.A]\!]_v = \mathfrak{G}$. Finally, if some $\Psi \in \mathbb{P}_-$ verifies $\Psi = [\![A]\!]_{v[X \leftarrow \Psi]}$ then $\Psi \subseteq \mathfrak{G}$. Hence we conclude the result. ∎

Note that Lemmas 4.24 and 6.2 imply soundness of the unfolding principle for $\boldsymbol{\nu} X.A$, that is we have $[\![\boldsymbol{\nu} X.A]\!]_v = [\![A\{X \leftarrow \boldsymbol{\nu} X.A\}]\!]_v$.

Similarly we can define the least fixpoint operator

$$\boldsymbol{\mu} Y.A \triangleq \forall Y.(A \mapsto Y) \Rightarrow Y \qquad \text{(Least fixpoint)}$$

and note that $[\![\boldsymbol{\mu} Y.A]\!]_v = \bigcap \{\Phi \mid \Phi \in \mathbb{P}_-\ \mathit{and}\ [\![A]\!]_{v[Y \leftarrow \Phi]} \subseteq \Phi\}$. We then have

PROPOSITION 6.3 (Induction and Coinduction). *Let the formula F be monotonic in X. For any formula A such that $X \notin fpv(A)$, we have*

1. *$valid(F\{X \leftarrow A\} \Rightarrow A) \Rightarrow valid(\boldsymbol{\mu} X.F \Rightarrow A)$*

C	$\mathrm{Neg}(C)$	$\mathrm{Pos}(C)$
$\mathbf{F}$ $\mathbf{0}$ $\eta\langle\eta'\rangle$	$\emptyset$	$\emptyset$
$A \wedge B$ $A\|B$	$\mathrm{Neg}(A) \cup \mathrm{Neg}(B)$	$\mathrm{Pos}(A) \cup \mathrm{Pos}(B)$
$A \Rightarrow B$ $A \triangleright B$	$\mathrm{Pos}(A) \cup \mathrm{Neg}(B)$	$\mathrm{Neg}(A) \cup \mathrm{Pos}(B)$
n®A $A \oslash n$ $\forall x.A$ $Иx.A$ $\Diamond A$	$\mathrm{Neg}(A)$	$\mathrm{Pos}(A)$
X	$\emptyset$	$\{X\}$
$\forall X.A$	$\mathrm{Neg}(A) \setminus \{X\}$	$\mathrm{Pos}(A) \setminus \{X\}$

FIG. 4 Negative and Positive occurrences.

2. *valid*$(A \Rightarrow F\{X \leftarrow A\}) \Rightarrow$ *valid*$(A \Rightarrow \boldsymbol{\nu} X.F)$

Proof. (1) We assume *valid*$(A \Rightarrow F\{X \leftarrow A\})$, and prove *valid*$(A \Rightarrow \boldsymbol{\nu} X.F)$. To that end, we select any valuation v for $A \Rightarrow \boldsymbol{\nu} X.F$, any substitution θ and show $[\![\theta(A)]\!]_v \subseteq [\![\theta(\boldsymbol{\nu} X.F)]\!]_v$. We have

$$[\![\theta(A)]\!]_v \subseteq [\![\theta(F\{X \leftarrow A\}))]\!]_v = [\![\theta(F)\{X \leftarrow \theta(A)\}]\!]_v$$

By Lemma 4.24, $[\![\theta(F)\{X \leftarrow \theta(A)\}]\!]_v = [\![\theta(F)]\!]_{v[X \leftarrow [\![\theta(A)]\!]_v]}$. By assumption we have $[\![\theta(A)]\!]_v \subseteq [\![\theta(F)]\!]_{v[X \leftarrow [\![\theta(A)]\!]_v]}$. Hence, $[\![\theta(A)]\!]_v \subseteq [\![\theta(\boldsymbol{\nu} X.F)]\!]_v$. (2) Similar to (1). ∎

Along usual lines, some syntactical conditions on the free occurrences of X in a formula A can be imposed in order to ensure monotonicity on X of A.

Definition 6.4 (Negative and Positive Occurrences). *For any formula C, the set* $\mathrm{Neg}(C)$ *(resp.* $\mathrm{Pos}(C)$*) of the variables which occur negatively (resp. positively) in C are inductively defined in Fig. 4.*

We say that a propositional variable X is *positive* (resp. *negative*) in A if $X \in \mathrm{Pos}(A)$ (resp. $X \in \mathrm{Neg}(A)$). We also say that a formula A is *monotonic in* X (resp. *antimonotonic in* X) whenever $X \notin \mathrm{Neg}(A)$ (resp. $X \notin \mathrm{Pos}(A)$). Note that a variable X can be both positive and negative in a formula A. Moreover, if X is either positive or negative in A then $X \in$ *fpv*(A). We have

Proposition 6.5 (Monotonicity). *For all formulas A, appropriate valuations v, and Psets Ψ, Φ*

1. *If $X \notin \mathrm{Neg}(A)$ and $\Psi \subseteq \Phi$ then $[\![A]\!]_{v[X \leftarrow \Psi]} \subseteq [\![A]\!]_{v[X \leftarrow \Phi]}$.*

2. *If $X \notin \mathrm{Pos}(A)$ and $\Psi \subseteq \Phi$ then $[\![A]\!]_{v[X \leftarrow \Phi]} \subseteq [\![A]\!]_{v[X \leftarrow \Psi]}$.*

Proof. See Appendix. ∎

Alternatively, a semantic monotonicity property can be expressed within the logic. Then, the definition of fixpoints and the derivation of their properties (e.g.

unfolding) can be carried out entirely within the logic (by carrying along monotonicity assumptions), without relying on metatheorems such as Proposition 6.5. To this end, for any formula A with Z free, and X, Y fresh we can define:

$$A\{Z+\} \triangleq\ !\forall X.\forall Y.(X \Rrightarrow Y) \Rightarrow (A\{Z{\leftarrow}X\} \Rrightarrow A\{Z{\leftarrow}Y\})$$

We can verify:

$$[\![A\{Z+\}]\!]_v = \textit{if (Forall } \Phi \textit{ and } \Psi.\ \Phi \subseteq \Psi \Rightarrow [\![A]\!]_{v[Z\leftarrow\Phi]} \subseteq [\![A]\!]_{v[Z\leftarrow\Psi]}) \textit{ then } \mathcal{P} \textit{ else } \emptyset$$

where the formula in parentheses expresses the monotonicity of the mapping $\Phi \mapsto [\![A]\!]_{v[Z\leftarrow\Phi]}$, which is from $\mathbb{P}_-$ to $\mathbb{P}_-$ by construction.

6.2. Recursion and Freshness

The semantics of the fresh quantifier $\text{И}x.A$ is based on finding fresh names outside of $fn^v(A) = fn^v(\text{И}x.A)$, and therefore outside the support of $[\![\text{И}x.A]\!]_v$ (by Lemma 4.22). The fact that names outside the support can be freely renamed (*cf.*, Theorem 4.21(2) and Lemma 4.25) implies that any choice for a fresh name will work equally well.

It is instructive to see how freshness interacts with recursion. Consider the formula

$$\boldsymbol{\nu}Y.\text{И}x.n\langle x\rangle \triangleright \Diamond Y$$

By the fixpoint unfolding property, this formula must have the same meaning as

$$\text{И}x.n\langle x\rangle \triangleright \Diamond(\text{И}x'.n\langle x'\rangle \triangleright \Diamond \cdots)$$

Obviously, the $fn^-(-)$ of the original formula and of its expansion are the same. So, "at each iteration" we have to choose fresh names outside the same set of names, and there is an infinite supply of them. Moreover, at each iteration, the $\setminus\{P \mid n \in fn(P)\}$ part of the semantic definition of $\text{И}x.A$ subtracts those processes that use names that have been chosen in previous iterations. Further, since the fresh names used at each iteration can be freely renamed, they do not affect the support set, by Definition 4.1(2).

As already discussed, the notion of finite support and Pset seems crucial for the semantics of $\text{И}x.A$. In particular, without a notion of finite support (*cf.*, Definition 4.1(2)), it seems natural to set $fn^v(X) \triangleq \{n \mid n \in fn(P) \text{ and } P \in v(X)\}$, since we must somehow take into account the names contributed by (a binding of) X. Then, consider the set $\Psi = \{p\langle p\rangle\}^{\equiv}$ and the formula $A = \text{И}x.(\neg x\langle x\rangle | X)$, with $fn^{[X\leftarrow\Psi]}(A) = \{p\}$. We can easily check that

$$q\langle q\rangle|p\langle p\rangle \in [\![A]\!]_{[X\leftarrow\Psi]} = \bigcup_{n\notin\{p\}} ([\![\neg n\langle n\rangle|X]\!]_{[X\leftarrow\Psi]} \setminus \{P \mid n \in fn(P)\})$$

Now consider $\Phi = \{r\langle r\rangle \mid r \in \Lambda\}^{\equiv}$ with $fn^{[X\leftarrow\Phi]}(A) = \Lambda$. So we have that $q\langle q\rangle|p\langle p\rangle \notin [\![A]\!]_{[X\leftarrow\Phi]} = \emptyset$. Hence, $\Psi \subseteq \Phi$, but $[\![A]\!]_{[X\leftarrow\Psi]} \not\subseteq [\![A]\!]_{[X\leftarrow\Phi]}$; a failure of monotonicity.

Instead, in our semantics $\text{И}x.A$ is a monotonic operator, (see Proposition 6.5 (1,2)), the functional associated with a fixpoint formula $\boldsymbol{\nu}X.A$ is in fact a monotonic operator $\mathbb{P}_- \to \mathbb{P}_-$ (see Lemma 6.1).

6.3. Using Recursion

By combining fixpoints formulas with hidden name quantification we can describe a "nonce generator", that is, a process that sends an unbounded number of fresh names on a channel:

$$\boldsymbol{\nu} X.\mathsf{H}x.m\langle x\rangle | X$$

We can verify that $!(\boldsymbol{\nu} n)m\langle n\rangle \models \boldsymbol{\nu} X.\mathsf{H}x.m\langle x\rangle | X$. Let $\Phi = \{P \mid P \equiv\, !(\boldsymbol{\nu} n)m\langle n\rangle\}$. We have that $\Phi \in \mathbb{P}_{\{m\}}$. It suffices to check that $\Phi \subseteq [\![\mathsf{H}x.m\langle x\rangle | X]\!]_{[X\leftarrow\Phi]}$. For this, take any $P \in \Phi$, so that we have $P \equiv\, !(\boldsymbol{\nu} n)m\langle n\rangle \equiv (\boldsymbol{\nu} n)m\langle n\rangle | !(\boldsymbol{\nu} n)m\langle n\rangle$. The left subprocess is in $[\![\mathsf{H}x.m\langle x\rangle]\!]_{[X\leftarrow\Phi]}$, and the right one is in $\Phi = [\![X]\!]_{[X\leftarrow\Phi]}$. More generally, we can verify that $P \models A$ implies $!P \models \boldsymbol{\nu} X.(A|X)$.

