Spatial and behavioral types in the pi-calculus*

Lucia Acciai Michele Boreale

Dipartimento di Sistemi e Informatica
Università di Firenze
{lacciai,boreale}@dsi.unifi.it

Abstract. We present a framework that combines ideas from spatial logics and Igarashi and Kobayashi's behavioural type systems, drawing benefits from both. In our approach, type systems for the pi-calculus are introduced where newly declared (restricted) names are annotated with spatial process properties, predicating on those names, that are expected to hold in the scope of the declaration. Types are akin to ccs terms and account for the processes abstract behaviour and "shallow" spatial structure. The type systems relies on spatial model checking, but properties are checked against types rather than against processes. The considered class of properties is rather general and, differently from previous proposals, includes both safety and liveness ones, and is not limited to invariants.
Keywords: pi-calculus, behavioural type systems, spatial logic.

1 Introduction

In the past few years, *spatial logics* [9,7] have emerged as promising tools for analyzing properties of systems described in process calculi. These logics aim at describing the spatial structure of processes. This makes them apt to express properties related to distribution and concurrency. An easy to grasp example is the race freedom property, stating that at any time, nowhere in the system there are two output actions ready on the same channel. The spectrum of properties that can be expressed by combination of simple spatial and behavioral connectives is very rich (see e.g. [7]). This richness is rather surprising, given the intensional nature of such logics: the process equivalences they induce coincide with, or come very close to, structural congruence (see e.g. [6]), a very fine equivalence that only permits elementary rearrangements of term structure.

A by known well-established trend in the field of process calculi is the use of *behavioural type systems* to simplify the analysis of concurrent message-passing programs [11,10,8]. Roughly, behavioural types are abstract representations of processes, yet sufficiently expressive to capture some properties of interest. In Igarashi and Kobayashi's work on generic type systems [10], the paper that pioneered this approach, processes are pi-calculus terms, while types are akin to simpler ccs terms. The crucial property enjoyed by the system is type soundness: in essence, for a certain class of properties (expressed in a simple modal logic), it holds that if a property is satisfied by a type then it is also satisfied by processes that inhabit that type. Results of this sort can in principle be used to effectively combine type checking and model checking. That is,

* Research partly supported by the EU within the FET-GC2 initiative, project SENSORIA.

in some cases it is possible to replace (expensive) model checking on message-passing processes by (cheaper) model checking on types. The paper [8] further elaborates on these themes. A limitation of behavioural type systems proposed so far concerns the kind of properties that can be tackled this way. Essentially, in [10,8], properties for which a general type soundness theorem works are safety invariants.

In the present paper, we try to combine the expressiveness of spatial logics with the effectiveness of behavioural type systems. More specifically, building on Igarashi and Kobayashi's work on generic type systems, we present type systems for the pi-calculus where newly declared (restricted) names are annotated with properties that predicate on those names. A process in the scope of a restriction is expected to satisfy, at run-time, the property expressed by the formula. We shall focus on properties expressible in a spatial logic – the *Shallow Logic* – which is a fragment of Caires and Cardelli's logic. Types are akin to CCS terms and account for (abstract) behaviour and "shallow" spatial structure of processes. The type system relies on (spatial) model checking: however, properties are checked against types rather than against processes. The considered class of properties is rather general and, unlike previous proposals [10,8], includes both safety and liveness ones, and it is not limited to invariants. Several examples of such properties – including race freedom, deadlock freedom and many others – are given throughout the paper. As another contribution of the paper, we elaborate a distinction between *locally* and *globally checkable* properties. Informally, a locally checkable property is one that can be model-checked against any type by looking at the (local) names it predicates about, while hiding the others; a globally checkable one requires looking also at names causally related to the local ones, hence in principle at names declared elsewhere in the process. These two classes of properties correspond in fact to two distinct type systems, exhibiting different degrees of compositionality and effectiveness (with the global one less compositional/effective). To sum up, we make the following contributions:

- we establish an explicit connection between spatial logics and behavioural type systems. In this respect, a key observation is that processes and the behavioural types they inhabit share the same "shallow" spatial structure, which allows us to prove quite precise correspondences between them and general type soundness theorems;
- we syntactically identify classes of formulae for which type soundness is guaranteed;
- unlike previous proposals, our type soundness results are not limited to safety properties nor to invariant properties;
- we investigate a distinction between locally and globally checkable properties.

Structure of the paper In Section 2 we introduce the language of processes, a standard polyadic pi-calculus. In Section 3 we introduce both spatial properties and the Shallow Logic, a simple language to denote them. In Section 4 the first type system, tailored to "local" properties, is presented and thoroughly discussed. Type soundness for this system is then discussed in Section 5 along with a few examples. A "global" variant of the type system, a soundness result and a few examples are presented and discussed in Section 6. Some limitations of our approach, and possible workarounds for them, are discussed in Section 7. A few remarks on further and related work conclude the paper in Section 8. Proofs are omitted due to space limitations.

$$P|\mathbf{0} \equiv P \qquad (P|Q)|R \equiv P|(Q|R) \qquad P|Q \equiv Q|P \qquad (\nu\tilde{x}:\tilde{\mathrm{t}})P|Q \equiv (\nu\tilde{x}:\tilde{\mathrm{t}})(P|Q) \quad \text{if } \tilde{x}\#Q$$

Table 1. Laws for structural congruence ≡ on processes

2 A process calculus

Processes. The language we consider is a synchronous polyadic pi-calculus [14] with guarded summations and replications. As usual, we presuppose a countable set $\mathcal{N}$ of *names*. We let lowercase letters $a, b, ..., x, y, ...$ range over names, and $\tilde{a}, \tilde{b}, ...$ range over tuples of names. Processes $P, Q, R, \ldots$ are defined by the grammar below

$$\alpha ::= a(\tilde{b}) \mid \overline{a}\langle\tilde{b}\rangle \mid \tau \qquad P ::= \textstyle\sum_{i\in I} \alpha_i.P_i \mid P|P \mid (\nu\tilde{b}:\tilde{\mathrm{t}})P \mid !a(\tilde{b}).P\,.$$

In the restriction clause, $\tilde{\mathrm{t}}$ is a tuple of channel types, to be defined later in Subsection 2, such that $|\tilde{\mathrm{t}}| = |\tilde{b}|$. Note that restriction acts on tuples $\tilde{b} = b_1, ..., b_n$, rather than on individual names. Indeed, the form $\nu\tilde{b}$ is equivalent to $\nu b_1 \cdots \nu b_n$ from an operational point of view. From the point of view of the type systems, however, the form $\nu\tilde{b}$ will allow us to specify properties that should hold of a group of names, as we shall see in later section. Notions of free names $\mathrm{fn}(\cdot)$, of bound names and of alpha-equivalence arise as expected and terms are identified up to alpha-equivalence. In particular, we let $\mathrm{fn}((\nu\tilde{b}:\tilde{\mathrm{t}})P) = (\mathrm{fn}(P) \cup \mathrm{fn}(\tilde{\mathrm{t}})) \setminus \tilde{b}$. To avoid arity mismatches in communications, we shall only consider terms that are well-sorted in some fixed sorting system (see e.g. [14]), and call $\mathcal{P}$ the resulting set of *processes*.

Notation. We shall write $\mathbf{0}$ for the empty summation. Trailing $\mathbf{0}$'s will be often omitted. Given $n \geq 0$ tuples of names $\tilde{b}_1, ..., \tilde{b}_n$, we abbreviate $(\nu\tilde{b}_1:\tilde{\mathrm{t}}_1)\cdots(\nu\tilde{b}_n:\tilde{\mathrm{t}}_n)P$ as $(\tilde{\nu}\,\tilde{b}_i:\tilde{\mathrm{t}}_i)_{i\in 1..n}P$, or simply $(\tilde{\nu}\,\tilde{b})P$ when no ambiguity about the $\tilde{\mathrm{t}}_i$ arises. In general, channel types annotations may be omitted when not relevant for the discussion. For any tuple/set of names $\tilde{x}$ and item t, $\tilde{x}\#t$ means that $\tilde{x} \cap \mathrm{fn}(t) = \emptyset$. This is extended to tuples of items $\tilde{t}$, written $\tilde{x}\#\tilde{t}$, as expected.

