数理解析研究所講究録
996巻 1997年 125-142

On Typing Systems for the Polyadic π-Calculus

Atsushi Togashi*

Department of Computer Science, Shizuoka University,
3-5-1, Johoku, Hamamatsu 432, Japan
Tel.: +81-53-478-1463
Fax.: +81-53-475-4595
togashi@cs.inf.shizuoka.ac.jp

Abstract

In the literature, there have been intensive studies on typing (sorting) systems for the polyadic π-calculus, originated by Milner's sorting discipline [10] based on name matching. The proposed systems, so far, are categorized into the two groups — systems by name matching and ones by structure matching (possibly with subtyping) — and obtain similar results. A natural question arises "Is there any relationship between the two paradigms ?". With this motivation, the present paper gives deeper investigations on typing systems between the two approaches. For this purpose, a sorting system by name matching, a quite similar to the system in [7], and a typing system by structure matching with subtyping, a slight extension of the system in [12], are presented, along with several basic properties. Then, correspondence between the sorting system and the typing system is investigated via transformations both form sortings to typings and from typings to sortings. It is shown that if a process is well-sorted *w.r.t.* a safe sorting in the sorting system, then it is well-typed for the transformed typing in the typing system, but not vice versa. This result can be straightforwardly extended to Liu and Walker's consistent sortings. Under a certain condition, we can show the reverse implication. Furtheremore, on the other direction from typings to sortings, it is shown that the derived typing from the sorting which is the result of applying transformation to a typing coincides with the original typing. However, the derived sorting from the typing which is the result of applying transformation to a sorting is proved to be a proper specialization of the original sorting.

*The work has been done during visiting the COGS, University of Sussex, Farmer, Brighton BN1 9QH, England.

1 Introduction

The *π-calculus* [11] has achieved a remarkable simplification by focusing on naming and allowing the communicated data along channels (names) to be names themselves. The calculus is sufficiently expressive to describe mobile systems and the ability of natural embeddings of both lazy and call-by-value λ-calculi into the π-calculus [9] suggests that it may form an appropriate foundation for the design of new programming languages. It has been shown that higher-order processes can be faithfully encoded in the π-calculus [13]. The *polyadic π-calculus* by Milner [10] is a straightforward generalization of the monadic π-calculus [11], in which finite tuples of names, instead of single names, are the atomic unit of communication. Furthermore, the fact that a tuple of names is exchanged at each communication step suggests a natural discipline of sorting.

In the literature, there have been intensive studies on the topic of *typing* (*sorting*) *systems* for the polyadic π-calculus, originated by Milner's *sorting discipline* [10] based on name matching. *Name matching* (or, *by-name matching*) determines sort equality by relying on the syntactical names assigned to communication channels (or names) in a given process, instead of no structure. An algorithm to infer the most general sorting of a term has been reported by Gay in [6]. Milner's original idea is further extended and explored by Liu and Walker in [7], where an input sorting and an output sorting are distinguished. On the other hand, typing systems based on structure matching are introduced in [16, 12]. In the *structure matching* (or, *by-structure matching*), type equality or subtyping is determined by some abstract type structure, no by how types are syntactically presented. The systems in both categories – the ones by name matching and the ones by structure matching – are used to verify run-time type error and obtain the similar results. A natural question arises "Is there any relationship between the two paradigms ?". The correspondence between Milner's sorting and the typing system [16] is discussed in [15].

With this motivation, the present paper gives deeper investigations on the typing systems between the two approaches. For this purpose, a sorting system by name matching, a quite similar to the system in [7], and a typing system by structure matching with subtyping, a slight extension of the system in [12], are presented, along with several basic properties. Then, correspondence between the sorting system and the typing system is investigated via two transformations form sortings to typings and from typings to sortings.

On the transformation from sortings to typings, it is shown that if a process is well-sorted *w.r.t.* for a safe sorting in the sorting system, then it is well-typed for the transformed typing in the typing system, but not vice versa. An illustrative counter example will be given. This result can be straightforwardly extended to Liu and Walker's consistent sortings. Under a certain condition, we can show the reverse implication.

On the other direction from typings to sortings, it is shown that the derived typing from the sorting which is the result of applying transformation to a typing coincides with the original typing. However, the derived sorting from the typing which is the result of applying transformation to a sorting is proved to be a proper specialization of the original sorting.

The outline of the paper is as follows: Section 2 presents the polyadic π-calculus to a certain extent needed for the study. Section 3 and 4 introduce a sorting system

and a typing system, respectively. Section 5, the main part of this paper, relates the sorting system and the typing system via both-directional transformations. This paper is concluded in Section 6 with some concluding remarks.

2 The Polyadic π-Calculus

This section introduces the polyadic π-calculus [10], a straightforward extension of the monadic π-calculus [11], to a certain extent needed for the study. Let $\mathcal{N}$ be a possibly infinite set of *names*. The basic syntax of *processes* we consider in this paper is defined by the following grammar:

$$P ::= \mathbf{0} \mid a(x_1,\ldots,x_n).P \mid \bar{a}\langle b_1,\ldots,b_n\rangle.P \mid P \mid Q \mid (\boldsymbol{\nu} x)P \mid !P$$

where $\mathbf{0}$ is the *nil* process; $a(x_1,\ldots,x_n).P$ and $\bar{a}\langle b_1,\ldots,b_n\rangle.P$ are *input-prefixes* and *output-prefixes*, respectively; $P \mid Q$ are *parallel compositions*; $(\boldsymbol{\nu} x)P$ are *restrictions*; $!P$ are *replications*. We use the metavariables a, b, c, x, y, z, etc for names; P, Q, and R for processes. A sequence $x_1,\ldots,x_n$ of names is often written $\tilde{x}$ if its length $|\tilde{x}|$ is not important. For a process P, the set $fn(P)$ of *free names* and the set $bn(P)$ of *bound names* are defined in the usual way. We formally identify processes P up to renaming bound names in P, so that it is assumed that $fn(P) \cap bn(P) = \emptyset$. This implies the usual conventions about substitutions to avoid capturing of free names during substitution, α-conversion, side-condition concerning freshness of names, etc.

A *structural congruence relation* $\equiv$ is defined to be the smallest congruence relation over processes which satisfies the axiom schemes listed below.

1. If $P \equiv_\alpha Q$ then $P \equiv Q$: Processes are identified if they differ only by a change of bound names.
2. $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$; $\quad P \mid Q \equiv Q \mid P$; $\quad P \mid \mathbf{0} \equiv P$.
3. $!P \equiv !P \mid P$.
4. $(\boldsymbol{\nu} x)P \equiv P \quad$ if $x \notin fn(P)$[1]; $\quad (\boldsymbol{\nu} x)(\boldsymbol{\nu} y)P \equiv (\boldsymbol{\nu} y)(\boldsymbol{\nu} x)P$;
   $(\boldsymbol{\nu} x)P \mid Q \equiv (\boldsymbol{\nu} x)(P \mid Q) \quad$ if $x \notin fn(Q)$[2].

Now, we define a *reduction relation* $\rightarrow$ over processes to be the smallest relation satisfying the following rules:

$$\textsc{Comm}\ \frac{}{a(\tilde{x}).P \mid \bar{a}\langle\tilde{b}\rangle.Q \rightarrow P\{\tilde{b}/\tilde{x}\} \mid Q}\ |\tilde{x}| = |\tilde{b}| \qquad \textsc{Par}\ \frac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q}$$

$$\textsc{Rest}\ \frac{P \rightarrow P'}{(\boldsymbol{\nu} x)P \rightarrow (\boldsymbol{\nu} x)P'} \qquad \textsc{Struct}\ \frac{Q \equiv P \quad P \rightarrow P' \quad P' \equiv Q'}{Q \rightarrow Q'}$$

[1] This induces the usual axiom schemes: $(\boldsymbol{\nu} x)\mathbf{0} \equiv \mathbf{0}$; $\quad (\boldsymbol{\nu} x)(\boldsymbol{\nu} x)P \equiv (\boldsymbol{\nu} x)P$.

[2] Note that the side condition can be viewed as a consequence of our convention of regarding bound names.

3 A Sorting System by Name Matching

The introduction of sorting discipline into the π-calculus [10] intends to ensure that names are used consistently. In this section we present a sorting system based on Milner's original sorting discipline, the resulting system is quite similar to the typing system by Liu and Walker [7].

Let Σ be a finite set of (*subject*) *sorts*. Σ^* denotes the set of all finite sequences of elements in Σ. An element in Σ^* is called an *object sort*, denoted by $(s_1, \ldots, s_n)$, or simplely $(\tilde{s})$ if the number of the sequence is not important. We use u and v to range over Σ^*. A *subject sorting* Γ on Σ is a finite set of *subject sort assignments* $a : s$, where $a \in \mathcal{N}$ and $s \in \Sigma$, such that $a : s$, $a : t \in \Gamma$ implies $s = t$. An *object sorting* Ω on Σ is a finite set of *object sort assignments* of the form either $s^+ : u$ or $s^- : u$, where $s \in \Sigma$ and $u \in \Sigma^*$, such that $s^\star : u$, $s^\star : v \in \Omega$ implies $u = v$, for $\star \in \{+, -\}$. A *sorting* on Σ is a pair $\Gamma; \Omega$ of a subject sorting Γ and an object sorting Ω. Ω is *safe* in s if $s^+ : u$, $s^- : v \in \Omega$ implies $u = v$. If Ω is safe in all s in Σ then Ω is called *safe*. A sorting $\Gamma; \Omega$ is *safe* if its object sorting Ω is safe.