The standard "always in the future" $\Box A$ and "eventually in the future" $\Diamond A$ modalities of (branching time) temporal logic are defined as usual:

$$\begin{array}{lcll} \Box A & \triangleq & \boldsymbol{\nu} X.(A \wedge \boxdot X) & \text{(Always)} \\ \Diamond A & \triangleq & \boldsymbol{\mu} X.(A \vee \Diamond\!\!\cdot\, X) & \text{(Sometime)} \end{array}$$

For these connectives we have

$P \in [\![\Box A]\!]_v$ if and only if for all Q such that $P \xrightarrow{*} Q$ we have $Q \in [\![A]\!]_v$

$P \in [\![\Diamond A]\!]_v$ if and only if there is Q such that $P \xrightarrow{*} Q$ and $Q \in [\![A]\!]_v$

By combining fixpoints with spatial and temporal connectives one can define many interesting properties. For instance, consider the following formulas:

$$\mathit{Client} \triangleq \mathsf{H}x.(\mathit{Proto}(x)|\mathit{Request}(x))$$

$$\mathit{Server} \triangleq \boldsymbol{\nu} Y.\mathsf{И}x.\mathit{Proto}(x) \triangleright \Diamond(\mathit{Handler}(x)|Y)$$

A process P realizes the *Client* specification if it is built out of two components satisfying $\mathit{Proto}(x)$ and $\mathit{Request}(x)$ where x is a hidden name of P. The intention is that $\mathit{Proto}(x)$ specifies some protocol that P can perform with its environment, making use of the hidden name x, while $\mathit{Request}(x)$ describes further properties of P with respect to x. For example, we may have

$$\mathit{Proto}(x) \triangleq n\langle x\rangle$$

meaning that the protocol is just to send the name x on channel n.

Formula *Server* specifies a somewhat more involved property. By unfolding the recursive definition, we conclude that *Server* denotes the greatest property such that

$$\mathit{Server} \Leftrightarrow \mathsf{И}y.\mathit{Proto}(y) \triangleright \Diamond(\mathit{Handler}(y)|\mathit{Server})$$

Hence, a process Q satisfies the *Server* specification if in presence of another process executing $\mathit{Proto}(y)$ for some/any fresh name y, Q is guaranteed to evolve to a system composed of two parts, one satisfying property $\mathit{Handler}(y)$, and other satisfying property *Server* (for example, a copy of the initial process). Note that the guarantee is provided just for protocols that play according to $\mathit{Proto}(y)$ *for a fresh name* (nonce) y (fresh with respect to the *Server*). We can then verify that the following entailment is valid

$$\mathit{Server}|\mathit{Client} \Rightarrow \Diamond(\mathit{Server}|\mathsf{H}x.(\mathit{Request}(x)|\mathit{Handler}(x)))$$

This formula means that any system composed by a *Client* and a *Server* is guaranteed to possibly evolve to a configuration where a *Server* persists along with a pair of processes, $Request(x)$ and $Handler(x)$, that share a secret x.

We can also introduce the following derived operators and study their characterization

$$
\begin{array}{llll}
\boldsymbol{group}\ A & \triangleq & \boldsymbol{\mu} X.(\mathbf{0} \vee A|X) & \text{(Group)} \\
\boldsymbol{inside}^{\exists}\ A & \triangleq & \boldsymbol{\mu} X.(A \vee \mathsf{H}x.X) & \\
\boldsymbol{inside}^{\forall}\ A & \triangleq & \neg\ \boldsymbol{inside}^{\exists}\ \neg A & \\
\Diamond A & \triangleq & \boldsymbol{inside}^{\exists}(A|\mathbf{T}) & \text{(Somewhere)} \\
\Box A & \triangleq & \boldsymbol{inside}^{\forall}(A\|\mathbf{F}) & \text{(Everywhere)}
\end{array}
$$

The formula $\boldsymbol{group}\ A$ holds of any process P such that $P \equiv Q_1|\ldots|Q_k$ for some processes $Q_1,\ldots,Q_k$, where each Q_i satisfies A.

The formula $\boldsymbol{inside}\ ^{\exists}A$ holds of any process P such that $P \equiv (\boldsymbol{\nu} n_1)\cdots(\boldsymbol{\nu} n_k)Q$ for some fresh names $n_1,\ldots,n_k$ and process Q, where Q satisfies A.

LEMMA 6.6. *We have*

$$[\![\boldsymbol{\mu} X.(A \vee \mathsf{H}x.X)]\!]_v = \{(\boldsymbol{\nu} n_1)\cdots(\boldsymbol{\nu} n_k)Q \mid Q \in [\![A]\!]_v \text{ and } n_i \notin fn^v(A)\}^{\equiv}$$

Proof. Let $\Psi \triangleq \{(\boldsymbol{\nu} n_1)\cdots(\boldsymbol{\nu} n_k)Q \mid Q \in [\![A]\!]_v \text{ and } n_i \notin fn^v(A)\}^{\equiv}$. Let $M = fn^v(A) = fn^v(\boldsymbol{inside}\ ^{\exists}A)$; it is easy to check that $\Psi \in \mathbb{P}_M$. Ψ is closed under structural congruence by construction. Also, pick any process $P \in \Psi$, names $p, q \notin M$ and let $\tau = \{p \leftrightarrow q\}$. Then $P \equiv (\boldsymbol{\nu} n_1)\cdots(\boldsymbol{\nu} n_k)Q$ where $Q \in [\![A]\!]_v$. Hence $\tau(P) = (\boldsymbol{\nu}\tau(n_1))\cdots(\boldsymbol{\nu}\tau(n_k))\tau(Q)$, and we have $\tau(Q) \in [\![A]\!]_v$. We want to show that $\Psi = [\![\boldsymbol{\mu} X.(A \vee \mathsf{H}x.X)]\!]_v$.

We first prove $\Psi \subseteq [\![\boldsymbol{\mu} X.(A \vee \mathsf{H}x.X)]\!]_v$. To that end, we take any $\Phi \in \mathbb{P}_M$ such that $[\![A \vee \mathsf{H}x.X]\!]_{v[X\leftarrow\Phi]} \subseteq \Phi$ and verify $\Psi \subseteq \Phi$.

Pick $P \in \Psi$. Then $P \equiv (\boldsymbol{\nu} n_1)\cdots(\boldsymbol{\nu} n_k)Q$ and $Q \in [\![A]\!]_v$, for some $k \geq 0$. We show by induction on k that $P \in \Phi$.

If $k = 0$, we have $P \equiv Q \in [\![A]\!]_v$, and thus $P \in [\![A \vee \mathsf{H}x.X]\!]_{v[X\leftarrow\Phi]}$ by Remark 4.19, since X is not free in A. Hence $P \in \Phi$. If $k > 0$ we have $P \equiv (\boldsymbol{\nu} n_1)P'$ where $P' = (\boldsymbol{\nu} n_2)\cdots(\boldsymbol{\nu} n_k)Q \in \Psi$. By induction hypothesis, we have $P' \in \Phi$. Hence $P' \in [\![X]\!]_{v[X\leftarrow\Phi]}$. Thus $P \in [\![n_1 \circledR X]\!]_{v[X\leftarrow\Phi]}$. We have $n_1 \notin fn(P)$ and $n_1 \notin M = fn^{v[X\leftarrow\Phi]}(X) = fn^{v[X\leftarrow\Phi]}(\mathsf{И}x.x \circledR X)$. Then $P \in [\![\mathsf{H}x.X]\!]_{v[X\leftarrow\Phi]}$, and so $P \in [\![A \vee \mathsf{H}x.X]\!]_{v[X\leftarrow\Phi]}$. We conclude $P \in \Phi$, also in this case. Hence $\Psi \subseteq [\![\boldsymbol{\mu} X.(A \vee \mathsf{H}x.X)]\!]_v$.

Finally, to show $[\![\boldsymbol{\mu} X.(A \vee \mathsf{H}x.X)]\!]_v \subseteq \Psi$, it suffices to verify the inclusion $[\![A \vee \mathsf{H}x.X]\!]_{v[X\leftarrow\Psi]} \subseteq \Psi$. If $P \in [\![A \vee \mathsf{H}x.X]\!]_{v[X\leftarrow\Psi]}$ then either $P \in [\![A]\!]_v$ and thus $P \in \Psi$, or $P \equiv (\boldsymbol{\nu} n)P'$ where $P' \in \Psi$ and $n \notin fn^v(A) \cup fn(P)$. In the last case, is also immediate that $P \in \Psi$. We conclude $\Psi = [\![\boldsymbol{\mu} X.(A \vee \mathsf{H}x.X)]\!]_v$. ∎

A process P satisfies $\Diamond A$ if and only if somewhere inside P, possibly under some restricted names, there is a component satisfying A. In a similar way, a process satisfies $\boldsymbol{inside}^{\forall}\ A$ if and only if for all names $n_1,\ldots,n_k$ fresh with respect to A, and processes Q such that $P \equiv (\boldsymbol{\nu} n_1)\cdots(\boldsymbol{\nu} n_k)Q$, Q satisfies A.

A process P satisfies $\Box A$ if and only if all components of P, regardless of the local name context in which they are placed, satisfy A. For example, a process P satisfies the closed formula $\Box\neg\exists x.\exists y.x\langle y\rangle$ if and only if it contains no unguarded messages. Thus the formula $\Box\neg\exists x.\exists y.x\langle y\rangle \Rightarrow \boxdot\mathbf{F}$, asserting that every process without unguarded messages is bound to inactivity, is logically valid.

For an example of a property making use of several of the spatial and temporal modalities consider

$$Member(x) \triangleq \copyright x \wedge \forall y.(y\langle x\rangle | \mathbf{T} \Rightarrow y = x)$$
$$Group \triangleq \Box \mathsf{H} x.(\boldsymbol{group}\ Member(x))$$

A process satisfies *Group* if all of its future states will always be composed by a group of processes sharing a secret x, and such that each member of the group is only able to send x on x itself: in particular, this means that no group member can send the secret x on a public channel, since x is a restricted (hence not public) name in the whole process.

7. DISCUSSION

7.1. Structural Congruence and Intensionality

Structural congruence is usually considered just as a technical device, as a convenient intermediate step towards what really matters: a behavioral semantics of processes. In this views, structural congruence seems to have no interesting meaning in itself. Processes are seen more as pure behaviors than as computational systems possessing interesting structural properties. Then, process operators are seen as mappings behaviors to behaviors, completely forgetting the structure of processes and their individual identity. So, any interpretation of structural congruence may appear as strangely intensional.

However, spatial structure (i.e. something finer than behavioral equivalence) has always been discussed in the concurrency literature. For instance, at the beginning of his 1989 "Communication and Concurrency" book, Milner says: "Underlying both these notions is the assumption that each of the several parts of such a system has its own identity which persists through time [...] For if we wish to identify a particular event we have little choice but to identify the agents which participate in it; this amounts to determining *where*, i.e. at which part or parts, the event occurs". Early papers on CCS made a precise distinction between static (spatial) and dynamic (temporal) operators. The view of a concurrent system as a spatially distributed collection of interacting agents is also implicitly present in the actors model and in the Chemical Abstract Machine model; the latter was the inspiration for the structural-congruence-based presentation of π-calculus [26]. The spatial view of a system of processes appears again in the bi-graphical models Milner is considering recently [28].