Over $\mathcal{P}$, we define a *reduction semantics*, based as usual on a notion of structural congruence and on a (labelled) reduction relation. These relations are defined, respectively, as the least congruence ≡ and as the least relation $\xrightarrow{\lambda}$ generated by the axioms in Table 1 and Table 2. As usual, the structural law for replication is replaced by a suitable reduction rule. Concerning Table 1, note that, similarly to [10], we drop two laws for restrictions ($(\nu\tilde{y})\mathbf{0} = \mathbf{0}$ and $(\nu\tilde{x})(\nu\tilde{y})P = (\nu\tilde{y})(\nu\tilde{x})P$): these laws become problematic once restrictions will be decorated with formulae containing free names. Concerning Table 2, note that we annotate each reduction with a label λ that carries information on the (free) subject name involved in the corresponding synchronization if any: $\lambda ::= \langle a\rangle \mid \langle\epsilon\rangle$. We define a hiding operator on labels, written $\lambda \uparrow_{\tilde{b}}$, as follows: $\lambda \uparrow_{\tilde{b}} = \langle a\rangle$ if $\lambda = \langle a\rangle$ and $a \notin \tilde{b}$, $\lambda \uparrow_{\tilde{b}} = \langle\epsilon\rangle$ otherwise.

Notation. In the sequel, for $\sigma = \lambda_1 \cdot \cdots \cdot \lambda_n$, $P \xrightarrow{\sigma} Q$ means $P \xrightarrow{\lambda_1} \cdots \xrightarrow{\lambda_n} Q$, and $P \to Q$ (resp. $P \to^* Q$) means $P \xrightarrow{\lambda} Q$ (resp. $P \xrightarrow{\sigma} Q$) for some λ (resp. σ). Moreover,

$$(\text{COM})\ \frac{\alpha_l = a(\tilde{x}) \quad \beta_n = \overline{a}\langle\tilde{b}\rangle \quad l \in I \quad n \in J}{\sum_{i\in I}\alpha_i.P_i \mid \sum_{j\in J}\beta_j.Q_j \xrightarrow{\langle a\rangle} P_l[\tilde{b}/\tilde{x}] \mid Q_n} \qquad (\text{TAU})\ \frac{j \in I \quad \alpha_j = \tau}{\sum_{i\in I}\alpha_i.P_i \xrightarrow{\langle\epsilon\rangle} P_j}$$

$$(\text{REP-COM})\ \frac{\beta_n = \overline{a}\langle\tilde{b}\rangle \quad n \in J}{!a(\tilde{x}).P \mid \sum_{j\in J}\beta_j.Q_j \xrightarrow{\langle a\rangle} !a(\tilde{x}).P \mid P[\tilde{b}/\tilde{x}] \mid Q_n} \qquad (\text{PAR})\ \frac{P \xrightarrow{\lambda} P'}{P \mid Q \xrightarrow{\lambda} P' \mid Q}$$

$$(\text{STRUCT})\ \frac{P \equiv Q \quad Q \xrightarrow{\lambda} Q' \quad Q' \equiv P'}{P \xrightarrow{\lambda} P'} \qquad (\text{RES})\ \frac{P \xrightarrow{\lambda} P'}{(\nu\tilde{x}:\tilde{\mathsf{t}})P \xrightarrow{\lambda\uparrow_{\tilde{x}}} (\nu\tilde{x}:\tilde{\mathsf{t}})P'}$$

Table 2. Rules for the reduction relation $\xrightarrow{\lambda}$ on processes.

we say that a process P has a *barb* a (resp. $\overline{a}$), written $P \searrow_a$ (resp. $P \searrow_{\overline{a}}$), if $P \equiv (\tilde{\nu}\tilde{b})(\sum_i \alpha_i.P_i + a(\tilde{x}).Q|R)$ (resp. $P \equiv (\tilde{\nu}\tilde{b})(\sum_i \alpha_i.P_i + \overline{a}\langle\tilde{c}\rangle.Q|R)$), with $a \notin \tilde{b}$.

Types. The set $\mathcal{T}$ of *types* $\mathsf{T}, \mathsf{S}, \mathsf{U}, \ldots$ is generated by the following grammar:

$$\mu ::= a(\mathsf{t}) \mid \overline{a} \mid \tau \qquad \mathsf{t} ::= (\tilde{x}:\tilde{\mathsf{t}})\mathsf{T} \qquad \mathsf{T} ::= \textstyle\sum_i \mu_i.\mathsf{T}_i \mid \mathsf{T}|\mathsf{T} \mid\ !a(\mathsf{t}).\mathsf{T} \mid (\nu\tilde{a}:\tilde{\mathsf{t}})\mathsf{T}$$

with $\tilde{x}\#\tilde{\mathsf{t}}$ and $\tilde{x} \subseteq \mathrm{fn}(\mathsf{T})$. In channel types $(\tilde{x}:\tilde{\mathsf{t}})\mathsf{T}$, we stipulate that $(\tilde{x}:\tilde{\mathsf{t}})$ is a binder with scope T. Informally, $a(\mathsf{t}).\mathsf{T}$ is a process type where a can transport names of channel-type t. In a channel type $(\tilde{x}:\tilde{\mathsf{t}})\mathsf{T}$, $\tilde{x}$ and $\tilde{\mathsf{t}}$ represent, respectively, the formal parameters and types of objects that can be passed along the channel, while type T is a process type prescribing a usage of those parameters. Note that, in $(\tilde{x}:\tilde{\mathsf{t}})\mathsf{T}$, it might in general be $\mathrm{fn}(\mathsf{T}) \setminus \tilde{x} \neq \emptyset$. In the sequel, we shall often omit writing the channel type $()\mathbf{0}$, writing e.g. $(x)\overline{x}$ instead of $(x:()\mathbf{0})\overline{x}$. Process types are akin to ccs terms bearing annotations on input prefixes and restrictions. Notions of free names, alpha-equivalence, structural congruence and reduction for types parallel those of processes. Note, that annotations contribute to the set of free names [1] of a type, but do not play a direct role in its reduction semantics (e.g. $\overline{c}.\mathsf{T}|c(\mathsf{t}).\mathsf{S} \xrightarrow{\langle c\rangle} \mathsf{T}|\mathsf{S}$).

3 Properties

We first take a general view of properties as *P-sets*, that is sets of processes and types (subject to certain conditions). Then introduce *Shallow Logic*, a simple language to denote a class of such properties. Although processes and types live in different worlds, for the purposes of this section it is possible and convenient to deal with them in a uniform manner. In what follows, we let $A, B, \ldots$ range over the set $\mathcal{U} \triangleq \mathcal{P} \cup \mathcal{T}$. Elements of $\mathcal{U}$ will be generically referred to as *terms*.

[1] Indeed, the reason for introducing these annotations is precisely to ensure that, in the type systems we shall introduce, whenever $\Gamma \vdash P : \mathsf{T}$ then $\mathrm{fn}(P) \subseteq \mathrm{fn}(\mathsf{T})$.

P-sets. Following [7,6], a property set, P-set in brief, is a set of terms closed under structural congruence and having a finite support: this intuitively means that the set of names that are "relevant" for the property is finite (somewhat analogous to the notion of free names for syntactic terms). In the following, we let $\{a \leftrightarrow b\}$ denote the *transposition* of a and b, that is, the substitution that assigns a to b and b to a, and leave the other names unchanged. For $\Phi \subseteq \mathcal{U}$, we let $\Phi\{a \leftrightarrow b\}$ denote $\{A\{a \leftrightarrow b\} | A \in \Phi\}$.

Definition 1 (support, P-set, least support). *Let be $\Phi \subseteq \mathcal{U}$ and $N \subseteq \mathcal{N}$. (a) N is a* support *of Φ if for each $a, b \notin N$, it holds that $\Phi\{a \leftrightarrow b\} = \Phi$. (b) A* property set *(*P-set*) is a set of terms $\Phi \subseteq \mathcal{U}$ that is closed under $\equiv$ and has finite support. (c) The* least support *of Φ, written* $\mathrm{supp}(\Phi)$*, is defined as* $\mathrm{supp}(\Phi) \stackrel{\triangle}{=} \bigcap_{N \text{ support of } \Phi} N$.

In other words, N is a support of Φ if renaming names *outside* N with fresh names does not affect Φ. P-sets have finite supports, and since countable intersection of supports is still a support, they also have a least support. In the rest of the paper we will deal with properties that need not be invariant through reductions. This calls for a notion of λ*-derivative* of a P-set Φ, describing the set of terms reachable via λ-reductions from terms in Φ: $\Phi_\lambda \stackrel{\triangle}{=} \{B | \exists A \in \Phi : A \stackrel{\lambda}{\rightarrow} B\}$. The λ-derivative of a P-set is a P-set.

