UNIVERSITY OF SUSSEX

COMPUTER SCIENCE

UNIVERSITY OF

Subtyping and Locality in Distributed Higher Order Processes

Nobuko Yoshida
Matthew Hennessy

with the core version of Facile [2, 9, 21] and CML [8] and can be regarded as an extension of Blue-calculus [5] to a higher-order term passing language.

A desirable feature of some distributed systems is that every channel name is associated with a ***unique*** receptor, which is called ***receptiveness*** in [32]; another property called ***locality*** where new receptors are not created by received channels, has also been studied in [3, 4, 22, 38] for an asynchronous version of the π-calculus. The combination of these constraints provides a model of a realistic distributed environment, which regards a receptor as an object or a thread existing in a unique name space. A generalisation is also proposed in Distributed Join-calculus where not only single receptor but also several receptors with the same input channel are allowed to exist in the same location [11]; in this paper we call this more general condition ***locality of channels***. In distributed object-oriented systems, objects with a given ID reside in a specific location even if multiple objects with the same ID are permitted to exist for efficiency reasons, as found for example in CONCURRENT AGGREGATES [7]. This locality constraint

Common Typing Rules:

$$\textsc{Id}: \quad \Gamma, u\!:\!\tau \vdash u : \tau \qquad \textsc{Sub}: \quad \frac{\Gamma \vdash P : \rho \quad \rho \leq \rho'}{\Gamma \vdash P : \rho'}$$

Functional Typing Rules:

$$\textsc{Const}: \quad \Gamma \vdash 1 : \mathtt{nat} \quad \text{etc.} \qquad \textsc{Abs}: \quad \frac{\Gamma, x\!:\!\tau \vdash P : \rho}{\Gamma \vdash \lambda(x\!:\!\tau).P : \tau}$$

Syntax:

System: $M,N,\ldots \in \mathit{System} \quad ::= \quad P \mid N \parallel M \mid (\nu a{:}\sigma)N$

Term: $P,Q,\ldots \in \mathit{Term} \quad ::= \quad \mathtt{Spawn}(P) \mid \cdots$ as Figure 1

Value: as Figure 1

Distributed Reduction Rules:

(process) $\dfrac{P \longmapsto P'}{P \longrightarrow P'}$ $P \longmapsto P'$ from Figure 2

(spawn) $(\cdots Q \mid \mathtt{Spawn}(P)) \longrightarrow (\cdots Q) \parallel P$

(com$_s$) $(u?(\tilde{x}{:}\tilde{\tau}).P \mid \cdots) \parallel (u!\langle\tilde{V}\rangle Q \mid \cdots) \longrightarrow (P\{\tilde{V}/\tilde{x}\} \mid \cdots) \parallel (Q \mid \cdots)$

(par$_s$) $\dfrac{M \longrightarrow M'}{M \parallel N \longrightarrow M' \parallel N}$ (res$_s$) $\dfrac{N \longrightarrow N'}{(\nu a{:}\sigma)N \longrightarrow (\nu a{:}\sigma)N'}$ (str$_s$) $\dfrac{N \equiv N' \longrightarrow M' \equiv M}{N \longrightarrow M}$

FIGURE 5. Syntax and Distributed Reduction in Dπλ

relation. The definition is outlined in Figure 5 and uses a structural equivalence on systems, defined by changing " $\mid$ " to " $\parallel$ " and P,Q,R to M,N,N' in Figure 2. The first two rules are the most important, namely spawning of a process at a new location (spawn) and communication between physically distinct locations, (com$_s$).

DEFINING LOCALITY We require that every input channel name is associated with a unique location. This is violated in, for example,

$$a?(y).\, P \;\parallel\; (a?(z).\, Q \mid b?(x_1).\, R_1 \mid b?(x_2).\, R_2)$$

because the name a can receive input at two distinct locations. Note however that the name b is located uniquely, although at that location a call can be serviced in two different ways.