Increasingly, the emphasis has been shifting, both in practice and in theory, from *concurrent* systems to *distributed* systems. We see a distributed system as a kind of active data structure, with localized parts and an internal architecture. Different internal architectures should not be equated, or everything might collapse to the single-machine case. So this is not a question of intensionality but rather a question of expressive (or observational) power. It is also important to realize that well-known extensional technical tools carry over naturally to the spatial case, and are not restricted to a particular view of the world. Such is the case for Hirshckoff, Lozes and Sangiorgi's space-time bisimulation [23].

Admittedly, we take a rather radical position here, basing our logic entirely on the intentional interpretation. It would be conceivable, for example, to have two kinds of parallel composition, only one of which having a spatial flavor. But, in any case, a structural congruence relation is needed to model some definite properties of space, in the same way as (labeled) reductions model properties of behavior. Both dimensions seem useful in the design of a calculus of concurrency. Both dimensions induce particular structures whose properties are important; we seek logics to specify both kinds of properties.

7.2. Structural Congruence: Technical Aspects

In retrospect, we may justify the introduction of structural congruence in process calculi, factoring it out from other process congruences like bisimulation, not just as a technical device towards something else, but as a way to model notions of spatial structure into the process model.

There is still, however, a hard question about what exactly should be the rules of structural congruence. There has always been a gray area between equivalences embedded in structural congruence and equivalences derived by extensional means. Even some properties of structural congruence that we adopt, such as the expansion law for replication $!P \equiv !P|P$, have a rather doubtful spatial interpretation. Still, over time, a rather standard set of rules has emerged. Some rules have been motivated by decidability concerns [18, 13, 12], which turn out to be central to any practical application of our logic.

We feel we have made some contribution to this issue, because some expected or convenient logical properties require certain properties of structural equivalence. This way, we helped wrestle some equations out of the gray area, namely the commutation of input with restriction.

Still, one may wonder: what is the equivalence induced by such an intensional logic, and how sensitive is it to the definition of structural congruence? (*C.f.* [22, 29].) For the closely related Ambient Logic, Sangiorgi has shown that logical equivalence characterizes (essentially) the structural congruence relation adopted there [33], and we expect similar results to hold in many spatial logics. We also expect this kind of results to facilitate decisions about the definition of structural congruence.

7.3. Logical Adjuncts

The guarantee operator was invented very early in the work on the Ambient Logic [10], in order to write security specifications for processes working in hostile contexts. Only afterwards we realized that guarantee was the natural adjunct of composition, and that the adjunction property greatly simplified the system of axioms that we were deriving from the satisfaction semantics.

This aspect of the early logic was so satisfactory that we actively went looking for other adjunctions (for ambient formation, and for revelation as in this paper). While these other adjuncts are not immediately intuitive, they too have strong logical properties that can be used effectively in algebraic manipulations (e.g. for hiding we get $\neg(A \oslash n) \dashv\vdash (\neg A) \oslash n$). Each adjunct has an interpretation in terms of a security property: for guarantee it is correctness in presence of a contiguous attacker, and for hiding it is correctness in presence of an attacker that cuts a channel to the outside world.

Adjuncts eliminate the need for other operators, and therefore ultimately simplify the logic. For example, via adjuncts we can define name equality. More interestingly, Sangiorgi characterized logical equivalence in the Ambient Logic [33] by cleverly defining logical formulas that capture certain process behaviors. One of those formulas corresponds to input processes, for which, like in the present logic, we did not have (and, as it turns out, we did not need) a primitive logical operator.

Rather surprisingly, the guarantee operator can be used to define an internal notion of validity and satisfiability. Initially this seemed a curiosity, but we were able to take advantage of it in a definition of recursive formulas that led to a considerable simplification in the proof of the main theorem (as discussed below).

Adjuncts fit naturally in the sequent structure investigated in Part II of this paper, where guarantee is the natural "implication" that corresponds to the composition "conjunction", leading to a "modus ponens" style rule. Moreover, in recent work [7] it is shown that the guarantee operator is tractable in a finite case, even if it appears to quantify over an infinite collection of contexts.

7.4. Second Order and Recursion

The second order quantifier was introduced at a late stage. Up to that point [4] we had greatest fix-point formulas as a primitive, with a syntactic monotonicity restriction. The proofs of Theorem 4.21 and the equivalent of Lemmas 6.1-6.2 were entangled by a mutual induction, because the fixpoint case of Theorem 4.21 had to rely on a monotonicity property. This made both the statement and the proof of Theorem 4.21, and the structure of the paper, more complex and obscure. The second-order quantifier does not introduce such difficulties, allowing for a self-contained proof of Theorem 4.21.

Via second order quantification and guarantee we are then able to define greatest and least fixpoints, and we can derive their rather subtle properties and rules. Moreover, we can define monotonicity assumptions internally, without having to rely on a meta-level restrictions on the syntax. This way, the treatment of recursion becomes completely formal and internal to the logic. We believe this is preferable even independently of the above argument about proof techniques.

Alternatively, we could have derived the recursion lemmas from general properties of Nominal Set Theory [19, 31] (in fact, Psets are nominal sets). However, for the sake of concreteness, we preferred to work them out in detail, in a standard set theoretic framework.

7.5. Alternative Semantics

In the definition of the semantics for our logic, the structure of Psets is put to use only in the clause for the second-order quantifier, and in the clause for the freshness quantifier (because $fn^v(A)$ uses $supp(-)$). But if we consider a restricted logic without second-order connectives (or fixpoints) and propositional variables, it turns out that $[\![A]\!]$ is a set of processes finitely supported by $fn(A)$, for very general reasons related to equivariance properties of satisfaction. Therefore, the framework of Psets and Gabbay-Pitts's theory of α-conversion based on transpositions [20, 19, 31] seems to be the natural setting to develop the semantics of our logic. Nevertheless, we may discuss alternative formalizations of notions of freshness.

For example, in [6, 3] the satisfaction relation for a related logic is indexed not only by a valuation v, as in our current semantics, but also by a finite set of names Σ (a signature): a closely related approach has been adopted by Dam in his development of modal logics for the π-calculus [16, 17]. In the former approaches, $\Sigma; P \models_v A$ is only defined when the free names of P and A are included in Σ (roughly, because free names in the image of v must also be taken into account), and we write $P \in [\![A]\!]_v^\Sigma$ when $\Sigma; P \models_v A$. The signature Σ is mainly useful to assist in the generation of fresh witnesses, namely, in the semantics in [2], Σ is extended element by element in the semantic clauses for the universal and the hidden name quantifier (a primitive). For instance, the semantics of the hidden name quantifier is defined thus:

$$\Sigma; P \models_v \mathsf{H}x.A \text{ if and only if } P \equiv (\boldsymbol{\nu}n)Q \text{ and } \Sigma, n; P \models_v A\{x\leftarrow n\}$$

where freshness is ensured by the proviso $n \notin \Sigma$.

Although in principle this approach should also work for our logic, we find that the technical development would not turn out to be simpler. The explicit indexing of the semantics over signatures causes satisfaction to be too sensitive to the name frame one chooses: *e.g.*, for no finite set of names Σ we get $[\![\mathbf{T}]\!]^\Sigma = \mathcal{P}$, while in our present semantics we conclude $[\![\mathbf{T}]\!] = \mathcal{P}$ immediately.

Another inconvenience of the explicit indexing is the need it brings, in our case, of obtaining technical principles of preservation of satisfaction under renaming, thining and weakening of the signature. Such principles are needed, for instance, to show the spatial extrusion properties of the hidden name quantifier (Lemma 5.5(2)), and the counterpart of the Gabbay-Pitts property, which are automatic in our more lax semantics. These are specific requirements of the semantics of our logic, related to the presence of the tensor and modalities for name restriction, that do not arise in more standard logics for the π-calculus [16, 17].

Some technical complications also arise in the definition of the semantics of fixpoints and related propositional variables, that would also carry over to the semantics of second-order quantification. In fact, suppose we define

$$[\![\boldsymbol{\mu}X.B]\!]_v^\Sigma \triangleq \bigcap\{\Psi \subseteq \mathcal{P}(\Sigma) \mid [\![B]\!]_{v[X\leftarrow\Psi]}^\Sigma \subseteq \Psi\}$$

where $\mathcal{P}(\Sigma)$ is the set of all processes with free names included in Σ. Now, consider the following formula (N.B. *NoRes* is defined in Section 5.3)

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

Intuitively, the formula A is satisfied by all processes one gets by embedding a restriction-free process under a certain number of restrictions, more precisely, by all processes of the form $(\boldsymbol{\nu}n_1)\cdots(\boldsymbol{\nu}n_k)Q$ with Q restriction-free. Now, consider $[\![A]\!]^\emptyset$, the set of all processes without free names that satisfy A. We expect *e.g.*, $(\boldsymbol{\nu}n)n\langle n\rangle \in [\![A]\!]^\emptyset$. However, by the candidate definition above we get $(\boldsymbol{\nu}n)n\langle n\rangle \notin [\![A]\!]^\emptyset$, because $[\![\mathit{NoRes}]\!]^\emptyset = \{\mathbf{0}\}$, the set $\Phi \triangleq \{\mathbf{0}\}$ verifies $[\![\mathit{NoRes} \vee \mathsf{H}x.X]\!]^\emptyset_{[X\leftarrow\Phi]} = \Phi$ and $(\boldsymbol{\nu}n)n\langle n\rangle \notin \Phi$. This shows that such semantic definition for the least fixpoint is not adequate. Then, to get a correct definition, compatible with the $[\![-]\!]_v^\Sigma$ semantics, instead of ranging Ψ over all subsets of $\mathcal{P}(\Sigma)$ we must range Ψ over a larger collection of subsets of $\mathcal{P}$. Such a collection turns out to be essentially the collection of all Psets supported by Σ (in the sense of the present semantics). In fact, by working inside the domain of Psets $\mathbb{P}_-$, we can define the semantics of least-fixpoint in quite standard terms:

$$[\![\boldsymbol{\mu}X.B]\!]_v \triangleq \bigcap\{\Psi \in \mathbb{P}_- \mid [\![B]\!]_{v[X\leftarrow\Psi]} \subseteq \Psi\}$$

as shown in Section 6 (and indirectly in [4]). Another possibility, adopted in [16, 17], it to forbid all occurrences of free names in fixpoint formulas, requiring instead the explicit parametrisation of propositional variables on the set of "free" names their denotations can use. Such technique seems difficult to adopt in our case, because it requires formulas and propositional variables to be assigned fixed arities. This makes it hard to obtain, *e.g.*, the needed signature weakening principle, or a general substitutivity principle (Lemma 4.29 and Proposition 4.30).

