Available online at www.sciencedirect.com

SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 128 (2005) 169–183

www.elsevier.com/locate/entcs

Name-Passing in an Ambient-Like Calculus and Its Proof Using Spatial Logic

Xudong Guan[1,2]

Departamento de Informática, FCT/UNL
Monte de Caparica, 2829-516 Caparica, Portugal

Abstract

This paper studies a restricted version of the ambient calculus. We only allow single-threaded ambients migrating in a network of immobile ambients, exchanging payloads, and delivering them. With this restriction, we arrive at a calculus free from grave interferences. In previous works, this is only possible by sophisticated type systems.
We focus on the expressiveness of the restricted calculus. We show that we can still repeat Zimmer's encoding of name-passing in our calculus. Moreover, we prove a stronger operational correspondence result using a novel spatial logic, which specifies spatial properties of processes invariant to process reductions.

Keywords: ambient calculus, expressiveness, encoding, spatial logic

1 Introduction

The ambient calculus [6] is something that combines "holy" and "evil". It is holy for its simplicity as an abstract model, its resemblance to mobile computation [3], and its expressive power [11,18]. It is evil for the difficulties of verifying process properties [8,10,13], mainly due to the interferences problem among the ambient primitives [10]. For this, quite a few variants were proposed in the literature [10,1,7,12]. In the Safe Ambient calculus (SA) [10] for

[1] Supported by FET-GC IST-2001-33100 Profundis. Most of the work was done when the author was a post-doc research fellow at INRIA-Sophia Antipolis, supported by FET-GC IST-2001-32222 MIKADO.

[2] Email: xudong.guan@di.fct.unl.pt

doi:10.1016/j.entcs.2004.11.035

example, CCS-style co-actions are introduced into the calculus. By an additional immobile and single-threaded type discipline, one is able to control the grave interferences in ambients and bring stronger behavior theory.

As a further step in the SA direction, we introduce in this paper an ambient-like calculus called the *wagon calculus*, obtained by only keeping a fragment of well-behaved SA, namely anonymous sinlge-threaded ambients and immobile ambients, and limiting their interaction patterns. A typical single-threaded ambient in SA is of the form

$$a[\mathtt{out}\ s.\mathtt{in}\ s'.\cdots.\overline{\mathtt{open}}\ a \mid a'[P] \mid s''[Q] \mid ...]$$

while that of an immobile ambient is

$$s[!\overline{\mathtt{in}}\ s \mid !\overline{\mathtt{out}}\ s \mid !\mathtt{open}\ a \mid a[P] \mid s'[Q] \mid ...]$$

In wagon, we drop the name a and write $M\langle P\rangle$ for single-threaded ambients, where M is the sequence of actions and P the sub-ambients. For immobile ambients, we drop replicated top-level actions and only write $s[P]$ where P is the collection of sub-ambients. Moreover, we allow only the following four patterns of interactions:

$$\begin{aligned} s.M\langle P\rangle \mid s[Q] &\to s[M\langle P\rangle \mid Q] \\ s[\uparrow.M\langle P\rangle \mid Q] &\to M\langle P\rangle \mid s[Q] \\ s[\mathtt{dis}\langle P\rangle \mid Q] &\to s[P \mid Q] \\ \mathtt{put}\langle P\rangle \mid \mathtt{get}.M\langle Q\rangle &\to M\langle P \mid Q\rangle \end{aligned}$$

Their counter parts in SA are respectively:

$$\begin{aligned} a[\mathtt{in}\ s.M \mid P] \mid s[!\overline{\mathtt{in}}\ s \mid Q] &\to s[a[M \mid P] \mid !\overline{\mathtt{in}}\ s \mid Q] \\ s[!\overline{\mathtt{out}}\ s \mid a[\mathtt{out}\ s.M \mid P] \mid Q] &\to a[M \mid P] \mid s[!\overline{\mathtt{out}}\ s \mid Q] \\ s[!\mathtt{open}\ a \mid a[\overline{\mathtt{open}}\ a \mid P] \mid Q] &\to s[!\mathtt{open}\ a \mid P \mid Q] \\ a[\mathtt{in}\ b.\overline{\mathtt{open}}\ a \mid P] \mid b[\overline{\mathtt{in}}\ b.\mathtt{open}\ a.M \mid Q] &\to\to b[M \mid P \mid Q] \end{aligned}$$

To justify our choice, we show that wagon still retain the expressive power of SA, in that it can still simulate π-calculus style name-passing, which is the state-of-the-art expressiveness example given by Zimmer [18].

The prove of a reasonable operational correspondence for the encoding is not trivial. To simulate name-passing, the encoding uses a lot of auxiliary steps to prepare for the communication and to build explicit substitution ambients. Indeed, Zimmer wrote regarding the analyze of these intermediate states that "only an automatic demonstration tool could maybe handle".

In this paper, we introduce a novel spatial logic where a formula is essentially a process with names replaced by name sets. It denotes the set of processes whose names are covered by its corresponding name sets. Together

with other spatial connectives, the logic is able to enumerate different types of packets in different locations of a process. Moreover, we are able to check that some formulas are *closures*, i.e. the spatial properties are reduction-invariant. This enable us to formally states for example that some packets will never appear in some where. As the application of this theory to the proof of the encoding, we are able to find a closure that specifies properties satisfied by all the encodings and their derivations (Lemma 6.5). To this end, we formalize a special contextual equivalence (Definition 7.3) limiting the testing context to be within this closure. We then prove that all the auxiliary steps are equivalences (Lemma 7.4) and the only non-equivalence reduction step in the encoding corresponds to one reduction in π, thus closing Zimmer's conjecture.

Organization of the paper: Section 2 presents the wagon calculus and its reduction semantics. Section 3 gives the encoding. Section 4 presents an equivalent chemical semantics on which the spatial logic in Section 5 is based. Section 6 presents formulas for the encoding and proves the closure property. Section 7 gives the operational correspondence proof. Due to page limitation, all proofs are omited. They may be found in the full paper [9].

2 Wagon processes

We first define the syntax and reduction semantics of the wagon calculus.