A formal definition of this concept (or rather its complement), *locality error*,

Types:

$$
\begin{array}{llll}
\text{Term Type:} & \rho & ::= & \pi \mid \tau \\
\text{Process Type:} & \pi & ::= & \mathtt{proc} \mid \mathbf{s}(\mathtt{proc}) \\
\text{Value Type:} & \tau & ::= & \mathtt{unit} \mid \mathtt{nat} \mid \mathtt{bool} \mid \sigma \mid \mathbf{s}(\tau) \quad \text{with } \tau \neq \sigma \\
 & & \mid & \tau \to \rho \quad \text{with } \rho \leq \mathbf{s}(\rho') \Rightarrow \tau \leq \mathbf{s}(\tau') \\
\text{Channel Type:} & \sigma & ::= & \langle S_{\mathtt{I}}, S_{\mathtt{O}} \rangle \text{ with } S_{\mathtt{I}} \geq S_{\mathtt{O}},\ S_{\mathtt{I}} \neq \bot \text{ and } S_{\mathtt{O}} \neq \top \\
 & & \mid & \mathbf{s}(\langle \top, S_{\mathtt{O}} \rangle) \\
\text{Sort Type:} & S & ::= & \text{as in Figure 3.}
\end{array}
$$

Ordering: All rules from Figure 3 and

$$
\begin{array}{ll}
\text{(trans)} & \rho_1 \leq \rho_2 \quad \rho_2 \leq \rho_3 \ \Rightarrow\ \rho_1 \leq \rho_3 \\
\text{(mono)} & \rho \leq \rho' \ \Rightarrow\ \mathbf{s}(\rho) \leq \mathbf{s}(\rho') \\
\text{(sendable)} & \mathbf{s}(\rho) \leq \rho \\
\text{(id)} & \mathbf{s}(\rho) \leq \mathbf{s}(\mathbf{s}(\rho)) \\
\text{(lift)} & \mathbf{s}(\tau) \to \mathbf{s}(\rho) \leq \mathbf{s}(\mathbf{s}(\tau) \to \mathbf{s}(\rho))
\end{array}
$$

FIGURE 8. Locality types for D$\pi\lambda$

occurrence of $b!\langle V \rangle$, where the term V can be exported to another location, it can only evaluate to a value of *sendable* type.

4 Type Inference System for Locality

This section formalises a new typing system for processes. The important point of our typing is that if each process in each location is statically type-checked, we can automatically ensure that, in the global environment, input capability always resides at a unique location even after arbitrary computation.

LOCAL TYPING SYSTEM We add a new type constructor $\mathbf{s}(\rho)$ for sendable terms; the formation rules and ordering are given in Figure 8. The side condition of arrow types simply avoids, as we will see, a sendable term having a non-sendable subterm; e.g. if either P or Q is non-sendable, then $P\,Q$ will automatically be non-sendable. A similar side condition on arrow types can be found in the *passive types* in [26].

The first rule ensures that $\leq$ is a preorder. The second extra ordering says that the constructor $\mathbf{s}(\ )$ preserves subtyping, and the third that all sendable values are also values. In conjunction with (id), this rule implies that sendability is idempotent, $\mathbf{s}(\mathbf{s}(\rho)) \simeq \mathbf{s}(\rho)$ with $\simeq \overset{\text{def}}{=} \leq \cap \geq$. Similarly with (lift), we have: $\mathbf{s}(\mathbf{s}(\tau) \to \mathbf{s}(\rho)) \simeq \mathbf{s}(\tau) \to \mathbf{s}(\rho)$. Indeed this can be generalised:

LEMMA 4.1. *For any $k \geq 0$, we have:*

$$\mathbf{s}(\mathbf{s}(\tau_1) \to \cdots \to \mathbf{s}(\tau_k) \to \mathbf{s}(\rho)) \quad \simeq \quad \mathbf{s}(\tau_1) \to \cdots \to \mathbf{s}(\tau_k) \to \mathbf{s}(\rho) \ .$$

Proof A simple induction on k. □

We will change the distributed typing system by ensuring a value V will only be exportable to another location if it can be assigned a type of the form $\mathbf{s}(\tau)$. However because our typing system has a subsumption rule a more general statement would be that to be exportable we assign to V a type τ' such that $\tau' \leq \mathbf{s}(\tau)$, for some τ. This leads to a formal definition of ***sendable types***.

DEFINITION 4.2. Let Sble, the set of sendable types, be the least set of types which satisfies:

- $\mathbf{s}(\rho) \in \mathsf{Sble}$ for any type $\mathbf{s}(\rho)$.
- $\tau,\ \rho \in \mathsf{Sble}$ implies $\tau \to \rho \in \mathsf{Sble}$.

We say ρ is ***sendable*** if $\rho \in \mathsf{Sble}$. □

The main properties of the set of sendable types is given in the following proposition:

PROPOSITION 4.3.

