Dynamic Spatial Logics: A Tutorial Survey

Luís Caires *

Abstract

Spatial logics for concurrency have been proposed with the aim of reasoning about distribution, resource usage, mobility, and other structural aspects of general computing systems. This tutorial aims to introduce and motivate the main intuitions behind the development of dynamic spatial logics, and then present some selected results and applications.

1 Introduction

Spatial logics for concurrency have been proposed with the aim of reasoning about distribution, resource usage, mobility, and other structural aspects of general computing systems. Some of these aspects are not particular to concurrency, but have been emerging in several forms since the early days of computing science, as witnessed by the progressive move of concern from purely functional models of computation to increasingly sophisticated theories aiming to tackle concepts such as state, aliasing, effects, sequentiality, resource usage, concurrency, interaction, distribution, and mobility. These fundamental developments, motivated not only by theoretical questions, but also by the permanent challenge posed by computing technology, have then contributed to wide the focus in concurrency research from centralized concurrent systems to distributed systems, in a broad sense.

While centralized processes may be accurately modeled as pure objects of behavior, in a distributed system many interesting phenomena besides pure interaction, such as location dependent behavior, partial failure, resource usage and competition are also of central interest, and must be internalized in the models. Most of these phenomena are related to notions of computation in some kind of space; intuitions about the essential role of space in computation are certainly not new, and may already be found in pioneering work on concurrency and state by Hoare, Milner, Reynolds, and others.

We find that many familiar properties of distributed systems may be understood in spatial terms. Simple examples include *connectivity*, stating that there is

*CITI / Departamento de Informática, Universidade Nova de Lisboa, Portugal

always an access route between two different sites; *unique handling*, stating that there is at most one server process listening on a given channel, *resource availability*, stating that a bound exists on the number of channels that can be allocated in a given location; or *race freedom*, stating that no competing requests from different clients will ever arise for the same server. At least certain security properties may be also understood in spatial terms. For example *secrecy*: a secret is a piece of data whose knowledge of is restricted to some parts of a system, but unforgeable by other parts. Reasoning about more classical properties of concurrent processes also seems to frequently require some sort of spatial reasoning. Examples are local deadlock freedom, absence of arity mismatch errors in communication, and safety in resource usage. Properties such as these have been addressed by type systems and are certainly important. However, they are not invariant under usual bisimilarities, and thus cannot be expressed by purely behavioral logics.

As traditional logics for concurrent systems have been introduced to characterize pure notions of behavior (modeled by transition systems, or sets of traces), dynamic spatial logics have been introduced with the aim to characterize the space-time dynamics of systems with some relevant spatial structure.

Many well-known models of computation have been proposed with the intent of modeling several kinds of concurrent, mobile, and distributed systems. Such models have been usually formalized by process calculi, where some of the process operators are to be understood as spatial constructors, that assemble subsystems into larger systems, rather than as behavioral operators, intended to define the control flow of processes. Many kinds of spatial structures are conceivable, at least in principle. For example, we may consider space to be modeled as a multiset of threads or of heap cells, as in the Berry and Boudol's CHAM model and in the structural congruence-based presentation of π-calculi, as a hierarchically nested collection of locations, sites, or ambients, as in Cardelli and Gordon's Mobile Ambients, or as some generalized graph structure, as in graph rewriting or as in Milner's bigraphical models. In general, each set of spatial constructors may give rise to appropriate spatial observations on systems. Such observations may then be captured by various logical connectives, enabling a spatial logic to talk about the dynamic interplay of behavior, interaction, and space in computing systems.

An important ingredient of spatial logics, also highlighted in other substructural logics, is their resource awareness, in the sense that spatial operators are able to separate, count, and identify local resources; this fact is frequently understood as adding an intensional character to these logics. On the contrary, we believe that in contexts where the underlying spatial models are well understood and extensionally characterized, the approach of dynamic spatial logics may provide a general basis for reasoning about general physical or virtual interactive systems.

This tutorial aims to introduce and motivate the main intuitions behind the development of dynamic spatial logics, and then present some selected results

and applications. We start by reviewing some basic notions about the π-calculus, which is our reference model of concurrency for the purposes of this note.

2 The π-Calculus Concurrency Model

We briefly introduce the syntax and semantics of the reference process model chosen for our presentation: a fragment of the monadic π-calculus introduced by Milner, Parrow and Walker [58] (see also [55, 57, 73]). The π-calculus is a concurrent process algebra able to express dynamic creation of resources and access mobility, based on the manipulation of communication channels as first-class entities. The π-calculus is a foundational calculus [66] for concurrency, expressive enough to capture most other proposed computational models. For example, unlike Hoare's CSP [44] and Milner's CCS [54], the π-calculus may express the λ-calculus [56], and concurrent-object oriented programming [77].

Given infinite sets Λ of *names* (m, n, p), and χ of *process variables* $(\mathcal{X}, \mathcal{Y})$ the set $\mathcal{P}$ of *processes* (P, Q, R) is defined as depicted in Figure 1. In restriction $(\nu n)P$ and input $m(n).P$ the distinguished occurrence of name n is binding, with scope the process P. We denote by $\equiv_\alpha$ the relation of α-equivalence on processes: we will implicitly consider processes up to α-equivalence, with care. For any process P, we assume defined as usual the set $fn(P)$ of *free names* of P. By $\{^m/_n\}$ (resp. $\{^{\mathcal{X}}/_Q\}$) we denote the safe substitution of m by n (resp. of $\mathcal{X}$ by Q), and by $\{m \leftrightarrow n\}$ the safe transposition of m and n.

Structural congruence expresses basic identities on the spatial structure of processes. *Structural congruence* $(P \equiv Q)$ is the least congruence relation on processes, defined in Figure 2. The dynamic behavior of processes is defined by a relation of reduction, that captures the computations that a process may perform by itself. *Reduction* $(P \rightarrow Q)$ is the least relation on processes defined in Figure 3. We denote by $\Rightarrow$ the reflexive-transitive closure of $\rightarrow$.

We present a simple example of a π-calculus model: the Bouncer system.

$$
\begin{array}{lcl}
Wr(c, s) & \triangleq & c\langle s\rangle.Rd(c) \\
Rd(c) & \triangleq & \mathtt{rec}\ Loop.c(x).c\langle x\rangle.Loop \\
Bouncer & \triangleq & (\nu secret)(\nu ch)(Wr(ch, secret) \mid Rd(ch))
\end{array}
$$

The system *Bouncer* is composed by two separate interacting processes, namely $Wr(c, s)$ and $Rd(c)$, that bounce a private name *secret* on a private channel *ch*.

2.1 Behavioral Semantics

The reduction semantics is enough to characterize the π-calculus as an abstract machine. However, to define a proper compositional semantics of the process con-

$$P, Q \quad ::= \quad \mathbf{0} \;\mid\; m(n).P \;\mid\; m\langle n\rangle.P \;\mid\; P \mid Q \;\mid\; (\nu n)P \;\mid\; \mathcal{X} \;\mid\; \mathtt{rec}\, \mathcal{X}.P$$

Figure 1: Processes.

$P \mid \mathbf{0} \equiv P$	(*Struct Par Void*)
$P \mid Q \equiv Q \mid P$	(*Struct Par Comm*)
$P \mid (Q \mid R) \equiv (P \mid Q) \mid R$	(*Struct Par Assoc*)
$n \notin fn(P) \Rightarrow P \mid (\nu n)Q \equiv (\nu n)(P \mid Q)$	(*Struct Res Par*)
$(\nu n)\mathbf{0} \equiv \mathbf{0}$	(*Struct Res Void*)
$(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$	(*Struct Res Comm*)
$\mathtt{rec}\, \mathcal{X}.P \equiv P\{^{\mathcal{X}}/_{\mathtt{rec}\, \mathcal{X}.P}\}$	(*Struct Unfold*)

Figure 2: Structural Congruence.

$m\langle n\rangle.Q \mid m(p).P \rightarrow Q \mid P\{^{p}/_{n}\}$	(*Red React*)
$Q \rightarrow Q' \Rightarrow P \mid Q \rightarrow P \mid Q'$	(*Red Par*)
$P \rightarrow Q \Rightarrow (\nu n)P \rightarrow (\nu n)Q$	(*Red Res*)
$P \equiv P',\ P' \rightarrow Q',\ Q' \equiv Q \Rightarrow P \rightarrow Q$	(*Red Struct*)

Figure 3: Reduction.

$$a(n).P \xrightarrow{a(n)} P\ (In) \qquad m\langle n\rangle.P \xrightarrow{m\langle n\rangle} P\ (Out)$$

$$\frac{P \xrightarrow{\alpha} Q}{P \mid R \xrightarrow{\alpha} Q \mid R}\ (Par) \qquad \frac{P \xrightarrow{\alpha} Q \quad n \notin fn(\alpha)}{(\nu n)P \xrightarrow{\alpha} (\nu n)Q}\ (Res) \qquad \frac{P\{^{\mathcal{X}}/_{\mathtt{rec}\, \mathcal{X}.P}\} \xrightarrow{\alpha} Q}{\mathtt{rec}\, \mathcal{X}.P \xrightarrow{\alpha} Q}\ (Rec)$$

$$\frac{P \xrightarrow{(\nu \tilde{s})n\langle m\rangle} P' \quad Q \xrightarrow{n(p)} Q'}{P \mid Q \xrightarrow{\tau} (\nu \tilde{s})(P' \mid Q'\{^{p}/_{m}\})}\ (Com) \qquad \frac{P \xrightarrow{m\langle n\rangle} Q}{(\nu n)P \xrightarrow{(\nu n)m\langle n\rangle} Q}\ (Open)$$

Figure 4: Labeled transition system.

structors, there is the need to introduce some notion of contextual or observational equivalence relation, usually in the form of bisimilarities [65, 54]. Bisimilarity is defined in terms of the processes' observable actions, in turn modeled by labels of a labeled transition system. In our case, the set of labels $\mathcal{L}(\alpha,\beta)$ given by

$$\alpha,\beta \quad ::= \quad (\nu n)\alpha \;\mid\; m\langle n\rangle \;\mid\; m(n) \;\mid\; \tau$$

Name restriction (νn) on a label $(\nu n)\alpha$ expresses bound output, that is, extrusion of a fresh name n to the environment (see [73]). The sets of $fn(\alpha)$ (free names) and $bn(\alpha)$ (bound names) of a label α are defined as usual.