Definition 2.1 (Wagon Process) Let a, b range over a set $\mathcal{N}$ of names, wagon headers (M, N) and wagon processes (P, Q) are defined by the following grammar:

$$M,\ N ::= \mathtt{dis} \mid \mathtt{put} \mid \mathtt{get}.M \mid \uparrow.M \mid a.M$$
$$P,\ Q ::= \mathbf{0} \mid (\nu a)P \mid (P \mid Q) \mid a[P] \mid M\langle P\rangle \mid !M\langle P\rangle$$

In the syntax, we find the standard nil process $\mathbf{0}$, restriction $(\nu a)P$, parallel composition $(P \mid Q)$, and location $a[P]$. A packet $M\langle P\rangle$ (or the replicated version $!M\langle P\rangle$) is made up of a header M and a payload process P. Header M is a sequence of actions. They may *dissolve* the packet and release the payload ($\mathtt{dis}$), *put* the payload to other packet ($\mathtt{put}$), *get* the payload from other packet ($\mathtt{get}.M$), move the packet *out* of its current location ($\uparrow.M$), or move the packet *in* to location a ($a.M$). We usually call $\mathtt{get}.M\langle P\rangle$ (resp. $\mathtt{put}\langle P\rangle$, $\uparrow.M\langle P\rangle$, $a.M\langle P\rangle$) a get-packet (resp. put-, out-, in-packet). Among process constructs, "(νa)" is the only binder. The notion of free names ($fn(P)$) and renaming of bound names are defined as usual. We use $=_\alpha$ to relate alpha-convertible processes.

Convention: In writing processes (and later solutions and formulas) we use the following conventions. We always let "$\mid$" have the least binding power

(S-Par-Zero)	$P \mid \mathbf{0}$	$\equiv$	P	
(S-Par-Sym)	$P \mid Q$	$\equiv$	$Q \mid P$	
(S-Par-Assoc)	$(P \mid Q) \mid R$	$\equiv$	$P \mid (Q \mid R)$	
(S-Res-Par)	$P \mid (\nu a)Q$	$\equiv$	$(\nu a)(P \mid Q)$	if $a \notin fn(P)$
(S-Res-Loc)	$b[(\nu a)P]$	$\equiv$	$(\nu a)b[P]$	if $a \neq b$
(S-Rep)	$!M\langle P\rangle$	$\equiv$	$!M\langle P\rangle \mid M\langle P\rangle$	
(S-Dis)	$\mathtt{dis}\langle P\rangle$	$\equiv$	P	
(S-Alpha)	$P =_\alpha Q$	$\Longrightarrow$	$P \equiv Q$	

Fig. 1. The structural congruence relation

(R-In)	$a.M\langle P\rangle \mid a[Q]$	$\rightarrow$	$a[M\langle P\rangle \mid Q]$
(R-Out)	$a[\uparrow.M\langle P\rangle \mid Q]$	$\rightarrow$	$M\langle P\rangle \mid a[Q]$
(R-Get-Put)	$\mathtt{get}.M\langle P\rangle \mid \mathtt{put}\langle Q\rangle$	$\rightarrow$	$M\langle P \mid Q\rangle$
(R-Par)	$P \rightarrow P'$	$\Longrightarrow$	$P \mid Q \rightarrow P' \mid Q$
(R-Res)	$P \rightarrow P'$	$\Longrightarrow$	$(\nu a)P \rightarrow (\nu a)P'$
(R-Loc)	$P \rightarrow P'$	$\Longrightarrow$	$a[P] \rightarrow a[P']$
(R-Struct)	$P \equiv P' \rightarrow Q' \equiv Q$	$\Longrightarrow$	$P \rightarrow Q$

Fig. 2. The reduction relation

and try to omit unnecessary parentheses whenever possible. So $(\nu a)P \mid Q$ means $(((\nu a)P) \mid Q)$. We often omit $\mathtt{dis}$ (together with the preceding dot, if applicable) and $\mathbf{0}$. Thus, $\mathtt{get}.b\langle\rangle \mid b[\,]$ is a shorthand for $\mathtt{get}.b.\mathtt{dis}\langle\mathbf{0}\rangle \mid b[\mathbf{0}]$.

Definition 2.2 (Structural Congruence) A usual structural congruence relation ($\equiv$) is defined over wagon processes. It is the least congruence satisfying the rules in Fig. 1.

Definition 2.3 (Reduction Semantics) The reduction relation ($\rightarrow$) on wagon processes is defined by the rules in Fig. 2.

The reduction semantics should be self-explaining. We find it technically more convenient using the structural rule $\mathtt{dis}\langle P\rangle \equiv P$ than $\mathtt{dis}\langle P\rangle \rightarrow P$. We only summarize here a few distinct properties of wagon:

(i) *Objective migration*: while there are arguments for subjective move in ambients, we use objective migration in wagon.

(ii) *Safe dissolving*: there are many arguments for dropping the "open" primitive, since it is too powerful and has a lot of side-effects [1,7]. In wagon, the dissolving of ambients only happens inside immobile ambients and no hijacking of the parent ambient could happen.

(iii) *Eternal locations*: locations in wagon can neither move nor disappear. New ones can be dynamically created during computation.

(iv) *A computation model of packet-peeling and payload-routing*: every reduction involves peeling off an action from a packet and routing the payload accordingly. Unlike ambients, wagon payloads are static, which reduces interferences and facilitates reasoning of process behaviors.

Example 2.4 Location transparency can be easily modelled in wagon. Locations like *toLisbon* below models routing service that will deliver payload transparently to the desired destination.

$$\begin{aligned} & Lisbon[P] \mid Marseille[toLisbon[!\texttt{get}.\uparrow.\uparrow.Lisbon\langle\rangle] \mid toLisbon.\texttt{put}\langle Q\rangle] \\ \rightarrow^* & Lisbon[P \mid Q] \mid Marseille[toLisbon[!\texttt{get}.\uparrow.\uparrow.Lisbon\langle\rangle]\,] \end{aligned}$$

3 The Encoding

Wagon calculus is a sub-calculus of SA free of grave interference (see full paper [9]). As a demonstration of the expressiveness of wagon, this paper focuses on the encoding of π-calculus in wagon and its proof technique. Our source language is the asynchronous finite π-calculus (we will call it π_{af}) with the following syntax:

$$p,\, q \;::=\; \mathbf{0} \mid a\langle b\rangle \mid a(b).p \mid (\nu a)p \mid (p \mid q)$$

We use $a,\ b$ to range over the set of channel names. We have the standard binding rules and the standard definition of the set of free names of processes $fn(p)$. We adopt the standard reduction semantics where the only axiom for the reduction relation is given by the rule: $a\langle c\rangle \mid a(b).p \rightarrow p\{c/b\}$. Process $p\{c/b\}$ stands for the capture-free substitution of all the free names b in p with c. Readers may refer to book [17] for a more detailed presentation.