1. Sble *is downwards closed with respect to subtyping:* $\rho' \leq \rho$ *and* $\rho \in \mathsf{Sble}$ *implies* $\rho' \in \mathsf{Sble}$
2. $\rho \in \mathsf{Sble}$ *if and only if* $\rho \simeq \mathbf{s}(\rho)$
3. $\rho \in \mathsf{Sble}$ *if and only if* $\rho \leq \mathbf{s}(\rho')$ *for some* ρ'*.*

Proof

1. The proof is by structural induction on ρ'; so we can assume that the statement is true of all ρ'' which are structurally less than ρ'. We now do a further induction on the proof of $\rho' \leq \rho$. Most of the cases are straightforward; for example the use of any of the axioms in Figure 8 are immediate. The only non-trivial case is when ρ', ρ have the structure $\tau_1 \to \rho_1, \tau_2 \to \rho_2$, respectively, where $\tau_2 \leq \tau_1$ and $\rho_1 \leq \rho_2$. We know $\rho_2 \in \mathsf{Sble}$ and therefore by induction $\rho_1 \in \mathsf{Sble}$. Also because of the constraint on the formation of arrow types we know that $\tau_1 \leq \mathbf{s}(\tau_3)$, for some τ_3. By construction $\mathbf{s}(\tau_3) \in \mathsf{Sble}$ and therefore, again by induction $\tau_1 \in \mathsf{Sble}$. It follows that $\tau_1 \to \rho_1 \in \mathsf{Sble}$.
2. Suppose $\rho \in \mathsf{Sble}$. A simple proof by induction on why $\rho \in \mathsf{Sble}$ gives $\rho \simeq \mathbf{s}(\rho)$, remembering that for any type ρ' we have $\mathbf{s}(\rho') \leq \rho'$ (if $\mathbf{s}(\rho')$ is defined). The case when ρ is an arrow type uses Lemma 4.1. The converse follows immediately from part one.
3. Now follows from parts one and two. □

Then **SqServ**

types. By asserting that the type on the bound variable Y is in Sble we can reject inappropriate inputs on the channel b. In a typing system based on the simple assertions of the form

$$\vdash \quad \tau \text{ is sendable}$$

where τ is a type from Section 2, such reasoning would be difficult.

SUBJECT REDUCTION In this subsection we prove locality is preserved under reduction. We take for granted that the revised type system satisfies the usual properties such as Narrowing, Weakening etc.; these may be checked by the reader.

LEMMA 4.12.

(1) (Algebra on environments) $\Gamma_1 \vdash \mathtt{SBL}$ *and* $\Gamma_2 \vdash \mathtt{SBL}$ *imply* $\Gamma_1 \sqcap \Gamma_2 \vdash \mathtt{SBL}$, and $\Delta_1 \asymp_1 \Delta_2$ and $\Delta_1 \asymp_1 \Delta_3$ implies $\Delta_1 \asymp_1 \Delta_2 \sqcap \Delta_3$.
(2) (Sendable types) If $\rho \in \mathsf{Sble}$ then $\Gamma \vdash_1 P : \rho$ *implies there exists* Δ *s.t.* $\Delta \geq \Gamma$, $\Delta \vdash_1 \mathtt{SBL}$, *and* $\Delta \vdash_1 P : \rho$.
(3) (Local substitution) *Suppose* $\Gamma, x : \tau \vdash_1 P : \rho$ *and* $\Gamma \vdash_1 V : \tau$. *Then we have* $\Gamma \vdash_1 P\{V/x\} : \rho$.

Suppose $\Gamma \vdash_1 (\nu a{:}\sigma)M \parallel N$. Then by induction, there are Γ_1 and Γ_2 which satisfy $\Gamma_1, a{:}\sigma \vdash_1 M$, $\Gamma_2 \vdash_1 N$ and $\Gamma_1 \asymp_1 \Gamma_2$. Since $a \notin \mathsf{dom}(\Gamma_2)$, we have $\Gamma_1, a{:}\sigma \asymp_1 \Gamma_2$. Hence we have $\Gamma_1, a{:}\sigma \sqcap \Gamma_2 \vdash_1 M \parallel N$ which implies, since $\Gamma_1, a{:}\sigma \sqcap \Gamma_2 = \Gamma_1 \sqcap \Gamma_2, a{:}\sigma$, that $\Gamma_1 \sqcap \Gamma_2 \vdash_1 (\nu a{:}\sigma)(M \parallel N)$, as required.