Proposition 1. *Let Φ be a P-set such that $\Phi_\lambda = \Phi$ for $\lambda = \langle\epsilon\rangle$ or $\lambda = \langle b\rangle$ with $b \notin \mathrm{supp}(\Phi)$. For any reduction label λ, Φ_λ is a P-set and* $\mathrm{supp}(\Phi_\lambda) \subseteq \mathrm{supp}(\Phi)$.

The Ok predicate defined below individuates P-sets that enjoy certain desirable conditions. (1) requires a P-set to be closed under parallel composition with terms not containing free names (2) demands a P-set to be invariant under reductions that do not involve names in its support. Finally, (3) requires preservation of (1) and (2) under derivatives.

Definition 2 ($\mathrm{Ok}(\cdot)$ **predicate).** *We define* $\mathrm{Ok}(\cdot)$ *as the largest predicate on P-sets such that whenever* $\mathrm{Ok}(\Phi)$ *then: (1) for any $A, B \in \mathcal{P}$ s.t.* $\mathrm{fn}(B) = \emptyset$*: $A \in \Phi$ if and only if $A|B \in \Phi$; similarly for $A, B \in \mathcal{T}$; (2) $\Phi_\lambda = \Phi$ for $\lambda = \langle\epsilon\rangle$ or $\lambda = \langle b\rangle$ with $b \notin \mathrm{supp}(\Phi)$; (3) for each λ,* $\mathrm{Ok}(\Phi_\lambda)$ *holds.*

In the rest of the paper, we shall focus on properties represented by Ok P-sets.

Shallow Logic. The logic for the pi-calculus we introduce below can be regarded as a fragment of Caires and Cardelli's Spatial Logic [7]. We christen this fragment *Shallow Logic*, as it allows us to speak about the dynamic as well as the "shallow" spatial structure of processes and types. In particular, the logic does not provide for modalities that allows one to "look underneath" prefixes. Another important feature of this fragment is that the basic modalities focus on channel *subjects*, ignoring the object part at all. This selection of operators is sufficient to express a variety of interesting process properties (race freedom, unique receptiveness [16], deadlock freedom, to mention a few), while being tractable from the point of view of verification (see also Caires' [6]).

Definition 3 (Shallow Logic). *The set $\mathcal{F}$ of* Shallow Logic *formulae $\phi, \psi, \ldots$ is given by the following syntax, where $a \in \mathcal{N}$ and $\tilde{a} \subseteq \mathcal{N}$:*

$$\phi ::= \mathbf{T} \mid \phi \vee \phi \mid \langle a\rangle\phi \mid \langle\tilde{a}\rangle^*\phi \mid \langle -\tilde{a}\rangle^*\phi \mid \neg\phi \mid a \mid \overline{a} \mid \phi|\phi \mid \mathrm{H}^*\phi.$$

$$
\begin{array}{ll}
[\![\mathbf{T}]\!] = \mathcal{U} & [\![\mathrm{H}^*\phi]\!] = \{A \,|\, \exists \tilde{a}, B : A \equiv (\tilde{\nu}\tilde{a})B,\ \tilde{a}\#\phi,\ B \in [\![\phi]\!]\} \\
[\![\phi_1 \vee \phi_2]\!] = [\![\phi_1]\!] \cup [\![\phi_2]\!] & [\![\phi_1|\phi_2]\!] = \{A \,|\, \exists A_1, A_2 : A \equiv A_1|A_2,\ A_1 \in [\![\phi_1]\!],\ A_2 \in [\![\phi_2]\!]\} \\
[\![\neg\phi]\!] = \mathcal{U} \setminus [\![\phi]\!] & [\![\langle a\rangle\phi]\!] = \{A \,|\, \exists B : A \xrightarrow{\langle a\rangle} B,\ B \in [\![\phi]\!]\} \\
[\![a]\!] = \{A \,|\, A \searrow_a\} & [\![\langle \tilde{a}\rangle^*\phi]\!] = \{A \,|\, \exists \sigma, B : A \xrightarrow{\sigma} B,\ \sigma \in \{\langle b\rangle | b \in \tilde{a}\}^*,\ B \in [\![\phi]\!]\} \\
[\![\overline{a}]\!] = \{A \,|\, A \searrow_{\overline{a}}\} & [\![\langle -\tilde{a}\rangle^*\phi]\!] = \{A \,|\, \exists \sigma, B : A \xrightarrow{\sigma} B,\ \tilde{a}\#\sigma,\ B \in [\![\phi]\!]\}
\end{array}
$$

Table 3. Interpretation of formulae over terms.

The free names of a formula ϕ, written $\mathrm{fn}(\phi)$, are defined as expected. We let $\mathcal{F}_{\tilde{x}} = \{\phi \in \mathcal{F} : \mathrm{fn}(\phi) \subseteq \tilde{x}\}$. The set of logical operators includes spatial $(a, \overline{a}, |, \mathrm{H}^*)$ as well as dynamic $(\langle a\rangle, \langle \tilde{a}\rangle^*, \langle -\tilde{a}\rangle^*)$ connectives, beside the usual boolean connectives, including a constant $\mathbf{T}$ for "true". The interpretation of $\mathcal{F}$ over the set of processes is given in Table 3. Connectives are interpreted in the standard manner. We write $A \models \phi$ if $A \in [\![\phi]\!]$.

Interpretations of formulae are P-sets, as stated below.

Lemma 1. *Let $\phi \in \mathcal{F}$. Then $[\![\phi]\!]$ is a P-set and $fn(\phi) \supseteq \mathrm{supp}([\![\phi]\!])$.*

Notation. In what follows, when no confusion arises, we shall often denote $\Phi = [\![\phi]\!]$ just as ϕ. Moreover, we shall write $A \models \Phi$ to mean $A \in \Phi$. We abbreviate $\neg\langle -\tilde{x}\rangle^*\neg\phi$ as $\Box^*_{-\tilde{x}}\phi$. Moreover, $\langle -\emptyset\rangle^*\phi$ and $\Box^*_{-\emptyset}\phi$ are abbreviated as $\Diamond^*\phi$ and $\Box^*\phi$, respectively. Note that $\Diamond^*$ and $\Box^*$ correspond to the standard "eventually" and "always" modalities as definable, e.g., in the mu-calculus.

A further motivation for our particular selection of modalities is that satisfaction of any formula of $\mathcal{F}$ is, so to speak, invariant under parallel composition. In particular, whether A satisfies or not a property ϕ of a bunch of names $\tilde{x}$, should not depend on the presence of a parallel closed context B. Formulae of Cardelli and Caires' Spatial Logic outside $\mathcal{F}$ do not, in general, meet this requirement. As an example, the requirement obviously fails for $\neg(\neg\mathbf{0}|\neg\mathbf{0})$, saying that there is at most one non-null thread in the process. As another example, take the formula $\Diamond\mathbf{T}$, where $\Diamond$ is the one-step modality, saying that one reduction is possible: the reduction might be provided by the context B and not by A. This explains the omission of the one-step modality from Shallow Logic.

Lemma 2. *Let A be a term and $\phi \in \mathcal{F}_{\tilde{x}}$. For any term B such that $A|B$ is a term and $\mathrm{fn}(B) = \emptyset$ we have that $A \models \phi$ if and only $A|B \models \phi$.*

Example 1 (sample formulae). The following formulae define properties depending on a generic channel name a. They will be reconsidered several time throughout the paper.

$$
\begin{array}{rrcl}
\text{Race freedom:} & \mathit{NoRace}(a) & \triangleq & \Box^*\ \ \neg\mathrm{H}^*(\overline{a}|\overline{a}) \\
\text{Unique receptiveness:} & \mathit{UniRec}(a) & \triangleq & \Box^*\ \ (a \wedge \neg\mathrm{H}^*(a|a)) \\
\text{Responsiveness:} & \mathit{Resp}(a) & \triangleq & \Box^*_a \Diamond^* \langle a\rangle \\
\text{Deadlock freedom:} & \mathit{DeadFree}(a) & \triangleq & \Box^*[\ (\overline{a} \rightarrow \mathrm{H}^*(\overline{a}|\Diamond^* a)) \wedge (a \rightarrow \mathrm{H}^*(a|\Diamond^* \overline{a}))\ ].
\end{array}
$$

NoRace(a) says that it will never be the case that there are two concurrent outputs competing for synchronization on a. *UniRec*(a) says that there will always be exactly one receiver ready on channel a. $Resp(a)$ says that, until a reduction on a does not take place, it is possible to reach a reduction on a. If a is a return channel of some invoked service or function, this means the service or function will, under a suitable fairness assumption, eventually respond (see also [3]). Finally, $DeadFree(a)$ says that each output on a will eventually meet a synchronizing input, and vice-versa.

