Available online at www.sciencedirect.com

SCIENCE DIRECT®

Theoretical Computer Science 356 (2006) 440–467

Theoretical Computer Science

www.elsevier.com/locate/tcs

Name-passing in an ambient-like calculus and its proof using spatial logic☆

Xudong Guan*

Critical Software SA, Parque Industrial de Taveiro, Lote 48, 3045-504 Coimbra, Portugal

Abstract

This paper studies a restricted version of the ambient calculus, a process model for mobile and distributed computation. 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 than Zimmer's. The proof takes advantage on a specification about the spatial structure and process distributions of the encoding that are invariant to reductions, using a novel spatial logic.

Keywords: Ambient calculus; π-Calculus; Operational correspondence; Spatial logic

1. Introduction

The ambient calculus [7] is something that combines "holy" and "evil". It is holy for its simplicity as an abstract model, its resemblance to mobile computation [5], and its expressive power [16,27]. It is evil for the difficulties of verifying process properties [12,15,18], mainly due to the *grave interference* problem among the ambient primitives [15]. A pair of grave interference redexes, apart from preventing each other from reducing, often results in two configurations such that one of them is apparently undesirable. For this, quite a few variants were proposed in the literature [15,3,10,17]. In the Safe Ambient calculus (SA) [15] for example, CCS-style co-actions are introduced into the calculus. By an additional *immobile* and *single-threaded* (ST) type discipline, one is able to control the grave interference problem and bring stronger behavior results.

As a further step in the SA direction, we introduce in this paper an ambient-like calculus called the *wagon calculus*, which is free of grave interference by syntactic means. It is obtained by keeping a fragment of well-behaved (i.e. ambients are either ST or immobile) SA processes. [1] To get an idea, consider the following reductions of an SA process

☆ 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. The author is currently supported by FET-GC IST-2001-33100 Profundis.

* Tel.: +351 239989100; fax: +351 239989119.

E-mail address: xudong.guan@criticalsoftware.com.

[1] In this paper, we only consider *pure* SA, i.e. SA without communication primitives.

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2006.02.005

consisting of an ST ambient *pkt* and an immobile ambient s [2]:

$$\begin{aligned} & pkt[\mathtt{in}\, s.\overline{\mathtt{open}}\ pkt \mid P] \mid s[!\overline{\mathtt{in}}\, s \mid !\overline{\mathtt{out}}\, s \mid !\mathtt{open}\ pkt \mid Q] \\ \longrightarrow_{\mathrm{SA}}\ & s[!\overline{\mathtt{in}}\, s \mid !\overline{\mathtt{out}}\, s \mid !\mathtt{open}\ pkt \mid pkt[\overline{\mathtt{open}}\ pkt \mid P] \mid Q] \\ \longrightarrow_{\mathrm{SA}}\ & s[!\overline{\mathtt{in}}\, s \mid !\overline{\mathtt{out}}\, s \mid !\mathtt{open}\ pkt \mid P \mid Q]. \end{aligned} \tag{1}$$

In ambient calculi, an ambient is written $a[P]$ with name a and process P running inside. We use $\longrightarrow_{\mathrm{SA}}$ for the reduction relation between SA processes. In this example, ambient *pkt* first moves into ambient s, as a result of the matching action/co-action pair $\mathtt{in}\, s$ in *pkt* and $\overline{\mathtt{in}}\, s$ in s ($!\overline{\mathtt{in}}\, s$ means an infinite supply of $\overline{\mathtt{in}}\, s$). After that, its boundary is dissolved due to the pair $\mathtt{open}\ pkt$ and $\overline{\mathtt{open}}\ pkt$, releasing process P into s. We show now the corresponding wagon calculus (or *wagon* for short) version [3]:

$$\begin{aligned} & s.\mathtt{dis}\langle P\rangle \mid s[Q] \\ \longrightarrow\ & s[\mathtt{dis}\langle P\rangle \mid Q] \\ \longrightarrow\ & s[P \mid Q]. \end{aligned} \tag{2}$$

Comparing (1) and (2), we may find that the ST ambient *pkt* becomes anonymous. It is written $s.\mathtt{dis}\langle P\rangle$ with $s.\mathtt{dis}$ representing the SA *capability* (i.e. a sequence of actions) $\mathtt{in}\, s.\overline{\mathtt{open}}\ pkt$. Meanwhile, the process $!\overline{\mathtt{in}}\, s \mid !\overline{\mathtt{out}}\, s \mid !\mathtt{open}\ pkt$ in the immobile ambient s is *assumed*, which means that in wagon we cannot exercise finer-grained access control.

In wagon, we use different syntax for ST and immobile ambients. We write $M\langle P\rangle$ for anonymous ST ambients, called *packets*. M is the sequence of actions (like $s.\mathtt{dis}$, which means *enter* s and then *dissolve* there) that it is going to execute. P is the collection of sub-ambients, called the *payload* of the packet. As for immobile ambients, which we call *locations*, we write $s[P]$ where s is the location name and P the collection of sub-ambients. We require that locations give full permissions to packets, i.e. requests from packets to enter a location, leave a location, or be opened, are always granted. All replicated top-level actions are thus dropped in locations.

In wagon, we also suppress some reduction patterns of SA. For example, a wagon payload is no longer a running process. Before it is released, a payload cannot reduce by itself, or interact with the surrounding environment. A packet can neither enter another packet without being opened. More specifically, we only allow the following four interaction patterns (they are the main reduction rules in wagon) [4]:

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

Their counter parts in SA are (roughly):

$$\begin{aligned} pkt[\mathtt{in}\, s.M \mid P] \mid s[!\overline{\mathtt{in}}\, s \mid Q] &\longrightarrow_{\mathrm{SA}} s[pkt[M \mid P] \mid !\overline{\mathtt{in}}\, s \mid Q] \\ s[!\overline{\mathtt{out}}\, s \mid pkt[\mathtt{out}\, s.M \mid P] \mid Q] &\longrightarrow_{\mathrm{SA}} pkt[M \mid P] \mid s[!\overline{\mathtt{out}}\, s \mid Q] \\ !\mathtt{open}\ pkt \mid pkt[\overline{\mathtt{open}}\ pkt \mid P] &\longrightarrow_{\mathrm{SA}} P \\ pkt[\mathtt{in}\ pkt.\overline{\mathtt{open}}\ pkt \mid P] & \\ \mid pkt[\overline{\mathtt{in}}\ pkt.\mathtt{open}\ pkt.M \mid Q] &\longrightarrow_{\mathrm{SA}} \longrightarrow_{\mathrm{SA}} pkt[M \mid P \mid Q]. \end{aligned}$$

With such a radically simplified sub-calculus of SA with the only non-deterministic behavior being the put/get interaction, [5] one may wonder its usefulness, especially regarding its expressive power. Surprisingly enough, we show in this paper that wagon still retains most of the expressive power of SA, in that it can still simulate π-calculus [19,26] style name-passing first done by Zimmer [27] using (pure) SA. Moreover, we prove a stronger operational correspondence

[2] For the process to be well-typed, P and Q must not contain top-level actions, i.e. they are the parallel composition of ambients.

[3] Strictly speaking, P and Q should also be changed here. For illustration purpose, we just use the same letters. The actual differences are implicitly understood in this informal introduction.

[4] In this paper, the reduction $\mathtt{dis}\langle P\rangle \longrightarrow P$ is actually formalized as a structural rule. The main reason is that latter the logical formulas can be less bulky by ignoring these dis-packets.

[5] Migrations in the wagon calculus are deterministic, since we impose a unique name requirements on sibling location. See Remark 12.

result that the encoding indeed only follows the reduction path of the source process, which Zimmer left as a conjecture for his encoding.

The prove of a reasonable *operational correspondence* for the encoding is not trivial. To simulate name-passing, the ambient process requires a lot of *auxiliary reductions* to prepare for the communication and to build explicit substitution ambients for later use. Indeed, Zimmer wrote regarding the analyze of these intermediate states that "only an automatic demonstration tool could maybe handle". For this purpose, we introduce a novel spatial logic for the wagon calculus. A formula in this logic is essentially a process with names replaced by name sets. Its denotation is the set of processes having the same structure with their free names chosen from the corresponding name sets in the formula. Together with other spatial connectives, the logic is able to tell the general shape of the location tree and the possible numbers and shapes of packets in each of the locations. Moreover, by showing that some formulas are *closures*, i.e. the set of processes is reduction-closed, we are able to formally state, for example, that some packets will never appear in some given location. As an application, we are able to find a closure that specifies properties of the encoding and all their derivations (Corollary 28). By formalizing a special contextual equivalence (Definition 32) limiting the testing context to be within this closure, we prove that all the auxiliary reductions are equivalences (Lemma 38) and every non-equivalent reduction step in the encoding corresponds exactly to one (non-deterministic) reduction in π.

Organization of the paper: Section 2 defines the wagon calculus and its reduction semantics. Section 3 shows that wagon is a sub-calculus of SA without grave interferences. Section 4 presents the encoding of name-passing to wagon. Section 5 introduces an equivalent chemical semantics on which the spatial logic in Section 6 is based. Section 6 formalizes the spatial logic and defines a formula that characterizes the encoding. Section 7 introduces a customized may-testing congruence using the characterizing formula. Section 8 gives the operational correspondence proof. Finally, Section 9 concludes.

2. Wagon processes and reduction semantics

In this section we define the syntax and reduction semantics of the wagon calculus.

Definition 1 (*Wagon process*). Let $a, b, \ldots$ range over a set $\mathcal{N}$ of names, *wagon headers* ($M, N, \ldots$) and *wagon processes* ($P, Q, \ldots$) are defined by the following grammar:

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

In the wagon calculus, we have the standard nil process $\mathbf{0}$, name restriction $(va)P$, parallel composition $(P \mid Q)$, and location $a[P]$, which are essentially the constructs needed to build *static trees with hidden labels* [6]. The dynamic part of the calculus lies in the packet construct $M\langle P\rangle$ (and its replicated version $!M\langle P\rangle$) which is made up of a *header* M and a *payload* P. A header is a sequence of actions. Actions may *dissolve* the packet and release the payload ($\mathtt{dis}\langle P\rangle$), *put* the payload to other packet ($\mathtt{put}\langle P\rangle$), *get* the payload from other packet ($\mathtt{get}.M\langle P\rangle$), move the packet *out* of its current location ($\uparrow.M\langle P\rangle$), or move the packet *in* to some location named a ($a.M\langle P\rangle$). We 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 the process constructs, $(va)(\cdot)$ is the only binder. The notion of free names $fn(M)$, $fn(P)$ and renaming of bound names are defined as usual. We use $=_\alpha$ to relate alpha-convertible processes.

Conventions: In writing processes (and later *solutions* and *formulas*) we use the following notational conventions. We often omit `dis` (together with the preceding dot, if applicable) and $\mathbf{0}$. Thus, $\mathtt{get}.b\langle\rangle \mid b[\,]$ is a shorthand for $\mathtt{get}.b.\mathtt{dis}\langle\mathbf{0}\rangle \mid b[\mathbf{0}]$. We let "$(\cdot) \mid (\cdot)$" have the least binding power and try to omit unnecessary parentheses whenever possible. So $(va)P \mid Q$ means $(((va)P) \mid Q)$.

Definition 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 3 (*Reduction semantics*). The reduction relation ($\longrightarrow$) among wagon processes is defined by the rules in Fig. 2.

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

Fig. 2. The reduction relation.

As explained in the introduction, the three rules, **(R-In), (R-Out), and (R-Get-Put)**, are derived from SA. We choose to use $\mathtt{dis}\langle P\rangle \equiv P$ instead of $\mathtt{dis}\langle P\rangle \longrightarrow P$ so that packets of the form $\mathtt{dis}\langle P\rangle$ are factored out (by structural congruence) in the spatial logic. The others are standard rules in process calculi to enable reductions inside contexts. Careful readers may notice that rule $P \longrightarrow P' \Longrightarrow M\langle P\rangle \longrightarrow M\langle P'\rangle$, which should be inherited together with **(R-Loc)** from ambient, is missing. In other words, we have *weak* process mobility in wagon: a payload is a *guarded* process, which remains inactive until the packet dissolves.

For a better understanding of the calculus, we summarize below a few distinct properties of wagon and then give an example.

(1) *Objective migration*: while there are arguments for *subjective moves* in ambients (in the sense of [7]), we use *objective moves*. The distinction between these two phrases is actually quite blur here, since a wagon packet, although appearing to move objectively, is only a disguise of a subjectively moving ambient.

(2) *Eternal locations*: locations in wagon can neither move nor disappear, which makes another radical difference from the original design of ambients. While ambients can be explicitly opened as a way of cleaning-up, wagon locations cannot, making it closer to the $D\pi$-style location semantics [14] enriched with a hierarchy [13]. Expressiveness is obtained by the dynamic creation of new locations by location-carrying packets. An example is given in the second part of Example 4.

(3) *Self-dissolving*: there are many arguments for dropping the `open` primitive in ambient calculi, which seems to be too powerful and has a lot of side-effects [3,10]. In wagon, we use the even more controversial self-dissolving action [7], based on the fact that a location is an immobile ambient with always the most permissive access policy. Extra processes released inside it, just like extra processes moved inside it, cannot affect its functionality, neither by hiding its presence nor by preventing other packets from entering/exiting. However, we do need to control what kind of processes is allowed to enter the location, and the usual name-hiding is enough for this purpose.

(4) *A computation model of packet-peeling and content-routing*: every reduction involves peeling off an action from a packet and routing the packet accordingly. Although stemmed from the ambient calculus, this compute-by-routing model of mobile computation is interesting in itself for its simplicity and the expressive power that it exhibits.

Example 4 (*Routing*). Process routing can be easily modelled in wagon. Location *RtSvr* in the following process models a routing server that will deliver payloads transparently to the particular destination, *Mars*. The interaction between a client $RtSvr.\mathtt{put}\langle P\rangle$ and the routing server results in the process P being delivered to *Mars*, without the client knowing where *Mars* is physically located:

$$\begin{aligned}
&Earth[\,RtSvr[\,!\mathtt{get}.\uparrow.\uparrow.Mars\langle\rangle]\mid RtSvr.\mathtt{put}\langle P\rangle]\mid Mars[Q]\\
\longrightarrow\ &Earth[\,RtSvr[\,!\mathtt{get}.\uparrow.\uparrow.Mars\langle\rangle\ \mid \mathtt{put}\langle P\rangle]]\mid Mars[Q]\\
\longrightarrow\ &Earth[\,RtSvr[\,!\mathtt{get}.\uparrow.\uparrow.Mars\langle\rangle\ \mid \uparrow.\uparrow.Mars\langle P\rangle]]\mid Mars[Q]\\
\longrightarrow\ &Earth[\,RtSvr[\,!\mathtt{get}.\uparrow.\uparrow.Mars\langle\rangle]\mid \uparrow.Mars\langle P\rangle]\mid Mars[Q]\\
\longrightarrow\ &Earth[\,RtSvr[\,!\mathtt{get}.\uparrow.\uparrow.Mars\langle\rangle]]\mid Mars\langle P\rangle\mid Mars[Q]\\
\longrightarrow\ &Earth[\,RtSvr[\,!\mathtt{get}.\uparrow.\uparrow.Mars\langle\rangle]]\mid Mars[P\mid Q].
\end{aligned}\tag{3}$$

Through name scoping, routing services like the above, but for fresh destinations, can be established as well. For example, the location *Mars* in (3) could be initially not known to *Earth*. The following configuration (in which (3) can be derived) gives such an example.

$$\begin{aligned}
&Earth[RtSvr.\mathtt{put}\langle P\rangle]\\
\mid\ &(\nu Mars)Mars[\uparrow.Earth\langle RtSvr[\,!\mathtt{get}.\uparrow.\uparrow.Mars\langle\rangle]\rangle\mid Q].
\end{aligned}$$

Similar mechanisms could be used to provide a private routing service for a trusted agent to cross a *firewall* [7].

3. Sub-language of SA and no grave interference

As explained in the introduction, the wagon calculus is designed to be sub-language of SA. In this section, we give a formal interpretation of wagon processes in SA. [6]

Choose two distinct names *pkt*, *lock* not in $\mathcal{N}$, we use $\mathcal{N}\uplus\{pkt,\ lock\}$ as the underlying set of ambient names in SA. The encoding of wagon processes into SA, $(\!|\cdot|\!)$, is defined in Fig. 3. A header M is translated into an SA capability $\{M\}$. A process P is first translated into an SA process $\{P\}$. We then supply a top-level process $!\mathtt{open}\ pkt\mid !\mathtt{open}\ lock$ to $\{P\}$ to get $(\!|P|\!)$. The use of $\{\cdot\}^L$ prevents payloads from reducing before they are released.

To see how the encoding works, we show below the reduction $\mathtt{get}.M\langle P\rangle\mid\mathtt{put}\langle Q\rangle\longrightarrow M\langle P\mid Q\rangle$ in the corresponding SA translation.

$$\begin{aligned}
&(\!|\mathtt{get}.M\langle P\rangle\mid\mathtt{put}\langle Q\rangle|\!)\\
=\ &pkt[\overline{\mathtt{in}}\ pkt\ .\ \mathtt{open}\ pkt\ .\ \{M\}\mid\{P\}^L]\mid pkt[\mathtt{in}\ pkt\ .\ \overline{\mathtt{open}}\ pkt\mid\{Q\}^L]\mid !\mathtt{open}\ pkt\mid !\mathtt{open}\ lock\\
\longrightarrow_{\mathrm{SA}}\ &pkt[\mathtt{open}\ pkt\ .\ \{M\}\mid\{P\}^L\mid pkt[\overline{\mathtt{open}}\ pkt\mid\{Q\}^L]\,]\mid !\mathtt{open}\ pkt\mid !\mathtt{open}\ lock\\
\longrightarrow_{\mathrm{SA}}\ &pkt[\{M\}\mid\{P\}^L\mid\{Q\}^L]\mid !\mathtt{open}\ pkt\mid !\mathtt{open}\ lock.
\end{aligned}\tag{4}$$

Process (4) differs from $(\!|M\langle P\mid Q\rangle|\!)$ in that it has $\{P\}^L\mid\{Q\}^L$ inside the packet while the latter has $\{P\mid Q\}^L$. We will show that these two processes are indistinguishable in any encoding context (Lemma 9).

