Under consideration for publication in Math. Struct. in Comp. Science

Multiparty Asynchronous Session Types

Nobuko Yoshida

Department of Computing, Imperial College London

Received 30 June 2014

This note gives definitions and proofs of basic multiparty session type system based on the communication typing system presented in (Coppo et al., 2014).

1. Introduction

This note is extracted from (Coppo et al., 2014). We give:

— a non-trivial example that illustrates the basic functionalities and features of the process calculus that we work with.
— a definition of a calculus of asynchronous, multiparty sessions (§3) as well as a *communication type system* (§4) assuring that processes behave correctly with respect to the sessions in which they are involved.
— proofs of the subject reduction theorem

2. The Three Buyer Protocol

In this section we present a simple but non-trivial example that illustrates the basic functionalities and features of the process calculus that we work with. This example comes from a Web service usecase in Web Service Choreography Description Language (WS-CDL) Primer 1.0 (Web Services Choreography Working Group, 2002), capturing a collaboration pattern typical to many business and distributed protocols (OOI, 2010; UNIFI, 2002; Scribble, 2008). The setting is that of a system involving Alice, Bob, and Carol that cooperate in order to buy a book from a Seller. The participants follow a protocol that is described informally below:

1 Alice sends a book title to Seller and Seller sends back a quote to Alice and Bob. Alice tells Bob how much she can contribute.
2 If the price is within Bob's budget, Bob notifies both Seller and Alice he accepts, then sends his address to Seller and Seller answers with the delivery date.
3 If the price exceeds Bob's budget, Bob asks Carol to collaborate by establishing a new session. Bob sends Carol how much she has to contribute and *delegates* the remaining interactions with Alice and Seller to her.
4 If Carol's contribution is within her budget, she accepts the quote, notifies Alice, Bob and Seller, and continues the rest of the protocol with Seller and Alice *as if she were Bob*. Otherwise, she notifies Alice, Bob and Seller to quit the protocol.

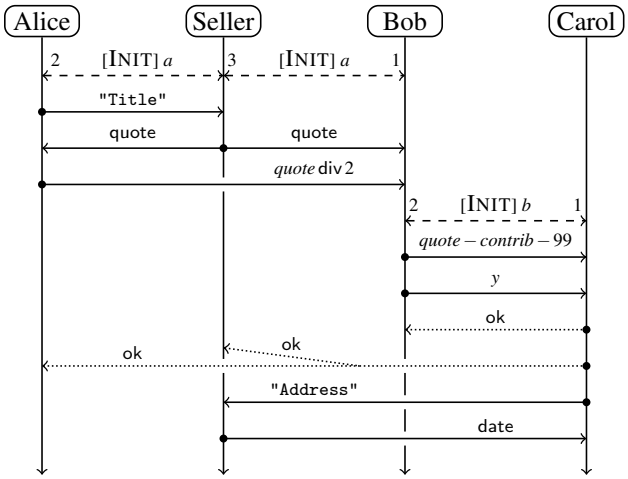

Fig. 1. An execution of the three buyer protocol.

Figure 1 depicts an execution of the above protocol where Bob asks Carol to collaborate (by delegating the remaining interactions with Alice and Seller) and the transaction terminates successfully.

Multiparty session programming consists of two steps: specifying the intended communication protocols using global types and implementing these protocols using processes. The specifications of the three-buyer protocol are given as two distinct global types: one is G_a among Alice, Bob and Seller and the other is G_b between Bob and Carol. In G_a Alice plays role 2, Bob plays role 1, and Seller plays role 3, while in G_b Bob plays role 2 and Carol plays role 1. We annotate the global types with line numbers (*i*) so that we can easily refer to the actions in them.

$$
\begin{array}{lll}
\mathsf{G}_a = & & \\
\quad (1) & 2 \longrightarrow 3 : & \langle \mathsf{string} \rangle . \\
\quad (2) & 3 \longrightarrow \{1,2\} : & \langle \mathsf{int} \rangle . \\
\quad (3) & 2 \longrightarrow 1 : & \langle \mathsf{int} \rangle . \\
\quad (4) & 1 \longrightarrow \{2,3\} : & \{\mathsf{ok} : \quad 1 \longrightarrow 3 : \langle \mathsf{string} \rangle . \\
\quad (5) & & \qquad\qquad 3 \longrightarrow 1 : \langle \mathsf{date} \rangle . \mathsf{end}, \\
\quad (6) & & \quad \mathsf{quit} : \ \mathsf{end} \}
\end{array}
$$

$$
\begin{array}{lll}
\mathsf{G}_b = & & \\
\quad (1) & 2 \longrightarrow 1 : & \langle \mathsf{int} \rangle . \\
\quad (2) & 2 \longrightarrow 1 : & \langle \mathsf{T} \rangle . \\
\quad (3) & 1 \longrightarrow 2 : & \{\mathsf{ok} : \mathsf{end}, \mathsf{quit} : \mathsf{end}\}
\end{array}
$$

$$
\mathsf{T} = \oplus \langle \{2,3\}, \{\mathsf{ok} : !\langle 3, \mathsf{string} \rangle .?(3, \mathsf{date}).\mathsf{end}, \mathsf{quit} : \mathsf{end}\} \rangle
$$

Global types provide an overall description of the two conversations, directly abstracting the scenario of the diagram. In G_a, line (1) denotes Alice sending a string value to Seller. Line (2) says that Seller multicasts the same integer value to Alice and Bob and line (3) says that Alice

$$\text{Seller} = \overline{a}[3](y).y?(2,\mathit{title}).y!\langle\{1,2\},\mathsf{quote}\rangle.y\&(1,\{\mathsf{ok} : y?(1,\mathit{address}).y!\langle 1,\mathsf{date}\rangle.\mathbf{0}, \mathsf{quit} : \mathbf{0}\})$$

$$\text{Alice} = a[2](y).y!\langle 3,\texttt{"Title"}\rangle.y?(3,\mathit{quote})).y!\langle 1,\mathit{quote}\,\mathsf{div}\,2\rangle.y\&(1,\{\mathsf{ok} : \mathbf{0},\ \mathsf{quit} : \mathbf{0}\})$$

$$\begin{aligned}\text{Bob} = a[1](y).y?(3,\mathit{quote}).y?(2,\mathit{contrib}).\mathsf{if}\ (\mathit{quote}-\mathit{contrib} < 100)\ \mathsf{then}\ y\oplus\langle\{2,3\},\mathsf{ok}\rangle.y!\langle 3,\texttt{"Address"}\rangle.y?(3,\mathit{date}).\mathbf{0}\\ \mathsf{else}\,\overline{b}[2](z).z!\langle 1,\mathit{quote}-\mathit{contrib}-99\rangle.z!\langle\langle 1,y\rangle\rangle.z\&(1,\{\mathsf{ok} : \mathbf{0},\mathsf{quit} : \mathbf{0}\})\end{aligned}$$

$$\begin{aligned}\text{Carol} = b[1](z).z?(2,x).z?((2,t)).\mathsf{if}\ (x < 100)\ \mathsf{then}\ z\oplus\langle 2,\mathsf{ok}\rangle.t\oplus\langle\{2,3\},\mathsf{ok}\rangle.t!\langle 3,\texttt{"Address"}\rangle.t?(3,\mathit{date}).\mathbf{0}\\ \mathsf{else}\ z\oplus\langle 2,\mathsf{quit}\rangle.t\oplus\langle\{2,3\},\mathsf{quit}\rangle.\mathbf{0}\end{aligned}$$

Table 1. Implementation of the three buyer protocol.

sends an integer to Bob. In lines (4–6) Bob sends either ok or quit to Seller and Alice. In the first case Bob sends a string to Seller and receives a date from Seller, in the second case there are no further communications.

Line (2) in G_b represents the delegation of a channel with the communication behaviour specified by the session type T from Bob to Carol (note that Seller and Alice in T concern the session on a).

Table 1 shows an implementation of the three buyer protocol conforming to G_a and G_b for the processes Seller, Alice, Bob, and Carol in the calculus that we will formally define in §3.1. The service name a is used for initiating sessions corresponding to the global type G_a. Seller initiates a three party session by means of the session request operation $\overline{a}[3](y)$, where the index 3 identifies Seller. Since 3 is also the overall number of participants in this session, a occurs with an over-bar. Alice and Bob get involved in the session by means of the session accept operations $a[1](y)$ and $a[2](y)$ and the indexes 2 and 1 identify them as Alice and Bob, respectively. Once the session has started, Seller, Alice and Bob communicate using their private channels y. Each channel y can be interpreted as a session endpoint connecting a participant with all the others in the same session; the receivers of the data sent on y are specified by giving the participant numbers. Line (1) of G_a is implemented by the matching output and input actions $y!\langle 3,\texttt{"Title"}\rangle.$ and $y?(2,\mathit{title})$. Line (3) of G_b is implemented by the selection and branching actions $z\oplus\langle 2,\mathsf{ok}\rangle$, $z\oplus\langle 2,\mathsf{quit}\rangle$ and $z\&(1,\{\mathsf{ok} : \mathbf{0},\mathsf{quit} : \mathbf{0}\})$.

In process Bob, if the quote minus Alice's contribution exceeds 100, another session between Bob and Carol is established through the shared service name b. Delegation occurs by passing the private channel y from Bob to Carol (actions $z!\langle\langle 1,y\rangle\rangle$ and $z?((2,t))$), so that the rest of the session with Seller and Alice is carried out by Carol.

In this particular example no deadlock is possible, even if different sessions are interleaved with each other and the communication topology changes because of delegation.

3. The Calculus for Multiparty Sessions

3.1. *Syntax*

The present calculus is a variant of the calculus in (Honda et al., 2008), as explained in §5. The syntax of *processes*, ranged over by $P,Q\ldots$, and *expressions*, ranged over by $e,e',\ldots$, is given by the grammar in Table 2, which shows also naming conventions.

The operational semantics is defined by a set of reduction rules. In the reduction of processes

P	::=	$\overline{u}[\mathtt{p}](y).P$	Multicast request	a, b			Service name
	$\mid$	$u[\mathtt{p}](y).P$	Accept	x			Value variable
	$\mid$	$c!\langle\Pi, e\rangle.P$	Value sending	y, z, t			Channel Variable
	$\mid$	$c?(\mathtt{p}, x).P$	Value reception	s			Session name
	$\mid$	$c!\langle\langle\mathtt{p}, c'\rangle\rangle.P$	Channel delegation	$\mathtt{p}, \mathtt{q}$			Participant number
	$\mid$	$c?((\mathtt{q}, y)).P$	Channel reception	X, Y			Process variable
	$\mid$	$c \oplus \langle\Pi, l\rangle.P$	Selection	l			Label
	$\mid$	$c\&(\mathtt{p}, \{l_i : P_i\}_{i\in I})$	Branching	$s[\mathtt{p}]$			Channel with role
	$\mid$	$\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q$	Conditional	u	::=	$x \mid a$	Identifier
	$\mid$	$P \mid Q$	Parallel	v	::=	$a \mid \mathsf{true}$	Value
	$\mid$	$\mathbf{0}$	Inaction		$\mid$	false	
	$\mid$	$(\nu a : \mathsf{G})P$	Service name hiding	e	::=	$v \mid x$	
	$\mid$	$\mathsf{def}\ D\ \mathsf{in}\ P$	Recursion		$\mid$	$e\ \mathsf{and}\ e'$	Expression
	$\mid$	$X\langle e, c\rangle$	Process call		$\mid$	$\mathsf{not}\ e \ldots$	
	$\mid$	$(\nu s)P$	Session hiding	Π	::=	$\{\mathtt{p}\} \mid \{\mathtt{p}\}\cup\Pi$	Set of participants
	$\mid$	$s : h$	Message queue	c	::=	$y \mid s[\mathtt{p}]$	Channel
D	::=	$X(x, y) = P$	Declaration	m	::=	$(\mathtt{q}, \Pi, v)$	Message in transit
$\mathscr{E}$	::=	$[\,] \mid P \mid (\nu a : G)\mathscr{E}$	Evaluation context		$\mid$	$(\mathtt{q}, \mathtt{p}, s[\mathtt{p}'])$	
	$\mid$	$(\nu s)\mathscr{E} \mid \mathsf{def}\ D\ \mathsf{in}\ \mathscr{E}$			$\mid$	$(\mathtt{q}, \Pi, l)$	
	$\mid$	$\mathscr{E} \mid \mathscr{E}$		h	::=	$h \cdot m \mid \varnothing$	Queue

Table 2. Process syntax and naming conventions.

it is handy to introduce elements, like queues of messages and runtime channels, which are not expected to occur in the source code written by users (*user processes*). These elements, which are referred as *runtime syntax*, appear shaded.

The processes of the form $\overline{u}[\mathtt{p}](y).P$ and $u[\mathtt{p}](y).P$ cooperate in the initiation of a multiparty session through a service name identified by u, where $\mathtt{p}$ denotes a *participant* to the session. Participants are represented by progressive numbers and are ranged over by $\mathtt{p}$, $\mathtt{q}$,... The barred identifier is the one corresponding to the participant with the highest number, which also gives the total number of participants needed to start the session. The (bound) variable y is the placeholder for the channel that will be used in the communications. After opening a session each channel placeholder will be replaced by a *channel with role* $s[\mathtt{p}]$, which represents the runtime channel of the participant $\mathtt{p}$ in the session s.