We shall sometimes need to be careful about the placement of the modality $\langle -\tilde{a}\rangle^*$ with respect to negation $\neg$. To this purpose, it is convenient to introduce two subsets of formulae, positive and negative ones.

Definition 4 (positive and negative formulae). *We say a formula ϕ is* positive *(resp.* negative*) if each occurrence of modality $\langle -\tilde{a}\rangle^*$ in ϕ is in the scope of an even (resp. odd) number of negations "$\neg$".*

We let $\mathcal{F}^+$ (resp. $\mathcal{F}^-$) denote the subset of positive (resp. negative) formulae in $\mathcal{F}$. The sets $\mathcal{F}^+_{\tilde{x}}$ and $\mathcal{F}^-_{\tilde{x}}$ are defined as expected.

Example 2. Concerning the formulae introduced in Example 1, note that *NoRace*(a) and *UniRec*(a) are negative, while both $Resp(a)$ and $DeadFree(a)$ are neither positive nor negative, as in both the modality $\Diamond^*$ occurs both in negative and in positive position.

Note that our definitions of "positive" and "negative" are more liberal than the ones considered by Igarashi and Kobayashi [10], where the position of all spatial modalities – including the analogs of $|$, a and $\overline{a}$ – w.r.t. negation must be taken into account (e.g., unique receptiveness would not be considered as negative in the classification of [10]). This difference will have influential consequences on the generality of the type soundness theorems of the type systems. In the rest of the paper, we shall mainly focus on formulae whose denotations are Ok P-sets. We write Ok(ϕ) if Ok($[\![\phi]\!]$) holds. The following lemma provides a sufficient syntactic condition for a formula to be Ok.

Lemma 3. *Let ϕ be a Shallow Logic formula of the form either $\Box^*\psi$ or $\Box^*_{-\tilde{a}}\Diamond^*\psi'$, where ψ' does not contain $\neg$. Then* Ok(ϕ).

Example 3. Formulas in Example 1 are in the format of Lemma 3, hence they are Ok.

4 A "Local" Type System

We present here our first type system. The adjective "local" refers to the controlled way P-set membership (that is, model checking, in practical cases) is checked.

Annotated processes. As anticipated in Section 2, the type system works on annotated processes. Each restriction introduces a property, under the form of an Ok P-set, that depends on the restricted names and is expected to be satisfied by the process in the restriction's scope. This means that, for annotated processes, the clause of restriction is modified thus $P ::= \cdots \mid (\nu\tilde{a} : \tilde{t}\,;\, \Phi)P$ with $\tilde{a} \supseteq \mathrm{supp}(\Phi)$ and Ok(Φ). For brevity, when

$$\text{(T-Inp)}\ \frac{\Gamma \vdash a : (\tilde{x} : \tilde{\mathsf{t}})\mathsf{T} \quad \Gamma, \tilde{x} : \tilde{\mathsf{t}} \vdash P : \mathsf{T}|\mathsf{T}' \quad \tilde{x}\#\mathsf{T}'}{\Gamma \vdash a(\tilde{x}).P : a((\tilde{x} : \tilde{\mathsf{t}})\mathsf{T}).\mathsf{T}'} \qquad \text{(T-Tau)}\ \frac{\Gamma \vdash P : \mathsf{T}}{\Gamma \vdash \tau.P : \tau.\mathsf{T}}$$

$$\text{(T-Out)}\ \frac{\Gamma \vdash a : (\tilde{x} : \tilde{\mathsf{t}})\mathsf{T} \quad \Gamma \vdash \tilde{b} : \tilde{\mathsf{t}} \quad \Gamma \vdash P : \mathsf{S}}{\Gamma \vdash \overline{a}\langle\tilde{b}\rangle.P : \overline{a}.(\mathsf{T}[\tilde{b}/\tilde{x}]\,|\,\mathsf{S})} \qquad \text{(T-Eq)}\ \frac{\Gamma \vdash P : \mathsf{T} \quad \mathsf{T} \equiv \mathsf{S}}{\Gamma \vdash P : \mathsf{S}}$$

$$\text{(T-Sum)}\ \frac{|I| \neq 1 \quad \forall i \in I :\ \Gamma \vdash \alpha_i.P_i : \mu_i.\mathsf{T}_i}{\Gamma \vdash \sum_i \alpha_i.P_i : \sum_i \mu_i.\mathsf{T}_i} \qquad \text{(T-Rep)}\ \frac{\Gamma \vdash a(\tilde{x}).P : a(\mathsf{t}).\mathsf{T}}{\Gamma \vdash\, !a(\tilde{x}).P :\, !a(\mathsf{t}).\mathsf{T}}$$

$$\text{(T-Res)}\ \frac{\Gamma, \tilde{a} : \tilde{\mathsf{t}} \vdash P : \mathsf{T} \quad \mathsf{T}\downarrow_{\tilde{a}} \models \Phi}{\Gamma \vdash (\nu\tilde{a} : \tilde{\mathsf{t}};\, \Phi)P : (\nu\tilde{a} : \tilde{\mathsf{t}})\mathsf{T}} \qquad \text{(T-Par)}\ \frac{\Gamma \vdash P : \mathsf{T} \quad \Gamma \vdash Q : \mathsf{S}}{\Gamma \vdash P|Q : \mathsf{T}|\mathsf{S}}$$

Table 4. Typing rules.

no confusion arises we shall omit writing explicitly channel types and properties in restrictions, especially when $\mathsf{t} = ()\mathbf{0}$ and $\Phi = [\![\mathbf{T}]\!]$. The reduction rule for restriction on annotated processes takes into account the λ-derivative of Φ in the continuation process:

$$(\textsc{res})\quad \frac{P \xrightarrow{\lambda} P'}{(\nu\tilde{x} : \tilde{\mathsf{t}};\, \Phi)P \xrightarrow{\lambda\uparrow_{\tilde{x}}} (\nu\tilde{x} : \tilde{\mathsf{t}};\, \Phi_\lambda)P'}\ .$$

For an annotated process P, we take $P \models \phi$ to mean that the plain process obtained by erasing all annotations from P satisfies ϕ. A "good" process is one that satisfies its own annotations at an active position. Formally:

Definition 5 (well-annotated processes). *A process $P \in \mathcal{P}$ is* well-annotated *if whenever $P \equiv (\tilde{\nu}\tilde{b})(\nu\tilde{a} : \Phi)Q$ then $Q \models \Phi$.*

Typing rules. Judgements of type system are of the form $\Gamma \vdash P : \mathsf{T}$, where: $P \in \mathcal{P}$, $\mathsf{T} \in \mathcal{T}$ and Γ is a *context*, that is, a finite partial map from names $a, b, c, \ldots$ to channel types $\mathsf{t}, \mathsf{t}', \ldots$. We write $\Gamma \vdash a : \mathsf{t}$ if $a \in \mathrm{dom}(\Gamma)$ and $\Gamma(a) = \mathsf{t}$. We say that a context is *well-formed* if whenever $\Gamma \vdash a : (\tilde{x} : \tilde{\mathsf{t}})\mathsf{T}$ then $\mathrm{fn}(\mathsf{T}, \tilde{\mathsf{t}}) \subseteq \tilde{x} \cup \mathrm{dom}(\Gamma)$. In what follows we shall only consider well-formed contexts. Contexts are assumed to be well-formed in rules of the type system. In the type system, we make use of a "hiding" operation on types, $\mathsf{T}\downarrow_{\tilde{x}}$, which masks the use of names not in $\tilde{x}$ in T (as usual, in the definition we assume that all bound names in T and t are distinct from each other and disjoint from the set of free names and from $\tilde{x}$).

