Language Primitives and Type Discipline for Structured Communication-Based Programming Revisited: *Two Systems for Higher-Order Session Communication*

Nobuko Yoshida[1]

Imperial College London

Vasco T. Vasconcelos[2]

University of Lisbon

Abstract

Session primitives and types provide a flexible programming style for structured interaction, and are used to statically check the safe and consistent composition of protocols in communication-centric distributed software. Unfortunately authors working on session types have recently realised that some of the previously published systems fail to satisfy the basic theorems of Subject Reduction and Type Safety.
This report discusses the issues involved in higher-order session communication, presents a formulation of the recursive types as well as proofs of the Subject Reduction and Type Safety Theorems of the original session typing system by Honda-Vasconcelos-Kubo in ESOP'98. It also proposes a variant which allows a more liberal higher-order session communication, based on an idea of Gay and Hole.
Keywords: Pi-Calculus, Session Types, Protocol Analysis, Subject Reduction Theorem, Type Safety, Rewriting System

1 Introduction

Session primitives and types provide a flexible programming style for structural interaction, and are used to statically check the safe and consistent composition of protocols in communication-centric distributed software. They have been studied for the π-calculus [2,11,12,13,17,20,23], Ambients [6], CORBA interfaces [21], multi-threaded functional languages [15,23,24], Web Description Languages [3,4,14], and distributed [8] and multi-threaded Java [7] and, at the industry level, WC3-CDL [5,25] and pi4Tech [16].

[1] Email: yoshida@doc.ic.ac.uk
[2] Email: vv@di.fc.ul.pt

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

This paper reports on recent active discussions on the two fundamental theorems, Subject Reduction and Type Safety, among the authors working on session types. In the presence of higher-order session communication, session instantiation dynamically changes the structure of sessions, so that it becomes non-trivial to preserve typability. Unfortunately the aforementioned authors have recently realised that some of the previous systems fail to satisfy these basic theorems. Interestingly, the subtlety of type preservation is related to a treatment of communication channels in the rewriting rules of the π-calculus.

After discussing the issues involved in higher-order session communication, this report also proposes a session typing system which allows a more liberal higher-order session communication, based on the work of Gay and Hole [12] and already used in [23]. The full proofs of the two theorems are firstly given in this report, which also clarifies some definitions absent in [13]. The motivation to why the present authors should redo the proofs nine years after is the discovery of a subtle counterexample to the results in some works on session types published after the work under consideration, although not to results of the original system [13]. We explain the problem in detail in Section 3.

The technical contributions of this report include: the formulation of recursive types, proofs for the Subject Reduction and Type Safety Theorems in the original session typing system by Honda-Vasconcelos-Kubo [13], as well as the presentation of a more liberal system and the corresponding proofs.

The outline of the paper is simple. The next section revisits the ESOP'98 system, presenting proofs for the above mentioned results. Section 3 presents the more liberal system. Section 4 concludes the paper.

2 The Honda-Vasconcelos-Kubo Session Typing System

Honda-Vasconcelos-Kubo's session typing system in ESOP'98 [13] is an extension of the first session typing system [20] that allows higher-order session communication. We first informally review the syntax, operational semantics and typing system of the ESOP system. We then state and prove the main theorems, Subject Reduction and Type Safety. Detailed examples and explanations of the language and typing system can be found in reference [13].

2.1 Syntax and Operational Semantics

A *session* is a series of reciprocal interactions between two parties, possibly with branching and recursion, and serves as a unit of abstraction for describing interaction. Communications belonging to a session are performed via a port, specific to that session, called a *channel*. A fresh channel is generated when initiating each session, for the use in safe communications.

We use the following base sets: *names*, ranged over by $a, b, x, y, z \ldots$; *channels*, ranged over by k, k'; *constants* (including names, integers and booleans), ranged over by $c, c', \ldots$; *labels*, ranged over by $l, l', \ldots$; and *process variables*, ranged over by $X, Y, \ldots$ Letters $u, u', \ldots$ denote names and channels together. Then *processes*,

$$
\begin{array}{rll}
P & ::= & \texttt{request}\ a(k)\ \texttt{in}\ P & \text{session request} \\
& | & \texttt{accept}\ a(k)\ \texttt{in}\ P & \text{session acceptance} \\
& | & k![\tilde{e}]; P & \text{data sending} \\
& | & k?(\tilde{x})\ \texttt{in}\ P & \text{data reception} \\
& | & k \lhd l; P & \text{label selection} \\
& | & k \rhd \{l_1 : P_1 [\!] \cdots [\!] l_n : P_n\} & \text{label branching} \\
& | & \texttt{throw}\ k[k']; P & \text{channel sending} \\
& | & \texttt{catch}\ k(k')\ \texttt{in}\ P & \text{channel reception} \\
& | & \texttt{if}\ e\ \texttt{then}\ P\ \texttt{else}\ Q & \text{conditional branch} \\
& | & P \mid Q & \text{parallel composition} \\
& | & \texttt{inact} & \text{inaction} \\
& | & (\nu u)P & \text{name/channel hiding} \\
& | & \texttt{def}\ D\ \texttt{in}\ P & \text{recursion} \\
& | & X[\tilde{e}\tilde{k}] & \text{process variables} \\
e & ::= & c & \text{constant} \\
& | & e + e' \ \mid\ e - e' \ \mid\ e \times e \ \mid\ \texttt{not}(e) \ \mid\ \ldots & \text{operators} \\
D & ::= & X_1(\tilde{x}_1\tilde{k}_1) = P_1\ \texttt{and} \cdots \texttt{and}\ X_n(\tilde{x}_n\tilde{k}_n) = P_n & \text{declaration for recursion}
\end{array}
$$

Fig. 1. Syntax

ranged over by $P, Q \ldots$, and *expressions*, ranged over by $e, e', \ldots$ are given by the grammar in Figure 1. The typing system in Figure 6 makes sure that, in process $X[\tilde{e}\tilde{k}]$, the channels in $\tilde{k}$ are pairwise distinct.

$$
\begin{array}{c}
P \equiv Q \text{ if } P \equiv_\alpha Q \\
P \mid \texttt{inact} \equiv P \qquad P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R) \\
(\nu u)P \mid Q \equiv (\nu u)(P \mid Q) \quad \text{if } u \notin \mathrm{fu}(Q) \\
(\nu u)\texttt{inact} \equiv \texttt{inact} \\
\texttt{def}\ D\ \texttt{in}\ \texttt{inact} \equiv \texttt{inact} \\
(\nu u)\texttt{def}\ D\ \texttt{in}\ P \equiv \texttt{def}\ D\ \texttt{in}\ (\nu u)P \quad \text{if } u \notin \mathrm{fu}(D) \\
(\texttt{def}\ D\ \texttt{in}\ P) \mid Q \equiv \texttt{def}\ D\ \texttt{in}\ (P \mid Q) \quad \text{if } \mathrm{dpv}(D) \cap \mathrm{fpv}(Q) = \emptyset \\
\texttt{def}\ D\ \texttt{in}\ (\texttt{def}\ D'\ \texttt{in}\ P) \equiv \texttt{def}\ D\ \texttt{and}\ D'\ \texttt{in}\ P \quad \text{if } \mathrm{dpv}(D) \cap \mathrm{dpv}(D') = \emptyset.
\end{array}
$$

Fig. 2. Structural Congruence

The *bindings* for names are $k?(\tilde{x})\ \texttt{in}\ P$, $X(\tilde{x}\tilde{k}) = P$, and $(\nu a)P$; those for channels are $\texttt{request}\ a(k)\ \texttt{in}\ P, \texttt{accept}\ a(k)\ \texttt{in}\ P, \texttt{catch}\ k(k')\ \texttt{in}\ P, X(\tilde{x}\tilde{k}) = P$, and $(\nu k)P$; and that for process variables are $\texttt{def}\ D\ \texttt{in}\ P$. The derived notions of bound and free identifiers, alpha equivalence $\equiv_\alpha$, and substitution are standard. For P a process, $\mathrm{fpv}(P)$ denotes the set of *free process variables*, $\mathrm{fn}(P)$ denotes the set of *free names*, and $\mathrm{fc}(P)$ the set of *free channels*. We also define $\mathrm{fu}(P) = \mathrm{fn}(P) \cup \mathrm{fc}(P)$.

$$(\texttt{accept}\ a(k)\ \texttt{in}\ P_1) \mid (\texttt{request}\ a(k)\ \texttt{in}\ P_2) \ \rightarrow\ (\nu k)(P_1 \mid P_2) \qquad \text{[Link]}$$

$$(k![\tilde{e}]; P_1) \mid\ (k?(\tilde{x})\ \texttt{in}\ P_2) \ \rightarrow\ P_1 \mid P_2[\tilde{c}/\tilde{x}] \qquad (\tilde{e} \downarrow \tilde{c}) \qquad \text{[Com]}$$

$$(k \lhd l_i; P) \mid (k \rhd \{l_1 : P_1 [\!] \cdots [\!] l_n : P_n\}) \ \rightarrow\ P \mid\ P_i \ \ (1 \leq i \leq n) \qquad \text{[Label]}$$

$$(\texttt{throw}\ k[k']; P_1) \mid\ (\texttt{catch}\ k(k')\ \texttt{in}\ P_2) \ \rightarrow\ P_1 \mid\ P_2 \qquad \text{[Pass]}$$

$$\texttt{if}\ e\ \texttt{then}\ P_1\ \texttt{else}\ P_2 \ \rightarrow\ P_1 \qquad (e \downarrow \texttt{true}) \qquad \text{[If1]}$$

$$\texttt{if}\ e\ \texttt{then}\ P_1\ \texttt{else}\ P_2 \ \rightarrow\ P_2 \qquad (e \downarrow \texttt{false}) \qquad \text{[If2]}$$

$$\texttt{def}\ D\ \texttt{in}\ (X[\tilde{e}\tilde{k}] \mid Q) \ \rightarrow\ \texttt{def}\ D\ \texttt{in}\ (P[\tilde{c}/\tilde{x}] \mid Q)\ (\tilde{e} \downarrow \tilde{c}, X(\tilde{x}\tilde{k}) = P \in D) \qquad \text{[Def]}$$

$$P \rightarrow P' \ \Rightarrow\ (\nu u)P \rightarrow (\nu u)P' \qquad \text{[Scop]}$$

$$P \rightarrow P' \ \Rightarrow\ P \mid Q \ \rightarrow\ P' \mid Q \qquad \text{[Par]}$$

$$P \rightarrow P' \ \Rightarrow\ \texttt{def}\ D\ \texttt{in}\ P \ \rightarrow\ \texttt{def}\ D\ \texttt{in}\ P' \qquad \text{[Defin]}$$

$$P \equiv P' \text{ and } P' \rightarrow Q' \text{ and } Q' \equiv Q \ \Rightarrow\ P \rightarrow Q \qquad \text{[Str]}$$

Fig. 3. Reduction

We also need to talk about the set of *process variables introduced in declarations* $\mathrm{dpv}(X_1(\tilde{x}_1\tilde{k}_1) = P_1\ \texttt{and} \cdots \texttt{and}\ X_n(\tilde{x}_n\tilde{k}_n) = P_n) = \{X_1, \ldots, X_n\}$.