Process communications (communications that can only take place inside initiated sessions) are performed using the next three pairs of primitives: the sending and receiving of a value; the channel delegation and reception (where the process performing the former action delegates to the process receiving it the capability to participate in a session by passing a channel associated with that session); and the selection and branching (where the former action sends one of the labels offered by the latter). The input/output operations (including the delegation ones) specify the channel and the sender or the receivers, respectively. Thus, $c!\langle\Pi, e\rangle$ denotes the sending of a value on channel c to all the participants in the non-empty set Π; accordingly, $c?(\mathtt{p}, x)$ denotes the intention of receiving a value on channel c from the participant $\mathtt{p}$. The same holds for delegation/reception (but the receiver is only one) and selection/branching. We use $c!\langle\mathtt{p}, e\rangle.P$ and $c \oplus \langle\mathtt{p}, l\rangle.P$ as short for $c!\langle\{\mathtt{p}\}, e\rangle.P$ and $c \oplus \langle\{\mathtt{p}\}, l\rangle.P$, as already done in previous examples.

An *output action* is a value sending, channel delegation or label selection: an *output process* is a process whose first action is an output action. An *input action* is a value reception, session

$$a[1](y).P_1 \mid ... \mid a[n-1](y).P_{n-1} \mid \overline{a}[n](y).P_n \longrightarrow (\nu s)(P_1\{s[1]/y\} \mid ... \mid P_{n-1}\{s[n-1]/y\} \mid P_n\{s[n]/y\} \mid s : \varnothing) \quad \text{[Init]}$$

$$s[\mathtt{p}]!\langle \Pi, e\rangle.P \mid s : h \longrightarrow P \mid s : h \cdot (\mathtt{p}, \Pi, v) \quad (e{\downarrow}v) \quad \text{[Send]}$$

$$s[\mathtt{p}]!\langle\langle \mathtt{q}, s'[\mathtt{p}']\rangle\rangle.P \mid s : h \longrightarrow P \mid s : h \cdot (\mathtt{p}, \mathtt{q}, s'[\mathtt{p}']) \quad \text{[Deleg]}$$

$$s[\mathtt{p}] \oplus \langle \Pi, l\rangle.P \mid s : h \longrightarrow P \mid s : h \cdot (\mathtt{p}, \Pi, l) \quad \text{[Sel]}$$

$$s[\mathtt{p}]?(\mathtt{q}, x).P \mid s : (\mathtt{q}, \mathtt{p}, v) \cdot h \longrightarrow P\{v/x\} \mid s : h \quad \text{[Rcv]}$$

$$s[\mathtt{p}]?((\mathtt{q}, y)).P \mid s : (\mathtt{q}, \mathtt{p}, s'[\mathtt{p}']) \cdot h \longrightarrow P\{s'[\mathtt{p}']/y\} \mid s : h \quad \text{[SRcv]}$$

$$s[\mathtt{p}]\&(\mathtt{q}, \{l_i : P_i\}_{i\in I}) \mid s : (\mathtt{q}, \mathtt{p}, l_j) \cdot h \longrightarrow P_j \mid s : h \quad (j \in I) \quad \text{[Branch]}$$

$$\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \longrightarrow P \quad (e \downarrow \mathsf{true}) \qquad \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \longrightarrow Q \quad (e \downarrow \mathsf{false}) \quad \text{[If-T, If-F]}$$

$$\mathsf{def}\ X(x,y) = P\ \mathsf{in}\ (X\langle e, s[\mathtt{p}]\rangle \mid Q) \longrightarrow \mathsf{def}\ X(x,y) = P\ \mathsf{in}\ (P\{v/x\}\{s[\mathtt{p}]/y\} \mid Q) \quad (e \downarrow v) \quad \text{[ProcCall]}$$

$$P \longrightarrow P' \quad \Rightarrow \quad \mathscr{E}[P] \longrightarrow \mathscr{E}[P'] \quad \text{[Ctxt]}$$

$$P \equiv P' \text{ and } P' \longrightarrow Q' \text{ and } Q \equiv Q' \quad \Rightarrow \quad P \longrightarrow Q \quad \text{[Str]}$$

Table 3. Reduction rules.

reception or label branching: an *input process* is a process whose first action is an input action. A *communication action* is either an output or an input action.

In the hiding of service name a, G denotes the global type of a, see next §.

For simplicity each recursively defined process has exactly one data parameter and one channel parameter.

As usual evaluation contexts are processes with some holes.

As in (Honda et al., 2008), we use message queues in order to model TCP-like asynchronous communications (where message order is preserved and sending is non-blocking). A message in a queue can be a value message, $(\mathtt{q}, \Pi, v)$, indicating that the value v was sent by the participant $\mathtt{q}$ and the recipients are all the participants in Π; a channel message (delegation), $(\mathtt{q}, \mathtt{p}, s[\mathtt{p}'])$, indicating that $\mathtt{q}$ delegates to $\mathtt{p}$ the role of $\mathtt{p}'$ on the session s (represented by the channel with role $s[\mathtt{p}']$); and a label message, $(\mathtt{q}, \Pi, l)$ (similar to a value message). The empty queue is denoted by $\varnothing$. By $h \cdot m$ we denote the queue obtained by concatenating m to the queue h. With some abuse of notation we will also write $m \cdot h$ to denote the queue with head element m. By $s : h$ we denote the queue h of the session s. Queues and channels with role are generated by the operational semantics (described later).

We call *pure* a process which does not contain message queues.

There are many binders: request/accept actions bind channel variables, value receptions bind value variables, channel receptions bind channel variables, declarations bind value and channel variables, recursions bind process variables, hidings bind service and session names. In $(\nu s)P$ all occurrences of $s[\mathtt{p}]$ and the queue s inside P are bound. We say that a process is *closed* if the only free names in it are service names (i.e. if it does not contain free variables or free session names).

$$P \mid \mathbf{0} \equiv P \quad P \mid Q \equiv Q \mid P \quad (P \mid Q) \mid R \equiv P \mid (Q \mid R)$$

$$(\nu r)P \mid Q \equiv (\nu r)(P \mid Q) \quad \text{if } r \notin \mathrm{fn}(Q)$$
$$(\nu r)(\nu r')P \equiv (\nu r')(\nu r)P \quad (\nu a : G)\mathbf{0} \equiv \mathbf{0} \quad (\nu s)(s : \varnothing) \equiv \mathbf{0}$$
$$\text{where } r ::= a : G \mid s$$

$$\mathsf{def}\ D\ \mathsf{in}\ \mathbf{0} \equiv \mathbf{0} \quad \mathsf{def}\ D\ \mathsf{in}\ (\nu r)P \equiv (\nu r)\mathsf{def}\ D\ \mathsf{in}\ P \quad \text{if } r \notin \mathrm{fn}(D)$$

$$(\mathsf{def}\ D\ \mathsf{in}\ P) \mid Q \equiv \mathsf{def}\ D\ \mathsf{in}\ (P \mid Q) \quad \text{if } \mathrm{dpv}(D) \cap \mathrm{fpv}(Q) = \emptyset$$

$$\mathsf{def}\ D\ \mathsf{in}\ (\mathsf{def}\ D'\ \mathsf{in}\ P) \equiv \mathsf{def}\ D'\ \mathsf{in}\ (\mathsf{def}\ D\ \mathsf{in}\ P) \quad \text{if } (\mathrm{dpv}(D) \cup \mathrm{fpv}(D)) \cap \mathrm{dpv}(D') = \mathrm{dpv}(D) \cap (\mathrm{dpv}(D') \cup \mathrm{fpv}(D')) = \emptyset$$

$$s : h \cdot (\mathtt{q}, \Pi, \zeta) \cdot (\mathtt{q}', \Pi', \zeta') \cdot h' \equiv s : h \cdot (\mathtt{q}', \Pi', \zeta') \cdot (\mathtt{q}, \Pi, \zeta) \cdot h' \quad \text{if } \Pi \cap \Pi' = \emptyset \text{ or } \mathtt{q} \neq \mathtt{q}'$$
$$s : h \cdot (\mathtt{q}, \Pi, \zeta) \cdot h' \equiv s : h \cdot (\mathtt{q}, \Pi', \zeta) \cdot (\mathtt{q}, \Pi'', \zeta) \cdot h' \quad \text{if } \Pi = \Pi' \cup \Pi'' \text{ and } \Pi' \cap \Pi'' = \emptyset$$
$$\text{where } \zeta ::= v \mid s[\mathtt{p}] \mid l$$

$$P \equiv P' \quad \Rightarrow \quad \mathscr{E}[P] \equiv \mathscr{E}[P']$$

Table 4. Structural equivalence.

3.2. *Operational Semantics*

Table 3 shows the reduction rules of processes (we use $\longrightarrow^*$ and $\longrightarrow^k$ with the expected. Rule [Init] describes the initiation of a new session among n participants that synchronise over the service name a. The last participant $\overline{a}[n](y).P_n$, distinguished by the overbar on the service name, specifies the number n of participants. After the initiation, the participants will share the private session name s, and the queue associated to s, which is initially empty. The variable y in each participant $\mathtt{p}$ will be replaced by the corresponding channel with role $s[\mathtt{p}]$. The output rules [Send], [Deleg] and [Sel] enqueue values, channels and labels, respectively, into the queue of the session s (in rule [Send], $e \downarrow v$ denotes the evaluation of the expression e to the value v). The input rules [Rcv], [SRcv] and [Branch] perform the corresponding complementary operations. Note that these operations check that the sender matches, and also that the message is actually meant for the receiver.

Processes are considered modulo structural equivalence, denoted by $\equiv$, and defined adding α-conversion to the rules in Table 4. By $r \notin \mathrm{fn}(Q)$ we mean that a is not a free name in Q if $r = a : G$ and that s is not a free name in Q if $r = s$. The meaning of $r \notin \mathrm{fn}(D)$ is similar. We denote by $\mathrm{dpv}(D)$ the set of process variables declared in D and by $\mathrm{fpv}(Q)$ the set of process variables which occur free in Q. Besides the standard rules (Milner, 1999), we have a rule for rearranging messages in a queue when the senders or the receivers are not the same, and a rule for splitting a message with more than one receiver.

4. Communication Type System for Pure Processes

This section introduces the communication type system for pure processes, by which we can check type soundness of the communications. This type system corresponds essentially to the one introduced in (Honda et al., 2008), but it is slightly simpler owing to the new formulation of the calculus. We need to introduce it here since we use it for defining progress property in next

S	::=	$\mathsf{bool} \mid \ldots \mid \mathsf{G}$	Sorts
U	::=	$S \mid \mathsf{T}$	Exchange types
	Global types		
G	::=	$\mathtt{p} \to \Pi : \langle S \rangle.G$	Value exchange
	$\mid$	$\mathtt{p} \to \mathtt{p} : \langle \mathsf{T} \rangle.G$	Channel exchange
	$\mid$	$\mathtt{p} \to \Pi : \{l_i : G_i\}_{i \in I}$	Branching
	$\mid$	$\mu \mathbf{t}.G \mid \mathbf{t} \mid \mathsf{end}$	Recursion/end
	Session types		
T	::=	$!\langle \Pi, S \rangle.T$	*send value*
	$\mid$	$!\langle \mathtt{p}, \mathsf{T} \rangle.T$	Send channel
	$\mid$	$?(\mathtt{p}, U).T$	Receive
	$\mid$	$\oplus \langle \Pi, \{l_i : T_i\}_{i \in I} \rangle$	Selection
	$\mid$	$\&(\mathtt{p}, \{l_i : T_i\}_{i \in I})$	Branching
	$\mid$	$\mu \mathbf{t}.T \mid \mathbf{t} \mid \mathsf{end}$	Recursion/end

Table 5. Global and session types.

§. Instead we give the typing rules for message queues and run time processes in Appendix A, since they are not central in our development.

4.1. *Global and Session Types*

Global types describe the whole conversation scenarios of multiparty session. *Session types* correspond to projections of global types on the individual participants: they are types of pure processes. The grammar for global and session types is given in Table 5. This grammar is slightly more permissive than necessary, in the sense that it allows session types that cannot be obtained as projections of global types. In practice, we are only interested in the subsets of *well-formed session types* (those that can be obtained as projections of well-formed global types) and *well-formed global types* (those that only contain well-formed session types). The formal notions of global type projection and well-formed global/session types will be given in Definitions 4.1 and 4.2.

Sorts $S, S', \ldots$ are associated to values (either base types or *closed* global types, ranged over by G). *Exchange types* $U, U', \ldots$ consist of sort types or *closed* session types, ranged over by T.