Definition 6 (hiding on types). *For any type T and $\tilde{x}$, we let $\mathsf{T}\downarrow_{\tilde{x}}$ denote the type obtained by replacing every occurrence of prefixes $a(\mathsf{t}).$ and $\overline{a}.$ with $\tau.$, for each $a \in \mathrm{fn}(\mathsf{T}) \setminus \tilde{x}$. Hiding on channel types, $\mathsf{t}\downarrow_{\tilde{x}}$, is defined similarly.*

E.g., $(\nu a : \mathsf{t})(a(\mathsf{t}).b(\mathsf{t}')|a(\mathsf{t}).c|\overline{c}|\overline{a})\downarrow_b = (\nu a : \mathsf{t}\downarrow_{a,b})(a(\mathsf{t}\downarrow_{a,b}).b(\mathsf{t}'\downarrow_{a,b})|a(\mathsf{t}\downarrow_{a,b}).\tau|\tau|\overline{a})$. The rules of the type system are shown in Table 4. The structure of the system is along the lines of [10]; the main differences are discussed in Section 7. The key rules are (T-Inp), (T-Out), (T-Res) and (T-Eq). By and large, the system works as follows: given

a process P, it computes an abstraction of P in the form of a type T. At any restriction $(\nu\tilde{a} : \tilde{\mathsf{t}}; \Phi)P$ (rule (T-Res)), the abstraction T obtained for P is used to check that P's usage of names $\tilde{a}$ fulfills property Φ ($\mathsf{T} \downarrow_{\tilde{a}} \models \Phi$; in practical cases Φ is a shallow logic formula and this is actually spatial model checking). Note that, thanks to $\downarrow_{\tilde{a}}$, this is checked without looking at the environment: only the part of T that depends on $\tilde{a}$, that is $\mathsf{T} \downarrow_{\tilde{a}}$, is considered, the rest is masked. In particular, in $\mathsf{T} \downarrow_{\tilde{a}}$, any masked subterm that appears in parallel to the non-masked subterm can be safely discarded (a consequence of condition 1 of Ok). In this sense the type system is "local".

Rules for input and output are asymmetric, in the sense that, when typing a receiver $a(\tilde{x}).P$, the type information on P that depends on the input parameters $\tilde{x}$ is moved to the sender process. The reason is that the transmitted names $\tilde{b}$ are statically known only by the sender (rule (T-Out)). Accordingly, on the receiver's side (rule (T-Inp)), one only keeps track of the part of the continuation type that does not depend on the input parameters, that is T'. More precisely, the type of the continuation P is required to decompose – modulo type congruence – as $\mathsf{T}|\mathsf{T}'$, where T is the type prescribed by the context for a and T', which should not mention the input parameters $\tilde{x}$, is anything else. In essence, in well typed processes, all receivers on a must share a common part that deals with the received names $\tilde{x}$ as prescribed by the type T.

Finally, (T-Eq) is related to sub-typing. As mentioned in the Introduction, a key point of our system is that types should reflect the (shallow) spatial structure of processes. When considering sub-typing, this fact somehow forces us to abandon preorders in favor of an equivalence relation that respects P-sets membership, which leads to structural congruence. Further discussion on this point is found in Section 7.

The judgements derivable in this type system are written as $\Gamma \vdash_{\mathrm{L}} P : \mathsf{T}$.

Example 4. Consider the formula $\phi = \Box^*\neg\mathrm{H}^*(\overline{a}\,|\,\overline{b})$ saying that it is not possible to reach a configuration where both an output barb on a and one on b are available at the same time. Ok(ϕ) holds by Lemma 3. Consider the process $P = (\nu a, b : \mathsf{t}, \mathsf{t}; \phi)Q$, where: $\mathsf{t} = ()\mathbf{0}$, $Q = ((\overline{d}\langle a\rangle + \overline{d}\langle b\rangle)\,|\,!a.\overline{b}\,|\,!b.\overline{a})|d(x).\overline{x}$ and a context Γ s.t. $\Gamma \vdash d : (x : \mathsf{t})\overline{x} = \mathsf{t}'$. By applying the typing rules for input, output, summation and parallel composition:

$$\Gamma, a : \mathsf{t}, b : \mathsf{t} \vdash_{\mathrm{L}} Q \;\; : \;\; (\overline{d}.\overline{a} + \overline{d}.\overline{b})\,|\,!a(\mathsf{t}).\overline{b}\,|\,!b(\mathsf{t}).\overline{a}\,|\,d(\mathsf{t}') \;\triangleq\; \mathsf{T}\,.$$

$\mathsf{T} \downarrow_{a,b} = (\tau.\overline{a} + \tau.\overline{b})\,|\,!a(\mathsf{t}).\overline{b}\,|\,!b(\mathsf{t}).\overline{a}\,|\,\tau \models \phi$; hence, by (T-Res), $\Gamma \vdash_{\mathrm{L}} P : (\nu a, b : \mathsf{t}, \mathsf{t})\mathsf{T}$.

Basic properties. We state here the basic properties of the type system presented in the preceding subsection. Let us write $\Gamma \vdash_{\mathrm{NL}} P : \mathsf{T}$ if there exists a *normal* derivation of $\Gamma \vdash_{\mathrm{L}} P : \mathsf{T}$, that is, a derivation where rule (T-Eq) is used only above rule (T-Inp). Modulo $\equiv$, every judgment derivable in the type system admits a normal derivation.

Proposition 2 (normal derivation). *If $\Gamma \vdash_{\mathrm{L}} P : \mathsf{T}$ then $\Gamma \vdash_{\mathrm{NL}} P : \mathsf{S}$ for some $\mathsf{S} \equiv \mathsf{T}$.*

Normal derivations are syntax-driven, that is, processes and their types share the same shallow structure. This fact carries over to all derivations, modulo $\equiv$. E.g., if $\Gamma \vdash_{\mathrm{L}} P : \mathsf{T}, \mathsf{T} \equiv (\tilde{\nu}\tilde{a} : \tilde{\mathsf{t}})(\mathsf{T}_1|\mathsf{T}_2)$ then $P \equiv (\tilde{\nu}\tilde{a} : \tilde{\mathsf{t}}; \tilde{\Phi})(P_1|P_2)$, with $\Gamma, \tilde{a} : \tilde{\mathsf{t}} \vdash_{\mathrm{L}} P_i : \mathsf{T}_i$, $i = 1, 2$.

Theorem 1 (subject reduction). *$\Gamma \vdash_{\mathrm{L}} P : \mathsf{T}$ and $P \xrightarrow{\lambda} P'$ implies that there exists a T' such that $\mathsf{T} \xrightarrow{\lambda} \mathsf{T}'$ and $\Gamma \vdash_{\mathrm{L}} P' : \mathsf{T}'$.*

5 Type Soundness for the Local System

In this section we prove a general type soundness result for our system and provide a few interesting examples of application of the type systems.

Definitions and results. We identify the general class of properties for which, at least in principle, model checking on processes can be reduced to a type checking problem whose solution requires only a (local) use of model checking on types. We do so by the following coinductive definition.

Definition 7 (locally checkable properties and formulae). *We let* Lc *be the largest predicate on P-sets such that whenever* $\mathrm{Lc}(\Phi)$ *then* $\mathrm{Ok}(\Phi)$ *and: (1) whenever* $\Gamma \vdash_{\mathrm{L}} P : \mathsf{T}$ *and* $\tilde{x} \supseteq \mathrm{supp}(\Phi)$ *and* $\mathsf{T} \downarrow_{\tilde{x}} \models \Phi$ *then* $P \models \Phi$*; (2)* $\mathrm{Lc}(\Phi_\lambda)$ *holds for each* λ*. If* $\mathrm{Lc}(\Phi)$ *then we say* Φ *is* locally checkable*.*

A formula $\phi \in \mathcal{F}$ is said to be locally checkable if $[\![\phi]\!]$ is locally checkable.

Theorem 2 (run-time soundness). *Suppose that* $\Gamma \vdash_{\mathrm{L}} P : \mathsf{T}$ *and that P is decorated with locally checkable P-sets only. Then* $P \rightarrow^* P'$ *implies that* P' *is well-annotated.*

Our task is now providing sufficient syntactic conditions on formula ϕ that guarantee $\mathrm{Lc}([\![\phi]\!])$.

Lemma 4. *Suppose* $\Gamma \vdash_{\mathrm{L}} P : \mathsf{T}$*. (a) If* $\phi \in \mathcal{F}^-_{\tilde{x}}$ *and* $\mathsf{T} \downarrow_{\tilde{x}} \models \phi$ *then* $P \models \phi$*. (b) If* $\phi \in \mathcal{F}^+_{\tilde{x}}$ *and* $P \models \phi$ *then* $\mathsf{T} \downarrow_{\tilde{x}} \models \phi$*.*

Theorem 3. *Any negative formula of the form* $\Box^*\phi$ *is locally checkable.*

The above result automatically provides us a type soundness result for an interesting class of formulae, that include both safety and liveness properties.