Structural congruence is the smallest congruence relation on processes that include the equations in Figure 2. The operational semantics is given by the *reduction relation*, denoted $P \rightarrow Q$, the smallest relation on processes generated by the rules in Figure 3, where $e \downarrow c$ says that expression e evaluates to constant c.

Rule [Link] establishes a new session between the server $\texttt{accept}\ a(k)\ \texttt{in}\ P_1$ and the client $\texttt{request}\ a(k)\ \texttt{in}\ P_2$ via shared name a. Rule [Com] transmits values between the client and the server at the private channel so that determinacy of value delivery is ensured among the two parties. Rule [Pass] is the key rule to allow *higher-order session communication*, i.e. session channel send and receive, with which various protocols are expressed, allowing complex nested structured communications. To show the difference between channels and names, for example,

$$\texttt{accept}\ a(k)\ \texttt{in}\ P_1 \mid \texttt{accept}\ a(k)\ \texttt{in}\ P_2 \mid \texttt{request}\ a(k)\ \texttt{in}\ Q$$

is accepted by the type system, while

$$\texttt{throw}\ k[k']; P_1 \mid \texttt{throw}\ k[k']; P_2 \mid \texttt{catch}\ k(k')\ \texttt{in}\ Q$$

is prohibited since two senders at k appear in context at the same time.

Relationship with the Rewriting Rules of the π-Calculus

The essence of rule [Pass] is related to a "trick" in a rule of the operational semantics of a variant of the π-calculus, called the πI-calculus in the literature [19]. This calculus restricts name passing to bound (private) name passing. Syntactically it restricts outputs to processes to the form:

$$(\nu\tilde{y})(\overline{x}\langle\tilde{y}\rangle \mid P) \quad \text{with } \tilde{y} \text{ pairwise distinct} \qquad (1)$$

$$\begin{array}{rrl} \text{Sort} & S & ::= \mathsf{nat} \mid \mathsf{bool} \mid \langle \alpha, \overline{\alpha} \rangle \\ \text{Type} & \alpha & ::= ?[\tilde{S}]; \alpha \mid ?[\alpha]; \beta \mid \&\{l_1 \colon \alpha_1, \ldots, l_n \colon \alpha_n\} \mid \mathtt{end} \mid \bot \mid \\ & & \quad ![\tilde{S}]; \alpha \mid ![\alpha]; \beta \mid \oplus\{l_1 \colon \alpha_1, \ldots, l_n \colon \alpha_n\} \mid t \mid \mu t.\alpha \end{array}$$

Fig. 4. The syntax of types

where $\tilde{y} = y_1 \ldots y_n$ denotes a potentially empty vector, $\mid$ denotes parallel composition, and $\overline{x}\langle \tilde{v} \rangle$ is an asynchronous output (or a message). We write the process in (1) as $\overline{x}(\tilde{y}).P$. The dynamics has the following form by the restriction to the bound output.

$$x(\tilde{y}).P \mid \overline{x}(\tilde{y}).Q \ \rightarrow \ (\nu \tilde{y})(P \mid Q) \tag{2}$$

Note that $\tilde{y}$, present both in the input and in the output, indicates that α-conversion is implicitly performed ahead of communication. One can easily observe a similarity between this rule, and rules [Link] and [Pass]: channel k is always freshly generated in rule [Link] and channel k' in rule [Pass] is already created and bound at a previous interaction. Hence no substitution is performed in (2), [Link] or [Pass].

2.2 Type Discipline

Structured communication-based programming allows a clear description of complex interaction structures beyond conventional communication primitives. The more complex the interaction becomes, the more difficult it is to capture the whole interactive behaviour and to write correct programs. The session type discipline offers a simple static checking framework to guarantee the correctness of communication patterns in such situations. It guarantees that well-typed programs are exempt from incompatibility in interaction patterns.

Types

Given a set of *type variables* ranged over by $t, t', \ldots$, the grammar in Figure 4 defines the set $\mathcal{S}$ of *sorts* ranged over by $S, S', \ldots$, and the set $\mathcal{T}$ of *types* ranged over by $\alpha, \beta, \ldots$.

The type $?[\tilde{S}]; \alpha$ represents the behaviour of first inputting values of sorts $\tilde{S}$, then performing the actions prescribed by type α; type $?[\alpha]; \beta$ represents a similar behaviour, which starts with channel input (catch) instead; types $![\tilde{S}]; \alpha$ and $![\alpha]; \beta$ are the dual of $?[\tilde{S}]; \alpha$ and $?[\alpha]; \beta$, sending values instead of receiving. Type $\&\{l_1 \colon \alpha_1, \ldots, l_n \colon \alpha_n\}$ describes a branching behaviour: it waits with n options, and behave as type α_i if i-th action is selected (external choice); type $\oplus\{l_1 \colon \alpha_1, \ldots, l_n \colon \alpha_n\}$ then represents the behaviour which would select one of l_i and then behaves as α_i, according to the selected l_i (internal choice). Type `end` represents inaction, acting as the unit of sequential composition; $\mu t.\alpha$ denotes a recursive behaviour, representing the behaviour that starts by doing α and, when t is encountered, recurs to α again; and finally $\bot$ is a specific type indicating that no further interaction is possible at a given name.

For a type α in which $\bot$ does not occur, we define $\overline{\alpha}$, the *co-type* (or dual) of α, by exchanging ! and ?, and & and $\oplus$. The inductive definition is in Figure 5.

$$\begin{array}{ccc}
\overline{![\tilde{S}];\alpha} = ?[\tilde{S}];\overline{\alpha} & \overline{\oplus\{l_i\colon \alpha_i\}_{i\in I}} = \&\{l_i\colon \overline{\alpha_i}\}_{i\in I} & \overline{![\alpha];\beta} = ?[\alpha];\overline{\beta} \\
\overline{?[\tilde{S}];\alpha} = ![\tilde{S}];\overline{\alpha} & \overline{\&\{l_i\colon \alpha_i\}_{i\in I}} = \oplus\{l_i\colon \overline{\alpha_i}\}_{i\in I} & \overline{?[\alpha];\beta} = ![\alpha];\overline{\beta} \\
\overline{\mathtt{end}} = \mathtt{end} & \overline{\mu t.\alpha} = \mu t.\overline{\alpha} & \overline{t} = t
\end{array}$$

Fig. 5. The co-type of a type

Recursive Types

One of the contributions of the present abstract is a precise definition and the fixed point theorem on recursive types which were omitted from the original paper. We follow the standard co-inductive treatment of recursive types [18]. The μ operator is a binder, giving rise, in the standard way, to notions of bound and free variables and alpha-equivalence. We do not distinguish between alpha-convertible types. Furthermore, we take an *equi-recursive* view of types, not distinguishing between a type $\mu t.\alpha$ and its unfolding $\alpha[\mu t.\alpha/t]$. We are interested on *contractive types* only.

Definition 2.1 (Contractive) *A type is* contractive *if for each of its sub-expressions $\mu t.\mu t_1 \ldots \mu t_n.\alpha$, the body α is not t.*

Henceforth we assume all types to be contractive.

Definition 2.2 (Type equivalence) *Two types α and β are said to be equivalent if the pair (α,β) is in the largest fix point of the monotone function $F\colon \mathcal{P}(\mathcal{T}\times\mathcal{T}) \to \mathcal{P}(\mathcal{T}\times\mathcal{T})$ defined by:*

$$\begin{aligned}
F(R) = {} & \{(\mathtt{end},\mathtt{end}),(\bot,\bot)\} \\
& \cup \{(?[\tilde{S}];\alpha, ?[\tilde{S}];\beta) \mid (\alpha,\beta)\in R\} \\
& \cup \{(![\tilde{S}];\alpha, ![\tilde{S}];\beta) \mid (\alpha,\beta)\in R\} \\
& \cup \{(?[\alpha];\beta, ?[\alpha'];\beta') \mid (\alpha,\alpha'),(\beta,\beta')\in R\} \\
& \cup \{(![\alpha];\beta, ![\alpha'];\beta') \mid (\alpha,\alpha'),(\beta,\beta')\in R\} \\
& \cup \{(\oplus\{l_i\colon \alpha_i\}_{i\in I}, \oplus\{l_i\colon \beta_i\}_{i\in I}) \mid (\alpha_i,\beta_i)\in R, \forall i\in I\} \\
& \cup \{(\&\{l_i\colon \alpha_i\}_{i\in I}, \&\{l_i\colon \beta_i\}_{i\in I}) \mid (\alpha_i,\beta_i)\in R, \forall i\in I\} \\
& \cup \{(\mu t.\alpha,\beta) \mid (\alpha[\mu t.\alpha/t],\beta)\in R\} \\
& \cup \{(\alpha,\mu t.\beta) \mid (\alpha,\beta[\mu t.\beta/t])\in R\}
\end{aligned}$$

Theorem 2.3 *The largest fix point of function F is an equivalence relation.*

Proof. For each of the three cases (reflexivity, symmetry, transitivity) we follow [18], Theorems 21.3.6–7. Take symmetry. A relation R is symmetric if it is closed under the monotone function $\mathrm{Sym}(R) = \{(\alpha,\beta\})) \mid (\beta,\alpha)\in R\}$. We start by noting that (cf. [18], Theorem 21.3.6):

$\mathrm{Sym}(F(R)) \subseteq F(\mathrm{Sym}(R))$ implies that the largest fixed point of F is symmetric.

We then show that $\mathrm{Sym}(F(R)) \subseteq F(\mathrm{Sym}(R))$. Let $(\alpha,\beta)\in \mathrm{Sym}(F(R))$. By definition of Sym, there exists $(\beta,\alpha)\in F(R)$. Our goal is to show that $(\beta,\alpha)\in F(\mathrm{Sym}(R))$. Consider all possible shapes of α. We focus on two cases; the remaining are similar.