Familiar characterizations of behavioral equivalence for the π-calculus are early bisimilarity and late bisimilarity. The standard (late) labeled transition system (see [73]) is presented in Figure 4, and specifies the relation *late labeled transition* ($P \xrightarrow{\alpha} Q$). To avoid clashes between fresh names and free names, the following provisos apply: rule (*Par*) subject to $fn(Q)\#bn(\alpha)$, rule (*Com*) subject to $p, \overline{s}\#fn(Q)$, rule (*Open*) subject to $n \neq m$ (we write $S\#U$ to say that the sets S and U are disjoint).

Notice that reduction $\rightarrow$ coincides with silent transition $\xrightarrow{\tau}$, and does not increase the set of free names of processes.

Strong late bisimilarity over the labeled transition system just defined may be taken as our reference behavioral equivalence of processes. A *strong late bisimulation* $\mathcal{R}$ is a symmetric binary relation over processes such that for all P, Q

- If $P\ \mathcal{R}\ Q$ and $P \xrightarrow{\alpha} P'$ for some P' and α is not an input, then there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $P'\ \mathcal{R}\ Q'$.
- If $P\ \mathcal{R}\ Q$ and $P \xrightarrow{n(p)} P'$ for some P' and $p\#fn(Q)$ then there exists Q' such that $Q \xrightarrow{n(p)} Q'$ and, for all m, $P'\{^p/_m\}\ \mathcal{R}\ Q'\{^p/_m\}$.

Strong late bisimilarity, written $\sim_L$, is the greatest strong late bisimulation. The intuition behind strong late bisimilarity is that the continuations of bisimilar input processes should denote the same function of the input name, hence the clause for input, cf. extensional equality of functions. A related notion of bisimilarity (with a simpler definition) results from allowing processes to instantiate input parameters "earlier" in the bisimulation game. To capture this alternative notion of bisimilarity, we must introduce different rules for input and communication:

$$a(p).P \xrightarrow{a(m)}_e P\{^p/_m\} \quad (InE) \qquad\qquad \frac{P \xrightarrow{(\nu \overline{s})n\langle m\rangle}_e P' \quad Q \xrightarrow{n(m)}_e Q'}{P \mid Q \xrightarrow{\tau}_e (\nu \overline{s})(P' \mid Q')}\ (ComE)$$

The relation of *early labeled transition* ($P \xrightarrow{\alpha}_e Q$) is defined by the rules in Figure 4, after replacing rule (*In*) by (*InE*) and (*Com*) by (*ComE*). A *strong early bisimulation* $\mathcal{R}$ is a symmetric binary relation over processes such that for all P, Q

- If $P \mathcal{R} Q$ and $P \overset{\alpha}{\rightarrow}_e P'$ for some P' and $bn(\alpha)\#fn(Q)$ then there exists Q' such that $Q \overset{\alpha}{\rightarrow}_e Q'$ and $P' \mathcal{R} Q'$.

Strong early bisimilarity, written $\sim_E$, is the greatest strong early bisimulation.

Bisimilarities have been defined with the intent of capturing undistinguishability of processes by certain contexts, and of giving a compositional semantics to the process operators (*e.g.*, bisimilarities should be congruences). Indeed, the relations $\sim_E$ and $\sim_L$ are congruences for all operators, if for the case of input processes we consider the higher-order congruence principle: if $P\{^x/_n\} \sim Q\{^x/_n\}$ for all n, then $a(x).P \sim a(x).Q$ (cf. [73] Theorem 2.2.8(2)). For an example of undistinguishability under bisimilarity, notice that an external observer only able to detect activity on publicly open interaction channels should not able to distinguish between the two following processes P_1 and P_2

$$P_1 \triangleq Bouncer \qquad P_2 \triangleq Bouncer \mid Bouncer$$

consisting of respectively one and two copies of the *Bouncer* system introduced above. Indeed, by constructing appropriate bisimulations we may show that in this example $P_1 \sim_E P_2$ and $P_1 \sim_L P_2$. In general, late bisimilarity is more discriminating than early bisimilarity. Tool support for π-calculus bisimilarity checking has been available for some time time now [74], allowing a user to automatically prove behavioral equivalences between (finite control) π-calculus models of concurrent systems. Many other important notions of observational equivalence have been proposed over the years. In any case, bisimilarity is usually considered the finer notion of observational equivalence one would like to consider on pure behaviors.

2.2 Behavioral Logics

Operational and behavioral specifications expressed by process terms are precise, in the sense that they usually have just one model. In order to carry out more general analyses of the expressive power of concurrent computation models, and also to support under-specification and abstract reasoning about process properties, a dual view of process specification by means of logical properties is then fundamental.

There are well-known characterizations of process behavior using modal logics, inspired on temporal logics: such logics are frequently variants of the logic initially introduced by Hennessy and Milner [40]. They are essentially modal logics for labeled transition systems, that include, besides the standard propositional logic operators, certain "world" accessibility modalities motivated by the labels of the given labeled transition system. The syntax and semantics of a simple behavioral π-calculus logic is shown in Figure 5, along the lines of Milner-Parrow-

$$
\begin{array}{lcl}
P \models \mathsf{T} & \triangleq & \textit{always} \\
P \models \neg A & \triangleq & \textit{not } P \models A \\
P \models A \wedge B & \triangleq & P \models A \textit{ and } P \models B \\
P \models \langle \alpha \rangle A & \triangleq & \textit{Some } Q \textit{ . } P \xrightarrow{\alpha} Q \textit{ and } Q \models A \\
P \models \langle x(y) \rangle^{L} A & \triangleq & \textit{Some } Q \textit{ . } P \xrightarrow{x(y)} Q \textit{ and All } m.\ Q\{^{y}/_{m}\} \models A\{^{y}/_{m}\}
\end{array}
$$

Figure 5: A Core Behavioral Logic (CBL).

Walker [59]; of course, other logical operators may be defined by abbreviation as expected, including the "necessity" version of the modal operators.

A main motivation for these kind of logics is the intent to specify processes up to bisimilarity, in other words, to specify pure behaviors.

Two processes P and Q are *logically equivalent* ($P =_{\mathcal{L}} Q$) for a logic $\mathcal{L}$ if they satisfy exactly the same formulas of $\mathcal{L}$.

For the strong late bisimilarity case, we may consider the logics of Milner-Parrow-Walker $\mathcal{LM}$ [59] including all operators above, interpreted over the late labeled transition system; for the strong early bisimilarity case, we may consider the logic $\mathcal{EM}$ [59], containing all operators except $\langle x(y) \rangle^{E} A$, and interpreted over the early labeled transition system. More precisely, we have [59]:

- $P =_{\mathcal{EM}} Q$ if and only if $P \sim_{E} Q$.
- $P =_{\mathcal{LM}} Q$ if and only if $P \sim_{L} Q$.

Several process logics obtained by combining variants of Hennessy-Milner and of Milner-Parrow-Walker logics with the modal μ-calculus (see [49]) were studied, and have motivated rich verification frameworks for mobile processes (see e.g., [34, 35]). For example, using co-induction (expressed by the greatest fixed point operator), we may express the property of liveness by the formula *Live*

$$
\textit{Live} \triangleq \mathbf{rec}\ X.(\langle \tau \rangle \mathsf{T} \wedge [\cdot]\, X)
$$

We have that $\textit{Bouncer} \models \textit{Live}$, since $\textit{Bouncer} \models \langle \tau \rangle \mathsf{T}$ and for all α, P if $\textit{Bouncer} \xrightarrow{\alpha} P$ then $P = \textit{Bouncer}$. Other particularities frequently relate to the way name binding in action modalities and fresh names are dealt with, to the introduction of parameters in recursion formulas, and of quantifiers. Tool support for checking CCS processes against rich μ-calculi based process logics has been available for quite a while, the same does not seem to have happen for π-calculi until recently [75, 76] (even if the MBW [74] already included a model-checker for a π-calculus logic).

3 Behavioral-Spatial Logics

If one wants to extend the scope of our study from monolithic concurrent systems towards distributed systems, or if one wants to describe properties related to resource usage in general, we may realize that purely behavioral specifications do not provide the adequate degree of expressiveness. Such a progressive shift of focus from notions of "pure computation" to "computations in resource spaces" has lead to the introduction of several spatial and resourceful models of computation, where many aspects traditionally considered as intensional are taken into account. Examples of such a situation may have included for a long time imperative features and effects of evaluation order in functional languages, but nowadays involve phenomena such as spatial distribution of behavior in global computers, where many non-behavioral aspects should not be abstracted away. Among those we may refer aspects related to topology (inside / outside, overlapping / separated), and aspects related to naming (private / secret, non-interference).

We may also realize that the kind of properties captured by most type systems for concurrent and mobile processes are not invariant under bisimilarity, and therefore cannot be expressed by logics that just rely on observing process actions. For example, we may want to specify and verify the property of deadlock absence. Notice that $Q \models \mathit{Live}$ does not really imply that Q is free from local deadlocks.

Purely behavioral logics such as $\mathcal{LM}$ and $\mathcal{EM}$ do not distinguish between bisimilar processes, and therefore do not distinguish between deadlock and termination, as do type systems for deadlock absence. Likewise, if one wants to talk about message protocols between partners interacting in a closed system, as for instance in a security protocol, an enhanced observational power, that must go beyond observing external actions, must be called for. As further example, if one needs to express correctness properties of mobile distributed systems, we need to refer to notions of concrete or abstract locations inside the system under consideration. Considerations of this kind have then motivated the introduction and development by several authors of dynamic spatial logics for concurrent and mobile computing systems [16, 3, 4, 26, 10, 11, 12, 13, 5].

Behavioral logics explore the dynamic structure of concurrent systems (a labeled transition system) in order to separate and specify models of behavior. The dynamic structure of systems is observed through modalities that navigate the label-indexed accessibility relation between states (worlds). Then systems are essentially modeled as action labeled trees, and a process operator such as parallel composition is understood as a mapping from trees to trees (see Figure 6 (a)). On the other hand, dynamic spatial logics have as their intended models certain structures where the states of the system (worlds), besides evolving in time, have also an internal spatial dimension, that may be inspected by spatial observations.