Convention: For precedence related with meta-notations like substitution, we assume in this paper that they always bind tighter than other language operators. So process $p \mid (\nu a)p\{c/b\}$ will stands for $(p \mid (\nu a)(q\{c/b\}))$. We use $X,\ Y,\ Z$ to denote finite sets of names. For a name set $X = \{a_1, ..., a_k\}$, we abbreviate $(\nu a_1)...(\nu a_k)P$ as $(\nu X)P$. We use $X\backslash Y$ to denote the set minus operation. We sometime omit $\cup$ in set union and the braces for singleton set where the context is clear, e.g. $X\backslash a$ for $X\backslash\{a\}$ and XY for $X \cup Y$. We use $\prod$ to abbreviate a sequence of parallel composition, e.g. we write $\prod_{i=1}^{k} P_i$ for $P_1 \mid ... \mid P_k$. Specifically, we often use $\prod_{a\in X} P$ as a short hand for $P\{a_1/a\} \mid ... \mid P\{a_k/a\}$ where $X = \{a_1,\ ...,\ a_k\}$. These conventions are applicable to all models in this paper.

Definition 3.1 (The Encoding) The encoding of π_{af} in wagon, $\langle\!\langle\cdot\rangle\!\rangle$, is defined in Fig. 3.

$$
\begin{array}{rcl@{\qquad}rcl}
[\![\mathbf{0}]\!] & =_{\mathtt{def}} & \mathbf{0} & \mathbf{chn} & =_{\mathtt{def}} & !\mathtt{get}\langle\rangle \mid comm[\mathbf{0}] \\
[\![(\nu a)p]\!] & =_{\mathtt{def}} & (\nu a)(a[\mathbf{chn}] \mid [\![p]\!]) & \mathbf{fwd}\ b & =_{\mathtt{def}} & !\mathtt{get}.\uparrow.b.\mathtt{put}\langle\rangle \\
[\![p \mid q]\!] & =_{\mathtt{def}} & [\![p]\!] \mid [\![q]\!] & \langle\!\langle p\rangle\!\rangle_X & =_{\mathtt{def}} & [\![p]\!] \mid \prod_{a\in X} a[\mathbf{chn}] \\
[\![a\langle b\rangle]\!] & =_{\mathtt{def}} & a.\mathtt{put}\langle comm.\mathtt{put}\langle \mathbf{fwd}\ b\rangle\rangle & \langle\!\langle p\rangle\!\rangle & =_{\mathtt{def}} & \langle\!\langle p\rangle\!\rangle_{fn(p)} \\
[\![a(b).p]\!] & =_{\mathtt{def}} & (\nu b)(b[\mathbf{0}] & & & \\
 & & \mid a.\mathtt{put}\langle comm.\mathtt{get}.\uparrow.\uparrow.b\langle\uparrow\langle [\![p]\!]\rangle\rangle\rangle) & & &
\end{array}
$$

Fig. 3. The encoding of π_{af} into wagon: $\langle\!\langle\cdot\rangle\!\rangle$

In the encoding, we suppose that $\mathcal{N}$ contains the set of channel names. Moreover, we suppose that there is a special name $comm \in \mathcal{N}$ that is not in the set of channel names. We use get/put-interaction to simulate channeled communication. For every channel a, we supply a location of the form $a[\mathbf{chn}]$ called *queue* where I/O-processes on channel a meet and interact. To ensure that one and only one queue is provided for each channel, we first define $[\![p]\!]$ as the wagon process where every private channel is provided with a queue but none for the free ones. We then let $\langle\!\langle p\rangle\!\rangle_X$ to be the process where queues associated with channel names in set X are provided. The final encoding $\langle\!\langle p\rangle\!\rangle$ is simply a short hand for $\langle\!\langle p\rangle\!\rangle_{fn(p)}$.

Given some reasonable equivalence relation $\simeq$ on the target language, a satisfactory operational correspondence involves two parts. The first part requires that the encoding *follows* the reduction path of the source process, up to $\simeq$. The second part requires that it *always follows* one of the reduction paths of the source process, up to $\simeq$. In our case where the encoding uses many intermediate steps, the second part is more difficult to prove.

Our approach is to find a reasonable equivalence that include all the auxiliary reduction steps. These reductions will never be prevented by interferences from the other reductions. For this, we first identify four groups of reductions used in the encoding and state their properties informally:

(i) **(R-Out)**: no interference could happen and this reduction is always successful, unless the packet is at the top-level.

(ii) **(R-In)**: no interference could happen if we can ensure that there is no duplicated sibling names.

(iii) **(R-Get-Put)** involving a replicated get-packet: no interference could happen if the get packet is "uniform", in the sense that there is no possibility of other get-packets appearing in the same location.

(iv) A usual **(R-Get-Put)**: this only happens inside *comm* locations and is in one-to-one correspondence with the interference between I/O processes in π_{af}.

To formally prove these properties, we introduce a novel spatial logic that can describe the general shape of a wagon process. For this, we first introduce a chemical representation of the wagon calculus.

4 Chemical Semantics

For clear specification, we often need to intrude the scope of a binder to contain only those components that know it. This is however particular difficult with calculi supporting explicit locality. Suppose in a distributed system, only some mobile agent P and its owner Q know some secret k. We have to write in the original syntax the following configuration when the agent is inside some foreign system $a[R]$:

$$(\nu k)(Q \mid a[P \mid R])$$

In this section, a straightforward chemical semantics is introduced to break down locations, e.g, writing $a \mid P@a \mid R@a$ for $a[P \mid R]$. Thus, the binders could shrink and we may rewrite the above process as:

$$(\nu k)(Q \mid P@a) \mid a \mid R@a$$

Remark 4.1 For this to work, there should be no duplicated sibling names in the location tree. This could be formalized for example using a small type system, which is included in the full paper [9].