For the other direction, suppose $\Gamma \vdash_1 (\nu a{:}\sigma)(M \parallel N)$ with $a \notin \mathsf{fn}(N)$. Then by induction, we have: $\Gamma'_1 \vdash_1 M$ and $\Gamma'_2 \vdash_1 N$ such that $\Gamma'_1 \sqcap \Gamma'_2 = \Gamma, a{:}\sigma$. Since $a \notin \mathsf{fn}(N)$, we know that $\Gamma'_2/a \vdash_1 N$. Then by $(\Gamma'_1 \sqcap \{a{:}\sigma\})(a) = \sigma$, we have $\Gamma'_1 \sqcap \{a{:}\sigma\} \vdash_1 M$, which implies $\Gamma'_1/a \vdash_1 (\nu a{:}\sigma)M$. Finally by $\Gamma'_1/a \asymp_1 \Gamma'_2/a$, we have done. □ □

THEOREM 4.16. **(Subject Reduction Theorem)**

- *If* $\Gamma \vdash_1 P : \rho$ *and* $P \longmapsto Q$*, then* $\Gamma \vdash_1 Q : \rho$.
- *If* $\Gamma \vdash_1 N$ *and* $N \longrightarrow M$*, then* $\Gamma \vdash_1 M$.

Proof The proof for processes is identical to that of Theorem 2.6 and therefore we concentrate on that for systems. We use induction on the derivation of $N \longrightarrow M$ and because of the previous Lemma, and the first part of the Theorem, there are only two non-trivial cases:

(1) $(\cdots Q \,|\, \mathtt{Spawn}(P)) \longrightarrow (\cdots Q) \parallel P$

(2) $(u?(\tilde{x}{:}\tilde{\tau}).P \,|\, \cdots) \parallel (u!\langle\tilde{V}\rangle Q \,|\, \cdots) \longrightarrow (P\{\tilde{V}/\tilde{x}\} \,|\, \cdots) \parallel (Q \,|\, \cdots)$.

Case (1): Assume $\Gamma \vdash_1 (\cdots Q \,|\, \mathtt{Spawn}(P))$ and therefore in particular that $\Gamma \vdash_1 \mathtt{Spawn}(P) : \mathtt{proc}$. Then $\Gamma \vdash_1 P : \mathbf{s}(\mathtt{proc})$. Hence by Lemma 4.12 (2), there exists $\Delta \vdash_1 \mathtt{SBL}$ such that $\Gamma \leq \Delta$ and $\Delta \vdash_1 P : \mathtt{proc}$. It follows that $\Gamma \vdash_1 (\cdots Q) \parallel P$ since $\Gamma = \Gamma \sqcap \Delta$ and $\Gamma \asymp_1 \Delta$.

Case (2): Let us just consider the case

$$u?(\tilde{x}{:}\tilde{\tau}).\,P \parallel u!\langle\tilde{V}\rangle.\,Q \longrightarrow P\{\tilde{V}/\tilde{x}\} \parallel Q \qquad (4.1)$$

We know that Γ has the form $\Gamma_1 \sqcap \Gamma_2$ where $\Gamma_1 \vdash_1 u?(\tilde{x}{:}\tilde{\tau}).\,P : \mathtt{proc}$ and $\Gamma_2 \vdash_1 u!\langle\tilde{V}\rangle.\,Q : \mathtt{proc}$ with $\Gamma_1 \asymp_1 \Gamma_2$. We show that for some Γ'_1 such that $\Gamma_1 \sqcap \Gamma_2 = \Gamma'_1 \sqcap \Gamma_2$ and $\Gamma'_1 \asymp_1 \Gamma_2$, we have $\Gamma'_1 \vdash_1 P\{\tilde{V}/\tilde{x}\}$ and $\Gamma_2 \vdash_1 Q$.

First, since u is not local in the left-hand side location, $\Gamma_2 \vdash_1 u!\langle$

We now briefly outline how our typing system can be adapted to these more general types; In the revised system judgements take the form

$$\Gamma \vdash_{g} P : \rho$$

First we need a more general replacement for the rule $\textsc{Chan}_I$ in Figure 9 to where both systems in the pair are typable under environment Γ. We show this relation is a strong barbed reduction-closed up to $\sim_\Gamma$ and restriction.

To establish this, we first define the same notion of Definition 5.3.1 in [27]; we say *a is only used as a trigger in P* if $\Gamma \vdash C[P] : \mathtt{proc}$ and there exists Δ s.t. $\Delta \geq \Gamma$, $\Delta \vdash_{\mathtt{1}} P : \mathtt{proc}$ and $\Delta(a) = \langle \top, S_{\mathtt{O}} \rangle$ or $\Delta(a) \leq \mathbf{s}(\tau)$. Then we observe that:

1. Suppose R is sendable. Then $\Gamma \vdash_{\mathtt{1}} (P \,|\, R) \,\|\, Q$ iff $\Gamma \vdash_{\mathtt{1}} P \,\|\, R \,\|\, Q$, and $(P \,|\, R) \,\|\, Q \sim_\Gamma P \,\|\, R \,\|\, Q$.
2. If R is sendable, then all free names in R are only used as a trigger.
3. a is only used as a trigger in P and Q since the left-hand side of the above equation is typable. Hence P and Q may only export the sendable value V via a.
4. If a is used as a trigger in Q and V is sendable, then a in $Q\{V/X\}$ is again only used as a trigger.