Case $\alpha = \mathtt{end}$**.** Since $(\beta, \alpha) \in F(R)$, the definition of F implies that $\beta = \mathtt{end}$ or $\beta = \mu t.\gamma$ with $(\alpha, \gamma[\beta/t]) \in R$. In the first case, notice that $(\mathtt{end}, \mathtt{end}) \in F(R)$ for any R, in particular for $F(\mathrm{Sym}(R))$. In the second case, we know that $(\gamma[\beta/t], \alpha) \in \mathrm{Sym}(R)$ (by the definition of Sym), hence that $(\beta, \alpha) \in F(\mathrm{Sym}(R))$ (by the definition of F).

Case $\alpha = \&\{l_i\colon \alpha_i\}_{i\in I}$**.** Since $(\beta, \alpha) \in F(R)$, the definition of F implies that $\beta = \&\{l_i\colon \beta_i\}_{i\in I}$ or $\beta = \mu t.\gamma$ with $(\alpha, \gamma[\beta/t]) \in R$. In the first case, by the definition of Sym, we have $(\beta_i, \alpha_i) \in \mathrm{Sym}(R)$ for all $i \in I$, hence $(\&\{l_i\colon \beta_i\}_{i\in I}, \&\{l_i\colon \alpha_i\}_{i\in I}) \in F(\mathrm{Sym}(R))$. In the second case proceed as above.

Henceforth types are understood up to type equivalence, so that, for example, in a typing derivation, types $\mu t.\alpha$ and $\alpha[\mu t.\alpha/t]$ can be used interchangeably. Due to the presence of record structures in the syntax of types ($\oplus\{l_1\colon \alpha_1, \ldots, l_n\colon \alpha_n\}$, $\&\{l_1\colon \alpha_1, \ldots, l_n\colon \alpha_n\}$), we do not pursue an interpretation of types as regular infinite trees (the interested reader may refer to [22] for such an interpretation).

Typing System

A *sorting* (resp. a *typing*, resp. a *basis*) is a finite partial map from names to sorts (resp. from channels to types, resp. from variables to sequences of sorts and types). We let $\Gamma, \Gamma', \ldots$ (resp. $\Delta, \Delta', \ldots$, resp. $\Theta, \Theta', \ldots$) range over sortings (resp. typings, resp. bases).

Definition 2.4 (Type algebra) *Typings* Δ_0 *and* Δ_1 *are* compatible*, written* $\Delta_0 \asymp \Delta_1$*, if* $\Delta_0(k) = \overline{\Delta_1(k)}$ *for all* $k \in \mathrm{dom}(\Delta_0) \cap \mathrm{dom}(\Delta_1)$*. When* $\Delta_0 \asymp \Delta_1$*, the* composition of Δ_0 and Δ_1*, written* $\Delta_0 \circ \Delta_1$*, is given as a typing such that* $(\Delta_0 \circ \Delta_1)(k)$ *is (1)* $\bot$*, if* $k \in \mathrm{dom}(\Delta_0) \cap \mathrm{dom}(\Delta_1)$*; (2)* $\Delta_i(k)$*, if* $k \in \mathrm{dom}(\Delta_i) \setminus \mathrm{dom}(\Delta_{i+1 \bmod 2})$ *for* $i \in \{0, 1\}$*; and (3) undefined otherwise.*

We write $\Delta \cdot k\colon \alpha$ when $k \notin \mathrm{dom}(\Delta)$. This notation is then extended to $\Delta \cdot \Delta'$. Also, $\Theta \setminus x$ denotes the result of taking off $x\colon \Theta(x)$ from Θ. Similarly for $\Gamma \setminus a$ and for $\Delta \setminus k$.

Typing judgement are of the form $\Theta; \Gamma \vdash P \triangleright \Delta$ which reads: "under the environment $\Theta; \Gamma$, process P has typing Δ". The typing system is defined by the axioms and rules in Figure 6. We call a typing *completed* when it contains only `end` types [12]. Rules [Var] and [Inact] make sure that the leaves in every derivation tree contain complete typings only (then, $\bot$ may be inserted via rule [Bot]). We also simplify the recursive definition to the single case; the extension to the multiple recursion is obvious.

2.3 Changes from the ESOP'98 system

For the syntax, we added $x, y, z, \ldots$ to the category of names, thus incorporating the set of variables into that of names. We have made clear the notions of bindings for the various identifiers in the calculus. For the structural congruence relation, we have replaced $\mathrm{fpv}(D)$ by $\mathrm{dpv}(D)$, the set of variables introduced in declaration D, and we added rule $\mathtt{def}\ D\ \mathtt{in}\ \mathtt{inact} \equiv \mathtt{inact}$ for flexibility. For types, we changed the syntax from `1` to `end`, from $\uparrow$ to !, and from $\downarrow$ to ?, following [12]. We have also added more accurate definitions for recursive types (Definitions 2.1 and 2.2) for clarification. For the typing system, we added [NameI], [Nat], [Bool], [Sum] and

$$\Gamma \cdot a\colon S \vdash a \triangleright S \qquad \Gamma \vdash 1 \triangleright \mathsf{nat} \qquad \Gamma \vdash \mathtt{true}, \mathtt{false} \triangleright \mathsf{bool} \qquad \frac{\Gamma \vdash e_i \triangleright \mathsf{nat}}{\Gamma \vdash e_1 + e_2 \triangleright \mathsf{nat}}$$
[NameI],[Nat],[Bool],[Sum]

$$\frac{\Theta; \Gamma \vdash P \triangleright \Delta \cdot k\colon \mathtt{end}}{\Theta; \Gamma \vdash P \triangleright \Delta \cdot k\colon \bot} \qquad \frac{\Delta \text{ completed}}{\Theta; \Gamma \vdash \mathtt{inact} \triangleright \Delta}$$
[Bot],[Inact]

$$\frac{\Gamma \vdash a \triangleright \langle \alpha, \overline{\alpha} \rangle \qquad \Theta; \Gamma \vdash P \triangleright \Delta \cdot k\colon \alpha}{\Theta; \Gamma \vdash \mathtt{accept}\ a(k)\ \mathtt{in}\ P \triangleright \Delta}$$
[Acc]

$$\frac{\Gamma \vdash a \triangleright \langle \alpha, \overline{\alpha} \rangle \qquad \Theta; \Gamma \vdash P \triangleright \Delta \cdot k\colon \overline{\alpha}}{\Theta; \Gamma \vdash \mathtt{request}\ a(k)\ \mathtt{in}\ P \triangleright \Delta}$$
[Req]

$$\frac{\Gamma \vdash \tilde{e} \triangleright \tilde{S} \qquad \Theta; \Gamma \vdash P \triangleright \Delta \cdot k\colon \alpha}{\Theta; \Gamma \vdash k![\tilde{e}]; P \triangleright \Delta \cdot k\colon ![\tilde{S}]; \alpha}$$
[Send]

$$\frac{\Theta; \Gamma \cdot \tilde{x}\colon \tilde{S} \vdash P \triangleright \Delta \cdot k\colon \alpha}{\Theta; \Gamma \vdash k?(\tilde{x})\ \mathtt{in}\ P \triangleright \Delta \cdot k\colon ?[\tilde{S}]; \alpha}$$
[Rcv]

$$\frac{\Theta; \Gamma \vdash P_1 \triangleright \Delta \cdot k\colon \alpha_1 \quad \cdots \quad \Theta; \Gamma \vdash P_n \triangleright \Delta \cdot k\colon \alpha_n}{\Theta; \Gamma \vdash \ k \triangleright \{l_1 : P_1 \rVert \cdots \rVert l_n : P_n\} \triangleright \Delta \cdot k\colon \&\{l_1\colon \alpha_1, \ldots, l_n\colon \alpha_n\}}$$
[Br]

$$\frac{\Theta; \Gamma \vdash P \triangleright \Delta \cdot k\colon \alpha_j}{\Theta; \Gamma \vdash k \lhd l_j; P \triangleright \Delta \cdot k\colon \ \oplus \{l_1\colon \alpha_1, \ldots, l_n\colon \alpha_n\}} \ (1 \leq j \leq n)$$
[Sel]

$$\frac{\Theta; \Gamma \vdash P \triangleright \Delta \cdot k\colon \beta}{\Theta; \Gamma \vdash \mathtt{throw}\ k[k']; P \triangleright \Delta \cdot k\colon ![\alpha]; \beta \cdot k'\colon \alpha}$$
[Thr]

$$\frac{\Theta; \Gamma \vdash P \triangleright \Delta \cdot k\colon \beta \cdot k'\colon \alpha}{\Theta; \Gamma \vdash \mathtt{catch}\ k(k')\ \mathtt{in}\ P \triangleright \Delta \cdot k\colon ?[\alpha]; \beta}$$
[Cat]

$$\frac{\Theta; \Gamma \vdash P \triangleright \Delta \qquad \Theta; \Gamma \vdash Q \triangleright \Delta'}{\Theta; \Gamma \vdash P \mid Q \triangleright \Delta \circ \Delta'} (\Delta \asymp \Delta')$$
[Conc]

$$\frac{\Gamma \vdash e \triangleright \mathsf{bool} \qquad \Theta; \Gamma \vdash P \triangleright \Delta \qquad \Theta; \Gamma \vdash Q \triangleright \Delta}{\Theta; \Gamma \vdash \mathtt{if}\ e\ \mathtt{then}\ P\ \mathtt{else}\ Q \triangleright \Delta}$$
[If]

$$\frac{\Theta; \Gamma \cdot a\colon S \vdash P \triangleright \Delta}{\Theta; \Gamma \vdash (\nu a)P \triangleright \Delta} \qquad \frac{\Theta; \Gamma \vdash P \triangleright \Delta \cdot k\colon \bot}{\Theta; \Gamma \vdash \ (\nu k)P \triangleright \Delta}$$
[NRes],[CRes]

$$\frac{\Delta \text{ completed} \qquad \Gamma \vdash \tilde{e} \triangleright \tilde{S}}{\Theta \cdot X\colon \tilde{S}\tilde{\alpha}; \Gamma \vdash X[\tilde{e}\tilde{k}] \triangleright \Delta \cdot \tilde{k}\colon \tilde{\alpha}}$$
[Var]

$$\frac{\Theta \cdot X\colon \tilde{S}\tilde{\alpha}; \Gamma \cdot \tilde{x}\colon \tilde{S} \vdash P \triangleright \tilde{k}\colon \tilde{\alpha} \qquad \Theta \cdot X\colon \tilde{S}\tilde{\alpha}; \Gamma \vdash Q \triangleright \Delta}{\Theta; \Gamma \vdash \mathtt{def}\ X(\tilde{x}\tilde{k}) = P\ \mathtt{in}\ Q \triangleright \Delta}$$
[Def]

Fig. 6. Typing System

revised [Acc], [Req], [Var], [Def]. All of these changes are improvements and do not imply any technical difference with respect to [13].