Definition 4.2 (Wagon Solution) *Wagon solutions* $(A,\ B)$ are defined by adding a singleton name construct [3] a and a located solution construct $A@a$ to the process grammar.

$$A,\ B \quad ::= \quad \mathbf{0} \mid (\nu a)A \mid (A \mid B) \mid a[A] \mid M\langle A\rangle \mid !M\langle A\rangle \mid a \mid A@a$$

By definition, a wagon process is also a wagon solution. We assume that "@" has a binding power tighter than "|", but less than any of the other operators.

Definition 4.3 (Chemical Structural Congruence) We define structural congruence relation ($\rightleftharpoons$) on wagon solutions as the least congruence satisfying the rules in Fig. 4.

Only the last three rules are new compared to $\equiv$.

Definition 4.4 (Evaluation Context) An *evaluation context* $\mathcal{C}$ is of the form $(\nu X)(A \mid -)$ where "$-$" is called a *hole*. For any solution B, $\mathcal{C}(B)$ is defined as the solution obtained by filling the hole in $\mathcal{C}$ with B.

[3] We need to keep the singleton name since otherwise it would be hard to define the corresponding chemical reduction rule for **(R-In).**

$$
\begin{array}{lrcll}
\textbf{(CS-Par-Zero)} & A \mid \mathbf{0} & \rightleftharpoons & A & \\
\textbf{(CS-Par-Sym)} & A \mid B & \rightleftharpoons & B \mid A & \\
\textbf{(CS-Par-Assoc)} & (A \mid B) \mid C & \rightleftharpoons & A \mid (B \mid C) & \\
\textbf{(CS-Res-Par)} & A \mid (\nu a)B & \rightleftharpoons & (\nu a)(A \mid B) & \text{if } a \notin fn(A) \\
\textbf{(CS-Res-At)} & (\nu a)A@b & \rightleftharpoons & (\nu a)(A@b) & \text{if } a \neq b \\
\textbf{(CS-Rep)} & !M\langle A\rangle & \rightleftharpoons & !M\langle A\rangle \mid M\langle A\rangle & \\
\textbf{(CS-Dis)} & \mathtt{dis}\langle A\rangle & \rightleftharpoons & A & \\
\textbf{(CS-Alpha)} & A =_\alpha B & \Longrightarrow & A \rightleftharpoons B & \\
\textbf{(CS-Loc)} & a[A] & \rightleftharpoons & a \mid A@a & \\
\textbf{(CS-Nil-At)} & \mathbf{0}@a & \rightleftharpoons & \mathbf{0} & \\
\textbf{(CS-Par-At)} & (A \mid B)@a & \rightleftharpoons & A@a \mid B@a &
\end{array}
$$

Fig. 4. The chemical structural congruence relation

$$
\begin{array}{lrcl}
\textbf{(CR-In)} & a.M\langle A\rangle@s \mid a@s & \hookrightarrow & M\langle A\rangle@a@s \mid a@s \\
\textbf{(CR-Out)} & \uparrow.M\langle A\rangle@a@s & \hookrightarrow & M\langle A\rangle@s \\
\textbf{(CR-Get-Put)} & \mathtt{get}.M\langle A\rangle@s \mid \mathtt{put}\langle B\rangle@s & \hookrightarrow & M\langle A \mid B\rangle@s \\
\textbf{(CR-Ctx)} & A \hookrightarrow A' & \Longrightarrow & \mathcal{C}(A) \hookrightarrow \mathcal{C}(A') \\
\textbf{(CR-Struct)} & A \rightleftharpoons A' \hookrightarrow B' \rightleftharpoons B & \Longrightarrow & A \hookrightarrow B
\end{array}
$$

Fig. 5. The chemical reduction relation

For notational purpose, we sometime use $A@s$ for either $A@a_1@\cdots@a_k$ (where s takes the sequence $a_1@\cdots@a_k$) or A (where s takes the empty sequence ϵ). We define $fn(s)$ accordingly.

Definition 4.5 (Chemical Reduction) The reduction relation ($\hookrightarrow$) over chemical solutions is defined by the rules in Fig. 5.

Proposition 4.6 (Correspondence of the Two Semantics) *We have:*

(i) *If $P \to Q$, then $P \hookrightarrow Q$;*

(ii) *If $P \hookrightarrow A$, then there is Q s.t. $P \to Q$ and $Q \rightleftharpoons A$.*

5 Spatial Logic

A formula in our logic is essentially a generalized wagon solution with names replaced by sets. We regard chemical solutions (wagon headers) as wagon formulas (header formulas) by taking them as singleton sets.

Definition 5.1 (Wagon Formula) *Wagon formulas* ($\mathbb{A}$, $\mathbb{B}$) are sets of wagon solutions. *Header formulas* ($\mathbb{M}$, $\mathbb{N}$) are sets of wagon headers. Given a set of set variables $\mathcal{V}$ ranged over by x, y, they are defined by the following gram-

mar [4] :

$$
\begin{aligned}
u, v &::= x \mid X \mid u \cup v \\
\mathbb{M}, \mathbb{N} &::= \mathtt{dis} \mid \mathtt{put} \mid \mathtt{get}.\mathbb{M} \mid \uparrow.\mathbb{M} \mid u.\mathbb{M} \\
\mathbb{A}, \mathbb{B} &::= \mathbf{0} \mid (\nu a)\mathbb{A} \mid (\mathbb{A} \mid \mathbb{B}) \mid a[\mathbb{A}] \mid \mathbb{M}\langle\mathbb{A}\rangle \mid !\mathbb{M}\langle\mathbb{A}\rangle \mid a \mid \mathbb{A}@u \\
&\quad \mid \mathbb{A}^* \mid \prod_{a\in u} \mathbb{A} \mid (\nu x)\mathbb{A} \mid \mathbb{A}(\vec{u})
\end{aligned}
$$

We have two different kinds of binders in wagon formulas: "(νa)" and "$\prod_{a\in u}$" are the binders of name a; "(νx)" is the binder of variable x. Free names ($fn(\cdot)$), free variables ($fv(\cdot)$), and substitutions ($(\cdot)\{a/b\}$, $(\cdot)\{u/x\}$) have standard definitions. Please note that substituting a name in a name set with another name in that set may result in the first name being absorbed. Also, name capture should be avoided when substituting a variable with a name set, as in $((\nu a)(a[\mathbf{0}] \mid x.\mathtt{put}\langle\rangle))\{\{a, b\}/x\}$.

