A Spatial Logic for Concurrency (Part II)

Luís Caires[1] and Luca Cardelli[2]

[1] Departamento de Informática FCT/UNL, Lisboa, Portugal
[2] Microsoft Research, Cambridge, UK

We present a modal logic for describing the spatial organization and the behavior of distributed systems. In addition to standard logical and temporal operators, our logic includes spatial operations corresponding to process composition and name hiding, and a fresh quantifier.

1 Introduction

We develop a logic to describe properties of distributed concurrent systems, for specification and model-checking purposes; we believe that the peculiar characteristics of such systems justify the introduction of new logical constructs. Our first emphasis is on *distributed* systems, meaning that we should be able to talk about properties of distinct subsystems, such as subsystems that reside at different locations, and subsystems that privately share hidden resources. For this purpose, we introduce *spatial* (as opposed to temporal) logical operators; for example, we may talk about a property holding somewhere (as opposed to sometimes). Our second emphasis is on *concurrent* systems: we want a logic that unambiguously talks about concurrency and (nowadays) privacy. For this purpose, the intended model of our logic is built explicitly from a standard process calculus (an asynchronous π-calculus). Our formulas denote collections of processes subject to certain closure conditions, with some logical operators mapping directly to process composition and name hiding. In Part I of this paper [2, 4] we study this intended model, which is used here to establish the soundness of the logical rules. The central focus of this Part II, however, is proof theory. We regularize and generalize the logics introduced in [1, 7, 8], and we prove a cut-elimination result for the first-order fragment, including cut-elimination for a fresh name quantifier (*cf.* Nominal Logic [13]).

A formula in our logic describes a property of a particular part of a concurrent system (a *world*) at a particular time; therefore it is modal in space as well as in time. In our sequents, formulas are indexed by the worlds they predicate over [17], so a sequent can talk about many distinct worlds at once. Each sequent incorporates also a finite set of constraints over the worlds, including process reduction and congruence constraints. In general, the constraint structure can be fashioned as an algebra [18]; which in our case is a relatively complex process algebra.

The fragment of our logic that deals with process composition is relatively straightforward: composition shows up in the logic as a tensor, which is strongly related to linear connectives. The sequent-style presentation of this fragment should look relatively familiar, except for the constraints part. The relevant constraints are essentially constraints over a (concurrency) monoid, with some specific interactions with reduction. Along these lines, we could also easily add an explicit structure of locations to the process calculus, and related logical operators, as done in [7].

Far less obvious is what to do about hiding of private resources, which is represented in π-calculus by the name hiding operator. The hiding of a name in a process should correspond, logically, to a "hiding quantifier" that binds a private name in a formula; such a formula could then describe the use of that private name in the process. The study of such a quantifier, from a logical point of view, was started in [5, 1], and later independently in [8]. Our current understanding is that it is best to decompose such a hiding quantifier into two operators: a modal version of the fresh quantifier of Gabbay and Pitts [10], and a logical operator, called *revelation* [8], that relates to name hiding in strong analogy to the way tensor relates to process composition. A simple combination of fresh quantification and revelation then yields hiding, in the intuitive sense that if something is hidden, we can choose to name it (reveal it) by any name that is fresh.

Many natural examples of use of our logic involve recursive formulas. Two typical examples of recursion that attract us in our context are: (1) a process having an arbitrary number of hidden resources, and (2) a process generating an infinite supply of fresh names. Particularly, the interaction of recursion and freshness is semantically quite challenging, and was investigated in Part I. In this paper we go one step further and introduce second-order quantification in the logic, from which we can define least and greatest fixpoints of formulas, almost along standard lines [14].

Structurally, our logic consists of a collection of left-right rules for logical operators, including essentially the standard rules of classical sequent calculus, plus the ones for temporal and spatial operators. In addition, there are special rules about the worlds: they add meaning to the logical operators, allowing us to capture deep properties of process calculi without interfering very much with the core left-right rules.

We highlight here the left and right rules for composition, $A|B$, which include many of the interesting features of our sequents. Sequents have the form $\langle S \rangle\, \Gamma \vdash \Delta$, where $\langle S \rangle$ is a finite set of constraints, and Γ , Δ are multisets of indexed formulas.

$$\frac{\begin{array}{c}[\mathcal{X} \textit{ and } \mathcal{Y} \textit{ not free in the conclusion}]\\ \langle S, u \doteq \mathcal{X}|\mathcal{Y} \rangle\, \Gamma, \mathcal{X} : A, \mathcal{Y} : B \vdash \Delta\end{array}}{\langle S \rangle\, \Gamma, u : A|B \vdash \Delta}\ (|\mathrm{L}) \quad \frac{\begin{array}{c}\langle S \rangle\, \Gamma \vdash v : A, \Delta\\ \langle S \rangle\, \Gamma \vdash t : B, \Delta \quad u \doteq_S v|t\end{array}}{\langle S \rangle\, \Gamma \vdash u : A|B, \Delta}\ (|\mathrm{R})$$

The $(|\mathrm{R})$ rule says: if we can show that index v satisfies formula A (i.e, that A holds at world v, written $v : A$), and that t satisfies B, and if we can show from the constraints in S that $u \doteq v|t$, then we can conclude that u satisfies $A|B$. Hence, the reading of this logical rules incorporates much of the intended satisfaction semantics [17]. The $(|\mathrm{L})$ rule features the assumption "$\mathcal{X}$ and $\mathcal{Y}$ not free in the conclusion (of the rule)". This assumption means, in particular, that $\mathcal{X}$ and $\mathcal{Y}$ are completely generic and unconstrained variables. A reading is: to show that $u : A|B$ entails Δ, we must show that for an arbitrary decomposition of u as $\mathcal{X}|\mathcal{Y}$, we have that $\mathcal{X} : A$ and $\mathcal{Y} : B$ entail Δ.

Composition also has a number of "rules about the world", as mentioned above. Here is a simple one:

$$\frac{\langle S, u \doteq \mathbf{0} \rangle\, \Gamma \vdash \Delta \quad u|v \doteq_S \mathbf{0}}{\langle S \rangle\, \Gamma \vdash \Delta}\ (\mathrm{S}|\mathbf{0})$$

Note that these world rules do not involve the logical connectives (we have $\Gamma \vdash \Delta$ above and below), and instead affect the $\langle S \rangle$ part. In most process calculi we have that if $u|v \equiv 0$ then both $u \equiv 0$ and $v \equiv 0$. This property does not derive from (|L) and (|R), but is embedded in (S|$\mathbf{0}$). The rule reads as follows: if we can already infer from the S part of the constraints that $u|v \doteq_S \mathbf{0}$, and we have an additional constraint that $u \doteq \mathbf{0}$, that constraint is redundant and we can remove it. In this style, we can incorporate many peculiar properties of process calculi as world rules; many such rules analyze the consequences of an equation between two spatial operators (above, | vs. $\mathbf{0}$), and are listed in Figure 8. All such rules have a similar reading in terms of eliminating "redundant" constraints.

Because of the regular left-right structure of our core rules, cut elimination falls largely along predictable lines; the indexes do not hinder, and rules such as (S|$\mathbf{0}$) can be dealt with separately. The main difficulty is in the cut elimination case for the freshness quantifier. As in Nominal Logic, the result depends on an "equivariance" property of the logic [13], which is used to perform an α-conversion of fresh names over a whole derivation. Equivariance is embedded, in our case, in the (τ) rule in Figure 7, whose soundness depends on the main theorem of Part I. Expressing this rule in the general case of open formulas, requires introducing explicit transpositions over formulas, which entail some technical complications.

Related Work A logic for a process calculus including a tensor operator and a hiding quantifier was developed by Luís Caires in [5, 1], but a satisfactory semantic treatment for the latter connective not was achieved before the contributions of [8, 2]. Andy Gordon was a coauthor with Luca Cardelli of initial versions of spatial logics for the Ambient Calculus [7, 8], which also investigated connections with linear logic. The present paper contains the first presentation of such a logic as a proper sequent calculus. Moreover, we now target the logic towards a more standard π-calculus.