An important observation about the encoding is its typability in the original immobile/ST type system of SA [15]. We can assign an immobile type `IAmb[Shh]` to locations and an ST type `STAmb` † `[Shh]` to packets. `Shh` means no communication. The symbol † means that there will be no top-level thread (action) when the ambient is opened. We refer the reader to the original paper for more details about the type system. We now define a type environment that assigns appropriate types to names used in the encoding.

Definition 5. For any finite set of names $X\subset\mathcal{N}$, we define Γ^X to be the type environment with domain $\{pkt,\ lock\}\uplus X$ that maps *pkt* and *lock* to `STAmb` † `[Shh]` and all $a\in X$ to `IAmb[Shh]`.

---

[6] Unfortunately, to simulate the action ↑, we have to modify slightly the original SA syntax by replacing the action "`out` a" with "`out`" (without the parameter a), and the reduction semantics accordingly. Most results in the SA paper trivially hold in this modified version. For simplicity, we will just use the original SA results.

$$
\begin{array}{rcl}
\{\!|\mathtt{dis}|\!\} & =_{\text{def}} & \overline{\mathtt{open}}\ pkt \\
\{\!|\mathtt{put}|\!\} & =_{\text{def}} & \mathtt{in}\ pkt\ .\ \overline{\mathtt{open}}\ pkt \\
\{\!|\mathtt{get}.M|\!\} & =_{\text{def}} & \overline{\mathtt{in}}\ pkt\ .\ \mathtt{open}\ pkt\ .\ \{\!|M|\!\} \\
\{\!|\uparrow.M|\!\} & =_{\text{def}} & \mathtt{out}\ .\ \{\!|M|\!\} \\
\{\!|a.M|\!\} & =_{\text{def}} & \mathtt{in}\ a\ .\ \{\!|M|\!\} \\
 & & \\
\{\!|P|\!\}^L & =_{\text{def}} & lock[\overline{\mathtt{open}}\ lock.\{\!|P|\!\}] \\
\{\!|\mathbf{0}|\!\} & =_{\text{def}} & \mathbf{0} \\
\{\!|M\langle P\rangle|\!\} & =_{\text{def}} & pkt[\{\!|M|\!\} \mid \{\!|P|\!\}^L] \\
\{\!|!M\langle P\rangle|\!\} & =_{\text{def}} & !\{\!|M\langle P\rangle|\!\} \\
\{\!|(\nu a)P|\!\} & =_{\text{def}} & (\nu a)\{\!|P|\!\} \\
\{\!|P \mid Q|\!\} & =_{\text{def}} & \{\!|P|\!\} \mid \{\!|Q|\!\} \\
\{\!|a[P]|\!\} & =_{\text{def}} & a[!\overline{\mathtt{in}}\ a \mid !\overline{\mathtt{out}}\ a \mid !\mathtt{open}\ pkt \mid !\mathtt{open}\ lock \mid \{\!|P|\!\}] \\
 & & \\
(\!|P|\!) & =_{\text{def}} & \{\!|P|\!\} \mid !\mathtt{open}\ pkt \mid !\mathtt{open}\ lock
\end{array}
$$

Fig. 3. The interpretation of wagon in SA $(\!|\cdot|\!)$.

Lemma 6 (*Typability*). *Under the type system of* [15], *we have*

(1) *For any wagon header* M, $\Gamma^{fn(M)} \vdash \{\!|M|\!\} : \mathtt{STCap} \dagger [\mathtt{Shh}]$;
(2) *For any wagon process* P, $\Gamma^{fn(P)} \vdash \{\!|P|\!\} : \mathtt{IProc}[\mathtt{Shh}]$ *and* $\Gamma^{fn(P)} \vdash (\!|P|\!) : \mathtt{IProc}[\mathtt{Shh}]$.

Proof (*Sketch*).

(1) In the type system of [15], the ST-path typing rule says that to obtain $M_1.M_2 : \mathtt{STCap} \dagger [\mathtt{Shh}]$ where M_1 and M_2 are SA capabilities, we need to have $M_1 : \mathtt{STCap} \uparrow [\mathtt{Shh}]$ and $M_2 : \mathtt{STCap} \dagger [\mathtt{Shh}]$. So for any string of capabilities $\{\!|M|\!\}$ to be typed $\mathtt{STCap} \dagger [\mathtt{Shh}]$, we need to have the last one typed $\mathtt{STCap} \dagger [\mathtt{Shh}]$ and all the others typed $\mathtt{STCap} \uparrow [\mathtt{Shh}]$. By analyzing the encoding, we know that $\{\!|M|\!\}$ always ends with $\overline{\mathtt{open}}\ pkt$. Since pkt has type $\mathtt{STAmb} \dagger [\mathtt{Shh}]$, we can assign $\overline{\mathtt{open}}\ pkt$ type $\mathtt{STCap} \dagger [\mathtt{Shh}]$. All the other non-ending actions in $\{\!|M|\!\}$ (including $\mathtt{open}\ pkt$) can be typed $\mathtt{STCap} \uparrow [\mathtt{Shh}]$.
(2) By induction on the structure of P, using the first result. Notice that we can assign $\mathtt{IProc}[\mathtt{Shh}]$ to $\{\!|P|\!\}^L$ and to replicated processes $!\overline{\mathtt{in}}\ a$, $!\overline{\mathtt{out}}\ a$, $!\mathtt{open}\ pkt$, and $!\mathtt{open}\ lock$. □

From the above typability result, we have

Proposition 7 (*No grave interference*). *For any wagon process* P, *there will be no grave interferences* [15, Definition 3.3] *in any derivations of its corresponding SA interpretation* $(\!|P|\!)$.

Proof. Corollary 5.14 of [15] says that no grave interference will occur in any closed (no recursion variable nor communication variable), well-typed process in which all ambient names are either immobile or ST. Obviously, $(\!|P|\!)$ is closed. So the result is a direct consequence of Lemma 6. □

By the typability lemma we can use the powerful typed equational laws of SA to prove properties of the encoding. We first restate the definitions of barbed bisimulation of SA processes.

For any SA process, we write $P \Downarrow_n$ if $P \longrightarrow^*_{\text{SA}} (\nu\tilde{a})(n[\mu.Q_1 \mid Q_2] \mid Q_3)$ where $\mu \in \{\overline{\mathtt{in}}\ n,\ \overline{\mathtt{open}}\ n\}$ and $n \notin \tilde{a}$.

Definition 8 (*Barb bisimulation*). A symmetric binary relation $\mathcal{R}$ on SA processes is called a *barbed bisimulation* if $P\ \mathcal{R}\ Q$ implies
(1) For any $P \longrightarrow_{\mathrm{SA}} P'$, there is Q' s.t. $Q \longrightarrow^*_{\mathrm{SA}} Q'$ and $P'\ \mathcal{R}\ Q'$;
(2) $P \Downarrow_n \Longrightarrow Q \Downarrow_n$.
We write $\approx$ the union of all barbed bisimulation, which is itself a barbed bisimulation.

We let a *wagon context* to be the usual notion of context with a *hole* "$-$" as a place-holder for a wagon process. For context C and process P, $C(P)$ is the process obtained by filling the hole of C with P. By assuming that $\{\cdot\}$ maps a wagon hole to an SA hole, we know that for any wagon context C, the corresponding encoding $[\![C]\!]$ is an SA context. For the operational correspondence result, we use a typed barbed congruence relation with respect to the encoding, $\approx^{\mathrm{c}}_{[\![\cdot]\!]}$, s.t. $P \approx^{\mathrm{c}}_{[\![\cdot]\!]} Q \iff [\![C]\!](P) \approx [\![C]\!](Q)$, for any wagon context C.

Lemma 9. *For any wagon header M and wagon processes P, Q, we have*

$$pkt[\{M\} \mid \{P\}^L \mid \{Q\}^L] \mid !\mathtt{open}\ pkt \mid !\mathtt{open}\ lock \approx^{\mathrm{c}}_{[\![\cdot]\!]} [\![M\langle P \mid Q\rangle]\!].$$

Proof. Call the left-side process R and the right-side one S. We need to show that for any wagon context C, $[\![C]\!](R) \approx [\![C]\!](S)$. Let $R_1 = \{P\}^L \mid \{Q\}^L$, $R_2 = \{P \mid Q\}^L$. We prove by showing that the relation

$$\mathcal{R} =_{\mathrm{def}} \{(D(R_1),\ D(R_2)) \mid \mathrm{pred}(D)\} \cup \approx$$

is a barbed bisimulation, where the condition $\mathrm{pred}(D)$ on arbitrary SA context D is defined as $\exists P.([\![P]\!] \longrightarrow^*_{\mathrm{SA}} D(\{\mathbf{0}\}^L)$. Clearly, $[\![C]\!](R)\ \mathcal{R}\ [\![C]\!](S)$. The diagram chasing on relation $\mathcal{R}$ is straightforward. For example, if the move $D(R_1) \longrightarrow_{\mathrm{SA}} R'$ involves R_1, the only possibility, by the definition of $[\![\cdot]\!]$, is that there is a process $!\mathtt{open}\ lock$ running in parallel with the hole in D. Since our encoding is typable, we can apply (a variant of) the equational law [15, Lemma 7.5 (3)] and get $R' \approx D(R_2)$, and thus $R'\ \mathcal{R}\ D(R_2)$. Similarly for any move of $D(R_2)$ involving R_2. □

Theorem 10 (*Operational correspondence*). *For any wagon process P, we have*
(1) *If $P \longrightarrow Q$, then $[\![P]\!] \longrightarrow^*_{\mathrm{SA}} \approx^{\mathrm{c}}_{[\![\cdot]\!]} [\![Q]\!]$;*
(2) *If $[\![P]\!] \longrightarrow_{\mathrm{SA}} Q$, then there is wagon process R s.t. $P \longrightarrow^* R$ and $[\![R]\!] \approx^{\mathrm{c}}_{[\![\cdot]\!]} Q$.*

Proof (*Sketch*). The first part is easy, using Lemma 9. The second part is by analyzing the structure of P. The relation $\approx^{\mathrm{c}}_{[\![\cdot]\!]}$ is obtained by applying the typed equational laws of SA and Lemma 9. □

4. The encoding of name-passing

As explained in the introduction, one of the main concerns of the wagon calculus is its expressiveness. In the rest of the paper, we focus on the encoding of π-calculus in wagon and its correctness proof.

Our source language is the *asynchronous finite π-calculus* (π_{af} for short) with the following syntax:

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

We also use $a,\ b,\ \ldots$ to range over the set of channel names: they later have one-to-one correspondence to location names in wagon. 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 $\longrightarrow_\pi$ is given by the rule:

$$a\langle c\rangle \mid a(b).p \longrightarrow_\pi p\{c/b\}. \tag{5}$$

Process $p\{c/b\}$ stands for the capture-free substitution of all the free names b in p with c. We use $\equiv_\pi$ for the usual structural congruence relation between π_{af}-processes. Readers may refer to the book [26] for a more systematic presentation.

Conventions: We assume that meta-notations like name substitution always bind tighter than other language constructs. So process $p \mid (va)q\{c/b\}$ stands for $(p \mid (va)(q\{c/b\}))$. We use $X,\ Y,\ Z$ to denote finite sets of names.

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

Fig. 4. The encoding of π_{af} into wagon: $\langle\!\langle\cdot\rangle\!\rangle$.

Suppose $X = \{a_1, \ldots, a_k\}$, we abbreviate $(va_1)\ldots(va_k)P$ as $(vX)P$. We write $X\backslash Y$ for set minus. We write $\prod_{i=1}^{k} P_i$ for $P_1 \mid \ldots \mid P_k$ and $\prod_{a\in X} P$ for $P\{a_1/a\} \mid \ldots \mid P\{a_k/a\}$ where $X = \{a_1, \ldots, a_k\}$.

To encode name-passing in a calculus without built-in communication primitives, the main difficulty lies in the simulation of the meta-level substitution operation. This is less straightforward than the encodings in the seminal papers [7,15] of ambients, where communication is taken as primitive. Using SA without communication, Zimmer first achieved his encoding of name-passing using a middle calculus where substitutions and channels are both made explicit. He then uses ambients to simulate explicit substitutions and channels.

We present below an encoding of name-passing following Zimmer's framework but using only wagon primitives. Surprisingly enough, using wagon as a kind of syntax sugar for SA, we get a much concise presentation of the encoding, using processes similar to the routing server of Example 4.

Definition 11 (*The encoding*). Suppose that $\mathcal{N}$ contains the set of channel names, and that there is a special name $comm \in \mathcal{N}$ that is not in the set of channel names, the encoding of π_{af} in wagon, $\langle\!\langle\cdot\rangle\!\rangle$, is defined in Fig. 4.

For every channel a, we create a wagon location a called *queue* where I/O-processes on channel a meet and interact. Since there should be one and only one queue provided for each channel, we choose to present the encoding in two stages. For any π_{af}-process p, $[\![p]\!]$ is the wagon process where every private channel is provided with a queue but none for the free ones, while $\langle\!\langle p\rangle\!\rangle_X$ is the process where queues associated with channel names in set X are provided. The final encoding $\langle\!\langle p\rangle\!\rangle$ is simply a shorthand for $\langle\!\langle p\rangle\!\rangle_{fn(p)}$.

To get a first impression of how the encoding works, we show the simulation of the communication rule (5). The encoding of the left-hand side is (where $X = fn(a\langle c\rangle \mid a(b).p)\backslash\{a\}$):

$$
\begin{array}{rcl}
\langle\!\langle a\langle c\rangle \mid a(b).p\rangle\!\rangle & =_{\text{def}} & a.\mathtt{put}\langle comm.\mathtt{put}\langle \mathbf{fwd}\ c\rangle\rangle \\
 & \mid & (vb)(b[\,] \mid a.\mathtt{put}\langle comm.\mathtt{get}.\uparrow.\uparrow.b\langle\uparrow\langle[\![p]\!]\rangle\rangle\rangle) \\
 & \mid & a[\mathbf{chn}] \mid \prod_{a\in X} a[\mathbf{chn}].
\end{array} \tag{6}
$$

We give below the reductions of (6) into process $(vb)(b[\mathbf{fwd}\ c] \mid [\![p]\!]) \mid a[\mathbf{chn}] \mid \prod_{a\in X} a[\mathbf{chn}]$ (omitting queues $\prod_{a\in X} a[\mathbf{chn}]$), which will be shown to be equivalent to $[\![p\{c/b\}]\!] \mid a[\mathbf{chn}] \mid \prod_{a\in X} a[\mathbf{chn}]$. The latter may have one extra queue $a[\mathbf{chn}]$ than the encoding of the right-hand side $\langle\!\langle p\{c/b\}\rangle\!\rangle$, and is not essential to the operational correspondence.

$$
\begin{array}{rl}
 & a.\mathtt{put}\langle comm.\mathtt{put}\langle \mathbf{fwd}\ c\rangle\rangle \mid (vb)(b[\,] \mid a.\mathtt{put}\langle comm.\mathtt{get}.\uparrow.\uparrow.b\langle\uparrow\langle[\![p]\!]\rangle\rangle\rangle) \mid a[\mathbf{chn}] \\
\longrightarrow^2 & (vb)(b[\,] \mid a[\mathbf{chn} \mid \mathtt{put}\langle comm.\mathtt{put}\langle \mathbf{fwd}\ c\rangle\rangle \mid \mathtt{put}\langle comm.\mathtt{get}.\uparrow.\uparrow.b\langle\uparrow\langle[\![p]\!]\rangle\rangle\rangle]) & (7) \\
\longrightarrow^2 & (vb)(b[\,] \mid a[\mathbf{chn} \mid comm.\mathtt{put}\langle \mathbf{fwd}\ c\rangle \mid comm.\mathtt{get}.\uparrow.\uparrow.b\langle\uparrow\langle[\![p]\!]\rangle\rangle]) & (8) \\
\longrightarrow^2 & (vb)(b[\,] \mid a[\,!\mathtt{get}\langle\rangle \mid comm[\mathtt{put}\langle \mathbf{fwd}\ c\rangle \mid \mathtt{get}.\uparrow.\uparrow.b\langle\uparrow\langle[\![p]\!]\rangle\rangle]]) & (9) \\
\longrightarrow & (vb)(b[\,] \mid a[\,!\mathtt{get}\langle\rangle \mid comm[\uparrow.\uparrow.b\langle \mathbf{fwd}\ c \mid\uparrow\langle[\![p]\!]\rangle\rangle]]) & (10) \\
\longrightarrow & (vb)(b[\,] \mid a[\mathbf{chn} \mid\uparrow.b\langle \mathbf{fwd}\ c \mid\uparrow\langle[\![p]\!]\rangle\rangle]) & (11) \\
\longrightarrow & (vb)(b[\,] \mid b\langle \mathbf{fwd}\ c \mid\uparrow\langle[\![p]\!]\rangle\rangle) \mid a[\mathbf{chn}] & (12) \\
\longrightarrow & (vb)(b[\mathbf{fwd}\ c \mid\uparrow\langle[\![p]\!]\rangle]) \mid a[\mathbf{chn}] & (13) \\
\longrightarrow & (vb)(b[\mathbf{fwd}\ c] \mid [\![p]\!]) \mid a[\mathbf{chn}]. & (14)
\end{array}
$$

After the communication, a private *forwarder* $b[\mathbf{fwd}\ c]$ is created, serving as an explicit substitution for the continuation process $[\![p]\!]$. Process $(vb)(b[\mathbf{fwd}\ c] \mid [\![p]\!])$ behaves almost identical to $[\![p\{c/b\}]\!]$, since any I/O-request of p on b will be turned into a same request but on c through the forwarder:

$$\begin{aligned} & (vb)(b[\mathbf{fwd}\ c] \mid b.\mathtt{put}\langle\ldots\rangle) \\ \longrightarrow\ & (vb)(b[\,!\mathtt{get}.\uparrow.c.\mathtt{put}\langle\rangle \mid \mathtt{put}\langle\ldots\rangle]) && (15)\\ \longrightarrow\ & (vb)(b[\,!\mathtt{get}.\uparrow.c.\mathtt{put}\langle\rangle \mid \uparrow.c.\mathtt{put}\langle\ldots\rangle]) && (16)\\ \longrightarrow\ & (vb)(b[\mathbf{fwd}\ c] \mid c.\mathtt{put}\langle\ldots\rangle). && (17) \end{aligned}$$

Given some reasonable equivalence relation $\simeq$, 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 *deterministic intermediate steps*, or *auxiliary reductions*, which includes all reductions except the `get`/`put` interactions inside *comm* (cf. the reduction that leads to (10)). These reductions will never be prevented by interferences from the other reductions. For this, we first classify reductions used in the encoding into the followings:

(1) Reductions using **(R-Out)**: no interference could happen and these reductions always succeed. These include (11), (12), (14), and (17).
(2) Reductions using **(R-In)**: no interference could happen if we can ensure that there is no duplicated sibling location names. These include (7), (9), (13), and (15).
(3) Reductions using **(R-Get-Put)** and involving a replicated `get`-packet: no interference could happen if the get packet is *uniform* and *receptive* [25], in the sense that there is no possibility of other `get`-packets appearing at the same location. These include (8) and (16).
(4) Other reductions using **(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}. This happens in (10).