Convention: For formula operators, we assume that "(νa)" and "(νx)" bind tighter than "*" and "@", then "$\prod_{a\in u}$", and finally "|". For logical expressions, we assume that "$\exists$" and "$\forall$" bind tighter than "$\wedge$" and "$\vee$".

Definition 5.2 (Denotation and Satisfaction) The denotation of formulas is given in Fig. 6, defined up to chemical equivalence $\rightleftharpoons$. We often call A an $\mathbb{A}$-*solution.*

Some explanations of the semantics are due. The semantics of $M \in \mathbb{M}$ is quite straightforward. $\mathbb{M}$ specifies a particular pattern of headers having the same sequence of actions, only the names of the move in actions could be chosen among those sets specified by $\mathbb{M}$. Note that $\mathbb{M}$ is $\emptyset$ if one of the associated set is empty. For the wagon formulas corresponding to the syntax of solutions, their semantics are more or less straightforward. We let $\emptyset\langle\mathbb{A}\rangle$, $!\emptyset\langle\mathbb{A}\rangle$, and $\mathbb{A}@\emptyset$ to be equal to $\mathbf{0}$ instead of $\emptyset$. The semantics of the last four constructs worth further explanations.

(i) $\mathbb{A}^*$: We use the Kleene star construct to specify processes that can be break down into components that all satisfy $\mathbb{A}$.

(ii) $\prod_{a\in u}\mathbb{A}$: We use this connective to specify processes that can be break down into components, each of which satisfies a particular instance of $\mathbb{A}$, that is, the result of substituting a in $\mathbb{A}$ with one particular name in u. It is normally used to enumerate a collection of locations ($\prod_{a\in u} a[\mathbf{0}]$).

(iii) $(\nu x)\mathbb{A}$: We use the variable binder to specify processes in which some particular fresh name set can be identified and after extending their scopes

[4] For the sake of the usage in this paper, we do not include the basic connectives $\neg$ and $\wedge$ for simplicity. They can be easily added using the set based semantics below.

$$
\begin{array}{rcl}
M \in \mathtt{dis} & \Longleftrightarrow & M = \mathtt{dis} \\
M \in \mathtt{put} & \Longleftrightarrow & M = \mathtt{put} \\
M \in \mathtt{get}.\mathbb{M} & \Longleftrightarrow & \exists N \in \mathbb{M}.M = \mathtt{get}.N \\
M \in \uparrow.\mathbb{M} & \Longleftrightarrow & \exists N \in \mathbb{M}.M = \uparrow.N \\
M \in u.\mathbb{M} & \Longleftrightarrow & \exists a \in u.\exists N \in \mathbb{M}.M = a.N \\
\\
A \in \mathbf{0} & \Longleftrightarrow & A \rightleftharpoons \mathbf{0} \\
A \in \mathbb{M}\langle\mathbb{A}\rangle & \Longleftrightarrow & \exists M \in \mathbb{M}.\exists B \in \mathbb{A}.A = M\langle B\rangle, \text{ if } \mathbb{M} \neq \emptyset; \text{ else } A \rightleftharpoons \mathbf{0} \\
A \in !\mathbb{M}\langle\mathbb{A}\rangle & \Longleftrightarrow & \exists M \in \mathbb{M}.\exists B \in \mathbb{A}.A = !M\langle B\rangle, \text{ if } \mathbb{M} \neq \emptyset; \text{ else } A \rightleftharpoons \mathbf{0} \\
A \in (\nu a)\mathbb{A} & \Longleftrightarrow & \exists B \in \mathbb{A}.A \rightleftharpoons (\nu a)B \\
A \in (\mathbb{A} \mid \mathbb{B}) & \Longleftrightarrow & \exists B_1 \in \mathbb{A}.\exists B_2 \in \mathbb{B}.A \rightleftharpoons B_1 \mid B_2 \\
A \in a[\mathbb{A}] & \Longleftrightarrow & \exists B \in \mathbb{A}.A \rightleftharpoons a[B] \\
A \in a & \Longleftrightarrow & A \rightleftharpoons a \\
A \in \mathbb{A}@u & \Longleftrightarrow & \exists B \in \mathbb{A} \;.\; \exists a \in u \;.\; \; A \rightleftharpoons B@a, \text{ if } u \neq \emptyset; \text{ else } A \rightleftharpoons \mathbf{0} \\
A \in \mathbb{A}^* & \Longleftrightarrow & \exists k \geq 0.\exists A_1, \cdots, A_k \in \mathbb{A} \;.A \rightleftharpoons \prod_{i=1..k} A_i \\
A \in \prod_{a\in u} \mathbb{A} & \Longleftrightarrow & \exists A_1, \cdots A_k.(\forall i.A_i \in \mathbb{A}\{a_i/a\} \wedge A \rightleftharpoons \prod_{i=1..k} A_i), \\
 & & \text{given } u = \{a_1, \cdots, a_k\} \\
A \in (\nu x)\mathbb{A} & \Longleftrightarrow & \exists X.(X \cap fn(\mathbb{A}) = \emptyset \wedge \exists B \in \mathbb{A}\{X/x\}.A \rightleftharpoons (\nu X)B) \\
A \in \mathbb{A}(\vec{u}) & \Longleftrightarrow & A \in \mathbb{B}\{\vec{u}/\vec{x}\}, \text{ if the definition clause for } \mathbb{A} \text{ is } \mathbb{A}(\vec{x}) = \mathbb{B}
\end{array}
$$

Fig. 6. The denotation of header formulas and wagon formulas

to the out-most, the inner process satisfies formula $\mathbb{A}'$, which is obtained by substituting x in $\mathbb{A}$ with the identified fresh name set. Note that the freshness is formalized by the condition that the identified set can not contain any fresh names in $\mathbb{A}$.

(iv) $\mathbb{A}(\vec{u})$: This is the usual way of invoking a definition clause, say, $\mathbb{A}(\vec{x}) =_{\mathtt{def}} \mathbb{B}$. The set of this formula is equal to $\mathbb{B}\{\vec{u}/\vec{x}\}$. We allow recursive definition clauses.