The first main difference between our logic and standard logics of concurrency (e.g. [11]) is the presence in our case of a tensor operator that corresponds to process composition. 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 by Dam [9]). 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 [16].

The work of Gabbay and Pitts on the freshness quantifier [10] has become central to our logic. The work of O'Hearn and Pym on Bunched Logics [12] and of Reynolds on Separation Logic [15] is closely related to ours, at least in intent. The style in which the logic is formalized is an extension of work by Alex Simpson [17], and is also related, at least superficially, to labeled deductive systems [18]. A decidable and complete propositional fragments of a related logic has been recently investigated [6].

In Section 2 we introduce the algebra of constraints. In Section 3 we introduce our sequent calculus, which can be shown sound by an interpretation in the model of Part I. In Section 4 we investigate proof theory, and in particular cut elimination for the first-order fragment of our logic. In Section 5 we go through a set of basic examples, to illustrate the expressive power of the logic.

2 Preliminaries

In this Section we introduce π-algebras, and constraint theories over the term π-algebra. A π-algebra is a two sorted algebra, with a sort for names and a sort for processes, and equipped with the basic process operations of composition, name hiding and name transposition. Hence, many process calculi are π-algebras, in particular the asynchronous π-calculus Aπ which is the intended model of our logic. We summarize the main concepts needed for our current purposes, a more detailed presentation can be found in [3].

Definition 2.1 (π-algebra). *A π-algebra is a structure $\langle \mathcal{L}, \mathcal{P}, \mathbf{0}, |, \boldsymbol{\nu}, (\leftrightarrow)_{\mathcal{L}}, (\leftrightarrow)_{\mathcal{P}} \rangle$ such that $\mathcal{L}$ is a countable set of* labels (ℓ)*, $\mathcal{P}$ is a set of* processes (P, Q, R)*, $\mathbf{0}$ (void) is a distinguished process in $\mathcal{P}$, $-|-$ (*composition*) is an operation $\mathcal{P} \times \mathcal{P} \to \mathcal{P}$, $(\boldsymbol{\nu}-)-$ (*name hiding*, a.k.a. restriction) is an operation $\mathcal{L} \times \mathcal{P} \to \mathcal{P}$, $(- \leftrightarrow -)_{\mathcal{L}} -$ (*transposition on labels*) is an operation $\mathcal{L} \times \mathcal{L} \times \mathcal{L} \to \mathcal{P}$ and $(- \leftrightarrow -)_{\mathcal{P}} -$ (*transposition on processes*) is an operation $\mathcal{L} \times \mathcal{L} \times \mathcal{P} \to \mathcal{L}$. We refer to the $\mathcal{L}$ part of a π-algebra Π by $\Pi_{\mathcal{L}}$, and likewise for the remaining components (e.g., $\Pi_{\mathcal{P}}$).*

Of special interest is the term π-algebra **P**, whose terms of sort process (called *indexes*) will be used to label formulas in sequents and denote the worlds of our modal logic, while terms of sort name (called *name terms*) denote the pure names used in processes. Given a set $\mathcal{V}$ of names variables and a set χ of process variables, the term π-algebra **P** is the free π-algebra over a set of variables $x \in \mathcal{V}$ for the sort of *name terms* $\mathcal{N}$, and over a set of variables $\mathcal{X} \in \chi$ for the sort of *indexes* $\mathcal{I}$. We use the meta-variables $m, n, p \in \mathcal{N}$; $u, v, t \in \mathcal{I}$; $\mathcal{X}, \mathcal{Y}, \mathcal{Z} \in \chi$; $x, y, z \in \mathcal{V}$; $\gamma \in \mathcal{G} = \mathcal{I} \cup \mathcal{N}$. Given an index or name term γ, we denote by *afv*(γ) its set of algebraic free (name and process) variables, defined simply as the collection of all the variables in $\mathcal{V}$ and χ occurring in such terms.

Processes relate to each other both by spatial and temporal constraints: in the term π-algebra such constraints are represented by *spatial constraints*, expressing that the processes denoted by the equated indexes have the same spatial structure (*cf.* π-calculus structural congruence), and *temporal constraints*, expressing that the processes denoted by the given indexes are related by a reduction (*cf.* π-calculus reduction).

Definition 2.2 (Constraint). *A* constraint *c is either an index equation, a reduction, a name apartness, or a set apartness, defined by*

$c ::=$	Constraints				
$u \doteq v$	Index equation	$(u, v \in \mathcal{I})$	$u \to v$	Reduction	$(u, v \in \mathcal{I})$
$m \# n$	Name apartness	$(m, n \in \mathcal{N})$	$m \# X$	Set apartness	$(m \in \mathcal{N}, X \in \mathcal{X})$

In order to handle freshness explicitly, we also introduce *apartness constraints*: $m \# n$ meaning that the name terms m and n denote distinct names, and $m \# X$ meaning that the name term m denotes a name distinct from any name in the (finite) support [2] of the property denoted by the propositional variable X (see Section 3.1).

A *constraint theory* is a finite set of constraints, presenting a particular world structure. Such a structure enjoys a number of structural properties, axiomatized by the set of closure rules in Fig. 1. For instance, (Spatial) characterizes properties of structural congruence (*e.g.,* monoidal laws for composition and name extrusion). We write $S \vdash c$ to say that the constraint c holds in the closure of S, writing $\gamma \doteq_S \gamma'$ for $S \vdash \gamma \doteq \gamma'$, and likewise for the relations $\#_S$ and $\to_S$.

(Basic)
$u \doteq v \in S \Rightarrow u \doteq_S v$
$m \,\#\, n \in S \Rightarrow m \,\#_S\, n$
$u \to v \in S \Rightarrow u \to_S v$

(Spatial)
$u|\mathbf{0} \doteq_S u$
$u|v \doteq_S v|u$
$(u|v)|t \doteq_S u|(v|t)$
$(\boldsymbol{\nu} n)\mathbf{0} \doteq_S \mathbf{0}$
$(\boldsymbol{\nu} n)(\boldsymbol{\nu} n)u \doteq_S (\boldsymbol{\nu} n)u$
$(\boldsymbol{\nu} m)(\boldsymbol{\nu} n)u \doteq_S (\boldsymbol{\nu} n)(\boldsymbol{\nu} m)u$
$(\boldsymbol{\nu} n)(u|(\boldsymbol{\nu} n)v) \doteq_S ((\boldsymbol{\nu} n)u)|(\boldsymbol{\nu} n)v$

(Congruence)
$\gamma \doteq_S \gamma$
$\gamma \doteq_S \gamma' \Rightarrow \gamma' \doteq_S \gamma$
$\gamma \doteq_S \gamma', \gamma' \doteq_S \gamma'' \Rightarrow \gamma \doteq_S \gamma''$
$u \doteq_S v \Rightarrow u|t \doteq_S v|t$
$u \doteq_S v, m \doteq_S n \Rightarrow (\boldsymbol{\nu} m)u \doteq_S (\boldsymbol{\nu} n)v$
$m \doteq_S n, \gamma \doteq_S \gamma' \Rightarrow (m \leftrightarrow r)\gamma \doteq_S (r \leftrightarrow n)\gamma'$

(Apartness)
$m \,\#_S\, p, n \,\#_S\, p \Rightarrow (m \leftrightarrow n)p \doteq_S p$
$m \,\#_S\, n \Rightarrow n \,\#_S\, m$
$p \,\#_S\, q, p \doteq_S p', q \doteq_S q' \Rightarrow p' \,\#_S\, q'$
$p \,\#_S\, X, p \doteq_S q \Rightarrow q \,\#_S\, X$

(Transposition)
$\tau\mathbf{0} \doteq_S \mathbf{0}$
$\tau(u|v) \doteq_S \tau u|\tau v$
$\tau(\boldsymbol{\nu} p)u \doteq_S (\boldsymbol{\nu} \tau p)\tau u$
$\tau(p \leftrightarrow q)\gamma \doteq_S (\tau p \leftrightarrow \tau q)\tau\gamma$
$\tau\tau\gamma \doteq_S \gamma$
$(n \leftrightarrow n)\gamma \doteq_S \gamma$
$(m \leftrightarrow n)m \doteq_S n$
$u \doteq_S (\boldsymbol{\nu} n)t, u \doteq_S (\boldsymbol{\nu} m)v \Rightarrow$
$(n \leftrightarrow m)u \doteq_S u$

(Reduction)
$u \to_S t, v \doteq_S u, t \doteq_S w \Rightarrow v \to_S w$
$u \to_S t \Rightarrow u|v \to_S t|v$
$u \to_S t \Rightarrow (\boldsymbol{\nu} n)u \to_S (\boldsymbol{\nu} n)t$
$u \to_S t \Rightarrow \tau u \to_S \tau t$

Fig. 1. Closure of constraint theories.