Many different kinds of spatial structures are conceivable. In this presenta-

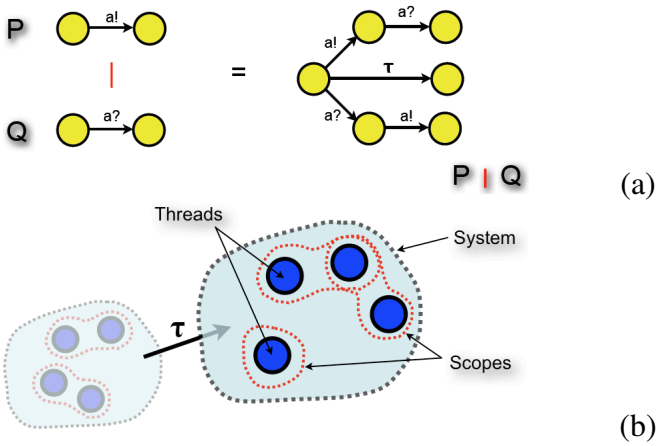

Figure 6: Behavioral models versus Spatial models.

tion, we consider the simplest spatial structure of π-calculus systems induced by interpreting the static constructors of the process calculus as system assembly operators, rather than as pure behavioral operators. In such a spatial model (see Figure 6 (b)), a process is seen as a multiset of "threads"; threads may be bound in groups by hidden links, corresponding to the presence of restricted names (a direct semantics for a of spatial structure of this kind was studied in [38], with the intention of providing a model for structural congruence).

A spatial logic then adds to a given set of behavioral modalities a set of spatial operators closely related to the static operators of the process calculus [16, 3, 4, 12]. For nominal process calculi, the static operators are at least the composition $P \mid Q$, its identity element $\mathbf{0}$ (denoting the empty system), and the name restriction $(\nu n)P$. These process constructors give rise to the composition formula $A \mid B$, that holds of a process that can be separated into a process that satisfies formula A and a process that satisfies formula B, to the void formula $\mathbf{0}$, that holds of the void process, and to the hidden name quantifier $\mathsf{H}x.A$ that allows us to quantify over locally restricted channels.

$$
\begin{array}{lcl}
P \models \mathbf{0} & \triangleq & P \equiv \mathbf{0} \\
P \models A \mid B & \triangleq & \mathit{Exists}\ Q, R\ .\ P \equiv Q \mid R\ \mathit{and}\ Q \models A\ \mathit{and}\ R \models B \\
P \models \mathsf{H}x.A & \triangleq & \mathit{Exists}\ Q, n\ .\ P \equiv (\nu n)Q\ \mathit{and}\ n\#\mathit{fn}(A)\ \mathit{and}\ Q \models A\{^{x}/_{n}\}
\end{array}
$$

Adding these operators to the basic set propositional operators, behavioral modalities, a name occurrence predicate, and recursion, we obtain what might be considered a core spatial-behavioral logic (CSBL). We depict in Figure 7, the syntax and satisfaction relation of such a CSBL.

The definition of the satisfaction relation follows the same lines as the one al-

$$
\begin{array}{lcl}
P \models_v \mathsf{T} & \triangleq & \textit{always} \\
P \models_v \neg A & \triangleq & \textit{not } P \models A \\
P \models_v A \wedge B & \triangleq & P \models A \textit{ and } P \models B \\
P \models \mathbf{0} & \triangleq & P \equiv \mathbf{0} \\
P \models A \mid B & \triangleq & \textit{Exists } Q, R \,.\, P \equiv Q \mid R \textit{ and } Q \models A \textit{ and } R \models B \\
P \models \mathsf{H}x.A & \triangleq & \textit{Exists } Q, n \,.\, P \equiv (\boldsymbol{\nu} n)Q \textit{ and } n\#fn(A) \textit{ and } Q \models A\{^{x}/_{n}\} \\
P \models_v \forall x.A & \triangleq & \textit{All } n \,.\, P \models A\{^{x}/_{n}\} \\
P \models_v @n & \triangleq & n \in fn(P) \\
P \models_v \langle\alpha\rangle A & \triangleq & \textit{Some } Q \,.\, P \xrightarrow{\alpha} Q \textit{ and } Q \models A \\
P \models_v \boldsymbol{\nu} X.A & \triangleq & P \in GFix(\lambda S.\{Q \mid Q \models_{v(X/S)} A\}) \\
P \models_v X & \triangleq & P \in v(X)
\end{array}
$$

Figure 7: A Core Spatial-behavioral Logic (CSBL).

ready given above for CBL (Figure 5). However, notice that to interpret the propositional variables and recursion, the satisfaction predicate $\models_v$ is now parametric on a valuation (v) mapping propositional variables (X) into Psets (essentially, a Pset is a set processes subject to certain closure conditions [10, 12]).

A more substantial ingredient is the presence of structural (or spatial) congruence in the definition of logical satisfaction. In our context of discussion, we understood structural congruence as a formalization of the identity of the intended spatial structure of processes. In the same way as a behavioral model isolates certain kinds of behavioral observations by means of labels of a labeled transition system, that are then inspected in the satisfaction clauses for the behavioral modalities, so a spatial model should be a priori equipped with a well-defined notion of identity of spatial structure, that may then be inspected in the satisfaction clauses by the spatial modalities. This aspect is perhaps intriguing for those used to see structural congruence as either a fragment of bisimilarity identities, or as a mere technical device to simplify the presentation of the operational semantics. In any case, we believe that the understanding of the structural congruence equations as laws of identity of spatial structure is much closer to the original source of inspiration for structural congruence as acknowledged in [55, 56], which is the CHAM model of Berry and Boudol [2] ("instead of composing their behaviors, we consider agents as *molecules* directly reacting with each other within a solution"). Then, in our context, we expect spatial congruence to be presented as a decidable congruence (e.g., as in [38], or as in Figure 2).

By combining spatial and the behavioral operators, we may specify many interesting properties of distributed and resource based systems. We illustrate this claim by the following sequence of simple examples:

- Property A holds somewhere in the system: $?A$.

  We may specify the property $?A$ by the formula $A \mid \top$. Then $P \models ?A$ if and only if there exists a decomposition $R \mid Q$ of the process P ($P \equiv R \mid Q$) such that $R \models A$. Notice that decomposition is taken up to spatial congruence, so the selected part R may result from grouping several subsystems of P (or none). For example, it is always the case that $P \models ?\mathbf{0}$, for any P.

- Property A holds everywhere in the system: $!A$.

  We may specify the property $!A$ by $\neg(\top \mid \neg A)$. Then $P \models !A$ if and only if for all decompositions $R \mid Q$ of the process P we have that $R \models A$.

- Process has precisely one component, cf., a single thread, site, etc.: $\mathbf{1}$.

  We may specify the property $\mathbf{1}$ by $\neg\mathbf{0} \wedge \neg(\neg\mathbf{0} \mid \neg\mathbf{0})$. Then we may express with $?(\mathbf{1} \wedge A)$ that a system has some site that satisfies A. Likewise, we may express with $!(\mathbf{1} \Rightarrow A)$ that all sites in the system satisfies A. It is then quite clear how more general arithmetical constraints on the number of components of a system may be expressed in the logic; connections between the expressiveness of static fragments of spatial logics and Presburger arithmetic has been pointed out by Dal-Zilio and Lugiez in [32].

- Spatial Dynamics

  We may specify how the structure of a system evolves in time, as an effect of the behavior of its subsystems. Consider the formula $(\neg\mathbf{0} \mid \neg\mathbf{0}) \wedge \langle\tau\rangle\,\mathbf{1}$. This formula asserts that the system has (at least) two separate non-trivial components, but after a reduction step, it becomes a single component. A possible model for this formula is the process

  $$\mathit{Mate} \triangleq s(x).x\langle\rangle.\mathbf{0} \mid (\nu n)s\langle n\rangle.n\langle\rangle.\mathbf{0}$$

  In fact, we have $\mathit{Mate} \xrightarrow{\tau} P'$, where $P' \equiv (\nu n)(n\langle\rangle.\mathbf{0} \mid n\langle\rangle.\mathbf{0})$, and $P' \models \mathbf{1}$. We may also check that although P' has a single component in the spatial model, this single component is actually composed by two separate subsystems, glued together by a private resource (an hidden channel). We may certify this property by checking that $P' \models \mathsf{H}x.(@x \mid @x)$. We may also see that $P' \models \neg\mathbf{0}$ and $P' \models \neg\langle\tau\rangle\top$, while $P' \sim \mathbf{0}$. Hence a spatial logic such as CSBL distinguishes between deadlocked and terminated processes.

- Hidden names

  We may state that a process has an occurrence of an hidden name x satisfying property $P(-)$ by the formula $\mathsf{H}x.(@x \wedge P(x))$. For example, in the *Bouncer* system we have $\mathit{Bouncer} \models \mathsf{H}x.(\forall y.\,\langle x(y)\rangle\mathsf{T})$. This means that *Bouncer* has a subsystem that is attempting to read on a private channel.

On the other hand, if $P \models \mathsf{H}x.(\neg @x \wedge P(x))$, then $P \equiv (\boldsymbol{\nu} n)Q$ and $Q \models P(n)$, but $n \notin fn(Q)$, so actually $P \equiv Q$ (recall that the structural congruence identity $P \equiv (\boldsymbol{\nu} n)P$ holds whenever $n \notin fn(P)$). We then have that $P \models \mathsf{H}x.(\neg @x \wedge P(x))$ if and only if $P \models \mathcal{N}x.P(x)$ where $\mathcal{N}x.A$ is a modal version of the Gabbay-Pitts freshness quantifier [68, 39].

We may express that a process has no "real" restricted name by the formula $Public \triangleq \mathsf{H}x.\neg @x$. Using recursion, we can then state that the subsystem obtained after removing all hidden names satisfies property A by the formula

$$inside(A) \triangleq \boldsymbol{\nu} X.((Public \wedge A) \vee \mathsf{H}x.(@x \wedge X))$$

For example, we have $Mate \models \langle \tau \rangle\, inside(\neg \mathbf{0} \mid \neg \mathbf{0})$.

- Action modalities

  Using the quantifiers and recursion, we may express general action modalities, in the style of temporal logics and the modal μ-calculus, e.g.,

$$\begin{array}{lcl}
[\alpha]\,A & \triangleq & \neg\,\langle \alpha \rangle\,\neg A \\
\langle x(-) \rangle\,A & \triangleq & \exists y.\,\langle x(y) \rangle\,A \\
\langle x\langle - \rangle \rangle\,A & \triangleq & \exists y.\,\langle x\langle y \rangle \rangle\,A \\
[\mathtt{out}]\,A & \triangleq & \forall x.\forall y.\,[x\langle y \rangle]\,A \\
[\mathtt{in}]\,A & \triangleq & \forall x.\forall y.\,[x(y)]\,A \\
[\star]\,A & \triangleq & [\tau]\,A \wedge [\mathtt{in}]\,A \wedge [\mathtt{out}]\,A \\
\Box A & \triangleq & (\boldsymbol{\nu} X)(A \wedge [\star]\,X)
\end{array}$$

- Resource control

  We may express that a process enjoys the unique handling property by the formula $\Box\, inside(\neg \exists x\,.\,(\langle x(-) \rangle\,\mathsf{T} \mid \langle x(-) \rangle\,\mathsf{T}))$.