Example 5.3 A few satisfaction results:

(i) $\mathtt{get}.\uparrow.a\langle\rangle \in \mathtt{get}.\uparrow.\{a, b\}\langle\rangle$

(ii) $!\mathtt{get}\langle\rangle \mid \mathtt{put}\langle\rangle \mid \mathtt{put}\langle\rangle \in !\mathtt{get}\langle\rangle \mid \mathtt{put}\langle\rangle^*$

(iii) $\{\mathbf{0},\ (\nu a)a[\,],\ (\nu a)(\nu b)(a[\,] \mid b[\,]),\ \cdots\} \subseteq (\nu x)\prod_{a\in x} a[\,]$

(iv) $(\nu a)(\nu b)(a[\,] \mid b[\,] \mid \mathtt{get}.a\langle\rangle) \in (\nu x)(\prod_{a\in x} a[\,] \mid \mathtt{get}.x\langle\rangle^*)$

6 Logical Formulas for the Encoding

In this section, we define the formula $\mathbb{CPI}_c$ (Definition 6.3) that contains all the derivations of the encoding (Lemma 6.5).

Definition 6.1 We define the following formulas for the encoding of π_{af}-

calculus:

(i) $\mathbb{C}(x) =_{\mathtt{def}} \prod_{a\in x} a[!\mathtt{get}\langle\rangle \mid comm]$

(ii) $\mathbb{F}(x) =_{\mathtt{def}} !\mathtt{get}.\uparrow.x.\mathtt{put}\langle\rangle$

(iii) $\mathbb{O}_1(x) =_{\mathtt{def}} x.\mathtt{put}\langle comm.\mathtt{put}\langle\mathbb{F}(x)\rangle\rangle$

(iv) $\mathbb{I}_1(x) =_{\mathtt{def}} (\nu b)(b \mid x.\mathtt{put}\langle comm.\mathtt{get}.\uparrow.\uparrow.b\langle\uparrow\langle\mathbb{PI}(x\cup b)\rangle\rangle\rangle\rangle)$

(v) $\mathbb{PI}(x) =_{\mathtt{def}} (\nu y)(\mathbb{O}_1(xy)^* \mid \mathbb{I}_1(xy)^* \mid \mathbb{C}(y))$

(vi) $\mathbb{PI}_c(x) =_{\mathtt{def}} \mathbb{PI}(x) \mid \mathbb{C}(x)$

Lemma 6.2 *For any π_{af}-process p, $[\![p]\!] \in \mathbb{PI}(fn(p))$ and $\langle\!\langle p\rangle\!\rangle \in \mathbb{PI}_c(fn(p))$.*

Definition 6.3 We introduce the following formula definitions:

(i) $\mathbb{O}_2(x) =_{\mathtt{def}} \mathtt{put}\langle comm.\mathtt{put}\langle\mathbb{F}(x)\rangle\rangle@x$

(ii) $\mathbb{O}_3(x,\ y) =_{\mathtt{def}} comm.\mathtt{put}\langle\mathbb{F}(x)\rangle@y$

(iii) $\mathbb{O}_4(x,\ y) =_{\mathtt{def}} \mathtt{put}\langle\mathbb{F}(x)\rangle@comm@y$

(iv) $\mathbb{O}_0(x,\ y) =_{\mathtt{def}} \uparrow.x.\mathtt{put}\langle comm.\mathtt{put}\langle\mathbb{F}(x)\rangle\rangle@y$

(v) $\mathbb{I}_2(x) =_{\mathtt{def}} (\nu b)(b \mid \mathtt{put}\langle comm.\mathtt{get}.\uparrow.\uparrow.b\langle\uparrow\langle\mathbb{PI}(x\cup b)\rangle\rangle\rangle@x)$

(vi) $\mathbb{I}_3(x,\ y) =_{\mathtt{def}} (\nu b)(b \mid comm.\mathtt{get}.\uparrow.\uparrow.b\langle\uparrow\langle\mathbb{PI}(x\cup b)\rangle\rangle@y)$

(vii) $\mathbb{I}_4(x,\ y) =_{\mathtt{def}} (\nu b)(b \mid \mathtt{get}.\uparrow.\uparrow.b\langle\uparrow\langle\mathbb{PI}(x\cup b)\rangle\rangle@comm@y)$

(viii) $\mathbb{I}_5(x,\ y) =_{\mathtt{def}} (\nu b)(b \mid\uparrow.\uparrow.b\langle\mathbb{F}(x) \mid\uparrow\langle\mathbb{PI}(x\cup b)\rangle\rangle@comm@y)$

(ix) $\mathbb{I}_6(x,\ y) =_{\mathtt{def}} (\nu b)(b \mid\uparrow.b\langle\mathbb{F}(x) \mid\uparrow\langle\mathbb{PI}(x\cup b)\rangle\rangle@y)$

(x) $\mathbb{I}_7(x) =_{\mathtt{def}} (\nu b)(b \mid b\langle\mathbb{F}(x) \mid\uparrow\langle\mathbb{PI}(x\cup b)\rangle\rangle)$

(xi) $\mathbb{I}_8(x,\ y) =_{\mathtt{def}} \uparrow\langle\mathbb{PI}(x)\rangle@y$

(xii) $\mathbb{I}_0(x,\ y) =_{\mathtt{def}} (\nu b)(b \mid\uparrow.\mathtt{put}\langle comm.\mathtt{get}.\uparrow.\uparrow.b\langle\uparrow\langle\mathbb{PI}(x\cup b)\rangle\rangle\rangle@y)$

(xiii) $\mathbb{V}(x,\ y) =_{\mathtt{def}} \prod_{a\in y} a[\mathbb{F}(xy)]$

(xiv) $\mathbb{D}(x,\ y) =_{\mathtt{def}} \mathbb{O}_1(xy)^* \mid \mathbb{O}_2(xy)^* \mid \mathbb{O}_3(xy,\ x)^* \mid \mathbb{O}_4(xy,\ x)^* \mid \mathbb{O}_0(xy,\ y)^* \mid \mathbb{I}_1(xy)^* \mid \mathbb{I}_2(xy)^* \mid \mathbb{I}_3(xy,\ x)^* \mid \mathbb{I}_4(xy,\ x)^* \mid \mathbb{I}_5(xy,\ x)^* \mid \mathbb{I}_6(xy,\ x)^* \mid \mathbb{I}_7(xy)^* \mid \mathbb{I}_8(xy,\ y)^* \mid \mathbb{I}_0(xy,\ y)^*$

(xv) $\mathbb{CPI}(x) =_{\mathtt{def}} (\nu y)(\nu z)(\mathbb{C}(y) \mid \mathbb{V}(xy,\ z) \mid \mathbb{D}(xy,\ z))$

(xvi) $\mathbb{CPI}_c(x) =_{\mathtt{def}} \mathbb{CPI}(x) \mid \mathbb{C}(x)$

We now introduce an important notion for formulas:

Definition 6.4 (Closure) Formula $\mathbb{A}$ is called a closure, if whenever $A \in \mathbb{A}$ and $A \hookrightarrow^* A'$, we have $A' \in \mathbb{A}$.