3 Sequent Calculus

3.1 Formulas and Sequents

Formulas, defined in Fig. 2, include classical propositional connectives, **F**, $\wedge$, $\Rightarrow$, and the basic spatial operators: $A|B$ (the tensor, representing the parallel composition of processes), **0** (the unit of the tensor, representing the collection of void processes), and $A \triangleright B$ (the linear implication associated with the tensor). This last operator corresponds to context-system specification of processes, which are the concurrency-theory equivalent of pre/post conditions. Name hiding induces a pair of adjunct logical operators. The formula $n®A$ means that a hidden name, which we choose to call n, exists in a restricted scope that satisfies property A. It is matched by a π-calculus term $(\boldsymbol{\nu} n)u$ provided that u satisfies A (see rule (®R) in Fig. 4, and the example in Section 5; see [8, 2] for further details.) The formula $A \oslash n$ is the logical adjunct of $n®A$. The notion of *fresh name* is introduced by a quantifier $\mathsf{И}x.A$; this means that x denotes a name that is fresh with respect to the names used either in A or in processes satisfying A. $\mathsf{И}x.A$ is defined along the lines of the freshness quantifier of Gabbay-Pitts [10, 13], and its semantics is designed to be compatible with recursive formulas. 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

$m, n, p ::=$	Name Terms		$(m, n, p \in \mathcal{N})$	
x		Name variable	$(x \in \mathcal{V})$	
$(m \leftrightarrow n)p$		Transposition term		
$A, B \quad ::=$	Formulas			
$\mathbf{F}$		False	$m \oslash A$	Hiding
$A \wedge B$		Conjunction	$m\langle n \rangle$	Message
$A \Rightarrow B$		Implication	$\Diamond A$	Next
$\mathbf{0}$		Void	$\forall x.A$	Name quantification
$A\|B$		Composition	$\text{И}x.A$	Freshness quantification
$A \triangleright B$		Guarantee	X	Propositional variable
$m Ⓡ A$		Revelation	$\forall X.A$	Property quantification

Formulas are subject to: no occurrence of a variable x in a transposition term is under the scope of a $\forall x.$ or a $\text{И}x.$

Fig. 2. Formulas

us to talk about the behavior of a process after a single (unspecified) reduction step. Finally, we have a second-order quantifier and related propositional variables.

Formulas are the same as in Part I [4], except that we now allow formulas to contain general name terms (including explicit transpositions), and not only variables and pure names. Therefore, we will refer to the formulas as defined in [2] by *simple* formulas.

In $\forall x.A$, $\text{И}x.A$ (and $\forall X.A$), the variables x (and X) are bound with scope the formula A. We assume defined on formulas the standard relation $\equiv_\alpha$ of α-conversion (safe renaming of bound variables), but will never implicitly take formulas "up to α-conversion": our manipulation of variables via α-conversion steps will always be quite explicit. We also assume defined as usual the set $fv(A)$ of *free name variables* in A, and the set $fpv(A)$ of *free propositional variables* in A. Then, we define the set of *logically free variables* of a formula A by $lfv(A) \triangleq fv(A) \cup fpv(A)$. If m is a name term and A is a formula then $A\{x{\leftarrow}m\}$ denotes the formula obtained by replacing of all free occurrences of x in A by the name term m, (nondeterministically) renaming bound variables as needed to avoid capture of names in m.

Any substitution $I_{\mathcal{L}} : \mathcal{V} \to \Lambda$ of pure names for name variables acts on a formula A (respectively, on a name term p) yielding a simple formula $I_{\mathcal{L}}(A)$ (respectively a pure name), defined as expected. The following definition lifts the equational theory of name terms to the level of formulas: $A \equiv_S B$ asserts that, under any substitution that satisfies all constraints in S, A and B denote the same simple formula.

Definition 3.1 (Equational equivalence of formulas). *Given a constraint theory S, $\equiv_S$ is the least equivalence relation on formulas such that $A\{x{\leftarrow}m\} \equiv_S A\{x{\leftarrow}n\}$ for all name terms m, n such that $m \doteq_S n$.*

The semantics of the freshness quantifier requires names to be chosen fresh with respect to the free names of both processes and formulas. By the formation condition at the bottom of Fig. 2, transposition terms in formulas *never contain bound occurrences of name variables*. Hence, a maximal term in a formula is either a bound name variable, or a term completely built out of free name variables: explicit transpositions in formulas just swap *free names*. Thus, it is sensible to define the set $ft(A)$ of *free terms* of a formula A to be the set of maximal name terms occurring in A which are not a bound occurrence of a variable. Given a constraint theory S and a formula A, we write $n \#_S A$ as an abbreviation of $n \#_S ft(A)$: $n \#_S A$ asserts that, under the constraints in S, n denotes a name *distinct* from all names denoted by the (free) terms in formula A. We can verify that the relations $\equiv_S$ (between formulas) and $\#_S$ (between name terms and formulas) defined are sound with respect to their intended interpretation.

A context (Γ, Δ) is a finite multiset of indexed formulas of the form $u : A$ where u is an index and A is a formula.

Definition 3.2 (Sequent). *A* sequent *is a judgment $\langle S \rangle\, \Gamma \vdash \Delta$ where S is a constraint theory, and Δ and Γ are logical contexts.*

The *right context* Δ is interpreted as the disjunction of its formulas, the *left context* Γ is interpreted as the conjunction of the formulas in it. Defining contexts as multisets allows for the implicit use of exchange in proofs.

Definition 3.3 (Variables in sequents). *The set of free (name, process, and propositional) variables of a context Δ is given by $lfv(\Delta) = \bigcup\{afv(u) \cup lfv(A) \mid u : A \in \Delta\}$. The set of free (name, process, and propositional) variables in a sequent $\langle S \rangle\, \Gamma \vdash \Delta$ is given by $afv(S) \cup fv(\Gamma) \cup fv(\Delta)$.*

N.B.: name variables x occur both in indexes u and in formulas A; process variables $\mathcal{X}$ occur only in indexes; propositional variables X occur only in formulas.

We now define the semantics of sequents, namely validity. To that end we need to interpret both constraints and formulas: this will be achieved by logical interpretations.

Definition 3.4 (Logical Interpretation). *A* logical interpretation *$\mathcal{J}$ is pair (v, I) where $I_{\mathcal{P}}$ is a map of process variables into $\mathrm{A}\pi$ processes, $I_{\mathcal{L}}$ is a map name of variables into $\mathrm{A}\pi$ names , and v is a map of propositional variables into property sets.*

Hence, I is an homomorphism I of **P** into $\mathrm{A}\pi$, and v is a valuation as defined in Part I.