Examples. The formulae *NoRace*(a) and *UniRec*(a) fits in the format given by Theorem 3, hence they are locally checkable. As an example, consider

$$P = (\nu a, b, c : ()\mathbf{0}, \mathsf{t}', \mathsf{t}\,;\ \mathit{UniRec}(a))((\overline{c}\langle a\rangle \mid a + b(x).x) \mid c(y).\overline{b}\langle y\rangle)$$

where $\mathsf{t} = (x)\overline{b}.x$ and $\mathsf{t}' = (y)y$. By the typing rules, we easily derive

$$\Gamma, a : ()\mathbf{0}, b : \mathsf{t}', c : \mathsf{t} \vdash_{\mathrm{L}} ((\overline{c}\langle a\rangle \mid a + b(x).x) \mid c(y).\overline{b}\langle y\rangle) : \mathsf{T}$$

with $\mathsf{T} \triangleq \overline{c}.\overline{b}.a \mid a + b(\mathsf{t}') \mid c(\mathsf{t})$. Since $\mathsf{T} \downarrow_{a,b,c} = \mathsf{T} \models \mathit{UniRec}(a)$, we can apply (T-Res) and get

$$\Gamma \vdash_{\mathrm{L}} P : (\nu a, b, c : ()\mathbf{0}, \mathsf{t}', \mathsf{t})\mathsf{T}\,.$$

For another example, consider the following access policy for a shared resource c. Before using the resource, a lock l must be acquired; the resource must then be used immediately, and the lock must be released immediately after that. If we identify an available resource c with an input barb on c, a use of c with a synchronization on c and

the availability of l with an output barb on l, the above policy can be described by the following formula, where $[c]$ stands for $\neg\langle c\rangle\neg$: $SafeLock(l,c) \triangleq \Box^*((\bar{l} \to c) \ \wedge \ [c]\bar{l})$. This is a negative formula fitting the format of Theorem 3, hence it is locally checkable. As an example of use of this formula, the process $Q = (\nu c, l; SafeLock(l,c))(\bar{l}|c|\overline{a}\langle l,c\rangle)\,|\,a(x,y).\,!x.(\overline{y}.(y|\overline{x}))$ is well typed under a Γ s.t. $\Gamma \vdash a : (x,y)!x.(\overline{y}.(y|\overline{x}))$. Note that neither (the analog of) $UniRec(a)$, nor $SafeLock(l,c)$ is included in the type soundness theorem of [10].

Finally, note that $Resp(a)$ and $DeadFree(a)$ do not fit the format of Theorem 3. Indeed, these formulae are not locally checkable. E.g., consider $R = (\nu a; Resp(a))(c.a|\overline{a})$. This process is easily seen to be well-typed under $c : ()\mathbf{0}$, simply because the c blocking a is masked (turned into τ) in (T-Res). However, $c.a|\overline{a}$ clearly fails to satisfy $Resp(a)$.

6 A "Global" Type System

The $Resp(a)$ example at the end of the preceding section makes it clear that it is not possible to achieve type soundness result for properties like responsiveness unless we drop the "locality" condition in the restriction rule. Indeed, those properties can only be checked if one can look at the part of the type involving names from which the restricted ones causally depend. In the previous example, where $\mathsf{T} = c.a|\overline{a}$, this means checking $Resp(a)$ against $\mathsf{T}\downarrow_{a,c} = \mathsf{T}$, rather than against $\mathsf{T}\downarrow_a$, thus detecting the failure of the property.

Below, we introduce a new type system that pursues this idea. Note that dropping locality implies some loss of compositionality and effectiveness. The type system relies on the use of dependency graphs, a technical device, introduced in the next subsection, which helps to individuate causal relations among names.

Dependency graphs. Let χ range over a set $\mathfrak{a} = \{\epsilon, \circ, \bullet\}$ of *annotations*. For $I \subseteq \mathcal{N}$, we let a set of annotated names $\hat{I}$ be a total function from I to $\mathfrak{a}$; by slight abuse of notation, we write $a^\chi \in \hat{I}$ rather than $\hat{I}(a) = \chi$. The informal meaning of annotations is: ϵ = free name, $\circ$ = input-bound name, $\bullet$ = restricted name. A *dependency graph* G is a pair $\langle V, E\rangle$, where: $V = \hat{I} \cup W$, with $W \subseteq \{(\nu\tilde{x}) \;\big|\; \tilde{x} \subseteq \mathcal{N}\}$, is a set of annotated names and restrictions representing *vertices*, and $E \subseteq V \times V$ is a set of *edges*.

A dependency graph $G = \langle V, E\rangle$, with $V = \hat{I} \cup W$ ranged over by $u, v, \ldots$, encodes causal relations among (free or bound) names in I. Vertices of the form $(\nu\tilde{x})$ are introduced for delimiting the scope of restrictions. Edges $(u,v) \in E$ are also written as $u \to_G v$; $\to_G^*$ is the reflexive and transitive closure of $\to_G$. A *root* of G is a vertex $u \in V$ such that for no v, $v \to_G u$; the set of G's roots is denoted by roots(G). Given a dependency graph $G = \langle V, E\rangle$, with $V = \hat{I} \cup W$, a name a is *critical in G with respect to $\tilde{b}$*, if it belongs to the set of names $G(\tilde{b})$ defined below.

$$G(\tilde{b}) \triangleq \{x \;\big|\; x^\epsilon \in \hat{I} \wedge \exists x \to_G v_1 \to_G \cdots \to_G v_n = b \in \tilde{b} \text{ s.t. } \forall 1 \le i < n : \; v_i = (\nu\tilde{y}) \text{ implies } b \notin \tilde{y}\}.$$

The set of *critical names* in G, written cr(G), is defined as $\mathrm{cr}(G) \triangleq \bigcup_{b^\bullet \in \hat{I}} G(b)$. Finally, we define $G[\tilde{b}] \triangleq G(\tilde{b}) \cup \tilde{b}$.

In order to define dependency graphs associated to types, we introduce three auxiliary operations on graphs: (i) union $G_1 \cup G_2$ is defined componentwise as expected, provided the sets of vertices V_1 and V_2 agree on annotations (otherwise union is not defined); (ii) χ-update $G \uparrow^{\chi}_{\tilde{x}}$ changes into χ the annotation of all names in $\tilde{x}$ occurring in V; (iii) a-rooting is defined as $a \to G \triangleq \langle\, V \cup \{a^{\epsilon}\}, E \cup \{(a,b) \mid b \in \mathrm{roots}(G)\} \,\rangle$, where $G = \langle V, E \rangle$, provided a does not occur in V with annotations different from ϵ (otherwise a-rooting is not defined); (iv) $(\nu\tilde{x})$-rooting is defined as $(\nu\tilde{x}) \to G \triangleq \langle\, V, E \cup \{((\nu\tilde{x}), b) \mid b \in \mathrm{roots}(G)\} \,\rangle$. Dependency graphs are inductively defined over types in *normal form*. Let us say a type is *prime* if it is of the form either $\sum_{i\in I} \mu_i.\mathsf{T}_i$ with $I \neq \emptyset$ or $!a(\mathsf{t}).\mathsf{T}$. Let us say a type is in *head normal form* if it is of the form $(\tilde{\nu}\tilde{a})(\mathsf{T}_1 | \cdots | \mathsf{T}_n)$ with the T_i's prime and the prefix continuations are recursively in *normal form* if the T_i's are recursively in normal form. Similar definition for processes. For any T and t in normal form, the dependency graphs G_{T} and G_{t} are defined by mutual induction on the structure of T and t as follows (it is assumed that in T and t bound names are distinct from each other and from free names).

$$G_{\overline{a}.\mathsf{T}} = a \to G_{\mathsf{T}} \qquad G_{a(\mathsf{t}).\mathsf{T}} = a \to (G_{\mathsf{t}} \cup G_{\mathsf{T}}) \qquad G_{!a(\mathsf{t}).\mathsf{T}} = G_{a(\mathsf{t}).\mathsf{T}}$$

$$G_{\sum_{i\in I}\mu_i.\mathsf{T}_i} = \bigcup_{i\in I} G_{\mu_i.\mathsf{T}_i} \quad |I| \neq 1 \qquad\qquad G_{\prod_i \mathsf{T}_i} = \bigcup_i G_{\mathsf{T}_i}$$

$$G_{(\nu\tilde{x}:\tilde{\mathsf{t}})\mathsf{T}} = (\nu\tilde{x}) \to ((G_{\mathsf{T}} \cup \bigcup_{\mathsf{t}\in\tilde{\mathsf{t}}} G_{\mathsf{t}}) \uparrow^{\bullet}_{\tilde{x}}) \qquad G_{(\tilde{x}:\tilde{\mathsf{t}})\mathsf{T}} = (G_{\mathsf{T}} \cup \bigcup_{\mathsf{t}\in\tilde{\mathsf{t}}} G_{\mathsf{t}}) \uparrow^{\circ}_{\tilde{x}} \ .$$

In essence, G_{T} encodes potential causal dependencies among (free or bound) names of T as determined by prefixes in T. In the sequel, we shall abbreviate $\mathrm{cr}(G_{\mathsf{T}})$ and $G_{\mathsf{T}}[\tilde{b}]$, for some $\tilde{b} \subseteq \mathrm{fn}(\mathsf{T})$, as $\mathrm{cr}(\mathsf{T})$ and $\mathsf{T}[\tilde{b}]$, respectively.