The global type $\mathtt{p} \to \Pi : \langle S \rangle.G$ says that participant $\mathtt{p}$ multicasts a value of sort S to the non-empty set of participants Π and then the interactions described in G take place. Similarly, the global type $\mathtt{p} \to \mathtt{q} : \langle \mathsf{T} \rangle.G$ says that participant $\mathtt{p} \neq \mathtt{q}$ delegates a channel of type T to participant $\mathtt{q}$ and the interaction continues according to G. Obviously only one receiver is expected in this case. When it does not matter we use $\mathtt{p} \to \Pi : \langle U \rangle.G$ to refer both to $\mathtt{p} \to \Pi : \langle S \rangle.G$ and $\mathtt{p} \to \mathtt{q} : \langle \mathsf{T} \rangle.G$. Type $\mathtt{p} \to \Pi : \{l_i : G_i\}_{i \in I}$ says participant $\mathtt{p}$ multicasts one of the labels l_i to the set of participants Π. If l_j is sent, interactions described in G_j take place. In both cases we assume $\mathtt{p} \notin \Pi$. Type $\mu \mathbf{t}.G$ is a recursive type, assuming type variables $(\mathbf{t}, \mathbf{t}', \ldots)$ are guarded in the standard way, i.e., type variables only appear under some prefix. We take an *equi-recursive* view of recursive types, not distinguishing between $\mu \mathbf{t}.G$ and its unfolding $G\{\mu \mathbf{t}.G/\mathbf{t}\}$ (Pierce, 2002, §21.8). Type end represents the termination of the session.

The *send types* $!\langle \Pi, S \rangle.T$, $!\langle \mathtt{p}, \mathsf{T} \rangle.T$express, respectively, the sending of a value of sort S to all participants in Π or the sending of a channel of type T to participant $\mathtt{p}$ followed by the

communications described by T. The *selection type* $\oplus\langle\Pi,\{l_i:T_i\}_{i\in I}\rangle$ represents the transmission to all participants in Π of a label l_i chosen in the set $\{l_i \mid i\in I\}$ followed by the communications described by T_i. The *receive* and *branching* types are dual of send and selection types: in them only one sender is considered. Recursion is guarded also in session types, and we consider them modulo fold/unfold as done for global types.

As in processes, when $\Pi=\{\mathtt{p}\}$ is a singleton we identify Π with $\mathtt{p}$.

The relation between global and session types is formalised by the notion of projection as in (Honda et al., 2008). We use this notion also for defining when global and session types are well formed.

Definition 4.1. The *projection of a global type G onto a participant* $\mathtt{q}$ ($G\upharpoonright\mathtt{q}$) is defined by induction on G:

$$(\mathtt{p}\to\Pi:\langle U\rangle.G')\upharpoonright\mathtt{q}=\begin{cases}!\langle\Pi,U\rangle.(G'\upharpoonright\mathtt{q}) & \text{if } \mathtt{q}=\mathtt{p},\\ ?(\mathtt{p},U).(G'\upharpoonright\mathtt{q}) & \text{if } \mathtt{q}\in\Pi,\\ G'\upharpoonright\mathtt{q} & \text{otherwise.}\end{cases}$$

$$(\mathtt{p}\to\Pi:\{l_i:G_i\}_{i\in I})\upharpoonright\mathtt{q}=\begin{cases}\oplus(\Pi,\{l_i:G_i\upharpoonright\mathtt{q}\}_{i\in I}) & \text{if } \mathtt{q}=\mathtt{p}\\ \&(\mathtt{p},\{l_i:G_i\upharpoonright\mathtt{q}\}_{i\in I}) & \text{if } \mathtt{q}\in\Pi\\ G_{i_0}\upharpoonright\mathtt{q} & \text{where } i_0\in I \text{ if } \mathtt{q}\neq\mathtt{p},\mathtt{q}\notin\Pi\\ & \text{and } G_i\upharpoonright\mathtt{q}=G_j\upharpoonright\mathtt{q} \text{ for all } i,j\in I.\end{cases}$$

$$(\mu\mathbf{t}.G)\upharpoonright\mathtt{q}=\begin{cases}\mu\mathbf{t}.(G\upharpoonright\mathtt{q}) & \text{if } G\upharpoonright\mathtt{q}\neq\mathbf{t},\\ \mathsf{end} & \text{otherwise.}\end{cases}\qquad \mathbf{t}\upharpoonright\mathtt{q}=\mathbf{t}\qquad \mathsf{end}\upharpoonright\mathtt{q}=\mathsf{end}.$$

As an example, we list two of the projections of the global types G_a and G_b of the three-buyer protocol in §2.

$$\begin{array}{lcl}\mathsf{G}_a\upharpoonright 3 & = & ?(2,\mathsf{string}).!\langle\{1,2\},\mathsf{int}\rangle;\&(1,\{\mathsf{ok}:?(1,\mathsf{string}).!\langle 1,\mathsf{date}\rangle.\mathsf{end},\mathsf{quit}:\mathsf{end}\})\\ \mathsf{G}_b\upharpoonright 1 & = & ?(2,\mathsf{int}).?(2,\mathsf{T}).\oplus\langle 2,\{\mathsf{ok}:\mathsf{end},\mathsf{quit}:\mathsf{end}\}\rangle\end{array}$$

where T is defined at page 3.

Well-formed global and session types can then be defined as the ones satisfying the following (mutually recursive) conditions:

Definition 4.2.

1 A global type is *well formed* if all session types occurring in it are well formed and closed.
2 A session type is *well formed* if it is the projection of some well-formed global type.

Notice that a global type without occurrences of session types (i.e. without channel exchanges) is always well formed. It is quite natural that when building a global type including the delegation of a channel of type T, the designer has already designed the global type G which includes the communications represented by T. In this case T is obtained from the projection of G onto one of its participants, assuring its well-formedness.

As an example, the global types G_a, G_b and the session type T introduced in §2 are all well formed. In fact G_a is well formed since it contains no session types, T is well formed since it

is the projection onto participant 1 of the global type defined by lines (4), (5), and (6) of the definition of G_a, and G_b is well formed since the exchanged type T is well formed.

According to the methodology first advocated in (Honda et al., 2008) and pursued in this work, a distributed system is first designed in terms of global types and then implemented as a set of processes respecting session types that are obtained as projections of such global types. For this reason, the notion of well-formed global/session type arises naturally and is not restrictive in such framework.

From now on we will implicitly make the assumption that all global and session types are well formed.

4.2. *Typing Rules for Pure Processes*

The typing judgements for expressions and pure processes are of the shapes:

$$\Gamma \vdash e : S \quad \text{and} \quad \Gamma \vdash P \triangleright \Delta$$

where

- Γ is the *standard environment* which associates variables to sort types, service names to closed global types and process variables to pairs of sort types and session types;
- Δ is the *session environment* which associates channels to session types.

Formally we define:

$$\Gamma ::= \emptyset \mid \Gamma, x : S \mid \Gamma, a : \mathsf{G} \mid \Gamma, X : S\,T \quad \text{and} \quad \Delta ::= \emptyset \mid \Delta, c : T$$

assuming that we can write $\Gamma, x : S$ only if $x \notin dom(\Gamma)$, where $dom(\Gamma)$ denotes the domain of Γ, i.e., the set of identifiers which occur in Γ. We use the same convention for $a : G$, $X : S\,T$ and $c{:}T$(thus we can write Δ, Δ' only if $dom(\Delta) \cap dom(\Delta') = \emptyset$).

Table 6 presents the typing rules for expressions and pure processes.

Rule (NAME) is standard: recall that u stands for x and a and S includes G.

Rule (MCAST) permits to type a request on a service identified by u, if the type of y is the $\mathtt{p}$-th projection of the global type G of u and the maximum participant in G (denoted by $\mathrm{mp}(\mathsf{G})$) is $\mathtt{p}$. Rule (MACC) permits to type the $\mathtt{p}$-th participant identified by u, which uses the channel y, if the type of y is the $\mathtt{p}$-th projection of the global type G of u and $\mathtt{p} < \mathrm{mp}(\mathsf{G})$.

In the typing of the example of the three-buyer protocol the types of the channels y in Seller and z in Carol are respectively the third projection of G_a and the first projection of G_b. By applying rule (MCAST) we can then derive $a : \mathsf{G}_a \vdash \text{Seller} \triangleright \emptyset$. Similarly by applying rule (MACC) we can derive $b : \mathsf{G}_b \vdash \text{Carol} \triangleright \emptyset$. (The processes Seller and Carol are defined in Table 1.)

The successive six rules associate the input/output processes to the input/output types in the expected way. For example we can derive:

$$\vdash t \oplus \langle \{2,3\}, \mathsf{ok} \rangle . t!\langle 3, \texttt{"Address"} \rangle ; t?(3, date).\mathbf{0} \triangleright \{t : \mathsf{T}\}$$

where $\mathsf{T} = \oplus\langle \{2,3\}, \{\mathsf{ok} : !\langle 3, \mathsf{string} \rangle . ?(3, \mathsf{date}).\mathsf{end},\ \mathsf{quit} : \mathsf{end}\} \rangle$. Note that, according to our notational convention on environments, in rule (DELEG) the channel which is sent cannot appear in the session environment of the premise, i.e., $c' \notin dom(\Delta) \cup \{c\}$.

Rule (PAR) permits to put in parallel two processes only if their session environments have disjoint domains.

$$\Gamma, u : S \vdash u : S \ (\textsc{Name}) \qquad \Gamma \vdash \mathsf{true}, \mathsf{false} : \mathsf{bool} \ (\textsc{Bool}) \qquad \frac{\Gamma \vdash e_i : \mathsf{bool} \qquad (i = 1, 2)}{\Gamma \vdash e_1 \ \mathsf{and} \ e_2 : \mathsf{bool}} \ (\textsc{And})$$

$$\frac{\Gamma \vdash u : \mathsf{G} \quad \Gamma \vdash P \triangleright \Delta, y : \mathsf{G} \upharpoonright \mathtt{p} \quad \mathtt{p} = \mathrm{mp}(\mathsf{G})}{\Gamma \vdash \overline{u}[\mathtt{p}](y).P \triangleright \Delta} \ (\textsc{MCast}) \qquad \frac{\Gamma \vdash u : \mathsf{G} \quad \Gamma \vdash P \triangleright \Delta, y : \mathsf{G} \upharpoonright \mathtt{p} \quad \mathtt{p} < \mathrm{mp}(\mathsf{G})}{\Gamma \vdash u[\mathtt{p}](y).P \triangleright \Delta} \ (\textsc{MAcc})$$

$$\frac{\Gamma \vdash e : S \quad \Gamma \vdash P \triangleright \Delta, c : T}{\Gamma \vdash c!\langle \Pi, e \rangle.P \triangleright \Delta, c : !\langle \Pi, S \rangle.T} \ (\textsc{Send}) \qquad \frac{\Gamma, x : S \vdash P \triangleright \Delta, c : T}{\Gamma \vdash c?(\mathtt{q}, x).P \triangleright \Delta, c : ?(\mathtt{q}, S).T} \ (\textsc{Rcv})$$

$$\frac{\Gamma \vdash P \triangleright \Delta, c : T}{\Gamma \vdash c!\langle\langle \mathtt{p}, c' \rangle\rangle.P \triangleright \Delta, c : !\langle \{\mathtt{p}\}, \mathsf{T} \rangle.T, c' : \mathsf{T}} \ (\textsc{Deleg}) \qquad \frac{\Gamma \vdash P \triangleright \Delta, c : T, y : \mathsf{T}}{\Gamma \vdash c?((\mathtt{q}, y)).P \triangleright \Delta, c : ?(\mathtt{q}, \mathsf{T}).T} \ (\textsc{Srcv})$$

$$\frac{\Gamma \vdash P \triangleright \Delta, c : T_j \quad j \in I}{\Gamma \vdash c \oplus \langle \Pi, l_j \rangle.P \triangleright \Delta, c : \oplus \langle \Pi, \{l_i : T_i\}_{i \in I} \rangle} \ (\textsc{Sel}) \qquad \frac{\Gamma \vdash P_i \triangleright \Delta, c : T_i \quad \forall i \in I}{\Gamma \vdash c \& (\mathtt{p}, \{l_i : P_i\}_{i \in I}) \triangleright \Delta, c : \& (\mathtt{p}, \{l_i : T_i\}_{i \in I})} \ (\textsc{Branch})$$

$$\frac{\Gamma \vdash P \triangleright \Delta \quad \Gamma \vdash Q \triangleright \Delta'}{\Gamma \vdash P \mid Q \triangleright \Delta, \Delta'} \ (\textsc{Par}) \qquad \frac{\Gamma \vdash e : \mathsf{bool} \quad \Gamma \vdash P \triangleright \Delta \quad \Gamma \vdash Q \triangleright \Delta}{\Gamma \vdash \mathsf{if} \ e \ \mathsf{then} \ P \ \mathsf{else} \ Q \triangleright \Delta} \ (\textsc{If})$$

$$\frac{\Delta \ \mathsf{end \ only}}{\Gamma \vdash \mathbf{0} \triangleright \Delta} \ (\textsc{Inact}) \qquad \frac{\Gamma, a : G \vdash P \triangleright \Delta}{\Gamma \vdash (\nu a : G)P \triangleright \Delta} \ (\textsc{NRes})$$

$$\frac{\Gamma \vdash e : S \quad \Delta \ \mathsf{end \ only}}{\Gamma, X : S \, T \vdash X\langle e, c \rangle \triangleright \Delta, c : T} \ (\textsc{Var}) \qquad \frac{\Gamma, X : S \, \mathbf{t}, x : S \vdash P \triangleright y : T \qquad \Gamma, X : S \, \mu\mathbf{t}.T \vdash Q \triangleright \Delta}{\Gamma \vdash \mathsf{def} \ X(x, y) = P \ \mathsf{in} \ Q \triangleright \Delta} \ (\textsc{Def})$$

Table 6. Typing rules for expressions and pure processes.