Definition 3.5 (Satisfaction and Validity). *The relation of* satisfaction *between logical interpretations $\mathcal{J} = (v, I)$ into $\mathrm{A}\pi$ and constraints is defined thus:*

$$\begin{array}{ll} \mathcal{J} \textit{ satisfies } u \doteq v \Leftrightarrow I_{\mathcal{P}}(u) \equiv I_{\mathcal{P}}(v) & \mathcal{J} \textit{ satisfies } u \to v \Leftrightarrow I_{\mathcal{L}}(u) \to I_{\mathcal{L}}(v) \\ \mathcal{J} \textit{ satisfies } n \# m \Leftrightarrow I_{\mathcal{L}}(n) \neq I_{\mathcal{L}}(m) & \mathcal{J} \textit{ satisfies } n \# X \Leftrightarrow I_{\mathcal{L}}(n) \notin \textit{supp}(v(X)) \end{array}$$

$\mathcal{J}$ satisfies *the constraint theory S if $\mathcal{J}$ satisfies all constraints in S. A constraint $S \vdash c$* is valid *if every interpretation that satisfies S also satisfies c.*

Semantics of formulas is defined in Part I by a mapping $[\![-]\!]_v$ that assigns a property set $[\![A]\!]_v$ to any simple formula A, given a valuation v.

Definition 3.6 (Valid Sequent in $\mathrm{A}\pi$). *A sequent $\langle S \rangle\, \Gamma \vdash \Delta$ is valid in $\mathrm{A}\pi$ if for all logical interpretations $\mathcal{J} = (v, I)$ such that $\mathcal{J}$ satisfies all constraints in S and $I(u) \in [\![I_{\mathcal{L}}(A)]\!]_v$ for all $u : A \in \Gamma$, then $I(v) \in [\![I_{\mathcal{L}}(A)]\!]_v$ for some $v : A \in \Delta$.*

3.2 Inference Rules

Inference rules may have for premises both sequents $\langle S \rangle\, \Gamma \vdash \Delta$ and assertions over the closure of the theory S of the form $u \doteq_S v$ (mostly in the rules for spatial connectives), $A \equiv_S B$ (in the identity axiom), $u \rightarrow_S v$ (in the temporal rules) or $n \#_S A$ (in the freshness rules).

The rules in the identity, structural and propositional groups (see Fig. 3) follow the standard. Note that in (Id) indexes are identified up to $\doteq_S$, while formulas are identified up to $\equiv_S$: (Id) absorbs the simple theory of equality captured by $\equiv_S$, thus axiomatizing the principle of substitution of equals for equals of name terms in formulas. We include explicit contraction rules (CL) and (CR); weakening is admissible, and exchange may be dealt with implicitly, since sequent contexts are multisets. In the rules for propositional connectives, indexes keep track of the processes for which the formulas are asserted to hold, but do not interfere in any way with the constraint part of sequents.

The rules for the spatial connectives (Fig. 4) make essential use of the constraint theories in sequents. Note that the left rules, when read bottom-up, introduce spatial constraints into the constraint theories, and the respective right rules, when read top-down, check corresponding constraints. While spatial rules rely on spatial constraints, temporal rules (Fig. 5) rely on reduction constraints.

The rules for first and second order quantifiers have the expected form (Fig. 6). We then introduce the rules for freshness (Fig. 7). Rule ($\reflectbox{N}$) asserts, when read bottom-up, that there is always a name (denoted by) x that is fresh with respect to the free names of (the process denoted by) the index u, and that is also fresh with respect to a set of names (denoted by the name and propositional variables in) N. Hence, rule ($\reflectbox{N}$) corresponds to the (Fresh) axiom of Pitts' Nominal Logic [13]. The transposition rule (τ) captures the property of invariance of the semantics under transposition of names (see the main theorem of Part I). Moreover, as we shall discuss below, explicit transpositions and the transposition rule play an crucial role in obtaining cut-elimination for the freshness quantifier. By $\{m \leftrightarrow n\} \cdot A$ we denote the formula obtained by swapping occurrences of m and n in A, possibly by introducing explicit transpositions: if $A[n_1, \ldots, n_k]$ is a formula with free terms $n_1, \ldots, n_k$, then $\tau \cdot A[n_1, \ldots, n_k]$ is the formula $A[\tau \cdot n_1, \ldots, \tau \cdot n_k]$.

The rules ($\reflectbox{N}$L/R) for the fresh quantifier do not show the symmetry one might expect of a left / right rule pair. This fact relates to the existential / universal ambivalence of freshness quantification (the Gabbay-Pitts property): note that ($\reflectbox{N}$L) follows the pattern of ($\forall$L), while ($\reflectbox{N}$R) follows the pattern of ($\exists$R). Then, ($\reflectbox{N}$) embodies the introduction of fresh witnesses usually present in *both* ($\forall$R) and ($\exists$L). Besides the freshness condition $n \#_S \reflectbox{N}x.A$ of the name denoted by n with respect to the free names in the formula $\reflectbox{N}x.A$, the assumption $u \doteq_S (\boldsymbol{\nu} n)v$ ensures that n denotes a name that does not occur free in the process denoted by u, *cf.* the semantics of $\reflectbox{N}x.A$.

As discussed in Section 1, world rules (Fig. 8) axiomatize certain deep (extra-logical) properties of the worlds (inversion principles for structural congruence and process reduction). We assert $\vdash \langle S \rangle\, \Gamma \vdash \Delta$ to state that the sequent $\langle S \rangle\, \Gamma \vdash \Delta$ has a derivation. We can show that all rules are sound with respect to the Aπ model of Part I:

Theorem 3.7 (Soundness). *All derivable sequents are valid in* Aπ.

$$\frac{u \doteq_S u' \quad A \equiv_S A'}{\langle S\rangle\, \Gamma, u : A \vdash u' : A', \Delta}\ (\mathrm{Id}) \qquad \frac{\langle S\rangle\, \Gamma \vdash u : A, \Delta \quad \langle S\rangle\, \Gamma, u : A \vdash \Delta}{\langle S\rangle\, \Gamma \vdash \Delta}\ (\mathrm{Cut})$$

$$\frac{\langle S\rangle\, \Gamma, u : A, u : A \vdash \Delta}{\langle S\rangle\, \Gamma, u : A \vdash \Delta}\ (\mathrm{CL}) \qquad \frac{\langle S\rangle\, \Gamma \vdash u : A, u : A, \Delta}{\langle S\rangle\, \Gamma \vdash u : A, \Delta}\ (\mathrm{CR})$$

$$\frac{}{\langle S\rangle\, \Gamma, u : \mathbf{F} \vdash \Delta}\ (\mathbf{F}\mathrm{L}) \qquad \frac{\langle S\rangle\, \Gamma \vdash \Delta}{\langle S\rangle\, \Gamma \vdash u : \mathbf{F}, \Delta}\ (\mathbf{F}\mathrm{R})$$

$$\frac{\langle S\rangle\, \Gamma, u : A, u : B \vdash \Delta}{\langle S\rangle\, \Gamma, u : A \wedge B \vdash \Delta}\ (\wedge\mathrm{L}) \qquad \frac{\begin{array}{c}\langle S\rangle\, \Gamma \vdash u : A, \Delta \\ \langle S\rangle\, \Gamma \vdash u : B, \Delta\end{array}}{\langle S\rangle\, \Gamma \vdash u : A \wedge B, \Delta}\ (\wedge\mathrm{R})$$

$$\frac{\langle S\rangle\, \Gamma \vdash u : A, \Delta \quad \langle S\rangle\, \Gamma, u : B \vdash \Delta}{\langle S\rangle\, \Gamma, u : A \Rightarrow B \vdash \Delta}\ (\Rightarrow\mathrm{L}) \qquad \frac{\langle S\rangle\, \Gamma, u : A \vdash u : B, \Delta}{\langle S\rangle\, \Gamma \vdash u : A \Rightarrow B, \Delta}\ (\Rightarrow\mathrm{R})$$

Fig. 3. Propositional Rules.