Definition 3.1 A *sorting judgment* (*by name matching*) on Σ is an expression of the form: $\Gamma; \Omega \vdash P : ()$, where $\Gamma; \Omega$ is a sorting, P is a process, and $()$ is the special symbol standing for well-behavedness of a process. □

An object sorting is usually called a sorting [10, 6, 7]. As usual at most one sort is assigned to a name in Γ and at most one object sort is assigned to each polarized subject sort, the subject sort with the polarity, in Ω. So that $\Gamma(a)$ and $\Omega(s^\star)$ denote the assigned subject sort to a and the object sort to $s^\star$, respectively. Let $dom(\Gamma)$ and $dom(\Omega)$ denote the domains of Γ and Ω, respectively. $s \in dom(\Omega)$ is an abuse notation to mean that $s^+ \in dom(\Omega)$ or $s^- \in dom(\Omega)$. Γ and Ω are often represented as sequences. $\Gamma, a : s$ denotes the subject sorting $\Gamma \cup \{a : s\}$, provided that $a \notin dom(\Gamma)$. We apply the same notational convention to Ω.

Definition 3.2 S is a *sorting system* (*by name matching*) consisting of the following inference rules:

$$\text{S-Nil}\ \frac{}{\Gamma; \Omega \vdash \mathbf{0} : ()} \qquad \text{S-Comp}\ \frac{\Gamma; \Omega \vdash P : () \quad \Gamma; \Omega \vdash Q : ()}{\Gamma; \Omega \vdash P \mid Q : ()}\ .$$

$$\text{S-In}\ \frac{\Gamma, a : s, \tilde{x} : \tilde{t};\ \Omega, s^+ : (\tilde{t}) \vdash P : ()}{\Gamma, a : s;\ \Omega, s^+ : (\tilde{t}) \vdash a(\tilde{x}).P : ()} \qquad \text{S-Out}\ \frac{\Gamma, a : s, \tilde{b} : \tilde{t};\ \Omega, s^- : (\tilde{t}) \vdash P : ()}{\Gamma, a : s, \tilde{b} : \tilde{t};\ \Omega, s^- : (\tilde{t}) \vdash a\langle\tilde{b}\rangle.P : ()}$$

$$\text{S-Repl}\ \frac{\Gamma; \Omega \vdash P : ()}{\Gamma; \Omega \vdash !P : ()} \qquad \text{S-Rest}\ \frac{\Gamma, x : s;\ \Omega \vdash P : ()}{\Gamma; \Omega \vdash (\nu x)P : ()}\ .$$

□

The interesting cases are the rules for input and output. In order to be sure that the input prefix $a(\tilde{x}).P$ is well-behaved in a given sorting, we must check that first the object

sort for the subject sort of a with the positive polarity matches the sort of the sequence of names read from a; secondly the continuation P is well-behaved in the augmented sorting by the sort assignments $\tilde{x} : \tilde{t}$. The case for the output prefix is analogous. The notation $\Gamma; \Omega \vdash_S P : ()$ indicates that the sorting judgment $\Gamma; \Omega \vdash P : ()$ is provable in the system S.

It is easy to take the correspondence between our sorting system and the typing system by Liu and Walker [7]. Ω represents a sorting signature consisting of the set Σ of sorts and the input, output sortings $ob^+, ob^- : \Sigma \rightharpoonup \wp(\Sigma^*)$ such that at most one input and one output object sorts are assigned to a subject sort though multiple object sorts assignments are allowed in the typing system [7]; Γ represents a partial function $\phi : \mathcal{N} \rightharpoonup \Sigma$ of sort assignments to names. Taking account of these correspondences, the inference rules are essentially same as the ones in [7]. In fact, we have the following proposition by induction on proofs.

Proposition 3.1 *Let $\Gamma; \Omega$ be a sorting and P a process. $\Gamma; \Omega \vdash_S P : ()$ iff P can be proved to possess the type $\langle \Omega, \Gamma \rangle$ in the type system* [7]. □

Definition 3.3 (Due to [7] though slight modifications are made.)

1. Let Ω_1 and Ω_2 be object sortings on Σ. A *homomorphism* from Ω_1 to Ω_2 is a function $\theta : \Sigma \to \Sigma$ such that if $s^\star : (\tilde{t}) \in \Omega_1$ then $\theta(s)^\star : (\theta(\tilde{t})) \in \Omega_2$, for $\star \in \{+, -\}$[3].
2. Let $\Gamma_1; \Omega_1$ and $\Gamma_2; \Omega_2$ be sortings and θ a homomorphism from Ω_1 to Ω_2. We write $\Gamma_1; \Omega_1 \sqsubseteq_\theta \Gamma_2; \Omega_2$ if $x : s \in \Gamma_1$ implies $x : \theta(s) \in \Gamma_2$ for all x and s. We write $\Gamma_1; \Omega_1 \sqsubseteq \Gamma_2; \Omega_2$ iff $\Gamma_1; \Omega_1 \sqsubseteq_\theta \Gamma_2; \Omega_2$ for some θ and $\Gamma_2; \Omega_2$ is called a *specialization* of $\Gamma_1; \Omega_1$.
3. An object sorting Ω is *self-consistent* if for every $s \in \Sigma$, whenever $s^+ : (s_1, \ldots, s_n)$, $s^- : (t_1, \ldots, t_m) \in \Omega$ then $n = m$ and there exists a homomorphism θ from Ω to Ω such that $\theta(s_i) = \theta(t_i)$, for $1 \leq i \leq n$.
4. A sorting $\Gamma; \Omega$ is *consistent* if there exists a self consistent sorting $\Gamma_0; \Omega_0$ such that $\Gamma; \Omega \sqsubseteq \Gamma_0; \Omega_0$. □

Proposition 3.2

1. *If $\Gamma_1; \Omega_1 \sqsubseteq \Gamma_2; \Omega_2$ and $\Gamma_1; \Omega_1 \vdash_S P : ()$ then $\Gamma_2; \Omega_2 \vdash_S P : ()$.*
2. *Any safe sorting $\Gamma; \Omega$ is a consistent sorting. Conversely, if $\Gamma; \Omega$ is a consistent sorting then there exists a safe sorting $\Gamma_0; \Omega_0$ such that $\Gamma; \Omega \sqsubseteq \Gamma_0; \Omega_0$.*

Proof: *Proof of* 1. By Proposition 3.1 and Lemma 8 in [7].

Proof of 2. It is straightforward from definition that any safe sorting is a consistent sorting. Now, suppose $\Gamma; \Omega$ is a consistent sorting on Σ. Then there is a self-consistent sorting $\Gamma_1; \Omega_1$ such that $\Gamma; \Omega \sqsubseteq_\theta \Gamma_1; \Omega_1$ for some homomorphism θ from Ω to Ω_1. If there is a sort s in Σ such that $s^+ : (\tilde{t}), s^- : (\tilde{u}) \in \Omega_1$ for some $\tilde{t}$ and $\tilde{u}$ with $\tilde{t} \neq \tilde{u}$ (note that $|\tilde{t}| = |\tilde{u}|$), then there exists a homomorphism θ_1 from Ω_1 to Ω_1 such that

[3]We use a total function rather than a partial function. See [7].

$\theta_1(\tilde{t}) = \theta_1(\tilde{u})$. If we let $\Gamma_2 = \theta_1(\Gamma_1)$ and $\Omega_2 = \theta_1(\Omega_1)$, then we have $\Gamma_1; \Omega_1 \sqsubseteq_{\theta_1} \Gamma_2; \Omega_2$. Thus, $\Gamma; \Omega \sqsubseteq_{\theta_1\theta} \Gamma_2; \Omega_2$. We repeat this process $n-1$ times for some n until there is no s in Σ such that $s^+ : (\tilde{t}), s^- : (\tilde{u}) \in \Omega_n$ for any $\tilde{t}$ and $\tilde{u}$ with $\tilde{t} \neq \tilde{u}$. By construction, $\Gamma_n; \Omega_n$ is a safe sorting and $\Gamma; \Omega \sqsubseteq \Gamma_n; \Omega_n$. □

Any safe object sorting Ω induces a consistent partition of Ω, see [7] for the definition of a *consistent partition* of a sorting signature. Conversely, the safe object sorting is derived from a consistent partition. Therefore, Proposition 3.2.2 corresponds to Theorem 14 in [7]. The next corollary is a direct consequence of this proposition.