However there is one important addition with respect to the typing system in [13]: the [Bot]-rule. Without the [Bot]-rule, subject congruence (Lemma 2.9) does not hold. Take for example process $\mathtt{throw}\ k[k']; \mathtt{inact} \mid \mathtt{inact}$ structural

congruent to $\texttt{throw}\ k[k'];\texttt{inact}$. We have

$$\vdash \texttt{throw}\ k[k'];\texttt{inact} \mid \texttt{inact} \triangleright k\colon ![\texttt{end}];\texttt{end} \cdot k'\colon \bot$$

but process $\texttt{throw}\ k[k'];\texttt{inact}$ is not typable under the *same typing* [1]. In [2], the authors fixed the problem by adding the condition $\beta \neq \texttt{end}$ in the [Thr]-rule. We believe the solution herein presented offers extra flexibility.

2.4 Subject Reduction and Type Safety

We start with a few auxiliary results; Subject-Reduction is on page 10, and Type Safety on page 12.

Lemma 2.5 (Weakening Lemma) *Let* $\Theta;\Gamma \vdash P \triangleright \Delta$.

(i) *If* $X \notin \mathrm{dom}(\Theta)$, *then* $\Theta, X\colon \tilde{S}\tilde{\alpha};\Gamma \vdash P \triangleright \Delta$.

(ii) *If* $a \notin \mathrm{dom}(\Gamma)$, *then* $\Theta;\Gamma, a\colon S \vdash P \triangleright \Delta$.

(iii) *If* $k \notin \mathrm{dom}(\Delta)$ *and* $\alpha = \bot$ *or* $\alpha = \texttt{end}$, *then* $\Theta;\Gamma \vdash P \triangleright \Delta \cdot k\colon \alpha$.

Proof. A simple induction on the derivation tree of each sequent. For iii, we note that in [Inact] and [Var], Δ contains only $\texttt{end}$.

Lemma 2.6 (Strengthening Lemma) *Let* $\Theta;\Gamma \vdash P \triangleright \Delta$.

(i) *If* $X \notin \mathrm{fpv}(P)$, *then* $\Theta \setminus X;\Gamma \vdash P \triangleright \Delta$.

(ii) *If* $a \notin \mathrm{fn}(P)$, *then* $\Theta;\Gamma \setminus a \vdash P \triangleright \Delta$.

(iii) *If* $k \notin \mathrm{fc}(P)$, *then* $\Theta;\Gamma \vdash P \triangleright \Delta \setminus k$.

Proof. Standard.

Lemma 2.7 (Channel Lemma) (i) *If* $\Theta;\Gamma \vdash P \triangleright \Delta \cdot k\colon \alpha$ *and* $k \notin \mathrm{fc}(P)$, *then* $\alpha = \bot, \texttt{end}$.

(ii) *If* $\Theta;\Gamma \vdash P \triangleright \Delta$ *and* $k \in \mathrm{fc}(P)$, *then* $k \in \mathrm{dom}(\Delta)$.

Proof. A simple induction on the derivation tree for each sequent.

We omit the standard renaming properties of variables and channels, but present the Substitution Lemma for names. Note that we do *not* require a substitution lemma for channels or process variables, for they are not communicated.

Lemma 2.8 (Substitution Lemma) *If* $\Theta;\Gamma, x\colon S \vdash P \triangleright \Delta$ *and* $\Theta;\Gamma \vdash c\colon S$, *then* $\Theta;\Gamma \vdash P[c/x] \triangleright \Delta$

Proof. Standard.

We write $\Delta \prec \Delta'$ if we obtain Δ' from Δ by replacing $k_1\colon \texttt{end}, ..., k_n\colon \texttt{end}$ $(n \geq 0)$ in Δ by $k_1\colon \bot, ..., k_n\colon \bot$. If $\Delta \prec \Delta'$, we can obtain Δ' from Δ by applying the [Bot]-rule zero or more times.

Lemma 2.9 (Subject Congruence) *If* $\Theta;\Gamma \vdash P \triangleright \Delta$ *and* $P \equiv Q$, *then* $\Theta;\Gamma \vdash Q \triangleright \Delta$.

Proof. Case $P \mid \texttt{inact} \equiv P$**.** We show that if $\Theta;\Gamma \vdash P \mid \texttt{inact} \triangleright \Delta$, then $\Theta;\Gamma \vdash P \triangleright \Delta$. Suppose

$$\Theta;\Gamma \vdash P \triangleright \Delta_1 \quad \text{and} \quad \Theta;\Gamma \vdash \texttt{inact} \triangleright \Delta_2.$$

with $\Delta_1 \circ \Delta_2 = \Delta$. Note that Δ_2 only contains $\texttt{end}$ or $\bot$, hence we can set: $\Delta_1 = \Delta_1' \circ \{\tilde{k}\colon \tilde{\texttt{end}}\}$ and $\Delta_2 = \Delta_2' \cdot \{\tilde{k}\colon \tilde{\texttt{end}}\}$ with $\Delta_1' \circ \Delta_2' = \Delta_1' \cdot \Delta_2'$ and $\Delta = \Delta_1' \cdot \Delta_2' \cdot \{\tilde{k}\colon \tilde{\bot}\}$. Then by the [BOT]-rule, we have:

$$\Theta;\Gamma \vdash P \triangleright \Delta_1' \cdot \{\tilde{k}\colon \tilde{\bot}\}$$

Notice that, given the form of Δ above, we know that $\mathrm{dom}(\Delta_2') \cap \mathrm{dom}(\Delta_1' \cdot \{\tilde{k}\colon \bot\}) = \emptyset$. Hence by applying Weakening, we have:

$$\Theta;\Gamma \vdash P \triangleright \Delta_1' \cdot \Delta_2' \cdot \{\tilde{k}\colon \tilde{\bot}\}$$

as required.

For the other direction, we set $\Delta = \emptyset$ in [INACT].

Case $P \mid Q \equiv Q \mid P$, $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$**.** By commutativity and associativity of $\asymp$.

Case $(\nu u)P \mid Q \equiv (\nu u)(P \mid Q)$ if $u \notin \mathrm{fu}(Q)$**.** The case when u is a name is standard. Suppose u is channel k and assume $\Theta;\Gamma \vdash (\nu k)(P \mid Q) \triangleright \Delta$. We have

$$\frac{\Theta;\Gamma \vdash P \triangleright \Delta_1' \qquad \Theta;\Gamma \vdash Q \triangleright \Delta_2'}{\Theta;\Gamma \vdash P \mid Q \triangleright \Delta' \cdot k\colon \bot}$$

with $\Delta' \cdot k\colon \bot = \Delta_1' \circ \Delta_2'$, and $\Delta' \prec \Delta$ by [BOT]. First notice that k can be in either Δ_i' or in both. The interesting case is when it occurs in both; from Lemma 2.7(i) and the fact that $k \notin \mathrm{fc}(Q)$ we know that $\Delta_1' = \Delta_1 \cdot k\colon \texttt{end}$ and $\Delta_2' = \Delta_2 \cdot k\colon \texttt{end}$. Then, by applying the [BOT]-rule to k in P, we have $\Theta;\Gamma \vdash P \triangleright \Delta_1 \cdot k\colon \bot$, and by applying [CRES] we obtain $\Theta;\Gamma \vdash (\nu k)P \triangleright \Delta_1$. On the other hand, by Strengthening, we have $\Theta;\Gamma \vdash Q \triangleright \Delta_2$. Then, the application of [CONC] yields $\Theta;\Gamma \vdash (\nu k)P \mid Q \triangleright \Delta'$. Then by applying the [BOT]-rule, we obtain $\Theta;\Gamma \vdash (\nu k)P \mid Q \triangleright \Delta$, as required. The other direction is easy.

Case $(\nu u)\texttt{inact} \equiv \texttt{inact}$**.** Standard by Weakening and Strengthening.

Case $\texttt{def}\ D\ \texttt{in}\ \texttt{inact} \equiv \texttt{inact}$**.** Similar to the first case using Weakening and Strengthening.

Case $(\nu u)\texttt{def}\ D\ \texttt{in}\ P \equiv \texttt{def}\ D\ \texttt{in}\ (\nu u)P$ if $u \notin \mathrm{fu}(D)$**.** Similar to the scope opening case using Weakening and Strengthening.

Case $(\texttt{def}\ D\ \texttt{in}\ P) \mid Q \equiv \texttt{def}\ D\ \texttt{in}\ (P \mid Q)$ if $\mathrm{dpv}(D) \cap \mathrm{fpv}(Q) = \emptyset$**.** Similar with the scope opening case using Weakening and Strengthening.

Theorem 2.10 (Subject Reduction) *If* $\Theta;\Gamma \vdash P \triangleright \Delta$ *and* $P \to^* Q$*, then* $\Theta;\Gamma \vdash Q \triangleright \Delta$*.*

Proof. We assume that

$$\Gamma \vdash e \triangleright S \quad \text{and} \quad e \downarrow c \quad \text{implies} \quad \Gamma \vdash c \triangleright S \tag{3}$$

and prove the result by induction on the last rule applied.

Case [Link] $(\texttt{accept}\ a(k)\ \texttt{in}\ P_1) \mid (\texttt{request}\ a(k)\ \texttt{in}\ P_2) \rightarrow (\nu k)(P_1 \mid P_2)$**.** Suppose $\Theta; \Gamma \vdash (\texttt{accept}\ a(k)\ \texttt{in}\ P_1) \mid (\texttt{request}\ a(k)\ \texttt{in}\ P_2) \triangleright \Delta$. Then the assumption is derived from:

$$\frac{\Gamma \vdash a \triangleright \langle \alpha, \overline{\alpha} \rangle \quad \Theta; \Gamma \vdash P_1 \triangleright \Delta_1' \cdot k \colon \alpha}{\Theta; \Gamma \vdash \texttt{accept}\ a(k)\ \texttt{in}\ P_1 \triangleright \Delta_1'} \quad \text{and} \quad \frac{\Gamma \vdash a \triangleright \langle \alpha, \overline{\alpha} \rangle \quad \Theta; \Gamma \vdash P_2 \triangleright \Delta_2' \cdot k \colon \overline{\alpha}}{\Theta; \Gamma \vdash \texttt{request}\ a(k)\ \texttt{in}\ P_2 \triangleright \Delta_2'}$$

and [Bot] with $\Delta_i' \prec \Delta_i$, [Conc] with $\Delta_1 \circ \Delta_2 = \Delta'$, and [Bot] with $\Delta' \prec \Delta$. Then applying [Bot] to P_1 and P_2, we have:

$$\frac{\Theta; \Gamma \vdash P_1 \triangleright \Delta_1' \cdot k \colon \alpha}{\Theta; \Gamma \vdash P_1 \triangleright \Delta_1 \cdot k \colon \alpha} \quad \text{and} \quad \frac{\Theta; \Gamma \vdash P_2 \triangleright \Delta_2' \cdot k \colon \overline{\alpha}}{\Theta; \Gamma \vdash P_2 \triangleright \Delta_2 \cdot k \colon \overline{\alpha}}$$

Then we apply [Conc] to P_1 and P_2 to obtain:

$$\frac{\Theta; \Gamma \vdash P_1 \triangleright \Delta_1 \cdot k \colon \alpha \quad \Theta; \Gamma \vdash P_2 \triangleright \Delta_2 \cdot k \colon \overline{\alpha}}{\Theta; \Gamma \vdash P_1 \mid P_2 \triangleright \Delta' \cdot k \colon \bot}$$

Now applying [CRes] and [Bot], we are done.