Now take the set $\mathbf{R}'$ of typable pairs of the form

$$\langle (\nu\, a)(N\{a/a'\} \,\|\, R \,\|\, P\{a/a'\} \,\|\, Q\{a/a'\}), (\nu\, a, a')(N \,\|\, (R \,|\, P) \,\|\, (R\{a'/a\} \,|\, Q)) \rangle$$

where $a' \notin \mathsf{fn}(R)$ and a and a' are used as triggers in N,R_i, P and Q (note this is a simple extension of the equation (25) in Appendix B in [27]). To establish $\mathbf{R}$ is in $\sim_\Gamma$, we show the above relation $\mathbf{R}'$ is again a strong barbed reduction-closed up to $\sim_\Gamma$, using 1 to 4 above. □

Note we do not require any side condition for P and Q, for example stating that P and Q must be of a certain syntactic form; instead the typing system enforces implicit constraints on the various components of the systems. Note also that this proposition can not be derived in the framework of [32] since a is neither a linear nor an ω-receptive name.

Such theorems will be useful for reasoning about object-oriented systems where templates are shared among locations. Further extension of typed equivalences studied in π-calculus (e.g. [32, 37]) to distributed higher-order processes is an interesting research topic which we intend to pursue.

5.3 Type Checking

For a practical use of a typing system, it is essential that we can check the well-typedness of a system N against a global type environment Γ. For this purpose, first we propose a typing system ($\mathbf{Min}_{\pi\lambda}$) which can induce the minimum type of a given term P and Γ (if it has a type) in $\pi\lambda$ in Section 2, by deleting SUB and modifying APP, OUT and IN in Figure 4 as follows.

$$\text{APP}_m: \quad \frac{\Gamma \vdash P : \tau_1 \rightarrow \rho \quad \Gamma \vdash Q : \tau_2 \quad \tau_2 \leq \tau_1}{\Gamma \vdash PQ : \rho}$$

$$\text{IN}_m: \quad \frac{\Gamma(u) \leq (\tilde{\tau})^{\mathtt{I}} \quad \Gamma, \tilde{x}:\tilde{\tau} \vdash P : \mathtt{proc}}{\Gamma \vdash u?(\tilde{x}:\tilde{\tau}).P : \mathtt{proc}} \qquad \text{OUT}_m: \quad \frac{\Gamma \vdash V_i : \tau_i \quad \Gamma(u) \leq (\tilde{\tau})^{\mathtt{O}} \quad \Gamma \vdash P : \mathtt{proc}}{\Gamma \vdash u!\langle \tilde{V} \rangle P : \mathtt{proc}}$$

PROPOSITION 5.4.

(1) (ordering) $\rho_1 \leq \rho_2$ is decidable.
(2) (the minimum type in $\pi\lambda$) In the typing system in Figure 4, if $\Gamma \vdash P : \rho$ then there exists ρ' such that $\Gamma \vdash P : \rho'$ and, for any ρ_0, if $\Gamma \vdash P : \rho_0$, then $\rho' \leq \rho_0$.
(3) (algorithm) There is a type-checking algorithm that given type environment Γ and term P, computes the minimum type ρ such that $\Gamma \vdash P : \rho$ if one exists.

PROOF. **(1)** we first note functional types of our system correspond the regular system in [20] (see Theorem 5 in [20]). Next we observe that a subtyping between channel types $\langle S_{1\mathtt{I}}, S_{1\mathtt{O}} \rangle \leq \langle S_{2\mathtt{I}}, S_{2\mathtt{O}} \rangle$ are also computed in the same way as the arrow types; hence the subtyping relation is decidable.

(2) by easy induction on the derivations of $\Gamma \vdash P : \rho$ in $\mathbf{Min}_{\pi\lambda}$, we obtain (a) if $\Gamma \vdash P : \rho$ is derived from $\mathbf{Min}_{\pi\lambda}$, then it is also derived from the system in Figure 4, (b, unique type) $\Gamma \vdash P : \rho_1$ and $\Gamma \vdash P : \rho_2$ are derived in $\mathbf{Min}_{\pi\lambda}$, then $\rho_1 = \rho_2$, and (c, $\mathbf{Min}_{\pi\lambda}$ has smaller types) If $\Gamma \vdash P : \rho$ is derived from the system in Figure 4, then $\Gamma \vdash P : \rho'$ is derived from $\mathbf{Min}_{\pi\lambda}$ for some ρ' such that $\rho' \leq \rho$. By (a,b,c), (2) is straightforward.