  We may express that a process is race free, in the sense that it will never be the case that two subsystems attempt to simultaneously write to a same reader by $\Box\, inside(\neg \exists x\,.\,(\langle x\langle - \rangle \rangle\,\mathsf{T} \mid \langle x\langle - \rangle \rangle\,\mathsf{T} \mid \langle x(-) \rangle\,\mathsf{T}))$.

- Security

  We may express that a process enjoys a secrecy property, in the sense that it will never leak private names on public channels by the formula $Secrecy \triangleq \Box\neg \exists p.\mathsf{H}x.\,\langle p\langle x \rangle \rangle\,\mathsf{T}$. For example, getting back to the *Bouncer* example, we may verify that $Bouncer \models Secrecy$.

  We may also check other interesting properties of *Bouncer*, for example, $System \models \Box \mathsf{H}x.((\mathbf{1} \wedge @x) \mid (\mathbf{1} \wedge \neg @x))$. This means that all future states of *Bouncer* are always composed by two separate parts, one part that holds a secret x, and other part that does not know about it.

4 A Spatial Logic Framework

It is well-known that minimal behavioral observations, leading to the definition of labeled transition systems with good properties, are not easy to define for every process calculi, in particular for spatial process calculi such as the Ambient Calculus [25]. Still, general techniques have been developed to derive transition systems labels as minimal contextual tests defined from a reduction semantics, following the barbed congruence approach to observational equivalence (see [50]). More generally, for specification purposes, the introduction of arbitrary contextual tests provides a more general approach to behavioral specifications than the observation of elementary actions; contextual tests provide uniform approach to observational equivalences, easily applicable to any process calculus equipped with a reduction semantics.

A logical primitive particularly adequate for expressing contextual tests, written $A \triangleright B$, was introduced in the Ambient Logic [26], a spatial logic for the Ambient Calculus, and shown to be the adjunct of the composition operator. The primitive was also coined "guarantee" in [10], because it allows us to specify context dependent properties in a way not far from those introduced rely-guarantee proof method [47]; a related logical operator was already proposed by Dam [33], with the aim of achieving a form of compositionality in specifications. It turns out that the derivation of behavioral observations, such as the HML like modalities present in CSBL, may be expressed as certain kinds of contextual tests using the guarantee operator, so the two approaches are not incompatible.

The semantics of the guarantee operator is given by

$$P \models_v A \triangleright B \quad \triangleq \quad \mathit{All}\ Q.\ Q \models_v A\ \mathit{implies}\ P \mid Q \models_v B$$

The logical adjunction is given by $A \models B \triangleright C$ if and only if $A \mid B \models C$, where by $A \models B$ we denote the semantic entailment relation between formulas.

The spatial logic for concurrency (SLC) introduced in [10, 12, 11, 13] thus explores the purely spatial approach. The SLC, depicted in Figure 8, includes a single dynamic observation primitive ($\Diamond A$), interpreted over the reduction system (so the only "observable" label is τ). It also includes several spatial operators, universal and freshness name quantifiers, and recursive definitions (which, for technical convenience, are definable from second-order quantification [12]). The intended model of SLC is the asynchronous π-calculus. In order to observe the primitive action capabilities of processes, a primitive message observation predicate $n\langle m\rangle$ is included in the logic; this primitive may be used to easily define elementary observation contexts. For example, modalities for testing the input and output behavior of processes, in the style of the input clause for asynchronous bisimilarities [1], may be rendered by

$$n(m)A \triangleq n\langle m\rangle \triangleright A \qquad n\langle m\rangle A \triangleq n\langle m\rangle \mid A$$

m, n, p	::=		Name Terms
		x	*Name variable*
		$(m \leftrightarrow n)p$	*Transposition term*
A, B	::=		Formulas
		F	*False*
		$(m \leftrightarrow n)A$	*Transposition*
		$A \wedge B$	*Conjunction*
		$A \Rightarrow B$	*Implication*
		$\mathbf{0}$	*Void*
		$A \mid B$	*Composition*
		$A \triangleright B$	*Guarantee*
		$n \circledR A$	*Revelation*
		$n \oslash A$	*Revelation Adjunct*
		$m\langle n \rangle$	*Message*
		$\Diamond A$	*Next step*
		$\forall x.A$	*First-order universal quantification*
		$\mathcal{N}x.A$	*Freshness quantification*
		X	*Propositional variable*
		$\forall X.A$	*Second-order universal quantification*

Figure 8: A Spatial logic for concurrency (SLC).

By combining the name occurrence predicate $@n$ (definable from the revelation operator $n \circledR A$ [24], see below) with other spatial operators it is possible to express action (or capability) observation modalities, as shown for the Ambient Calculus in [72], and for the π-calculus in [43]. To that end, properties characterizing processes and testers of various forms must be defined in fairly indirect and technically challenging way. For instance, the message predicate $n\langle m \rangle$ for the π-calculus fragment under consideration might be expressed:

$$A \blacktriangleright B \triangleq \neg(A \triangleright \neg B)$$
$$test(m, n) \triangleq Public \wedge \mathbf{1} \wedge @n \wedge @m \wedge (\mathbf{1} \blacktriangleright \Diamond \mathbf{0})$$
$$n() \triangleq Public \wedge \mathbf{1} \wedge \mathcal{N}x.(test(n, x) \triangleright \Diamond \mathsf{T})$$
$$n\langle m \rangle \triangleq test(m, n) \wedge (n() \triangleright \Diamond \mathsf{T})$$

More recently, a related approach was followed in [31] to show that, at least in some cases, a primitive temporal modality may be defined just from the available spatial operators (in the context of a spatial logic for bigraphs [60]).

SLC first-order terms denote pure names. In order to support object level reasoning about freshness constraints, the term language, besides variables and names (constants), also includes explicit transpositions at both the term and for-

mula level. For example, consider the valid entailment of SLC

$$A \wedge \neg @n \wedge \neg @m \models (m \leftrightarrow n)A$$

This statement captures, at the object level, the fundamental principle of equivariance, or freshness: "fresh names are all alike", if a property holds of a fresh name, it also holds of any other fresh name. Thus, the treatment of freshness in SLC is inherited from the Gabbay-Pitts models of freshness [68, 39]. In particular, SLC includes a modal version of the Gabbay-Pitts freshness quantifier $И x.A$. The semantics of the freshness quantifier is defined by

$$P \models И x.A \quad \triangleq \quad \textit{Exists } n\#И x.A \textit{ and } \; P \models A\{^{x}/_{n}\}$$

In order to logically express properties of restricted names, alternative spatial operators to the hidden name quantifier and the free name predicate have been proposed by Cardelli and Gordon in [24]. It turns out that the hidden name quantifier $\mathsf{H}x.A$ may be expressed by combining the freshness quantifier with the revelation operator, while the name occurrence predicate $@-$ may be expressed by the revelation operator:

$$\mathsf{H}x.A \triangleq И x.x \circledR A \qquad @n \triangleq \neg x \circledR \mathsf{T}$$

Conversely, we may also express revelation and the freshness quantifier from name occurrence and hidden name quantification.

$$И x.A \triangleq \mathsf{H}x.(\neg @x \wedge A) \qquad n \circledR A \triangleq \neg @n \wedge \mathsf{H}x.(n \leftrightarrow x)A$$

The guarantee operator is very expressive, the implicit quantification over the set of all processes brings it fairly close to a general set comprehension mechanism. For example, validity may be defined inside the logic [26] by

$$\textit{unsatisfiable}(A) \triangleq A \triangleright \mathsf{F} \qquad \textit{valid}(A) \triangleq \textit{unsatisfiable}(\neg A)$$

These constructions also allow an internal definition of entailment and then of fixed point operators as derived constructions:

$$A \Mapsto B \triangleq \textit{valid}(A \Rightarrow B) \qquad (\boldsymbol{\nu} X)A \triangleq \exists X.X \wedge (X \Mapsto A)$$

In [10, 12, 11, 13] the semantics, meta-theoretic properties, and a proof theory for SLC have been developed in detail. In particular, a cut-free sequent calculus formulation of a proof system for the whole spatial logic is provided. In order to capture the spatial structure of processes, sequents are tagged with a set of constraints, that specify dependencies between processes and between names, such

as reduction constraints and name apartness conditions:

$\langle S \rangle \Gamma \vdash \Delta$	Sequents, of the form
	$\langle S \rangle u_1 : A_1, \ldots, u_n : A_n \vdash v_1 : B_1, \ldots, v_m : B_m$
A_i, B_i	Formulas
u_i, v_j	Indexes, members of a process algebra (the worlds)
S	finite set of constraints (*e.g.*, equations, reductions)

The proof system is shown to be sound with respect to the intended process model. Due to the sequent based presentation, directed by the syntax of formulas, proofs of intuitive results may be obtained in a rather mechanical way, by decomposing formulas into subformulas. For the sake of illustration, we discuss the proof of a simple sequent, involving spatial reasoning. The sequent $(A \mid B) \wedge \mathbf{0} \vdash A$ asserts that if a process is void and satisfies $(A \mid B)$ then it also satisfies A. This fact is intuitively true, since all spatial decompositions of void are themselves void. Here is how a proof of $(A \mid B) \wedge \mathbf{0} \vdash A$ looks like

$$
\begin{array}{ll}
5.\ \langle S, u \doteq \mathcal{X} \mid \mathcal{Y}, u \doteq \mathbf{0}, \mathcal{X} \doteq \mathbf{0} \rangle \mathcal{X} : A, \mathcal{Y} : B \vdash u : A & \text{(by (Id), } u \doteq_S \mathcal{X}\text{)} \\
4.\ \langle S, u \doteq \mathcal{X} \mid \mathcal{Y}, u \doteq \mathbf{0} \rangle \mathcal{X} : A, \mathcal{Y} : B \vdash u : A & \text{(by 5, (S} \mid \mathbf{0}\text{), } \mathcal{X} \mid \mathcal{Y} \doteq_S \mathbf{0}\text{)} \\
3.\ \langle S, u \doteq \mathcal{X} \mid \mathcal{Y} \rangle \mathcal{X} : A, \mathcal{Y} : B, u : \mathbf{0} \vdash u : A & \text{(by 4, (}\mathbf{0}\text{L))} \\
2.\ \langle S \rangle u : (A \mid B), u : \mathbf{0} \vdash u : A & \text{(by 3, (} \mid \text{L))} \\
1.\ \langle S \rangle u : (A \mid B) \wedge \mathbf{0} \vdash u : A & \text{(by 2, (}\wedge\text{L))}
\end{array}
$$

Several techniques introduced in the proof-theoretic development of SLC are interesting from many perspectives. Worth to highlight are the use of explicit name transpositions to talk about freshness related properties, and its key role in the proof of cut-elimination. Building on a variant of SLC, a fully formalized development in spatial logic of the correctness proof for a peer to peer distributed algorithm (the Arrow Distributed Protocol of [37]) was carried out in [71].