In rules (INACT) and (VAR) we take environments Δ which associate end to arbitrary channels, denoted by "Δ end only".

The present formulation of rule (DEF) forces to type process variables only with μ-types, while the formulation in (Bettini et al., 2008; Honda et al., 2008):

$$\frac{\Gamma, X : S \, T, x : S \vdash P \triangleright y : T \qquad \Gamma, X : S \, T \vdash Q \triangleright \Delta}{\Gamma \vdash \mathsf{def} \ X(x, y) = P \ \mathsf{in} \ Q \triangleright \Delta}$$

allows to type unguarded process variables with arbitrary types, which can be meaningless. For example with the more permissive rule we can derive $\vdash \mathsf{def} \ X(x, y) = X(x, y) \ \mathsf{in} \ X\langle \mathsf{true}, z \rangle \triangleright \{z : \mathsf{T}\}$ for an arbitrary closed T, while in our system we cannot type this process since its only possible type would be $\mu\mathbf{t}.\mathbf{t}$, which is not guarded and then forbidden.

4.3. *Subject Reduction*

We end this section by formulating subject reduction for closed user processes. We clearly need typing judgments for run time processes. In these judgments the turn style is decorated by sets of session names, which are the names of the current queues. Reducing a closed user process we obtain processes in which all session names are bound, so the turn style is decorated by the empty set.

Theorem 4.3 (Subject Reduction for Closed User Processes). If $\Gamma \vdash P \triangleright \emptyset$ and $P \longrightarrow^* P'$, then $\Gamma \vdash_\emptyset P' \triangleright \emptyset$.

Appendix A gives the typing rules for run time processes and the proof of subject reduction for arbitrary processes.

5. Related Work

Multiparty sessions. The first works on multiparty session types are (Bonelli and Compagnoni, 2008) and (Honda et al., 2008). The paper (Bonelli and Compagnoni, 2008) uses a distributed calculus where each channel connects a master endpoint to one or more slave endpoints; instead of global types, they solely use (recursion-free) local types. For type checking, local types are projected to binary sessions, so that type safety is ensured using duality, but it loses sequencing information: hence progress in a session interleaved with other sessions is not guaranteed.

The present calculus is an essential improvement and simplification of (Honda et al., 2008): both processes and types in (Honda et al., 2008) share a vector of channels and each communication uses one of these channels. In the present work, processes and types use indexes for denoting the participants of a session. The new communication type system improves the one of (Honda et al., 2008) in three main technical points without sacrificing its expressivity. First, it avoids the overhead of global linearity-check in (Honda et al., 2008) because our global types automatically satisfy the linearity condition in (Honda et al., 2008) due to the limitation to bi-directional channel communications. Second, it provides a more liberal policy in the use of variables in delegation since we do not require to delegate a set of session channels. Finally, it implicitly provides each participant of a service with a runtime channel indexed by its role on which he can communicate with all other participants, therefore enabling broadcast communication in a natural way. The use of indexed channels, moreover, allows to define light-weight interaction type system. The global types in (Honda et al., 2008) have a parallel composition operator, but its projectability from global to local types limits to disjoint senders and receivers; hence our global types do not affect the expressivity.

Further works on multiparty session types include: Java protocol optimisation (Sivaramakrishnan et al., 2010), a generation of multiparty cryptographic protocols (Bhargavan et al., 2009), asynchronous commutative multiparty session types for a refinement (Mostrous et al., 2009), parametrised global types for parallel programming and Web service descriptions (Deniélou et al., 2012), access control and secrecy (Capecchi et al., 2010a), communication buffered analysis (Deniélou and Yoshida, 2010), extensions to the sumtype and its encoding (Nielsen et al., 2010), applications to Healthcare (Henriksen et al., 2013) and exception handling for multiparty conversations (Capecchi et al., 2010b). Multiparty session types can be extended with logical assertions following design by contract framework (Bocchi et al., 2010). A recent work (Chen and Honda, 2012) offers more fine-grained property analysis for multiparty session types with stateful logical assertions. The inference of global types from a set of local types is studied in (Lange and Tuosto, 2012). In (Deniélou and Yoshida, 2011) roles are inhabited by an arbitrary number of participants which can dynamically join and leave. The paper (Swamy et al., 2011) shows that the multirole session types (Deniélou and Yoshida, 2011) can be naturally represented in a dependent-typed language. To enhance expressivity and flexibility of multiparty session types,

the work (Demangeon and Honda, 2012) proposes nested, higher-order multiparty session types and the work (Castagna et al., 2012) studies a generalisation of choices and parallelism. The paper (Carbone and Montesi, 2013) directly types a global description language (Carbone et al., 2012) by multiparty session types without using local types. This direct approach can type processes which are untypable in the original multiparty session types (i.e. the communication typing system in this article). The paper (Montesi and Yoshida, 2013) extends the work in (Carbone and Montesi, 2013) to compositional global languages. The work (Deniélou and Yoshida, 2012) gives a linkage between communicating automata (Brand and Zafiropulo, 1983) and a general graphical version of multiparty session types, proving a one-to-one correspondence between the properties of communicating automata and multiparty session types. The paper (Deniélou and Yoshida, 2013) studies the sound and complete characterisation of the multiparty session types in communicating automata and applies the result to the synthesis of the multiparty session types. The work (Kouzapas and Yoshida, 2013) shows semantics effects of the multiparty session types in the context of typed bisimulations and reduction-closed theories.

Implementations based on Multiparty Session Types. We are currently designing and implementing a modelling and specification language with multiparty session types (Savara, 2010; Scribble, 2008) for the standards of business and financial protocols with our industry collaborators (Honda et al., 2011; Honda et al., 2013). We also apply the multiparty session types to parallel programming in C (Ng et al., 2012) and MPI (Ng and Yoshida, 2014). An artcle (Yoshida et al., 2013) also explains the origin and recent development on Scribble.

Not only the static type checking, we are implementing a runtime monitoring to collaborate with Ocean Observatories Initiative (OOI, 2010). The correctness of the monitoring implementation is proved in (Bocchi et al., 2013). We also extend the Python implementation to the Actor framework (Neykova and Yoshida, 2014).

References

Bettini, L., Coppo, M., D'Antoni, L., Luca, M. D., Dezani-Ciancaglini, M., and Yoshida, N. (2008). Global Progress in Dynamically Interleaved Multiparty Sessions. In *CONCUR'08*, volume 5201 of *LNCS*, pages 418–433. Springer.

Bhargavan, K., Corin, R., Deniélou, P.-M., Fournet, C., and Leifer, J. J. (2009). Cryptographic Protocol Synthesis and Verification for Multiparty Sessions. In *CSF'09*, pages 124–140. IEEE Computer Society Press.

Bocchi, L., Chen, T.-C., Demangeon, R., Honda, K., and Yoshida, N. (2013). Monitoring networks through multiparty session types. In *FMOODS/FORTE 2013*, volume 7892 of *LNCS*, pages 50–65. Springer.

Bocchi, L., Honda, K., Tuosto, E., and Yoshida, N. (2010). A Theory of Design-by-Contract for Distributed Multiparty Interactions. In *CONCUR'10*, volume 6269 of *LNCS*, pages 162–176. Springer.

Bonelli, E. and Compagnoni, A. (2008). Multipoint Session Types for a Distributed Calculus. In *TGC'07*, volume 4912 of *LNCS*, pages 240–256. Springer.

Brand, D. and Zafiropulo, P. (1983). On Communicating Finite-State Machines. *Journal of the ACM*, 30:323–342.

Capecchi, S., Castellani, I., Dezani-Ciancaglini, M., and Rezk, T. (2010a). Session Types for Access and Information Flow Control. In *CONCUR'10*, volume 6269 of *LNCS*, pages 237–252. Springer.

Capecchi, S., Giachino, E., and Yoshida, N. (2010b). Global Escape in Multiparty Sessions. In *FSTTCS'10*, volume 8 of *LIPIcs*, pages 338–351. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

Carbone, M., Honda, K., and Yoshida, N. (2012). Structured Communication-Centered Programming for Web Services. *ACM Transactions on Programming Languages and Systems*, 34(2):8.

Carbone, M. and Montesi, F. (2013). Deadlock-freedom-by-design: Multiparty Asynchronous Global Programming. In *POPL'13*, pages 263–274. ACM.

Castagna, G., Dezani-Ciancaglini, M., and Padovani, L. (2012). On Global Types and Multi-Party Session. *Logical Methods in Computer Science*, 8(1):24.

Chen, T.-C. and Honda, K. (2012). Specifying Stateful Asynchronous Properties for Distributed Programs. In *CONCUR'12*, volume 7454 of *LNCS*, pages 209–224. Springer.

Coppo, M., Dezani-Ciancaglini, M., Yoshida, N., and Padovani, L. (2014). Global progress for dynamically interleaved multiparty sessions. *MSCS*.

Demangeon, R. and Honda, K. (2012). Nested Protocols in Session Types. In *CONCUR'12*, volume 7454 of *LNCS*, pages 272–286. Springer.

Deniélou, P.-M. and Yoshida, N. (2010). Buffered Communication Analysis in Distributed Multiparty Sessions. In *CONCUR'10*, volume 6269 of *LNCS*, pages 343–357. Springer.

Deniélou, P.-M. and Yoshida, N. (2011). Dynamic Multirole Session Types. In *POPL'11*, pages 435–446. ACM Press.

Deniélou, P.-M. and Yoshida, N. (2012). Multiparty Session Types Meet Communicating Automata. In *ESOP'12*, volume 7211 of *LNCS*, pages 194–213. Springer.

Deniélou, P.-M. and Yoshida, N. (2013). Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. In *ICALP'13*, volume 7966 of *LNCS*, pages 174–186. Springer.

Deniélou, P.-M., Yoshida, N., Bejleri, A., and Hu, R. (2012). Parameterised Multiparty Session Types. *Logical Methods in Computer Science*, 8(4).

Henriksen, A., Nielsen, L., Hildebrandt, T., Yoshida, N., , and Henglein, F. (2013). Trustworthy Pervasive Healthcare Services via Multi-party Session Type. In *FHIES'12*, volume 7789 of *LNCS*, pages 124–141.

Honda, K., Hu, R., Neykova, R., Chen, T.-C., Demangeon, R., Deniélou, P.-M., and Yoshida, N. (2013). Structuring Communication with Session Types. In *COB'12*, LNCS. Springer. To appear.

Honda, K., Mukhamedov, A., Brown, G., Chen, T.-C., and Yoshida, N. (2011). Scribbling Interactions with a Formal Foundation. In *ICDCIT'11*, volume 6536 of *LNCS*, pages 55–75. Springer.

Honda, K., Yoshida, N., and Carbone, M. (2008). Multiparty Asynchronous Session Types. In *POPL'08*, pages 273–284. ACM Press.

Kouzapas, D. and Yoshida, N. (2013). Governed Session Semantics. In *CONCUR'13*, LNCS. Springer. To appear.

Lange, J. and Tuosto, E. (2012). Synthesising Choreographies from Local Session Types. In *CONCUR'12*, volume 7454 of *LNCS*, pages 225–239. Springer.

Milner, R. (1999). *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press.

Montesi, F. and Yoshida, N. (2013). Compositional Choreographies. In *CONCUR'13*, LNCS. Springer. To appear.

Mostrous, D., Yoshida, N., and Honda, K. (2009). Global Principal Typing in Partially Commutative Asynchronous Sessions. In *ESOP'09*, volume 5502 of *LNCS*, pages 316–332. Springer.

Neykova, R. and Yoshida, N. (2014). Multiparty session actors. In *COORDINATION 2014*, volume 8459 of *LNCS*. Springer.

Ng, N. and Yoshida, N. (2014). Pabble: Parameterised scribble for parallel programming. In *PDP 2014*, pages 707–714. IEEE Computer Society.

Ng, N., Yoshida, N., and Honda, K. (2012). Multiparty Session C: Safe Parallel Programming with Message Optimisation. In *TOOLS'12*, volume 7304 of *LNCS*, pages 202–218. Springer.

Nielsen, L., Yoshida, N., and Honda, K. (2010). Multiparty Symmetric Sum Types. In *EXPRESS'10*, volume 41 of *EPTCS*, pages 121–135.

