SPATIAL TYPES FOR CONCURRENCY
A Spatial Logic to Specify and Verify Distributed Systems

(EXTENDED ABSTRACT)

Tiago Carvalho*
SQIG, Dep. Mathematics, IST – UTL Lisbon, Portugal

October, 2007

1 Introduction

The problem of specifying and verifying properties of computational systems is considered to be a classical problem in computer science. Herein the focus is on distributed systems: concurrent systems where behaviour is spatially distributed. Some examples of properties, inherently spatial, of these systems are for instance *connectivity*, stating that there exists an access route between two different sites, *unique handling*, stating that there is at most one server process listening on a given channel, and *resource availability*, stating that a bound exists on the number of channels that can be allocated at a given location.

This work aims to answer the following question. Is it possible to check behavioural and spatial properties of distributed systems in an efficient way?

Regarding the problem of specification, several spatial logics – modal logics with modalities for observing spatial structure – have been proposed previously. Some examples are those of Cardelli and Gordon – *The Ambient Logic* [LG00] – and of Caires and Cardelli – *A Spatial Logic for Concurrency* [CC03]. However, these have proof systems which are, in general, undecidable; an exception is the model-checker of Luís Caires [Cai03].

Traditionally, type systems are used to guarantee type safety, *i.e.* the absence of errors in programming languages, due to being decidable and to their low complexity. Recently, they have also been used to ensure spatial properties like those referred above.

We define a logic (syntax, semantics and deductive system), which allows both spatial and behavioural properties of concurrent processes to be specified and verified. We adopt a "propositions as types" approach, where types are formulas, denotational semantics is established based on certain structural and transition relations, and subtyping is interpreted as logical entailment. Furthermore, a type system is defined as a deductive system and some results are proved about it, namely weak consistency and completeness.

To model processes we consider the non-deterministic choice free fragment of Milner's CCS. Namely, there is the inactive process $\mathbf{0}$, action prefixing $\pi.P$ for sequential behaviour, parallel composition $P \mid Q$ to express concurrency, restricted names $(\nu a)P$ to restrain the scope of a name to a process and recursion $(\mathbf{rec}X.P)$ to express infinitary behaviour. To complete the setting of processes, we provide operational semantics via two different relations: a static one – the *structural congruence relation* – and a dynamic one – a *labelled transition relation*.

A Motivating Application

The purpose of this application is to exemplify a functionality of the system. The aim is to prove a spatial invariant about a given process.

*This work was supported by the Space-Time-Types Project, POSC/EIA/55582/2004, and by SQIG at IT (former CLC).

Consider the following process

$$Looper \stackrel{\text{def}}{=} (\nu a)(a \mid (\mathbf{rec} X.\bar{a}.(a \mid a \mid X))).$$

Looper is a process with no observable behaviour. In fact it can only perform internal transitions and for each transition, it creates a new copy of a private name (name a). The important feature of this process is that it has always more than one thread running in parallel, *i.e.* it is not a sequential process, and that is the property we want to capture.

More precisely, we want to infer in the system the following typing judgment

$$\Gamma \vdash Looper : \Box(\bar{\mathbf{0}} \mid \bar{\mathbf{0}})$$

where type $\Box(\bar{\mathbf{0}} \mid \bar{\mathbf{0}})$ specifies that *Looper* will always have at least two separate threads running in parallel.

2 Type System

Syntax

The type language is largely inspired in the *Spatial Logic* of [CC03] and in the *Process Logic* described in [Mil89]. Some type operators are closely related with the Spatial Logic operators and others, the modal and propositional ones, are taken from Milner's Process Logic. The former aspire to express spatial properties while the latter are meant to express behaviour.