5 Expressiveness of Dynamic Spatial Logics

It is clear that spatial logics are more expressive than more traditional behavioral logics for concurrency, in the sense that more processes are distinguished. The question arises then about what spatial logics are really talking about: in technical terms, what is exactly the separating power of spatial logics for each model considered. Posed in this terms, the issue was first addressed by Sangiorgi in [72], where it is shown that the equivalence induced on processes by the Ambient Logic [26] is "intensional", in the sense that it coincides with structural congruence (usually considered an intensional congruence, at least when compared with the extensional observational congruence). Related technical results

$$P \equiv^e Q\{^{X}/_{(\overline{q})P}\} \Rightarrow P \equiv^e (\mathbf{rec}\ X(\overline{q}).Q)[\overline{q}] \quad (\textit{Struct Rec Solve})$$
$$\alpha.P + \alpha.P \equiv^e \alpha.P \quad (\textit{Struct Cho Abs})$$

Figure 9: Extended Structural Congruence

about the expressiveness of spatial logics were also presented in [42] and in Lozes thesis [52]. Since a spatial logic may distinguish between deadlock and termination, we immediately conclude that the logical equivalence induced by such a logic must be strictly finer than usual bisimilarities. However, we don't need to necessarily understand this fact as revealing some form of intensionality, but possibly as a result of an enhanced observational power on our model. In general, one may consider several natural degrees of observational power, constrained by the observable structure of each intended spatial model. All studies contributing to drawing a map of the spectrum of spatial observational power along these lines are then quite interesting.

The case of the full synchronous π-calculus (with guarded choice and parametric recursion) is considered in [5], and the separating power and expressiveness of (a variant of) CBSL interpreted in that model was studied. It is concluded that the logical equivalence $=_L$ lies strictly in between structural congruence and strong bisimilarity. A (complete) equational axiomatization of $=_L$ is also presented, by means of the relation of "extended structural congruence" $\equiv^e$. It consists of all the usual rules and axioms for structural congruence, plus the two additional principles shown in Figure 9. The axiom (*Struct Rec Solve*) expresses a coinduction principle, asserting the existence of unique solutions to systems of equations modulo extended structural congruence; and the axiom (*Struct Cho Abs*) expresses the familiar choice idempotency law of strong bisimilarity. Notice that the gap between the relations $\equiv^e$ and $\sim$ is essentially due to the failure of the expansion law. Clearly, the expansion law is not expected to be valid in spatial models, as it collapses distributed behavior into centralized behavior.

If one removes from the logic the operators allowing the observation of restricted names (revelation or the hidden name quantifier), an even more coarser logical equivalence than $\equiv^e$ is obtained, but still strictly contained in strong bisimilarity. The resulting spatial model seems to be close, for example, to the intended models of complexation calculi [36] introduced for modeling biological systems. In this model, processes inside a system appear like collections of "blobs", that interact and change behavior as usual, but may also glue to each other, or break a part in several different blobs, as effect of interactions. To our best knowledge no precise equational characterization results about such a spatial model have been published to date. Also related to this line of research are several early studies on location-based process equivalences (see [28]), addressed for variants of CCS

even before mobile process models have become popular. As far as we are aware, precise connections between such notions of distributed bisimilarities and process equivalences induced by spatial logics did not have been much investigated yet.

One may wonder whether extensional observational equivalences, in the traditional sense, may be captured by purely spatial logics. Along this lines, Hirschkoff developed a spatial logic characterization of bisimilarity in the (finite) π-calculus [41]. Key to the characterization is the introduction of formulas characterizing certain processes up to bisimilarity, and the use of the guarantee operator to express contextual tests, in the spirit of context barbed congruence (the name occurrence predicate $@n$ then plays a role similar to barb observation). However, the logic considered makes a rather weak use of spatial connectives: in the absence of the composition operator $A \mid B$, the guarantee operator may be fairly interpreted in purely behavioral terms, as a contextual behavioral testing primitive. The results in [41] are then particularly interesting due to the relationship they establishes between the guarantee operator and the contextual tests present in the formulation of context barbed congruences.

On the opposite extreme, it is shown in [17] that there are very natural models of distributed computation where extensional observational equivalences (in the standard sense) may be characterized by spatial logics. For simplicity, we have based this study on a minimalist distributed process calculus, that nevertheless captures essential features of distributed systems, namely local synchronous communication, local computation, asynchronous remote communication, and partial failures. As a main result we conclude that standard context barbed congruence for such a model is characterized by a spatial logic making essential use of the "intensional" composition $A \mid B$ and void $\mathbf{0}$ operators.

In fact, it turns out that in such a case the composition operator $A \mid B$ is necessary to capture (extensional) observational equivalence, while on the other hand the guarantee operator is not considered at all in the characterization. These results support the claim that spatial observations do not need to be always considered intensional, even if expressive enough to talk about the structure of systems. Retrospectively, we may realize that the extensional behavioral equivalence considered by Sangiorgi in [72], already makes several distinctions between processes close to those realized by spatial observations. For example, by adapting some techniques of [17] to the intended model of [72] (the name-restriction-free Ambient Calculus), one may show that barbed congruent processes must always have the same number of ambients at the top level. This suggests that barbed congruence on the public Ambient Calculus may be not so far from structural congruence as one might think, and that spatial observations of the kind motivated by spatial logics may be appropriate, at least in some settings, to characterize extensional observational equivalences of distributed and mobile computing systems.

6 Decidability and Model-Checking

The study of decision problems and associated algorithms for spatial logics have been considered by several authors. Such results are important to assess the expressiveness of the logics, and contribute to guide the design of automated verification tools. A first result in the area was presented in [29], where it was shown that both model-checking and validity checking of pure spatial logics is undecidable in the presence of the guarantee, and universal / existential quantifiers over names. It soon become believed that the guarantee operator, due to the quantification over all processes it implicitly performs, and to the fact that it may be used to represent logical validity at the object level, would be a cause of undecidability and intractability of spatial logics, thus limiting its use in automatic proof tools.

However, building on a result of Calcagno, Yang and O'Hearn [19], it was shown in [18] that the static fragment of spatial logics is in fact decidable (even if with an extremely high complexity). About the same time, Lozes has proved, using related techniques, that the guarantee operator may be eliminated (on behalf of the remaining operators) from the static fragments of spatial logics, even in the presence of the freshness quantifier and the revelation operator, without any loss of expressiveness [51, 53]. Clearly, the elimination process is not constructive, since Ghelli and Conforti have given (almost simultaneously) a proof that static spatial logics including the freshness quantifiers and the name revelation operator are undecidable [30] (in fact, the source of undecidability turns out to be the revelation operator, not the fresh name quantification).

Right after that result became known, we have developed with Lozes an embedding of first-order logic into a fragment of dynamic spatial logic, using as the underlying model a tiny fragment of CCS. As a result, it was proved [14, 15, 52] that dynamic spatial logics with the guarantee operator are essentially undecidable, and thus that there cannot exist a complete proof system for SLC. An interesting particularity of these results is that they illustrates, in a purely propositional spatial logic, how one may express the quantifiers and variable bindings needed to represent satisfaction of first-order formulas by processes of certain special forms, specified by means of context-system formulas (using the guarantee operator). We thus obtain an unusual quantifier elimination result that does not build on replacing variables by constants, but rather encodes variables and bindings by means of the specification of an underlying environment-based abstract machine.

The results described above lead to the conclusion that pure spatial logics, involving the use of composition adjunct (guarantee) operator are very expressive and very powerful for general compositional specification and reasoning about the distributed behavior of processes, but also that the associated model and validity checking problems are undecidable, and incomplete in general. On the other hand, it is shown shown in [5] that adjunct-free spatial-behavioral logics are still

very expressive and useful for verification purposes, and that the associated model checking problem is decidable and complete for large classes of processes. A model-checking algorithm for the full synchronous π-calculus with choice and recursion against a variant of CSBL with fixpoints is also developed, such development provided a foundation for implementation of tools.

7 Model-Checking Tools

The Spatial Logic Model Checker (SLMC) is a tool supporting the automatic verification of behavioral-spatial properties of distributed concurrent systems expressed in a version of the synchronous π-calculus. The logic already includes an extension of MPW logic with parametric fixpoint operators, first-order quantifiers and definitions, so that the SLMC probably also qualifies as the most complete freely available tool for analyzing standard behavioral π-calculus logical specifications. The SLMC system was implemented by Vieira [75, 76], and builds on the decidable logic and algorithms discussed (and proved correct) in [5]. It is freely available on the web, both as a binary executable and in (OCaml) source form. Both the process modeling and logical specification languages are expressive enough to tackle many interesting examples. The system is very easy to use, and fairly fast (it builds on an on-the-fly algorithm).

We illustrate some of SLMC's capabilities, and the concrete syntax used for processes and for logical properties with a toy example inspired by a scenario of service oriented computing (borrowed from [20]). We thus consider a distributed system consisting of three partners: a Buyer, a Seller, and a Shipper. Each subsystem is modeled by a single-threaded process; we show in Figure 10 the concrete SLMC specification.