Corollary 3.1 *Let P be a process. P has a safe sorting on Σ iff P has a consistent sorting on Σ, i.e. $\Gamma; \Omega \vdash_S P : ()$ for a safe sorting $\Gamma; \Omega$ iff $\Gamma'; \Omega' \vdash_S P : ()$ for a consistent sorting $\Gamma'; \Omega'$.* □

Corollary 3.2 *If a sorting $\Gamma; \Omega$ has a safe specialization, i.e. $\Gamma; \Omega \sqsubseteq \Gamma'; \Omega'$ for some safe sorting $\Gamma'; \Omega'$, then $\Gamma; \Omega$ has a most general safe sorting $\Gamma_0; \Omega_0$ in the sense that*

1. *$\Gamma; \Omega \sqsubseteq \Gamma_0; \Omega_0$ and $\Gamma_0; \Omega_0$ is safe;*
2. *$\Gamma; \Omega \sqsubseteq \Gamma'; \Omega'$ for some safe sorting $\Gamma'; \Omega'$ implies $\Gamma_0; \Omega_0 \sqsubseteq \Gamma'; \Omega'$,*

and $\Gamma_0; \Omega_0$ is unique up to isomorphism[4]

Proof: By the assumption, $\Gamma; \Omega$ is consistent. The construction of a safe sorting from a consistent sorting in the proof of Proposition 3.2 gives a required most general safe sorting. The uniqueness is obvious from the construction. □

Proposition 3.3 *If $\Gamma; \Omega \vdash_S P : ()$ and $P \equiv Q$ then $\Gamma; \Omega \vdash_S Q : ()$.* □

Proposition 3.4 *If $\Gamma; \Omega \vdash_S P : ()$ for a safe sorting $\Gamma; \Omega$ and $P \to Q$ then $\Gamma; \Omega \vdash_S Q : ()$.*

Proof: By induction on the proof of the reduction $P \to Q$. □

In the inference rule COMM of the reduction relation, it is required that the arities of the input-prefix and the output prefix must be equal. If a process P contains *unguarded* prefixes $a(\tilde{x}).Q$ and $\bar{a}\langle\tilde{b}\rangle.R$ with $|\tilde{x}| \neq |\tilde{b}|$, then P is said to *contain a communication mismatch* [7] or a *run-time type error* [12, 16]. P is *free from communication mismatch* if whenever $P \xrightarrow{*} P'$ then P' does not contain a communication mismatch. Thus, by Corollary 3.1 and Corollary 12 in [7] we can conclude that if a process has a safe sorting then P is free from communication mismatch.

[4]Let Ω_1 and Ω_2 be sortings on Σ. An *isomorphism* form Ω_1 to Ω_2 is a bijective homomorphism $\theta : \Omega_1 \to \Omega_2$ such that its inverse $\theta^{-1} : \Omega_2 \to \Omega_1$ is also a homomorphism.

4 Typing Systems by Subtyping

In this section we introduce a typing system based on subtyping by Pierce and Sangiorgi [12], which is a slight variant/extension of the one by Pierce and Sangiorgi [12] with the constant types $\top$ (*top*) and $\bot$ (*bottom*) are added as the universal type and the inconsistent type, respectively. Some basic preliminaries are stated as well for later discussions.

Let $I \leq J$ be the least preorder on the *tags* $\{\mathsf{r}, \mathsf{w}, \mathsf{b}\}$ containing $\mathsf{b} \leq \mathsf{r}$ and $\mathsf{b} \leq \mathsf{w}$. A *type*, ranged over by T or S, is defined by the grammar:

$$\begin{array}{lcl} T & ::= & \alpha \mid \top \mid \bot \mid (T_1, \ldots, T_n)^I \mid \mu\alpha.S \\ I & ::= & \mathsf{r} \mid \mathsf{w} \mid \mathsf{b}, \end{array}$$

where α is a *type-variables*; $\top$ and $\bot$ are *constant types top* and *bottom*, respectively; $(T_1, \ldots, T_n)^I$ is a *tagged tuple*; $\mu\alpha.S$ is a *recursive type*. Let $\boldsymbol{T}$ and $\boldsymbol{T}$ denote the set of all (open) types and the set of all closed types, respectively, where α-convergent types are identified. The identification over types will be justified by the equality over types defined below. A type is called *finite* if it contains no recursive types as subterms. The symbols t and s range over finite types. A type T is *contractive* in a type variable α if every free occurrence of α in T is within some tagged tuple $(T_1, \ldots, T_n)^I$. An I/O-*tree* is a finitely branching tree whose nodes are labeled with the *labels* — tags in $\{\mathsf{r}, \mathsf{w}, \mathsf{b}\}$, type variables, or constants $\top$, $\bot$ (cf. [12]). We can identify an I/O-tree with a partial function $\mathcal{T}$ from the tree domain N_+^* — the set of all finite sequences of non-zero natural numbers — to the set of labels [3,4]. $[\mathcal{T}_1, \ldots, \mathcal{T}_n]^I$ denotes the tree whose root is labeled with I and whose subtrees are $\mathcal{T}_1, \ldots, \mathcal{T}_n$, where I is a tag. With each type T we associate the I/O-tree $Tree(T)$; it is the unique tree satisfying the following equations:

$$\begin{array}{rcl} Tree(\alpha) & = & \alpha; \\ Tree(\top) & = & \top; \\ Tree(\bot) & = & \bot; \\ Tree((T_1, \ldots, T_n)^I) & = & [Tree(T_1), \ldots, Tree(T_n)]^I; \\ Tree(\mu\alpha.T) & = & \begin{cases} Tree(T\{\mu\alpha.T/\alpha\}) & \text{if } T \text{ is contractive in } \alpha, \\ \bot & \text{otherwise.} \end{cases} \end{array}$$

The *equality of types* is defined by $S =_t T$ iff $Tree(S) = Tree(T)$. This equality justifies the identification of α-convergent types in the sense that $S \equiv_\alpha T$ implies $S =_t T$. Furthermore, we have $\mu\alpha.T =_t T\{\mu\alpha.T/\alpha\}$. It is easy to see the equality $=_t$ is a congruence relation on types. For every type T, $Tree(T)$ is a *regular tree*, a tree with a finite number of different subtrees. Every tree is completely specified by the language of its occurrences of the labels, which is a regular language [3]. It follows that for given types S and T the decision problem of the identity of types, $S =_t T$, is reducible to the equivalence problem of deterministic finite-state automata, thus is decidable.

To simplify the case analysis in the following proofs we introduce canonical forms for types. A *type in canonical form* T is defined by the grammar:

$$T \quad ::= \quad \alpha \mid \top \mid \bot \mid (T_1 \ldots, T_n)^I \mid \mu\alpha.(T_1 \ldots, T_n)^I,$$

where in the case $\mu\alpha.(T_1,\ldots,T_n)^I$, α must occur free in the body $(T_1,\ldots,T_n)^I$. Hence, the body of a μ in canonical form must immediately start with a tugged tuple. The following two results are inspired by the ones on type equivalence for a recursive types of the typed λ-calculus [1] and can be proved in a similar way.

Lemma 4.1 *The following equalities hold on types with respect to type equality.*

1. $\mu\alpha.\alpha = \bot$
2. $\mu\alpha.T = T\{\mu\alpha.T/\alpha\}$
3. *If T is contractive in α then $T\{S/\alpha\} = S$ and $T\{S'/\alpha\} = S'$ implies $S = S'$.*
4. $\mu\alpha.T = \mu\alpha T\{T/\alpha\}$
5. $\mu\alpha.\mu\beta.T = \mu\gamma.T\{\gamma/\alpha,\gamma/\beta\}$. □

Proposition 4.1 *For any type T there is a type S in canonical form such that $T =_t S$.* □

By this proposition, in the remainder of this paper, unless specified otherwise, a type will always mean a type in canonical form.

Let Λ be a sequence of pairs of types $S \leq T$. A *subtyping judgment* is an expression of the form $\Lambda \vdash S \leq T$, pronounced as S is a *subtype* of T under the assumption Λ.

Definition 4.1 A is a *subtyping system* consisting of the following rules[5]:

$$\textsc{Asmp}\ \frac{}{\Lambda, S \leq T \vdash S \leq T} \qquad \textsc{Ref}\ \frac{}{\Lambda \vdash T \leq T}$$

$$\textsc{Top}\ \frac{}{\Lambda \vdash S \leq \top} \qquad \textsc{Btm}\ \frac{}{\Lambda \vdash \bot \leq T}$$

$$\textsc{BB}\ \frac{\text{for each } i,\quad \Lambda \vdash S_i \leq T_i \quad \Lambda \vdash T_i \leq S_i}{\Lambda \vdash (S_1,\ldots,S_n)^{\mathsf{b}} \leq (T_1,\ldots,T_n)^{\mathsf{b}}}$$

$$\text{RB-R}\ \frac{I \leq \mathsf{r} \quad \text{for each } i,\quad \Lambda \vdash S_i \leq T_i}{\Lambda \vdash (S_1,\ldots,S_n)^I \leq (T_1,\ldots,T_n)^{\mathsf{r}}} \qquad \text{WB-W}\ \frac{I \leq \mathsf{w} \quad \text{for each } i,\quad \Lambda \vdash T_i \leq S_i}{\Lambda \vdash (S_1,\ldots,S_n)^I \leq (T_1,\ldots,T_n)^{\mathsf{w}}}$$

$$\textsc{Rec-L}\ \frac{\Lambda, \mu\alpha.S \leq T \vdash S\{\mu\alpha.S/\alpha\} \leq T}{\Lambda \vdash \mu\alpha.S \leq T} \qquad \textsc{Rec-R}\ \frac{\Lambda, S \leq \mu\alpha.T \vdash S \leq T\{\mu\alpha.T/\alpha\}}{\Lambda \vdash S \leq \mu\alpha.T}$$

□

[5]In the subtyping system regarding closed types only as in [12], the rule REF is derivable by well-founded induction on subtyping judgments. See [14]. However, it is no more derivable when open types are concerned.