Consider a countable set of names, $\mathcal{A}$, ranged over by $a, b, c, ...$ and let $\overline{\mathcal{A}}$ denote the set of co-names, ranged over by $\bar{a}, \bar{b}, \bar{c}, ...$ Set $\mathcal{L} \triangleq \mathcal{A} \cup \overline{\mathcal{A}}$, where $\mathcal{L}$ is the set of labels, and let $Act \triangleq \mathcal{L} \cup \{\tau\}$ where τ is the silent or invisible action. As in CCS, we say that $\bar{a}$ is the *complement* of a and we extend complementation to all actions by defining $\bar{\bar{a}} = a$ and $\bar{\tau} = \tau$. Further, consider a countable set of *type variables*, $\mathcal{Z}$, disjoint from these.

Definition 2.1. (Syntax of types) *Let I be a nonempty countable index set. The following grammar defines the set $\mathcal{T}$ of* types.

A	$::=$	$\mathbf{0}$	*inaction*
	$\mid$	$\pi.A$	*prefix*
	$\mid$	$A \mid A$	*parallel composition*
	$\mid$	$a\circledR A$	*restriction*
	$\mid$	$(\mathbf{rec}\mathcal{Z}.A)$	*recursion*
	$\mid$	$\mathcal{Z}$	*type variable*
	$\mid$	$\bar{\mathbf{0}}$	*active*
	$\mid$	$[\pi]\, A$	*if* π
	$\mid$	$\langle\pi\rangle\, A$	*may* π
	$\mid$	$\bigwedge_{i\in I} A_i$	*conjunction*
	$\mid$	$\bigvee_{i\in I} A_i$	*disjunction*

Every operator in the calculus of processes has a counterpart among types: inaction, prefix, parallel composition and recursion combinators are also defined within types. Restriction of names in types is also defined, though with a different notation taken from [CC03] – $a\circledR A$. We call these type combinators the *structural operators*. We have the active type $\bar{\mathbf{0}}$, the dual of $\mathbf{0}$; two modalities: $[\pi]\, A$, to express the type of a process after performing any transition and its dual, $\langle\pi\rangle\, A$, to express the possibility of transition by a process and its type after that transition; conjunction and disjunction to enable the definition of more expressive modalities.

Semantics

The semantics of types is defined by assigning to each type A a set of processes, namely the set of processes which satisfy the property denoted by A. With this purpose, we introduce a binary relation between processes and types, named *satisfaction*, which induces the mentioned set. But before formally defining

satisfaction we need to mention the notion of *environment*, denoted by Γ, needed to cater for free variables within the body of recursion, which is an injective function mapping process variables into type variables. *Satisfaction*, denoted by $\models_\Gamma$, is formalized in the following definition.

Definition 2.2. (Interpretation of Types) *The satisfaction relation, $\models_\Gamma$, is inductively defined over types by the following rules:*

Voi $P \models_\Gamma \mathbf{0}$, *if* $P \stackrel{\pi}{\not\longrightarrow}$, *for every* π;

Non $P \models_\Gamma \bar{\mathbf{0}}$, *if* $P \stackrel{\pi}{\longrightarrow} P'$, *for some* π *and* P';

Var $P \models_\Gamma \mathcal{Z}$, *if* P *is a variable* X *and* $X : \mathcal{Z}$ *is in* Γ;

Par $P \models_\Gamma A \mid B$, *if* $P \equiv Q \mid R, Q \models_\Gamma A$ *and* $R \models_\Gamma B$, *for some* Q *and* R;

Pre $P \models_\Gamma \pi.A$, *if for some* P' $P \stackrel{\pi}{\longrightarrow} P'$ *and* $P' \models_\Gamma A$, *and, for every* π' *and* P'', *if* $P \stackrel{\pi'}{\longrightarrow} P''$ *then* π' *is* π *and,* P'' *and* P' *are weakly bisimilar*;

New $P \models_\Gamma a\circledR A$, *if* $P \equiv (\nu a)Q$ *and* $Q \models_\Gamma A$, *for some* Q;

Rec $P \models_\Gamma (\mathbf{rec}\mathcal{Z}.A)$, *if* $P \equiv (\mathbf{rec}X.P')$ *and* $P' \models_{\Gamma,\{X:\mathcal{Z}\}} A'$, *for some* X *and* P';