OOI (2010). Ocean Observatories Initiative. `http://www.oceanleadership.org/programs-and-partnerships/ocean-observing/ooi/`.

Pierce, B. C. (2002). *Types and Programming Languages*. MIT Press.

Savara (2010). SAVARA JBoss RedHat Project. `http://www.jboss.org/savara`.

Scribble (2008). Scribble JBoss RedHat Project. `http://www.jboss.org/scribble`.

Sivaramakrishnan, K. C., Nagaraj, K., Ziarek, L., and Eugster, P. (2010). Efficient Session Type Guided Distributed Interaction. In *COORDINATION'10*, volume 6116 of *LNCS*, pages 152–167. Springer.

Swamy, N., Chen, J., Fournet, C., Strub, P.-Y., Bhargavan, K., and Yang, J. (2011). Secure Distributed Programming with Value-Dependent Types. In *ICFP'11*, pages 266–278. ACM Press.

UNIFI (2002). International Organization for Standardization ISO 20022 UNIversal Financial Industry message scheme. `http://www.iso20022.org`.

Web Services Choreography Working Group (2002). Web Services Choreography Description Language. http://www.w3.org/2002/ws/chor/.

Yoshida, N., Hu, R., Neykova, R., and Ng, N. (2013). The scribble protocol language. In *TGC 2013*, volume 8358 of *LNCS*, pages 22–41. Springer.

Appendix A. Communication Type System for Processes and its Properties

This appendix completes the description of the communication type system given in §4. §A.1 starts with typing rules for run time processes. Auxiliary lemmas, in particular inversion lemmas, are the content of §A.2. Lastly §A.3 formulates subject reduction for arbitrary processes and proves it.

A.1. *Types and Typing Rules for Processes*

We now extend the communication type system to processes containing queues.

$$\begin{array}{llll} \text{Message Types} & M & ::= & !\langle \Pi, U\rangle \quad \textit{message send} \\ & & \mid & \oplus\langle \Pi, l\rangle \quad \textit{message selection} \\ & & \mid & M;M \quad \textit{message sequence} \\ \\ \text{Generalised} & \tau & ::= & T \quad \textit{session} \\ & & \mid & M \quad \textit{message} \\ & & \mid & M;T \quad \textit{continuation} \end{array}$$

Message types are the types for queues: they represent the messages contained in the queues. The *message send type* $!\langle \Pi, U\rangle$ expresses the presence in a queue of an element of type U to be communicated to all participants in Π. The *message selection type* $\oplus\langle \Pi, l\rangle$ represents the communication to all participants in Π of the label l and $M;M$ represents sequencing of message types (we assume associativity for ";"). For example $\oplus\langle \{1,3\}, \mathsf{ok}\rangle$ is the message type for the message $(2, \{1,3\}, \mathsf{ok})$.

A *generalised type* is either a session type, or a message type, or a message type followed by a session type. Type $M;T$ represents the continuation of the type M associated to a queue with the type T associated to a pure process. Examples of generalised types are $\oplus\langle \{1,3\}, \mathsf{ok}\rangle; !\langle 3, \mathsf{string}\rangle.?(3, \mathsf{date}).\mathsf{end}$ and $\oplus\langle \{1,3\}, \mathsf{ok}\rangle; !\langle 3, \mathsf{string}\rangle; ?(3, \mathsf{date}).\mathsf{end}$, which only differ for the replacement of the leftmost "." by ";". In the first the type $!\langle 3, \mathsf{string}\rangle$ corresponds

$$\frac{}{\Gamma \vdash_{\{s\}} s : \varnothing \triangleright \emptyset}\ (\text{QINIT}) \qquad \frac{\Gamma \vdash_{\{s\}} s : h \triangleright \Delta \qquad \Gamma \vdash v : S}{\Gamma \vdash_{\{s\}} s : h \cdot (\mathsf{q}, \Pi, v) \triangleright \Delta; \{s[\mathsf{q}] : !\langle \Pi, S\rangle\}}\ (\text{QSEND})$$

$$\frac{\Gamma \vdash_{\{s\}} s : h \triangleright \Delta}{\Gamma \vdash_{\{s\}} s : h \cdot (\mathsf{q}, \mathsf{p}, s'[\mathsf{p}']) \triangleright (\Delta; \{s[\mathsf{q}] : !\langle \mathsf{p}, \mathsf{T}\rangle\}), s'[\mathsf{p}'] : \mathsf{T}}\ (\text{QDELEG})$$

$$\frac{\Gamma \vdash_{\{s\}} s : h \triangleright \Delta}{\Gamma \vdash_{\{s\}} s : h \cdot (\mathsf{q}, \Pi, l) \triangleright \Delta; \{s[\mathsf{q}] : \oplus\langle \Pi, l\rangle\}}\ (\text{QSEL})$$

Table 7. Typing rules for queues.

to an output action sending a string to participant 3, while in the second type $!\langle 3, \mathsf{string}\rangle$ corresponds to a message for participant 3 with a value of type string. See the examples of typing judgments at the end of this §.

We start by defining the typing rules for single queues, in which the turnstile $\vdash$ is decorated with $\{s\}$ (where s is the session name of the current queue) and the session environments are mappings from channels to message types. The empty queue has the empty session environment. Each message adds an output type to the current type of the channel which has the role of the message sender. Table 7 lists the typing rules for queues, where all types in session environments are message types. The operator ";" between an arbitrary session environment and a session environment containing only one association is defined by:

$$\Delta; \{s[\mathsf{q}] : M\} = \begin{cases} \Delta', s[\mathsf{q}] : M'; M & \text{if } \Delta = \Delta', s[\mathsf{q}] : M', \\ \Delta, s[\mathsf{q}] : M & \text{otherwise.} \end{cases}$$

For example we can derive $\vdash_{\{s\}} s : (3, \{1,2\}, \mathsf{ok}) \triangleright \{s[3] : \oplus\langle \{1,2\}, \mathsf{ok}\rangle\}$.

For typing pure processes in parallel with queues, we need to use generalised types in session environments and to add further typing rules.

In order to take into account the structural congruence between queues (see Table 4) we consider message types modulo the equivalence relation $\approx$ induced by the rules shown in Table 8 (with $\natural \in \{!, \oplus\}$ and $Z \in \{U, l\}$).

The equivalence relation on message types extends to generalised types by:

$$M \approx M' \text{ implies } M; \tau \approx M'; \tau$$

We say that two session environments Δ and Δ' are equivalent (notation $\Delta \approx \Delta'$) if $c : \tau \in \Delta$ and $\tau \neq \mathsf{end}$ imply $c : \tau' \in \Delta'$ with $\tau \approx \tau'$ and vice versa. The reason for ignoring end types is that rules (INACT) and (VAR) allow to freely introduce them.

In composing two session environments we want to put in sequence a message type and a session type for the same channel with role. For this reason we define the partial composition $*$ between generalised types as:

- $M; \natural\langle \Pi, Z\rangle; \natural'\langle \Pi', Z\rangle; M' \approx M; \natural'\langle \Pi', Z\rangle; \natural\langle \Pi, Z\rangle; M' \quad \text{if } \Pi \cap \Pi' = \emptyset$
- $M; \natural\langle \Pi, Z\rangle; M' \approx M; \natural\langle \Pi', Z\rangle; \natural\langle \Pi'', Z\rangle; M' \quad \text{if } \Pi = \Pi' \cup \Pi'', \Pi' \cap \Pi'' = \emptyset$

Table 8. Equivalence relation on message types.

$$\frac{\Gamma \vdash P \triangleright \Delta}{\Gamma \vdash_{\emptyset} P \triangleright \Delta}\ (\text{GINIT}) \qquad \frac{\Gamma \vdash_{\Sigma} P \triangleright \Delta \quad \Delta \approx \Delta'}{\Gamma \vdash_{\Sigma} P \triangleright \Delta'}\ (\text{EQUIV}) \qquad \frac{\Gamma \vdash_{\Sigma} P \triangleright \Delta \quad \Gamma \vdash_{\Sigma'} Q \triangleright \Delta' \quad \Sigma \cap \Sigma' = \emptyset}{\Gamma \vdash_{\Sigma \cup \Sigma'} P \mid Q \triangleright \Delta * \Delta'}\ (\text{GPAR})$$

$$\frac{\Gamma \vdash_{\Sigma} P \triangleright \Delta}{\Gamma \vdash_{\Sigma \setminus s} (\nu s) P \triangleright \Delta \setminus s}\ (\text{GSRES}) \qquad \frac{\Gamma, a : \mathsf{G} \vdash_{\Sigma} P \triangleright \Delta}{\Gamma \vdash_{\Sigma} (\nu a : \mathsf{G}) P \triangleright \Delta}\ (\text{GNRES})$$

$$\frac{\Gamma, X : S\ \mathbf{t}, x : S \vdash P \triangleright \{y : T\} \qquad \Gamma, X : S\ \mu \mathbf{t}.T \vdash_{\Sigma} Q \triangleright \Delta}{\Gamma \vdash_{\Sigma} \mathsf{def}\ X(x,y) = P\ \mathsf{in}\ Q \triangleright \Delta}\ (\text{GDEF})$$

Table 9. Typing rules for processes.

$$\tau * \tau' = \begin{cases} \tau ; \tau' & \text{if } \tau \text{ is a message type,} \\ \tau' ; \tau & \text{if } \tau' \text{ is a message type.} \end{cases}$$

Notice that $\tau * \tau'$ is defined only if at least one between τ and τ' is a message type.

We extend $*$ to session environments as expected:

$$\Delta * \Delta' = \Delta \backslash dom(\Delta') \cup \Delta' \backslash dom(\Delta) \cup \{c : \tau * \tau' \mid c : \tau \in \Delta \wedge c : \tau' \in \Delta'\}.$$

Note that $*$ is commutative, i.e., $\Delta * \Delta' = \Delta' * \Delta$. Also if we can derive message types only for channels with roles, we consider channel variables in the definition of $*$ for session environments since we want to get for example that $\{y : \mathsf{end}\} * \{y : \mathsf{end}\}$ is undefined (message types do not contain end).

Table 9 lists the typing rules for processes containing queues. The judgement $\Gamma \vdash_{\Sigma} P \triangleright \Delta$ means that P contains the queues whose session names are in Σ. Rule (GINIT) promotes the typing of a pure process to the typing of an arbitrary process without session names, since a pure process does not contain queues. When two arbitrary processes are put in parallel (rule (GPAR)) we need to require that each session name is associated to at most one queue (condition $\Sigma \cap \Sigma' = \emptyset$).

Examples of derivable judgements are:

$$\vdash_{\{s\}} P \mid s : (3, \{1,2\}, \mathsf{ok}) \triangleright \{s[3] : \oplus\langle\{1,2\}, \mathsf{ok}\rangle ; !\langle 1, \mathsf{string}\rangle.?(1, \mathsf{date}).\mathsf{end}\}$$

where $P = s[3]!\langle 1, \texttt{"Address"}\rangle ; s[3]?(1, date) ; \mathbf{0}$ and

$$\vdash_{\{s\}} P' \mid s : (3, \{1,2\}, \mathsf{ok}) \cdot (3, 1, \texttt{"Address"}) \triangleright \{s[3] : \oplus\langle\{1,2\}, \mathsf{ok}\rangle ; !\langle 1, \mathsf{string}\rangle ; ?(1, \mathsf{date}).\mathsf{end}\}$$

where $P' = s[3]?(1, date) ; \mathbf{0}$. Note that

$$P \mid s : (3, \{1,2\}, \mathsf{ok}) \longrightarrow P' \mid s : (3, \{1,2\}, \mathsf{ok}) \cdot (3, 1, \texttt{"Address"})$$

A.2. *Auxiliary Lemmas*

We start with inversion lemmas which can be easily shown by induction on derivations.

Lemma A.1 (Inversion Lemma for Pure Processes).

1 If $\Gamma \vdash u : S$, then $u : S \in \Gamma$.
2 If $\Gamma \vdash \mathsf{true} : S$, then $S = \mathsf{bool}$.
3 If $\Gamma \vdash \mathsf{false} : S$, then $S = \mathsf{bool}$.
4 If $\Gamma \vdash e_1$ and $e_2 : S$, then $\Gamma \vdash e_1 : \mathsf{bool}$ and $\Gamma \vdash e_2 : \mathsf{bool}$ and $S = \mathsf{bool}$.