In the same way as the sorting system, $\Lambda \vdash_{\mathrm{A}} S \leq T$ indicates the judgement $\Lambda \vdash S \leq T$ is provable in A. We write $S \leq_{sub} T$ when $\vdash_{\mathrm{A}} S \leq T$ and $S =_{sub} T$ when $S \leq_{sub} T$ and $T \leq_{sub} S$.

Proposition 4.2 *The subtype relation $\leq_{sub}$ is a partial order on $\boldsymbol{T}$ with the top element $\top$ and the bottom element $\bot$.* □

As a direct consequence of this proposition, $S =_{sub} T$ implies $S =_t T$. The other inclusion can be proved as follows. On the one hand, along the same line as [1], $=_t$ restricted on types in canonical form can be proved to coincide with the least congruent relation with respect to the type constructors satisfying the properties Lemma 4.1.2 and Lemma 4.1.3. On the other hand, $=_{sub}$ can be proved to be the congruence relation satisfying the same properties in Lemma 4.1. Thus, $S =_t T$ implies $S =_{sub} T$. So we use the symbol $=_t$ to denote the provable identification instead of $=_{sub}$.

Proposition 4.3 $\boldsymbol{T}$ *is a lattice with the meet $\wedge$ and the join $\vee$ satisfying, for instance, the following equalities (We will drop the dual equalities and quite obvious equalities regarding $\top$, $\bot$, and type variables α):*

1. $(S_1, \ldots, S_n)^I \wedge (T_1, \ldots, T_m)^J = \bot \ \ (n \neq m)$.
2. $(S_1, \ldots, S_n)^I \wedge (S_1, \ldots, S_n)^J = (S_1, \ldots, S_n)^{I \wedge I}$.
3. $(S_1, \ldots, S_n)^{\mathsf{r}} \wedge (T_1, \ldots, T_n)^{\mathsf{r}} = (S_1 \wedge T_1, \ldots, S_n \wedge T_n)^{\mathsf{r}}$.
4. $(S_1, \ldots, S_n)^{\mathsf{w}} \wedge (T_1, \ldots, T_n)^{\mathsf{w}} = (S_1 \vee T_1, \ldots, S_n \vee T_n)^{\mathsf{w}}$.
5. $(S_1, \ldots, S_n)^{\mathsf{b}} \wedge (S_1, \ldots, S_n)^{\mathsf{b}} = \begin{cases} (S_1, \ldots, S_n)^{\mathsf{b}} & \text{if } S_i = T_i, \text{ for } 1 \leq i \leq n \\ \bot & \text{otherwise.} \end{cases}$

The meet on the set $\{\mathsf{r}, \mathsf{w}, \mathsf{b}\}$ is defined by $I \wedge J = I$ if $I = J$ and $I \wedge J = \mathsf{b}$ otherwise. □

Definition 4.2 A *typing judgment* (*by subsorting*) is an expression of the form $\Delta \vdash P : \mathsf{o}$, where Δ is a set of *type assignments* $a : T$, P is a process, and o is the special symbol standing for well-behavedness of the process. □

Definition 4.3 T is a *typing system* (*by subtyping*) consisting of the following rules:

$$\text{T-Nil}\ \frac{}{\Delta \vdash \mathbf{0} : \mathsf{o}} \qquad \text{T-Comp}\ \frac{\Delta \vdash P : \mathsf{o} \quad \Delta \vdash Q : \mathsf{o}}{\Delta \vdash P \mid Q : \mathsf{o}}$$

$$\text{T-In}\ \frac{\vdash \Delta(a) \leq (\tilde{T})^{\mathsf{r}} \quad \Delta, \tilde{x} : \tilde{T} \vdash P : \mathsf{o}}{\Delta \vdash a(\tilde{x}).P : \mathsf{o}} \qquad \text{T-Out}\ \frac{\vdash \Delta(a) \leq (\Delta(\tilde{b}))^{\mathsf{w}} \quad \Delta \vdash P : \mathsf{o}}{\Delta \vdash \bar{a}\langle \tilde{b} \rangle.P : \mathsf{o}}$$

$$\text{T-Repl}\ \frac{\Delta \vdash P : \mathsf{o}}{\Delta \vdash !P : \mathsf{o}} \qquad \text{T-Rest}\ \frac{\Delta, x : T \vdash P : \mathsf{o}}{\Delta \vdash (\nu x)P : \mathsf{o}}.$$

□

Pierce and Sangiorgi [12] have formulated their typing system in the Church style (*á* a Church), where typing information for the input parameters and restricted names are given explicitly. As there is often a simple relationship between the two styles in the typed λ-calculi [2] there is a simple relationship between the type system in this paper and the one by Pierce and Sangiorgi [12]. This will be explained below: Let $|\ |$ be the function mapping process terms with type ornamentations into the ordinary processes in this paper by erasing the all type information.

Proposition 4.4

1. *Let Q be a process with type annotations. If $\Delta \vdash Q : \mathsf{o}$ is provable in the Pierce and Sangiorgi's Church style typing system* [12]*, then $\Delta \vdash_{\mathrm{T}} |Q| : \mathsf{o}$.*
2. *Let P be a process. If $\Delta \vdash_{\mathrm{T}} P : \mathsf{o}$ then there is a process Q with type annotations such that $\Delta \vdash Q : \mathsf{o}$ is provable in the Pierce and Sangiorgi's system and $|Q| = P$.*

Proof: 1. By induction on the proof of the judgment $\Delta \vdash Q : \mathsf{o}$.
2. Type annotations can be found from the proof $\Delta \vdash_{\mathrm{T}} P : \mathsf{o}$. The proof is by induction on the proof $\Delta \vdash_{\mathrm{T}} P : \mathsf{o}$. For instance, suppose

$$\frac{\vdash \Delta(a) \leq (T_1, \ldots, T_n)^{\mathsf{r}} \quad \Delta, x_1 : T_1, \ldots, x_n : T_n \vdash P : \mathsf{o}}{\Delta \vdash a(x_1, \ldots, x_n).P : \mathsf{o}}$$

is the last inference. The annotated process is given as $a(x_1 : T_1, \ldots, x_n : T_n).P'$, where P' is the annotated process corresponding to P obtained by the induction hypothesis. Suppose

$$\frac{\Delta, x : T \vdash P : \mathsf{o}}{\Delta \vdash (\boldsymbol{\nu} x)P : \mathsf{o}}$$

is the last inference. The annotated process is given as $(\boldsymbol{\nu} x : T)P'$, where P' is the annotated version of P. □

5 Relating Sortings and Typings

5.1 From Sortings to Typings

With each sorting judgment $\Gamma; \Omega \vdash P : ()$ we will associate a typing judgment $[\![\Gamma]\!]_\Omega \vdash P : \mathsf{o}$ such that hopefully we expect $\Gamma; \Omega \vdash_{\mathrm{S}} P : ()$ iff $[\![\Gamma]\!]_\Omega \vdash_{\mathrm{T}} P : \mathsf{o}$. For this purpose, given an object sorting Ω and an environment $\rho : \Sigma \to \boldsymbol{T}$, mapping (free) sorts to closed types, for each sort s in Σ the corresponding type $[\![s]\!]^\rho_\Omega$ of s with respect to Ω and ρ is defined as follows:

$$[\![s]\!]^\rho_\Omega \stackrel{def}{=} \mathrm{S}^\rho_\Omega(s, \emptyset);$$

$$\mathrm{S}^\rho_\Omega(s, X) \stackrel{def}{=} \begin{cases} s & \text{if } s \in X \\ \mu s.(\mathrm{T}^\rho_\Omega(s^+, X \cup \{s\}) \wedge \mathrm{T}^\rho_\Omega(s^-, X \cup \{s\})) & \text{otherwise;} \end{cases}$$

$$\mathrm{T}^\rho_\Omega(s^\star, X) \stackrel{def}{=} \begin{cases} (\mathrm{S}^\rho_\Omega(\tilde{t}, X))^{\mathsf{r}} & \text{if } \star = + \text{ and } s^+ : (\tilde{t}) \in \Omega \\ (\mathrm{S}^\rho_\Omega(\tilde{t}, X))^{\mathsf{w}} & \text{if } \star = - \text{ and } s^- : (\tilde{t}) \in \Omega \\ \rho(s) & \text{otherwise.} \end{cases}$$

In the definition we use the notational convention $\mathrm{S}^{\rho}_{\Omega}(\tilde{t}, X)$ to denote the sequence $\mathrm{S}^{\rho}_{\Omega}(t_1, X)$, $\ldots, \mathrm{S}^{\rho}_{\Omega}(t_n, X)$, for $\tilde{t} = t_1, \ldots, t_n$.