Another aspect that deserves some discussion relates to our use of $fn^v(A)$ in the semantic definition of freshness. Our definition of the semantics of freshness makes use of the auxiliary concept $fn^v(A)$, the set of free names in a formula A under a valuation v (Definition 4.16). The set of names $fn^v(A)$ gives a bound to the support of the property set denoted by A under v, and is fully justified by Theorem 4.21 (1). However, we may question why in the definition of $fn^v(A)$ both syntactic (the free names of A) and semantic information (the support of the Psets in v) come into play. To remove this explicit reference to the free names of A, we might consider an alternative presentation of the logic, where pure names are banned from the syntax of formulas, and semantics is defined with relation to extended valuations interpreting not just propositional variables, as in our preferred semantics, but also name variables. This alternative $[\![-]\!]^*_-$ semantics avoids any mention of the free names of the formula A and is equivalent to the one given: we can verify that if $v_\mathcal{V}$ and $v_\mathcal{X}$ are the restrictions of a valuation v to $\mathcal{V}$ and $\mathcal{X}$ respectively, then we have $[\![A]\!]^*_v = [\![v_\mathcal{V}(A)]\!]_{v_\mathcal{X}}$ for all formulas A and valuations v. The semantics of the freshness quantifier could then be given by

$$[\![\reflectbox{N}x.A]\!]^*_v \triangleq \bigcup_{n \notin \mathfrak{I}(v)} ([\![A]\!]^*_{v[x\leftarrow n]} \setminus \{P \mid n \in fn(P)\})$$

where $\mathfrak{I}(v)$ is the image (or range) of a valuation v, defined by $\mathfrak{I}(v) \triangleq \{v(x) \mid x \in \mathcal{V} \cap \mathfrak{D}(v)\} \cup \bigcup\{supp(v(X)) \mid X \in \mathcal{X} \cap \mathfrak{D}(v)\}$. Note that in this definition names are picked fresh with respect to the "global" set $\mathfrak{I}(v)$, instead of the "local" set $fn^v(A)$: this fact has some unfortunate consequences. Namely, the counterpart of Remark 4.19, stating invariance of the semantics with respect to weakenings of valuations, would then require an extended amount of properties (*e.g.*, the Gabbay-Pitts property, Proposition 5.1), to be established at an early stage of the development, namely as additional clauses to the statement of the main theorem (Theorem 4.21).

8. CONCLUSIONS

We have investigated the satisfaction relation for a logic of concurrent processes that includes spatial operators, freshness quantifiers, and recursive formulas. In particular, we have shown how coinductively defined logical properties including freshness quantification can be given a semantics in terms of maximal fixpoints in a lattice of finitely supported sets of processes.

The logical rules arising from such a satisfaction relation are investigated in Part II of this paper [5]. Several interesting logical properties have already been discussed with respect to this model.

Some properties of the logic are very sensitive to the formulation of structural congruence (in fact, Sangiorgi has shown that the process equivalence induced by a similar logic is essentially structural congruence [33]). There is always a tension between embedding an expected process equivalence into the definition of structural congruence, or leaving it as a derived behavioral equivalence. Some desired logical properties have provided new insights into this delicate balance.

The general structure of our definitions can be easily adapted to various process calculi, and is also largely independent from the details of the operational semantics. Although the semantics considered here is based on unlabeled transition systems, it could be extended in a natural way to labeled transition systems, motivating then the introduction of Hennessy-Milner-like "labeled" modalities into the spatial logic. Nevertheless, some basic features of what a formula denotes, namely closure under structural congruence and finite support, should be expected to hold in all variations.

In conclusion, the general term of "spatial logic" has a fairly well defined, though informal, meaning. A spatial logic always offers a degree of intensionality, in order to talk about fine details of process structure. This is what is required if we want to meaningfully describe the distribution of processes and the use of resources over a network.

Acknowledgments Andy Gordon contributed to the early stages of this paper. Remarks by Andy Pitts lead to several improvements in the semantics of our logic. We also acknowledge many useful comments by Mads Dam. Thanks also to Giorgio Ghelli, Philippa Gardner and Luís Monteiro for related discussions. The anonymous referees also contributed useful comments. Caires acknowledges the support of Microsoft Research for a visit to the Cambridge Lab during the time we worked on this paper, and Profundis FET IST-2001-33100.

REFERENCES

[1] G. Boudol. Asynchrony and the π-calculus (note). Rapport de Recherche 1702, INRIA Sofia-Antipolis, May 1992.

[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. A Specification Logic for Mobility. Technical Report 4, Dept. de Informática, FCT, Universidade Nova de Lisboa, 2000.

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

[5] 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.

[6] L. Caires and L. Monteiro. Verifiable and Executable Specifications of Concurrent Objects in $\mathcal{L}_\pi$. In C. Hankin, editor, *Programming Languages and Systems: Proceedings of the 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. In *ACM Workshop on Types in Language Design and Implementation*, pages 62–73, New Orleans, USA, 2003. ACM Press.

[8] 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.

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

[10] 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.

[11] 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.

[12] W. Charatonik and J.-M. Talbot. The decidability of model checking mobile ambients. In *Proceedings of the 15th Annual Conference of the European Association for Computer Science Logic*, Lecture Notes in Computer Science. Springer-Verlag, 2001. To appear.

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

[14] M. Dam. Relevance Logic and Concurrent Composition. In *Proceedings, Third Annual Symposium on Logic in Computer Science*, pages 178–185, Edinburgh, Scotland, 5–8 July 1988. IEEE Computer Society.

[15] M. Dam. *Relevance Logic and Concurrent Composition*. PhD thesis, 1989.

[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] P. Gardner. From Process Calculi to Process Frameworks. In Catuscia Palamidessi, editor, *CONCUR 2000: Concurrency Theory (11th International Conference, University Park, PA, USA)*, volume 1877 of *Lecture Notes in Computer Science*, pages 69–88. Springer, August 2000.

[22] M. Hennessy and R. Milner. Algebraic laws for Nondeterminism and Concurrency. *JACM*, 32(1):137–161, 1985.

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

[24] K. Honda and M. Tokoro. On Asynchronous Communication Semantics. In M. Tokoro, O. Nierstrasz, and P. Wegner, editors, *Object-Based Concurrent Computing 1991*, number 612 in Lecture Notes in Computer Science, pages 21–51. Springer-Verlag, 1992.

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

[26] R. Milner. Functions as processes. *Math. Struc. in Computer Sciences*, 2(2):119–141, 1992.

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

[28] R. Milner. Bigraphical reactive systems: basic theory. Technical Report 523, Cambridge Computer Laboratory, 2002.

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

[30] P. O'Hearn and D. Pym. The Logic of Bunched Implications. *The Bulletin of Symbolic Logic*, 5(2):215–243, 1999.

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

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

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

Appendix (Proofs)

Lemma 4.3

For all finite $N \subseteq \Lambda$,

1. If $N \subseteq N'$ then $\mathbb{P}_N \subseteq \mathbb{P}_{N'}$.
2. (Bottom and Top) $\emptyset \in \mathbb{P}_N$ and $\mathcal{P} \in \mathbb{P}_N$.
3. (Meet and Join) If $S \subseteq \mathbb{P}_N$ then $\bigcap S \in \mathbb{P}_N$ and $\bigcup S \in \mathbb{P}_N$.

4. (Inverse) If $\Psi \in \mathbb{P}_N$ then $\overline{\Psi} = \mathcal{P} \setminus \Psi \in \mathbb{P}_N$.

Proof.

1. Pick $\Psi \in \mathbb{P}_N$. Now, Ψ is closed under $\equiv$; we must just verify that Ψ is supported by N'. Pick $m, n \notin N'$. Since $N \subseteq N'$, we have $m, n \notin N$. Therefore, since $\Psi \in \mathbb{P}_N$, for all $P \in \Psi$, we have $P\{m{\leftrightarrow}n\} \in \Psi$. Thus $\Psi \in \mathbb{P}_{N'}$.

2. Immediate.

3. Let $S \subseteq \mathbb{P}_N$.

   (a) Pick $P \in \bigcap S$. Then, for all $\Psi \in S$, we have $P \in \Psi$ and $\Psi \in \mathbb{P}_N$. Now, if $Q \equiv P$, then $Q \in \Psi$, for all $\Psi \in S$. Thus $Q \in \bigcap S$. Now let $m, n \notin N$. We have $P\{m{\leftrightarrow}n\} \in \Psi$, for all $\Psi \in S$. Hence $P\{m{\leftrightarrow}n\} \in \bigcap S$. We conclude $\bigcap S \in \mathbb{P}_N$.

   (b) Pick $P \in \bigcup S$. Then, there is $\Psi \in S$ such that $P \in \Psi$ and $\Psi \in \mathbb{P}_N$. Thus, if $Q \equiv P$, then $Q \in \Psi \subseteq \bigcup S$. Now let $m, n \notin N$. Then, $P\{m{\leftrightarrow}n\} \in \Psi \subseteq \bigcup S$. Hence $\bigcup S \in \mathbb{P}_N$.

4. Assume $\Psi \in \mathbb{P}_N$. Pick $P \in \overline{\Psi}$. Let $Q \equiv P$ and suppose $Q \notin \overline{\Psi}$. Then $Q \in \Psi$ and $P \in \Psi$, contradiction. Hence $Q \in \overline{\Psi}$. Likewise, pick $m, n \notin N$ and suppose $P\{m{\leftrightarrow}n\} \notin \overline{\Psi}$. Then $P\{m{\leftrightarrow}n\} \in \Psi$, and this implies $P \in \Psi$, a contradiction. Hence we must have $P\{m{\leftrightarrow}n\} \in \overline{\Psi}$. We conclude $\overline{\Psi} \in \mathbb{P}_N$.

∎

Lemma 4.11

1. For any process P and Pset Ψ, $P \in \tau(\Psi)$ if and only if $\tau(P) \in \Psi$.

2. $\Psi \in \mathbb{P}_N$ if and only if $\tau(\Psi) \in \mathbb{P}_{\tau(N)}$.

3. If $m, n \notin N$ and $\Psi \in \mathbb{P}_N$ then $\{m{\leftrightarrow}n\}(\Psi) = \Psi$.

Proof.

1. Let $P \in \tau(\Psi)$. Then there is $Q \in \Psi$ with $\tau(Q) = P$. Thus $Q \equiv \tau(P)$. Since Ψ is closed under $\equiv$, $\tau(P) \in \Psi$.

2. Assume $\Psi \in \mathbb{P}_N$. Pick $P \in \tau(\Psi)$. We have $P = \tau(Q)$ for some $Q \in \Psi$.

   Pick $R \equiv P$. Hence we have $\tau(R) \equiv Q$, and thus $\tau(R) \in \Psi$, by closure under $\equiv$ of Ψ. Therefore $R \in \tau(\Psi)$, and we conclude $\tau(\Psi)$ is closed under $\equiv$.