Convention: If $\mathbb{A}(\vec{x})$ is a definition, we use $\mathbb{A}$ to stand for the union: $\bigcup_{\vec{X}} \mathbb{A}(\vec{X})$. If $A \in \mathbb{A}$, then $A \in \mathbb{A}(\vec{X})$ for some $\vec{X} = X_1, \cdots, X_k$.

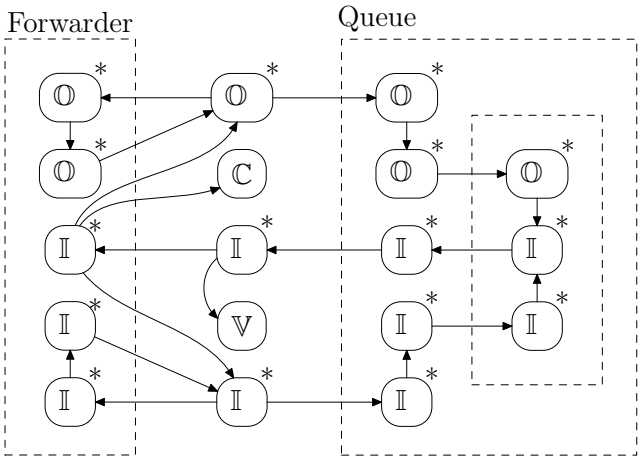

Fig. 7. Closure property of $\mathbb{CPI}_c$

Lemma 6.5 *For any π_{af}-process p, $\langle\langle p \rangle\rangle \in \mathbb{CPI}_c$ and $\mathbb{CPI}_c$ is a closure.*

Fig. 7 gives a rough idea of the closure property of $\mathbb{CPI}_c$. With this, we can ensure that properties such as uniformity of replicated get-packets are indeed satisfied by our encoding, which are essentail to prove our key lemma (Lemma 7.4).

7 Operational Correspondence

To establish the operational correspondence, we need to find some measurements of process equivalence. Often, it is not easy to find a good process equivalence that is of the right size, and easy to prove. Labelled bisimulation is a choice. However, for process calculi with explicit location and process migration like the wagon calculus, it is often difficult to establish a good labelled transition system and the associated bisimulation [13]. Constraints on the translation environment would be another problem.

In this paper, we use may-testing congruence [15,8], which requires that the two processes have exactly the same external observations when being put inside any contexts (observers). Barbed bisimulation and congruence [14,10] would be another choice, although constructing appropriate relations in the proof requires more work.

Definition 7.1 (Barb) We write $A \downarrow_s$ if $A \rightleftharpoons (\nu X)(A_1 \mid \texttt{get}.M\langle A_2\rangle@s)$ with $fn(s) \cap X = \emptyset$. $A \Downarrow_s \iff A \hookrightarrow^* B \downarrow_s$.

Definition 7.2 We define (the unrestrict) may-testing congruence as: $A \simeq B$ if and only if for any $\mathcal{C}$, $\mathcal{C}(A) \Downarrow_s \iff \mathcal{C}(B) \Downarrow_s$.

Sadly, we can not use the above may-testing congruence directly for the operational correspondence, for we have no way of ensuring that the behavior of the tester complies with the basic assumptions for our encoding, namely, the receptiveness of the replicated get-packets in channel locations. The basic requirement of

$$a[\mathbf{chn}] \mid \mathtt{put}\langle C\rangle@a \simeq a[\mathbf{chn}] \mid C@a$$

would fail if the tester is of the form $\mathtt{get}.\uparrow\langle\rangle@a \mid -$. So we define a restricted version of the congruence (written $\mathbb{CPI}_c \vdash A \simeq B$) in which the tester always obeys the basic assumptions of the encoding, i.e. the whole system always forms a valid $\mathbb{CPI}_c$-solution.

Before we proceed, we first define the formula $\mathbb{CPI}_{c^-}(x_1, x_2, z_1)$ that contains all the solutions that we are going to associate with the congruence. It is a $\mathbb{CPI}$-solution with some extra free queues and forwarders. Here x_1 contains the names of free queues provided, x_2 contains those not provided, and z_1 contains the names of free forwarders. We assume that x_1, x_2, and z_1 do not overlap.

$$\begin{array}{rl}\mathbb{CPI}_{c^-}(x_1, x_2, z_1) =_{\mathtt{def}} & (\nu y)(\nu z_2)(\mathbb{C}(y) \mid \mathbb{V}(x_1x_2yz_1,\ z) \mid \mathbb{D}(x_1x_2yz_1,\ z)) \\ \mid & \mathbb{C}(x_1) \mid \mathbb{V}(x_1x_2,\ z_1)\end{array}$$

It is easy to check that any $\mathbb{CPI}_c$-solution is also a $\mathbb{CPI}_{c^-}$-solution ($\mathbb{CPI}_c(X) = \mathbb{CPI}_{c^-}(X, \emptyset, \emptyset)$). For any $\mathbb{CPI}_{c^-}$-solution A, we write $chn(A)$, $var(A)$ the free $\mathbb{C}$-solution and $\mathbb{V}$-solution in A respectively.