Let $\Gamma; \Omega$ be a sorting then the corresponding typing $[\![\Gamma]\!]^{\rho}_{\Omega}$ is defined by

$$[\![\Gamma]\!]^{\rho}_{\Omega} \stackrel{def}{=} \{a : [\![s]\!]^{\rho}_{\Omega} \mid a : s \in \Gamma\}.$$

Usually, the environment ρ_{T}, $\rho_{\mathsf{T}}(s) \stackrel{def}{=} \top$ for each $s \in \Sigma$, is used to assign types to sorts. However, almost results stated in this section hold for any environment ρ. So that $[\![s]\!]_{\Omega}$ and $[\![\Gamma]\!]_{\Omega}$ are the abbreviations of $[\![s]\!]^{\rho}_{\Omega}$ and $[\![\Gamma]\!]^{\rho}_{\Omega}$ for any environment ρ, respectively, when ρ is not very important. Note that $[\![s]\!]_{\Omega} =_t \bot$ if Ω possesses object assignments to s having mismatch in number with the I/O parameters.

Lemma 5.1 *Let Ω be a safe object sorting and ρ be an environment. If $s^+ : (\tilde{t}) \in \Omega$ ($s^- : (\tilde{t}) \in \Omega$) then*

$$\vdash [\![s]\!]^{\rho}_{\Omega} =_{sub} ([\![\tilde{t}]\!]^{\rho}_{\Omega})^I,$$

for some I such that $I \leq \mathsf{r}$ ($I \leq \mathsf{w}$). Thus, $[\![s]\!]^{\rho}_{\Omega} =_t ([\![\tilde{t}]\!]^{\rho}_{\Omega})^I$.

Proof: Suppose $s^+ : (\tilde{t}) \in \Omega$ ($s^- : (\tilde{t}) \in \Omega$). By the definition of $\mathrm{S}^{\rho}_{\Omega}(s, X)$, the safety property of Ω implies that $[\![s]\!]^{\rho}_{\Omega}$ can be expressed as $[\![s]\!]^{\rho}_{\Omega} = \mu s.(\mathrm{S}^{\rho}_{\Omega}(\tilde{t}, \{s\}))^I$, for some I, where $I \leq \mathsf{r}$ ($I \leq \mathsf{w}$). Since unfolding of the recursive definition preserves the identity, see Corollary 2.4.6 in [12],

$$\vdash [\![s]\!]^{\rho}_{\Omega} =_{sub} (\mathrm{S}^{\rho}_{\Omega}(\tilde{t}, \{s\})\{[\![s]\!]^{\rho}_{\Omega}/s\})^I.$$

Let t_i be the i-th element in the sequence $\tilde{t}$. If $t_i = s$ then $\mathrm{S}^{\rho}_{\Omega}(t_i, \{s\})\{[\![s]\!]^{\rho}_{\Omega}/s\} = [\![s]\!]^{\rho}_{\Omega} = [\![t_i]\!]^{\rho}_{\Omega}$. If $t_i \neq s$ then $\mathrm{S}^{\rho}_{\Omega}(t_i, \{s\})\{[\![s]\!]^{\rho}_{\Omega}/s\} = [\![t_i]\!]^{\rho}_{\Omega}$. □

Theorem 5.1 *If $\Gamma; \Omega \vdash_{\mathrm{S}} P : ()$ for a safe sorting $\Gamma; \Omega$ then $[\![\Gamma]\!]_{\Omega} \vdash_{\mathrm{T}} P : \mathsf{o}$.*

Proof: By induction on the proof of $\Gamma; \Omega \vdash P : ()$ in S. Interesting case is the one when the last inference is by S-IN or S-OUT.

Case S-IN: Suppose

$$\frac{\Gamma, a : s, \tilde{x} : \tilde{t}; \Omega, s^+ : (\tilde{t}) \vdash P : ()}{\Gamma, a : s; \Omega, s^+ : (\tilde{t}) \vdash a(\tilde{x}).P : ()}$$

is the last inference by applying S-IN. Let $\Omega' = \Omega \cup \{s^+ : (\tilde{t})\}$. By the induction hypothesis, $[\![\Gamma]\!]_{\Omega'}, a : [\![s]\!]_{\Omega'}, \tilde{x} : [\![\tilde{t}]\!]_{\Omega'} \vdash_{\mathrm{T}} P : \mathsf{o}$. It remains to show that $\vdash [\![s]\!]_{\Omega'} \leq ([\![\tilde{t}]\!]_{\Omega'})^{\mathsf{r}}$ to deduce the typing judgment $[\![\Gamma']\!]_{\Omega'}, a : [\![s]\!]_{\Omega'} \vdash_{\mathrm{T}} a(\tilde{x}).P : \mathsf{o}$. This can be obtained by Lemma 5.1. The case by S-OUT is similar. □

The theorem insists that if a process is well-sorted with respect to a safe sorting in the system employing by-name matching, then it is well-typed as well in the system employing by-structure matching with subtyping.

Example 5.1 As an example, let us consider the process $P_1 = \bar{a}\langle a,b\rangle.\mathbf{0} \mid a(x,y).\bar{y}\langle x\rangle.\mathbf{0}$ and the sorting $\Gamma_1 = \{a : s, b : t\}; \Omega_1 = \{s^+ : (s,t), s^- : (s,t), t^- : (s)\}$ on $\Sigma_1 = \{s,t\}$. P_1 can be proved to be well-behaved in S under the assumption $\Gamma_1; \Omega_1$.

$$\dfrac{\dfrac{\Gamma_1;\Omega_1 \vdash \mathbf{0} : ()}{\Gamma_1;\Omega_1 \vdash \bar{a}\langle a,b\rangle.\mathbf{0} : ()} \quad \dfrac{\dfrac{\Gamma_1, x : s, y : t;\Omega_1 \vdash \mathbf{0} : ()}{\Gamma_1, x : s, y : t;\Omega_1 \vdash \bar{y}\langle x\rangle.\mathbf{0} : ()}}{\Gamma_1;\Omega_1 \vdash a(x,y).\bar{y}\langle x\rangle.\mathbf{0} : ()}}{\Gamma_1;\Omega_1 \vdash P_1 : ()}$$

From definition, $[\![s]\!]_{\Omega_1} = \mu s.(s, \mu t.(s)^{\mathsf{w}})^{\mathsf{b}} =_t \mu s.(s,(s)^{\mathsf{w}})^{\mathsf{b}} = S$, $[\![t]\!]_{\Omega_1} = \mu t.(\mu s.(s,t)^{\mathsf{b}})^{\mathsf{w}} = T$. Let $\Delta_1 = [\![\Gamma_1]\!]_{\Omega_1} = \{a : S, b : T\}$.

$$\dfrac{\dfrac{\vdash S \leq (S,T)^{\mathsf{w}} \quad \Delta_1 \vdash \mathbf{0} : \mathsf{o}}{\Delta_1 \vdash \bar{a}\langle a,b\rangle.\mathbf{0} : \mathsf{o}} \quad \dfrac{\vdash S \leq (S,T)^{\mathsf{r}} \quad \dfrac{\vdash T \leq (S)^{\mathsf{w}} \quad \Delta_1, x : S, y : T \vdash \mathbf{0} : \mathsf{o}}{\Delta_1, x : S, y : T \vdash \bar{y}\langle x\rangle.\mathbf{0} : \mathsf{o}}}{\Delta_1 \vdash a(x,y).\bar{y}\langle x\rangle.\mathbf{0} : \mathsf{o}}}{\Delta_1 \vdash P_1 : \mathsf{o}}$$

□

Theorem 5.1 can be extended to a consistent sorting in a straightforward way.

Corollary 5.1 *Let $\Gamma;\Omega$ be a consistent sorting. Let $\Gamma_0;\Omega_0$ be the unique most general safe sorting, its existence is guaranteed by Corollary 3.2. Then for a process P, $\Gamma;\Omega \vdash_{\mathrm{S}} P : ()$ implies $[\![\Gamma_0]\!]_{\Omega_0} \vdash_{\mathrm{T}} P : \mathsf{o}$.*

Proof: By Proposition 3.2.1 and Theorem 5.1. □

The converse of Theorem 5.1 is not true in general. The following simple counter example illustrates the fact:

Example 5.2 Let us consider the process $P_2 = \bar{a}\langle b\rangle.\mathbf{0}$ under the safe sorting $\Gamma_2 = \{a : s, b : r\}$; $\Omega_2 = \{s^- : (t), t^+ : (), r^+ : (), r^- : ()\}$ on $\Sigma_2 = \{s,t,r\}$. By the transformation, $[\![s]\!]_{\Omega_2} = (\mathsf{r})^{\mathsf{w}}$, $[\![t]\!]_{\Omega_2} = \mathsf{r}$, $[\![r]\!]_{\Omega_2} = \mathsf{b}$, and $[\![\Gamma_2]\!]_{\Omega_2} = \{a : (\mathsf{r})^{\mathsf{w}}, b : \mathsf{b}\}$. Then, trivially we have $[\![\Gamma_2]\!]_{\Omega_2} \vdash_{\mathrm{T}} P_2 : \mathsf{o}$. But, $\Gamma_2;\Omega_2 \not\vdash_{\mathrm{S}} P_2$ because $t \neq r$. □

If the transformation defined by a safe object sorting Ω from sorts into types satisfies a certain condition, then the converse of Theorem 5.1 holds. To show this fact, we need the following lemma.