   Now, pick $m, n \notin \tau(N)$, and let $\tau' = \{m{\leftrightarrow}n\}$. We have $\tau(m), \tau(n) \notin N$. Thus $\{\tau(m){\leftrightarrow}\tau(n)\}(Q) \in \Psi$. This implies $\tau(\{\tau(m){\leftrightarrow}\tau(n)\}(Q)) \in \tau(\Psi)$. But $\tau(\{\tau(m){\leftrightarrow}\tau(n)\}(Q)) \equiv \{m{\leftrightarrow}n\}(\tau(Q)) \equiv \tau'(P)$, by Proposition 2.4(2). Hence $\tau'(P) \in \tau(\Psi)$. We conclude $\tau(\Psi) \in \mathbb{P}_{\tau(N)}$.

3. Assume $P \in \Psi \in \mathbb{P}_N$. Then $\{m{\leftrightarrow}n\}(P) \in \Psi$, and thus $P \in \{m{\leftrightarrow}m\}(\Psi)$. On the other hand, if $P \in \{m{\leftrightarrow}m\}(\Psi)$ then there is $Q \in \Psi$ such that $P = \{m{\leftrightarrow}m\}(Q)$. But $\{m{\leftrightarrow}m\}(Q) \in \Psi$, so $P \in \Psi$.

∎

Proposition 4.10

For all finite $N \subseteq \Lambda$, $(\mathbb{P}_N, \subseteq, \bigcup, \otimes, \mathbf{1})$ is a commutative quantale.

Proof. We first verify that $(\mathbb{P}_N, \otimes, \mathbf{1})$ is a commutative monoid.

First, note that $\mathbf{1}$ is closed under $\equiv$ by definition. Moreover, for any $P \in \mathbf{1}$ and transposition τ, we have $\tau(P) \equiv \tau(\mathbf{0}) \equiv \mathbf{0}$. Hence, $\tau(P) \in \mathbf{1}$, for any transposition τ. We conclude that $\mathbf{1} \in \mathbb{P}_N$.

To verify that $\otimes$ is a binary operation on $\mathbb{P}_N$, pick any Ψ and Φ in $\mathbb{P}_N$, and some $P \in \Psi \otimes \Phi$. If $P' \equiv P$ then $P' \equiv Q|R$ where $Q \in \Psi$ and $R \in \Psi$. Hence $P' \in \Psi \otimes \Phi$. Moreover, if $m, n \notin N$ and $\tau = \{m \leftrightarrow n\}$, then $\tau(P) \equiv \tau(Q|R) \equiv \tau(Q)|\tau(R)$. Since $\tau(Q) \in \Psi$ and $\tau(R) \in \Phi$, we conclude $\tau(P) \in \Psi \otimes \Phi$. Now, from simple properties of structural congruence it follows that $\otimes$ is commutative, associative, and has unit $\mathbf{1}$.

Since $(\mathbb{P}_N, \subseteq, \cup, \cap)$ is a complete lattice, and thus closed under arbitrary joins, it remains to verify that $\otimes$ distributes over arbitrary joins, that is $\Psi \otimes \bigcup S = \bigcup\{\Psi \otimes \Phi \mid \Phi \in S\}$, for any $S \subseteq \mathbb{P}_N$. But this is an immediate consequence of the definition of $\otimes$. ∎

Proposition 4.13

Let $\Phi \in \mathbb{P}_N$ then

1. There is a least set of names $supp(\Phi)$ such that $\Phi \in \mathbb{P}_{supp(\Phi)}$.
2. For any transposition τ, if $supp(\Phi) = M$ then $supp(\tau(\Phi)) = \tau(M)$.

Proof. (1) See [19] Proposition 3.5.

(2) Let τ be a transposition, Φ be a Pset and assume $supp(\Phi) = M$. This means that $\Phi \in \mathbb{P}_M$ and for all N such that $\Phi \in \mathbb{P}_N$ we have $M \subseteq N$. To verify that $supp(\tau(\Phi)) = \tau(M)$ we need to show $\tau(\Phi) \in \mathbb{P}_{\tau(M)}$ (which holds by Lemma 4.11(2)) and that for all finite M' such $\tau(\Phi) \in \mathbb{P}_{M'}$ we have $\tau(M) \subseteq M'$. So, take a finite set of names M' and assume $\tau(\Phi) \in \mathbb{P}_{M'}$. Thus $\Phi \in \mathbb{P}_{\tau(M')}$, by Lemma 4.11(2). By assumption, we conclude $M \subseteq \tau(M')$. But then $\tau(M) \subseteq M'$ (for τ is an bijective mapping $\wp_{fin}(\Lambda) \to \wp_{fin}(\Lambda)$), and we are done. ∎

Theorem 4.21

For any formula A and valuation v

1. $[\![A]\!]_v \in \mathbb{P}_{fn^v(A)}$.
2. For all transpositions τ, $\tau([\![A]\!]_v) = [\![\tau(A)]\!]_{\tau(v)}$.

Proof. The proof of (1—2) proceeds by mutual induction on the size of the formula A. Instead of the equality in (2) we prove that $\tau([\![A]\!]_v) \subseteq [\![\tau(A)]\!]_{\tau(v)}$. From this fact, the equality immediately follows. Indeed, pick $P \in [\![\tau(A)]\!]_{\tau(v)}$, we have $\tau(P) \in \tau([\![\tau(A)]\!]_{\tau(v)}) \subseteq [\![\tau\tau(A)]\!]_{\tau\tau(v)} = [\![A]\!]_v$, hence $P \in \tau([\![A]\!]_v)$.

- Case $A = \mathbf{F}$

  (1) Clearly $[\![\mathbf{F}]\!]_v = \emptyset \in \mathcal{P}_{fn^v(\mathbf{F})}$.

  (2) Trivial, since $\tau([\![\mathbf{F}]\!]_v) = \tau(\emptyset) = \emptyset$.

- Case $A = B \wedge C$.

  (1) Then $[\![B \wedge C]\!]_v = [\![B]\!]_v \cap [\![C]\!]_v$.

  By induction hypothesis (1) and Lemma 4.3(1) both $[\![B]\!]_v$ and $[\![C]\!]_v$ belong to $\mathcal{P}_{fn^v(A)}$. We conclude by Lemma 4.3(3).

  (2) Let τ be a transposition. If $P \in \tau([\![A \wedge B]\!]_v)$ then there is P' such that $P = \tau(P')$ with $P' \in [\![A]\!]_v$ and $P' \in [\![B]\!]_v$. Then $P \in \tau([\![A]\!]_v)$ and $P \in \tau([\![B]\!]_v)$. By induction hypothesis (2), $P \in [\![\tau(A)]\!]_{\tau(v)}$ and $P \in [\![\tau(B)]\!]_{\tau(v)}$. Hence, $P \in [\![\tau(A \wedge B)]\!]_{\tau(v)}$.

- Case $A = B \Rightarrow C$.

  (1) Pick $P \in [\![B \Rightarrow C]\!]_v$. Therefore, if $P \in [\![B]\!]_v$ then $P \in [\![C]\!]_v$. By induction hypothesis (1), $[\![B]\!]_v \in \mathbb{P}_{fn^v(B)}$ and $[\![C]\!]_v \in \mathbb{P}_{fn^v(C)}$.

  Pick any $Q \equiv P$. Assume $Q \in [\![B]\!]_v$. Then $P \in [\![B]\!]_v$ and thus $P \in [\![C]\!]_v$. Hence $Q \in [\![C]\!]_v$, and we conclude $Q \in [\![B \Rightarrow C]\!]_v$.

  Now pick $m, n \notin fn^v(B \Rightarrow C)$ and let $\tau = \{m \leftrightarrow n\}$. We have, $m, n \notin fn^v(B)$ and $m, n \notin fn^v(C)$. Assume $\tau(P) \in [\![B]\!]_v$. Thus $P \in [\![B]\!]_v$ by induction hypothesis (1). Then $P \in [\![C]\!]_v$ and by induction hypothesis (1) again $\tau(P) \in [\![C]\!]_v$. Hence $\tau(P) \in [\![B \Rightarrow C]\!]_v$.

  (2) Let τ be a transposition. Pick $P \in \tau([\![B \Rightarrow C]\!]_v)$. Then, $\tau(P) \in [\![B]\!]_v$ implies $\tau(P) \in [\![C]\!]_v$. Assume $P \in [\![\tau(B)]\!]_{\tau(v)}$. By induction hypothesis (2), we conclude $\tau(P) \in [\![B]\!]_v$. Then, $\tau(P) \in [\![C]\!]_v$. By induction hypothesis (2) again, $P \in [\![\tau(C)]\!]_{\tau(v)}$. Therefore $P \in [\![\tau(B \Rightarrow C)]\!]_{\tau(v)}$.

- Case $A = \mathbf{0}$.

  (1) $[\![\mathbf{0}]\!]_v = \mathbf{1} \in \mathbb{P}_{fn^v(\mathbf{0})} = \mathbb{P}_\emptyset$ by Proposition 4.10.

  (2) Let τ be a transposition. Pick $P \in \tau([\![\mathbf{0}]\!]_v)$. Then $\tau(P) \equiv \mathbf{0}$ and $P \equiv \mathbf{0}$. So $P \in [\![\tau(\mathbf{0})]\!]_{\tau(v)} = \mathbf{1}$.

- Case $A = B|C$.

  (1) By induction hypothesis (1) and Lemma 4.3(1) we have $[\![B]\!]_v \in \mathbb{P}_{fn^v(B)} \subseteq \mathbb{P}_{fn^v(B|C)}$. Likewise we conclude $[\![C]\!]_v \in \mathbb{P}_{fn^v(B|C)}$.

  By Proposition 4.10, $[\![B|C]\!]_v = [\![B]\!]_v \otimes [\![C]\!]_v \in \mathbb{P}_{fn^v(B|C)}$.

  (2) Let τ be a transposition. Pick $P \in \tau([\![B|C]\!]_v)$. Hence $\tau(P) \equiv Q'|Q''$ with $Q' \in [\![B]\!]_v$ and $Q'' \in [\![C]\!]_v$, for some Q' and Q''. Then $\tau(Q') \in \tau([\![B]\!]_v)$, and $\tau(Q') \in [\![\tau(B)]\!]_{\tau(v)}$, by induction hypothesis (2). Likewise, $\tau(Q'') \in [\![\tau(C)]\!]_{\tau(v)}$. Hence $\tau(Q'|Q'') \in [\![\tau(B|C)]\!]_{\tau(v)}$. To conclude, just note that $P \equiv \tau(Q'|Q'')$.

- Case $A = B \triangleright C$.

  (1) Pick $P \in [\![B \triangleright C]\!]_v$. Then, for all $Q \in [\![B]\!]_v$, we have $Q|P \in [\![C]\!]_v$. Pick any $P' \equiv P$, and any $Q' \in [\![B]\!]_v$. We have $Q'|P' \equiv Q'|P$. By assumption, $Q'|P \in [\![C]\!]_v$, and, by induction hypothesis (1), $Q'|P' \in [\![C]\!]_v$. Hence $P' \in [\![B \triangleright C]\!]_v$.