Definition 7.3 $\mathbb{CPI}_{c^-}$-solutions A and B are said to be *may testing congruent under* $\mathbb{CPI}_c$, written $\mathbb{CPI}_c \vdash A \simeq B$, if

(May-Loc) $chn(A) \mid var(A) \rightleftharpoons chn(B) \mid var(B)$; and

(May-Test) for any $\mathcal{C}$ s.t. $\mathcal{C}(A) \in \mathbb{CPI}_c$ and $\mathcal{C}(B) \in \mathbb{CPI}_c$, we have $\mathcal{C}(A) \Downarrow_{comm@a}$ $\iff$ $\mathcal{C}(B) \Downarrow_{comm@a}$ for any a.

We are in the place of presenting the operational correspondence. We can now show that:

Lemma 7.4 *All the auxiliary reductions in the encoding are may-testing congruent under* $\mathbb{CPI}_c$.

Theorem 7.5 (Operational Correspondence) *For any* π_{af}*-process* p *we have:*

(i) *If* $p \to q$*, then we can find* R *with* $\langle\!\langle p\rangle\!\rangle \to^* R$ *and* $\mathbb{CPI}_c \vdash \langle\!\langle q\rangle\!\rangle_{fn(p)} \simeq R$*;*

(ii) *If* $\langle\!\langle p\rangle\!\rangle \to^* R$*, then we can find* q *with* $p \to^* q$ *and* $\mathbb{CPI}_c \vdash \langle\!\langle q\rangle\!\rangle_{fn(p)} \simeq R$.

8 Concluding Remarks

In this paper, we present the wagon calculus obtained by restricting the syntax of Safe Ambients. By the restriction, we arrive at a simpler calculus (wagon is in fact a sub-calculus of SA) with similar behavior theory to that of typed SA. Moreover, we obtain a stronger operational correspondence result in the encode of name-passing, using a novel spatial logic.

The logic is build upon a chemical semantics of the wagon calculus. Our experience show that in calculi with name scoping and explicit location, a chemical semantics like ours could enable the transformation of a hierarchical process tree to a flat structure. This enable a finer granularity for scope intrusion and extrusion which could help a deeper analysis of process structures.

Our encoding follows that of Zimmer's, but with a few simplifications like the unified treatment of channel names and variable names. The main contribution is the close of Zimmer's conjecture on the "always-follows" part. We doubt the practibility of apply our proof method directly to Zimmer's encoding, since working on typed SA is much more complex than working on wagon, and Zimmer's encoding needs to be adjusted first to be typable. For the sake of simplicity and presentation, we choose the finite asynchronous π-calculus as our source language. There is little difficulties in extending the encoding to the synchronous π-calculus without replication (but without choice). The version with replication could involve some more work by extending the calculus to use replicated processes/solutions instead of replication packets, but should also be possible.

The theory of our spatial logic is still in its primary stage, used only as a tool for proving the specific application of encoding π-processes. We will further investigate in this direction, especially an abstract reduction relation on the level of formulas, aiming at the mechanical deduction of the closure of any given formula. It would also be very interesting to investigate the relation of our spatial logic with works on behavior types [16] and spatial logic [2,5,4].

Acknowledgement

The author would like to thank Gérard Boudol, Ilaria Castellani, and Mariangiola Dezani-Ciancaglini for their encouragements. Comments from the anonymous referees improved this paper.

References

[1] Bugliesi, M., G. Castagna and S. Crafa, *Access control for mobile agents: the calculus of boxed ambients*, ACM Transactions on Programming Languages and Systems **26** (2004), pp. 57–124.

[2] Caires, L. and L. Cardelli, *A spatial logic for concurrency (part i)*, Information and Computation **186** (2003), pp. 194–235.

[3] Cardelli, L., *Abstraction for mobile computation*, in: J. Vitek and C. Jensen, editors, *Secure Internet Programming*, LNCS **1603**, 1999, pp. 51–94.

[4] Cardelli, L. and A. Gordon, *Logical properties of name restriction*, in: *Proceedings of TLCA'2001*, LNCS **2044**, 2001.

[5] Cardelli, L. and A. D. Gordon, *Anytime, anywhere: Modal logics for mobile ambients*, in: *Proceedings of POPL '00*, ACM, 2000, pp. 365–377.

[6] Cardelli, L. and A. D. Gordon, *Mobile ambients*, Theoretical Computer Science **240** (2000), pp. 177–213, an extended abstract appeared in *Proceedings of FoSSaCS '98*: 140–155.

[7] Coppo, M., M. Dezani-Ciacaglini, E. Giovannetti and I. Salvo, *M3: mobility types for mobile processes in mobile ambients*, in: *CATS 2003*, ENTCS **78**, 2003.

[8] Gordon, A. D. and L. Cardelli, *Equational properties of mobile ambients*, in: W. Thomas, editor, *Proceedings of FoSSaCS '99*, LNCS **1578** (1999), pp. 212–226.

[9] Guan, X., *Name-passing in an ambient-like calculus and its proof using spatial logic* (2004), full version of this paper. Available from author's homepage http://ctp.di.fct.unl.pt/~xguan.

[10] Levi, F. and D. Sangiorgi, *Mobile safe ambients*, Transactions on Programming Languages and Systems **25** (2003), pp. 1–69.

[11] Maffeis, S. and I. Phillips, *On the computational strength of pure ambient calculi*, in: *Proc. Express 2003*, ENTCS **91**, 2003.

[12] Merro, M. and M. Hennessy, *Bisimulation congruences in safe ambients*, in: *Conference Record of POPL '02 The 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2002, pp. 71–80.

[13] Merro, M. and F. Z. Nardelli, *Bisimulation proof methods for mobile ambients*, in: *Proceedings of ICALP 2003*, LNCS **2719** (2003), available as COGS Technical Report 2003:1.

[14] Milner, R. and D. Sangiorgi, *Barbed bisimulation*, in: W. Kuich, editor, *Proceedings of ICALP '92*, LNCS **623** (1992), pp. 685–695.

[15] Morris, J. H., "Lambda-Calculus Method of Programming Language," Ph.D. thesis, MIT (1968).

[16] Ravara, A. and V. T. Vasconelos, *Typing non-uniform concurrent objects*, in: *Proceedings of the International Conference on Concurrency Theory*, LNCS **1877**, 2000.

[17] Sangiorgi, D. and D. Walker, "The π-calculus: a Theory of Mobile Processes," Cambridge University Press, 2001.

[18] Zimmer, P., *On the expressiveness of pure safe ambients*, Mathematical Structures in Computer Science **13** (2003), pp. 721–770.