(3) It can be constructed straightforwardly based on $\mathbf{Min}_{\pi\lambda}$. See Appendix C for that algorithm. □

$$\begin{array}{lcll}
\mathtt{up}(\rho_0) & = & \mathbf{s}(\tau_1) \to \mathbf{s}(\tau_2) \cdots \mathbf{s}(\tau_n) \to \rho_1 & \text{if } \rho_0 \text{ is in the form of (1) above} \\
\mathtt{up}(\rho_0) & = & \rho & \text{else if } \rho_0 = \mathbf{s}(\rho) \\
\mathtt{up}(\rho_0) & = & \rho_0 & \text{else}
\end{array}$$

We can easily observe that if $\rho \lneq \rho'$, then $\mathtt{up}(\rho) \notin \mathsf{Sble}$ and $\rho \leq \mathtt{up}(\rho) \leq \rho'$.

and the functional aspects of the language, as in Facile [9]. However extensions of our capability based typing systems to more advanced distributed primitives, such as hierarchical location spaces [36], process mobility [6, 33, 11], and cryptographic constructs [1, 14] will be more challenging. Since in our language we inherit the standard subtyping of the λ-calculus, it is also be possible to consider the introduction of richer subtyping relations, for example those based on records, recursive types, or polymorphic types into type systems for distributed languages.

As argued in [9, 8, 21, 25, 33], many practical applications call for parameterised higher-order process passing, which may be difficult to represent directly without functional constructions, even in languages which support migration of the processes. Moreover their presence leads to a natural and powerful program-

(i) (substitution) *If* $\Gamma, x:\tau \vdash P:\rho$ *and* $\Gamma \vdash V:\tau$. *Then* $\Gamma \vdash P\{V/x\}:\rho$.

(ii) (struct) *If* $\Gamma \vdash P:\mathtt{proc}$ *and* $P \equiv Q$*, then we have* $\Gamma \vdash Q:\mathtt{proc}$.

PROOF. (1) is proved by the induction on P in the standard way. For (2), since $P \equiv Q$ is defined only between processes, the proof follows the standard manner as done in [35, 27, 15]. Hence we omit the proofs. □

Now we prove Theorem 2.6. Then by Lemma A.1 (2) above, we only have to prove the following two cases.

(1) Suppose $\Gamma \vdash (\lambda(x:\tau).Q)V:\rho$ and $(\lambda(x:\tau).Q)V \longmapsto Q\{V/x\}$. Then we have: $\Gamma \vdash Q\{V/x\}:\rho$.

(2) Suppose $\Gamma \vdash u?(\tilde{x}:\tilde{\tau}).P \,|\, u!\langle\tilde{V}\rangle Q:\mathtt{proc}$ and $u?(\tilde{x}:\tilde{\tau}).P \,|\, u!\langle\tilde{V}\rangle Q \longmapsto P\{\tilde{V}/\tilde{x}\} \,|\, Q$. Then we have: $\Gamma \vdash P\{\tilde{V}/\tilde{x}\} \,|\, Q:\mathtt{proc}$.

Case (1): Suppose $\Gamma \vdash ((\lambda(x:\tau).P)\,V):\rho'$. Then we have the following derivation.

$$\frac{\Gamma, x:\tau \vdash_1 P:\rho_1 \quad \Gamma \vdash_1 V:\tau_2 \quad \tau_2 \le \tau \quad \rho_1 \le \rho'}{\Gamma \vdash (\lambda(x:\tau).P)\,V:\rho'}$$

Then by SUB, we have $\Gamma \vdash_1 V:\tau$. Hence by Lemma A.1 (1), we get: $\Gamma \vdash P\{V/x\}:\rho_1$. By applying SUB again, we have $\Gamma \vdash P\{V/x\}:\rho$, as desired.