  Pick $m, n \notin fn^v(B \triangleright C)$ and let $\tau = \{m \leftrightarrow n\}$. Then $m, n \notin fn^v(B)$ and $m, n \notin fn^v(C)$. Now, pick any Q' such that $Q' \in [\![B]\!]_v$. By induction hypothesis (1), $\tau(Q') \in [\![B]\!]_v$. Therefore $\tau(Q')|P \in [\![C]\!]_v$. By induction hypothesis (1) and Proposition 2.12(3), $\tau(\tau(Q')|P) \equiv Q'|\tau(P) \in [\![C]\!]_v$. Hence $\tau(P) \in [\![B \triangleright C]\!]_v$.

(2) Let τ be a transposition. Pick $P \in \tau(\llbracket B \triangleright C \rrbracket_v)$. Then there is $P' \equiv \tau(P)$ such that for all $Q \in \llbracket B \rrbracket_v$, we have $Q|P' \in \llbracket C \rrbracket_v$. Pick any $Q' \in \llbracket \tau(B) \rrbracket_{\tau(v)}$. Then $\tau(Q') \in \tau(\llbracket \tau(B) \rrbracket_{\tau(v)})$. By induction hypothesis (2), $\tau(Q') \in \llbracket B \rrbracket_v$. Then $\tau(Q')|\tau(P) \in \llbracket C \rrbracket_v$. So, $Q'|P \in \tau(\llbracket C \rrbracket_v)$. By induction hypothesis (2), $Q'|P \in \llbracket \tau(C) \rrbracket_{\tau(v)}$. Hence $P \in \llbracket \tau(B \triangleright C) \rrbracket_{\tau(v)}$.

- Case $A = q \circledR B$.

  (1) Pick any $P \in \llbracket q \circledR B \rrbracket_v$. Then $P \equiv (\boldsymbol{\nu} q)Q$ and $Q \in \llbracket B \rrbracket_v$. Pick any $P' \equiv P$, we have $P' \equiv (\boldsymbol{\nu} q)Q$, and thus $P' \in \llbracket q \circledR B \rrbracket_v$. Pick $m, n \notin fn^v(q \circledR B)$ and let $\tau = \{m \leftrightarrow n\}$. Then $m, n \notin fn^v(B)$, $m \neq q$ and $n \neq q$. Hence $\tau(P) \equiv (\boldsymbol{\nu} q)\tau(Q)$, by Proposition 2.12(3). By induction hypothesis (1), we have $\tau(Q) \in \llbracket B \rrbracket_v$. So $\tau(P) \in \llbracket q \circledR B \rrbracket_v$.

  (2) Let τ be a transposition. Pick $P \in \tau(\llbracket q \circledR B \rrbracket_v)$. Then there is $P' = \tau(P)$ such that $P' \equiv (\boldsymbol{\nu} q)Q$ and $Q \in \llbracket B \rrbracket_v$. We have $\tau(P') \equiv (\boldsymbol{\nu} \tau(q))\tau(Q)$, by Lemma 2.8(1). By induction hypothesis (2), we have $\tau(Q) \in \llbracket \tau(B) \rrbracket_{\tau(v)}$. Hence $P = \tau(P') \in \llbracket \tau(q) \circledR \tau(B) \rrbracket_{\tau(v)} = \llbracket \tau(q \circledR B) \rrbracket_{\tau(v)}$.

- Case $A = B \oslash q$.

  (1) Pick any $P \in \llbracket B \oslash q \rrbracket_v$. Then $(\boldsymbol{\nu} q)P \in \llbracket B \rrbracket_v$. Pick any $P' \equiv P$, we have $(\boldsymbol{\nu} q)P' \equiv (\boldsymbol{\nu} q)P$, and thus, by induction hypothesis (1), $(\boldsymbol{\nu} q)P' \in \llbracket B \rrbracket_v$. Therefore, $P' \in \llbracket B \oslash q \rrbracket_v$. Pick $m, n \notin fn^v(B \oslash q)$ and let $\tau = \{m \leftrightarrow n\}$. Then $m, n \notin fn^v(B)$, $m \neq q$ and $n \neq q$. By induction hypothesis (1), we have $\tau((\boldsymbol{\nu} q)P) \in \llbracket B \rrbracket_v$. Since $\tau((\boldsymbol{\nu} q)P) = (\boldsymbol{\nu} q)\tau(P)$, we have $\tau(P) \in \llbracket B \oslash q \rrbracket_v$.

  (2) Let τ be a transposition. Pick $P \in \tau(\llbracket B \oslash q \rrbracket_v)$. Then there is $P' = \tau(P)$ such that $(\boldsymbol{\nu} q)P' \in \llbracket B \rrbracket_v$. Then $\tau(P) \in \llbracket B \oslash q \rrbracket_v$, that is, $(\boldsymbol{\nu} q)\tau(P) \in \llbracket B \rrbracket_v$. Then $\tau((\boldsymbol{\nu} q)\tau(P) = (\boldsymbol{\nu} \tau(q))P \in \tau(\llbracket B \rrbracket_v)$. By induction hypothesis (2), $(\boldsymbol{\nu} \tau(q))P \in \llbracket \tau(B) \rrbracket_{\tau(v)}$. Hence $P \in \llbracket \tau(B) \oslash \tau(q) \rrbracket_{\tau(v)} = \llbracket \tau(B \oslash q) \rrbracket_{\tau(v)}$.

- Case $A = p\langle q \rangle$.

  (1) We have $\llbracket p\langle q \rangle \rrbracket_v = \{P \mid P \equiv p\langle q \rangle\}$, which is closed under $\equiv$ by definition. Additionally, for all $m, n \notin \{p, q\}$, we have that $\{n \leftrightarrow m\}P \equiv p\langle q \rangle$ for all $P \equiv p\langle q \rangle$.

  (2) Let $\tau = \{m \leftrightarrow n\}$. If $P \in \tau(\llbracket p\langle q \rangle \rrbracket_v)$ then $\tau(P) \equiv p\langle q \rangle$, and $P \equiv \tau(p\langle q \rangle)$. It is clear that $P \in \llbracket \tau(p\langle q \rangle) \rrbracket_{\tau(v)}$.

- Case $A = Ⅎx.B$.

  (1) Pick $P \in \llbracket Ⅎx.B \rrbracket_v$. Then, there is q such that $q \notin fn^v(B) \cup fn(P)$ and $P \in \llbracket B\{x \leftarrow q\} \rrbracket_v$. Pick $Q \equiv P$, by induction hypothesis (1) also $Q \in \llbracket B\{x \leftarrow q\} \rrbracket_v$, and $q \notin fn(Q)$ by Proposition 2.12(1). Thus $Q \in \llbracket Ⅎx.B \rrbracket_v$.

  Now, pick $\tau = \{m \leftrightarrow n\}$ with $m, n \notin fn^v(Ⅎx.B) = fn^v(B)$. Pick any $p \notin fn^v(B) \cup \{m, n, q\} \cup fn(P) \cup fn(\tau(P))$ and let $\tau' = \{q \leftrightarrow p\}$. By induction hypothesis (2), we have $\tau'(P) \in \llbracket \tau'(B\{x \leftarrow q\}) \rrbracket_{\tau'(v)}$, that is $P \in \llbracket B\{x \leftarrow p\} \rrbracket_v$, by Lemma 4.11(3). By induction hypothesis (1), we conclude $\tau(P) \in \llbracket B\{x \leftarrow p\} \rrbracket_v$. Thus, $\tau(P) \in \llbracket Ⅎx.B \rrbracket_v$, since $p \notin fn^v(B) \cup fn(\tau(P))$.

  (2) Let $\tau = \{m \leftrightarrow n\}$. If $P \in \tau(\llbracket Ⅎx.B \rrbracket_v)$ then $\tau(P) \in \llbracket Ⅎx.B \rrbracket_v$ and thus there is some $q \notin fn^v(B) \cup fn(\tau(P))$ such that $\tau(P) \in \llbracket B\{x \leftarrow q\} \rrbracket_v$.

  Now, pick $p \notin \{m, n\} \cup fn(\tau(P)) \cup fn(P) \cup fn^v(B) \cup fn^{\tau(v)}(\tau(B))$ and let $\tau' = \{q \leftrightarrow p\}$. $\tau'(\tau(P)) = \tau(P) \in \llbracket \tau'(B\{x \leftarrow q\}) \rrbracket_{\tau'(v)} = \llbracket B\{x \leftarrow p\} \rrbracket_{\tau'(v)}$.

By Remark 4.19 we have $[\![B\{x\leftarrow p\}]\!]_{\tau'(v)} = [\![B\{x\leftarrow p\}]\!]_{\tau'(w)}$, where w is the restriction of v to $fpv(B)$.

By Lemma 4.20(1) we have $[\![B\{x\leftarrow p\}]\!]_{\tau'(w)} = [\![B\{x\leftarrow p\}]\!]_w$, since $q, p \notin fn(w)$. So, $\tau(P) \in [\![B\{x\leftarrow p\}]\!]_w$.

By Remark 4.19, this implies $\tau(P) \in [\![B\{x\leftarrow p\}]\!]_v$.

But then $P \in [\![\tau(B)\{x\leftarrow p\}]\!]_{\tau(v)}$, by induction hypothesis (2). We conclude $P \in [\![\tau(\text{И}x.B)]\!]_{\tau(v)}$, since $p \notin fn^{\tau(v)}(\tau(B)) \cup fn(P)$.

- Case $A = \forall x.B$.

  (1) Pick any $P \in [\![\forall x.B]\!]_v$. Then, for all $q \in \Lambda$, $P \in [\![B\{x\leftarrow q\}]\!]_v$.

  Pick any $Q \equiv P$. For all $q \in \Lambda$, by induction hypothesis (1), $Q \in [\![B\{x\leftarrow q\}]\!]_v$. Therefore, $Q \in [\![\forall x.B]\!]_v$.

  Pick $m, n \notin fn^v(\forall x.B) = fn^v(B)$ and let $\tau = \{m\leftrightarrow n\}$. We need to show that for all $q \in \Lambda$, $\tau(P) \in [\![B\{x\leftarrow q\}]\!]_v$.

  For all $q \notin \{m, n\}$, by induction hypothesis (1) we have $\tau(P) \in [\![B\{x\leftarrow q\}]\!]_v$.

  For $q = m$, we have $P \in [\![B\{x\leftarrow m\}]\!]_v$. Then $\tau(P) \in [\![B\{x\leftarrow n\}]\!]_{\tau(v)}$, by induction hypothesis (2).

  Let w be the restriction of v to $fpv(B)$. Note that $m, n \notin fn(w)$, for if (say) $n \in fn(w)$, then $n \in supp(w(X))$ for some $X \in fpv(B)$, and we would have $n \in fn^w(B) = fn^v(B)$, a contradiction.

  By Remark 4.19, $\tau(P) \in [\![B\{x\leftarrow n\}]\!]_{\tau(w)}$.

  Since we have $m, n \notin fn(w)$, by Lemma 4.20(1) we conclude $[\![B\{x\leftarrow n\}]\!]_{\tau(w)} = [\![B\{x\leftarrow n\}]\!]_w$. By Remark 4.19, $[\![B\{x\leftarrow n\}]\!]_w = [\![B\{x\leftarrow n\}]\!]_v$ and then $\tau(P) \in [\![B\{x\leftarrow n\}]\!]_v$.