Dia $P \models_\Gamma \langle\pi\rangle A$, *if* $P \stackrel{\pi}{\longrightarrow} P'$ *and* $P' \models_\Gamma A$, *for some* P';

Box $P \models_\Gamma [\pi] A$, *if* $P \models_\Gamma \pi.A$ *or* $P \stackrel{\pi}{\not\longrightarrow}$;

Con $P \models_\Gamma \bigwedge_{i\in I} A_i$, *if* $P \models_\Gamma A_i$ *for all* $i \in I$;

Dis $P \models_\Gamma \bigvee_{i\in I} A_i$, *if* $P \models_\Gamma A_i$ *for some* $i \in I$.

It follows a brief explanation of the relation defined above.

- $\mathbf{0}$ is satisfied by every process that can not perform any transition, for instance $\mathbf{0}$ and $(\nu a)a.\mathbf{0}$;
- $\bar{\mathbf{0}}$, the dual of $\mathbf{0}$, is satisfied by every process that can perform an action;
- a type variable $\mathcal{Z}$ is exactly satisfied by the process variable given by the environment Γ;
- $A \mid B$ is satisfied by every process that can be spatially decomposed in two parts, provided each part satisfies A and B, respectively;
- $\pi.A$ is satisfied by every process which has a π-derivative P' that satisfies A and such that every π-derivative of it is bisimilar to P';
- $a\circledR A$ is satisfied by every process which can be transformed (using structural congruence) into a process restricted by a and that the scope of the restriction satisfies A;
- $(\mathbf{rec}\mathcal{Z}.A)$ is satisfied by every process which can be transformed into a process recursion, the body of the recursion satisfies A and that the variable of the recursion satisfies $\mathcal{Z}$;
- $\langle\pi\rangle A$ is satisfied by every process having a π-derivative that satisfies A;
- $[\pi] A$ is satisfied by every process such that either every π-derivative satisfies A or does not have any π-derivative;
- $P \models_\Gamma \bigwedge_{i\in I} A_i$ is satisfied by every process that satisfies A_i for every $i \in I$;
- $P \models_\Gamma \bigvee_{i\in I} A_i$ is satisfied by every process that satisfies A_i for some $i \in I$;

The use of both structural congruence and labelled transition in the definition of satisfaction enables the expressiveness to talk about spatial properties and behavioural properties. Let us now introduce some derived forms for the system. These will enable other connectives to be expressed, which subsequently will make its power more apparent. First, it is more convenient to subscribe a conjunction or a disjunction by a condition different from $i \in I$. Thus, it becomes easier and more intuitive to define the following logical operators - modal operators.

Definition 2.3. (Abbreviations)

$$\begin{array}{lll} \langle t \rangle A \stackrel{\text{abv}}{=} \langle \pi_1 \dots \pi_n \rangle A \stackrel{\text{abv}}{=} \langle \pi_1 \rangle \dots \langle \pi_n \rangle A, \textit{ where } n \geq 1 \textit{ and } t = \pi_1 \dots \pi_n; \\ [t] A \stackrel{\text{abv}}{=} [\pi_1 \dots \pi_n] A \stackrel{\text{abv}}{=} [\pi_1] \dots [\pi_n] A, \textit{ where } n \geq 1 \textit{ and } t = \pi_1 \dots \pi_n; \\ \Diamond A \stackrel{\text{abv}}{=} \bigvee_{t \in Act^*} \langle t \rangle A; \\ \Box A \stackrel{\text{abv}}{=} \bigwedge_{t \in Act^*} [t] A; \\ !A \stackrel{\text{abv}}{=} (\textbf{\textit{rec}} \mathcal{Z}.A \mid \mathcal{Z}). \end{array}$$

Again, it follows a brief explanation of these operators.