Typing rules. We need some additional notations. A channel type $(\tilde{x})\mathsf{T}$ is said to be *well-formed* if $\tilde{x}\#\mathrm{cr}(\mathsf{T})$; in what follows, we only consider contexts Γ containing well-formed channel types. For any type T we let $\mathsf{T} \Downarrow_{\tilde{x}}$ denote $\mathsf{T} \downarrow_{\mathsf{T}[\tilde{x}]}$ (note that $\mathrm{fn}(\mathsf{T} \Downarrow_{\tilde{x}}) = \mathsf{T}[\tilde{x}]$ by definition). Intuitively, in $\mathsf{T} \Downarrow_{\tilde{x}}$, we keep the names in $\tilde{x}$ and those that are causes of $\tilde{x}$ in T; the others are masked. We also need a more permissive notion of well-annotated process, that allows re-arranging of top-level restrictions before checking annotations (property). To see why this is necessary, consider $\phi = \Box^*(\Diamond^* a | \Diamond^* a)$, a typical property one would like to check in the new system. Consider the processes $P = (\nu b)(\nu a; \phi)R$ and $Q = (\nu a; \phi)(\nu b)R$, with $R = b.\overline{c} \,|\, b.\overline{d} \,|\, \overline{b} \,|\, \overline{b} \,|\, c.a \,|\, d.a$. We observe that $(\nu b)R \not\models \phi$, so that Q is not well-annotated according to Definition 5; on the other hand, $Q \equiv P$ and $R \models \phi$, which suggests that P, hence Q, could be considered as well-annotated up to a swapping of (νa) and (νb).

Definition 8 (globally well-annotated processes). *A process $P \in \mathcal{P}$ is* globally well-annotated *if whenever $P \equiv (\tilde{\nu}\tilde{b})(\nu\tilde{a}; \Phi)(\tilde{\nu}\tilde{c})Q$, with Q a parallel composition of prime processes, then there is a permutation $\tilde{b}'\,\tilde{c}'$ of $\tilde{b}\,\tilde{c}$ such that $P \equiv (\tilde{\nu}\tilde{b}')(\nu\tilde{a}; \Phi)(\tilde{\nu}\tilde{c}')Q$ and $(\tilde{\nu}\tilde{c}')Q \models \Phi$.*

The global type system is obtained by replacing some rules of the local one (Table 4) with those reported in Table 5. The type system makes use of an auxiliary relation $\propto_{\tilde{x}}$ among P-sets and types, defined coinductively as follows (the use of this relation is explained in the sequel).

$$\text{(T-Res)}\ \frac{\Gamma,\tilde{a}:\tilde{\mathsf{t}}\vdash P:\mathsf{T}\quad \Phi\propto_{\tilde{a}}\mathsf{T}}{\Gamma\vdash(\nu\tilde{a}:\tilde{\mathsf{t}};\Phi)P:(\nu\tilde{a}:\tilde{\mathsf{t}})\mathsf{T}}\qquad \text{(T-Eq-P)}\ \frac{\Gamma\vdash P:\mathsf{T}\quad P\equiv Q}{\Gamma\vdash Q:\mathsf{T}}$$

$$\text{(T-Rep)}\ \frac{\Gamma\vdash P:\mathsf{T}\ \ \mathrm{cr}(\mathsf{T})=\emptyset}{\Gamma\vdash !P:!\mathsf{T}}\qquad \text{(T-Par)}\ \frac{\Gamma\vdash P:\mathsf{T}\quad \Gamma\vdash Q:\mathsf{S}\quad \mathrm{cr}(\mathsf{T})\#\mathsf{S}\ \ \mathrm{cr}(\mathsf{S})\#\mathsf{T}}{\Gamma\vdash P|Q:\mathsf{T}|\mathsf{S}}$$

$$\text{(T-Out)}\ \frac{\Gamma\vdash a:(\tilde{x}:\tilde{\mathsf{t}})\mathsf{T}\quad \Gamma\vdash\tilde{b}:\tilde{\mathsf{t}}\quad \Gamma\vdash P:\mathsf{S}\quad \tilde{b}\#\mathrm{cr}(\mathsf{T})\ \ \mathrm{cr}(\mathsf{T}[\tilde{b}/\tilde{x}])\#\mathsf{S}\ \ \mathsf{T}[\tilde{b}/\tilde{x}]\#\mathrm{cr}(\mathsf{S})}{\Gamma\vdash\overline{a}\langle\tilde{b}\rangle.P:\overline{a}.(\mathsf{T}[\tilde{b}/\tilde{x}]\,|\,\mathsf{S})}$$

Table 5. Typing rules.

Definition 9 ($\propto_{\tilde{x}}$). *We let $\propto_{\tilde{x}}$ be the largest relation on P-sets and types such that whenever $\Phi\propto_{\tilde{x}}\mathsf{T}$ then: (1) $\mathsf{T}\downarrow_{\mathsf{T}[\tilde{x}]}\models\Phi$; (2) for each λ,T' such that $\mathsf{T}\downarrow_{\mathsf{T}[\tilde{x}]}\xrightarrow{\lambda}\mathsf{T}'\downarrow_{\mathsf{T}[\tilde{x}]}$ then $\Phi_\lambda\propto_{\tilde{x}}\mathsf{T}'$.*

Note the presence of a new structural rule for processes, (T-Eq-P) forcing subject congruence, which is not derivable from the other rules of the system. As an example, while $P=(\nu a:\mathsf{t};Resp(a))(b.a|\overline{b}|\overline{a})$ can be typed without using rule (T-Eq-P), the structurally congruent process $(\nu a:\mathsf{t};Resp(a))(b.a|\overline{a})|\overline{b}$ *could not* be typed without using that rule. The condition on critical names in rule (T-Par) ensures that any Q put in parallel to P will not break well-annotated-ness of P (and vice-versa). A similar remark applies to the rules for output and replication. In rule (T-Res), use of the relation $\propto_{\tilde{a}}$ ensures that each derivative of T satisfies the corresponding derivative of Φ. It is worth noticing that checking $\Phi\propto_{\tilde{a}}\mathsf{T}$ could be undecidable, given that in general we are in the presence of infinite state systems: at the end of the next section, we will identify a class of formulas for which checking $[\![\phi]\!]\propto_{\tilde{a}}\mathsf{T}$ reduces to checking $\mathsf{T}\Downarrow_{\tilde{a}}\models\phi$. The judgements derivable in the new type system are written as $\Gamma\vdash_{\mathrm{G}}P:\mathsf{T}$. It is worth noticing that the system introduced in Table 5 is not syntax-driven, but a syntax-directed version can be easily defined by adding some constraints on the structure of processes in the premises of the typing rule for parallel composition (we omit the details for lack of space).

Type Soundness. Similarly to the local case, we identify a general class of properties for which, at least in principle, model checking on processes can be reduced to a type checking problem whose solution requires only model checking on types, then give sufficient syntactic conditions for global-checkable-ness. The definition of *globally checkable property* (omitted) is the same as the local one, except that the local hiding operator "$\downarrow_{\tilde{x}}$" is replaced by "$\Downarrow_{\tilde{x}}$".