  For $q = n$, we conclude $\tau(P) \in [\![B\{x\leftarrow m\}]\!]_v$ in a similar way.

  Then $\tau(P) \in [\![B\{x\leftarrow q\}]\!]_v$, for all $q \in \Lambda$; this implies $\tau(P) \in [\![\forall x.B]\!]_v$.

  (2) Let τ be a transposition. Pick $P \in \tau([\![\forall x.B]\!]_v)$. Then $\tau(P) \in [\![B\{x\leftarrow q\}]\!]_v$, for all $q \in \Lambda$.

  By induction hypothesis (2), for all $q \in \Lambda$, $P \in [\![\tau(B)\{x\leftarrow\tau(q)\}]\!]_{\tau(v)}$. Then $P \in [\![\tau(B)\{x\leftarrow q\}]\!]_{\tau(v)}$, for all $q \in \Lambda$, since τ is a bijection $\Lambda \to \Lambda$. Therefore, $P \in [\![\tau(\forall x.B)]\!]_{\tau(v)}$.

- Case $A = \Diamond B$.

  (1) If $P \in [\![\Diamond B]\!]_v$ then there is R such that $P \to R$ and $R \in [\![B]\!]_v$. If $Q \equiv P$ then also $Q \to R$, so $Q \in [\![\Diamond B]\!]_v$. Now, pick $m, n \notin fn^v(\Diamond B) = fn^v(B)$, and let $\tau = \{m\leftrightarrow n\}$. By Proposition 2.15(2), $\tau(P) \to \tau(R)$. By induction hypothesis (1), $\tau(R) \in [\![B]\!]_v$. Hence $\tau(P) \in [\![\Diamond B]\!]_v$.

  (2) Let τ be a transposition. If $P \in \tau([\![\Diamond B]\!]_v)$ then $\tau(P) \in [\![\Diamond B]\!]_v$. Thus there is Q such that $\tau(P) \to Q$ and $Q \in [\![B]\!]_v$. By Proposition 2.15(2), and induction hypothesis (2), $P \to \tau(Q)$ and $\tau(Q) \in [\![\tau(B)]\!]_{\tau(v)}$. Hence $P \in [\![\tau(\Diamond B)]\!]_{\tau(v)}$.

- Case $A = Z$.

  (1) We have $[\![Z]\!]_v = v(Z)$. Since $fn^v(Z) = supp(v(Z))$, we have that $[\![Z]\!]_v \in \mathbb{P}_{fn^v(Z)}$.

(2) Let τ be a transposition. If $P \in \tau([\![Z]\!]_v)$ then $\tau(P) \in v(Z)$. Therefore, $P \in \tau(v(Z)) = \tau(v)(Z) = [\![Z]\!]_{\tau(v)}$.

- Case $A = \forall Z.B$.

  (1) Let v' be the restriction of v to the free propositional variables of A. By Remark (4.19), since $[\![A]\!]_v = [\![A]\!]'_v$, we show the property w.r.t. v'.

  Let $M = fn^{v'}(\forall Z.B) = fn^{v'[Z\leftarrow\emptyset]}(B)$. By Lemma 4.8, to verify $[\![\forall Z.B]\!]_{v'} \in \mathbb{P}_M$, it suffices to check that $S \triangleq \{[\![B]\!]_{v'[Z\leftarrow\Psi]} \mid \Psi \in \mathbb{P}_-\}$ is a finitely supported (by M) set of Psets. Pick $m, n \notin M$ and $\Phi \in S$: then $\Phi = [\![B]\!]v'[Z\leftarrow\Psi]$ for some $\Psi \in \mathbb{P}_-$ and let $\tau = \{m\leftrightarrow n\}$. By induction hypothesis (2), we have that $\tau(\Phi) = [\![\tau(B)]\!]_{\tau(v')[Z\leftarrow\tau(\Phi)]}$. Since $\tau(v') = v'$ and $\tau(B) = B$, we have $\tau(\Phi) = [\![B]\!]_{v'[Z\leftarrow\tau(\Phi)]} \in S$.

  (2) Let τ be a transposition. Let $P \in \tau([\![\forall Z.B]\!]_v)$. Then $\tau(P) \in [\![\forall Z.B]\!]_v$. This means that for all $\Phi \in \mathbb{P}_-$, we have $\tau(P) \in [\![B]\!]_{v[Z\leftarrow\Phi]}$. Therefore $P \in \tau([\![B]\!]_{v[Z\leftarrow\Phi]})$. By induction hypothesis (2), we have $P \in [\![\tau(B)]\!]_{\tau(v)[Z\leftarrow\tau(\Phi)]}$, for all $\Phi \in \mathbb{P}_-$. Since τ is a bijection $\mathbb{P}_- \mapsto \mathbb{P}_-$, we conclude that for all $\Phi \in \mathbb{P}_-$, $P \in [\![\tau(B)]\!]_{\tau(v)[Z\leftarrow\Phi]}$. So, $P \in [\![\tau(\forall Z.B)]\!]_{\tau(v)}$ and indeed $\tau([\![\forall Z.B]\!]_v) \subseteq [\![\tau(\forall Z.B)]\!]$.

∎

Lemma 4.29

Let $[\![\theta(A)]\!]_v = [\![\theta(B)]\!]_v$ for all substitutions θ and valuations v, and let $F[-]$ be a context. Then, for all substitutions σ and valuations w we have $[\![\sigma(F[A])]\!]_w = [\![\sigma(F[B])]\!]_w$.

Proof. By induction on the size of the context $F[-]$.

- Case $F[-] = G[-] \Rightarrow H[-]$.

  By definition, $[\![\sigma(F[A])]\!]_w$ is the set of all processes Q such that if $Q \in [\![\sigma(G[A])]\!]_w$ then $Q \in [\![\sigma(H[A])]\!]_w$. By induction hypothesis, $[\![\sigma(G[A])]\!]_w = [\![\sigma(G[B])]\!]_w$ and $[\![\sigma(H[A])]\!]_w = [\![\sigma(H[B])]\!]_w$.

  Hence $[\![\sigma(F[A])]\!]_v = [\![\sigma(F[B])]\!]_v$.

- Cases $F[-] = G[-] \wedge H[-]$, $F[-] = G[-]|H[-]$, $F[-] = G[-] \triangleright H[-]$, $F[-] = \boxdot G[-]$, $F[-] = n \circledR G[-]$, and $F[-] = n \oslash G[-]$. By induction hypothesis, as above.

- Case $F[-] = \forall x.G[-]$. Assume $x \notin \mathfrak{D}(\sigma)$.

  We have $[\![\sigma(F[A])]\!]_w = \bigcap_{n\in\Lambda} [\![\sigma(G[A])\{x\leftarrow n\}]\!]_w$. By induction hypothesis, $[\![\sigma(G[A])\{x\leftarrow n\}]\!]_w = [\![\sigma(G[B])\{x\leftarrow n\}]\!]_w$, for all n.

  Therefore $\bigcap_{n\in\Lambda} [\![\sigma(G[A])\{x\leftarrow n\}]\!]_w = \bigcap_{n\in\Lambda} [\![\sigma(G[B])\{x\leftarrow n\}]\!]_w$.

  But $\bigcap_{n\in\Lambda} [\![\sigma(G[B])\{x\leftarrow n\}]\!]_w = [\![\sigma(F[B])]\!]_w$.

- Case $F[-] = \text{И}x.G[-]$. Assume $x \notin \mathfrak{D}(\sigma)$.

  (Left to right inclusion) Pick $P \in [\![\sigma(\text{И}x.G[A])]\!]_w$. Then there is $n \notin fn^w(\sigma(G[A])) \cup fn(P)$ such that $P \in [\![\sigma(G[A])\{x\leftarrow n\}]\!]_w$.

  Let $m \notin fn^w(\sigma(G[A])) \cup fn^w(\sigma(G[B])) \cup fn(P)$, let $\tau = \{m\leftrightarrow n\}$. Let u be the restriction of w to the free propositional variables of $G[A]$, we have

$[\![\sigma(G[A])\{x\leftarrow n\}]\!]_w = [\![\sigma(G[A])\{x\leftarrow n\}]\!]_u$. By Theorem 4.21(2), $\tau(P) = P \in \tau([\![\sigma(G[A])\{x\leftarrow n\}]\!]_u) = [\![\sigma(G[A])\{x\leftarrow m\}]\!]_u = [\![\sigma(G[A])\{x\leftarrow m\}]\!]_w$.

By the induction hypothesis we conclude $P \in [\![\sigma(G[B])\{x\leftarrow m\}]\!]_w$.

Since $m \notin fn^w(\sigma(G[B]))$, we obtain $P \in [\![\sigma(F[B])]\!]_w$.

(Right to left inclusion) Symmetrical.

- Case $F[-] = X$. Then $F[A] = X = F[B]$ and we have $[\![\sigma(F[A])]\!]_w = [\![\sigma(X)]\!]_w = w(X) = [\![\sigma(X)]\!]_w = [\![\sigma(F[B])]\!]_w$.
- Case $F[-] = \forall Z.G[-]$.

  (Left to right inclusion) Take $P \in [\![\sigma(\forall Z.G[A])]\!]_w$. By definition, for all $\Psi \in \mathbb{P}_-$, $P \in [\![\sigma(G[A])]\!]_{w[Z\leftarrow\Psi]}$.

  By induction hypothesis, for all $\Psi \in \mathbb{P}_-$, $P \in [\![\sigma(G[B])]\!]_{w[Z\leftarrow\Psi]}$. We conclude $P \in [\![\sigma(\forall Z.G[B])]\!]_w$.

  (Right to left inclusion) Handled symmetrically.

∎

Lemma 5.3

1. $[\![Иx.(A|B)]\!]_v = [\![Иx.A|Иx.B]\!]_v$
2. $[\![Иx.(A \triangleright B)]\!]_v \subseteq [\![Иx.A \triangleright Иx.B]\!]_v$
3. $[\![Иx.\Diamond A]\!]_v = [\![\Diamond Иx.A]\!]_v$
4. $[\![Иx.n \circledR A]\!]_v = [\![n \circledR Иx.A]\!]_v$
5. $[\![Иx.\forall y.A]\!]_v \subseteq [\![\forall y.Иx.A]\!]_v$
6. $[\![Иx.\forall X.A]\!]_v \subseteq [\![\forall X.Иx.A]\!]_v$

Proof. 1. (Right to left inclusion) Let $P \models_v Иx.A|Иx.B$. Then there are processes Q and R such that $P \equiv Q|R$ and $Q \models_v Иx.A$ and $R \models_v Иx.B$. Pick a name $n \notin fn(P) \cup fn^v(Иx.(A|B))$. Then $n \notin fn(Q) \cup fn^v(Иx.A)$ and $n \notin fn(R) \cup fn^v(Иx.B)$. By Proposition 5.1(3) we have $Q \models_v A\{x\leftarrow n\}$ and $R \models_v B\{x\leftarrow n\}$. Then $P \models_v Иx.(A|B)$ as claimed. (Left to right inclusion) Similar, use Proposition 5.1(2).