To formally verify that these non-interference properties hold, we introduce a novel spatial logic, which can specify for example that *all* `get`-packets in *comm* are of the form $\mathtt{get}.\uparrow.\uparrow.b\langle\mathbb{A}\rangle$, where b is some private location name and $\mathbb{A}$ is another logical formula specifying the property of the payload. We then use this logic to formalize the general shape of the encoding, in a quite complex formula named $\mathbb{CPI}_c$ (Definition 24), and prove that the first three kinds of reductions are indeed included in some equivalence (Lemma 38).

For presentation reasons, the logic is built on top of a chemical form [2] of wagon processes, which we introduce in the next section.

5. 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 that we have in a distributed system a mobile agent P and its base Q sharing a secret key k, we have to use the following configuration when the agent is inside some foreign system $a[R]$:

$$(vk)(Q \mid a[P \mid R]). \tag{18}$$

In this section, we introduce a chemical semantics where a process $a[P \mid R]$ can be break down into $P@a \mid R@a$. Thus, (18) could be rewritten as follows, with the binder shrunk as desired:

$$(vk)(Q \mid P@a) \mid R@a.$$

Remark 12. For all this to work, a simple requirement on the naming of the location tree should be imposed. There should be no duplicated sibling names in the location tree. For the wagon calculus, this could be easily formalized using a small typed system (see Appendix A). In the rest of this paper, we will assume that all the wagon processes (and later wagon solutions) satisfy this requirement.

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

Fig. 5. The chemical structural congruence relation.

Definition 13 (*Wagon solution*). *Wagon solutions* $(A,\ B,\ \ldots)$ are defined by adding a singleton name construct [7] (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 "$(\cdot)@(\cdot)$" has a binding power tighter than "$(\cdot) \mid (\cdot)$", but less than any of the other constructs.

Definition 14 (*Chemical structural congruence*). The *chemical structural congruence* relation ($\rightleftharpoons$) between wagon solutions is defined as the least congruence satisfying the rules in Fig. 5.

Only the last three rules are new compared to $\equiv$, and it is not difficult to show that

Lemma 15. *For any wagon processes P and Q*, $P \equiv Q \iff P \rightleftharpoons Q$.

Proof. By introducing a partial function that *cools down* (good) solutions back to processes, and showing that the function preserves structural relation. □

By breaking down processes into smaller parts, we no longer require reductions to happen inside locations, and can define *evaluation contexts* just like those in calculi without a hierarchical location tree.

Definition 16 (*Evaluation context*). An *evaluation context* $\mathcal{C}$ for wagon solutions is of the form $(\nu X)(A \mid -)$, with "$-$" a *hole*. For any solution B, $\mathcal{C}(B)$ is defined as the solution obtained by filling the hole in $\mathcal{C}$ with B.

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

Definition 17 (*Chemical reduction*). The reduction relation ($\hookrightarrow$) between chemical solutions is defined by the rules in Fig. 6.

A correspondence result of the two semantics is due.

[7] We need to keep the singleton name since otherwise it would be hard to define a reduction rule that matches **(R-In)**.

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

Fig. 6. The chemical reduction relation.

Proposition 18. *For any wagon processes P and Q, and wagon solution A, we have*
(1) *If $P \longrightarrow Q$, then $P \hookrightarrow Q$;*
(2) *If $P \hookrightarrow A$, then there is Q s.t. $P \longrightarrow Q$ and $Q \rightleftharpoons A$.*

Proof (*Sketch*).
(1) By induction on the derivation of $P \longrightarrow Q$, using Lemma 15.
(2) By the transitivity of $\rightleftharpoons$ we can suppose that $P \hookrightarrow A$ is caused by $P \rightleftharpoons A' \hookrightarrow A'' \rightleftharpoons A$ where the derivation of $A' \hookrightarrow A''$ does not use rule **(CR-Struct)**. Then, the proof is by the fact that any packet in A' of the form $M\langle C\rangle@s$ corresponds to some packet $M\langle R\rangle$ in P, located in a location denoted by s, with $C \rightleftharpoons R$. Due to the unique sibling name assumption, we can prove by induction on the derivation of $A' \hookrightarrow A''$ that any $A' \hookrightarrow A''$ can be matched by a corresponding $P \longrightarrow Q$ with $Q \rightleftharpoons A''$. Then the result is by Lemma 15. □

6. Spatial logic

In process calculi, names and binders are omnipresent. In this section, we introduce a spatial logic that classifies names and binders into different groups, specifies their usage patterns, and checks these patterns against processes. The basic idea is like this: formulas are constructed from processes by replacing names with name sets; a process satisfies a formula if their structures match, up to structural rearrangements, and names in the process are included in the corresponding sets of the formula. To get some intuition, consider the following π-process:

$$!prn1(job).\mathbf{0} \mid !prn2(job).\mathbf{0} \mid \overline{prn1}\langle some\_paper\rangle. \tag{19}$$

We have two printer processes that repeatedly receives jobs from the two ports $prn1$ and $prn2$, respectively, and does some internal processing with the job (represented by the empty process $\mathbf{0}$). A client process is trying to use the first printer to print some paper. Suppose we are given a set of valid port names $X = \{prn1,\ prn2\}$ and a set Y of valid job names, with $some\_paper \in Y$. To say that the client can use whatever printers and print anything specified by Y, we may do replacements in (19) and obtain the formula

$$!prn1(job).\mathbf{0} \mid !prn2(job).\mathbf{0} \mid \overline{X}\langle Y\rangle,$$

or, in a more concise presentation,

$$\prod_{p\in X} !p(job).\mathbf{0} \mid \overline{X}\langle Y\rangle, \tag{20}$$

which matches (19). Or, using a more strict specification, we can only allow the client to use one printer:

$$\prod_{p\in X} !p(job).\mathbf{0} \mid \overline{\{prn1\}}\langle Y\rangle.$$

Nothing is very interesting here. To allow more clients with the same specification, we further introduce the Kleene star construct to formulas. Adding it to (20), we obtain

$$\prod_{p\in X} !p(job).\mathbf{0} \mid (\overline{X}\langle Y\rangle)^*, \tag{21}$$

where $(\overline{X}\langle Y\rangle)^*$ matches zero or more parallel processes satisfying formula $\overline{X}\langle Y\rangle$. One interesting point about (21) is that it is reduction closed: any reduction of a matched process decreases the number of clients by one, which is still matched by (21); for processes without clients, no reduction is possible.

The more interesting part of the logic lies in the treatment of name binders. Suppose now all printer names are protected, as in process

$$(\nu prn1,\ prn2)(!prn1(job).\mathbf{0} \mid !prn2(job).\mathbf{0} \mid \overline{prn1}\langle some\_paper\rangle), \tag{22}$$

then the specification

$$(\nu X)\left(\prod_{p\in X} !p(job).\mathbf{0} \mid \overline{X}\langle Y\rangle^*\right) \tag{23}$$

says that there are two private printers and some clients using them to print jobs from Y. In our logic, we also introduce a more powerful construct, the *fresh set binder* $(\gamma x)(\cdot)$. We can change (23) to

$$(\gamma x)\left(\prod_{p\in x} !\{p\}(job).\mathbf{0} \mid \overline{x}\langle Y\rangle^*\right). \tag{24}$$

When checking (22) against (24), (γx) matches the binders $(\nu prn1,\ prn2)$, as the (νX) in (23) does. Then all occurrences of the set variable x in the rest of the formula are substituted by $\{prn1,\ prn2\}$ and the match goes on. The power of this binder lies in that it can match any fresh name set. As a matter of fact, (24) can match any process that has some fixed number of internal printers and zero or more clients who are allowed to use any of these printers.

6.1. The spatial logic for wagon solutions

Applying the above idea to wagon headers and solutions, we obtain the spatial logic for wagon solutions.

Definition 19 (*Wagon formula*). *Wagon formulas* ($\mathbb{A},\ \mathbb{B},\ \ldots$) are sets of wagon solutions. *Header formulas* ($\mathbb{M},\ \mathbb{N},\ \ldots$) are sets of wagon headers. Given a set of set variables $\mathcal{V}$ ranged over by $x,\ y,\ \ldots$, they are defined by the following grammar [8]:

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

We use $u,\ v,\ \ldots$ to denote a set, built by the union of set variables (x) and set constants (X). A header formula is just a header with names replaced by sets. A wagon formula, or simply formula, is obtained similarly by changing names into sets, with some additional constructs that we will explain later. When no confusion arises, we sometimes write a formula like $\{a\}.\mathtt{get}.\{b\}.\mathtt{put}\langle\rangle$ as the solution $a.\mathtt{get}.b.\mathtt{put}\langle\rangle$.

We have two different kinds of binders in wagon formulas: "$(\nu a)(\cdot)$" and "$\prod_{a\in u}(\cdot)$" are the binders of name a; "$(\gamma x)(\cdot)$" is the binder of variable x. Free names $fn(\cdot)$, free variables $fv(\cdot)$, and substitutions $((\cdot)\{b/a\},(\cdot)\{u/x\})$ are defined accordingly. Note that substituting a name in a set with another name in that set may result in the first name being absorbed.

Substituting set variables with name sets is more delicate. Apart from the usual name capture, like in $((\nu a)(a[\mathbf{0}] \mid x.\mathtt{put}\langle\rangle))\{\{a,b\}/x\}$, we also assume that a set variable represents a fresh name set disjoint with any other set variables or free names in the formula. As a result, only a set with all the names fresh can be used in substitutions. This point will be more clear when the semantics of the bind $(\gamma x)(\cdot)$ is discussed.

---

[8] For the sake of the usage in this paper, we do not include the basic connectives, e.g. $\neg$ and $\wedge$. 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 & \text{there is } N \in \mathbb{M} \text{ s.t. } M = \mathtt{get}.N \\
M \in \uparrow.\mathbb{M} & \Longleftrightarrow & \text{there is } N \in \mathbb{M} \text{ s.t. } M = \uparrow.N \\
M \in u.\mathbb{M} & \Longleftrightarrow & \text{there are } a \in u \text{ and } N \in \mathbb{M} \text{ s.t. } M = a.N \\
 & & \\
A \in \mathbf{0} & \Longleftrightarrow & A \rightleftharpoons \mathbf{0} \\
A \in \mathbb{M}\langle\mathbb{A}\rangle & \Longleftrightarrow & \text{there are } M \in \mathbb{M} \text{ and } B \in \mathbb{A} \text{ s.t. } A \rightleftharpoons M\langle B\rangle, \text{ if } \mathbb{M} \neq \emptyset; \\
 & & A \rightleftharpoons \mathbf{0}, \text{ otherwise.} \\
A \in !\mathbb{M}\langle\mathbb{A}\rangle & \Longleftrightarrow & \text{there are } M \in \mathbb{M} \text{ and } B \in \mathbb{A} \text{ s.t. } A = !M\langle B\rangle, \text{ if } \mathbb{M} \neq \emptyset; \\
 & & A \rightleftharpoons \mathbf{0}, \text{ otherwise.} \\
A \in (\nu a)\mathbb{A} & \Longleftrightarrow & \text{there is } B \in \mathbb{A} \text{ s.t. } A \rightleftharpoons (\nu a)B \\
A \in (\mathbb{A} \mid \mathbb{B}) & \Longleftrightarrow & \text{there are } B_1 \in \mathbb{A} \text{ and } B_2 \in \mathbb{B} \text{ s.t. } A \rightleftharpoons B_1 \mid B_2 \\
A \in a[\mathbb{A}] & \Longleftrightarrow & \text{there is } B \in \mathbb{A} \text{ s.t. } A \rightleftharpoons a[B] \\
A \in a & \Longleftrightarrow & A \rightleftharpoons a \\
A \in \mathbb{A}@u & \Longleftrightarrow & \text{there are } B \in \mathbb{A} \text{ and } a \in u \text{ s.t. } A \rightleftharpoons B@a, \text{ if } u \neq \emptyset; \\
 & & A \rightleftharpoons \mathbf{0}, \text{ otherwise.} \\
A \in \mathbb{A}^* & \Longleftrightarrow & \text{there are } k \geq 0 \text{ and } A_1, \cdots, A_k \in \mathbb{A} \text{ s.t. } A \rightleftharpoons \prod_{i=1..k} A_i \\
A \in \prod_{a \in u} \mathbb{A} & \Longleftrightarrow & \text{there are } A_1, \cdots A_k \text{ s.t. } A_i \in \mathbb{A}\{a_i/a\} \ (i = 1..k) \\
 & & \text{and } A \rightleftharpoons \prod_{i=1..k} A_i, \text{ given } u = \{a_1, \cdots, a_k\} \\
A \in (\gamma x)\mathbb{A} & \Longleftrightarrow & \text{there is } X \text{ with } X \cap fn(\mathbb{A}) = \emptyset \text{ s.t. we can find} \\
 & & B \in \mathbb{A}\{X/x\} \text{ satisfying } A \rightleftharpoons (\nu X)B \\
A \in \mathbb{A}(\vec{u}) & \Longleftrightarrow & A \in \mathbb{B}\{\vec{u}/\vec{x}\}, \text{ if } \mathbb{A} \text{ is defined as } \mathbb{A}(\vec{x}) = \mathbb{B}
\end{array}
$$

Fig. 7. The denotation of header formulas and wagon formulas.

Conventions: For formula constructs, we assume the order of precedence as "$(va)(\cdot)$", "$(\gamma x)(\cdot)$", "$(\cdot)^*$", "$(\cdot)@(\cdot)$", "$\prod_{a\in u}(\cdot)$", and finally "$(\cdot) \mid (\cdot)$".

Definition 20 (*Denotation and satisfaction*). The denotations of formulas are given in Fig. 7, defined up to $\rightleftharpoons$. The satisfaction relation is defined as usual: $A \vDash \mathbb{A} \iff A \in \mathbb{A}$. When $A \in \mathbb{A}$, 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, where only the names of the move-in actions could be chosen among those sets specified by $\mathbb{M}$. $\mathbb{M}$ is defined to be the empty-set if one of the associated set is empty. The semantics of wagon formulas are defined similarly. 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 some detailed explanations.

- $\mathbb{A}^*$: We use the Kleene star construct to specify processes that can be break down into components that all satisfy $\mathbb{A}$.
- $\prod_{a\in u}\mathbb{A}$: 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 (e.g. $\prod_{a\in u} a[\mathbf{0}]$).

- $(\gamma x)\mathbb{A}$: We introduce in our logic this *fresh set binder*, read "given a fresh set x", to specify processes in which some particular fresh name set X can be identified. After extending the scopes of all these binders to the out-most, the inner process satisfies the formula obtained by substituting x in $\mathbb{A}$ with X. In particular, X can be an empty set. Note that the freshness is formalized by the condition that the identified set X cannot contain any free names in $\mathbb{A}$.
- $\mathbb{A}(\vec{u})$: This is the usual way of calling a *definition clause*, say, $\mathbb{A}(\vec{x}) =_{\text{def}} \mathbb{B}$. Then this formula is equal to $\mathbb{B}\{\vec{u}/\vec{x}\}$. We allow recursive definition clauses. A definition clause must be *closed*, in that the definition body can only contains free variables from the parameter list.

Example 21. We give a few satisfaction examples below:
(1) $\mathtt{get}.\uparrow.a\langle\rangle \in \mathtt{get}.\uparrow.\{a,b\}\langle\rangle$;
(2) $!\mathtt{get}\langle\rangle \mid \mathtt{put}\langle\rangle \mid \mathtt{put}\langle\rangle \in !\mathtt{get}\langle\rangle \mid \mathtt{put}\langle\rangle^*$;
(3) $\{\mathbf{0},\ (va)a[\,],\ (va)(vb)(a[\,] \mid b[\,]),\ \cdots\} \subseteq (\gamma x)\prod_{a\in x} a[\,]$;
(4) $(va)(vb)(a[\,] \mid b[\,] \mid \mathtt{get}.a\langle\rangle) \in (\gamma x)(\prod_{a\in x} a[\,] \mid \mathtt{get}.x\langle\rangle^*)$.