- $\langle t \rangle A$, read as *possible t-A*, is satisfied by every process which has a t-descendant that satisfies A;
- $[t] A$, read as *necessarily t-A*, is satisfied by every process such that every t-descendant satisfies A;
- $\Diamond A$, read as *some time* A, is satisfied by every process which has a t-descendant that satisfies A, for some $t \in Act*$;
- $\Box A$, read as *always* A, is satisfied by every process such that every t-descendant satisfies A, for every $t \in Act*$;
- $!A$, read as *iterate* A, is satisfied by every process recursion with $A \mid \mathcal{Z}$ in the body of the recursion.

Deductive System

Judgments are of the form $\Gamma \vdash P : A$, where P is a process, A is a type and Γ is an environment written in the following form $X_1 : \mathcal{Z}_1, \dots, X_n : \mathcal{Z}_n$. Moreover, $\Gamma, X : \mathcal{Z}$ denotes the disjoint union of Γ and $\{X : \mathcal{Z}\}$, where $\mathcal{Z} \notin \text{codom}(\Gamma)$.

The deductive system is inductively defined through a few inference rules:

- A collection of structural rules;
- A subsumption rule;
- A conjunction rule.

The structural inference rules are those listed on table 1. There is exactly one rule for each connective of the process calculus; these rules build a correspondence between the calculus operators and the structural type operators. Rule **T-Voi** states that the process **0** has type **0**; rule **T-Var** uses the environment $\Gamma, X : \mathcal{Z}$ to determine the type of the process variable X; in rules **T-Par**, **T-Pre** and **T-New** a type of a composite process can be inferred from its components with the same environment; rule **T-Rec** states that recursive processes have recursive types which can be inferred from the type of the body of the recursion, extending the environment to cope with the variables that become free.

The structural typing rules introduced are insufficient to infer interesting properties about processes. Therefore, we introduce a binary relation over types which brings inference power and expressiveness to the system.

Definition 2.4. (Subtyping) Subtyping, $<:$, *is the smallest binary relation over types such that, for every types* A *and* B, $A <: B$ *if, for every process* P *and environment* Γ *such that* $P \models_\Gamma A$, *there exists a process* Q *such that* $P <: Q$ *and* $Q \models_\Gamma B$.
We say that A *is a* subtype *of* B *if* $A <: B$.

$$\frac{}{\Gamma \vdash \mathbf{0} : \mathbf{0}}\ \text{T-Voi} \qquad \frac{}{\Gamma, X : \mathcal{Z} \vdash X : \mathcal{Z}}\ \text{T-Var}$$

$$\frac{\Gamma \vdash P : A \quad \Gamma \vdash Q : B}{\Gamma \vdash P \mid Q : A \mid B}\ \text{T-Par}$$

$$\frac{\Gamma \vdash P : A}{\Gamma \vdash \pi.P : \pi.A}\ \text{T-Pre} \qquad \frac{\Gamma \vdash P : A}{\Gamma \vdash (\nu a)P : a\circledR A}\ \text{T-New}$$

$$\frac{\Gamma, X : \mathcal{Z} \vdash P : B}{\Gamma \vdash (\mathbf{rec} X.P) : (\mathbf{rec}\mathcal{Z}.B)}\ \text{T-Rec}$$

Table 1: The Structural Type System.

A simple intuition is to read $A <: B$ as logical entailment to a certain extent, that is, "if a process P has type A, then it is weakly simulated by a process Q with type B".

The following definition introduces the *subsumption rule*, which builds a bridge between typing and subtyping.

Definition 2.5. (Subsumption rule)

$$\frac{\Gamma \vdash P : A \quad A <: B}{\Gamma \vdash P : B},\ \textit{with } A \not<:\!\!> \mathbf{0} \textit{ or } B \neq \bar{\mathbf{0}} \qquad \textit{T-Sub}$$

This rule states that, if P has type A and A is a subtype of B, then P also has type B.