We now define some properties we may be interested in analyzing. The first one is (global) deadlock freedom, this is a simple purely behavioral property, that might already be expressed in a purely behavioral logic with fixed-point operators.

```
/* GLOBAL DEADLOCK FREEDOM */
defprop aLive = maxfix X. ((<tau>true) and [tau]X);
check System |= aLive;

Processing ...
- Number of states visited: 50 -
* Process System satisfies formula aLive *
```

We next consider a couple of spatial properties, related to resource usage: local deadlock absence, race freedom, and choreography conformance. These kinds of properties cannot be handled by other existing tools that support more standard

```
defproc Buyer =
  new session in (
        quoteCh!(session,buyer).
        session?(quote).
        select {
           session!(accept).
              session?(deliverydetails).
              Buyer;
           session!(reject).
              Buyer
        });

defproc Seller =
  quoteCh?(session,bu).
  session!(price).
  session?(choice).
  select {
    [choice=accept].
      new t in (
        deliveryCh!(t,seller).
        t?(deliverydetails,sh).
        session!(deliverydetails).
        Seller );
    [choice=reject].
      Seller
   };

defproc Shipper =
  deliveryCh?(t,se).
  t!(deliverydetails,shipper).
  Shipper;

defproc System = Buyer | Seller | Shipper;
```

Figure 10: SLMC π-calculus model of the Buyer-Shipper-Seller Scenario.

temporal or behavioral logics. We may specify (and verify) them as follows (see related examples in Section 3):

```
/* STUCK (LOCAL DEADLOCK) FREEDOM */
defprop  stuck =
  inside (exists x. ( 1 and <x!>true | always ( not <x?>true)));
check System |= always ( not ( stuck ));

Processing...
- Number of state visits: 1925 -
* Process System satisfies the formula always (not (stuck)) *

/* RACE FREEDOM */
defprop write(x) = (1 and <x!>true);
defprop read(x)  = (1 and <x?>true);
defprop hasRace =
    inside (exists x.( write(x) | write(x) | read(x) | true));
defprop raceFree = maxfix X.((not hasRace) and []X);
check System |= raceFree;

Processing ...
- Number of states visited: 17648 -
* Process System satisfies formula raceFree *
```

As our last illustration of SLMC, we present an application of spatial logic model checking to choreography conformance in web services, using our running example. We specify the "choreography" as a spatial logic definable property, and check whether the system verifies it. This is another nice example of spatial reasoning: one really wants to look at what messages are being sent from partner to partner inside the distributed system system, what sessions are being created an so on, in a situation where such interactions are not externally observable. In particular, we define properties talking about message exchanges inside the system, for example, `sBuyer2Seller(message)` states what one usually means in message sequence descriptions by an assertion of the form *message* : `Buyer` $\rightarrow$ `Seller`.

The property `sGlobalDescription` describes an infinite behaviour (via the greatest fixed point formula), according to a certain pattern. Namely, `Buyer` sends to `Seller` a message on `quoteCh`. Then, a new fresh channel `session` is created, and `Seller` replies to `Buyer` on `session`. Then `Buyer` answers the `Seller` proposal, also on channel `session`. After that, two situations are possible (captured by the disjunction in the formula): either the `Buyer` does not accept the `Seller`'s quote, and the protocol restarts, or the `Buyer` accepts the `Seller`'s proposal, and

a a sub-interaction between Seller and Shipper proceeds, until the whole transaction completes, and a new one may start.

```
/* CONFORMANCE TO CHOREOGRAPHY */
defprop sArrow(message,src,dst) =
  inside(
          (1 and @src and <message!>true)  |
          (1 and @dst and <message?>true ) | true);

defprop sBuyer2Seller(message) = sArrow(message,buyer,seller);
defprop sSeller2Buyer(message) = sArrow(message,seller,buyer);
defprop sShipper2Seller(message) = sArrow(message,shipper,seller);
defprop sSeller2Shipper(message) = sArrow(message,seller,shipper);

defprop sGlobalDescription =
  maxfix X.(
    sBuyer2Seller(quoteCh) and
    [] hidden session.
      ( sSeller2Buyer(session) and
        []( sBuyer2Seller(session) and
        [][]( X
              or
              ( sSeller2Shipper(deliveryCh) and
                [] hidden t.
                   ( sShipper2Seller(t) and
                     [] ( sSeller2Buyer(session) and []X)))))));

check System |= sGlobalDescription;

Processing ...
Number of states visited: 469 -
* Process System satisfies formula sGlobalDescription *
```

Several other examples may be found in the SLMC web site.

8 Spatial Type Systems

What new notions of types for concurrent and distributed processes spatial logics might motivate in a natural way? This certainly was a natural question to ask since the very first works on spatial logics started to appear. More recently, we have investigated notions of spatial types for concurrency and resource control in systems of distributed objects (or services) [8, 9]. This line of work motivated

a preliminary study of a notion of semantic types for concurrent systems, where types are interpreted as spatial-logic definable properties [6, 7]. In this section, we briefly overview these developments.

8.1 Types for Concurrency Control

A notion of spatial typing for systems of concurrent objects was introduced in [8, 9], the key idea is to expose at the interface level information about possible concurrent invocation of operations. The proposed type structure reflects a resource sensitive model, where a parallel composition type operator expresses resource independence, a sequential composition type operator expresses resource synchronization, and a type modality expresses resource ownership, thus capturing several fundamental constraints on resource access that arise in general concurrent systems. The basic constructors of the type system are the following ones

$$U, V ::= \mathbf{0} \mid \mathtt{l}(U)V \mid U \mid V \mid U \wedge V \mid U; V \mid U^{\circ} \mid U \triangleright V$$

to which a recursion operator (and type variables) is added. Types are interpreted as spatial logic formulas, denoting spatial properties of systems in a precise sense. Superficially, the underlying structure may seem close to a behavioral algebra, with parallel and sequential composition operators, and a primitive action operator (the method call $\mathtt{l}(U)V$). However, the aim now is not just to talk about the behavior of systems, but also about distribution of objects, and ownership transfer [63] in concurrent systems. For example, the spatial composition type $U \mid V$ states that a service may be safely used accordingly to U and V by independent (in terms of resource usage) clients, one using it as specified by U, the other as specified by V. In particular, the tasks U and V may be activated concurrently. An object typed by the *Travel* type defined as

$$\mathit{Travel} \triangleq (\mathtt{flight} \mid \mathtt{hotel}); \mathtt{order}$$

will be able to service the `flight` (we abbreviate $\mathtt{l}(\mathbf{0})\mathbf{0}$ by `l`) and `hotel` tasks simultaneously and after that (and only after that), the `order` task. The spatial interpretation of $U \mid V$ implies further consequences, namely that the (distributed) resources used by U and V do not interfere; this property is important to ensure closure under composition of certain safety properties of typed systems. Owned types, of the form U°, state not only that the service is usable as specified by U, but also that such usage is completely owned (so that a object possessing a reference of owned type may, for example, store it for later use). Owned types discipline the delegation (or transfer) of resources or service references between interacting partners. More familiar behavioral types may also be easily expressed in our type system. For example, using sequential composition and conjunction, the usage protocol of a file object might be specified thus:

$$
\begin{array}{rl}
F \triangleq & f[\ \texttt{flight}() = \cdots \mid \texttt{book}() = \cdots \mid \texttt{free}() = \cdots \parallel \ \parallel\ ] \\
H \triangleq & h[\ \ \texttt{hotel}() = \cdots \mid \texttt{book}() = \cdots \mid \texttt{free}() = \cdots \parallel \ \parallel\ ] \\
G \triangleq & gw[\ \texttt{pay}(s) = \textbf{if}\ bk.\texttt{debit}()\ \textbf{then}\ s.\texttt{book}()\ \textbf{else}\ s.\texttt{free}() \parallel \ \parallel\ ] \\
B \triangleq & br[\ \texttt{flight}() = f.\texttt{flight}() \mid \texttt{hotel}() = h.\texttt{hotel}() \mid \\
& \quad \texttt{order}() = (gw.\texttt{pay}(f); gw.\texttt{pay}(h)) \parallel \ \parallel\ ]
\end{array}
$$

Figure 11: The Travel Booking Service.

$$File(V) \triangleq (\texttt{open}; (\texttt{read}()V \wedge \texttt{write}(V))^\star; \texttt{close})^\star$$

where $U^\star \triangleq \textbf{rec}\ \alpha.(\mathbf{0} \wedge (U; \alpha))$ expresses iteration. By combining recursion with spatial types, we may also define shared types. A shared type $U!$ states of an object it types that it may be used according to an unbounded number of independent sessions, each one conforming to type U. By combining such operators, we may specify fine grained shared access protocols, such as the standard "multiple readers/unique writer" access pattern for state variables (we may represent state variables by objects with a "read" and "write" methods):

$$RW(V) \triangleq ((\texttt{read}()V)!; \texttt{write}(V^\circ))^\star$$

Finally, and crucially, guarantee types $U \triangleright V$ allow us to compose subsystems into larger systems, while preserving the properties ensured by their typings.

Before summarizing the type system, and to illustrate the intended distributed object model, we sketch a toy scenario of service composition, where several sites cooperate to provide a travel booking service (see Figure 11). First, there is an object F implementing a service for finding and booking flights. It provides three methods: `flight` to look for and reserve a flight, `book` to commit the booking, and `free` to release a reservation. A similar service is provided by object H, used for booking hotel rooms. We elide method implementations in F and G, but assume that the operations must be called in good order to avoid disruption, namely that after calling `flight`, a client is supposed to call either `book` or `free`. The broker B, that implements the front-end of the whole system, is client of F and H, and also of a payment gateway G. The gateway books items if succeeds in processing their payment through a remote bank service named bk. Our travel booking service, available at br, is used by first invoking the `flight` and `hotel` operations in any order. In fact, these operations may be called concurrently, since they trigger separate computations. Afterwards, the `order` operation may be invoked to book and pay for both items, delegating access to f and h to the gateway. The session will then terminate, and the broker will become ready for another round.

We show below how usage patterns such as these may be specified by typing, and how the type of a whole system may be compositionally defined from the types of its components. Intuitively, a type T describes a usage pattern for a given object. An assertion of the form $n : T$ states that the object named n may be safely used as specified by the type T. In general, the type of a network P is expressed by a composite assertion

$$n_1 : T_1 \mid \ldots \mid n_k : T_k$$

specifying types of various objects named $n_1, \ldots, n_k$ available from P to the external environment. Such an assertion (or typing environment) states that the system provides *independent* services at the names n_i, each one able to be safely used as specified by the type T_i respectively. We now explain the intuitive meaning of the various kinds of types, by interpreting them as properties of objects.