Case [Com] $(k![\tilde{e}]; P_1) \mid (k?(\tilde{x})\ \texttt{in}\ P_2) \rightarrow P_1 \mid P_2[\tilde{c}/\tilde{x}]$ with $\tilde{e} \downarrow \tilde{c}$**.** The assumption is derived from:

$$\frac{\Gamma \vdash \tilde{e} \triangleright \tilde{S} \quad \Theta; \Gamma \vdash P_1 \triangleright \Delta_1' \cdot k \colon \alpha}{\Theta; \Gamma \vdash k![\tilde{e}]; P_1 \triangleright \Delta_1' \cdot k \colon ![\tilde{S}]; \alpha} \text{ and } \frac{\Theta; \Gamma \cdot \tilde{x} \colon \tilde{S} \vdash P_2 \triangleright \Delta_2' \cdot k \colon \overline{\alpha}}{\Theta; \Gamma \vdash k?(\tilde{x})\ \texttt{in}\ P_2 \triangleright \Delta_2' \cdot k \colon ?[\tilde{S}]; \overline{\alpha}}$$

and [Bot] with $\Delta_i' \prec \Delta_i$, [Conc] with $\Delta_1 \circ \Delta_2 \cdot k \colon \bot = \Delta'$, and [Bot] with $\Delta' \prec \Delta$. Then by (3), we know $\Gamma \vdash \tilde{c} \triangleright \tilde{S}$. By applying Substitution Lemma, we have:

$$\Theta; \Gamma \vdash P_2[\tilde{c}/\tilde{x}] \triangleright \Delta_2' \cdot k \colon \overline{\alpha}$$

Now the application of [Bot] and [Conc] to P_1 and $P_2[\tilde{c}/\tilde{x}]$, then by [Bot], we complete this case.

Case [Label] $(k \lhd l_i; P_1) \mid (k \rhd \{l_1 : P_1 [\!] \cdots [\!] l_n : P_n\}) \rightarrow P \mid P_i \quad (1 \leq i \leq n)$**.** Similar to the above case.

Case [Pass] $(\texttt{throw}\ k[k']; P_1) \mid (\texttt{catch}\ k(k')\ \texttt{in}\ P_2) \rightarrow P_1 \mid P_2$**.** The assumption is derived from:

$$\frac{\Theta; \Gamma \vdash P_1 \triangleright \Delta_1' \cdot k \colon \beta}{\Theta; \Gamma \vdash \texttt{throw}\ k[k']; P_1 \triangleright \Delta_1' \cdot k \colon ![\alpha]; \beta \cdot k' \colon \alpha}$$

and

$$\frac{\Theta; \Gamma \vdash P_2 \triangleright \Delta_2' \cdot k \colon \overline{\beta} \cdot k' \colon \alpha}{\Theta; \Gamma \vdash \texttt{catch}\ k(k')\ \texttt{in}\ P_2 \triangleright \Delta_2' \cdot k \colon ?[\alpha]; \overline{\beta}}$$

and [Bot] with $\Delta_i' \prec \Delta_i$, [Conc] with $\Delta_1 \circ \Delta_2 \cdot k \colon \bot \cdot k' \colon \alpha = \Delta'$ and [Bot] with $\Delta' \prec \Delta$. Note that $k, k' \notin \mathrm{dom}(\Delta_1, \Delta_2, \Delta_1', \Delta_2')$. By applying [Bot], [Conc] to P_1 and P_2, and then by [Bot], we obtain the required result.

Case [If1],[If2]**.** Trivial.

Case [DEF] $\texttt{def}\ D\ \texttt{in}\ (X[\tilde{e}\tilde{k}] \mid Q) \rightarrow \texttt{def}\ D\ \texttt{in}\ (P[\tilde{c}/\tilde{x}] \mid Q)$ with $\tilde{e} \downarrow \tilde{c}$ and $X(\tilde{x}\tilde{k}) = P \in D$. Simplifying the recursive definition to the single case, we set $D = (X(\tilde{x}\tilde{k}) = P)$. Then the assumption is derived from:

$$\dfrac{\Theta\cdot X\colon \tilde{S}\tilde{\alpha};\Gamma\cdot\tilde{x}\colon\tilde{S}\vdash P\triangleright\tilde{k}\colon\tilde{\alpha} \qquad \dfrac{\Theta\cdot X\colon\tilde{S}\tilde{\alpha};\Gamma\vdash X[\tilde{e}\tilde{k}]\triangleright\Delta_1'\cdot\tilde{k}\colon\tilde{\alpha} \quad \Theta\cdot X\colon\tilde{S}\tilde{\alpha};\Gamma\vdash Q\triangleright\Delta_2'}{\Theta\cdot X\colon\tilde{S}\tilde{\alpha};\Gamma\vdash X[\tilde{e}\tilde{k}]\mid Q\triangleright\Delta''\cdot\tilde{k}\colon\tilde{\alpha} \quad \Delta''\prec\Delta'}}{\Theta;\Gamma\vdash \texttt{def}\ X(\tilde{x}\tilde{k}) = P\ \texttt{in}\ (X[\tilde{e}\tilde{k}]\mid Q)\triangleright\Delta'\cdot\tilde{k}\colon\tilde{\alpha}}$$

with $\Delta_0 = \Delta'\cdot\tilde{k}\colon\tilde{\alpha}$, $\Delta' = \Delta_1'\circ\Delta_2'$ and $\Delta_0\prec\Delta$. Note that Δ_1' contains only $\bot$ or $\texttt{end}$. Then applying Substitution Lemma to P, we have:

$$\Theta\cdot X\colon\tilde{S}\tilde{\alpha};\Gamma\vdash P[\tilde{c}/\tilde{x}]\triangleright\tilde{k}\colon\tilde{\alpha}$$

Notice that $\tilde{k}\cap\mathrm{dom}(\Delta_1') = \emptyset$, since $(\Delta_1'\circ\Delta_2')\cdot\tilde{k}\colon\tilde{\alpha}$ is defined. Then by Weakening, we have:

$$\Theta\cdot X\colon\tilde{S}\tilde{\alpha};\Gamma\vdash P[\tilde{c}/\tilde{x}]\triangleright\Delta_1'\cdot\tilde{k}\colon\tilde{\alpha}$$

Now by [CONC], we have

$$\Theta\cdot X\colon\tilde{S}\tilde{\alpha};\Gamma\vdash P[\tilde{c}/\tilde{x}]\mid Q\triangleright\Delta''\cdot\tilde{k}\colon\tilde{\alpha}$$

Finally by [BOT] ($\Delta''\prec\Delta'$), then by [DEF], we obtain:

$$\Theta;\Gamma\vdash\texttt{def}\ X(\tilde{x}\tilde{k}) = P\ \texttt{in}\ (P[\tilde{c}/\tilde{x}]\mid Q)\triangleright\Delta'\cdot\tilde{k}\colon\tilde{\alpha}$$

Then we can apply [BOT] to obtain Δ, as desired.

Case [STR]**.** By Subject-Congruence.

To formalise Type Safety, we need the following notions. A *k-process* is a process prefixed by subject k (such as $k![\tilde{e}];P$ and $\texttt{catch}\ k(k')\ \texttt{in}\ P$). Next, a *k-redex* is the parallel composition of two k-processes, i.e. either of form $(k![\tilde{e}];P \mid k?(\tilde{x})\ \texttt{in}\ Q)$, $(k\lhd l;P \mid k\rhd\{l_1 : Q_1 [\!] \cdots [\!] l_n : Q_n\})$, or $(\texttt{throw}\ k[k'];P \mid \texttt{catch}\ k(k'')\ \texttt{in}\ Q)$. Then P is an *error* if $P\equiv(\nu\tilde{u})(\texttt{def}\ D\ \texttt{in}\ (Q\mid R))$ where Q is, for some k, the parallel composition of *either* two k-processes that do not form a k-redex, *or* three or more k-processes. We then have:

Theorem 2.11 (Type Safety) *A typable program never reduces to an error.*

Proof. By Subject Reduction it suffices to show that typable programs are not errors. The proof is by reductio ad absurdum, assuming error processes typable. Suppose that $\Theta;\Gamma\vdash\texttt{def}\ D\ \texttt{in}\ (\nu\tilde{u})(P\mid Q)\triangleright\Delta$. Analysing the derivation tree for the process, we conclude that $\Theta;\Gamma\vdash P\triangleright\Delta'$, for some Δ'. We now analyse the two classes of error processes.

When $P = P_1 \mid P_2$ is the parallel composition of two k-processes that do not form a redex, there are several cases to consider. They are all alike; take for example the pair label-select/throw. Applying [CONC] on P, we have $\Theta;\Gamma\vdash P_1\triangleright\Delta_1'$ and $\Theta;\Gamma\vdash P_2\triangleright\Delta_2'$ with $\Delta'\prec\Delta_1'\circ\Delta_2'$. Applying [SEL] on P_1 and [THR] on P_2 we conclude that $k\colon\oplus\{l_1\colon\alpha_1,\ldots,l_n\colon\alpha_n\}\in\Delta_1'$ and $k\colon![\alpha];\beta\in\Delta_2'$. But then $\Delta_1'\circ\Delta_2'$ is not defined, hence $\texttt{def}\ D\ \texttt{in}\ (\nu\tilde{u})(P\mid Q)$ is not typable.