The *Subtyping System* consists of all the subtyping rules and lemmas, that is, the rules and the two lemmas listed in table 2. Therefore, *valid subtyping judgments* are those which can be inferred from these rules and a *subtyping derivation* is a proof of the validity of a certain subtyping judgment.

It is through the subsumption rule and subtyping that most of the non-structural operators are introduced in the deductive system. For instance, there is a subtyping rule to infer the disjunction of types. Contrary to disjunction, conjunction can not be inferred through subtyping. Therefore it must be introduced in the system by a separate rule.

Definition 2.6. (Conjunction rule)

$$\frac{\Gamma \vdash P : A_i \textit{ for all } i \in I}{\Gamma \vdash P : \bigwedge_{i \in I} A_i}\ \textit{T-Con}$$

Also it should be mentioned that, unlike for all the other connectives, it was not proved thus far that subtyping is not preserved by conjunction. This fact raises some consequences, namely regarding the preservation of subtyping by the necessity operator, as this is defined through conjunction. This issue will be further discussed in the following.

The *Type System* consists of all the structural rules, the subsumption rule and the conjunction rule. Therefore, *valid type judgments* are those which can be inferred from these rules and a *typing derivation* is a proof of the validity of a certain type judgment.

Some Properties

The system is proven to have certain interesting properties. First, unsurprisingly, every process has a type derivable structurally, *i.e.* derived using only structural rules.

Lemma 2.9. *If $\Gamma \vdash P : A$, then there is B such that $B <: A$ and $\Gamma \vdash P : B$ is derived using a structural type rule or the conjunction rule in the final step of the derivation.*

$A <: A$

$$\frac{A <: B \quad B <: C}{A <: C}$$

$$\frac{A <: B}{\pi.A <: \pi.B}$$

$$\frac{A <: B}{A \mid C <: B \mid C}$$

$$\frac{A <: B}{a\circledR A <: a\circledR B}$$

$$\frac{A <: B}{(\mathbf{rec}\mathcal{Z}.A) <: (\mathbf{rec}\mathcal{Z}.B)}$$

$$\frac{A <: B}{[\pi]\,A <: [\pi]\,B}$$

$$\frac{A <: B}{\langle\pi\rangle\,A <: \langle\pi\rangle\,B}$$

$$\frac{A_j <: B_j}{\bigvee_{i\in I} A_i <: \bigvee_{i\in I} B_i}, \text{ if } j \in I$$

$a\circledR A <:> b\circledR(A\{b/a\})$, b fresh in A

$(\mathbf{rec}\mathcal{Z}.A) <:> (\mathbf{rec}\mathcal{W}.A\{\mathcal{W}/\mathcal{Z}\})$, $\mathcal{W}$ fresh in A

$A \mid \mathbf{0} <:> A$

$A \mid B <:> B \mid A$

$(A \mid B) \mid C <:> A \mid (B \mid C)$

$a\circledR\mathbf{0} <:> \mathbf{0}$

$b\circledR(a\circledR A) <:> a\circledR(b\circledR A)$

$a\circledR(A \mid B) <:> (a\circledR A \mid B)$, $a \notin \text{fn}(B)$

$A <: !A \mid A$

$A \mid !A <:> !A$

$\pi.A <: \langle\pi\rangle\,A$

$\pi.A <: [\pi]\,A$

$a\circledR(\langle\pi\rangle\,A) <: \langle\pi\rangle\,(a\circledR A)$, if $a \neq \pi, \bar{\pi}$

$(\mathbf{rec}\mathcal{Z}.A) <: A\{(\mathbf{rec}\mathcal{Z}.A)/\mathcal{Z}\}$

$A_j <: \bigvee_{i\in I} A_i$, if $j \in I$

$$\frac{A <: [\pi]\,A}{A <: \Box A}, \text{ for every } \pi \in Act$$

$$\frac{A <: \langle\pi\rangle\,B}{A \mid C <: \langle\pi\rangle\,(B \mid C)}$$

$$\frac{A_1 <: \langle\alpha\rangle\,B_1 \quad A_2 <: \langle\bar{\alpha}\rangle\,B_2}{A_1 \mid A_2 <: \langle\tau\rangle\,(B_1 \mid B_2)}$$