- An object satisfies $n : \mathbf{0}$ if it is idle, that is, if it has no running threads.
- An object satisfies $n : T \mid U$ if it can independently satisfy both $n : T$ and $n : U$. We may also understand such a typing as the specification of two independent views for the object n. More precisely, a $n : T \mid U$ typing says that the interfaces T and U provided by object n are based in disjoint (in a sense to be made precise below) sets of resources / subsystems, and thus may be safely invoked concurrently.
- An object satisfies $n : T \wedge U$ if it can satisfy both $n : T$ and $n : U$, although not necessarily concurrently. Conservatively, such an object may only be used either as specified by $n : T$ or as specified by $n : U$, being the choice made by the object's client.
- An object satisfies $n : T; U$ if it can satisfy first $n : T$ and afterwards $n : U$, in sequence. In particular, it will only be obliged to satisfy $n : U$ after being used as specified by $n : T$. Implicit in this description is the notion of usage according to a type, and termination of such an usage; we will get back to this point later.
- The owned type $n : T^\circ$ means that the object may be used as specified by T, but furthermore (and crucially) that this T view is *exclusively owned*. For example, a reference of type $n : T^\circ$ may be stored in the local state of an object, or returned by a method call, although a reference of type $n : T$ may not, because of possible liveness constraints associated to the type T.
- An object satisfies $n : \mathtt{l}(U)V$ if it offers a method $\mathtt{l}$ that whenever passed an argument of type U is ensured to return back a result of type V, and exercise, during the call, an usage of the argument conforming to type U. Thus, method types specify both safety and liveness properties.
- Recursive types are interpreted as greatest fixed points, as expected.

The type system is then based on the following forms of formal judgments, related to the three syntactical categories of the calculus, plus subtyping:

$\mathbf{A} <: \mathbf{B}$	(*Subtyping*)	$P :: \mathbf{A} \triangleright \mathbf{B}$	(*Networks*)
$[M; t] :: \mathbf{A} \shortmid \sigma \triangleright \mathbf{B} \shortmid \delta[U]$	(*Objects*)	$e :: \mathbf{A} \shortmid \sigma \triangleright \mathbf{B} \shortmid \delta[U]$	(*Expressions*)

The main judgment is the network typing assignment judgment, of the form $P :: \mathbf{A} \triangleright \mathbf{B}$. Intuitively, a judgment $P :: \mathbf{A} \triangleright \mathbf{B}$ asserts that if a network P is composed with any network Q that satisfies the typing $\mathbf{A}$, one is guaranteed to obtain a network $(P \mid Q)$ that satisfies the typing $\mathbf{B}$. Getting back to the travel booking example, we may show how types may be assigned to the system components. For F and H we may expect the typings $F :: \triangleright f : T_f$ and $H :: \triangleright h : T_h$, where

$$T_f \triangleq \mathbf{rec}\ \alpha.\mathtt{flight}(); (\mathtt{book}() \wedge \mathtt{free}()); \alpha$$
$$T_h \triangleq \mathbf{rec}\ \alpha.\mathtt{hotel}(); (\mathtt{book}() \wedge \mathtt{free}()); \alpha$$

For the gateway G, we let $G :: bk : T_{bank} \triangleright\ gw : T_{gw}$ where

$$T_{bk} \triangleq \mathbf{rec}\ \alpha.\mathtt{debit}()\mathbf{bool}; \alpha \qquad T_{gw} \triangleq \mathbf{rec}\ \alpha.\mathtt{pay}(\mathtt{book}() \wedge \mathtt{free}()); \alpha$$

We set $T_{br} \triangleq \mathbf{rec}\ \alpha.(\mathtt{flight}() \mid \mathtt{hotel}()); \mathtt{order}(); \alpha$. Then, the following judgment is derivable

$$(F \mid H \mid G \mid B) :: bk : T_{bk} \triangleright\ br : T_{br}$$

This judgment asserts that the network $(F \mid H \mid G \mid B)$, when composed with any system providing the T_{bk} type at bk, will be safe for use at br as specified by type T_{br}. Such typing may be obtained compositionally from the types of the subsystems in many ways. A possible root level split of the system may be between the broker $B :: gw : T_{gw}, f : T_f, h : T_h \triangleright\ br : T_{br}$ and the back-end subsystem $(G \mid H \mid F) :: bk : T_{bk} \triangleright\ gw : T_{gw}, f : T_f, h : T_h$, where we conclude by the forward composition typing rule (*TComp*).

The proof of soundness of our spatial type system is obtained by semantic means, building on a interpretation of types as properties expressible in a spatial logic. This approach may be seen as an instance of the general method of logical relations, well understood in the setting of functional programming, but still quite unexplored in a concurrency setting. A full description of the spatial type system briefly described in this section, and of the related safety results, may be found in [9], where a treatment of sharing and shared types is also further developed.

8.2 Logical Semantics of Types

The original understanding of types as predicates has not been a formal guiding principle for the design of types for concurrent calculi, where a syntactical view prevails (an exception is [27], where semantic subtyping for names was proposed).

In recent work [6, 7], it was demonstrated a semantic approach to types in concurrency, based on an interpretation of types as spatial logic definable properties. More precisely, it is show how the semantics of a general type structure for processes modeled in the π-calculus may be inductively (and compositionally) defined by resorting to a logical interpretation, reminiscent of the logical relations method, and considering as underlying semantic model the standard labeled transition system. A formal type system **T** then assigns types to processes by induction on the structure of processes (as expected from a type system) rather than by induction on the structure of types / formulas (as in the proof system of [11, 13] already mentioned in Section 4).

This semantic approach extends in scope other proposals to generic typing [46], that rely on standard syntactic techniques. Further advantages result from the way the several properties of interest are factored out in definitions and proofs. Soundness of each typing rule is proven separately by showing that it preserves validity: this is a standard technique in semantic proofs of soundness for formal systems, but quite alien to the common practice in type systems for concurrency, where the syntactic and monolithic (even if extremely well-succeeded and practical) subject-reduction technique [78] prevails. Subtyping may be dealt with as a completely orthogonal (with respect to typing) aspect, so that the soundness proofs do not depend on the syntactic presentation of subtyping, but only on its semantic properties. We can then pick for the subtyping relation any sound axiomatization of semantic entailment in the underlying logic; soundness of each instance of **T** is immediately granted as a consequence of the modular approach.

As we have already argued, most interesting process properties of the kinds considered by type systems (*e.g.*, channel arity mismatch, local deadlock) are not invariant under standard behavioral equivalences of processes, for instance, bisimilarities. Therefore, to characterize such kinds of properties traditional behavioral logics are not adequate. It turns out that spatial logics for concurrency offer an appropriate expressiveness. An important feature of spatial logics, shared by other resource-sensitive logics such as the separation logics [70, 62], is that its operators are able to separate and count resources; this sometimes understood as an "intensional" flavor to spatial logics. It is precisely such intensional character that seems necessary for the logical characterization of many "type-like" properties.

The system **T** manipulates type judgment of the form $P :: A \vdash B$ where A and B are spatial-behavioral types, and the turnstyle $\vdash$ is interpreted as the composition adjunct (guarantee). A sample rule of system **T** is the composition rule (*Seq*).

$$\frac{\begin{array}{c}(\textit{fn}(B)\ \textit{not free in the conclusion.})\\ P :: A \vdash A' \mid B \quad Q :: B \mid B' \vdash C\end{array}}{(\nu B)(P \mid Q) :: A \mid B' \vdash A' \mid C}$$

The logical characterizations of certain kinds of types we have introduced in

this work allowed us to understand certain familiar notions in types for concurrency, such as sharing and linearity [48], in a rather abstract setting. For example, we give a semantically defined notion of *sharing type*. Roughly, a type A is sharing if $A \mid A \models A$ (i.e., if it enjoys a contraction principle with respect to spatial decomposition). It turns out that kinds of types such as simple types [55] and I/O types [67] are sharing in this sense. We may show that simple types and I/O types may be embedded in **T**, by extending the basic subtyping relation $\mathbf{ST}_{<:}$ with additional subtyping axioms. Certain forms of session types [45] may also be embedded in the generic type system **T**, by combining behavioral types with simple types. Again, soundness of the obtained type system is obtained for "free", after one proves certain abstract properties (*e.g.*, sharing) of new type constructions.

9 Concluding Remarks

In our tutorial overview of dynamic spatial logics for distributed concurrent systems we have attempted to find a balance between the variety of covered aspects and the depth of the analysis; our aim was fundamentally to provide the key intuitions, concepts and results, while hinting to further developments and related work. In the initial sections, we have reviewed the technical context needed for the introduction of dynamic spatial logics, and introduced some of the key motivations and concepts in the area. The last sections have been devoted to the discussion of a selection of results, and applications. We now add some remarks on closely related work.

The general notion of spatial observation has motivated the introduction of logics for models with various kinds of static and dynamic structure, in particular for models where the dynamic aspects are not relevant at all. These include logics for trees [23], for graphs [21], and for XML-like semi-structured data [22].

Many of these logics, and the static fragment of dynamic spatial logics, are closely related to Bunched Logic [61], in particular to boolean BI, at least formally. BI is also behind the foundations of separation logic [70] developed by O'Hearn and Reynolds, and then of the development of applications of separation logic to shared variable concurrency [63].

The distinctive characteristic of spatial logics for concurrency, when compared with the separation logics, is their original aim of talking about structural dynamics. The difference of standpoint is also manifest in the underlying technical tools and approaches: process calculi, abstract interaction models, and modal logics in the case of dynamic spatial logics, and imperative programs, communicating processes, and assertion logics in the case of the separation logics; in line with the two main established approaches to concurrency in computer science for several decades now. The intent of dynamic spatial logics is to express general properties

of active structures evolving in space-time, while the separation logics aim is to express, at the logical level, independence constraints on passive resource spaces (such as the heap of imperative programs). The separation logic view of the composition operator is strongly motivated by notions of non-interference [69] in state manipulating programs; if a program gets past the proof rules of separation logic then (implicitly) it must also necessarily be race-free. In the case of dynamic spatial logic, this does not be the case, although properties such as race-freedom (as race-presence) may be expressed (explicitly) in such a logic. On the other hand, separation logic is proving to be quite successful in providing effective concepts, techniques and models to reason about programs written in practical low-level programming languages. The two views are not exclusive, and ideas arising in one field may hopefully inspire developments in the other, for example, applications of separation logic to process calculi [64], and introduction of separation (as a form of non-interference) in dynamic spatial logics [8].

As the general concept of space is becoming increasingly recognized as a fundamental ingredient in models for distributed interactive systems, as witnessed by the development of Mobile Ambients and more recently of bigraphs, we believe that logics combining spatial and behavioral observations are likely to play an important rule in the analysis of such models, both for increasing our fundamental understanding about concurrency and distributed computation, but also for the development of applications to the specification and verification of concrete technological artifacts.

Acknowledgments I thank Luca Aceto for the invitation to contribute for the Concurrency Column. Parts of this tutorial are based on joint work with Luca Cardelli, Etienne Lozes, and Hugo Vieira. Some of the views expressed also benefited from suggestions and challenging remarks in various moments from Daniel Hirshkoff, Luis Monteiro, Peter O'Hearn, and Davide Sangiorgi.