Case (2): Suppose $\Gamma \vdash u?(\tilde{x}:\tilde{\tau}).Q \,|\, u!\langle\tilde{V}\rangle R:\mathtt{proc}$. Then we have the following derivations:

$$\Gamma \vdash u:(\tilde{\tau})^{\mathtt{I}}, \quad \text{and} \quad \Gamma,\, \tilde{x}:\tilde{\tau} \vdash Q:\mathtt{proc}$$

and

$$\Gamma \vdash u:(\tilde{\tau}')^{\mathtt{O}}, \quad \Gamma \vdash u:V_i:\tau_i', \quad \text{and} \quad \Gamma \vdash R:\mathtt{proc}$$

Let us define $\Gamma(u) = \langle S_{\mathtt{I}}, S_{\mathtt{O}}\rangle$. Then by SUB, we know: $S_{\mathtt{I}} \le (\tilde{\tau})$ and $S_{\mathtt{O}} \ge (\tilde{\tau}')$. Since always $S_{\mathtt{O}} \le S_{\mathtt{I}}$ by definition, we know $(\tilde{\tau}') \le (\tilde{\tau})$, which implies $\Gamma \vdash u: V_i:\tau_i$ by SUB. Now applying Lemma A.1 (1), we have $\Gamma \vdash P\{\tilde{V}/\tilde{x}\} \,|\, Q:\mathtt{proc}$. □

B Proofs of Section 4

B.1 Proof of Proposition 4.5

As in the proof of Proposition 6.2 in [15], we show the operators, $\sqcap$ and $\sqcup$ defined in the proof of Proposition 4.5 are partial meet and join operators. We only have to show, for every type α, β, γ,

$$\begin{array}{llllll}
\text{(a)} & \alpha \le \beta \text{ and } \alpha \le \gamma & \text{imply} & \beta \sqcap \gamma \text{ defined} & \text{and} & \alpha \le \beta \sqcap \gamma \\
\text{(b)} & \beta \le \alpha \text{ and } \gamma \le \alpha & \text{imply} & \beta \sqcup \gamma \text{ defined} & \text{and} & \beta \sqcup \gamma \le \alpha \\
\text{(c)} & \beta \sqcap \gamma & \text{implies} & \beta \sqcap \gamma \le \beta & & \\
\text{(d)} & \beta \sqcup \gamma & \text{implies} & \beta \le \beta \sqcup \gamma & &
\end{array}$$

By Lemma 4.4, we can consider every type takes a form either (1),(2) or (3) in Lemma 4.4. Only interesting cases are β is sendable but γ is not sendable. Others are easy by inductive hypothesis on types.

Base case ($k = 0$ in (2,3) of Lemma 4.4): The only interesting case is channel types. Suppose $\beta = \mathbf{s}(\langle \top, S_{1\mathtt{O}}\rangle)$ and $\gamma = \langle S_{2\mathtt{I}}, S_{2\mathtt{O}}\rangle$. For (a), suppose $\alpha \le \beta$ and $\alpha \le \gamma$. Then $\alpha \le \beta$ implies $\alpha = \mathbf{s}(\langle \top, S_{2\mathtt{O}}\rangle)$ by Proposition 4.3. Then by this and $\alpha \le \gamma$, we have $S_{2\mathtt{I}} = \top$. By induction on $S_{1\mathtt{O}}$ and $S_{2\mathtt{O}}$, $S_{1\mathtt{O}} \sqcup S_{2\mathtt{O}}$ always exists, and we have $S_{1\mathtt{O}} \sqcup S_{2\mathtt{O}} \le S_{0\mathtt{O}}$. Hence $\beta \sqcap \gamma = \mathbf{s}(\langle \top, S_{1\mathtt{O}} \sqcup S_{2\mathtt{O}}\rangle)$ is always defined, and $\alpha \le \beta \sqcap \gamma$. For (b), we first note that $\gamma \le \alpha$ implies α should be non-sendable, and, by $\beta \le \alpha$, we can set $\alpha = \langle \top, S_{2\mathtt{O}}\rangle$. Then the rest of reasoning is just similar with (a). (c) and (d) are also similar with (a) and (b), respectively.

Inductive Case ($k \ge 1$ in Lemma 4.4): The only interesting cases are β is in (1) in Lemma 4.4, while γ is in (2) or (3). Here we must check operations $\beta \sqcap \gamma$ and $\beta \sqcup \gamma$ do not induce illegal arrow types $\tau \to \rho$ such that $\rho \in \mathsf{Sble}$ but $\tau \notin \mathsf{Sble}$. We only show the case of γ in (3). The case (2) is just similar. For (a), suppose

$$\beta \simeq \mathbf{s}(\tau_{11}) \to \cdots \to \mathbf{s}(\tau_{k1}) \to \mathbf{s}(\rho_1) \quad \text{and} \quad \gamma \simeq \mathbf{s}(\tau_{12} \to \cdots \tau_{k2} \to \rho_2)$$