6.2. Logical formulas for the encoding

In this section, we define the formula $\mathbb{CPI}_c$ (Definition 24) that contains all the derivations of the encoding (Corollary 28). The formula gives a global vision containing all the possible interaction patterns of any encoded π_{af}-processes. As a result, the definition is quite complex and depends on quite a lot of intermediate definition clauses, each corresponds to an intermediate state in the computation.

We first define $\mathbb{PI}_c(x)$, which characterizes the encoding of any π_{af}-process with free names in x, before any reduction happens.

Conventions: When no confusion arises, we often abbreviate $x \cup y$ as xy and $x \cup \{b\}$ as $x{\cdot}b$ for the sake of simplicity.

Definition 22. We define the following formulas for the encoding of π_{af}:

$$
\begin{aligned}
\mathbb{C}(x) &=_{\text{def}} \prod_{a\in x} a[\,!\mathtt{get}\langle\rangle \mid \mathit{comm}] \\
\mathbb{F}(x) &=_{\text{def}} !\mathtt{get}.\uparrow.x.\mathtt{put}\langle\rangle \\
\mathbb{I}_1(x) &=_{\text{def}} (vb)(b \mid x.\mathtt{put}\langle \mathit{comm}.\mathtt{get}.\uparrow.\uparrow.b\langle\uparrow\langle\mathbb{PI}(x{\cdot}b)\rangle\rangle\rangle) \\
\mathbb{O}_1(x) &=_{\text{def}} x.\mathtt{put}\langle \mathit{comm}.\mathtt{put}\langle\mathbb{F}(x)\rangle\rangle \\
\mathbb{PI}(x) &=_{\text{def}} (\gamma y)(\mathbb{C}(y) \mid \mathbb{I}_1(xy)^* \mid \mathbb{O}_1(xy)^*) \\
\mathbb{PI}_c(x) &=_{\text{def}} \mathbb{PI}(x) \mid \mathbb{C}(x)
\end{aligned}
$$

Informally, a bunch of (empty) queues of names X is a $\mathbb{C}(X)$-solution, i.e. $\prod_{a\in X} a[\mathbf{chn}] \in \mathbb{C}(X)$. The replicated $\mathtt{get}$-packet to be used inside forwarders, with its destination belonging to some set X, is an $\mathbb{F}(X)$-solution. The encoding of any input (resp. output) process having free names X is an $\mathbb{I}_1(X)$-solution (resp. $\mathbb{O}_1(X)$-solution). The encoding of any π_{af}-process having free names X and bound names Y is made of a $\mathbb{C}(X)$-solution and a $\mathbb{PI}(X)$-solution, which has, under binders (vY), the parallel composition of a bunch of queues of names Y ($\mathbb{C}(Y)$), some input solutions with free names $X \cup Y$ ($\mathbb{I}_1(XY)^*$), and some output solutions with free names $X \cup Y$ ($\mathbb{O}_1(XY)^*$).

Lemma 23. *For any π_{af}-process p, $[\![p]\!] \in \mathbb{PI}(fn(p))$ and $\langle\!\langle p\rangle\!\rangle \in \mathbb{PI}_c(fn(p))$.*

Proof. See Appendix C. □

We now give the definition of $\mathbb{CPI}_c$. In the definition, formulas $\mathbb{I}_{2..4}$ and $\mathbb{O}_{2..4}$ correspond to the three intermediate states of the I/O-packets (cf. reductions (7), (8), and (9) in Section 4) before their interaction inside *comm*. $\mathbb{I}_0$ and $\mathbb{O}_0$ correspond to one of the two intermediate states of I/O-packets inside forwarders (cf. reduction (16) in Section 4). The other is shared by $\mathbb{I}_2$ and $\mathbb{O}_2$ (cf. reduction (15) in Section 4). Finally, $\mathbb{I}_{5..8}$ represent the intermediate states starting from the interaction in *comm* till the building of the forwarder (cf. reductions (10), (11), (12), (13) in Section 4).

Definition 24. We introduce the following formula definitions [9]:

$$
\begin{aligned}
\mathbb{I}_0(x,\ y) &=_{\text{def}} (vb)(b \mid \uparrow.\mathtt{put}\langle comm.\mathtt{get}.\uparrow.\uparrow.b\langle\uparrow\langle\mathbb{PI}(x\cdot b)\rangle\rangle\rangle@y)\\
\mathbb{I}_2(x) &=_{\text{def}} (vb)(b \mid \mathtt{put}\langle comm.\mathtt{get}.\uparrow.\uparrow.b\langle\uparrow\langle\mathbb{PI}(x\cdot b)\rangle\rangle\rangle@x)\\
\mathbb{I}_3(x,\ y) &=_{\text{def}} (vb)(b \mid comm.\mathtt{get}.\uparrow.\uparrow.b\langle\uparrow\langle\mathbb{PI}(x\cdot b)\rangle\rangle@y)\\
\mathbb{I}_4(x,\ y) &=_{\text{def}} (vb)(b \mid \mathtt{get}.\uparrow.\uparrow.b\langle\uparrow\langle\mathbb{PI}(x\cdot b)\rangle\rangle@comm@y)\\
\mathbb{I}_5(x,\ y) &=_{\text{def}} (vb)(b \mid \uparrow.\uparrow.b\langle\mathbb{F}(x) \mid \uparrow\langle\mathbb{PI}(x\cdot b)\rangle\rangle@comm@y)\\
\mathbb{I}_6(x,\ y) &=_{\text{def}} (vb)(b \mid \uparrow.b\langle\mathbb{F}(x) \mid \uparrow\langle\mathbb{PI}(x\cdot b)\rangle\rangle@y)\\
\mathbb{I}_7(x) &=_{\text{def}} (vb)(b \mid b\langle\mathbb{F}(x) \mid \uparrow\langle\mathbb{PI}(x\cdot b)\rangle\rangle)\\
\mathbb{I}_8(x,\ y) &=_{\text{def}} \uparrow\langle\mathbb{PI}(x)\rangle@y\\
\mathbb{O}_0(x,\ y) &=_{\text{def}} \uparrow.x.\mathtt{put}\langle comm.\mathtt{put}\langle\mathbb{F}(x)\rangle\rangle@y\\
\mathbb{O}_2(x) &=_{\text{def}} \mathtt{put}\langle comm.\mathtt{put}\langle\mathbb{F}(x)\rangle\rangle@x\\
\mathbb{O}_3(x,\ y) &=_{\text{def}} comm.\mathtt{put}\langle\mathbb{F}(x)\rangle@y\\
\mathbb{O}_4(x,\ y) &=_{\text{def}} \mathtt{put}\langle\mathbb{F}(x)\rangle@comm@y\\
\mathbb{V}(x,\ y) &=_{\text{def}} \prod_{a\in y} a[\mathbb{F}(xy)]\\
\mathbb{D}(x,\ y) &=_{\text{def}} \mathbb{I}_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)^*\\
&\quad\ \mid \quad \mathbb{I}_5(xy,\ x)^* \mid \mathbb{I}_6(xy,\ x)^* \mid \mathbb{I}_7(xy)^* \mid \mathbb{I}_8(xy,\ y)^*\\
&\quad\ \mid \quad \mathbb{O}_0(xy,\ y)^* \mid \mathbb{O}_1(xy)^* \mid \mathbb{O}_2(xy)^* \mid \mathbb{O}_3(xy,\ x)^* \mid \mathbb{O}_4(xy,\ x)^*\\
\mathbb{CPI}(x) &=_{\text{def}} (\gamma y)(\gamma z)(\mathbb{C}(y) \mid \mathbb{V}(xy,\ z) \mid \mathbb{D}(xy,\ z))\\
\mathbb{CPI}_c(x) &=_{\text{def}} \mathbb{CPI}(x) \mid \mathbb{C}(x)
\end{aligned}
$$

It is easy to show that every $\mathbb{PI}_c(X)$-solution is also a $\mathbb{CPI}_c(X)$-solution.

Lemma 25. *For any X,* (1) $\mathbb{PI}(X) \subseteq \mathbb{CPI}(X)$; *and (2)* $\mathbb{PI}_c(X) \subseteq \mathbb{CPI}_c(X)$.

Definition 26 (*Closure*). Formula $\mathbb{A}$ is called a *closure*, if for any $A \in \mathbb{A}$ and A', $A \hookrightarrow^* A'$ implies $A' \in \mathbb{A}$.

As a simple example, the formula in Example 21.(2) is a closure.
Conventions: If $\mathbb{A}(\vec{x}) =_{\text{def}} \mathbb{B}$ is a definition, we use $\mathbb{A}$ to denote the union:

$$\bigcup_{(\uplus_i X_i)\subseteq(\mathcal{N}\setminus fn(\mathbb{B}))} \mathbb{A}(\vec{X}).$$

If $A \in \mathbb{A}$, then $A \in \mathbb{A}(\vec{X})$ for some name sets $X_1, \ldots, X_{|\vec{X}|}$ disjoint with each other and with the free names in $\mathbb{B}$.

Lemma 27. (1) *For any X,* $\mathbb{CPI}_c(X)$ *is a closure.* (2) $\mathbb{CPI}_c$ *is a closure.*

Proof. See Appendix D. Fig. 8 gives an illustration of all the possible packet flows caused by reductions. □

As a corollary of Lemmas 25 and 27, every solution produced by the reduction of an encoding is a $\mathbb{CPI}_c$-solution.

Corollary 28. *For any* π_{af}*-process p, and any solution A s.t.* $\langle\!\langle p\rangle\!\rangle \hookrightarrow^* A$, *we have* $A \in \mathbb{CPI}_c$.

To this end, we can ensure that properties such as the uniformity of replicated `get`-packets inside forwarders or queues are indeed satisfied by our encoding, by the definition of $\mathbb{CPI}_c$.

---

[9] The size of the definition of $\mathbb{CPI}_c$ has been reduced to its current form by: using $\mathtt{dis}\langle P\rangle \equiv P$ instead of $\mathtt{dis}\langle P\rangle \longrightarrow P$ so that the logic does not consider `dis`-packets, and using the finite asynchronous version of the π-calculus.

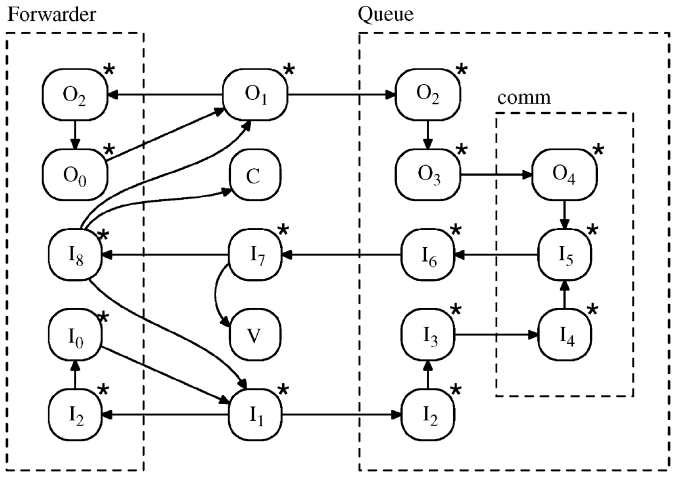

Fig. 8. Closure property of $\mathbb{CPI}_c$.

7. May-testing congruence

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 [18]. Constraints on the translation environment would be another problem.

In our proof, we use may-testing congruence [21,12], which requires that the two processes have exactly the same external observations when being put inside arbitrary evaluation contexts (observers). Barbed bisimulation and congruence [20,15] would be another choice, although constructing appropriate relations in the proof requires different work.

Definition 29 (*Barb*). We write $A \downarrow_s$ if $A \rightleftharpoons (vX)(A_1 \mid \mathtt{get}.M\langle A_2\rangle@s)$ with $fn(s) \cap X = \emptyset$. $A \Downarrow_s \iff A \hookrightarrow^* B \downarrow_s$.

Definition 30. We define (the unrestricted) *may-testing congruence* between wagon solutions as: $A \simeq B$ if and only if for any $\mathcal{C}$ and s, $\mathcal{C}(A) \Downarrow_s \iff \mathcal{C}(B) \Downarrow_s$.

Example 31. Some example of may-testing congruence:
(1) $\uparrow.M\langle A\rangle@a \mid a \simeq M\langle A\rangle \mid a$;
(2) $a.M\langle A\rangle \mid a \simeq M\langle A\rangle@a \mid a$.
The proof is by analyzing all the possible interactions of these processes with the context. The second example is not true without the unique sibling-name requirement (Remark 12).

Sadly, we cannot 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 essential equivalence of

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

would fail if the tester is, for example, $\mathtt{get}.\uparrow\langle\rangle@a \mid -$. This situation is typical of encodings between different calculi. The solution is to consider testers that are translations of source contexts. So we define a restricted version of the above 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 $\mathbb{CPI}^{\circ}(x_1,\ x_2,\ z_1)$ that contains all the solutions that we are going to associate with the congruence. Informally, a $\mathbb{CPI}^{\circ}(x_1,\ x_2,\ z_1)$-solution is a $\mathbb{CPI}(x_1x_2)$-solution with some free queues missing (x_2) and some extra free forwarders (z_1) not yet binded.

$$\mathbb{CPI}^{\circ}(x_1,\ x_2,\ z_1) =_{\text{def}} (\gamma y)(\gamma z_2)(\mathbb{C}(y) \mid \mathbb{V}(x_1x_2yz_1,\ z_2) \mid \mathbb{D}(x_1x_2y,\ z_1z_2)) \mid \mathbb{C}(x_1) \mid \mathbb{V}(x_1x_2,\ z_1).$$

For any $\mathbb{CPI}^{\circ}$-solution A, we write $chn(A)$, $var(A)$ the names of the free $\mathbb{C}$-solution and $\mathbb{V}$-solution in A, respectively, and we let $freeloc(A) = fn(A)\backslash(chn(A) \uplus var(A))$. It is easy to check that any $\mathbb{CPI}_c$-solution is also a $\mathbb{CPI}^{\circ}$-solution ($\mathbb{CPI}_c(X) = \mathbb{CPI}^{\circ}(X,\ \emptyset,\ \emptyset)$). Moreover, from any $\mathbb{CPI}^{\circ}$-solution A, the minimum context required to make $\mathcal{C}(A)$ a $\mathbb{CPI}_c$-solution is $(\nu\, var(A))(\prod_{a\in freeloc(A)} a[!\texttt{get}\langle\rangle \mid comm] \mid -)$.

Definition 32. $\mathbb{CPI}^{\circ}$-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) $chn(A) = chn(B)$, $var(A) = var(B)$, and $freeloc(A) = freeloc(B)$; 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.

Remark 33. We only test barbs at location *comm*, since other `get`-packets are all replicated ones and barbs caused by them are not significant for comparison.

We first show that this relation is a congruence (Lemmas 34 and 35) and contains the chemical structural equivalence (Lemma 36).

Lemma 34 (*Transitivity*). *If* $\mathbb{CPI}_c \vdash A \simeq B$ *and* $\mathbb{CPI}_c \vdash B \simeq C$, *then* $\mathbb{CPI}_c \vdash A \simeq C$.

Proof. By checking the definition. **(May-Loc)** is obvious. For **(May-Test)**: given any $\mathcal{C}$ s.t. $\mathcal{C}(A) \in \mathbb{CPI}_c$ and $\mathcal{C}(C) \in \mathbb{CPI}_c$, we need to show that $\mathcal{C}(A) \Downarrow_{comm@a} \iff \mathcal{C}(C) \Downarrow_{comm@a}$. Consider now $\mathcal{C}(B)$. To use the **(May-Test)** clauses of the premises $\mathbb{CPI}_c \vdash A \simeq B$ and $\mathbb{CPI}_c \vdash B \simeq C$, we need to show that $\mathcal{C}(B)$ is a $\mathbb{CPI}_c$-solution. This is indeed the cases from the requirement of **(May-Loc)**: $chn(A) = chn(B)$, $var(A) = var(B)$, and $freeloc(A) = freeloc(B)$. Since $\mathcal{C}(A) \in \mathbb{CPI}_c$, $\mathcal{C}$ must already supplied the minimum amount of binders $\nu\, var(A)$ and the minimum amount of queues $\prod_{a\in freeloc(A)} a[!get.\langle\rangle \mid comm]$. These fact together with the requirements of **(May-Loc)** makes $\mathcal{C}(B)$ also a $\mathbb{CPI}_c$-solution. □

Lemma 35 (*Congruence*). *Given* $\mathbb{CPI}_c \vdash A \simeq B$, *for any* $\mathcal{C}$ *s.t.* $\mathcal{C}(A)$ *and* $\mathcal{C}(B)$ *are both* $\mathbb{CPI}^{\circ}$*-solutions*, *we have* $\mathbb{CPI}_c \vdash \mathcal{C}(A) \simeq \mathcal{C}(B)$.

Proof. Easy. □

Lemma 36. *Suppose* A *is a* $\mathbb{CPI}^{\circ}$*-solution*, *then* $A \rightleftharpoons B$ *implies* $\mathbb{CPI}_c \vdash A \simeq B$.

Proof. Easy. □

We show in the next example that the congruence is not a trivial total relation: it distinguishes the encoding of the following two different π_{af}-processes.