$a\circledR(!a \mid (\mathbf{rec}\mathcal{Z}.\bar{a}.(a \mid a \mid \mathcal{Z}))) <: [\pi]\,(a\circledR(!a \mid (\mathbf{rec}\mathcal{Z}.\bar{a}.(a \mid a \mid \mathcal{Z}))))$, for every $\pi \in Act$

$A <: \bar{\mathbf{0}}$

$a\circledR(\bar{\mathbf{0}} \mid \bar{\mathbf{0}}) <: \bar{\mathbf{0}} \mid \bar{\mathbf{0}}$

$$\frac{A <: \bar{\mathbf{0}} \quad B <: \bar{\mathbf{0}}}{A \mid B <: \bar{\mathbf{0}} \mid \bar{\mathbf{0}}}$$

Lemma 2.7. *For every process P such that $P \models_\Gamma B$ implies $P \stackrel{\pi}{\not\longrightarrow}$, then $B <: [\pi]\,A$.*

Lemma 2.8. *For every process P such that $P \models_\Gamma \langle\pi\rangle\,A$ implies that, there exists a process P' such that if $P \xrightarrow{\pi'} P''$, then π' is π and $P'' \approx P'$, for every process P'' and action π', then $\langle\pi\rangle\,A <: \pi.A$.*

Table 2: The Subtyping System.

Also, the system is weakly consistent and complete with respect to the semantics. This fact is formalized in the following two results.

Theorem 2.10. (Consistency of the Type System) *If $\Gamma \vdash P : A$, then there exists a process Q such that $P <: Q$ and $Q \models_\Gamma A$, where Γ is a well-defined environment.*

Theorem 2.11. (Completeness of the Type System) *If $P \models_\Gamma A$, then $\Gamma \vdash P : A$, where Γ is a well-defined environment.*

The Application Revisited

Consider the *Looper* process again

$$Looper \stackrel{\text{def}}{=} (\nu a)(a \mid (\mathbf{rec} X.\bar{a}.(a \mid a \mid X)))$$

and the following two abbreviations:

$LoopType \stackrel{\text{abv}}{=} (\mathbf{rec} Z.\bar{a}.(a \mid a \mid Z))$, and

$MsgsType \stackrel{\text{abv}}{=} \quad !a \mid a.$

As already stated in the introduction, the goal is to infer in the system the following judgment

$$\Gamma \vdash Looper : \Box(\bar{\mathbf{0}} \mid \bar{\mathbf{0}}),$$

which states that *Looper* will always have two separate threads running in parallel.

The Type System first proves

$$\vdash Looper : a®(MsgsType \mid LoopType) \tag{1}$$

using structural rules in all steps except one. Table 3 exhibits a derivation of (1).

Tables 4 and 5 exhibit subtyping derivations for

$$a®(!a \mid a \mid (\mathbf{rec} Z.\bar{a}.(a \mid a \mid Z))) <: a®(!a \mid (\mathbf{rec} Z.\bar{a}.(a \mid a \mid Z))) \tag{2}$$

and

$$[\pi]\,(a®(!a \mid (\mathbf{rec} Z.\bar{a}.(a \mid a \mid Z))) <: [\pi]\,(a®(!a \mid a \mid (\mathbf{rec} Z.\bar{a}.(a \mid a \mid Z)))) \tag{3}$$

respectively.