When P is the parallel composition of three or more k-processes, we concentrate on the case of three processes, for the remaining cases reduce to this. So let $P = (P_1 \mid P_2) \mid P_3$. Applying [Conc], we know that $\Theta; \Gamma \vdash P_1 \mid P_2 \triangleright \Sigma$ and $\Theta; \Gamma \vdash P_3 \triangleright \Sigma'$ with $\Delta' \prec \Sigma \circ \Sigma'$. If $P_1 \mid P_2$ is not a k-redex, we use the case above. Otherwise, it must be the case that $k \colon \bot \in \Sigma$. From Lemma 2.7(ii), we know that $k \in \mathrm{dom}(\Sigma')$, thus $\Sigma \circ \Sigma'$ is not defined, hence $\mathtt{def}\ D\ \mathtt{in}\ (\nu \tilde{u})(P \mid Q)$ is not typable.

3 A More Liberal Session Passing Style

Rule [Pass] in the original ESOP'98 system

$$(\mathtt{throw}\ k[k']; P_1) \mid\ (\mathtt{catch}\ k(k')\ \mathtt{in}\ P_2)\ \rightarrow\ P_1 \mid\ P_2$$

does not allow the transmission of an arbitrary channel. In most situations a process $\mathtt{catch}\ k(k'')\ \mathtt{in}\ P_2$ can be alpha-converted ahead of communication [3] so that the bound variable k'' syntactically matches the free variable k' in the `throw` process. The exception happens exactly when k' is free in P_2: alpha-conversion becomes impossible (for it would capture free variable k'), and communication cannot occur.

A more liberal rule would allow the transmission of an arbitrary channel, implying a substitution on the client side.

$$(\mathtt{throw}\ k[k']; P_1) \mid\ (\mathtt{catch}\ k(k'')\ \mathtt{in}\ P_2)\ \rightarrow\ P_1 \mid\ P_2[k'/k'']$$

Unfortunately this rule breaks Subject Reduction (Theorem 2.10). A counter-example is a process which, possessing one end of a channel, receives the second end. The process:

$$\mathtt{throw}\ k[k'] \mid \mathtt{catch}\ k(k'')\ \mathtt{in}\ k''?(y)\ \mathtt{in}\ k'![1] \tag{4}$$

is typable under typing $k \colon \bot, k' \colon \bot$, but reduces to process

$$k'?(x)\ \mathtt{in}\ k'![1]$$

which is not typable under the *same typing* [7].

One might think that the simplest solution of the above problem is to add the side condition $k' \notin \mathrm{fc}(P_2)$ to the above rule proposal. This reduction rule, however, implies that the condition of free channels is checked *at runtime*, which contradicts the aim of static type checking to preserve Subject Reduction. The same sort of situation occurs in the ESOP'98 system, where, in presence of a process $\mathtt{throw}\ k[k']; P_1 \mid \mathtt{catch}\ k(k'')\ \mathtt{in}\ P_2$, the runtime system has to check whether $k' \in \mathrm{fc}(P_2)$ in order to alpha-convert the `catch`-process before applying rule [Pass] above.

A different alternative would be to type the contractum with a *different* typing. In the above case and for the `catch` process in the redex, we have $k' \colon ![\mathsf{nat}]; \mathtt{end}$, and $k'' \colon ?[\mathsf{nat}]; \mathtt{end}$. In the contractum, channels k' and k'' are aliased and it is not obvious how to build, from the premises, the correct type $?[\mathsf{nat}]; ![\mathsf{nat}]; \mathtt{end}$ for k'.

[3] Cf. Paragraph *Relationship with the Rewriting Rules of the π-Calculus* in Section 2.1.

A solution, due to Gay and Hole [12], explicitly distinguishes between the two ends of a channel. For a channel κ, its two ends are denoted κ^+ and κ^-. Channels are now runtime entities (they are not supposed to occur in programs) created by rule [Link], which becomes:

$$(\texttt{accept}\ a(x)\ \texttt{in}\ P_1) \mid (\texttt{request}\ a(x)\ \texttt{in}\ P_2) \ \rightarrow\ (\nu\kappa)(P_1[\kappa^+/x] \mid P_2[\kappa^-/x])$$

Rules that synchronise two processes on a given channel are updated so that each process explicitly mentions one of the ends. For example rule [Thr] becomes:

$$(\texttt{throw}\ \kappa^p[k'];P_1) \mid (\texttt{catch}\ \kappa^{\overline{p}}(x)\ \texttt{in}\ P_2) \ \rightarrow\ P_1 \mid P_2[k'/x]$$

where p denotes one end (one *polarity*) of κ and $\overline{p}$ the other.

A further change allows a typing Δ to contain one type for κ^+ and a different type (not necessarily dual) for κ^-. Parallel composition juxtaposes the typings of the two operands (provided they have disjoint domains), rather than composing using $\circ$ (cf. Definition 2.4).

$$\frac{\Theta;\Gamma \vdash P \triangleright \Delta \quad \Theta;\Gamma \vdash Q \triangleright \Delta'}{\Theta;\Gamma \vdash P \mid Q \triangleright \Delta \cdot \Delta'} \qquad [\textsc{Conc}]$$

An immediate consequence of the new rule is that we do not need the bottom $\bot$ type anymore, or the notions of typing compatibility and composition. One the other hand, the new rule for channel restriction requires the two ends of the channel to be of dual types.

$$\frac{\Theta;\Gamma \vdash P \triangleright \Delta \cdot \kappa^+ : \alpha \cdot \kappa^- : \overline{\alpha}}{\Theta;\Gamma \vdash (\nu\kappa)P \triangleright \Delta} \qquad [\textsc{CRes}]$$

Notice that the original rule (in Figure 6) requires an entry $k : \bot$ in typing Δ.

To understand how the new system works, consider process (4) refined into the new syntax:

$$\texttt{throw}\ \kappa^+[\kappa'^+] \mid \texttt{catch}\ \kappa^-(x)\ \texttt{in}\ x?(y)\ \texttt{in}\ \kappa'^-![1]$$

The process is typable under the typing $\kappa^+ : ![\alpha];\texttt{end}, \kappa^- : ?[\alpha];\texttt{end}, \kappa'^+ : \alpha, \kappa'^- : \overline{\alpha}$ where α is the type $?[\texttt{nat}];\texttt{end}$. It now reduces to

$$\kappa'^+?(x)\ \texttt{in}\ \kappa'^-![1]$$

which is still typable (this time under typing $\kappa'^+ : \alpha, \kappa'^- : \overline{\alpha}$).

Clearly, typability over arbitrary channel environments is not closed under reduction any more. For example, the process

$$\kappa^+![\texttt{true}] \mid \kappa^-?(x)\ \texttt{in}\ \kappa'^-![x+1] \qquad (5)$$

is typable under typing $\kappa^+ : ![\texttt{bool}];\texttt{end}, \kappa^- : ?[\texttt{nat}];\texttt{end}, \kappa'^- : ![\texttt{nat}];\texttt{end}$, but reduces to

$$\kappa'^-![\texttt{true}+1]$$

which is not typable. The last step is then to consider, for Subject Reduction and Type Safety purposes, only typings where the two ends of a channel are of dual

types. We call such typings *balanced* [12]. This restriction rules out the above typing (since ![bool];end is not dual to ?[nat];end), hence process (5) is not guaranteed to preserve typability under reduction or to be type safe.

3.1 Syntax and Operational Semantics

With respect to the syntax in Figure 1, we let κ, rather than k, range over *channels*. Identifier k now stands for polarised channels (κ^+, κ^-) or names (a, x). As such k cannot occur in a binding position anymore; four process constructors need to be updated: `accept`, `request`, `catch`, and `def`. The grammar of the language is given by the rules in Figure 1, replacing the productions for `accept`, `request`, `catch`, and `def` by the ones below.

$$
\begin{array}{rcll}
P & ::= & \texttt{request}\ a(x)\ \texttt{in}\ P & \text{session request} \\
& | & \texttt{accept}\ a(x)\ \texttt{in}\ P & \text{session acceptance} \\
& | & \texttt{catch}\ k(x)\ \texttt{in}\ P & \text{channel reception} \\
& | & \ldots & \\
D & ::= & X_1(\tilde{x}_1\tilde{y}_1) = P_1\ \texttt{and} \cdots \texttt{and}\ X_n(\tilde{x}_n\tilde{y}_n) = P_n & \text{declaration for recursion} \\
k & ::= & x \mid \kappa^p & \text{channel variables and values} \\
p & ::= & + \mid - & \text{channel polarities}
\end{array}
$$

Duality on polarities is defined as $\overline{+} = -$ and $\overline{-} = +$. Variable x is now bound in any of $\texttt{request}\ a(x)\ \texttt{in}\ P, \texttt{accept}\ a(x)\ \texttt{in}\ P, \texttt{catch}\ k(x)\ \texttt{in}\ P$, and the variables in $\tilde{x}\tilde{y}$ become bound in $X(\tilde{x}\tilde{y}) = P$, so that, in contrast with the system in Section 2.1, we have three more name binders, and only $(\nu\kappa)P$ remains as a binder for channels.