Example 37. $\mathbb{CPI}_c \vdash [\![a\langle b\rangle]\!] \not\simeq [\![a\langle c\rangle]\!]$

Proof. We let $\mathcal{C} = \mathbb{C}(\{a,\ b,\ c\}) \mid [\![a(x).x(y).\mathbf{0}]\!] \mid -$, then $\mathcal{C}([\![a\langle b\rangle]\!]) \in \mathbb{CPI}_c$ and $\mathcal{C}([\![a\langle c\rangle]\!]) \in \mathbb{CPI}_c$. However, we have $\mathcal{C}([\![a\langle b\rangle]\!] \mid \mathbb{C}(\{a,\ b,\ c\})) \Downarrow_{comm@b}$ but $\mathcal{C}([\![a\langle c\rangle]\!] \mid \mathbb{C}(\{a,\ b,\ c\})) \not\Downarrow_{comm@b}$. □

8. Operational correspondence

We are in the place of presenting the main operational correspondence result. We first prove that all the auxiliary reductions are may-testing congruent under $\mathbb{CPI}_c$. These auxiliary reductions are given as pairs. For each pair of the form $(A,\ B)$, we have $A \hookrightarrow B$. Moreover, each tells the possible reduction caused by some particular $\mathbb{I}/\mathbb{O}$-solution in $\mathbb{CPI}_c$. Only excluded are the `get`/`put` interactions inside *comm*, caused by $\mathbb{I}_4$- or $\mathbb{O}_4$-solutions.

Lemma 38 (*Laws for auxiliary reductions*). *For any of the following pairs of solutions A and B, we have* $\mathbb{CPI}_c \vdash A \simeq B$:

(I - 0)	$A = a[\mathbf{chn}] \mid (vb)(b \mid\uparrow.c.\mathtt{put}\langle A'\rangle@a)\ with\ (vb)(b \mid\uparrow.c.\mathtt{put}\langle A'\rangle@a) \in \mathbb{I}_0$, $B = a[\mathbf{chn}] \mid (vb)(b \mid c.\mathtt{put}\langle A'\rangle)$;
(I - 1)	$A = (vb)(b \mid a.\mathtt{put}\langle A'\rangle) \in \mathbb{I}_1\ and\ B = (vb)(b \mid \mathtt{put}\langle A'\rangle@a)$;
(I - 2.1)	$A = a[\mathbf{chn}] \mid (vb)(b \mid \mathtt{put}\langle A'\rangle@a)\ with\ (vb)(b \mid \mathtt{put}\langle A'\rangle@a) \in \mathbb{I}_2$, $B = a[\mathbf{chn}] \mid (vb)(b \mid A'@a)$;
(I - 2.2)	$A = a[\mathbf{fwd}\ c] \mid (vb)(b \mid \mathtt{put}\langle A'\rangle@a)\ with\ (vb)(b \mid \mathtt{put}\langle A'\rangle@a) \in \mathbb{I}_2$, $B = a[\mathbf{fwd}\ c] \mid (vb)(b \mid\uparrow.c.\mathtt{put}\langle A'\rangle@a)$;
(I - 3)	$A = (vb)(b \mid comm.\mathtt{get}.\uparrow.\uparrow.b\langle A'\rangle@a) \in \mathbb{I}_3$, $B = (vb)(b \mid \mathtt{get}.\uparrow.\uparrow.b\langle A'\rangle@comm@a)$;
(I - 5)	$A = (vb)(b \mid\uparrow.\uparrow.b\langle A'\rangle@comm@a) \in \mathbb{I}_5$, $B = (vb)(b \mid\uparrow.b\langle A'\rangle@a)$;
(I - 6)	$A = a[\mathbf{chn}] \mid (vb)(b \mid\uparrow.b\langle A'\rangle@a)\ with\ (vb)(b \mid\uparrow.b\langle A'\rangle@a) \in \mathbb{I}_6$, $B = a[\mathbf{chn}] \mid (vb)(b \mid b\langle A'\rangle)$;
(I - 7)	$A = (vb)(b \mid b\langle A'\rangle) \in \mathbb{I}_7\ and\ B = (vb)(b \mid A'@b)$;
(I - 8)	$A = \uparrow\langle B\rangle@a \in \mathbb{I}_8$;
(O - 0)	$A = a[\mathbf{chn}] \mid\uparrow.b.\mathtt{put}\langle A'\rangle@a\ with\ \uparrow.b.\mathtt{put}\langle A'\rangle@a \in \mathbb{O}_0$, $B = a[\mathbf{chn}] \mid b.\mathtt{put}\langle A'\rangle$;
(O - 1)	$A = a.\mathtt{put}\langle A'\rangle \in \mathbb{O}_1\ and\ B = \mathtt{put}\langle A'\rangle@a$;
(O - 2.1)	$A = a[\mathbf{chn}] \mid \mathtt{put}\langle A'\rangle@a\ with\ \mathtt{put}\langle A'\rangle@a \in \mathbb{O}_2$, $B = a[\mathbf{chn}] \mid A'@a$;
(O - 2.2)	$A = a[\mathbf{fwd}\ b] \mid \mathtt{put}\langle A'\rangle@a\ with\ \mathtt{put}\langle A'\rangle@a \in \mathbb{O}_2$, $B = a[\mathbf{fwd}\ b] \mid\uparrow.b.\mathtt{put}\langle A'\rangle@a$;
(O - 3)	$A = comm.\mathtt{put}\langle A'\rangle@a \in \mathbb{O}_3\ and\ B = \mathtt{put}\langle A'\rangle@comm@a$.

Proof. We prove by checking the definition. **(May-Loc)** is obvious for all the cases (notice the additional location $a[\mathbf{chn}]$ in **(I-0)**, **(I-6)**, and **(O-0)** to ensure **(May-Loc)**). Regarding **(May-Test)**, for any of the cases above we do the following analysis. For any such $\mathcal{C}$ satisfying $\mathcal{C}(A) \in \mathbb{CPI}_c$ and $\mathcal{C}(B) \in \mathbb{CPI}_c$, we need to prove that $\mathcal{C}(A) \Downarrow_{comm@b} \iff \mathcal{C}(B) \Downarrow_{comm@b}$ for any b. Since we can easily show that $\mathcal{C}(A) \hookrightarrow \mathcal{C}(B)$ for all the cases, we only need to show one direction, i.e. $\mathcal{C}(A) \Downarrow_{comm@b} \Longrightarrow \mathcal{C}(B) \Downarrow_{comm@b}$. We prove by induction on the derivation of $\mathcal{C}(A) \Downarrow_{comm@b}$.

(1) If $\mathcal{C}(A) \downarrow_{comm@b}$, then the barb can only be contributed by $\mathcal{C}$, for all the cases, so $\mathcal{C}(B) \downarrow_{comm@b}$.

(2) If $\mathcal{C}(A) \longrightarrow A_1 \Downarrow_{comm@b}$, then the reduction either involves solution A or not. If not, we have another context $\mathcal{D}$ and we can use induction hypothesis. Otherwise, we analyze the possible reductions involving A for all the cases. For all these, we either show that A_1 is chemical structural congruence to $\mathcal{C}(B)$, which implies $\mathcal{C}(B) \Downarrow_{comm@b}$, or A_1 is of the form $\mathcal{D}(A)$, in which case induction hypothesis applies.

(O-0) One obvious reduction is by $A \hookrightarrow B$ using **(CR-Out)** with A_1 chemically equivalent to $\mathcal{C}(B)$. The only other possible reduction involving A is by some solution in $\mathcal{C}$ interacting with $a[\mathbf{chn}]$. In any case, $a[\mathbf{chn}]$ remains intact. The reduction will produce another context $\mathcal{D}$ and we can use induction hypothesis.

(O-1) Since $a \in fn(A)$ and $\mathcal{C}(A) \in \mathbb{CPI}_c$, we must have $\mathcal{C}(A) \rightleftharpoons (vX)(E \mid a \mid A)$ for some set X and solution E. Then the only possible reduction involving A is by $(vX)(E \mid a \mid A) \hookrightarrow A_1 \rightleftharpoons \mathcal{C}(B)$ using **(CR-In)**.

(O-2.1) By the fact that $\mathcal{C}(A) \in \mathbb{CPI}_c$, there is no $\mathtt{get}$-packet at location a in $\mathcal{C}(A)$ except those from $a[\mathbf{chn}]$. So the only possible reduction involving the $\mathtt{put}$-packet in A is caused by $A \hookrightarrow B$ using **(CR-Get-Put)** with A_1 chemical structural congruent to $\mathcal{C}(B)$. The reduction involving $a[\mathbf{chn}]$, as in case **(O-0)**, will produce another context $\mathcal{D}$ and we can use induction hypothesis.

The other cases are similar to one of the three above cases. □

The first part of the operational correspondence, i.e. the encoding follows every reduction of the source process, is based on the following explicit substitution lemma. We user a meta-notation $A\{b := c\}$ for the solution $(vb)(b[\mathbf{fwd}\ c] \mid A)$.

Lemma 39 (*Explicit substitution*). *For any π_{af}-process p, we have $\mathbb{CP}\mathbb{I}_c \vdash [\![p]\!]\{b := c\} \simeq [\![p\{c/b\}]\!]$.*

Proof. By using the results in Lemma 38. See Appendix E. □

For the second part, we first show that any derivation A of $\langle\!\langle p \rangle\!\rangle$ is equivalent to some $\mathbb{PI}_c$-solution $A^\diamond$ (Lemma 40). We then show that we can find for any $\mathbb{PI}_c$-solution A a corresponding π_{af}-process A^{-1} (Lemma 42). To this end, we obtain a *decoding* function $((\cdot)^\diamond)^{-1}$ that turns every $\mathbb{CP}\mathbb{I}_c$-solution A to a π_{af}-process $(A^\diamond)^{-1}$ whose encoding is equivalent to A. Using this function, we can show that if $\langle\!\langle p \rangle\!\rangle \hookrightarrow^* A$, then $(A^\diamond)^{-1}$ is the π_{af}-process that can be derived from p.

Lemma 40 (*Flattening*). *For any solution A with $\langle\!\langle p \rangle\!\rangle \hookrightarrow^* A$ for some π_{af}-process p, we can find a $\mathbb{PI}_c$-solution $A^\diamond$ s.t. $\mathbb{CP}\mathbb{I}_c \vdash A \simeq A^\diamond$.*

Proof. See Appendix F. □

Definition 41 (*Reverse process*). For any $A \in \mathbb{PI}$, we define its *reverse process* A^{-1} to be the π_{af}-process obtained using the following inductive definition:
(1) *If* $A \rightleftharpoons \mathbf{0}$, *then* $A^{-1} =_{\mathrm{def}} \mathbf{0}$;
(2) *If* $A \rightleftharpoons (vY)(\prod_i a_i.\mathtt{put}\langle comm.\mathtt{put}\langle \mathbf{fwd}\ b_i\rangle\rangle \mid \prod_j (vb)(b[\mathbf{0}] \mid a_j.\mathtt{put}\langle comm.\mathtt{get}.\uparrow.\uparrow.b\langle\uparrow\langle A_j\rangle\rangle\rangle) \mid \prod_{a\in Y} a[\mathbf{chn}])$, then $A^{-1} =_{\mathrm{def}} (vY)(\prod_i a_i\langle b_i\rangle \mid \prod_j a_j(b).(A_j^{-1}))$.

We extend the definition to $A \in \mathbb{PI}_c$ s.t. $A^{-1} =_{\mathrm{def}} B^{-1}$ where $B \in \mathbb{PI}$ is the solution obtained by removing all the free queues in A.

Lemma 42 (*Reverse*). *We have*
(1) *For any $A \in \mathbb{PI}$, $[\![A^{-1}]\!] \rightleftharpoons A$;*
(2) *For any $A \in \mathbb{PI}_c$, $\langle\!\langle A^{-1} \rangle\!\rangle \rightleftharpoons A$;*
(3) *For any π_{af}-process p, $\langle\!\langle p \rangle\!\rangle^{-1} \equiv_\pi [\![p]\!]^{-1} \equiv_\pi p$.*

Proof. The first is by induction on the size of A. The second is by the first result. The third one is by induction on the structure of p. □

Theorem 43 (*Operational correspondence*). *For any π_{af}-process p we have*
(1) *If $p \longrightarrow_\pi q$, then we can find R with $\langle\!\langle p \rangle\!\rangle \longrightarrow^* R$ and $\mathbb{CP}\mathbb{I}_c \vdash \langle\!\langle q \rangle\!\rangle_{fn(p)} \simeq R$;*
(2) *If $\langle\!\langle p \rangle\!\rangle \longrightarrow^* R$, then we can find q with $p \longrightarrow_\pi^* q$ and $\mathbb{CP}\mathbb{I}_c \vdash \langle\!\langle q \rangle\!\rangle_{fn(p)} \simeq R$.*

Proof. See Appendix G. □

9. Conclusion

In this paper, we present a small variant of the ambient calculus called the wagon calculus. It is obtained by restricting the syntax of Safe Ambients, a popular variant of the original ambient calculus with better behavior theory. By the restriction, we arrive at a sub-calculus of SA that is free of grave interference syntactically. We then focus on the expressiveness of wagon and show that wagon is also able to encode name-passing. Based on properties ensured by a novel spatial logic, we prove the correctness of the encoding through an operational correspondence result.

The logic is build upon a chemical semantics of the wagon calculus. Without this, it would be difficult to tell the one-to-one relationship between an input process and the corresponding private forwarder where it will return to afterwards. Our experience shows 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 enables a finer granularity for scope intrusion and extrusion, which often helps the normalization of process terms.

Upon the introduction of the ambient calculus, similar objective move primitives and self-dissolving have already been discussed. They are not adopted in the original version for that they can be used for example to insert processes to other ambients and trap them. In the wagon calculus, however, these primitives can only insert processes to locations, which are by nature immobile. By separating immobile locations from mobile packets, objective move and self-dissolving form another primitive set for modelling mobility, upon which the wagon calculus is based.

We have briefly stated that wagon is a sub-calculus of SA. We discuss a bit here about the converse. Since grave interferences in untyped SA is not possible to simulate in wagon, we believe that it is impossible to fully embed SA in wagon. For the subset of well-typed SA, the problem is difficult, but not at all impossible. In typed SA, unleashing a sub-ambient on the move could be simulated in wagon by a dissolving action with the parent ambient wrapped together with the sub-ambient in the payload. Collecting a sub-ambient without opening could be simulated by a `get/put` interaction with the sub-ambient wrapped in an extra layer. However, there are many subtle problems left for a reasonable encoding. We leave this for future investigation.

Our encoding of the π-calculus adopts the same mechanism of Zimmer's, only a few simplifications like the unified treatment of channel names and variable names. Actually, our encoding is also built upon the same intermediate language called the *π-calculus with explicit substitutions and channels* [27]. However, Zimmer's operational correspondence result only shows that principle reductions of the encoding can always be matched by the original process. He conjectured that the same is true for other non-principle reductions. Thanks to the simplicity of wagon semantics, we are able to formalize the properties of the encoding and prove a more complete operational correspondence result.

Moreover, we conjecture that the prove techniques used here could also be adapted to the proof of Zimmer's encoding, since our logic could be quite easily generalized to other name-based process calculi, and a chemical semantics for (typed) SA could be developed similarly. We believe that it is only a matter of the size of the final closure formula corresponding to $\mathbb{CPI}_c$, due to the size of Zimmer's encoding and the richer interaction patterns of SA.

To minimize the complexity of the formulas, we choose the finite asynchronous π_{af}-calculus as our source language. There would be little difficulties in extending the result in this paper to the synchronous π-calculus that Zimmer used in his encoding. For the output construct *with* continuation, $a\langle b\rangle.p$, the following translation suffices:

$$[\![a\langle b\rangle.p]\!] =_{\mathrm{def}} a.\mathtt{put}\langle comm.\mathtt{put}\langle \mathbf{fwd}\ b \mid \uparrow\langle [\![p]\!]\rangle\rangle\rangle.$$

Similarly, we could add replication:

$$[\![!p]\!] =_{\mathrm{def}} ![\![p]\!].$$

Although these require an alternative wagon syntax using general replications and an even larger definition for $\mathbb{CPI}_c$, we do not see any major obstacles in obtaining a result similar to Theorem 43.

It is yet unknown whether one could encode π-calculus with mixed-choice in pure ambients, as the latter is strictly more expressive [22] and is able to solve the symmetric leader election (SLE) problem. However, recent results reveal that pure ambients could also solve SLE [23]. This implies the extra expressiveness of pure ambients over asynchronous π-calculus. It would be very interesting to further investigate this problem.

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. Using the Kleene star to count parallel components in spatial logic has also been studied in the spatial logic of [11]. Constructs of the fresh set binder $(\gamma x)(\cdot)$ and the product $\prod_{a\in u}(\cdot)$ involving fresh sets are new, up to our knowledge. It would also be very interesting to investigate the relation of our spatial logic with works on behavior types [24,1] and spatial logic [4,8,9], especially the relation of our fresh set binder with other hidden name quantifiers.

Acknowledgments

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.

$$\frac{-}{\emptyset \vdash \mathbf{0}} \qquad \frac{\emptyset \vdash P}{\emptyset \vdash M\langle P\rangle,\ \emptyset \vdash !M\langle P\rangle} \qquad \frac{X \vdash P}{X\backslash\{a\} \vdash (\nu a)P}$$

$$\frac{X_i \vdash P_i \quad (i = 1, 2), \quad X_1 \cap X_2 = \emptyset}{X_1 \uplus X_2 \vdash P_1 \mid P_2} \qquad \frac{X \vdash P}{\{a\} \vdash a[P]}$$

Fig. A.1. The unique sibling name discipline.