Using some of the subtyping rules listed in 2, we infer from (2) and (3)

$$a®(MsgsType \mid LoopType) <: \Box(a®(MsgsType \mid LoopType)). \tag{4}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash \mathbf{0} : \mathbf{0}}{\vdash a : a}\,\text{T-Pre} \quad a <: !a \mid a}{\vdash a : !a \mid a}\,\text{T-Sub} \quad \dfrac{\dfrac{\dfrac{\dfrac{X : Z \vdash \mathbf{0} : \mathbf{0}}{X : Z \vdash a : a}\,\text{T-Pre} \quad \dfrac{X : Z \vdash \mathbf{0} : \mathbf{0}}{X : Z \vdash a : a}\,\text{T-Pre} \quad X : Z \vdash X : Z}{X : Z \vdash a \mid X : a \mid Z}\,\text{T-Par}}{X : Z \vdash a \mid a \mid X : a \mid a \mid Z}\,\text{T-Par} \quad \dfrac{X : Z \vdash \bar{a}.(a \mid a \mid X) : \bar{a}.(a \mid a \mid Z)}{\vdash (\mathbf{rec} X.\bar{a}.(a \mid a \mid X)) : (\mathbf{rec} Z.\bar{a}.(a \mid a \mid Z))}\,\text{T-Pre}}{\vdash a \mid (\mathbf{rec} X.\bar{a}.(a \mid a \mid X)) : MsgsType \mid LoopType}\,\text{T-Rec}\,\text{T-Par}}{\vdash Looper : a®(MsgsType \mid LoopType)}\,\text{T-New}$$

Table 3: Structural derivation.

$$\dfrac{\dfrac{\dfrac{!a \mid a <: a \mid !a \qquad a \mid !a <: !a}{!a \mid a <: !a}}{!a \mid a \mid (\mathbf{rec}\mathcal{Z}.\bar{a}.(a \mid a \mid \mathcal{Z})) <: !a \mid (\mathbf{rec}\mathcal{Z}.\bar{a}.(a \mid a \mid \mathcal{Z}))}}{a\circledR(!a \mid a \mid (\mathbf{rec}\mathcal{Z}.\bar{a}.(a \mid a \mid \mathcal{Z}))) <: a\circledR(!a \mid (\mathbf{rec}\mathcal{Z}.\bar{a}.(a \mid a \mid \mathcal{Z})))}$$

Table 4: Subtyping derivation 1.

$$\dfrac{\dfrac{\dfrac{\dfrac{!a <: a \mid !a \qquad a \mid !a <: !a \mid a}{!a <: !a \mid a}}{!a \mid (\mathbf{rec}\mathcal{Z}.\bar{a}.(a \mid a \mid \mathcal{Z})) <: !a \mid a \mid (\mathbf{rec}\mathcal{Z}.\bar{a}.(a \mid a \mid \mathcal{Z}))}}{a\circledR(!a \mid (\mathbf{rec}\mathcal{Z}.\bar{a}.(a \mid a \mid \mathcal{Z}))) <: a\circledR(!a \mid a \mid (\mathbf{rec}\mathcal{Z}.\bar{a}.(a \mid a \mid \mathcal{Z})))}}{[\pi]\,(a\circledR(!a \mid (\mathbf{rec}\mathcal{Z}.\bar{a}.(a \mid a \mid \mathcal{Z}))) <: [\pi]\,(a\circledR(!a \mid a \mid (\mathbf{rec}\mathcal{Z}.\bar{a}.(a \mid a \mid \mathcal{Z}))))}$$

Table 5: Subtyping derivation 2.

Table 6 exhibits a derivation of

$$\Box(a\circledR(MsgsType \mid LoopType)) <: \Box(\bar{\mathbf{0}} \mid \bar{\mathbf{0}}). \tag{5}$$

It must be underlined that the last step in this derivation (the dashed line) remains an open problem. This issue is closely related to the fact mentioned previously concerning the preservation of subtyping by conjunction.

Summing up, we have that:

1. $\vdash Looper : a\circledR(MsgsType \mid LoopType)$ from (1);
2. $a\circledR(MsgsType \mid LoopType) <: \Box(a\circledR(MsgsType \mid LoopType))$ from (4);
3. $\Box(a\circledR(MsgsType \mid LoopType)) <: \Box(\bar{\mathbf{0}} \mid \bar{\mathbf{0}})$ from (5).