Lemma 5.2 *Let $\Gamma;\Omega$ be a safe sorting on Σ.*

1. *Let $s \in \Sigma$. For any $T \in sub([\![s]\!]^{\rho\mathsf{T}}_{\Omega})$, there exists a sort $t \in \Sigma$ such that $T =_t [\![t]\!]^{\rho\mathsf{T}}_{\Omega}$.*
2. *For any sort $s \in \Sigma$, if $[\![s]\!]^{\rho\mathsf{T}}_{\Omega} = (T_1,\ldots,T_n)^I$, where $I \leq \mathsf{r}$ ($I \leq \mathsf{w}$), then $s \in dom(\Omega)$ and there exists $t_i \in \Sigma$, for each $1 \leq i \leq n$, such that*
   - 2a. *$T_i =_t [\![t_i]\!]^{\rho\mathsf{T}}_{\Omega}$, $1 \leq i \leq n$;*
   - 2b. *$s^+ : (t_1,\ldots,t_n) \in \Omega$ ($s^- : (t_1,\ldots,t_n) \in \Omega$).*
3. *For any sort $s \in \Sigma$, if $[\![s]\!]^{\rho\mathsf{T}}_{\Omega} = \top$, then $s \notin dom(\Omega)$.* □

Theorem 5.2 *Let $\Gamma;\Omega$ be a safe sorting on Σ such that $[\![s]\!]^{\rho\top}_{\Omega} \leq_{sub} [\![t]\!]^{\rho\top}_{\Omega}$ implies $s = t$, for any sorts $s,t \in \Sigma$[6]. Then, $[\![\Gamma]\!]^{\rho\top}_{\Omega} \vdash_{\mathrm{T}} P : \mathrm{o}$ implies $\Gamma;\Omega \vdash_{\mathrm{S}} P : ()$, for any process P.*

Proof: By induction on the proof $[\![\Gamma]\!]^{\rho\top}_{\Omega} \vdash_{\mathrm{T}} P : \mathrm{o}$ and by case analysis of the applied rules. For detailed proof, refer to [14]. □

5.2 From Typings to Sortings

We will define a sorting $\Delta^{@};\Delta^{\#}$ in terms of a typing Δ. To this end, we need some preliminaries. Given an open type T, let $Sub(T)$ be the set obtained from the set of all the subterms of T by replacing each bound type variable appearing in a subterm by its definition, formally $Sub(T)$ is defined inductively as follows:

$$\begin{aligned}
Sub(\alpha) &\stackrel{def}{=} \{\alpha\};\\
Sub(\top) &\stackrel{def}{=} \{\top\};\\
Sub(\bot) &\stackrel{def}{=} \{\bot\};\\
Sub((T_1,\ldots,T_n)^I) &\stackrel{def}{=} \{(T_1,\ldots,T_n)^I\} \cup Sub(T_1) \cup \cdots \cup Sub(T_n);\\
Sub(\mu\alpha.T) &\stackrel{def}{=} \{\mu\alpha.T\} \cup \{S\{\mu\alpha.T/\alpha\} \mid S \in Sub(T)\}.
\end{aligned}$$

From definition it is easy to see that $Sub(T)$ is finite for any type T. In fact, $Sub(T)$ can have no more elements than the number of distinct subterms of T.

With an open type in canonical form $T \in \boldsymbol{T}$ we associate a tuple $\langle \Sigma(T), T^{\#} \rangle$ consisting of the set $\Sigma(T)$ of sorts and the object sorting $T^{\#}$. The sorts are defined by

$$\Sigma(T) \stackrel{def}{=} \{\,[S] \mid S \in Sub(T)\}$$

for a type T, where $[T] \stackrel{def}{=} \{S \mid T =_t S\}$ is the congruence class of T with respect to the identity relation $=_t$ on $\boldsymbol{T}$. The object sorting $T^{\#}$ is defined by structural induction on T.

$$T^{\#} \stackrel{def}{=} \begin{cases} \emptyset & \text{if } T = \alpha, \text{ or } \top\\ \Omega_{\bot} & \text{if } T = \bot\\ \Omega_t \cup T_1^{\#} \cup \cdots \cup T_n^{\#} & \text{if } T = (T_1,\ldots,T_n)^I\\ \Omega_r \cup (T_1^{\#} \cup \cdots \cup T_n^{\#})\{\,[T]\,/\,[\alpha]\,\} & \text{if } T = \mu\alpha.(T_1,\ldots,T_n)^I, \end{cases}$$

where

$$\Omega_{\bot} \stackrel{def}{=} \{[\bot]^{+} : (), [\bot]^{-} : ([\bot])^{\mathsf{w}}\}$$

$$\Omega_t \stackrel{def}{=} \begin{cases} \{[T]^{+} : ([T_1],\ldots,[T_n])\} & \text{if } I = \mathsf{r}\\ \{[T]^{-} : ([T_1],\ldots,[T_n])\} & \text{if } I = \mathsf{w}\\ \{[T]^{+} : ([T_1],\ldots,[T_n]),\ [T]^{-} : ([T_1],\ldots,[T_n])\} & \text{if } I = \mathsf{b}. \end{cases}$$

[6]This condition means that Ω represents the unique object sorting up to renaming of sorts such that no distinct sorts represent the same type where the type equality by forgetting the tags is used as the identity of types. Under this condition, $[\![\Gamma]\!]^{\rho\top}_{\Omega} \vdash_{\mathrm{T}} P : \mathrm{o}$ means P is well-typed with respect to $[\![\Gamma]\!]^{\rho\top}_{\Omega}$, where *only by-structure matching without subtyping* is used.

$$\Omega_r \stackrel{def}{=} \begin{cases} \{[T]^+ : ([T_1\{T/\alpha\}], \ldots, [T_n\{T/\alpha\}])\} & \text{if } I = \mathsf{r} \\ \{[T]^- : ([T_1\{T/\alpha\}], \ldots, [T_n\{T/\alpha\}])\} & \text{if } I = \mathsf{w} \\ \{[T]^+ : ([T_1\{T/\alpha\}], \ldots, [T_n\{T/\alpha\}]), \\ \quad [T]^- : ([T_1\{T/\alpha\}], \ldots, [T_n\{T/\alpha\}])\} & \text{if } I = \mathsf{b}. \end{cases}$$

Let Δ be a typing. The corresponding set of subject sorts is defined by

$$\Sigma(\Delta) \stackrel{def}{=} \cup \{\Sigma(T) \mid x : T \in \Delta, \text{ for some } x\}.$$

The associated sorting $\Delta^{@}; \Delta^{\#}$ on $\Sigma(\Delta)$ with Δ is defined as follows:

$$\begin{aligned} \Delta^{@} &\stackrel{def}{=} \{x : [T] \mid x : T \in \Delta\}; \\ \Delta^{\#} &\stackrel{def}{=} \cup \{T^{\#} \mid x : T \in \Delta, \text{ for some } x\}. \end{aligned}$$

For notational simplicity, the square brackets are often omitted and the sort $(T_1, \ldots, T_n)^I$ is sometimes written as $I(T_1, \ldots, T_n)$ using the prefix notation. So that, e.g. the object sort assignment $\mathsf{b}(T_1, \ldots, T_n)^+ : (T_1, \ldots, T_n)$ is the abbreviation of $[(T_1, \ldots, T_n)^{\mathsf{b}}]^+ : ([T_1], \ldots, [T_n])$.