The new reduction relation adapts the rules that directly work with channels. Reduction is given by replacing, in Figure 3, rules [Link], [Com], [Label], [Pass], and [Def] by the rules below. Structural congruence (Figure 2) remains unchanged.

$$
\begin{array}{rcll}
(\texttt{accept}\ a(x)\ \texttt{in}\ P_1) \mid (\texttt{request}\ a(x)\ \texttt{in}\ P_2) & \to & (\nu\kappa)(P_1[\kappa^+/x] \mid P_2[\kappa^-/x]) & \text{[Link]} \\
(\kappa^p![\tilde{e}]; P_1) \mid (\kappa^{\overline{p}}?(\tilde{x})\ \texttt{in}\ P_2) & \to & P_1 \mid P_2[\tilde{c}/\tilde{x}] \quad (\tilde{e} \downarrow \tilde{c}) & \text{[Com]} \\
(\kappa^p \lhd l_i; P) \mid (\kappa^{\overline{p}} \rhd \{l_1 : P_1 [\!] \cdots [\!] l_n : P_n\}) & \to & P \mid P_i \quad (1 \le i \le n) & \text{[Label]} \\
(\texttt{throw}\ \kappa^p[\kappa'^q]; P_1) \mid (\texttt{catch}\ \kappa^{\overline{p}}(x)\ \texttt{in}\ P_2) & \to & P_1 \mid P_2[\kappa'^q/x] & \text{[Pass]} \\
\multicolumn{4}{l}{\texttt{def}\ D\ \texttt{in}\ (X[\tilde{e}\tilde{\kappa^p}] \mid Q) \to \texttt{def}\ D\ \texttt{in}\ (P[\tilde{c}/\tilde{x}][\tilde{\kappa^p}/\tilde{y}] \mid Q)\ (\tilde{e} \downarrow \tilde{c}, X(\tilde{x}\tilde{y}) = P \in D)\ \text{[Def]}}
\end{array}
$$

3.2 Type Discipline

Types in Figure 4 remain unchanged, except that bottom $\bot$ is no longer needed. Typings still feature entries of the form $k\colon \alpha$, only that k can now be a name x, or a polarised channel κ^+ or κ^-. The type system needs adjustments in rules [Acc], [Req], [Cat] and [Def] due to the change in syntax. Also, the absence of rule [Bot] is compensated by a new rule [CRes'] for $(\nu\kappa)P$. The main change however happens in rules [Conc] and [CRes]. The new type system is given by replacing, in Figure 6, rules [Acc], [Req], [Cat], [Def], [Conc], [CRes], and [Bot] by the rules below.

$$\frac{\Gamma \vdash a\colon \langle \alpha, \overline{\alpha} \rangle \qquad \Theta; \Gamma \vdash P \triangleright \Delta \cdot x\colon \alpha}{\Theta; \Gamma \vdash \texttt{accept}\ a(x)\ \texttt{in}\ P \triangleright \Delta} \quad [\textsc{Acc}]$$

$$\frac{\Gamma \vdash a\colon \langle \alpha, \overline{\alpha} \rangle \qquad \Theta; \Gamma \vdash P \triangleright \Delta \cdot x\colon \overline{\alpha}}{\Theta; \Gamma \vdash \texttt{request}\ a(x)\ \texttt{in}\ P \triangleright \Delta} \quad [\textsc{Req}]$$

$$\frac{\Theta; \Gamma \vdash P \triangleright \Delta \cdot k\colon \beta \cdot x\colon \alpha}{\Theta; \Gamma \vdash \texttt{catch}\ k(x)\ \texttt{in}\ P \triangleright \Delta \cdot k\colon ?[\alpha]; \beta} \quad [\textsc{Cat}]$$

$$\frac{\Theta \cdot X\colon \tilde{S}\tilde{\alpha}; \Gamma \cdot \tilde{x}\colon \tilde{S} \vdash P \triangleright \tilde{y}\colon \tilde{\alpha} \quad \Theta \cdot X\colon \tilde{S}\tilde{\alpha}; \Gamma \vdash Q \triangleright \Delta}{\Theta; \Gamma \vdash \texttt{def}\ X(\tilde{x}\tilde{y}) = P\ \texttt{in}\ Q \triangleright \Delta} \quad [\textsc{Def}]$$

$$\frac{\Theta; \Gamma \vdash P \triangleright \Delta \qquad \Theta; \Gamma \vdash Q \triangleright \Delta'}{\Theta; \Gamma \vdash P \mid Q \triangleright \Delta \cdot \Delta'} \quad [\textsc{Conc}]$$

$$\frac{\Theta; \Gamma \vdash P \triangleright \Delta \cdot \kappa^+\colon \alpha \cdot \kappa^-\colon \overline{\alpha}}{\Theta; \Gamma \vdash (\nu\kappa)P \triangleright \Delta} \quad [\textsc{CRes}]$$

$$\frac{\Theta; \Gamma \vdash P \triangleright \Delta \qquad \kappa \text{ not in } \Delta}{\Theta; \Gamma \vdash (\nu\kappa)P \triangleright \Delta} \quad [\textsc{CRes'}]$$

In rule [Var], Figure 6, typing $k_1 \ldots k_n\colon \alpha_1 \ldots \alpha_n$ is understood as $k_1\colon \alpha_1 \cdot \ldots \cdot k_n\colon \alpha_n$, defined only when the k_i are pairwise distinct. This restriction is crucial in controlling channel aliasing during reduction via the [Def] rule, now that a substitution is performed. Suppose that we judge as valid a sequent of the form $X\colon SS \vdash X[kk] \triangleright k\colon S$. Then, taking for D the process definition $X(k'k'') = k'![1].k''![2]$, process $\texttt{def}\ D\ \texttt{in}\ X[kk]$ would be typable under typing $k\colon !\texttt{nat.end}$, but reduces to process $\texttt{def}\ D\ \texttt{in}\ k![1].k![2]$ which is not typable under the same typing.

3.3 Subject Reduction and Type Safety

The absence of typing compatibility (in rule [Conc]) is compensated by balanced typings. We say that a typing Δ is *balanced* if whenever $\kappa^+\colon \alpha, \kappa^-\colon \beta \in \Delta$, then $\alpha = \overline{\beta}$ [12]. Subject-Reduction (Theorem 3.3) and Type Safety (Theorem 3.4) hold only in presence of balanced typings.

We rely on the Weakening, Strengthening, Channel and Substitution Lemmas of Section 2.4, adapted to the syntax and typing system of this section. Since we now replace channels in processes, we need a Channel Replacement Lemma, a result not needed for the ESOP'98 system [13]. The proofs below are adapted from those in references [12,23], except that our scope extrusion rule (in Figure 2) is more general than that of [12].

Lemma 3.1 (Channel Replacement) *If* $\Theta; \Gamma \vdash P \triangleright \Delta \cdot x\colon \alpha$, *then* $\Theta; \Gamma \vdash P[\kappa^p/x] \triangleright \Delta \cdot \kappa^p\colon \alpha$.

Proof. A straightforward induction on the derivation tree for P.

Lemma 3.2 (Subject Congruence) *If* $\Theta; \Gamma \vdash P \triangleright \Delta$ *and* $P \equiv Q$, *then* $\Theta; \Gamma \vdash Q \triangleright \Delta$.

Proof. The proof follows the pattern of that of Lemma 2.9, albeit slightly simplified by the absence of the non-structural [Bot] rule. We detail the two most interesting

cases.

Case $P \mid \mathtt{inact} \equiv P$**.** We show that if $\Theta;\Gamma \vdash P \mid \mathtt{inact} \triangleright \Delta$, then $\Theta;\Gamma \vdash P \triangleright \Delta$. Suppose that

$$\Theta;\Gamma \vdash P \triangleright \Delta_1 \quad \text{and} \quad \Theta;\Gamma \vdash \mathtt{inact} \triangleright \Delta_2$$

with $\Delta_1 \cdot \Delta_2 = \Delta$. Note that Δ_2 only contains `end`. Applying Weakening to P, we have $\Theta;\Gamma \vdash P \triangleright \Delta_1 \cdot \Delta_2$ as required.

For the other direction we start with derivation $\Theta;\Gamma \vdash \mathtt{inact} \triangleright \emptyset$, and then apply rule [Conc].

Case $(\nu u)(P \mid Q) \equiv (\nu u)P \mid Q$ if $u \notin \mathrm{fu}(Q)$**.** The case when u is a name is standard. Suppose u is channel k and assume $\Theta;\Gamma \vdash (\nu\kappa)(P \mid Q) \triangleright \Delta$. We consider the [CRes] case (the [CRes'] case is simpler):

$$\frac{\Theta;\Gamma \vdash P \triangleright \Delta_1 \qquad \Theta;\Gamma \vdash Q \triangleright \Delta_2}{\Theta;\Gamma \vdash P \mid Q \triangleright \Delta \cdot \kappa^p : \alpha \cdot \kappa^{\overline{p}} : \overline{\alpha}}$$

First notice that κ^p and $\kappa^{\overline{p}}$ can be both in either Δ_i or one in each. When they are both in Δ_1 we conclude the case by applying [CRes] and [Conc]. When they are both in Δ_2, by the Channel Lemma we know that the types for κ^p and $\kappa^{\overline{p}}$ in Δ_2 are `end`. We conclude the case by applying Strengthening twice to Q before applying [CRes'] and [Conc]. Finally, when κ^p is in Δ_1' and $\kappa^{\overline{p}}$ in Δ_2', we apply Strengthening to Q and Weakening to P, before applying [CRes] and [Conc].

The other direction is simpler.

Theorem 3.3 (Subject Reduction) *If $\Theta;\Gamma \vdash P \triangleright \Delta$ with Δ balanced and $P \to^* Q$, then $\Theta;\Gamma \vdash Q \triangleright \Delta'$ and Δ' balanced.*

Proof. The proof is similar to that of Theorem 2.10. We concentrate on the four new reduction rules, and reuse the remaining cases.

Case [Link] $(\mathtt{accept}\ a(x)\ \mathtt{in}\ P_1) \mid (\mathtt{request}\ a(x)\ \mathtt{in}\ P_2) \to (\nu\kappa)(P_1[\kappa^+/x] \mid P_2[\kappa^-/x])$**.** The assumption is derived from

$$\frac{\Theta;\Gamma \vdash P_1 \triangleright \Delta \cdot x : \alpha}{\Theta;\Gamma, a : \langle \alpha, \overline{\alpha} \rangle \vdash \mathtt{accept}\ a(x)\ \mathtt{in}\ P_1 \triangleright \Delta}$$

from

$$\frac{\Theta;\Gamma \vdash P_2 \triangleright \Delta \cdot x : \overline{\alpha}}{\Theta;\Gamma, a : \langle \alpha, \overline{\alpha} \rangle \vdash \mathtt{request}\ a(x)\ \mathtt{in}\ P_2 \triangleright \Delta}$$

and from [Conc] with $\Delta_1 \cdot \Delta_2 = \Delta$. Applying the Channel Replacement Lemma to P_1 and also to P_2, we have $\Theta;\Gamma \vdash P_1[\kappa^+/x] \triangleright \Delta \cdot \kappa^+ : \alpha$, and $\Theta;\Gamma \vdash P_2[\kappa^-/x] \triangleright \Delta \cdot \kappa^- : \overline{\alpha}$. The case concludes with the application of rule [Conc] followed by rule [CRes].