$$\frac{\langle S, t \doteq \mathbf{0}\rangle\, \Gamma \vdash \Delta}{\langle S\rangle\, \Gamma, t : \mathbf{0} \vdash \Delta}\ (\mathbf{0}\mathrm{L}) \qquad \frac{u \doteq_S \mathbf{0}}{\langle S\rangle\, \Gamma \vdash u : \mathbf{0}, \Delta}\ (\mathbf{0}\mathrm{R})$$

$$\frac{\begin{array}{c}[\mathcal{X} \textit{ and } \mathcal{Y} \textit{ not free in the conclusion}] \\ \langle S, u \doteq \mathcal{X}|\mathcal{Y}\rangle\, \Gamma, \mathcal{X} : A, \mathcal{Y} : B \vdash \Delta\end{array}}{\langle S\rangle\, \Gamma, u : A|B \vdash \Delta}\ (|\mathrm{L}) \qquad \frac{\begin{array}{c}\langle S\rangle\, \Gamma \vdash v : A, \Delta \\ \langle S\rangle\, \Gamma \vdash t : B, \Delta \quad u \doteq_S v|t\end{array}}{\langle S\rangle\, \Gamma \vdash u : A|B, \Delta}\ (|\mathrm{R})$$

$$\frac{\langle S\rangle\, \Gamma \vdash t : A, \Delta \quad \langle S\rangle\, \Gamma, t|u : B \vdash \Delta}{\langle S\rangle\, \Gamma, u : A \triangleright B \vdash \Delta}\ (\triangleright\mathrm{L}) \qquad \frac{\begin{array}{c}[\mathcal{X} \textit{ not free in the conclusion}] \\ \langle S\rangle\, \Gamma, \mathcal{X} : A \vdash v : B, \Delta \quad v \doteq_S \mathcal{X}|u\end{array}}{\langle S\rangle\, \Gamma \vdash u : A \triangleright B, \Delta}\ (\triangleright\mathrm{R})$$

$$\frac{\begin{array}{c}[\mathcal{X} \textit{ not free in the conclusion}] \\ \langle S, u \doteq (\boldsymbol{\nu} n)\mathcal{X}\rangle\, \Gamma, \mathcal{X} : A \vdash \Delta\end{array}}{\langle S\rangle\, \Gamma, u : n®A \vdash \Delta}\ (®\mathrm{L}) \qquad \frac{\langle S\rangle\, \Gamma \vdash u : A, \Delta \quad t \doteq_S (\boldsymbol{\nu} n)u}{\langle S\rangle\, \Gamma \vdash t : n®A, \Delta}\ (®\mathrm{R})$$

$$\frac{\langle S\rangle\, \Gamma, t : A \vdash \Delta \quad t \doteq_S (\boldsymbol{\nu} n)u}{\langle S\rangle\, \Gamma, u : A \oslash n \vdash \Delta}\ (\oslash\mathrm{L}) \qquad \frac{\langle S\rangle\, \Gamma \vdash u : A, \Delta \quad u \doteq_S (\boldsymbol{\nu} n)t}{\langle S\rangle\, \Gamma \vdash t : A \oslash n, \Delta}\ (\oslash\mathrm{R})$$

Fig. 4. Spatial Rules.

$$\frac{\begin{array}{c}[\mathcal{X} \textit{ not free in the conclusion}] \\ \langle S, u \to \mathcal{X}\rangle\, \Gamma, \mathcal{X} : A \vdash \Delta\end{array}}{\langle S\rangle\, \Gamma, u : \Diamond A \vdash \Delta}\ (\Diamond\mathrm{L}) \qquad \frac{\langle S\rangle\, \Gamma \vdash v : A, \Delta \quad u \to_S v}{\langle S\rangle\, \Gamma \vdash u : \Diamond A, \Delta}\ (\Diamond\mathrm{R})$$

Fig. 5. Temporal Rules.

$$\frac{\langle S\rangle\, \Gamma, u : A\{x\leftarrow n\} \vdash \Delta}{\langle S\rangle\, \Gamma, u : \forall x.A \vdash \Delta}\ (\forall\mathrm{L}) \qquad \frac{\begin{array}{c}[y \textit{ not free in the conclusion}]\\ \langle S\rangle\, \Gamma \vdash u : A\{x\leftarrow y\}, \Delta\end{array}}{\langle S\rangle\, \Gamma \vdash u : \forall x.A, \Delta}\ (\forall\mathrm{R})$$

$$\frac{\langle S\rangle\, \Gamma, u : A\{X\leftarrow B\} \vdash \Delta}{\langle S\rangle\, \Gamma, u : \forall X.A \vdash \Delta}\ (\forall^2\mathrm{L}) \qquad \frac{\begin{array}{c}[Y \textit{ not free in the conclusion}]\\ \langle S\rangle\, \Gamma \vdash u : A\{X\leftarrow Y\}, \Delta\end{array}}{\langle S\rangle\, \Gamma \vdash u : \forall X.A, \Delta}\ (\forall^2\mathrm{R})$$

(In (∀L) and (∀R), the formula $\forall x.A$ must verify the condition in Fig. 2)

Fig. 6. Quantifier Rules.

$$\frac{\begin{array}{c}[\mathcal{X}, x \textit{ not free in the conclusion, } u \textit{ or } N]\\ \langle S, x \,\#\, N, u \doteq (\boldsymbol{\nu}x)\mathcal{X}\rangle\, \Gamma \vdash \Delta\end{array}}{\langle S\rangle\, \Gamma \vdash \Delta}\ (\text{И}) \qquad \frac{\begin{array}{c}m, n \,\#_S\, \mathit{fpv}(A)\\ \langle S\rangle\, \Gamma, (m\leftrightarrow n)u : \{m\leftrightarrow n\}\cdot A \vdash \Delta\end{array}}{\langle S\rangle\, \Gamma, u : A \vdash \Delta}\ (\tau)$$

$$\frac{\begin{array}{c}u \doteq_S (\boldsymbol{\nu}n)v\\ n \,\#_S\, \text{И}x.A\\ \langle S\rangle\, \Gamma, u : A\{x\leftarrow n\} \vdash \Delta\end{array}}{\langle S\rangle\, \Gamma, u : \text{И}x.A \vdash \Delta}\ (\text{И}\mathrm{L}) \qquad \frac{\begin{array}{c}u \doteq_S (\boldsymbol{\nu}n)v\\ n \,\#_S\, \text{И}x.A\\ \langle S\rangle\, \Gamma \vdash u : A\{x\leftarrow n\}, \Delta\end{array}}{\langle S\rangle\, \Gamma \vdash u : \text{И}x.A, \Delta}\ (\text{И}\mathrm{R})$$

(In (ИL) and (ИR), the formula $\text{И}x.A$ must verify the condition in Fig. 2)