Therefore, using transitivity of the subtyping relation, we obtain from 2. and 3.

$$a\circledR(MsgsType \mid LoopType) <: \Box(\bar{\mathbf{0}} \mid \bar{\mathbf{0}}) \tag{6}$$

Finally, using the subsumption rule we conclude from (6) that $\vdash Looper : \Box(\bar{\mathbf{0}} \mid \bar{\mathbf{0}})$.

3 Last Remarks

During the research process it was necessary to make some decisions which impact was not totally visible at the time. Looking back, it seems as though other alternatives were equally possible, however leading to necessarily different results. Some of them are worthwhile further investigating. This work is left to be done in the future.

$$\dfrac{\dfrac{\dfrac{MsgsType <: \bar{\mathbf{0}} \qquad LoopType <: \bar{\mathbf{0}}}{MsgsType \mid LoopType <: \bar{\mathbf{0}} \mid \bar{\mathbf{0}}}}{a\circledR(MsgsType \mid LoopType) <: a\circledR(\bar{\mathbf{0}} \mid \bar{\mathbf{0}})} \qquad a\circledR(\bar{\mathbf{0}} \mid \bar{\mathbf{0}}) <: \bar{\mathbf{0}} \mid \bar{\mathbf{0}}}{\dfrac{a\circledR(MsgsType \mid LoopType) <: \bar{\mathbf{0}} \mid \bar{\mathbf{0}}}{\Box(a\circledR(MsgsType \mid LoopType)) <: \Box(\bar{\mathbf{0}} \mid \bar{\mathbf{0}})}}$$

Table 6: Subtyping derivation 3.

Some of these issues are addressed in the following.

Treatment of variables and semantics: The satisfaction relation uses environments to map process variables into type variables. In fact, it demands a one-to-one correspondence. This decision was to simplify the treatment of recursion, but it is rather abusive. Instead, it would be more reasonable to consider each type variable to be satisfied by a set of process variables, *i.e.*, defining *valuation*. Moreover, an alternative definition of semantics should be considered. For instance, it would be more natural to define *satisfaction* through definitions of *valuation* and *denotation*.

Interpretation of prefix: The interpretation of the prefix operator requires that every π-derivative of a process satisfying $\pi.A$, must be bisimilar. This condition is rather demanding. Instead, one could require that any π-derivative of a process satisfying $\pi.A$ must satisfy A.

Recursive types: Recursive types are still an open field of discussion nowadays, due to their complexity. Therefore, this is an aspect that, most certainly, will be worthwhile discussing in a future approach. Nevertheless, herein recursive types are interpreted using structural congruence. This choice was made to simplify the work. Instead, type recursion could have been defined as a fixed point of an operator as is done in [CC03].

Subtyping rules: The rules and lemmas listed in table 2 suffice to ensure completeness and to derive the motivating example. But, most probably, there are rules which can be derived from others and therefore should be dropped. On the other hand, most probably, there are rules yet to be devised which generalize some of the rules considered. In other words, the set of rules presented might not be the minimal one.

Conjunction rule: The rule adopted for conjunction, was deliberately considered in the infinitary case to achieve completeness of the modality $\Box A$. Instead, giving up completeness, one could consider two separate rules: a global binary rule for conjunction and a local infinitary rule for the conjunction $[t]\,A$, where t is a sequence of actions.

References

[Cai03] Luís Caires. Behavioral and Spatial Properties in a Logic for the π-Calculus. In I. Walukiwicz, editor, *Proc. of Foundations of Software Science and Computation Structures'04*. Lecture Notes in Computer Science, Springer Verlag, 2003.

[CC03] L. Caires and L. Cardelli. A Spatial Logic for Concurrency (Part I). *Journal of Information and Computation*, 186(2):194–235, 2003.

[LG00] 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.

[Mil89] Robin Milner. *Communication and concurrency*. New York : Prentice Hall, 1989.