Appendix A. The unique sibling name discipline

In ambient calculi, parallel ambients can have the same name to model replication of resources like network printers. However, sometimes we need to avoid the interferences caused by the duplicated sibling names, like the channel locations in our encoding of π-calculus. Also, the type discipline of unique sibling name enables us to break down locations into smaller components with a chemical semantics. This appendix presents such a type discipline.

Judgements in discipline are of the form $X \vdash P$. Recall that X is a set of names. $X \vdash P$ assures that (1) P has top-level location names in X, (2) all sibling location names in P are unique, and (3) every packet in P contains no free location in the payload. The typing rules are reported in Fig. A.1. Their meanings should be self-evident.

The subject reduction result is obtained by observing that no free locations will be released during the dissolving of any packets.

Proposition 44 (*Subject reduction*). *If $X \vdash P$ and $P \longrightarrow Q$, then $X \vdash Q$.*

Proof. The proof is by induction on the derivation of $P \longrightarrow Q$.

- Case $P = a.M\langle R_1\rangle \mid a[R_2]$ and $Q = a[M\langle R_1\rangle \mid R_2]$: From $X \vdash P$ we know that $X = \{a\}$, $\emptyset \vdash R_1$, $Y \vdash R_2$. So we have $Y \vdash M\langle R_1\rangle \mid R_2$ and $X \vdash Q$.
- Case $P = a[\uparrow.M\langle R_1\rangle \mid R_2]$ and $Q = M\langle R_1\rangle \mid a[R_2]$: From $X \vdash P$ we know that $X = \{a\}$, $\emptyset \vdash R_1$, $Y \vdash R_2$. So we have $X \vdash a[R_2]$ and $X \vdash Q$.
- Case $P = \mathtt{get}.M\langle R_1\rangle \mid \mathtt{put}\langle R_2\rangle$ and Q=$M\langle R_1 \mid R_2\rangle$: From $X \vdash P$ we know that $X = \emptyset$, $\emptyset \vdash R_1$, $\emptyset \vdash R_2$. So we have $\emptyset \vdash R_1 \mid R_2$ and $X \vdash Q$.
- The other rules and the subject congruence results are similar. □

Specifically, we can easily check that the encoding obeys the unique sibling name discipline.

Lemma 45. *For any π_{af}-process p, we have $\emptyset \vdash [\![p]\!]$ and $fn(p) \vdash \langle\!\langle p\rangle\!\rangle$.*

Proof. Easy by induction on the structure of p. □

Appendix B. Laws of the spatial logic

This section presents a few results of our spatial logic, especially laws that state the equivalence or inclusion of different formulas.

By definition, we can only compare closed formulas, i.e. formulas having no free variables. However, we extend such relations naturally to open formulas: a relation between open formulas is true if and only if the relation is true for any valid instantiations of the free variables in these formulas. For example, we have $\mathbf{0} \subseteq \mathbb{A}^*$, even if $\mathbb{A}^*$ is open.

The following lemma gives some basic laws, which enable syntactic transforming of formulas. They are quite similar to the chemical structural congruence of solutions.

Lemma 46 (*Basic Laws—1*). *The followings hold*:

(F-Par-Zero) $\mathbb{A} \mid \mathbf{0} = \mathbb{A}$

(F-Par-Sym) $\mathbb{A} \mid \mathbb{B} = \mathbb{B} \mid \mathbb{A}$

(F-Par-Assoc) $(\mathbb{A} \mid \mathbb{B}) \mid \mathbb{C} = \mathbb{A} \mid (\mathbb{B} \mid \mathbb{C})$
(F-Res-Par) $\mathbb{A} \mid (va)\mathbb{B} = (va)(\mathbb{A} \mid \mathbb{B})$, *if* $a \notin fn(\mathbb{A})$
(F-FS-Par) $\mathbb{A} \mid (\gamma x)\mathbb{B} = (\gamma x)(\mathbb{A} \mid \mathbb{B})$, *if* $x \notin fv(\mathbb{A})$
(F-Res-Loc) $a[(vb)\mathbb{A}] = (vb)a[\mathbb{A}]$, *if* $a \neq b$
(F-FS-Loc) $a[(\gamma x)\mathbb{A}] = (\gamma x)a[\mathbb{A}]$
(F-Rep) $!\mathbb{M}\langle\mathbb{A}\rangle = !\mathbb{M}\langle\mathbb{A}\rangle \mid \mathbb{M}\langle\mathbb{A}\rangle$
(F-Dis) $\mathtt{dis}\langle\mathbb{A}\rangle = \mathbb{A}$
(F-Loc-Par) $a[\mathbb{A} \mid \mathbb{B}] = \mathbb{A}@a \mid a[\mathbb{B}]$
(F-Loc-Nil) $a[\mathbf{0}] = a$
(F-At-Par) $(\mathbb{A} \mid \mathbb{B})@a = \mathbb{A}@a \mid \mathbb{B}@a$
(F-Res-At) $(va)\mathbb{A}@u = (va)(\mathbb{A}@u)$, *if* $a \notin fn(u)$
(F-FS-At) $(\gamma x)\mathbb{A}@u = (\gamma x)(\mathbb{A}@u)$, *if* $x \notin fv(u)$

Proof. Easy by checking their definitions. We give an example for the proof of $\mathbb{A} \mid (\gamma x)\mathbb{B} = (\gamma x)(\mathbb{A} \mid \mathbb{B})$ if $x \notin fv(\mathbb{A})$.

(1) For any $A \in \mathbb{A} \mid (\gamma x)\mathbb{B}$, we know that there is A_1, A_2 with $A_1 \in \mathbb{A}$, $A_2 \in (\gamma x)\mathbb{B}$, and $A \rightleftharpoons A_1 \mid A_2$. From $A_2 \in (\gamma x)\mathbb{B}$ we know that there is some fresh set X with $X \cap fn(\mathbb{B}) = \emptyset$, $A_2 \rightleftharpoons (vX)A_3$ and $A_3 \in \mathbb{B}\{X/x\}$. So we have $A_1 \mid A_3 \in \mathbb{A} \mid \mathbb{B}\{X/x\}$. From the condition that $x \notin fv(\mathbb{A})$, we know that $\mathbb{A}\{X/x\} = \mathbb{A}$, so we have $A_1 \mid A_3 \in \mathbb{A}\{X/x\} \mid \mathbb{B}\{X/x\} = (\mathbb{A} \mid \mathbb{B})\{X/x\}$. Since X is chosen fresh, we may assume that X is disjoint with $fn(\mathbb{A})$ and $fn(A_1)$. So by $X \cap (fn(\mathbb{A}) \cup fn(\mathbb{B})) = \emptyset$, we have $(vX)(A_1 \mid A_3) \in (\gamma x)(\mathbb{A} \mid \mathbb{B})$. So we can have $A \rightleftharpoons A_1 \mid A_2 \rightleftharpoons A_1 \mid (vX)A_3 \rightleftharpoons (vX)(A_1 \mid A_3)$, we know that $A \in (\gamma x)(\mathbb{A} \mid \mathbb{B})$.

(2) For any $A \in (\gamma x)(\mathbb{A} \mid \mathbb{B})$, we know that there is X with $X \cap fn(\mathbb{A} \mid \mathbb{B}) = \emptyset$, $A \rightleftharpoons (vX)A_1$, and $A_1 \in (\mathbb{A} \mid \mathbb{B})\{X/x\}$. By $x \notin fv(\mathbb{A})$ we know that $A_1 \in \mathbb{A} \mid \mathbb{B}\{X/x\}$. So there is A_2, A_3 with $A_2 \in \mathbb{A}$, $A_3 \in \mathbb{B}\{X/x\}$, and $A_1 \rightleftharpoons A_2 \mid A_3$. Then from $X \cap fn(\mathbb{A} \mid \mathbb{B}) = \emptyset$, we know that $X \cap fn(\mathbb{B}) = \emptyset$, thus $(vX)A_3 \in (\gamma x)\mathbb{B}$. So we have $A_2 \mid (vX)A_3 \in \mathbb{A} \mid (\gamma x)\mathbb{B}$. From $X \cap fn(\mathbb{A} \mid \mathbb{B}) = \emptyset$ and $A_2 \in \mathbb{A}$ we also have $X \cap fn(A_2) = \emptyset$. So $A \rightleftharpoons (vX)A_1 \rightleftharpoons (vX)(A_2 \mid A_3) \rightleftharpoons A_2 \mid (vX)A_3$, which implies $A \in \mathbb{A} \mid (\gamma x)\mathbb{B}$. □

Definition 47 (*Product variable*). We define the set of *product variables* of a formula, $pv(\mathbb{A})$, as those variables that appear in the subscripts of products. It is define to be non-trivial only on the product construct ($pv(\prod_{a\in u}\mathbb{A}) =_{\text{def}} fv(u) \cup pv(\mathbb{A})$), and is homomorphic on all the other constructs.

We use $\mathcal{E}$ to denote a formula context with a single hole. We write $\mathcal{E}(\mathbb{A})$ for the resulting formula by filling the hole in $\mathcal{E}$ with $\mathbb{A}$.

Lemma 48 (*Monotonicity*). *The followings hold*:
(1) *For any formula context* $\mathcal{E}$, $\mathbb{A} \subseteq \mathbb{B}$ implies $\mathcal{E}(\mathbb{A}) \subseteq \mathcal{E}(\mathbb{B})$;
(2) *If* $x \notin pv(\mathbb{A})$ *and* $Y \cap fn(\mathbb{A}) = \emptyset$, *then* $X \subseteq Y$ *implies* $\mathbb{A}\{X/x\} \subseteq \mathbb{A}\{Y/x\}$.

Proof. (1) By induction on the structure of $\mathcal{E}$. (2) By induction on the structure of $\mathbb{A}$. □

Remark 49. The second monotonic property fails if the variable is inside the product subscript, for example, $b[\mathbf{0}] \in (\prod_{a\in x} a[\mathbf{0}])\{b/x\}$, but $b[\mathbf{0}] \notin (\prod_{a\in x} a[\mathbf{0}])\{\{b,\ c\}/x\}$.

Before ending this section, we give a few more laws. They will be used (sometimes implicitly) in our latter proofs.

Lemma 50 (*Basic laws—2*). *We have the followings hold*:
(F-Empty-Header) $\emptyset.\mathbb{M}\langle\mathbb{A}\rangle = \mathbf{0}$
(F-Prod-Nil) $\prod_{a\in u}\mathbf{0} = \mathbf{0}$
(F-Prod-Empty) $\prod_{a\in\emptyset}\mathbb{A} = \mathbf{0}$
(F-Prod-Par) $\prod_{a\in u}(\mathbb{A} \mid \mathbb{B}) = \prod_{a\in u}\mathbb{A} \mid \prod_{a\in u}\mathbb{B}$

(F-Star-Single) $\mathbb{A} \subseteq \mathbb{A}^*$
(F-Star-Plus) $\mathbb{A}^* \mid \mathbb{A} \subseteq \mathbb{A}^*$
(F-Star-Nil) $\mathbf{0} \subseteq \mathbb{A}^*$
(F-Star-Par-1) $u.\mathbb{M}\langle\mathbb{A}\rangle^* \mid v.\mathbb{M}\langle\mathbb{A}\rangle^* = u \cup v.\mathbb{M}\langle\mathbb{A}\rangle^*$
(F-Star-Par-2) $(\mathbb{A} \mid \mathbb{B})^* \subseteq \mathbb{A}^* \mid \mathbb{B}^*$
(F-FS-Absorb) $(\gamma x)(va)\mathbb{A}\{x \cup \{a\}/x\} \subseteq (\gamma x)\mathbb{A}$, *if* $a \notin fn(\mathbb{A})$
(F-FS-Prod) $(\gamma x)(\prod_{a\in x} \mathbb{A} \mid (va)\mathbb{A}) \subseteq (\gamma x)\prod_{a\in x} \mathbb{A}$, *if* $x \notin pv(\mathbb{A})$
(F-FS-Prod-Star) $(\gamma x)\prod_{a\in x} \mathbb{A} = (va)\mathbb{A}^*$, *if* $x \notin fv(\mathbb{A})$
(F-FS-Res) $(\gamma x)(va)\mathbb{A} = (va)(\gamma x)\mathbb{A}$
(F-FS-Empty) $\mathbb{A}\{\emptyset/x\} \subseteq (\gamma x)\mathbb{A}$
(F-Nil-Star) $\mathbf{0}^* = \mathbf{0}$

Proof. We prove these results one by one.

(F-Empty-Header) $\emptyset.\mathbb{M}\langle\mathbb{A}\rangle = \mathbf{0}$: By definition, $\emptyset.\mathbb{M} = \emptyset$ and $\emptyset\langle\mathbb{A}\rangle = \mathbf{0}$. So we know $\emptyset.\mathbb{M}\langle\mathbb{A}\rangle = \mathbf{0}$.

(F-Prod-Nil) $\prod_{a\in u} \mathbf{0} = \mathbf{0}$: For any name b, we know that $\mathbf{0}\{b/a\} = \mathbf{0}$. So for any set $u = \{a_1, \ldots, a_k\}$ s.t. $|u| = k$, any element belonging to the left side is of the form $A_1 \mid \cdots \mid A_k$ where $A_i \in \mathbf{0}$. That is, any element of the left side is chemical structural congruent to $\mathbf{0}$, which is just the definition of the formula $\mathbf{0}$.

(F-Prod-Empty) $\prod_{a\in\emptyset} \mathbb{A} = \mathbf{0}$: By the definition, $A \in \prod_{a\in\emptyset} \mathbb{A} \iff A \rightleftharpoons \prod_{i=0} A_i \iff A \rightleftharpoons \mathbf{0} \iff A \in \mathbf{0}$.

(F-Prod-Par) $\prod_{a\in u} (\mathbb{A} \mid \mathbb{B}) = \prod_{a\in u} \mathbb{A} \mid \prod_{a\in u} \mathbb{B}$: Suppose $u = \{a_1, \ldots, a_k\}$. We have $A \in \prod_{a\in u} (\mathbb{A} \mid \mathbb{B}) \Leftarrow\Rightarrow \exists A_1 \ldots A_k \,.\, (A \rightleftharpoons A_1 \mid \ldots \mid A_k \wedge A_i \in (\mathbb{A} \mid \mathbb{B})\{a_i/a\})$. We know that $A_i \in (\mathbb{A} \mid \mathbb{B})\{a_i/a\} \iff \exists B_i \in \mathbb{A}\{a_i/a\} \,.\, \exists B_i' \in \mathbb{B}\{a_i/a\} \,.\, A_i \rightleftharpoons B_i \mid B_i'$. Then we have $\prod_i B_i \in \prod_{a\in u} \mathbb{A}$ and $\prod_i B_i' \in \prod_{a\in u} \mathbb{B}$. So we know $A \rightleftharpoons \prod_i B_i \mid \prod_i B_i' \in \prod_{a\in u} \mathbb{A} \mid \prod_{a\in u} \mathbb{B}$. The other direction is similar.

(F-Star-Single) $\mathbb{A} \subseteq \mathbb{A}^*$: Easy.

(F-Star-Plus) $\mathbb{A}^* \mid \mathbb{A} \subseteq \mathbb{A}^*$: Easy.

(F-Star-Nil) $\mathbf{0} \subseteq \mathbb{A}^*$: Easy.

(F-Star-Par-1) $u.\mathbb{M}\langle\mathbb{A}\rangle^* \mid v.\mathbb{M}\langle\mathbb{A}\rangle^* = u \cup v.\mathbb{M}\langle\mathbb{A}\rangle^*$: (1) Suppose $A \in u.\mathbb{M}\langle\mathbb{A}\rangle^* \mid u.\mathbb{M}\langle\mathbb{A}\rangle^*$, we know $\exists B_1 \in u.\mathbb{M}\langle\mathbb{A}\rangle^* \,.\, \exists B_2 \in v.\mathbb{M}\langle\mathbb{A}\rangle^* \,.\, A \rightleftharpoons B_1 \mid B_2$. We know that B_1 is the parallel composition of several $u.\mathbb{M}\langle\mathbb{A}\rangle$-solutions, which are also $u \cup v.\mathbb{M}\langle\mathbb{A}\rangle$-solutions, and similar for B_2. So we conclude that A is the parallel composition of several $u \cup v.\mathbb{M}\langle\mathbb{A}\rangle$-solutions. So $A \in u \cup v.\mathbb{M}\langle\mathbb{A}\rangle^*$. (2) Suppose $A \in u \cup v.\mathbb{M}\langle\mathbb{A}\rangle^*$, we know that A is the parallel composition of several $u \cup v.\mathbb{M}\langle\mathbb{A}\rangle$-solutions. For any of these $u \cup v.\mathbb{M}\langle\mathbb{A}\rangle$-solutions, either it is a $u.\mathbb{M}\langle\mathbb{A}\rangle$-solution, or it is not a $u.\mathbb{M}\langle\mathbb{A}\rangle$-solution. We arrange all the $u.\mathbb{M}\langle\mathbb{A}\rangle$-solutions together and call their parallel composition B_1. We call the rest B_2. We know that all those in B_2 are $v.\mathbb{M}\langle\mathbb{A}\rangle$-solutions. So we have $B_1 \in u.\mathbb{M}\langle\mathbb{A}\rangle^* \wedge B_2 \in v.\mathbb{M}\langle\mathbb{A}\rangle^* \wedge A \rightleftharpoons B_1 \mid B_2$. So $A \in u.\mathbb{M}\langle\mathbb{A}\rangle^* \mid v.\mathbb{M}\langle\mathbb{A}\rangle^*$.

(F-Star-Par-2) $(\mathbb{A} \mid \mathbb{B})^* \subseteq \mathbb{A}^* \mid \mathbb{B}^*$: Easy.