Fig. 7. Freshness Rules

$$\frac{\langle S, u \doteq \mathbf{0}\rangle\, \Gamma \vdash \Delta \quad u|v \doteq_S \mathbf{0}}{\langle S\rangle\, \Gamma \vdash \Delta}\ (\mathrm{S}|\mathbf{0}) \quad \frac{\langle S, u \doteq \mathbf{0}\rangle\, \Gamma \vdash \Delta \quad (\boldsymbol{\nu}n)u \doteq_S \mathbf{0}}{\langle S\rangle\, \Gamma \vdash \Delta}\ (\mathrm{S}\boldsymbol{\nu}\mathbf{0})$$

$$\frac{\begin{array}{c}[\mathcal{X} \textit{ and } \mathcal{Y} \textit{ not free in the conclusion}]\\ \langle S, u \doteq \mathcal{X}|\mathcal{Y}, (\boldsymbol{\nu}n)\mathcal{X} \doteq t, (\boldsymbol{\nu}n)\mathcal{Y} \doteq v\rangle\, \Gamma \vdash \Delta \quad (\boldsymbol{\nu}n)u \doteq_S t|v\end{array}}{\langle S\rangle\, \Gamma \vdash \Delta}\ (\mathrm{S}\boldsymbol{\nu}|)$$

$$\frac{\begin{array}{c}[\mathcal{X}, \mathcal{X}', \mathcal{Y} \textit{ and } \mathcal{Y}' \textit{ not free in the conclusion}]\\ \langle S, u \doteq \mathcal{X}|\mathcal{X}', w \doteq \mathcal{Y}|\mathcal{Y}', t \doteq \mathcal{X}|\mathcal{Y}, v \doteq \mathcal{X}'|\mathcal{Y}'\rangle\, \Gamma \vdash \Delta \quad u|w \doteq_S t|v\end{array}}{\langle S\rangle\, \Gamma \vdash \Delta}\ (\mathrm{S}||)$$

$$\frac{\begin{array}{c}[\mathcal{X} \textit{ not free in the conclusion}]\\ \langle S, u \doteq (m\leftrightarrow n)v\rangle\, \Gamma \vdash \Delta\\ \langle S, u \doteq (\boldsymbol{\nu}m)\mathcal{X}, v \doteq (\boldsymbol{\nu}n)\mathcal{X}\rangle\, \Gamma \vdash \Delta \quad (\boldsymbol{\nu}n)u \doteq_S (\boldsymbol{\nu}m)v\end{array}}{\langle S\rangle\, \Gamma \vdash \Delta}\ (\mathrm{S}\boldsymbol{\nu}\boldsymbol{\nu})$$

$$\frac{\mathbf{0} \to_S u}{\langle S\rangle\, \Gamma \vdash \Delta}\ (\mathrm{S}\mathbf{0}\to) \quad \frac{\begin{array}{c}[\mathcal{X} \textit{ not free in the conclusion}]\\ \langle S, u \to \mathcal{X}, v \doteq (\boldsymbol{\nu}n)\mathcal{X}\rangle\, \Gamma \vdash \Delta \quad (\boldsymbol{\nu}n)u \to_S v\end{array}}{\langle S\rangle\, \Gamma \vdash \Delta}\ (\mathrm{S}\boldsymbol{\nu}\to)$$

Fig. 8. World Rules.

4 Basic Proof Theory

In this Section we develop some proof-theory for our logic, stating several admissible proof principles and a cut elimination result for the first-order fragment. Most of the presented proof principles are size-preserving, and instrumental to the proof of cut elimination. We introduce a measure for the *size* of a derivation, in which certain occurrences of the (τ) rule are not weighted. An occurrence of the inference rule (τ) in a derivation is *simple* if it applies either to an instance of (Id) or to another simple occurrence of (τ). We define the *size of a derivation* as the number of rule occurrences it contains, other than simple occurrences of rule (τ). We then assert $\vdash_n \langle S\rangle\, \Gamma \vdash \Delta$ to state that the given sequent has a derivation of size not exceeding n. We have the following useful admissible rules

Lemma 4.1 (Basic). *The following proof principles are admissible:*

$$\frac{[\ \Gamma \equiv_\alpha \Gamma' \text{ and } \Delta \equiv_\alpha \Delta'\ ] \quad \vdash_n \langle S\rangle\, \Gamma \vdash \Delta}{\vdash_n \langle S\rangle\, \Gamma' \vdash \Delta'}\ (\alpha) \qquad \frac{[\ \varphi, \varphi' \in \chi \text{ or } \varphi, \varphi' \in \mathcal{V},\ \varphi' \text{ not free in the premise}\ ] \quad \vdash_n \langle S\rangle\, \Gamma \vdash \Delta}{\vdash_n \langle S\{\varphi \leftarrow \varphi'\}\rangle\, \Gamma\{\varphi \leftarrow \varphi'\} \vdash \Delta\{\varphi \leftarrow \varphi'\}}\ \text{(Ren)}$$

$$\frac{\vdash_n \langle S\rangle\, \Gamma \vdash \Delta}{\vdash_n \langle S, S'\rangle\, \Gamma, \Gamma' \vdash \Delta, \Delta'}\ \text{(W)} \qquad \frac{\vdash_n \langle S\rangle\, \Gamma \vdash \Delta}{\vdash_n \langle S\{\mathcal{X} \leftarrow u\}\rangle\, \Gamma\{\mathcal{X} \leftarrow u\} \vdash \Delta\{\mathcal{X} \leftarrow u\}}\ (\text{Inst}\mathcal{I})$$

$$\frac{\vdash_n \langle S, c\rangle\, \Gamma \vdash \Delta \quad S \vdash c}{\vdash_n \langle S\rangle\, \Gamma \vdash \Delta}\ \text{(CS)} \qquad \frac{\vdash_n \langle S\rangle\, \Gamma \vdash \Delta}{\vdash_n \langle S\{x \leftarrow m\}\rangle\, \Gamma\{x \leftarrow m\} \vdash \Delta\{x \leftarrow m\}}\ (\text{Inst}\mathcal{N})$$

N.B. We write $\Delta \equiv_\alpha \Delta'$ if Δ' is obtained from Δ by α-converting some formulas in it.

Lemma 4.2 (Replacement and Instantiation). *The following rules are admissible*

$$\frac{[\ \mathcal{X} \text{ not free in } S\ ] \quad \langle S\rangle\, \mathcal{X} : A \vdash \mathcal{X} : B \quad \langle S\rangle\, \mathcal{X} : B \vdash \mathcal{X} : A}{\langle S\rangle\, \mathcal{Y} : C[A] \vdash \mathcal{Y} : C[B]} \qquad \frac{[\ X \text{ not free in the conclusion}\ ] \quad \langle S\rangle\, \Gamma \vdash \Delta}{\langle S\rangle\, \Gamma\{X \leftarrow A\} \vdash \Delta\{X \leftarrow A\}}$$

The first-order fragment of the spatial logic enjoys the cut elimination property, and therefore the subformula property. This result is a reasonable evidence that our addition of structural and freshness constraints to sequents and inference rules is rather canonical. For instance, cuts on spatial formulas are eliminated in a rather uniform way, by matching process eigenvariables (on one side) against the given witnesses (on the other), and then eliminating the remaining (redundant) structural constraints. The cut case for freshness quantifications is more interesting, since it relies on a proof transformation that can be interpreted as α-conversion in elements of the intended model (processes) *inside our logic* (to be distinguished from (α) in Lemma 4.1, which is about the syntax of the derivations). For this transformation to go through, explicit transpositions needed to be included both in the π-algebra and in the syntax of terms in formulas.

Theorem 4.3 (Cut Elimination). *If a sequent has a first-order derivation then it has a derivation without instances of the* (Cut) *rule.*

Full proofs of Theorem 4.3 and related results can be found in [3], where a set of derived connectives and respective proof rules is also presented.

5 Examples

We now go through a sequence of short examples to show how our logic is applicable to reasoning about distributed concurrent systems. We are necessarily brief here, and show only very elementary examples, but most interesting logical operators are covered.

A Simple Property We show a simple derivation of the fact that $(A|B) \wedge \mathbf{0}$ entails A, meaning that if a process satisfies $(A|B) \wedge \mathbf{0}$ then it satisfies A. The intuition is that if a process P satisfies both $(A|B)$ and $\mathbf{0}$, then P is (structurally equivalent to) the $\mathbf{0}$ process, which is the same as $\mathbf{0}|\mathbf{0}$; so $\mathbf{0}$ satisfies A (and B). We conclude that P satisfies A. This derivation illustrates: a property combining spatial and propositional operators; the use of constraint manipulation; and the use of one of the world rules, namely, $(S|\mathbf{0})$ corresponding to the "zero law" of Aπ processes: if $P|Q \equiv \mathbf{0}$ then $P \equiv \mathbf{0}$.