5 If $\Gamma \vdash \overline{a}[\mathtt{p}](y).P \triangleright \Delta$, then $\Gamma \vdash a : \mathsf{G}$ and $\Gamma \vdash P \triangleright \Delta, y : \mathsf{G} \upharpoonright \mathtt{p}$ and $\mathtt{p} = \mathrm{mp}(\mathsf{G})$.
6 If $\Gamma \vdash a[\mathtt{p}](y).P \triangleright \Delta$, then $\Gamma \vdash a : \mathsf{G}$ and $\Gamma \vdash P \triangleright \Delta, y : \mathsf{G} \upharpoonright \mathtt{p}$ and $\mathtt{p} < \mathrm{mp}(\mathsf{G})$.
7 If $\Gamma \vdash c!\langle \Pi, e\rangle.P \triangleright \Delta$, then $\Delta = \Delta', c : !\langle \Pi, S\rangle.T$ and $\Gamma \vdash e : S$ and $\Gamma \vdash P \triangleright \Delta', c : T$.
8 If $\Gamma \vdash c?(\mathtt{q}, x).P \triangleright \Delta$, then $\Delta = \Delta', c : ?(\mathtt{q}, S).T$ and $\Gamma, x : S \vdash P \triangleright \Delta', c : T$.
9 If $\Gamma \vdash c!\langle\langle \mathtt{p}, c'\rangle\rangle.P \triangleright \Delta$, then $\Delta = \Delta', c : !\langle \mathtt{p}, \mathsf{T}\rangle.T, c' : \mathsf{T}$ and $\Gamma \vdash P \triangleright \Delta', c : T$.
10 If $\Gamma \vdash c?((\mathtt{q}, y)).P \triangleright \Delta$, then $\Delta = \Delta', c : ?(\mathtt{q}, \mathsf{T}).T$ and $\Gamma \vdash P \triangleright \Delta', c : T, y : \mathsf{T}$.
11 If $\Gamma \vdash c \oplus \langle \Pi, l_j\rangle.P \triangleright \Delta$, then $\Delta = \Delta', c : \oplus\langle \Pi, \{l_i : T_i\}_{i \in I}\rangle$ and $\Gamma \vdash P \triangleright \Delta', c : T_j$ and $j \in I$.
12 If $\Gamma \vdash c\&(\mathtt{p}, \{l_i : P_i\}_{i \in I}) \triangleright \Delta$, then $\Delta = \Delta', c : \&(\mathtt{p}, \{l_i : T_i\}_{i \in I})$ and $\Gamma \vdash P_i \triangleright \Delta', c : T_i \quad \forall i \in I$.
13 If $\Gamma \vdash P \mid Q \triangleright \Delta$, then $\Delta = \Delta', \Delta''$ and $\Gamma \vdash P \triangleright \Delta'$ and $\Gamma \vdash Q \triangleright \Delta''$.
14 If $\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta$, then $\Gamma \vdash e : \mathsf{bool}$ and $\Gamma \vdash P \triangleright \Delta$ and $\Gamma \vdash Q \triangleright \Delta$.
15 If $\Gamma \vdash \mathbf{0} \triangleright \Delta$, then Δ end only.
16 If $\Gamma \vdash (\nu a : \mathsf{G})P \triangleright \Delta$, then $\Gamma, a : \mathsf{G} \vdash P \triangleright \Delta$.
17 If $\Gamma \vdash X\langle e, c\rangle \triangleright \Delta$, then $\Gamma = \Gamma', X : S\ T$ and $\Delta = \Delta', c : T$ and $\Gamma \vdash e : S$ and Δ' end only.
18 If $\Gamma \vdash \mathsf{def}\ X(x, y) = P\ \mathsf{in}\ Q \triangleright \Delta$, then $\Gamma, X : S\ \mathbf{t}, x : S \vdash P \triangleright \{y : T\}$ and $\Gamma, X : S\ \mu\mathbf{t}.T \vdash Q \triangleright \Delta$.

Lemma A.2 (Inversion Lemma for Processes).

1 If $\Gamma \vdash_\Sigma P \triangleright \Delta$ and P is a pure process, then $\Sigma = \emptyset$ and $\Gamma \vdash P \triangleright \Delta$.
2 If $\Gamma \vdash_\Sigma s : h \triangleright \Delta$, then $\Sigma = \{s\}$.
3 If $\Gamma \vdash_{\{s\}} s : \varnothing \triangleright \Delta$, then Δ end only.
4 If $\Gamma \vdash_{\{s\}} s : h \cdot (\mathtt{q}, \Pi, v) \triangleright \Delta$, then $\Delta \approx \Delta'; \{s[\mathtt{q}] : !\langle \Pi, S\rangle\}$ and $\Gamma \vdash_{\{s\}} s : h \triangleright \Delta'$ and $\Gamma \vdash v : S$.
5 If $\Gamma \vdash_{\{s\}} s : h \cdot (\mathtt{q}, \mathtt{p}, s'[\mathtt{p}']) \triangleright \Delta$, then $\Delta \approx (\Delta'; \{s[\mathtt{q}] : !\langle \mathtt{p}, \mathsf{T}\rangle\}), s'[\mathtt{p}'] : \mathsf{T}$ and $\Gamma \vdash_{\{s\}} s : h \triangleright \Delta'$.
6 If $\Gamma \vdash_{\{s\}} s : h \cdot (\mathtt{q}, \Pi, l) \triangleright \Delta$, then $\Delta \approx \Delta'; \{s[\mathtt{q}] : \oplus\langle \Pi, l\rangle\}$ and $\Gamma \vdash_{\{s\}} s : h \triangleright \Delta'$.
7 If $\Gamma \vdash_\Sigma P \mid Q \triangleright \Delta$, then $\Sigma = \Sigma_1 \cup \Sigma_2$ and $\Sigma_1 \cap \Sigma_2 = \emptyset$ and $\Delta = \Delta_1 * \Delta_2$ and $\Gamma \vdash_{\Sigma_1} P \triangleright \Delta_1$ and $\Gamma \vdash_{\Sigma_2} Q \triangleright \Delta_2$.
8 If $\Gamma \vdash_\Sigma (\nu s)P \triangleright \Delta$, then $\Sigma = \Sigma' \setminus s$ and $\Delta = \Delta' \setminus s$ and $\mathsf{co}(\Delta', s)$ and $\Gamma \vdash_{\Sigma'} P \triangleright \Delta'$.
9 If $\Gamma \vdash_\Sigma (\nu a : \mathsf{G})P \triangleright \Delta$, then $\Gamma, a : \mathsf{G} \vdash_\Sigma P \triangleright \Delta$.
10 If $\Gamma \vdash_\Sigma \mathsf{def}\ X(x, y) = P\ \mathsf{in}\ Q \triangleright \Delta$, then $\Gamma, X : S\ \mathbf{t}, x : S \vdash P \triangleright y : T$ and $\Gamma, X : S\ \mu\mathbf{t}.T \vdash_\Sigma Q \triangleright \Delta$.

The following lemma allows to characterise the types due to the messages which occur in queues. The proof is standard by induction on the lengths of queues.

Lemma A.3.

1 If $\Gamma \vdash_{\{s\}} s : h_1 \cdot (\mathtt{q}, \Pi, v) \cdot h_2 \triangleright \Delta$, then $\Delta = \Delta_1 * \{s[\mathtt{q}] : !\langle \Pi, S\rangle\} * \Delta_2$ and $\Gamma \vdash_{\{s\}} s : h_i \triangleright \Delta_i$ $(i = 1, 2)$ and $\Gamma \vdash v : S$.
Vice versa $\Gamma \vdash_{\{s\}} s : h_i \triangleright \Delta_i$ $(i = 1, 2)$ and $\Gamma \vdash v : S$ imply
$\Gamma \vdash_{\{s\}} s : h_1 \cdot (\mathtt{q}, \Pi, v) \cdot h_2 \triangleright \Delta_1 * \{s[\mathtt{q}] : !\langle \Pi, S\rangle\} * \Delta_2$.
2 If $\Gamma \vdash_{\{s\}} s : h_1 \cdot (\mathtt{q}, \mathtt{p}, s'[\mathtt{p}']) \cdot h_2 \triangleright \Delta$, then $\Delta = (\Delta_1 * \{s[\mathtt{q}] : !\langle \mathtt{p}, \mathsf{T}\rangle\} * \Delta_2), s'[\mathtt{p}'] : \mathsf{T}$ and $\Gamma \vdash_{\{s\}} s : h_i \triangleright \Delta_i$ $(i = 1, 2)$.
Vice versa $\Gamma \vdash_{\{s\}} s : h_i \triangleright \Delta_i$ $(i = 1, 2)$ imply
$\Gamma \vdash_{\{s\}} s : h_1 \cdot (\mathtt{q}, \mathtt{p}, s'[\mathtt{p}']) \cdot h_2 \triangleright (\Delta_1 * \{s[\mathtt{q}] : !\langle \mathtt{p}, \mathsf{T}\rangle\} * \Delta_2), s'[\mathtt{p}'] : \mathsf{T}$.
3 If $\Gamma \vdash_{\{s\}} s : h_1 \cdot (\mathtt{q}, \Pi, l) \cdot h_2 \triangleright \Delta$, then $\Delta = \Delta_1 * \{s[\mathtt{q}] : \oplus\langle \Pi, l\rangle\} * \Delta_2$ and $\Gamma \vdash_{\{s\}} s : h_i \triangleright \Delta_i$ $(i = 1, 2)$.
Vice versa $\Gamma \vdash_{\{s\}} s : h_i \triangleright \Delta_i$ $(i = 1, 2)$ imply
$\Gamma \vdash_{\{s\}} s : h_1 \cdot (\mathtt{q}, \Pi, l) \cdot h_2 \triangleright \Delta_1 * \{s[\mathtt{q}] : \oplus\langle \Pi, l\rangle\} * \Delta_2$.

We end this § with two classical results: type preservation under substitution and under equivalence of processes.

Lemma A.4 (Substitution lemma).

1 If $\Gamma, x : S \vdash P \triangleright \Delta$ and $\Gamma \vdash v : S$, then $\Gamma \vdash P\{v/x\} \triangleright \Delta$.
2 If $\Gamma \vdash P \triangleright \Delta, y : T$, then $\Gamma \vdash P\{s[\mathtt{p}]/y\} \triangleright \Delta, s[\mathtt{p}] : T$.

Proof. Standard induction on type derivations, with a case analysis on the last applied rule. □

Theorem A.5 (Type Preservation under Equivalence). If $\Gamma \vdash_\Sigma P \triangleright \Delta$ and $P \equiv P'$, then $\Gamma \vdash_\Sigma P' \triangleright \Delta$.

Proof. By induction on $\equiv$. We only consider some interesting cases (the other cases are straightforward).

— $P \mid \mathbf{0} \equiv P$. First we assume $\Gamma \vdash_\Sigma P \triangleright \Delta$. From $\Gamma \vdash_\emptyset \mathbf{0} \triangleright \emptyset$ by applying (GPAR) to these two sequents we obtain $\Gamma \vdash_\Sigma P|\mathbf{0} \triangleright \Delta$.
For the converse direction assume $\Gamma \vdash_\Sigma P|\mathbf{0} \triangleright \Delta$. Using A.2(7) we obtain: $\Gamma \vdash_{\Sigma_1} P \triangleright \Delta_1$, $\Gamma \vdash_{\Sigma_2} \mathbf{0} \triangleright \Delta_2$, where $\Delta = \Delta_1 * \Delta_2$, $\Sigma = \Sigma_1 \cup \Sigma_2$ and $\Sigma_1 \cap \Sigma_2 = \emptyset$. Using A.2(1) we get $\Sigma_2 = \emptyset$, which implies $\Sigma = \Sigma_1$, and $\Gamma \vdash \mathbf{0} \triangleright \Delta_2$. Using A.1(15) we get Δ_2 end only which implies $\Delta_1 \approx \Delta_1 * \Delta_2$, so we conclude $\Gamma \vdash_\Sigma P \triangleright \Delta_1 * \Delta_2$ by applying (EQUIV).

— $P \mid Q \equiv Q \mid P$. By the symmetry of the rule we have to show only one direction. Suppose $\Gamma \vdash_\Sigma P \mid Q \triangleright \Delta$. Using A.2(7) we obtain $\Gamma \vdash_{\Sigma_1} P \triangleright \Delta_1$, $\Gamma \vdash_{\Sigma_2} Q \triangleright \Delta_2$, where $\Delta = \Delta_1 * \Delta_2$, $\Sigma = \Sigma_1 \cup \Sigma_2$ and $\Sigma_1 \cap \Sigma_2 = \emptyset$. Using (GPAR) we get $\Gamma \vdash_\Sigma Q \mid P \triangleright \Delta_2 * \Delta_1$. Thanks to the commutativity of $*$, we get $\Delta_2 * \Delta_1 = \Delta$ and so we are done.

— $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$. Suppose $\Gamma \vdash_\Sigma P \mid (Q \mid R) \triangleright \Delta$. Using A.2(7) we obtain $\Gamma \vdash_{\Sigma_1} P \triangleright \Delta_1$, $\Gamma \vdash_{\Sigma_2} Q \mid R \triangleright \Delta_2$, where $\Delta = \Delta_1 * \Delta_2$, $\Sigma = \Sigma_1 \cup \Sigma_2$ and $\Sigma_1 \cap \Sigma_2 = \emptyset$. Using A.2(7) we obtain $\Gamma \vdash_{\Sigma_{21}} Q \triangleright \Delta_{21}, \Gamma \vdash_{\Sigma_{22}} R \triangleright \Delta_{22}$ where $\Delta_2 = \Delta_{21} * \Delta_{22}$, $\Sigma_2 = \Sigma_{21} \cup \Sigma_{22}$ and $\Sigma_{21} \cap \Sigma_{22} = \emptyset$. Using (GPAR) we get $\Gamma \vdash_{\Sigma_1 \cup \Sigma_{21}} P \mid Q \triangleright \Delta_1 * \Delta_{21}$. Using (GPAR) again we get $\Gamma \vdash_\Sigma (P \mid Q) \mid R \triangleright \Delta_1 * \Delta_{21} * \Delta_{22}$ and so we are done by the associativity of $*$. The proof for the other direction is similar.