Example 5.3 Consider the typing $\Delta_3 = \{b : (\mathsf{b}, \mathsf{b})^{\mathsf{b}}\}$. The set of sorts is given by $\Sigma(\Delta_3) = \{\mathsf{b}(\mathsf{b}, \mathsf{b}), \mathsf{b}\}$. Let $s = \mathsf{b}(\mathsf{b}, \mathsf{b})$, $t = \mathsf{b}$. The corresponding sorting is obtained by $\Delta_3^{@} = \{b : s\}$; $\Delta_3^{\#} = \{s^+ : (t, t), s^- : (t, t), t^+ : (), t^- : ()\}$. It is worth while to note that $[\![s]\!]_{\Delta_3^{\#}} = (\mathsf{b}, \mathsf{b})^{\mathsf{b}}$; $[\![t]\!]_{\Delta_3^{\#}} = \mathsf{b}$ and $[\![\Delta_3^{@}]\!]_{\Delta_3^{\#}} = \Delta_3$. □

Example 5.4 As a more involved example, let us consider the typing

$$\Delta_4 = \{a : \mu\alpha.(\alpha, (\alpha)^{\mathsf{w}})^{\mathsf{b}}, b : \mu\beta.(\mu\alpha.(\alpha, \beta)^{\mathsf{b}})^{\mathsf{w}}\}.$$

Let $S = \mu\alpha.(\alpha, (\alpha)^{\mathsf{w}})^{\mathsf{b}}$; $T = \mu\beta.(\mu\alpha.(\alpha, \beta)^{\mathsf{b}})^{\mathsf{w}}$; $U = \mu\alpha.(\alpha, T)^{\mathsf{b}}$. Then we obtain the sorts by construction: S, $\mathsf{b}(S, \mathsf{w}(S))$, $\mathsf{w}(S)$, T, $\mathsf{w}(U)$, U, and $\mathsf{b}(U, T)$. Among them we have $s \stackrel{def}{=} S = \mathsf{b}(S, \mathsf{w}(S)) = U = \mathsf{b}(U, T)$; $t \stackrel{def}{=} \mathsf{w}(U) = \mathsf{w}(S)$. Thus, the set of sort is given by $\Sigma(\Delta_4) = \{s, t\}$. The sorting from Δ_4 is given by $\Delta_4^{@} = \{a : s, b : t\}$; $\Delta_4^{\#} = \{s^+ : (s, t), s^- : (s, t), t^- : (s)\}$.

Recall that Δ_4 is the resulting typing obtained from the sorting $\Gamma_1; \Omega_1$ in Example 5.1 and the derived sorting $\Delta_4^{@}; \Delta_4^{\#}$ coincides with the original sorting. Thus, $\Gamma_1; \Omega_1 = [\![\Gamma_1]\!]_{\Omega_1}^{@}; [\![\Gamma_1]\!]_{\Omega_1}^{\#}$; $\Delta_4 = [\![\Delta_4^{@}]\!]_{\Delta_4^{\#}}$, where $[\![\Gamma_1]\!]_{\Omega_1} = \Delta_4$. □

We hope that for instance $\Delta \vdash_{\mathrm{T}} P : \mathsf{o}$ implies $\Delta^{@}; \Delta^{\#} \vdash_{\mathrm{S}} P : ()$. But, unfortunately there is a simple counter example. Let us consider the context $\Delta = \{a : (\top)^{\mathsf{w}}, b : (\top)^{\mathsf{r}}\}$ and the process $P = \bar{a}\langle b \rangle.\mathbf{0}$. P is well-typed under the context Δ.

$$\frac{(\top)^{\mathsf{w}} \leq ((\top)^{\mathsf{r}})^{\mathsf{w}} \quad a : (\top)^{\mathsf{w}}, b : (\top)^{\mathsf{r}} \vdash \mathbf{0} : \mathsf{o}}{\bar{a} : (\top)^{\mathsf{w}}, b : (\top)^{\mathsf{r}} \vdash a\langle b \rangle.\mathbf{0} : \mathsf{o}}$$

Thus, $\Delta \vdash_{\mathrm{T}} P : \mathsf{o}$. By definition, $\Sigma(\Delta) = \{\mathsf{w}(\top), \mathsf{r}(\top), \top\}$; $\Delta^{@} = \{a : \mathsf{w}(\top), b : \mathsf{r}(\top)\}$; $\Delta^{\#} = \{\mathsf{w}(\top)^- : (\top), \mathsf{r}(\top)^+ : (\top)\}$. Because $\top \neq \mathsf{r}(\top)$, $\Delta^{@}; \Delta^{\#} \not\vdash_{\mathrm{S}} P : ()$.

Proposition 5.1 *Let T be a type and ρ an environment, then $[\![[S]]\!]^{\rho}_{T\#} =_t [\![[S]]\!]^{\rho}_{S\#}$, for any $S \in sub(T)$.* □

Lemma 5.3 *Let σ be any function mapping type variables α to closed types $\sigma(\alpha) \in \boldsymbol{T}$ and T be any type. Define the environment $\rho : \Sigma(T) \to \boldsymbol{T}$ by*

$$\rho([S]) \stackrel{def}{=} \begin{cases} \sigma(\alpha) & \text{if } S = [\alpha] \text{ for some } \alpha \\ S & \text{otherwise,} \end{cases}$$

for $[S] \in \Sigma(T)$. Then, we have $[\![[T]]\!]^{\rho}_{T\#} =_t \sigma(T)$.

Proof: We will show $Tree([\![[T]]\!]^{\rho}_{T\#})(\pi) = Tree(\sigma(T))(\pi)$ by induction on $\pi \in \mathrm{N}^*_+$ and by case analysis on T. The interesting case arrises when $T = \mu\alpha.(T_1, \ldots, T_n)^I$. By definition,

$$\begin{aligned} \Sigma(T) &= \{[T]\} \cup \{[S\{T/\alpha\}] \mid S \in sub((T_1, \ldots, T_n)^I)\}; \\ T^{\#} &= \Omega_r \cup (T_1^{\#} \cup \cdots \cup T_n^{\#})\{[T]/[\alpha]\}. \end{aligned}$$

where

$$\Omega_r \stackrel{def}{=} \begin{cases} \{[T]^+ : ([T_1\{T/\alpha\}], \ldots, [T_n\{T/\alpha\}])\} & \text{if } I = \mathsf{r} \\ \{[T]^- : ([T_1\{T/\alpha\}], \ldots, [T_n\{T/\alpha\}])\} & \text{if } I = \mathsf{w} \\ \{[T]^+ : ([T_1\{T/\alpha\}], \ldots, [T_n\{T/\alpha\}]), & \text{if } I = \mathsf{b}. \\ \quad [T]^- : ([T_1\{T/\alpha\}], \ldots, [T_n\{T/\alpha\}])\} & \end{cases}$$

Thus, by Lemma 5.1 and Proposition 5.1

$$\begin{aligned} Tree(\sigma(T)) &= [Tree(\sigma(T_1\{T/\alpha\})), \ldots, Tree(\sigma(T_n\{T/\alpha\}))]^I; \\ Tree([\![[T]]\!]^{\rho}_{T\#}) &= [Tree([\![[T_1\{T/\alpha\}]]\!]^{\rho}_{T\#}), \ldots, Tree([\![[T_n\{T/\alpha\}]]\!]^{\rho}_{T\#})]^I \\ &= [Tree([\![[T_1\{T/\alpha\}]]\!]^{\rho}_{T_1\{T/\alpha\}\#}), \ldots, Tree([\![[T_n\{T/\alpha\}]]\!]^{\rho}_{T_n\{T/\alpha\}\#})]^I. \end{aligned}$$

Obviously

$$Tree([\![[T]]\!]^{\rho}_{T\#})(\varepsilon) = Tree(\sigma(T))(\varepsilon).$$

Let $k\pi$ be a current path. If $k \leq n$ then

$$Tree([\![[T]]\!]^{\rho}_{T\#})(k\pi) = Tree(\sigma(T))(k\pi),$$

by the induction hypothesis

$$Tree([\![[T_i]]\!]^{\rho}_{T_i^{\#}})(\pi) = Tree(\sigma(T_i))(\pi),$$

for each i, $1 \leq i \leq n$. If $k > n$ then both the trees are undefined on $k\pi$.

Theorem 5.3

1. *$\Gamma; \Omega \sqsubseteq [\![\Gamma]\!]^{@}_{\Omega}; [\![\Gamma]\!]^{\#}_{\Omega}$, for any safe sorting $\Gamma; \Omega$ on Σ.*
2. *$\Gamma; \Omega \neq [\![\Gamma]\!]^{@}_{\Omega}; [\![\Gamma]\!]^{\#}_{\Omega}$, for some safe sorting $\Gamma; \Omega$ on Σ.*
3. *$\Delta = [\![\Delta^{@}]\!]_{\Delta\#}$, for any typing Δ.*

Proof: 1. Let $\theta : \Sigma \to \Sigma([\![\Gamma]\!]_\Omega)$ be the function defined by $\theta(s) \stackrel{def}{=} [[\![s]\!]_\Omega]$, for $s \in \Sigma$. Suppose $x : s \in \Gamma$ then $x : [[\![s]\!]_\Omega] \in [\![\Gamma]\!]^{@}_\Omega$ by definition. It remains to show that θ is a homomorphism from Ω to $[\![\Gamma]\!]^{\#}_\Omega$. Suppose $s^\star : (t_1, \ldots, t_n) \in \Omega$, where $\star = +$ (or $\star = -$). By Lemma 5.1, $[\![s]\!]_\Omega =_t ([\![t_1]\!]_\Omega, \ldots, [\![t_n]\!]_\Omega)^I$, where $I \leq \mathsf{r}$ (or $I \leq \mathsf{w}$). Thus by construction, we have $[[\![s]\!]_\Omega]^\star : ([[\![t_1]\!]_\Omega], \ldots, [[\![t_n]\!]_\Omega]) \in [\![\Gamma]\!]^{\#}_\Omega$, as required.