(F-FS-Absorb) $(\gamma x)(va)\mathbb{A}\{x \cup \{a\}/x\} \subseteq (\gamma x)\mathbb{A}$ if $a \notin fn(\mathbb{A})$: Suppose $A \in (\gamma x)(va)\mathbb{A}\{x \cup \{a\}/x\}$. There is a set X s.t. $X \cap fn((va)\mathbb{A}\{x \cup \{a\}/x\}) = \emptyset \wedge A \rightleftharpoons (vX)B$ and $B \in ((va)\mathbb{A}\{x \cup \{a\}/x\})\{X/x\} = (va)(\mathbb{A}\{X \cup \{a\}/x\})$. We know there is B' s.t. $B = (va)B'$ and $B' \in \mathbb{A}\{X \cup \{a\}/x\}$. So we have $A \rightleftharpoons (vX \cup \{a\})B'$. From $a \notin fn(\mathbb{A})$ and $X \cap fn((va)\mathbb{A}\{x \cup \{a\}/x\}) = \emptyset$, we know that $(X \cup \{a\}) \cap fn(\mathbb{A}) = \emptyset$. So by definition, $A \in (\gamma x)\mathbb{A}$.

(F-FS-Prod) $(\gamma x)(\prod_{a\in x} \mathbb{A} \mid (va)\mathbb{A}) \subseteq (\gamma x)\prod_{a\in x} \mathbb{A}$, if $x \notin pv(\mathbb{A})$: We know that the left side equals to $(\gamma x)(va)\prod_{a\in x\cup\{a\}} \mathbb{A}$. By the Monotonicity Lemma 48, we know that the latter is a subset of $(\gamma x)(va)\prod_{a\in x\cup\{a\}} \mathbb{A}\{x \cup \{a\}/x\}$. By applying the previous result, we know that the latter is a subset of $(\gamma x)\prod_{a\in x} \mathbb{A}$.

(F-FS-Prod-Star) $(\gamma x)\prod_{a\in x} \mathbb{A} = (va)\mathbb{A}^*$, if $x \notin fv(\mathbb{A})$: (1) Suppose $A \in (\gamma x)\prod_{a\in x} \mathbb{A}$. By definition, we have $\exists X \,.\, (X \cap fn(\mathbb{A}) = \emptyset \wedge A \rightleftharpoons (vX)B \wedge B \in \prod_{a\in X} \mathbb{A}\{X/x\} = \prod_{a\in X} \mathbb{A})$. Moreover, we know that B is made by the parallel composition of k components (suppose $|X| = k$). From $X \cap fn(\mathbb{A}) = \emptyset$, we know that we can shrink the scope of each restricted name in X to the component that contains it. That is, we have $A \rightleftharpoons (vX)B \rightleftharpoons \prod_i (va_i)B_i$, where $B_i \in \mathbb{A}\{a_i/a\}$. So we know $A \in (va)\mathbb{A}^*$. (2) Suppose $A \in (va)\mathbb{A}^*$, A must be the composition of k components of the form $(va)B_i$ with $B_i \in \mathbb{A}$. We can extend those k private restrictions to the out-most and have $A \rightleftharpoons (va_1 \ldots a_k)\prod_i B_i\{a_i/a\}$. This means $A \in (\gamma x)\prod_{a\in x} \mathbb{A}$.

(F-FS-Res) $(\gamma x)(va)\mathbb{A} = (va)(\gamma x)\mathbb{A}$: Easy.

(F-FS-Empty) $\mathbb{A}\{\emptyset/x\} \subseteq (\gamma x)\mathbb{A}$: Easy.

(F-Nil-Star) $\mathbf{0}^* = \mathbf{0}$: Easy. □

Appendix C. Proof of Lemma 23

Lemma 51. *The followings hold*:
(1) $\mathbb{O}_1(X) \subseteq \mathbb{PI}(X)$
(2) $\mathbb{I}_1(X) \subseteq \mathbb{PI}(X)$

Proof. By the law $\mathbb{A}\{\emptyset/x\} \subseteq (\gamma x)\mathbb{A}$, we have $\mathbb{O}_1(X)^* \mid \mathbb{I}_1(X)^* \subseteq (\gamma y)(\mathbb{O}_1(Xy)^* \mid \mathbb{I}_1(Xy)^* \mid \prod_{a\in y} a[\mathbb{C}]) = \mathbb{PI}(X)$. By the law $\mathbf{0} \subseteq \mathbb{A}^*$ and the Monotonicity Lemma 48.(1), we know $\mathbb{O}_1(X)^* \subseteq \mathbb{O}_1(X)^* \mid \mathbb{I}_1(X)^*$. By the law $\mathbb{A} \subseteq \mathbb{A}^*$, we know $\mathbb{O}_1(X) \subseteq \mathbb{O}_1(X)^*$. So we have result (1). The proof of result (2) is similar. □

Lemma 23. *For any π-process p, $[\![p]\!] \in \mathbb{PI}(fn(p))$ and $\langle\!\langle p \rangle\!\rangle \in \mathbb{PI}_c(fn(p))$.*

Proof. The first result is by induction on the structure of p. The second is direct by definition and the first result.
$p = \mathbf{0}$ We have $[\![p]\!] =_{\text{def}} \mathbf{0} \in \mathbb{PI}(\emptyset)$.
$p = (va)q$ We have by induction hypothesis that $[\![q]\!] \in \mathbb{PI}(fn(q))$. Then $[\![(va)q]\!] =_{\text{def}} (va)(a[!\texttt{get}\langle\rangle \mid comm[]] \mid [\![q]\!]) \in (va)(\mathbb{C}(\{a\}) \mid (\gamma y)(\mathbb{O}_1(fn(q)\cup y)^* \mid \mathbb{I}_1(fn(q)\cup y)^* \mid \mathbb{C}(y)))$. Since $a \notin fn(p)$ and $fn(q) \subseteq fn(p) \uplus \{a\}$, the latter is a subset of $(va)(\gamma y)(\mathbb{O}_1(fn(p) \cup (\{a\} \cup y))^* \mid \mathbb{I}_1(fn(p) \cup (\{a\} \cup y))^* \mid \mathbb{C}(y \cup a))$, which is itself a subset of $\mathbb{PI}(fn(p))$, by law **(F-FS-Absorb)**.
$p = p_1 \mid p_2$ We have by induction hypothesis that $[\![p_i]\!] \in \mathbb{PI}(fn(p_i))$ $(i = 1, 2)$. So we have $[\![p_i]\!] \rightleftharpoons (vX_i)(\mathbb{C}(X_i) \mid O_i \mid I_i)$, with $O_i \in \mathbb{O}_1(X_i \cup fn(p_i))^*$ and $I_i \in \mathbb{I}_1(X_i \cup fn(q_i))^*$. Assume $X_1 \cap X_2 = \emptyset$, we have $[\![p]\!] = (vX_1)(\mathbb{C}(X_1) \mid O_1 \mid I_1) \mid (vX_2)(\mathbb{C}(X_2) \mid O_2 \mid I_2) \rightleftharpoons (vX_1X_2)(\mathbb{C}(X_1 \cup X_2) \mid O_1 \mid O_2 \mid I_1 \mid I_2)$. It is easy to show that $O_1 \mid O_2 \in \mathbb{O}_1(X_1X_2 \cup fn(p))^*$ and $I_1 \mid I_2 \in \mathbb{I}_1(X_1X_2 \cup fn(p))^*$. So we have $[\![p]\!] \in \mathbb{PI}(fn(p))$.
$p = a\langle b\rangle$ We have $[\![p]\!] = a.\texttt{put}\langle comm.\texttt{put}\langle !\texttt{get}.\uparrow.b.\texttt{put}\langle\rangle\rangle\rangle \in \mathbb{O}_1(\{a, b\}) \subseteq \mathbb{PI}(\{a, b\})$, by Lemma 51.(1).
$p = a(b).q$ We have by induction hypothesis that $[\![q]\!] \in \mathbb{PI}(fn(q))$. So we know that $[\![p]\!] =_{\text{def}} (vb)(b[] \mid a.\texttt{put}\langle comm.\texttt{get}.\uparrow.\uparrow.b\langle\uparrow\langle[\![q]\!]\rangle\rangle\rangle) \in \mathbb{I}_1(\{a\} \cup (fn(q)\backslash\{b\})) = \mathbb{I}_1(fn(p)) \subseteq \mathbb{PI}(fn(p))$, by Lemma 51.(2). □

Appendix D. Proof of Lemma 27

Lemma 52. *For any X and any $A \in \mathbb{CPI}_c(X)$, if $A \hookrightarrow B$, then $B \in \mathbb{CPI}_c(X)$.*

Proof. The proof is by analyzing the structure of A.

Since $A \in \mathbb{CPI}_c(X)$, we know that there are Y, Z, C, C', V, A_1 s.t. $A \rightleftharpoons (vY)(vZ)(C \mid V \mid A_1) \mid C'$ with $C \in \mathbb{C}(Y)$, $C' \in \mathbb{C}(X)$, $V \in \mathbb{V}(XY, Z)$, and $A_1 \in \mathbb{D}(XY, Z)$. Since C, C', and V have no reduction alone, we know that every reduction of A must involve (some part of) A_1. Since $A_1 \in \mathbb{D}(XY, Z)$, this means that the reduction must involve at least one solution belonging to the following sets: $\mathbb{I}_0, \ldots, \mathbb{I}_8, \mathbb{O}_0, \ldots, \mathbb{O}_4$. We prove by enumerating all these 14 cases. We only show two cases here. The other cases are similar. The general packet state transitions with these reductions are depicted in Fig. 8.

(1) The reduction involves an $\mathbb{I}_0$-solution: Then we must have $A_1 \rightleftharpoons A_2 \mid A_3$ with $A_2 \in \mathbb{D}(XY, Z)$ and $A_3 \in \mathbb{I}_0(XYZ, Z)$ contributing to the reduction. The only possible reduction that A_3 can contribute is an out reduction by itself and becomes $A_3' \in \mathbb{I}_1(XYZ)$, with $B \rightleftharpoons (vY)(vZ)(C \mid V \mid A_2 \mid A_3') \mid C'$. So we know $A_2 \mid A_3' \in \mathbb{D}(XY, Z)$ and we conclude with $B \rightleftharpoons (vY)(vZ)(C \mid V \mid A_2 \mid A_3') \mid C' \in \mathbb{CPI}_c(X)$.
(2) The reduction involves an $\mathbb{I}_7$-solution: Then we must have $A_1 \rightleftharpoons A_2 \mid A_3$ with $A_2 \in \mathbb{D}(XY, Z)$ and the involved solution $A_3 \rightleftharpoons (vb)(b \mid b\langle F \mid G\rangle) \in \mathbb{I}_7(XYZ)$, for some $F \in \mathbb{F}(XYZ)$ and $G \in \uparrow\langle\mathbb{PI}(XYZ \cdot b)\rangle$. The only possible reduction that A_3 can contribute is an reduction by itself and becomes $A_3' \rightleftharpoons (vb)(b[F] \mid G@b)$. We know that $b[F] \in \mathbb{V}(XYZ, b)$ and thus $V \mid b[F] \in \mathbb{V}(XY, Z \cdot b)$. We also know that $G@b \in \mathbb{I}_8(XYZ \cdot b, Z \cdot b)$ and thus $A_2 \mid G@b \in \mathbb{D}(XY, Z \cdot b)$. So we conclude this case with $B \rightleftharpoons (vY)(vZ)(C \mid V \mid A_2 \mid A_3') \mid C' \rightleftharpoons (vY)(vZ \cdot b)(C \mid (V \mid b[F]) \mid (A_2 \mid G@b)) \mid C' \in \mathbb{CPI}_c(X)$, the final step by choosing the fresh z in the definition of $\mathbb{CPI}(X)$ to be $Z \cdot b$. □

Lemma 27. (1) *For any X, $\mathbb{CPI}_c(X)$ is a closure.* (2) *$\mathbb{CPI}_c$ is a closure.*

Proof. (1) Easy by Lemma 52.

(2) By definition, the union of two closures is also a closure. Since any $\mathbb{CPI}_c(X)$ is a closure, so their union $\mathbb{CPI}_c$ is also a closure. □

Appendix E. Proof of the explicit substitution Lemma 39

Lemma 53. *If* $A\{b := c\} \in \mathbb{CPI}^\circ$, *then* $A\{b := c\} \Downarrow_{comm@a} \Longrightarrow A\{c/b\} \Downarrow_{comm@a}$.

Proof. We consider all the top-level packets in the $\mathbb{O}_1$-solutions and $\mathbb{I}_1$-solutions of A: they can only be of the form $a.\mathtt{put}\langle C\rangle$ or $(\nu b')(b' \mid a.\mathtt{put}\langle C\rangle)$, with either $a = b$ or $a \neq b$. For all those packets having $a = b$, we change a into c, and denote the resulting process $A^=$. From laws **(I/O-1), (I/O-2.2)** and **(I/O-0)**, we know that $\mathbb{CPI}_c \vdash A\{b := c\} \simeq A^=\{b := c\}$. So we only need to prove $A^=\{b := c\} \Downarrow_{comm@a} \Longrightarrow A\{c/b\} \Downarrow_{comm@a}$. For this, we prove by induction on the derivation of $A^=\{b := c\} \Downarrow_{comm@a}$:

(1) $A^=\{b := c\} \downarrow_{comm@a}$: we can show that this implies $A^= \downarrow_{comm@a}$ with $a \neq b$, which further implies $A \downarrow_{comm@a}$ by the formulation of $A^=$. So we have $A\{c/b\} \Downarrow_{comm@a}$.

(2) $A^=\{b := c\} \hookrightarrow D \Downarrow_{comm@a}$: any reduction of $A^=\{b := c\}$ can be matched by $A\{c/b\}$ since by the formulation of $A^=$ there is no direct interaction between $A^=$and b[**fwd** c]. For any of the possible reductions, we can find A' s.t. $D \rightleftharpoons A'\{b := c\}$ and $A\{c/b\} \hookrightarrow A'\{c/b\}$. By the same reasoning, we know that $\mathbb{CPI}_c \vdash A'\{b := c\} \simeq A'^=\{b := c\}$. So $D \Downarrow_{comm@a}$ implies $A'^=\{b := c\} \Downarrow_{comm@a}$. Now, we can apply the induction hypothesis and have $A'\{c/b\} \Downarrow_{comm@a}$. So from $A\{c/b\} \hookrightarrow A'\{c/b\}$, we know $A\{c/b\} \Downarrow_{comm@a}$. □

Lemma 54. *If* $A\{b := c\} \in \mathbb{CPI}^\circ$, *then* $A\{c/b\} \Downarrow_{comm@a} \Longrightarrow A\{b := c\} \Downarrow_{comm@a}$.

Proof. By induction on the derivation of $A\{c/b\} \Downarrow_{comm@a}$.

(1) $A\{c/b\} \downarrow_{comm@a}$: then we have $A\{c/b\} \rightleftharpoons (\nu X)(A_1 \mid \mathtt{get}.M\langle A_2\rangle@comm@a)$ with $a \notin X$. We must have $A \rightleftharpoons (\nu X)(B_1 \mid \mathtt{get}.M'\langle A_2'\rangle@comm@a')$ with $a' \notin X$ and $a'\{c/b\} = a$. From $A\{b := c\} \in \mathbb{CPI}^\circ$ we know that b is a forwarder name, which means $a' \neq b$, this means $a = a'\{c/b\} = a'$. So we have $A\{b := c\} \downarrow_{comm@a}$.

(2) $A\{c/b\} \hookrightarrow C \Downarrow_{comm@a}$: any reduction of $A\{c/b\}$ can be matched by either one or four (in the case of $A\{c/b\} \hookrightarrow C$ due to $(b.\mathtt{put}\langle C_1\rangle)\{c/b\} \hookrightarrow \mathtt{put}\langle C_1\rangle@c$) reductions of $A\{b := c\} \hookrightarrow^* D$. Moreover, in any of these cases we can find A' s.t. $C \rightleftharpoons A'\{c/b\}$ and $D \rightleftharpoons A'\{b := c\}$, so we can apply the induction hypothesis. □

Lemma 55. *For any solution* A, *if* $A\{b := c\} \in \mathbb{CPI}^\circ$, *then* $\mathbb{CPI}_c \vdash A\{b := c\} \simeq A\{c/b\}$.

Proof. By checking the definition.

(May-Loc) Easy.

(May-Test) For any $\mathcal{C}$ s.t. $\mathcal{C}(A\{b := c\}) \in \mathbb{CPI}_c$ and $\mathcal{C}(A\{c/b\}) \in \mathbb{CPI}_c$, we need to show $\mathcal{C}(A\{b := c\}) \Downarrow_{comm@a} \iff \mathcal{C}(A\{c/b\}) \Downarrow_{comm@a}$. Suppose $\mathcal{C} = (\nu X)(- \mid B)$. Since b does not appear free in $A\{c/b\}$ and is bound in $A\{b := c\}$, we can suppose b fresh and does not appear free in B. So we have $(\nu X)((A \mid B)\{b := c\}) \in \mathbb{CPI}_c$ which implies $(A \mid B)\{b := c\} \in \mathbb{CPI}^\circ$. So by the previous two lemmas, we know that $(A \mid B)\{b := c\} \Downarrow_{comm@a} \iff (A \mid B)\{c/b\} \Downarrow_{comm@a}$, which means $A\{b := c\} \mid B \Downarrow_{comm@a} \iff A\{c/b\} \mid B \Downarrow_{comm@a}$. Since we can easily show that for any A, B, and X: $A \Downarrow_{comm@a} \iff B \Downarrow_{comm@a}$ implies $(\nu X)A \Downarrow_{comm@a} \iff (\nu X)B \Downarrow_{comm@a}$, the result is proved. □

Lemma 39 (*Explicit substitution*). *For any* π_{af}*-process* p, *we have* $\mathbb{CPI}_c \vdash [\![p]\!]\{b := c\} \simeq [\![p\{c/b\}]\!]$.