References

[1] R. M. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous pi-calculus. *Theoretical Computer Science*, 195(2):291–324, March 1998.

[2] G. Berry and G. Boudol. The Chemical Abstract Machine. *Theoretical Computer Science*, 96(1):217–248, April 1992.

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

[4] L. Caires. A Specification Logic for Mobility. Technical report, Universidade Nova de Lisboa, DI/FCT, 2000.

[5] L. Caires. Behavioral and Spatial Properties in a Logic for the Pi-Calculus. In I. Walukiewicz, editor, *Foundations of Software Science and Computation Structures*, number 2987 in Lecture Notes in Computer Science. Springer Verlag, 2004.

[6] L. Caires. Logical Semantics of Types for Concurrency. In U. Montanari and T. Mossakowski, editors, *Calco'07, 2nd Conference on Algebra and Coalgebra in Computer Science*, Lecture Notes in Computer Science. Springer-Verlag, 2007.

[7] L. Caires. Logical Semantics of Types for Concurrency. Technical Report 2/07, Departamento de Informatica FCT/UNL, 2007.

[8] L. Caires. Spatial-Behavioral Types, Distributed Services, and Resources. In U. Montanari and D. Sanella, editors, *TGC 2006 2dn Intl. Symp. on Trustworthy Global Computing*, Lecture Notes in Computer Science. Springer-Verlag, 2007.

[9] L. Caires. Spatial-Behavioral Types for Concurrency and Resource Control in Distributed Systems. Technical Report 3/07, Departamento de Informática FCT/UNL, 2007.

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

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

[12] L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part I). *Information and Computation*, 186(2):194–235, 2003.

[13] L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part II). *Theoretical Computer Science*, 3(322):517–565, 2004.

[14] L. Caires and E. Lozes. Elimination of Quantifiers and Undecidability in Spatial Logics for Concurrency. In P. Gardner and N. Yoshida, editors, *CONCUR 2004 - Concurrency Theory, 15th International Conference,Proceedings*, volume 3170 of *Lecture Notes in Computer Science*, pages 240–257. Springer-Verlag, 2004.

[15] L. Caires and E. Lozes. Elimination of Quantifiers and Undecidability in Spatial Logics for Concurrency. *Theoretical Computer Science*, 3(358):293–314, 2006.

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

[17] L. Caires and H. Vieira. Extensionality of Spatial Observations in Distributed Systems. *Electronic Notes in Theoretical Computer Science*, 2007.

[18] C. Calcagno, L. Cardelli, and A. D. 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.

[19] C. Calcagno, H. Yang, and O'Hearn. Computability and complexity results for a spatial assertion language for data structures. In R. Hariharan, M. Mukund, and V. Vinay, editors, *FSTTCS'2001*, volume 2245. Springer-Verlag, 2001.

[20] M. Carbone, K. Honda, and N. Yoshida. Structured Communication-Centred Programming for Web Services. In R. De Nicola, editor, *Programming Languages and Systems, 8th European Symposium on Programming, ESOP'07*, volume 1576 of *Lecture Notes in Computer Science*, pages 91–108. Springer-Verlag, 2007.

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

[22] L. Cardelli, P. Gardner, and G. Ghelli. Manipulating trees with hidden labels. *Electr. Notes Theor. Comput. Sci.*, 172:177–201, 2007.

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

[24] L. Cardelli and A. 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.

[25] L. Cardelli and A. D. Gordon. Mobile ambients. In Maurice Nivat, editor, *Proceedings of the First International Conference on Foundations of Software Science and Computation Structures (FoSSaCS '98)*, volume 1378 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[26] 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.

[27] G. Castagna, R. De Nicola, and D. Varacca. Semantic Subtyping for the π-Calculus. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005)*, pages 92–101. IEEE Computer Society, 2005.

[28] I. Castellani. Process Algebras with Localities. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*. North-Holland, 2001.

[29] W. Charatonik and J.-M. Talbot. The Decidability of Model-Checking Mobile Ambients. In D. Metayer, editor, *11th European Symposium on Programming (ESOP 2002)*, number 2305 in Lecture Notes in Computer Science. Springer-Verlag, 2001.

[30] G. Conforti and G. Ghelli. Decidability of Freshness, Undecidability of Revelation. In Igor Walukiwicz, editor, *Proc. of Foundations of Software Science and Computation Structures'2004*, number 2987 in Lecture Notes in Computer Science. Springer Verlag, 2004.

[31] G. Conforti, D. Macedonio, and V. Sassone. Spatial Logics for Bigraphs. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP*

2005, volume 3580 of *Lecture Notes in Computer Science*, pages 766–778. Springer-Verlag, 2005.

[32] Silvano Dal Zilio and Denis Lugiez. A logic you can count on. In *POPL 2004 – 31st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2004.

[33] M. Dam. *Relevance Logic and Concurrent Composition*. PhD thesis, University of Edinburgh, 1989.

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

[35] M. Dam. Proof Systems for π-calculus Logics. In *Logic for concurrency and synchronisation*, pages 145–212. Kluwer Academic Publishers, 2003.

[36] V. Danos and Laneve C. Formal molecular biology. *Theoretical Computer Science*, 325(1):69–110, 2004.

[37] M. J. Demmer and M. Herlihy. The Arrow Distributed Directory Protocol. In S. Kutten, editor, *Distributed Computing, 12th International Symposium, DISC '98*, volume 1499 of *Lecture Notes in Computer Science*, pages 119–133. Springer-Verlag, 1998.

[38] J. Engelfriet and Tj. Gelsema. Multisets and Structural Congruence of the π-calculus with Replication. *Theoretical Computer Science*, 211(1-2):311–337, 1999.

[39] 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.

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

[41] D. Hirschkoff. An Extensional Spatial Logic for Mobile Processes. In P. Gardner and N. Yoshida, editors, *CONCUR - Concurrency Theory, 15th International Conference,Proceedings*, volume 3170 of *Lecture Notes in Computer Science*, pages 325–339. Springer-Verlag, 2004.

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

[43] D. Hirschkoff, E. Lozes, and D. Sangiorgi. Minimality results for the spatial logics. In *Proc. of FSTTCS'2003*, LNCS. Springer Verlag, 2003.

[44] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1984.

[45] K. Honda, V. T. Vasconcelos, and M. Kubo. Language Primitives and Type Discipline for Structured Communication-Based Programming. In C. Hankin, editor, *7th European Symposium on Programming*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer-Verlag, 1998.

[46] A. Igarashi and N. Kobayashi. A Generic Type System for the Pi-Calculus. In *POPL 2001: 28th ACM Symp. on Principles of Programming Languages*, 2001.

[47] Cliff B. Jones. Tentative Steps Toward a Development Method for Interfering Programs. *ACM Trans. Program. Lang. Syst.*, 5(4):596–619, 1983.

[48] N. Kobayashi, B. C. Pierce, and D. N. Turner. Linearity and the Pi-Calculus. *ACM Trans. Program. Lang. Syst.*, 21(5):914–947, 1999.

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

[50] J. J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. In Catuscia Palamidessi, editor, *CONCUR 2000 - Concurrency Theory, 11th International Conference, Proceedings*, volume 1877 of *Lecture Notes in Computer Science*, pages 243–258. Springer-Verlag, 2000.

[51] E. Lozes. Adjuncts Elimination in the Static Ambient Logic. *Electr. Notes Theor. Comput. Sci.*, 96:51–72, 2004.

[52] E. Lozes. *Expressivit des Logiques d'Espaces*. PhD thesis, Ecole Normal Superieure de Lyon - Universidade Nova de Lisboa, 2004.

[53] E. Lozes. Elimination of Spatial Connectives in Static Spatial Logics. *Theoretical Computer Science*, 330(3):475–499, 2005.

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

[55] R. Milner. The Polyadic π-Calculus: A Tutorial. Technical Report 180, University of Edinburgh LFCS, 1991.

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

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

[58] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Part I + II. *Information and Computation*, 100(1):1–77, 1992.

[59] R. Milner, J. Parrow, and D. Walker. Modal Logics for Mobile Processes. *Theoretical Computer Science*, 114:149–171, 1993.

[60] Robin Milner. Pure bigraphs: Structure and dynamics. *Information and Computation*, 204(1):60–122, 2006.

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

[62] P. W. O'Hearn. Resources, Concurrency and Local Reasoning. In P. Gardner and N. Yoshida, editors, *Concur 15th Intl. Conf. on Concurrency Theory*, volume 3170 of *Lecture Notes in Computer Science*, pages 49–67. Springer-Verlag, 2004.

[63] P. W. O'Hearn. Resources, Concurrency, and Local Reasoning. *Theoretical Computer Science*, 375(1-3):271–307, 2007.

[64] P. W. O'Hearn. Separation Logic Semantics for Communicating Processes. In *Invited Lecture at Concur'07 - Unpublished*, 2007.

[65] D. M. R. Park. Concurrency and Automata on Infinite Sequences. In *Theoretical Computer Science, 5th GI-Conference, Proceedings*, volume 104 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[66] B. Pierce. Foundational calculi for programming languages. In *Computer Science and Engineering Handbook*. CRC Press, 1996.

[67] B. C. Pierce and D. Sangiorgi. Typing and Subtyping for Mobile Processes. *Mathematical Structures in Computer Science*, 6(5):409–453, 1996.

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

[69] J. C. Reynolds. Syntactic control of interference. In *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages, Tucson, Arizona, January 1978*, pages 39–46, 1978.

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

[71] A. Ribeiro, L. Caires, and L. Monteiro. Verifying the Arrow Distributed Protocol in a Spatial Logic. Technical Report 4, DI/FCT/UNL, December 2004.

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

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

[74] B. Victor and F. Moller. The mobility workbench - a tool for the pi-calculus. In D. L. Dill, editor, *Computer Aided Verification, 6th International Conference, CAV '94*, volume 818 of *Lecture Notes in Computer Science*, pages 428–440. Springer-Verlag, 1994.

[75] H. Vieira and L. Caires. Spatial Model Checker User's Manual and Tutorial (v0.9). Technical Report 2/05, Departamento de Informatica FCT/UNL, 2005.

[76] H. Vieira, L. Caires, and R. Viegas. Spatial Model Checker User's Manual and Tutorial (v1.0). Technical Report 5/05, Departamento de Informatica FCT/UNL, 2005.

[77] D. Walker. Objects in the π-calculus. *Journal of Information and Computation*, 116(2):253–271, 1995.

[78] A. K. Wright and M. Felleisen. A Syntactic Approach to Type Soundness. *Inf. Comput.*, 115(1):38–94, 1994.