2. Let $P \models_v Иx.(A \triangleright B)$. Then for all names $n \notin fn(P) \cup fn^v(A) \cup fn^v(B)$ and any process Q such that $Q \models_v A\{x\leftarrow n\}$ we have $Q|P \models_v B\{x\leftarrow n\}$. Pick any R such that $R \models_v Иx.A$. We need to show that $P|R \models_v Иx.B$. Pick a name p where $p \notin fn(P) \cup fn(R) \cup fn^v(B) \cup fn(R) \cup fn^v(A)$. We have $R \models_v A\{x\leftarrow p\}$. By the assumption, we conclude $R|P \models_v B\{x\leftarrow p\}$. But then $R|P \models_v Иx.B$.

3. (Left to right inclusion) Let $P \models_v Иx.\Diamond A$. Then there is a process Q, and a name n, fresh w.r.t. P and $fn^v(A)$ such that $P \to Q$ and $Q \models_v A\{x\leftarrow n\}$. But if n is fresh w.r.t P it is also fresh w.r.t. Q since $fn(Q) \subseteq fn(P)$. Hence $Q \models_v Иx.A$ and $P \models_v \Diamond Иx.A$. (Right to left inclusion) Let $P \models_v \Diamond Иx.A$. Then there is Q and n fresh w.r.t. Q and $fn^v(A)$ such that $P \to Q$ and $Q \models_v A\{x\leftarrow n\}$. Pick any name m, fresh w.r.t. v and P (and thus fresh w.r.t. Q). Let $\tau = \{m\leftrightarrow n\}$, by Theorem 4.21(2) we have $Q \models_v A\{x\leftarrow m\}$, since $\tau(Q) = Q$, $\tau(v) = v$ and $\tau(A\{x\leftarrow n\}) = A\{x\leftarrow m\}$. Hence $P \models_v \Diamond A\{x\leftarrow m\}$, and then $P \models_v Иx.\Diamond A$.

4. (Right to left inclusion) Let $P \models_v n \circledR Иx.A$. Then there is a process Q such that $P \equiv (\boldsymbol{\nu} n)Q$ and $Q \models_v A\{x\leftarrow p\}$ for all $p \notin fn^v(A) \cup fn(Q)$. Let $q \notin fn^v(A) \cup fn(Q) \cup fn(P) \cup \{n\}$. Then $P \models_v n \circledR A\{x\leftarrow q\}$. We conclude $P \models_v Иx.n \circledR A$. (Left to right inclusion) Let $P \models_v Иx.n \circledR A$. Then there is $q \notin fn(P) \cup \{n\} \cup fn^v(A)$ such that $P \equiv (\boldsymbol{\nu} n)Q$ and $Q \models_v A\{x\leftarrow q\}$. Since $q \neq n$, we conclude $q \notin fn(Q)$. Thus $Q \models_v Иx.A$. We conclude $P \models_v n \circledR Иx.A$.

5. Let $P \models_v Иx.\forall y.A$. Then for all $p \notin fn^v(\forall y.A) \cup fn(P)$ and all $n \in \Lambda$ we have $P \models_v A\{x \leftarrow p\}\{y \leftarrow n\}$. Thus, for all $n \in \Lambda$ and all $p \notin \{n\} \cup fn^v(\forall y.A) \cup fn(P)$ we have $P \models_v A\{x \leftarrow p\}\{y \leftarrow n\}$. Hence $P \models_v \forall y.Иx.A$.

6. Let $P \models_v Иx.\forall X.A$. Then for all $p \notin fn^v(\forall X.A) \cup fn(P)$ and all $\Psi \in \mathbb{P}_-$ we have $P \models_{v[X \leftarrow \Psi]} A\{x \leftarrow p\}$. Thus, for all $\Psi \in \mathbb{P}_-$ and all $p \notin supp(\Psi) \cup fn^v(\forall X.A) \cup fn(P)$ we have $P \models_{v[X \leftarrow \Psi]} A\{x \leftarrow p\}$. Thus, for all $\Psi \in \mathbb{P}_-$ we have $P \models_{v[X \leftarrow \Psi]} Иx.A$. Hence $P \models_v \forall X.Иx.A$. ∎

Proposition 6.5

For any formula A and valuation v, and Psets Ψ, Φ

1. If $X \notin \mathrm{Neg}(A)$ and $\Psi \subseteq \Phi$ then $[\![A]\!]_{v[X \leftarrow \Psi]} \subseteq [\![A]\!]_{v[X \leftarrow \Phi]}$.
2. If $X \notin \mathrm{Pos}(A)$ and $\Psi \subseteq \Phi$ then $[\![A]\!]_{v[X \leftarrow \Phi]} \subseteq [\![A]\!]_{v[X \leftarrow \Psi]}$.

Proof. The proof of (1–2) proceeds by mutual induction on the size of the formula A. Note that the size of a formula does not change when replacing names and variables for names.

- Cases $A = \mathbf{F}$, $A = \mathbf{0}$ and $p\langle q \rangle$.

  Immediate.

- Cases $A = B \wedge C$ and $A = \Diamond B$.

  Immediate, by the induction hypothesis.

- Case $A = B \Rightarrow C$.

  (1) Then $X \notin \mathrm{Neg}(C)$ and $X \notin \mathrm{Pos}(B)$. Assume $P \in [\![B \Rightarrow C]\!]_{v[X \leftarrow \Psi]}$. This implies that if $P \in [\![B]\!]_{v[X \leftarrow \Psi]}$ then $P \in [\![C]\!]_{v[X \leftarrow \Psi]}$. Assume $P \in [\![B]\!]_{v[X \leftarrow \Phi]}$. Since $X \notin \mathrm{Pos}(B)$, by induction hypothesis (2), $P \in [\![B]\!]_{v[X \leftarrow \Psi]}$. Then $P \in [\![C]\!]_{v[X \leftarrow \Psi]}$. By induction hypothesis (1), $P \in [\![C]\!]_{v[X \leftarrow \Phi]}$. Hence $P \in [\![B \Rightarrow C]\!]_{v[X \leftarrow \Phi]}$.

  (2) Symmetric to (3).

- Cases $A = B|C$ and $A = B \triangleright C$

  Like (Case $A = B \wedge C$) and (Case $A = B \Rightarrow C$) above.

- Case $A = q{\circledR}B$.

  (1) Then $X \notin \mathrm{Pos}(B)$. Assume $P \in [\![q{\circledR}B]\!]_{v[X \leftarrow \Psi]}$. Then $P \equiv (\boldsymbol{\nu} q)Q$ and $Q \in [\![B]\!]_{v[X \leftarrow \Psi]}$. By induction hypothesis (1), $Q \in [\![B]\!]_{v[X \leftarrow \Phi]}$. Hence $P \in [\![q{\circledR}B]\!]_{v[X \leftarrow \Phi]}$.

  (2) Symmetrical.

- Case $A = B \oslash q$.

  Like (Case $A = q{\circledR}B$) above.

- Case $A = Иx.B$.

  (1) We have $X \notin \mathrm{Neg}(B\{x \leftarrow n\})$, for all $n \in \Lambda$. Let $P \in [\![Иx.B]\!]_{v[X \leftarrow \Psi]}$ and w be the restriction of v to the free propositional variables of $Иx.B$ (we assume that $X \in fpv(A)$ otherwise the result is immediate).

By Remark 4.19, $P \in \llbracket \text{И} x.B \rrbracket_{w[X \leftarrow \Psi]}$. Then, there is q such that $q \notin \mathit{fn}^{w[X \leftarrow \Psi]}(B) \cup \mathit{fn}(P)$ and $P \in \llbracket B\{x \leftarrow q\} \rrbracket_{w[X \leftarrow \Psi]}$.

Now, pick $p \notin \mathit{fn}^{w[X \leftarrow \Psi]}(B) \cup \mathit{fn}(P) \cup \mathit{fn}^{w[X \leftarrow \Phi]}(B)$ and define $\tau = \{q \leftrightarrow p\}$. Note that $p, q \notin \mathit{supp}(\Psi) \cup \mathit{fn}(w)$. By Theorem 4.21(2), we have $\tau(P) = P \in \tau(\llbracket B\{x \leftarrow q\} \rrbracket_{w[X \leftarrow \Psi]}) \subseteq \llbracket B\{x \leftarrow p\} \rrbracket_{w[X \leftarrow \Psi]}$, since $\tau(\Psi) = \Psi$ and $\tau(w) = w$.

By induction hypothesis (1), $P \in \llbracket B\{x \leftarrow p\} \rrbracket_{w[X \leftarrow \Phi]}$.

But p was chosen such that $p \notin \mathit{fn}^{w[X \leftarrow \Phi]}(A)$, so $P \in \llbracket \text{И} x.B \rrbracket_{w[X \leftarrow \Phi]}$. This implies $P \in \llbracket \text{И} x.B \rrbracket_{v[X \leftarrow \Phi]}$, by Remark 4.19.

(4) Symmetrical.

- Case $A = \forall x.B$.

  (1) We have $X \notin \mathrm{Neg}(B\{x \leftarrow n\})$, for all $n \in \Lambda$. Let $P \in \llbracket \forall x.B \rrbracket_{v[X \leftarrow \Psi]}$, that is, for all $n \in \Lambda$, $P \in \llbracket B\{x \leftarrow n\} \rrbracket_{v[X \leftarrow \Psi]}$. By induction hypothesis (1), for each $n \in \Lambda$, $P \in \llbracket B\{x \leftarrow n\} \rrbracket_{v[X \leftarrow \Phi]}$. Hence $P \in \llbracket \forall x.B \rrbracket_{v[X \leftarrow \Phi]}$.

  (2) Symmetrical.

- Case $A = Z$.

  (1) The case $Z \neq X$ is trivial. If $Z = X$, the assumption yields $\llbracket X \rrbracket_{v[X \leftarrow \Psi]} = \Psi \subseteq \Phi = \llbracket X \rrbracket_{v[X \leftarrow \Phi]}$.

  (2) $X \notin \mathrm{Pos}(Z)$ implies $X \neq Z$, and we conclude.

- Case $A = \forall Z.B$.

  (1) Then $Z \notin \mathrm{Neg}(B)$. By Lemma 4.18, we may assume $Z \neq X$. Assume $P \in \llbracket \forall Z.B \rrbracket_{v[X \leftarrow \Psi]}$. This implies that $P \in \llbracket B \rrbracket_{v[X \leftarrow \Psi][Z \leftarrow \Theta]}$ for all $\Theta \in \mathbb{P}_-$. By induction hypothesis (1), $P \in \llbracket B \rrbracket_{v[X \leftarrow \Phi][Z \leftarrow \Theta]}$, for all $\Theta \in \mathbb{P}_-$. Hence $P \in \llbracket \forall Z.B \rrbracket_{v[X \leftarrow \Phi]}$.

  (2) Symmetrical.

∎