Proof. Direct result of the previous lemma, by observing that $[\![p\{c/b\}]\!] = [\![p]\!]\{c/b\}$. □

Appendix F. Proof of the flattening Lemma 40

Definition 56. We define the formula $\mathbb{CPI}^-(x) =_{\mathrm{def}} (\gamma y)(\gamma z)(\mathbb{C}(xy) \mid \mathbb{V}(xyz,\ z) \mid \mathbb{O}_1(xyz)^* \mid \mathbb{I}_1(xyz)^*)$.

We first prove that for any $\mathbb{CPI}_c$-solution, we can obtain an equivalent $\mathbb{CPI}^-$-solution using Lemma 38.

Definition 57. For any solution A in $\mathbb{CPI}_c$, define A^- to be the solution obtained from A by the following steps:
(1) $\mathbb{O}_4 \Rightarrow \mathbb{O}_3$ using **(O-3)**: For any sub-solution $B \in \mathbb{O}_4$ of A, use **(O-3)** s.t. B is replaced by an $\mathbb{O}_3$-solution. (For simplicity, we will not expand in the following, and use $\mathbb{A} \Rightarrow \mathbb{B} \Rightarrow \mathbb{C}$ to stand for 2 consecutive steps: $\mathbb{A} \Rightarrow \mathbb{B}$ then $\mathbb{B} \Rightarrow \mathbb{C}$)
(2) $\mathbb{O}_3 \Rightarrow \mathbb{O}_2 \Rightarrow \mathbb{O}_1$ using **(O-2.1)** and **(O-1)**;
(3) $\mathbb{O}_0 \Rightarrow \mathbb{O}_1$ using **(O-2.2)**;
(4) $\mathbb{I}_4 \Rightarrow \mathbb{I}_3 \Rightarrow \mathbb{I}_2 \Rightarrow \mathbb{I}_1$ using **(I-3)**, **(I-2.1)**, and **(I-1)**;
(5) $\mathbb{I}_0 \Rightarrow \mathbb{I}_1$ using **(I-2.2)**;
(6) $\mathbb{I}_5 \Rightarrow \mathbb{I}_6 \Rightarrow \mathbb{I}_7 \Rightarrow (\mathbb{V} \mid \mathbb{I}_8)$ using **(I-5)**, **(I-6)**, **(I-7)**
(7) $\mathbb{I}_8 \Rightarrow (\gamma x)(\mathbb{C} \mid \mathbb{O}_1^* \mid \mathbb{I}_1^*)$ using **(I-8)**.

Lemma 58. *For any solution $A \in \mathbb{CPI}_c$, we have $A^- \in \mathbb{CPI}^-$ and $\mathbb{CPI}_c \vdash A \simeq A^-$.*

Proof. From Lemma 38, we know that the solution after each transformation step is equivalent to the one before. Then by congruence, we know $\mathbb{CPI}_c \vdash A \simeq A^-$. Moreover, after every step, the corresponding components are removed. As a result, the final solution A^- will not have any $\mathbb{O}_i$-solutions and $\mathbb{I}_j$-solutions ($i = 0, 2..4,\ j = 0, 2..8$). So it is a $\mathbb{CPI}^-$-solution. □

Definition 59 (*Zimmer [27]*). A (*tree*) *substitution* is a partial function $\sigma : \mathcal{N} \mapsto \mathcal{N}$ satisfying: for any $a \in dom(\sigma)$, there is integer k_a s.t. $a\sigma^{k_a} \notin dom(\sigma)$ (i.e. it is acyclic). We define σ^* to be the corresponding partial function s.t. for any $a \in dom(\sigma^*)$, $a\sigma^* =_{\text{def}} a\sigma^{k_a} \notin dom(\sigma^*)$.

For any solution $A \in \mathbb{CPI}_c$, we know that there are $X \subseteq \mathcal{N}$, $Y \subseteq \mathcal{N}$, $C \in \mathbb{C}$, $V \in \mathbb{V}$, $D \in \mathbb{D}$, s.t. $A \rightleftharpoons (vY)(vZ)(C \mid V \mid D)$. Suppose $V \rightleftharpoons \prod_{i=1}^k a_i[\mathbf{fwd}\ b_i]$. We associate A with a partial function σ_A which is defined as $\sigma_A =_{\text{def}} \bigcup_{i=1}^k \{(a_i, b_i)\}$.

Lemma 60. *For any π_{af}-process p, if $\langle\!\langle p \rangle\!\rangle \hookrightarrow^* A$, then σ_A is a tree substitution.*

Proof. The proof is by the observation that forwarders in A are created in some order. So we can assign a creation order to the domain of σ_A. Using this order, we know that any element in σ_A is a pair from a higher order name to a lower order name. So there will be no cycle. □

Definition 61 (*Flattening*). For any solution A with $\langle\!\langle p \rangle\!\rangle \hookrightarrow^* A$ for some π_{af}-process p, we define its *flattening* $A^\diamond$ to be the solution obtained by removing in A^- all the forwarders and the associated restrictions, then do the tree substitution σ_A^* to the resulting solution.

Lemma 40 (*Flattening*). *For any solution A with $\langle\!\langle p \rangle\!\rangle \hookrightarrow^* A$ for some π_{af}-process p, we have $A^\diamond \in \mathbb{PI}_c$ and $\mathbb{CPI}_c \vdash A \simeq A^\diamond$.*

Proof. The first result is easy. The equivalence is by Lemmas 58 and 39.

Suppose $fn(A) = X$ and $A^- \rightleftharpoons (vY)(vZ)(C \mid V \mid O_1 \mid I_1)$, with $C \in \mathbb{C}(XY)$, $V \in \mathbb{V}(XYZ,\ Z)$, $O_1 \in \mathbb{O}_1(XYZ)^*$, and $I_1 \in \mathbb{I}_1(XYZ)^*$.

First we prove $A^\diamond \rightleftharpoons (vY)(C \mid (O_1 \mid I_1)\sigma_A^*) \in \mathbb{PI}_c$. We know $\sigma_A = \sigma_{A^-}$, $dom(\sigma_A) = dom(\sigma_{A^-}) = Z$, and $im(\sigma_A) = im(\sigma_{A^-}) \in X \cup Y \cup Z$. So by the definition of tree substitution, we know that $im(\sigma_A^*) \in X \cup Y$. Since $O_1 \in \mathbb{O}_1(XYZ)^*$, we know that $O_1\sigma_A^* \in \mathbb{O}_1(XY)^*$. Similarly, $I_1\sigma_A^* \in \mathbb{I}_1(XY)^*$. With these, we know that $A \rightleftharpoons (vY)(C \mid O_1\sigma_A^* \mid I_1\sigma_A^*) \in \mathbb{PI}_c(X)$.

For the equational result, by Lemma 58, we know that $\mathbb{CPI}_c \vdash A \simeq A^-$, we only need to show that $\mathbb{CPI}_c \vdash A^- \simeq A^\diamond$, or $\mathbb{CPI}_c \vdash (vY)(vZ)(C \mid V \mid O_1 \mid I_1) \simeq (vY)(C \mid (O_1 \mid I_1)\sigma_A^*)$. From $A^\diamond \in \mathbb{PI}_c$, by Lemma 42.(1), we know that there is π_{af}-process $p = (O_1 \mid I_1)^{-1}$ s.t. $O_1 \mid I_1 \rightleftharpoons [\![p]\!]$. By repeated application of Lemma 39, we know that $\mathbb{CPI}_c \vdash (vZ)([\![p]\!] \mid V) \simeq [\![p\sigma_A^*]\!]$. Then by applying the context $(vY)(- \mid C)$ to the result, we get the final result. □

Appendix G. Proof of Theorem 43

Lemma 62. *For any π_{af}-process p, if $p \longrightarrow_\pi q$, then we can find a $\mathbb{CPI}_c$-solution A s.t. $\langle\!\langle p \rangle\!\rangle \hookrightarrow^* A$ and $\mathbb{CPI} \vdash \langle\!\langle q \rangle\!\rangle_{fn(p)} \simeq A$.*

Proof. From $p \longrightarrow_\pi q$, we can find names a, b, c, a π_{af}-process r, and a π_{af}-context C where the hole in context C is not guarded, s.t. $p \equiv_\pi C(a\langle c\rangle \mid a(b).r)$ and $q \equiv_\pi C(r\{c/b\})$. We can then find a chemical evaluation context $\mathcal{C}$ s.t. $\langle\!\langle p \rangle\!\rangle \rightleftharpoons \mathcal{C}(a.\mathtt{put}\langle comm.\mathtt{put}\langle \mathbf{fwd}\ c\rangle\rangle \mid (\nu b)(b \mid a.\mathtt{put}\langle comm.\mathtt{get}.\uparrow.\uparrow.b\langle\uparrow [\![r]\!]\rangle\rangle))$ and $\langle\!\langle q \rangle\!\rangle_{fn(p)} \rightleftharpoons \mathcal{C}([\![r\{c/b\}]\!])$. We let $A =_{\mathrm{def}} \mathcal{C}((\nu b)(b[\mathbf{fwd}\ c] \mid [\![r]\!]))$. We know that $\langle\!\langle p \rangle\!\rangle \hookrightarrow^* A$ and $\mathbb{CPI}_c \vdash \langle\!\langle q \rangle\!\rangle_{fn(p)} \simeq A$ (by Lemma 39). □

Lemma 63. *For any π_{af}-process p, for any A s.t. $\langle\!\langle p \rangle\!\rangle \hookrightarrow^* A$, there is a π_{af}-process q s.t. $\mathbb{CPI}_c \vdash \langle\!\langle q \rangle\!\rangle_{fn(p)} \simeq A$.*

Proof. We let $q = (A^\diamond)^{-1}$. First we know that $fn(A) = fn(p)$. So by Lemma 42.(2), $\langle\!\langle q \rangle\!\rangle_{fn(p)} = [\![(A^\diamond)^{-1}]\!] \mid \prod_{a \in fn(p)} a[\mathbf{chn}] \rightleftharpoons A^\diamond$. Then by Lemma 40, we know the result holds. □

Lemma 64. *For any π_{af}-process p, if $\langle\!\langle p \rangle\!\rangle \hookrightarrow^* A$, then $p \longrightarrow_\pi^* (A^\diamond)^{-1}$.*

Proof. The proof is by induction on the derivation of $\langle\!\langle p \rangle\!\rangle \hookrightarrow^* A$.

(1) If $A \rightleftharpoons \langle\!\langle p \rangle\!\rangle$, then $(A^\diamond)^{-1} \equiv_\pi (\langle\!\langle p \rangle\!\rangle^\diamond)^{-1} \equiv_\pi \langle\!\langle p \rangle\!\rangle^{-1} \equiv_\pi p$. So we have $p \longrightarrow_\pi^* (A^\diamond)^{-1}$.

(2) If $\langle\!\langle p \rangle\!\rangle \hookrightarrow^* A$ is by $\langle\!\langle p \rangle\!\rangle \hookrightarrow^* A'$ and $A' \hookrightarrow A$, then by induction hypothesis, we know that $p \longrightarrow_\pi (A'^\diamond)^{-1}$. We only need to show that $(A'^\diamond)^{-1} \longrightarrow_\pi^* (A^\diamond)^{-1}$. We prove this by analyzing the derivation of $A' \hookrightarrow A$.
   (a) The cases where Lemma 38 can be applied: we can show that $A'^\diamond \rightleftharpoons A^\diamond$. So we have $(A'^\diamond)^{-1} \longrightarrow_\pi^* (A^\diamond)^{-1}$.
   (b) The special case where $A' \rightleftharpoons \mathcal{C}(\mathtt{put}\langle \mathbf{fwd}\ c\rangle @ comm @ a \mid (\nu b)(b \mid \mathtt{get}.\uparrow.\uparrow.b\langle\uparrow\langle B\rangle\rangle @ comm @ a))$ and $A \rightleftharpoons \mathcal{C}((\nu b)(b \mid\uparrow.\uparrow.b\langle \mathbf{fwd}\ c \mid\uparrow\langle B\rangle\rangle @ comm @ a))$. Then, we can find a π_{af}-context C s.t. $(A'^\diamond)^{-1} \equiv_\pi C(a\langle c\rangle \mid a(b).B^{-1})$ and $(A^\diamond)^{-1} \equiv_\pi C(B^{-1}\{c/b\})$. So we have $(A'^\diamond)^{-1} \longrightarrow_\pi^* (A^\diamond)^{-1}$. □

Theorem 43 (*Operational correspondence*). *For any π_{af}-process p we have*

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

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

Proof. First we have the results in chemical semantics $\hookrightarrow$ by combining Lemmas 62–64. Then we can have the results in normal semantics by Proposition 18. □

References

[1] T. Amtoft, H. Makholm, J.B. Wells, PolyA: true type polymorphism for mobile ambients, in: J.-J. Levy, E.W. Mayr, J.C. Mitchell (Eds.), Proc. TCS 2004, Kluwer Academic Publishers, Dordrecht, 2004, pp. 591–604.

[2] G. Berry, G. Boudol, The chemical abstract machine, Theoret. Comput. Sci. 96 (1992) 217–248.

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

[4] L. Caires, L. Cardelli, A spatial logic for concurrency (part i), Inform. and Comput. 186 (2) (2003) 194–235.

[5] L. Cardelli, Abstraction for mobile computation, in: J. Vitek, C. Jensen (Eds.), Secure Internet Programming, Lecture Notes in Computer Science, Vol. 1603, Springer, Berlin, 1999, pp. 51–94.

[6] L. Cardelli, P. Gardner, G. Ghelli, Manipulating trees with hidden labels, in: Proc. FoSSaCS 2003, Lecture Notes in Computer Science, Vol. 2620, Springer, Berlin, 2003, pp. 216–232.

[7] L. Cardelli, A.D. Gordon, Mobile ambients, Theoret. Comput. Sci. 240 (1) (2000) 177–213, An extended abstract appeared in Proc. FoSSaCS '98, pp. 140–155.

[8] L. Cardelli, A.D. Gordon, Anytime anywhere: modal logics for mobile ambients, in: Proc. POPL '00, ACM, New York, January 2000, pp. 365–377.

[9] L. Cardelli, A.D. Gordon, Logical properties of name restriction, in: Proc. TLCA 2001, Lecture Notes in Computer Science, Vol. 2044, Springer, Berlin, 2001.

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

[11] S. Dal-Zilio, D. Lugiez, C. Meyssonnier, A logic you can count on, in: Proc. POPL 2004, ACM Press, New York, 2004, pp. 135–146.
[12] A.D. Gordon, L. Cardelli, Equational properties of mobile ambients, in: W. Thomas (Ed.), Proc. FoSSaCS '99, Lecture Notes in Computer Science, Vol. 1578, Springer, Berlin, 1999, pp. 212–226.
[13] X. Guan, Towards a tree of channels, in: V. Sassone (Ed.), Electronic Notes in Theoretical Computer Science, Vol. 85, Elsevier, Amsterdam, 2003.
[14] M. Hennessy, J. Riely, Resource access control in systems of mobile agents, J. Inform. and Comput. 173 (1) (2002) 82–120, An extended abstract appeared in Proc. HLCL'98, pp. 3–17.
[15] F. Levi, D. Sangiorgi, Mobile safe ambients, Trans. Programming Languages Systems 25 (1) (2003) 1–69.
[16] S. Maffeis, I. Phillips, On the computational strength of pure ambient calculi, in: Proc. Express 2003, ENTCS, Vol. 91, 2003.
[17] M. Merro, M. Hennessy, Bisimulation congruences in safe ambients, in: Proc. POPL 2002, 2002, pp. 71–80.
[18] M. Merro, F.Z. Nardelli, Bisimulation proof methods for mobile ambients, Proc. ICALP 2003, Lecture Notes in Computer Science, Vol. 2719, Springer, Berlin, July 2003(Available as COGS Technical Report 2003:1).
[19] R. Milner, J. Parrow, D. Walker, A calculus of mobile processes, part I/II, J. Inform. and Comput. 100 (1992) 1–77.
[20] R. Milner, D. Sangiorgi, Barbed bisimulation, in: W. Kuich (Ed.), Proc. ICALP '92, Lecture Notes in Computer Science, Vol. 623, Springer, Berlin, 1992, pp. 685–695.
[21] J.H. Morris, Lambda-calculus method of programming language, Ph.D. Thesis, MIT, December 1968.
[22] C. Palamidessi, Comparing the expressive power of the synchronous and the asynchronous π-calculus, in: Proc. POPL '97, ACM, New York, January 1997, pp. 256–265.
[23] I. Phillips, M.G. Vigliotti, Electoral systems in ambient calculi, in: Proc. FoSSaCS 2004, Lecture Notes in Computer Science, Vol. 2987, Springer, Berlin, 2004, pp. 408–422.
[24] A. Ravara, V.T. Vasconelos, Typing non-uniform concurrent objects, in: Proc. CONCUR 2000, Lecture Notes in Computer Science, Vol. 1877, Springer, Berlin, 2000.
[25] D. Sangiorgi, The name discipline of uniform receptiveness, Theoret. Comput. Sci. 221 (1–2) (1999) 457–493, An abstract appeared in the Proc. ICALP '97, Lecture Notes in Computer Science, Vol. 1256, Springer, Berlin, pp. 303–313.
[26] D. Sangiorgi, D. Walker, The π-calculus: a Theory of Mobile Processes, Cambridge University Press, Cambridge, 2001.
[27] P. Zimmer, On the expressiveness of pure safe ambients, Math. Structures Comput. Sci. 13 (2003) 721–770.