Theorem 4 (run-time soundness). *Suppose that $\Gamma\vdash_{\mathrm{G}}P:\mathsf{T}$ and that P is decorated with globally checkable P-sets only. Then $P\rightarrow^*P'$ implies that P' is globally well-annotated.*

Like in the local case, we can give syntactic conditions for a formula to be globally checkable.

Theorem 5. *Suppose ϕ is of the form: (a) $\Box^*\psi$ with negation not occurring underneath any $\langle-\tilde{y}\rangle^*$ in ψ; or (b) $\Box^*_{-\tilde{y}}\Diamond^*\psi'$, with negation not occurring in ψ'. Then ϕ is globally checkable.*

The following proposition guarantees that for formulas that satisfy the premises of Theorem 5 checking $[\![\phi]\!] \propto_{\tilde{a}} \mathsf{T}$ reduces to checking $\mathsf{T} \Downarrow_{\tilde{a}} \models \phi$.

Proposition 3. *Suppose $\phi \in \mathcal{F}_{\tilde{x}}$ is of the form of the form (a) and (b) as specified in Theorem 5. If $\mathsf{T} \Downarrow_{\tilde{x}} \models \phi$ then $[\![\phi]\!] \propto_{\tilde{x}} \mathsf{T}$.*

Examples. All properties defined in Example 1 fit the format of Theorem 5, hence they are globally checkable. As an example, consider $P = (\nu a : Resp(a))(\overline{c}\langle a\rangle)|Q$, where $Q = !c(x).(\overline{x}|x)|\overline{c}\langle b\rangle$. Under a suitable Γ, we derive $\Gamma \vdash_{\mathrm{G}} \overline{c}\langle a\rangle|Q : \overline{c}.(\overline{a}|a)|!c|\overline{c}.(\overline{b}|b) \triangleq \mathsf{T}$. Since $\mathsf{T} \downarrow_{\mathsf{T}[a]} = \overline{c}.(\overline{a}|a)|!c|\overline{c}.(\tau|\tau) \models Resp(a)$, by (T-Res), we get $\Gamma \vdash_{\mathrm{G}} (\nu a : Resp(a))(\overline{c}\langle a\rangle|Q) : (\nu a)\mathsf{T}$, hence we can conclude that $\Gamma \vdash_{\mathrm{G}} P : (\nu a)\mathsf{T}$ using (T-Eq-P).

It is worth to notice that (the analogs of) responsiveness and deadlock freedom escape the type soundness theorem of [10], although, for deadlock freedom, a soundness result can still be proven by ad-hoc reasoning on certain basic properties of the system.

7 Discussion

We discuss here some limits, and possible workarounds, of our approach, and contrast them with the generic type system approach of [10]. In [10], the subtyping relation makes an essential use of a "sub-divide" law, $\mathsf{T} \equiv \mathsf{T} \uparrow_{\tilde{x}} | \mathsf{T} \downarrow_{\tilde{x}}$. This rule allows one to split *any* type into a part depending only on $\tilde{x}$, $\mathsf{T} \downarrow_{\tilde{x}}$, and a part not depending on $\tilde{x}$, $\mathsf{T} \uparrow_{\tilde{x}}$. As an example, with this law one has $a.b.\overline{x} \equiv a.b.\tau|\tau.\tau.\overline{x}$. This law enhances the flexibility of the input rule, hence of the type system. On the other hand, it disregards the spatial properties of terms, leading to a lack of structural correspondence between types and processes. In our system, we stick to spatial-preserving laws, thus trading off some flexibility for precision. As seen, this gain in precision has influential consequences on the class of properties for which type soundness can be proven (e.g., the class includes interesting liveness properties). An example of process that cannot be treated in our type systems because of the absence of the "sub-divide" law is the process $Q = !a(x).(\nu c)\Big(b(y).\big((\nu z)(\overline{c}\langle x,z\rangle|z.\overline{y})|c(x,z).(\overline{x}|\overline{z})\big)\Big)$. Here, a can be viewed as an invocation channel, x as a formal invocation parameter and y as an acknowledgement channel, introduced by another input (on b). It appears that y and x are related (via c), which makes the type of b dependent on the bound name x, which cannot be expressed in our system. This dependency could be discarder using the sub-divide law. In the example, the very dependency of y from x suggests a way to re-write the process into a conceptually equivalent one that can be dealt with in our systems. E.g., $!a(x,y).(\nu c)\Big((\nu z)(\overline{c}\langle x,z\rangle|z.\overline{y})|c(x,z).(\overline{x}|\overline{z})\Big)$.

8 Conclusion, further and related work

We have provided a framework that incorporates ideas from both spatial logics and behavioural type systems, drawing benefits from both. Implementation issues are not in the focus of this paper. In this respect, the normal derivation property already provides us with syntax driven systems. Of course, implementing the model checks $\mathsf{T} \models \phi$

is an issue. One possibility would be re-using existing work on spatial model checking: Caires' work [6] seems to be a promising starting point. Also, approximations of possibly infinite-state ccs types with Petri Nets, or even finite-state automata, in the vein of [12], seem unavoidable to obtain effective tools. Finally, it would be interesting to cast our approach in more applicative scenarios, like calculi for service-oriented computing [1].

Apart from the already cited works, also related to our approach are some recent proposals by Caires. In [5,4], a logical semantics approach to types for concurrency is pursued. Closest to our work is [4], where a generic type system for the pi-calculus - parameterized on the subtyping relation - is proposed. The author identifies a family of types, the so called shared types, which allow to modularly and safely compose spatial and shared (classical invariants) properties and to safely factorize spatial properties. A preliminary investigation of the ideas presented in this paper, in a much simpler setting, is in [2].

References

1. Acciai, L., Boreale, M.: A type system for client progress in a service-oriented calculus. Degano, P. et al. (eds.) Montanari Festschrift, LNCS, vol. 5065, pp. 642–658 (2008)
2. Acciai, L., Boreale, M.: Type abstractions of name-passing processes. Arbab, F. and Sirjani, M. (eds.) FSEN'07. LNCS, vol. 4767, pp. 302–317 (2007)
3. Acciai, L., Boreale, M.: Responsiveness in process calculi. Okada, M. and Satoh, I. (eds.) ASIAN'06. LNCS, vol. 4435, pp. 136–150 (2008)
4. Caires, L.: Logical Semantics of Types for Concurrency. In: Mossakowski, T., Montanari, U., Haveraaen, M. (eds.) CALCO'07. LNCS, vol. 4624, pp. 16–35 (2007)
5. Caires, L.: Spatial-Behavioral Types, Distributed Services, and Resources. In: Montanari, U., Sannella, D., Bruni, R. (eds.) TGC'06. LNCS, vol. 4661, pp. 98–115 (2007)
6. Caires, L.: Behavioral and Spatial Observations in a Logic for the pi-Calculus. In: Walukiewicz, I. (eds) FoSSaCS'04. LNCS, vol. 2987, pp. 72–89 (2004)
7. Caires, L., Cardelli, L.: A spatial logic for concurrency (part I). Inf. Comput. 186(2), 194–235 (2003)
8. Chaki, S., Rajamani, S.K., Rehof, J.: Types as models: model checking message-passing programs. POPL'02, pp. 45–57 (2002)
9. Cardelli, L., Gordon, A.D.: Anytime, Anywhere: Modal Logics for Mobile Ambients. POPL 2000, pp. 365–377 (2000)
10. Igarashi, A., Kobayashi, N.: A generic type system for the Pi-calculus. Theor. Comput. Sci. 311(1-3), 121–163 (2004)
11. Kobayashi, N.: Type-based information flow analysis for the pi-calculus. Acta Inf. 42(4-5), 291–347 (2005)
12. Kobayashi, N., Suenaga, K., Wischik, L.: Resource Usage Analysis for the pi-Calculus. Logical Methods in Computer Science 2(3) (2006)
13. Kobayashi, N., Suto, T.: Undecidability of 2-Label BPP Equivalences and Behavioral Type Systems for the pi -Calculus. In: Arge, L., Cachin, C., Jurdzinski, T., Tarlecki, A. (eds) ICALP'07. LNCS, vol. 4596, pp. 740-751 (2007)
14. Milner, R.: The polyadic π-calculus: a tutorial. In Logic and Algebra of Specification, Springer, pp. 203–246 (1993)
15. Milner, R.: A Calculus of Communicating Systems. Springer (1980)
16. Sangiorgi, D.: The Name Discipline of Uniform Receptiveness. Theoretical Computer Science 221(1-2), 457–493 (1999)