— $s : h_1 \cdot (\mathtt{q}, \Pi, v) \cdot (\mathtt{q}', \Pi', v') \cdot h_2 \equiv s : h_1 \cdot (\mathtt{q}', \Pi', v') \cdot (\mathtt{q}, \Pi, v) \cdot h_2$ where $\Pi \cap \Pi' = \emptyset$ or $\mathtt{q} \neq \mathtt{q}'$. We assume $\Pi \cap \Pi' = \emptyset$ and $\mathtt{q} = \mathtt{q}'$, the proof in the case $\mathtt{q} \neq \mathtt{q}'$ being similar and simpler. If $\Gamma \vdash_\Sigma s : h_1 \cdot (\mathtt{q}, \Pi, v) \cdot (\mathtt{q}, \Pi', v') \cdot h_2 \triangleright \Delta$, then $\Sigma = \{s\}$ by Lemma A.2(2). This implies $\Delta = \Delta_1 * \{s[\mathtt{q}] : !\langle \Pi, S \rangle;\ !\langle \Pi', S' \rangle\} * \Delta_2$ and $\Gamma \vdash_{\{s\}} s : h_i \triangleright \Delta_i$ $(i = 1, 2)$ and $\Gamma \vdash v : S$ and $\Gamma \vdash v' : S'$ by Lemma A.3(1). By the same lemma we can derive
$$\Gamma \vdash_{\{s\}} s : h_1 \cdot (\mathtt{q}, \Pi', v') \cdot (\mathtt{q}, \Pi, v) \cdot h_2 \triangleright \Delta_1 * \{s[\mathtt{q}] : !\langle \Pi', S' \rangle;\ !\langle \Pi, S \rangle\} * \Delta_2,$$
and we conclude using rule (EQUIV), since by definition $\Delta_1 * \{s[\mathtt{q}] : !\langle \Pi', S' \rangle;\ !\langle \Pi, S \rangle\} * \Delta_2 \approx \Delta$.

— $s : h_1 \cdot (\mathtt{q}, \Pi, v) \cdot h_2 \equiv s : h_1 \cdot (\mathtt{q}, \Pi', v) \cdot (\mathtt{q}, \Pi'', v) \cdot h_2$ where $\Pi = \Pi' \cup \Pi''$ and $\Pi' \cap \Pi'' = \emptyset$. If $\Gamma \vdash_\Sigma s : h_1 \cdot (\mathtt{q}, \Pi, v) \cdot h_2 \triangleright \Delta$, then $\Sigma = \{s\}$ by Lemma A.2(2). This implies
$$\Delta = \Delta_1 * \{s[\mathtt{q}] : !\langle \Pi, S \rangle\} * \Delta_2 \text{ and } \Gamma \vdash_{\{s\}} s : h_i \triangleright \Delta_i \ (i = 1, 2) \text{ and } \Gamma \vdash v : S$$
by Lemma A.3(1). By the same lemma we can derive
$$\Gamma \vdash_{\{s\}} s : h_1 \cdot (\mathtt{q}, \Pi', v) \cdot (\mathtt{q}, \Pi'', v) \cdot h_2 \triangleright \Delta_1 * \{s[\mathtt{q}] : !\langle \Pi', S \rangle;\ !\langle \Pi'', S \rangle\} * \Delta_2,$$
and we conclude using rule (EQUIV), since by definition $\Delta_1 * \{s[\mathtt{q}] : !\langle \Pi', S \rangle;\ !\langle \Pi'', S \rangle\} * \Delta_2 \approx \Delta$.

□

A.3. *Subject Reduction*

We first introduce consistency of session environments, which assures that each pair of participants in a multiparty conversation performs their mutual communications in a consistent way. Consistency is defined using the notions of projection of generalised types and of duality, given respectively in Definitions A.6 and A.7.

Definition A.6. The *projection of the generalised type* τ *onto* $\mathtt{q}$, denoted by $\tau \upharpoonright \mathtt{q}$, is defined by:

$$(!\langle \Pi, U\rangle.T) \upharpoonright \mathtt{q} = \begin{cases} !U.T \upharpoonright \mathtt{q} & \text{if } \mathtt{q} \in \Pi, \\ T \upharpoonright \mathtt{q} & \text{otherwise.} \end{cases} \qquad (?(\mathtt{p}, U).T) \upharpoonright \mathtt{q} = \begin{cases} ?U.T \upharpoonright \mathtt{q} & \text{if } \mathtt{p} = \mathtt{q}, \\ T \upharpoonright \mathtt{q} & \text{otherwise.} \end{cases}$$

$$(\,!\langle \Pi, U\rangle; \tau') \upharpoonright \mathtt{q} = \begin{cases} !U; \tau' \upharpoonright \mathtt{q} & \text{if } \mathtt{q} \in \Pi, \\ \tau' \upharpoonright \mathtt{q} & \text{otherwise.} \end{cases} \qquad (\oplus\langle \Pi, l\rangle; \tau') \upharpoonright \mathtt{q} = \begin{cases} \oplus l; \tau' \upharpoonright \mathtt{q} & \text{if } \mathtt{q} \in \Pi, \\ \tau' \upharpoonright \mathtt{q} & \text{otherwise.} \end{cases}$$

$$(\oplus\langle \Pi, \{l_i : T_i\}_{i\in I}\rangle) \upharpoonright \mathtt{q} = \begin{cases} \oplus\{l_i : T_i \upharpoonright \mathtt{q}\}_{i\in I} & \text{if } \mathtt{q} \in \Pi, \\ T_1 \upharpoonright \mathtt{q} & \text{if } \mathtt{q} \notin \Pi \text{ and } T_i \upharpoonright \mathtt{q} = T_j \upharpoonright \mathtt{q} \text{ for all } i, j \in I. \end{cases}$$

$$(\&(\mathtt{p}, \{l_i : T_i\}_{i\in I})) \upharpoonright \mathtt{q} = \begin{cases} \&\{l_i : T_i \upharpoonright \mathtt{q}\}_{i\in I} & \text{if } \mathtt{q} = \mathtt{p}, \\ T_1 \upharpoonright \mathtt{q} & \text{if } \mathtt{q} \neq \mathtt{p} \text{ and } T_i \upharpoonright \mathtt{q} = T_j \upharpoonright \mathtt{q} \text{ for all } i, j \in I. \end{cases}$$

$$(\mu \mathbf{t}.T) \upharpoonright \mathtt{q} = \begin{cases} \mu \mathbf{t}.(T \upharpoonright \mathtt{q}) & \text{if } T \upharpoonright \mathtt{q} \neq \mathbf{t}, \\ \mathsf{end} & \text{otherwise.} \end{cases} \qquad \mathbf{t} \upharpoonright \mathtt{q} = \mathbf{t} \qquad \mathsf{end} \upharpoonright \mathtt{q} = \mathsf{end}$$

Definition A.7. The *duality relation* between projections of generalised types ($\bowtie$) is the minimal symmetric relation which satisfies:

$$\mathsf{end} \bowtie \mathsf{end} \qquad \mathbf{t} \bowtie \mathbf{t} \qquad \mathfrak{T} \bowtie \mathfrak{T}' \implies \mu \mathbf{t}.\mathfrak{T} \bowtie \mu \mathbf{t}.\mathfrak{T}'$$
$$\mathfrak{T} \bowtie \mathfrak{T}' \implies !U.\mathfrak{T} \bowtie ?U.\mathfrak{T}' \qquad \mathfrak{T} \bowtie \mathfrak{T}' \implies !U;\mathfrak{T} \bowtie ?U.\mathfrak{T}'$$
$$\forall i \in I\ \mathfrak{T}_i \bowtie \mathfrak{T}'_i \implies \oplus\{l_i : \mathfrak{T}_i\}_{i\in I} \bowtie \&\{l_i : \mathfrak{T}'_i\}_{i\in I}$$
$$\exists i \in I\ l = l_i \wedge \mathfrak{T} \bowtie \mathfrak{T}_i \implies \oplus l;\mathfrak{T} \bowtie \&\{l_i : \mathfrak{T}_i\}_{i\in I}$$

where $\mathfrak{T}$ ranges over projections of generalised types.

Definition A.8. A session environment Δ is *consistent* if $s[\mathtt{p}] : \tau \in \Delta$ and $s[\mathtt{q}] : \tau' \in \Delta$ imply $\tau \upharpoonright \mathtt{q} \bowtie \tau' \upharpoonright \mathtt{p}$.

It is easy to check that projections of a same global type are always dual.

Proposition A.9. Let G be a global type and $\mathtt{p} \neq \mathtt{q}$. Then $(G \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q} \bowtie (G \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p}$.

This proposition assures that session environments obtained by projecting global types are always consistent.

The vice versa is not true, i.e. there are consistent session environments which are not projections of global types. An example is:

$$\{s[1] : ?(2, \mathsf{bool}).!\langle 3, \mathsf{bool}\rangle.\mathsf{end}, s[2] : ?(3, \mathsf{bool}).!\langle 1, \mathsf{bool}\rangle.\mathsf{end}, s[3] : ?(1, \mathsf{bool}).!\langle 2, \mathsf{bool}\rangle.\mathsf{end}\}$$

Note that for sessions with only two participants, instead, all consistent session environments are projections of global types.

Since session environments represent the forthcoming communications, by reducing processes session environments can change. This can be formalised as in (Honda et al., 2008) by introducing the notion of reduction of session environments, whose rules are:

— $\{s[\mathtt{p}] : M; !\langle \Pi, U\rangle.T\} \Rightarrow \{s[\mathtt{p}] : M;\ !\langle \Pi, U\rangle; T\}$
— $\{s[\mathtt{p}] : !\langle \mathtt{q}, U\rangle; \tau, s[\mathtt{q}] : M; ?(\mathtt{p}, U).T\} \Rightarrow \{s[\mathtt{p}] : \tau, s[\mathtt{q}] : M; T\}$
— $\{s[\mathtt{p}] : M; \oplus\langle \Pi, \{l_i : T_i\}_{i\in I}\rangle\} \Rightarrow \{s[\mathtt{p}] : M; \oplus(\Pi, l_j); T_j\}$ for $j \in I$
— $\{s[\mathtt{p}] : \oplus\langle \mathtt{q}, l\rangle; \tau, s[\mathtt{q}] : M; \&(\mathtt{p}, \{l_i : T_i\}_{i\in I})\} \Rightarrow \{s[\mathtt{p}] : \tau, s[\mathtt{q}] : M; T_i\}$ if $l = l_i$
— $\Delta, \Delta'' \Rightarrow \Delta', \Delta''$ if $\Delta \Rightarrow \Delta'$

where M can be missing and message types are considered modulo the equivalence relation of Table 8.
The first rule corresponds to putting in a queue a message with sender $\mathtt{p}$, set of receivers Π and content of type U. The second rule corresponds to reading from a queue a message with sender $\mathtt{p}$, receiver $\mathtt{q}$ and content of type U. The third and fourth rules are similar, but a label is transmitted.

The main result concerning the communication type system is the subject reduction theorem. Since session types are behavioural types subject reduction only assures that well-typed processes remain well typed during reduction. In fact communication actions consume the corresponding types (as represented by the $\Rightarrow$ relation). Consistency of environments assures that all communications, when they can take place, are performed in a correct way. The subject reduction for closed user processes (Theorem 4.3) follows immediately.

Theorem A.10 (Subject Reduction). If $\Gamma \vdash_\Sigma P \triangleright \Delta$ with Δ consistent and $P \longrightarrow^* P'$, then $\Gamma \vdash_\Sigma P' \triangleright \Delta'$ for some consistent Δ' such that $\Delta \Rightarrow^* \Delta'$. Moreover Δ closed implies Δ' closed.

Proof. By induction on a derivation of $P \longrightarrow^* P'$, with a case analysis on the final rule (using Theorem A.5 for the structural equivalence). We only consider some paradigmatic cases.