Case [Com] $(\kappa^p![\tilde{e}]; P_1) \mid (\kappa^{\overline{p}}?(\tilde{x})\ \mathtt{in}\ P_2) \to P_1 \mid P_2[\tilde{c}/\tilde{x}]$**.** The assumption is derived from:

$$\frac{\Gamma \vdash \tilde{e} \triangleright \tilde{S} \qquad \Theta;\Gamma \vdash P \triangleright \Delta_1 \cdot \kappa^p : \alpha}{\Theta;\Gamma \vdash k![\tilde{e}]; P_1 \triangleright \Delta_1 \cdot \kappa^p : ![\tilde{S}]; \alpha} \qquad \frac{\Theta;\Gamma \cdot \tilde{x} : \tilde{S} \vdash P_2 \triangleright \Delta_2 \cdot \kappa^{\overline{p}} : \overline{\alpha}}{\Theta;\Gamma \vdash k?(\tilde{x})\ \mathtt{in}\ P_2 \triangleright \Delta_2 \cdot \kappa^{\overline{p}} : ?[\tilde{S}]; \overline{\alpha}}$$

and [Conc] with $\Delta_1 \cdot \kappa^p : ![\tilde{S}]; \alpha \cdot \Delta_2 \cdot \kappa^{\overline{p}} : ?[\tilde{S}]; \overline{\alpha} = \Delta$. Notice that the types for κ^p and for $\kappa^{\overline{p}}$ are dual since Δ is balanced by hypothesis. Then by (3), page 10, we

know $\Gamma \vdash \tilde{c} \triangleright \tilde{S}$. We conclude the case by applying Substitution Lemma to P_2, and the [CONC]-rule to P_1 and to $P_2[\tilde{c}/\tilde{x}]$.

Case [LABEL]**.** Similar to the homonymous case in the proof of Theorem 2.10, relying, as above, on the fact that Δ is balanced.

Case [PASS] $(\texttt{throw}\ \kappa^p[\kappa'^q]; P_1) \mid (\texttt{catch}\ \kappa^{\overline{p}}(x)\ \texttt{in}\ P_2) \rightarrow P_1 \mid P_2[\kappa'^q/x]$**.** The assumption is derived from

$$\frac{\Theta; \Gamma \vdash P_1 \triangleright \Delta_1 \cdot \kappa^p \colon \beta}{\Theta; \Gamma \vdash \texttt{throw}\ \kappa^p[\kappa'^q]; P_1 \triangleright \Delta_1 \cdot \kappa^p \colon ![\alpha]; \beta \cdot \kappa'^q \colon \alpha}$$

from

$$\frac{\Theta; \Gamma \vdash P_2 \triangleright \Delta_2 \cdot \kappa^{\overline{p}} \colon \overline{\beta} \cdot x \colon \alpha}{\Theta; \Gamma \vdash \texttt{catch}\ \kappa^{\overline{p}}(x)\ \texttt{in}\ P_2 \triangleright \Delta_2 \cdot \kappa^{\overline{p}} \colon ?[\alpha]; \overline{\overline{\beta}}}$$

and from [CONC] with $\Delta_1 \cdot \kappa^p \colon ![\alpha]; \beta \cdot \kappa'^q \colon \alpha \cdot \Delta_2 \cdot \kappa^{\overline{p}} \colon ?[\alpha]; \overline{\beta} = \Delta$. Once again, the types for κ^p and for $\kappa^{\overline{p}}$ are dual since Δ is balanced by hypothesis. Applying the Channel Replacement Lemma to P_2, we obtain

$$\Theta; \Gamma \vdash P_2[\kappa'^q/x] \triangleright \Delta_2 \cdot \kappa^{\overline{p}} \colon \overline{\beta} \cdot \kappa'^q \colon \alpha$$

By applying [CONC] to P_1 and $P_2[\kappa'^q/x]$, we obtain the required result.

Case [DEF] $(\texttt{def}\ D\ \texttt{in}\ (X[\tilde{e}\tilde{\kappa^p}] \mid Q) \rightarrow \texttt{def}\ D\ \texttt{in}\ (P[\tilde{c}/\tilde{x}][\tilde{\kappa^p}/\tilde{y}] \mid Q)$ with $\tilde{e} \downarrow \tilde{c}$ and $X(\tilde{x}\tilde{y}) = P \in D$**.** Similar to the homonymous case in the proof of Theorem 2.10, with the difference that, when building the derivation tree for the contractum, after obtaining

$$\Theta \cdot X \colon \tilde{S}\tilde{\alpha}; \Gamma \vdash P[\tilde{c}/\tilde{x}] \triangleright \tilde{y} \colon \tilde{\alpha}$$

by the application of the Substitution Lemma, we apply Channel replacement to obtain

$$\Theta \cdot X \colon \tilde{S}\tilde{\alpha}; \Gamma \vdash P[\tilde{c}/\tilde{x}][\tilde{\kappa^p}/\tilde{y}] \triangleright \tilde{\kappa^p} \colon \tilde{\alpha}.$$

Theorem 3.4 (Type Safety) *A program typable under a balanced channel environment never reduces to an error.*

Proof. The proof follows the pattern of that of Theorem 2.11, only that the contradiction happens not because typing composition ($\Delta \circ \Delta'$) is not defined, but because the resulting typing ($\Delta \cdot \Delta'$) is not balanced.

4 Conclusion

The study of session typing system is now widespread due to the need for structured communications in various scenarios in distributed applications. They have been studied, at the research level, for the π-calculus [2,11,12,13,17,20,23], Ambients [6], CORBA interfaces [21], multi-threaded functional languages [23,24,15], Web Description Languages [3,4,14], and distributed [8] and multi-threaded Java [7] and, at the industry level, WC3-CDL [5,25] and pi4Tech [16].

In the presence of higher-order session communication, session instantiation dynamically changes structures of sessions during execution, so that it becomes non-trivial to preserve typability. Unfortunately the authors of previous session typing

systems did not realise (or even *forgot* about) the key point of rule [Pass], so that some of the systems published after [13] fail to satisfy the Subject Reduction Theorem. As discussed in this report, the subtlety of the type preservation is related to a treatment of communication channels in the operational semantics of the π-calculus: aliasing of channels, structured safe communication, types, new name creation and the α-conversion are tightly related with this issue.

As a future work, it would be nice to investigate the relationship uniformly using rewriting frameworks [9,10].

Acknowledgements

The authors thank Eduardo Bonelli for his pointing out the issue of the subject congruence in ESOP'98 paper; Kohei Honda for his suggestion of the [Bot]-rule; Simon Gay for his comments on the liberal typing system; and the SecReT reviewers for their comments and suggestions. This work is partially supported by EU IST–2005–015905 MOBIUS project, EU IST–2005–16004 SENSORIA project, EPSRC GR/T03208, EPSRC GR/S55538, EPSRC GR/T04724, EPSRC GR/S68071, FEDER, and Fundação para a Ciência e a Tecnologia (via CLC, CITI, and POSC/EIA/55582/2004 Space-Time-Types project).

References

[1] Bonelli, E., A. Compagnoni and E. Gunter, *Private communication* (2003).

[2] Bonelli, E., A. Compagnoni and E. Gunter, *Correspondence Assertions for Process Synchronization in Concurrent Communications*, Journal of Functional Programming **15** (2005), pp. 219–248.

[3] Carbone, M., K. Honda and N. Yoshida, *A calculus of global interaction based on session types*, in: *2nd Workshop on Developments in Computational Models (DCM)*, ENTCS, 2006.

[4] Carbone, M., K. Honda and N. Yoshida, *Structured communication-centred programming for web services*, in: *ESOP'07*, LNCS, 2007.

[5] Carbone, M., K. Honda, N. Yoshida, R. Milner, G. Brown and S. Ross-Talbot, *A Theoretical Basis of Communication-Centred Concurrent Programming*, To be published by W3C. Available at http://www.dcs.qmul.ac.uk/∼carbonem/cdlpaper/workingnote.pdf (2006).

[6] Compagnoni, A., M. Dezani-Ciancaglini and P. Garralda, *BASS:Boxed Ambients with Safe Sessions*, in: *Proceedings of PPDP'06* (2006), pp. 119–148.

[7] Dezani-Ciancaglini, M., D. Mostrous, N. Yoshida and S. Drossopoulou, *Session types for object-oriented languages*, in: *The 20th European Conference on Object-Oriented Programming*, LNCS (2006), pp. 328–352.

[8] Dezani-Ciancaglini, M., N. Yoshida, A. Ahern and S. Drossopoulou, *A distributed object-oriented language with session types*, in: *Symposium on Trustworthy Global Computing*, LNCS (2005), pp. 299–318.

[9] Fernández, M. and M. J. Gabbay, *Nominal rewriting with name generation: abstraction vs. locality*, in: *Proc. 7th ACM-SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'05)* (2005), pp. 47–58.

[10] Fernández, M., M. J. Gabbay and I. Mackie, *Nominal rewriting systems*, in: *Proc. 6th ACM-SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'04)* (2004), pp. 108–119.

[11] Gay, S. J. and M. J. Hole, *Types and subtypes for client-server interactions*, in: *ESOP'99*, LNCS (1999), pp. 74–90.

[12] Gay, S. J. and M. J. Hole, *Subtyping for session types in the pi calculus*, Acta Informatica **42** (2005), pp. 191–225.

[13] Honda, K., V. T. Vasconcelos and M. Kubo, *Language primitives and type disciplines for structured communication-based programming*, in: *ESOP'98*, LNCS **1381** (1998), pp. 22–138.

[14] Honda, K., N. Yoshida and M. Carbone, *Web services, mobile processes and types*, EATCS **2** (2007), to appear.

[15] Neubauer, M. and P. Thiemann, *An implementation of session types.*, in: *PADL*, Lecture Notes in Computer Science **3057** (2004), pp. 56–70.

[16] O'Hanlon, C., *Conversation with Steve Ross-Talbot*, ACM Queue **4** (2006), pp. 14–23.

[17] Ostrovský, K., "On modelling and analysing concurrent systems," Ph.D. thesis, Chalmers University of Technology (2006).

[18] Pierce, B. C., "Types and Programming Languages," MIT Press, 2002.

[19] Sangiorgi, D., *π-calculus, internal mobility and agent-passing calculi*, Theoretical Computer Science **167** (1996), pp. 235–274.

[20] Takeuchi, K., K. Honda and M. Kubo, *An Interaction-based Language and its Typing System*, in: *PARLE'94*, LNCS **817** (1994), pp. 398–413.

[21] Vallecillo, A., V. T. Vasconcelos and A. Ravara, *Typing the behavior of objects and components using session types*, Fundamenta Informaticæ**73** (2006).

[22] Vasconcelos, V. T., *Recursive types in a calculus of objects*, Transactions of Information Processing Society of Japan **35** (1994), pp. 1828–1836.

[23] Vasconcelos, V. T., S. Gay and A. Ravara, *Typechecking a multithreaded functional language with session types*, Theoretical Computer Science (2006), to appear.

[24] Vasconcelos, V. T., A. Ravara and S. Gay, *Session Types for Functional Multithreading*, in: *CONCUR'04*, LNCS **3170** (2004), pp. 497–511.

[25] Web Services Choreography Working Group, *Web Services Choreography Description Language*, http://www.w3.org/2002/ws/chor/.