where $\alpha \le \beta$ and $\alpha \le \gamma$ for some α. First we immediately know α can not take the form (2) in Lemma 4.4 by $\alpha \le \beta$ and Proposition 4.3. So let us define $\alpha \simeq \mathbf{s}(\tau_1 \to \cdots \to \rho)$. Then $\alpha \le \beta$ implies $\rho \le \mathbf{s}(\rho_1)$, hence by Proposition 4.3 again, we have $\rho \simeq \mathbf{s}(\rho')$ for some ρ'. By the formation of arrow type, we have: $\tau_i \simeq \mathbf{s}(\tau_i')$. Hence α should take a form

$$\alpha \simeq \mathbf{s}(\tau_1') \to \cdots \mathbf{s}(\tau_k') \to \mathbf{s}(\rho')$$

for some τ'

$$\begin{array}{llll}
G(\Delta,\, u) & = & & \Delta \vdash u : \Delta(x) \\
G(\Delta,\, l) & = & & \Delta \vdash l : \mathtt{nat}\ \ \text{etc.} \\
G(\Delta,\, P\,Q) & = & \text{let} & \Delta \vdash P : \tau_1 \rightarrow \rho = G(\Delta,\, P) \\
 & & & \Delta \vdash Q : \tau_2 = G(\Delta,\, Q) \\
 & & \text{in} & \text{if}\ \mathtt{Check}(\tau_2 \leq \tau_1)\ \text{then}\ \Delta \vdash P\,Q : \rho \\
G(\Delta,\, \lambda(x{:}\tau).P) & = & \text{let} & \Delta \vdash P : \rho = G(\Delta,\, P) \\
 & & \text{in} & \Delta \vdash \lambda(x{:}\tau).P : \tau \rightarrow \rho\ \ \text{with}\ \ x{:}\tau \in \Delta \\
G(\Delta,\, \mathbf{0}) & = & & \Delta \vdash \mathbf{0} : \mathtt{proc} \\
G(\Delta,\, P\,|\,Q) & = & \text{let} & \Delta \vdash P : \mathtt{proc} = G(\Delta,\, P) \\
 & & & \Delta \vdash Q : \mathtt{proc} = G(\Delta,\, Q) \\
 & & \text{in} & \Delta \vdash P\,|\,Q : \mathtt{proc}
\end{array}$$

proc G(Δ,λ∗1P4)8198]Tϑ /P=17 9let162Σ2 TP 5.76914 0.75Δ,1P5 Tµ23818]Tϑ /P.19 9ϕ1 0294.2684.2652[
()()

$$\begin{array}{llll}
 & & \text{in} & \Delta \vdash *P : \mathtt{proc} \\
G(\Delta,\, (\nu a{:}\sigma & & &
\end{array}$$

$$
\begin{array}{lcll}
G_s(\Delta,\ P) & = & \text{let} & \Delta;\Gamma \vdash_1 P : \pi = G_l(\Delta,\ P) \\
 & & \text{in} & \Gamma \vdash_1 P \\
G_s(\Delta,\ N_1 \,\|\, N_2) & = & \text{let} & \Gamma_1 \vdash_1 N_1 = G_s(\Delta,\ N_1) \\
) & = & & \Gamma_2 \vdash_1 N_2 = G_s(\Delta,\ N_2) \\
 & & \text{in} & \text{if } \mathtt{Check}(\Gamma_1 \asymp_1 \Gamma_2) \text{ then } \Gamma_1 \sqcap \Gamma_2 \vdash_1 N_1 \,\|\, N_2 \\
G_s(\Delta,\ (a\!:\!\sigma)N) & = & \text{let} & \Gamma \vdash_1 \\
 & & & \qquad\qquad\qquad \text{ƽ}
\end{array}
$$

[25] De Nicola, R., Ferrari, G. and Pugliese, R., Klaim: a Kernel Language for Agents Interaction and Mobility IEEE Trans. on Software Engineering, Vol.24(5), May, 1998.

[26] O'Hearn, P., Power, J., Takeyama, M., and Tennent, D., Syntactic Control of Interference Revised, *Proc. MFPS'97*, *ENCS*, Elsevier, 1997.

[27] Pierce, B.C. and Sangiorgi. D, Typing and subtyping for mobile processes. *MSCS*, 6(5):409–454, 1996.

[28] Pierce, B. and Turner, D., Pict: A Programming Language Based on the Pi-calculus, Indiana University, CSCI Technical Report, 476, March, 1997.

[29] Plotkin, G., Call-by-name, call-by-value and the lambda-calculus, *TCS*, 1:125–159, 1975.

[30] Riely, J. and Hennessy, M., Trust and Partial Typing in Open Systems of Mobile Agents, CS