— [Init] $a[1](y).P_1 \mid ... \mid \overline{a}[n](y).P_n \longrightarrow (\nu s)(P_1\{s[1]/y_1\} \mid ... \mid P_n\{s[n]/y\} \mid s : \varnothing)$.
By hypothesis $\Gamma \vdash_\Sigma a[1](y).P_1 \mid a[2](y_2).P_2 \mid \ldots \mid \overline{a}[n](y).P_n \triangleright \Delta$; then, since the redex is a pure process, $\Sigma = \emptyset$ and $\Gamma \vdash a[1](y).P_1 \mid a[2](y_2).P_2 \mid \ldots \mid \overline{a}[n](y).P_n \triangleright \Delta$ by Lemma A.2(1). Using Lemma A.1(13) on all the processes in parallel we have

$$\Gamma \vdash a[i](y).P_i \triangleright \Delta_i \quad (1 \le i \le n-1) \tag{1}$$
$$\Gamma \vdash \overline{a}[n](y).P_n \triangleright \Delta_n \tag{2}$$

where $\Delta = \bigcup_{i=1}^{n} \Delta_i$. Using Lemma A.1(6) on (1) we have

$$\frac{\Gamma \vdash a : \mathsf{G}}{\Gamma \vdash P_i \triangleright \Delta_i, y : \mathsf{G} \upharpoonright i \quad (1 \le i \le n-1).} \tag{3}$$

Using Lemma A.1(5) on (2) we have

$$\frac{\Gamma \vdash a : \mathsf{G}}{\Gamma \vdash P_n \triangleright \Delta_n, y : \mathsf{G} \upharpoonright n} \tag{4}$$

and $\mathrm{mp}(\mathsf{G}) = n$. Using Lemma A.4(2) on (4) and (3) we have

$$\Gamma \vdash P_i\{s[i]/y\} \triangleright \Delta_i, s[i] : \mathsf{G} \upharpoonright i \quad (1 \le i \le n). \tag{5}$$

Using (PAR) on all the processes of (5) we have

$$\Gamma \vdash P_1\{s[1]/y\}|...|P_n\{s[n]/y\} \triangleright \bigcup_{i=1}^{n}(\Delta_i, s[i] : \mathsf{G} \upharpoonright i). \tag{6}$$

Note that $\bigcup_{i=1}^{n}(\Delta_i, s[i] : \mathsf{G} \upharpoonright i) = \Delta, s[1] : G \upharpoonright 1, \ldots, s[n] : \mathsf{G} \upharpoonright n$. Using (GINIT), (QINIT) and (GPAR) on (6) we derive

$$\Gamma \vdash_{\{s\}} P_1\{s[1]/y\}|...|P_n\{s[n]/y\} \mid s : \varnothing \triangleright \Delta, s[1] : \mathsf{G} \upharpoonright 1, \ldots, s[n] : \mathsf{G} \upharpoonright n. \tag{7}$$

Using (GSRES) on (7) we conclude

$$\Gamma \vdash_{\emptyset} (\nu s)(P_1\{s[1]/y\}|...|P_n\{s[n]/y\} \mid s : \varnothing) \triangleright \Delta$$

since $\{s[1] : \mathsf{G} \upharpoonright 1, \ldots, s[n] : \mathsf{G} \upharpoonright n\}$ is consistent and $(\Delta, s[1] : \mathsf{G} \upharpoonright 1, \ldots, s[n] : \mathsf{G} \upharpoonright n) \setminus s = \Delta$.

— [Send] $s[\mathtt{p}]!\langle \Pi, e\rangle.P \mid s : h \longrightarrow P \mid s : h \cdot (\mathtt{p}, \Pi, v)$ $(e \downarrow v)$.
By hypothesis, $\Gamma \vdash_{\Sigma} s[\mathtt{p}]!\langle \Pi, e\rangle.P \mid s : h \triangleright \Delta$. Using Lemma A.2(7), (1), and (2) we have $\Sigma = \{s\}$ and

$$\Gamma \vdash s[\mathtt{p}]!\langle \Pi, e\rangle.P \triangleright \Delta_1 \tag{8}$$

$$\Gamma \vdash_{\{s\}} s : h \triangleright \Delta_2 \tag{9}$$

where $\Delta = \Delta_2 * \Delta_1$. Using A.1(7) on (8) we have

$$\Delta_1 = \Delta_1', s[\mathtt{p}] : !\langle \Pi, S\rangle.T$$

$$\Gamma \vdash e : S \tag{10}$$

$$\Gamma \vdash P \triangleright \Delta_1', s[\mathtt{p}] : T. \tag{11}$$

From (10) by subject reduction on expressions we have

$$\Gamma \vdash v : S. \tag{12}$$

Using (QSEND) on (9) and (12) we derive

$$\Gamma \vdash_{\{s\}} s : h \cdot (\mathtt{q}, \Pi, v) \triangleright \Delta_2; \{s[\mathtt{p}] : !\langle \Pi, S\rangle\}. \tag{13}$$

Using (GINIT) on (11) we derive

$$\Gamma \vdash_{\emptyset} P \triangleright \Delta_1', s[\mathtt{p}] : T \tag{14}$$

and then using (GPAR) on (14), (13) we conclude

$$\Gamma \vdash_{\{s\}} P \mid s : h \cdot (\mathtt{q}, \Pi, v) \triangleright (\Delta_2; \{s[\mathtt{p}] : !\langle \Pi, S\rangle\}) * (\Delta_1', s[\mathtt{p}] : T).$$

Note that $\Delta_2 * (\Delta_1', s[\mathtt{p}] : !\langle \Pi, S\rangle.T) \Rightarrow (\Delta_2; \{s[\mathtt{p}] : !\langle \Pi, S\rangle\}) * (\Delta_1', s[\mathtt{p}] : T)$.

— [Rcv] $s[\mathtt{p}]?(\mathtt{q}, x).P \mid s : (\mathtt{q}, \{\mathtt{p}\}, v) \cdot h \longrightarrow P\{v/x\} \mid s : h$.
By hypothesis, $\Gamma \vdash_{\Sigma} s[\mathtt{p}]?(\mathtt{q}, x).P \mid s : (\mathtt{q}, \{\mathtt{p}\}, v) \cdot h \triangleright \Delta$. By Lemma A.2(7), (1), and (2) we have $\Sigma = \{s\}$ and

$$\Gamma \vdash s[\mathtt{p}]?(\mathtt{q}, x).P \triangleright \Delta_1 \tag{15}$$

$$\Gamma \vdash_{\{s\}} s : (\mathtt{q}, \{\mathtt{p}\}, v) \cdot h \triangleright \Delta_2 \tag{16}$$

where $\Delta = \Delta_2 * \Delta_1$. Using Lemma A.1(8) on (15) we have

$$\Delta_1 = \Delta_1', s[\mathtt{p}] :?(\mathtt{q}, S).T$$
$$\Gamma, x : S \vdash P \triangleright \Delta_1', s[\mathtt{p}] : T \tag{17}$$

Using Lemma A.3(1) on (16) we have

$$\Delta_2 = \{s[\mathtt{q}] : !\langle\{\mathtt{p}\}, S'\rangle\} * \Delta_2'$$
$$\Gamma \vdash_{\{s\}} s : h \triangleright \Delta_2' \tag{18}$$
$$\Gamma \vdash v : S'. \tag{19}$$

The consistency of Δ implies $S = S'$. Using Lemma A.4(1) from (17) and (19) we get $\Gamma \vdash P\{v/x\} \triangleright \Delta_1', s[\mathtt{p}] : T$, which implies by rule (GINIT)

$$\Gamma \vdash_{\emptyset} P\{v/x\} \triangleright \Delta_1', s[\mathtt{p}] : T. \tag{20}$$

Using rule (GPAR) on (20) and (18) we conclude

$$\Gamma \vdash_{\{s\}} P\{v/x\} \mid s : h \triangleright \Delta_2' * (\Delta_1', s[\mathtt{p}] : T).$$

Note that $(\{s[\mathtt{q}] : !\langle\{\mathtt{p}\}, S\rangle\} * \Delta_2') * (\Delta_1', s[\mathtt{p}] :?(\mathtt{q}, S); T) \Rightarrow \Delta_2' * (\Delta_1', s[\mathtt{p}] : T)$.

— [Sel] $s[\mathtt{p}] \oplus \langle \Pi, l\rangle.P \mid s : h \longrightarrow P \mid s : h \cdot (\mathtt{p}, \Pi, l)$.
By hypothesis, $\Gamma \vdash_{\Sigma} s[\mathtt{p}] \oplus \langle \Pi, l\rangle.P \mid s : h \triangleright \Delta$. Using Lemma A.2(7), (1), and (2) we have $\Sigma = \{s\}$ and

$$\Gamma \vdash s[\mathtt{p}] \oplus \langle \Pi, l\rangle.P \triangleright \Delta_1 \tag{21}$$
$$\Gamma \vdash_{\{s\}} s : h \triangleright \Delta_2 \tag{22}$$

where $\Delta = \Delta_2 * \Delta_1$. Using Lemma A.1(11) on (21) we have for $l = l_j$ $(j \in I)$:

$$\Delta_1 = \Delta_1', s[\mathtt{p}] : \oplus\langle \Pi, \{l_i : T_i\}_{i \in I}\rangle$$
$$\Gamma \vdash P \triangleright \Delta_1', s[\mathtt{p}] : T_j. \tag{23}$$

Using rule (QSEL) on (22) we derive

$$\Gamma \vdash_{\{s\}} s : h \cdot (\mathtt{p}, \Pi, l) \triangleright \Delta_2; \{s[\mathtt{p}] : \oplus\langle \Pi, l\rangle\}. \tag{24}$$

Using (GPAR) on (23) and (24) we conclude

$$\Gamma \vdash_{\{s\}} P \mid s : h \cdot (\mathtt{p}, \Pi, l) \triangleright (\Delta_2; \{s[\mathtt{p}] : \oplus\langle \Pi, l\rangle\}) * (\Delta_1', s[\mathtt{p}] : T_j).$$

Note that $\Delta_2 * (\Delta_1', s[\mathtt{p}] : \oplus\langle \Pi, \{l_i : T_i\}_{i \in I}\rangle) \Rightarrow (\Delta_2; \{s[\mathtt{p}] : \oplus\langle \Pi, l\rangle\}) * (\Delta_1', s[\mathtt{p}] : T_j)$.

— [Branch] $s[\mathtt{p}]\&(\mathtt{q}, \{l_i : P_i\}_{i \in I}) \mid s : (\mathtt{q}, \{\mathtt{p}\}, l_j) \cdot h \longrightarrow P_j \mid s : h$.
By hypothesis, $\Gamma \vdash_{\Sigma} s[\mathtt{p}]\&(\mathtt{q}, \{l_i : P_i\}_{i \in I}) \mid s : (\mathtt{q}, \{\mathtt{p}\}, l_j) \cdot h \triangleright \Delta$. Using Lemma A.2(7), (1), and (2) we have $\Sigma = \{s\}$ and

$$\Gamma \vdash s[\mathtt{p}]\&(\mathtt{q}, \{l_i : P_i\}_{i \in I}) \triangleright \Delta_1 \tag{25}$$
$$\Gamma \vdash_{\{s\}} s : (\mathtt{q}, \{\mathtt{p}\}, l_j) \cdot h \triangleright \Delta_2 \tag{26}$$

where $\Delta = \Delta_2 * \Delta_1$. Using Lemma A.1(12) on (25) we have

$$\begin{aligned} \Delta_1 &= \Delta_1', s[\mathtt{p}] : \&(\mathtt{q}, \{l_i : T_i\}_{i \in I}) \\ \Gamma &\vdash P_i \triangleright \Delta_1', s[\mathtt{p}] : T_i \quad \forall i \in I. \end{aligned} \tag{27}$$

Using Lemma A.3(3) on (26) we have

$$\begin{aligned} \Delta_2 &= \{s[\mathtt{q}] : \oplus\langle \mathtt{p}, l_j \rangle\} * \Delta_2' \\ \Gamma &\vdash_{\{s\}} s : h \triangleright \Delta_2'. \end{aligned} \tag{28}$$

Using (GPAR) on (27) and (28) we conclude

$$\Gamma \vdash_{\{s\}} P_j \mid s : h \triangleright \Delta_2' * (\Delta_1', s[\mathtt{p}] : T_j).$$

Note that for $j \in I$

$$(\{s[\mathtt{q}] : \oplus\langle \mathtt{p}, l_j \rangle\} * \Delta_2') * (\Delta_1', s[\mathtt{p}] : \&(\mathtt{q}, \{l_i : T_i\}_{i \in I})) \Rightarrow \Delta_2' * (\Delta_1', s[\mathtt{p}] : T_j).$$

□

A simple example showing that consistency is necessary for subject reduction is the process:

$$P = s[1]!\langle 2, \mathsf{true} \rangle.s[1]?(2,x).\mathbf{0} \mid s[2]?(1,x').s[2]!\langle 1, x'+1 \rangle.\mathbf{0}$$

which can be typed with the non consistent session environment

$$\{s[1] : !\langle 2, \mathsf{bool} \rangle.?(2, \mathsf{nat}).\mathsf{end}, s[2] : ?(1, \mathsf{nat}).!\langle 1, \mathsf{nat} \rangle.\mathsf{end}\}$$

In fact P reduces to the process

$$s[1]?(2,x).\mathbf{0} \mid s[2]!\langle 1, \mathsf{true}+1 \rangle.\mathbf{0}$$

which cannot be typed and it is stuck.

All processes considered in this article can be typed in the communication type system with consistent session environments.

Note that communication safety (Honda et al., 2008, Theorem 5.5) is a corollary of Theorem A.10.