5. $\langle S, u \doteq \mathcal{X}|\mathcal{Y}, u \doteq \mathbf{0}, \mathcal{X} \doteq \mathbf{0}\rangle\, \Gamma, \mathcal{X} : A, \mathcal{Y} : B \vdash u : A, \Delta$ (by (Id) since $u \doteq_S \mathcal{X}$)
4. $\langle S, u \doteq \mathcal{X}|\mathcal{Y}, u \doteq \mathbf{0}\rangle\, \Gamma, \mathcal{X} : A, \mathcal{Y} : B \vdash u : A, \Delta$ (by 5, (S$|\mathbf{0}$) since $\mathcal{X}|\mathcal{Y} \doteq_S \mathbf{0}$)
3. $\langle S, u \doteq \mathcal{X}|\mathcal{Y}\rangle\, \Gamma, \mathcal{X} : A, \mathcal{Y} : B, u : \mathbf{0} \vdash u : A, \Delta$ (by 4, ($\mathbf{0}$L))
2. $\langle S\rangle\, \Gamma, u : (A|B), u : \mathbf{0} \vdash u : A, \Delta$ (by 3, ($|$L))
1. $\langle S\rangle\, \Gamma, u : (A|B) \wedge \mathbf{0} \vdash u : A, \Delta$ (by 2, ($\wedge$L))

Note that the proof is fairly simple, particularly if conducted bottom up. Most constraints are generated from the goal by using all the applicable left rules, and the final constraint $\mathcal{X} \doteq \mathbf{0}$ is generated by closing up the constraint set under deduction, via (S$|\mathbf{0}$). Finally, (Id) involves a simple equivalence check in S. It is common for our derivations, when read bottom-up, to have this mechanical flavor.

Freshness We show a derivation of the fact that $\neg \text{И} x.A$ entails $\text{И} x.\neg A$. This (and its converse) is a well-known property of $\text{И} x.A$ [10]; the purpose here is to show the use of the rules for freshness in a simple case. We abbreviate by $y \# \text{И} x.A$ the set of constraints $y \# \mathit{lfv}(\text{И} x.A)$.

6. $\langle S, y \# \text{И} x.A, u \doteq (\boldsymbol{\nu} y)\mathcal{X}\rangle\, \Gamma, u : A\{x\leftarrow y\} \vdash u : A\{x\leftarrow y\}, \Delta$ (by (Id) choose $y, \mathcal{X}$ fresh)
5. $\langle S, y \# \text{И} x.A, u \doteq (\boldsymbol{\nu} y)\mathcal{X}\rangle\, \Gamma \vdash u : A\{x\leftarrow y\}, u : \neg A\{x\leftarrow y\}, \Delta$ (by 6, ($\neg$ R))
4. $\langle S, y \# \text{И} x.A, u \doteq (\boldsymbol{\nu} y)\mathcal{X}\rangle\, \Gamma \vdash u : \text{И} x.A, u : \neg A\{x\leftarrow y\}, \Delta$ (by 5, (ИR))
3. $\langle S, y \# \text{И} x.A, u \doteq (\boldsymbol{\nu} y)\mathcal{X}\rangle\, \Gamma \vdash u : \text{И} x.A, u : \text{И} x.\neg A, \Delta$ (by 4, (ИR))
2. $\langle S, y \# \text{И} x.A, u \doteq (\boldsymbol{\nu} y)\mathcal{X}\rangle\, \Gamma, u : \neg \text{И} x.A \vdash u : \text{И} x.\neg A, \Delta$ (by 3, ($\neg$ L))
1. $\langle S\rangle\, \Gamma, u : \neg \text{И} x.A \vdash u : \text{И} x.\neg A, \Delta$ (by 2, (И) $y, \mathcal{X}$ not in conclusion)

We start with $A\{x\leftarrow y\}$ for a fresh y, instead of simply with A, so that we can apply (И) in the last step even when x occurs free in Γ, Δ. It is usually the case that an application of rules (И L) or (И R) is followed by an application of rule (И), to clean up the constraints. Note, however, that having (И) decoupled from (ИL) and (ИR) allow us to apply, in this case, (ИR) twice before applying (И).

Along similar lines, we can derive interesting properties combining $\text{И} x.A$ with spatial operators, for example the following one, which is important for deriving properties of the hiding quantifier (it takes about eight steps in each direction, but with a rather more involved set of constraints):

$$\langle S\rangle\, \Gamma, u : (\text{И} x.A)|(\text{И} x.B) \dashv\vdash u : \text{И} x.(A|B), \Delta$$

This derivation uses the world rule (S$\nu|$), which embeds a rather deep lemma about π-calculus structural congruence; namely, that if $(\boldsymbol{\nu} n)P \equiv Q|R$ then there exist P', Q' such that $P \equiv P|P''$ and $(\boldsymbol{\nu} n)P' \equiv Q$ and $(\boldsymbol{\nu} n)P'' \equiv R$.

Equivariance In general, we have that a process P satisfies the formula $n®A$ if $P \equiv (\boldsymbol{\nu} n)Q$, where Q is a process that satisfies A. Then n is a name which is hidden, and hence not free, in P. So, the revelation operator has a useful meaning also in the special case $n®\mathbf{T}$: the process P satisfies $n®\mathbf{T}$ if and only if the name n is fresh in P (©n abbreviates $\neg n®A$). We can show than $A \wedge m®\mathbf{T} \wedge n®\mathbf{T}$ entails $\{m{\leftrightarrow}n\}{\cdot}A$:

3. $\langle \mathcal{Z} \doteq (\boldsymbol{\nu} n)\mathcal{X}, u \doteq (\boldsymbol{\nu} m)\mathcal{Y}\rangle\, (m \leftrightarrow n)\mathcal{Z} : \{m{\leftrightarrow}n\}{\cdot}A, \mathcal{X} : \mathbf{T}, \mathcal{Y} : \mathbf{T} \vdash \mathcal{Z} : \{m{\leftrightarrow}n\}{\cdot}A$ (Id)
2. $\langle \mathcal{Z} \doteq (\boldsymbol{\nu} n)\mathcal{X}, u \doteq (\boldsymbol{\nu} m)\mathcal{Y}\rangle\, \mathcal{Z} : A, \mathcal{X} : \mathbf{T}, \mathcal{Y} : \mathbf{T} \vdash \mathcal{Z} : \{m{\leftrightarrow}n\}{\cdot}A$ (by 3, (τ))
1. $\langle\rangle\, \mathcal{Z} : A \wedge m®\mathbf{T} \wedge n®\mathbf{T} \vdash u : \{m{\leftrightarrow}n\}{\cdot}A$ (by 2, ($\wedge$L and ®L))

(Note the use of (Swap Erase) in step 3) This property can be interpreted as saying that, for any process P, if it satisfies A, it also satisfies $\{m{\leftrightarrow}n\}{\cdot}A$ for any fresh names m and n. This fact is a consequence of the equivariance property of the semantics: intuitively, if the name denoted by (say) m occurs in the formula A but not in the process P, then we would expect the name m to be irrelevant to the fact that P satisfies A. This means that if we swap in formula A the name m by any other fresh name n, we would expect that P would still satisfy it (since a fresh name is as good as any other fresh name). For example, the following provable sequent

$$\langle n \# p, m \# p\rangle\, n®(p\langle n\rangle|\mathbf{T}) \wedge m®\mathbf{T} \wedge n®\mathbf{T} \vdash m®(p\langle m\rangle|\mathbf{T})$$

says that if a process is about to send a fresh name on a public channel p, it can send any other fresh name as well.

Input In our logic we have a primitive formula to observe messages, $n\langle m\rangle$, corresponding to the output operator of the asynchronous π-calculus. We do not have a corresponding input formula, but it can be expressed from output along the lines of [16]. The guarantee operator is crucial to this; recall that a process P satisfies $A \triangleright B$ if for any Q that satisfies A, we have that $P|Q$ satisfies B (this can be read out from ($\triangleright$R)). We say that P satisfies B "in presence" of any Q that satisfies A. We can take the following definition of input: $x(y).A \triangleq \forall y.x\langle y\rangle \triangleright \Diamond A$.