2. Consider the safe sorting $\Gamma_0 = \{a : t, b : s\}$; $\Omega_0 = \{t^+ : (), s^+ : ()\}$ on $\{s, t\}$. The inequality is obvious from the followings: $[\![\Gamma_0]\!]_{\Omega_0} = \{a : \mathsf{r}, b : \mathsf{r}\}$; $[\![\Gamma_0]\!]^{@}_{\Omega_0} = \{a : [\mathsf{r}], b : [\mathsf{r}]\}$; $[\![\Gamma_0]\!]^{\#}_{\Omega_0} = \{[\mathsf{r}]^+ : ()\}$.

3. The proof is by Lemma 5.3 since any type in Δ is closed. □

Corollary 5.2

1. *If* $\Gamma; \Omega \vdash_{\mathrm{S}} P : ()$ *for a safe sorting* $\Gamma; \Omega$ *then* $[\![\Gamma]\!]^{@}_\Omega; [\![\Gamma]\!]^{\#}_\Omega \vdash_{\mathrm{S}} P : ()$.
2. $[\![\Gamma]\!]_\Omega = [\![[\![\Gamma]\!]^{@}_\Omega]\!]_{[\Gamma]^{\#}_\Omega}$. □

6 Concluding Remarks

In this paper, the sorting system by name matching and the typing system by structure matching with subtyping were related via the transformations. The introduced sorting (typing) system is quite closed to the typing system by Liu and Walker [7] (by Pierce and Sangiorgi [12]). So the results obtained in this paper are applicable to the investigation of the correspondence between them. If we forget the polarities (the tags and subtyping), then the resulting sorting (typing) system turns out to coincide with a variant of Milner's sorting system [10] (the typing system by Vasconcelos and Honda [16]). Thus, our results interpret the relationship between both the systems as well.

The correspondence between Milner's sorting and the typing system [16] is informally discussed with the illustrative example in [16] and more formally discussed in [16]. The idea is that a set of basic sorts and sorting defines a regular system of equations; such a system has a unique solution whose components are represented as regular trees; then derive a typing from the solution. Conversely, trees in a finite set of regular trees are components of the unique solution of a single system of equation [3]; from such a system the set of sorts and sorting are obtained. But, the preliminary theorem Theorem 5.1.3 in [15] is erroneous. Because distinct sorts may correspond to the same type. Refer to Example 5.2.

The transformation from sortings to typings has a similar flavor to the one from regular system equations in canonical form to recursive types discussed in [1,14]. To make clear the correspondence between the two transformations, we derive a regular system of equations from an object sorting. Let Ω be a safe object sorting on Σ. For $s \in \Sigma$, the finite type $\langle s \rangle_\Omega$ with subject sorts taken as type variables is defined by

$$\langle s \rangle_\Omega \stackrel{def}{=} \begin{cases} (\tilde{t})^{\mathsf{b}} & \text{if } s^+ : (\tilde{t}), s^- : (\tilde{t}) \in \Omega \text{ for some } (\tilde{t}) \\ (\tilde{t})^{\mathsf{r}} & \text{if } s^+ : (\tilde{t}) \in \Omega \text{ for some } (\tilde{t}), \text{ and } s^- \notin dom(\Omega) \\ (\tilde{t})^{\mathsf{w}} & \text{if } s^- : (\tilde{t}) \in \Omega \text{ for some } (\tilde{t}), \text{ and } s^+ \notin dom(\Omega) \\ \top & \text{if } s \notin dom(\Omega). \end{cases}$$

The system $E(\Omega)$ of equations is obtained from Ω by

$$E(\Omega) \stackrel{def}{=} \{s = \langle s \rangle_\Omega \mid s \in \Sigma\}.$$

Proposition 6.1 ([14]) *Let Ω be a safe object sorting on Σ. For any sort s in Σ, $[\![s]\!]_\Omega =_t [\![\langle s, E(\Omega)\rangle]\!]$, where $[\![\langle s, E(\Omega)\rangle]\!]$ is the type represented by the type variable s w.r.t. the system of equations $E(\Omega)$. (Note that $Tree\langle s, E(\Omega)\rangle = Tree[\![\langle s, E(\Omega)\rangle]\!]$ this means that s w.r.t. $E(\Omega)$ represents the same tree as the type $[\![\langle s, E(\Omega)\rangle]\!]$).* □

Conversely, given a regular system E of equations in canonical form we define the corresponding object sorting $\Omega(E)$ with type variables appearing in E and two constants $\top$, $\bot$ taken as subjects sorts.

$$\begin{aligned}\Omega(E) \stackrel{def}{=} & \{\alpha^+ : (\tilde{\beta}) \mid \alpha = (\tilde{\beta})^{\mathsf{b}} \in E\} \cup \{\alpha^- : (\tilde{\beta}) \mid \alpha = (\tilde{\beta})^{\mathsf{b}} \in E\} \\ & \cup \{\alpha^+ : (\tilde{\beta}) \mid \alpha = (\tilde{\beta})^{\mathsf{r}} \in E\} \cup \{\alpha^- : (\tilde{\beta}) \mid \alpha = (\tilde{\beta})^{\mathsf{w}} \in E\} \cup \Omega_\bot(E),\end{aligned}$$

where

$$\Omega_\bot(E) \stackrel{def}{=} \begin{cases} \{\bot^+ : (), \bot^- : (\bot)^{\mathsf{w}}\} & \text{if } E \text{ contains } \bot \\ \emptyset & \text{otherwise.} \end{cases}$$

Proposition 6.2 ([14]) *Let E be a regular system of equations in canonical form and p be a type variable, $\top$, or $\bot$, then $[\![p]\!]_{\Omega(E)} =_t [\![\langle p, E\rangle]\!]$.* □

From typings without subtyping to Milner's sortings, as stated in [16], well-typing induces well-sorting. But, as illustrated in Section 5.2, in general well-typing doesn't always implies well-sorting along the given translation. But, we convince that the following conjecture must hold.

Conjecture 6.1 *If $\Delta \vdash_{\mathrm{T}} P : \circ$ then there exists a typing Δ_0 such that $\Delta_0 \preceq \Delta$ — $dom(\Delta_0) \supset dom(\Delta)$ and $\Delta_0(x) \leq \Delta(x)$ for all $x \in dom(\Delta)$ — and $\Delta_0^{@}; \Delta_0^{\#} \vdash_{\mathrm{S}} P : ()$. Note that $\Delta_0 \vdash_{\mathrm{T}} P : \circ$. See* [12, 14]. □

Finally, the relations between incremental systems and non-incremental systems are discussed in both sorting and typing in [14].

References

[1] Amadio, R.M., Cardelli, L., Subtyping recursive types, *TOPLAS*, **15**, No. 4, pp.575–631, 1993.

[2] Barendregt, H.P., Lambda calculi with types, in: *Handbook of Logic in Computer Science*, Volume 2, Background: Computational Structures, Edts. S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, Oxford Science Publications, 1992.

[3] Courcelle, B., Fundamental properties of infinite trees, *TCS*, 25, pp.95–169, 1983.

[4] G., Cousineau, M., Nivar., On rational expressions representing infinite rational trees: *8th MFCS*, *LNCS*, **74**, pp.567–580, 1979.

[5] Hindley, R., The completeness theorem for typing λ-terms, *TCS*, 22, pp.1–7, pp.127–133, 1983.

[6] Gay, S.J., A sort inference algorithm for the polyadic π-calculus, in 20th *POPL*, 1993.

[7] Liu, X., Walker, D., A polymorphic type system for the polyadic π-calculus, *CONCUR'95*, *LNCS*, **962**, pp.103–116, 1995.

[8] Milner, R., *Communication and Concurrency*, Prentice Hall, 1989.

[9] Milner, R., Functions as processes, *th ICALP* '**90**, *LNCS*, **443**, 1990.

[10] Milner, R., The polyadic π-calculus: A tutorial, Technical Report ECS-LFCS-91-180, LFCS, Dept. of Comput. Sci., Edinburgh Univ., 1991.

[11] Milner, R., Parrow, J., Walker, D., A calculus of mobile processes, Part I and II, *Info. & Comp.*, **100**, No.1, pp.1–40, pp.41–47, 1992.

[12] B., Pierce, D. Sangiorgi, Typing and subtyping for mobile processes, Dep. of Computer Science, University of Edinburgh, 1994.

[13] Sangiorgi, D., Expressing mobility in process algebras: first-order and higher-order paradigms, Ph.D. Thesis, Edinburgh University, 1992.

[14] Togashi, A., On typing systems for the polyadic π-calculus, to appear in *Technical Report*, COGS, University of Sussex, 1996.

[15] Vasconcelos, V.T., Honda, K., Principal typing-schemes in a polyadic π-calculus, CS 92-4, Keio University, 1992.

[16] Vasconcelos, V.T., Honda, K., Principal typing schemes in a polyadic π-calculus, CONCUR'93, *LNCS*, **715**, pp.524–538, 1993.