The intention is that a process satisfies the input specification $x(y).A$ if it performs an input over a given channel x of any name y (with y bound in A), and then satisfies the property A. The above definition says literally, that an input process is one that, in presence of any output message y over the given channel x, at the next step (after input) it behaves according to A.

It is then easy to verify that because of the adjunction between $|$ and $\triangleright$, input and output interact as expected in Aπ communication, that is, $x\langle z\rangle|x(y).A$ entails $\Diamond A\{y{\leftarrow}z\}$:

4.2. $\langle S, u = \mathcal{X}|\mathcal{Y}\rangle\, \Gamma, \mathcal{X} : x\langle z\rangle \vdash \mathcal{X} : x\langle z\rangle, u : \Diamond A\{y{\leftarrow}z\}, \Delta$ (by (Id) choose $\mathcal{X}, \mathcal{Y}$ fresh)
4.1. $\langle S, u = \mathcal{X}|\mathcal{Y}\rangle\, \Gamma, \mathcal{X} : x\langle z\rangle, \mathcal{X}|\mathcal{Y} : \Diamond A\{y{\leftarrow}z\} \vdash u : \Diamond A\{y{\leftarrow}z\}, \Delta$ (by (Id), for $\mathcal{X}|\mathcal{Y} \doteq_S u$)
3. $\langle S, u = \mathcal{X}|\mathcal{Y}\rangle\, \Gamma, \mathcal{X} : x\langle z\rangle, \mathcal{Y} : x\langle z\rangle \triangleright \Diamond A\{y{\leftarrow}z\} \vdash u : \Diamond A\{y{\leftarrow}z\}, \Delta$ (by 4.1, 4.2, ($\triangleright$L))
2. $\langle S, u = \mathcal{X}|\mathcal{Y}\rangle\, \Gamma, \mathcal{X} : x\langle z\rangle, \mathcal{Y} : \forall y.x\langle y\rangle \triangleright \Diamond A \vdash u : \Diamond A\{y{\leftarrow}z\}, \Delta$ (by 3, ($\forall$L))
1. $\langle S\rangle\, \Gamma, u : x\langle z\rangle|(\forall y.x\langle y\rangle \triangleright \Diamond A) \vdash u : \Diamond A\{y{\leftarrow}z\}, \Delta$ (by 2, ($|$L) $\mathcal{X}, \mathcal{Y}$ not in conclusion)

Hiding In Part I we defined a hiding quantifier: $\mathsf{H}x.A \triangleq \mathsf{И}x.x®A$ which is related to π-calculus name restriction in an appropriate way; namely, that if process P satisfies formula $A\{x{\leftarrow}n\}$, then $(\boldsymbol{\nu}n)P$ satisfies $\mathsf{H}x.A$. An interesting use of $\mathsf{H}x.A$ is in specifying "nonce generators", that is processes that output freshly generated names on a given channel. In π-calculus, a nonce generator can be written simply as $(\boldsymbol{\nu}n)nc\langle n\rangle$, for a given channel nc. A nonce generator over nc can then be specified by the following formula: $\mathcal{N}c \triangleq \mathsf{H}x.nc\langle x\rangle$. We can show that, when a nonce generator interacts with an input, the result is the acquisition of a private name:

$$\langle S\rangle\, \Gamma, u : \mathcal{N}c|nc(y).A \vdash u : \Diamond \mathsf{H}z.A\{y{\leftarrow}z\}, \Delta$$

Before input we have a nonce generator $\mathcal{N}c$ separate from the input process. After one step, we have that the A part has acquired a name z; but noticeably this z is "hidden" within $A\{y{\leftarrow}z\}$ by the scope of the hiding quantifier. Hence the A part of the system has acquired, from the nonce generator, a private name not shared with other parts of the system (at least, not yet).

Inductive Definitions We just sketch here our treatment of recursive formulas. First, we can combine the spatial operator $\triangleright$ with classical negation to obtain an operator $!A \triangleq (A \Rightarrow \mathbf{F}) \triangleright \mathbf{F}$ that has the meaning that A is valid (is satisfied by any process). Through second-order quantification, we can then define least and greatest fixpoint operators in a style similar to F-algebraic encodings.

$$\begin{aligned} \mu X.A &\triangleq \forall Y.(!(A\{X{\leftarrow}Y\} \Rightarrow Y)) \Rightarrow Y \quad \textit{(where } Y \textit{ is not free in } A\textit{)} \\ \nu X.A &\triangleq \neg \mu X.\neg A\{X{\leftarrow}\neg X\} \end{aligned}$$

These definitions turn out to enjoy the expected properties of recursive formulas, in the form of derivable left and right rules; for example the derivable rule (νR) corresponds to a coinduction principle. The folding and unfolding of $\mu X.A$ and $\nu X.A$ can be derived under an assumption of monotonicity of $A\{X\}$; this can be expressed via the ! operator. As an example of the use of recursion, we can specify a recursive nonce generator (a process producing an unbounded number of fresh names) as follows: $\mathcal{UN}c \triangleq \nu X.\mathcal{N}c|X$. By a standard coinductive argument, we can then show that $\mathcal{UN}c|\mathcal{UN}c$ entails $\mathcal{UN}c$. This is simple but significant: it means that (without any knowledge of the π-calculus implementation) two recursive nonce generators running in parallel behave like a single recursive nonce generator; in particular, the two generators do not risk generating independently the same name twice.

6 Conclusion

We have presented a sequent calculus that has a direct interpretation in terms of distributed concurrent behaviors, including notions of resource hiding.

We believe we have obtained a unique combination of, on one hand, good proof-theoretical structures and properties, and, on the other hand, direct applicability to concurrency. These twin aims have driven us towards a "many worlds" formulation of modal sequents that has been able to accommodate a wide range of unusual but strongly motivated logical constructions.

Acknowledgements Andy Gordon contributed to early stages of Part I of this paper. Thanks also to Peter O'Hearn for early discussions on substructural logics. Caires acknowledges Microsoft Research and Profundis IST200133100.

References

1. L. Caires. *A Model for Declarative Programming and Specification with Concurrency and Mobility*. PhD thesis, Dept. de Informática, FCT, Universidade Nova de Lisboa, 1999.
2. 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.
3. L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part II). Technical Report 3/2002/DI/PLM/FCTUNL, DI/PLM FCT Universidade Nova de Lisboa, 2002.
4. L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part I). *Information and Computation*, to appear.
5. 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.
6. C. Calcagno, Luca Cardelli, and Andrew Gordon. Deciding Validity in a Spatial Logic of Trees. to appear, 2002.
7. 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.
8. 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.
9. 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.
10. 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.
11. M. Hennessy and R. Milner. Algebraic laws for Nondeterminism and Concurrency. *JACM*, 32(1):137–161, 1985.
12. P. O'Hearn and D. Pym. The Logic of Bunched Implications. *The Bulletin of Symbolic Logic*, 5(2):215–243, 1999.
13. 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.
14. G. Plotkin and M. Abadi. A logic for parametric polymorphism. In M. Bezem and J. F. Groote, editors, *International Conference on Typed Lambda Calculi and Applications*, number 664 in Lecture Notes in Computer Science, pages 361–375, Utrecht, The Netherlands, March 1993. Springer-Verlag. TLCA'93.
15. 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.
16. D. Sangiorgi. Extensionality and Intensionality of the Ambient Logics. In *28th Annual Symposium on Principles of Programming Languages*, pages 4–13. ACM, 2001.
17. A. Simpson. *The Proof Theory and Semantics of Intuitionistic Modal Logic*. Ph.D. thesis, Dept. of Computer Science, Edingburgh University, 1994.
18. Luca Viganò. *Labelled Non-Classical Logics*. Kluwer Academic Publishers, Dordrecht, 2000.