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

Global Progress for Dynamically Interleaved Multiparty Sessions

Mario Coppo[1], Mariangiola Dezani-Ciancaglini[1], Nobuko Yoshida[2], Luca Padovani[1]

[1]*Dipartimento di Informatica, Università di Torino*
[2]*Department of Computing, Imperial College London*

Received 11 August 2014

A multiparty session forms a unit of structured communication among many participants which follow communication sequences specified as a global type. When a process is engaged in two or more sessions simultaneously, different sessions can be interleaved and can interfere at runtime. Previous work on multiparty session types has ignored session interleaving, providing a limited progress property ensured only within a single session, by assuming non-interference among different sessions and by forbidding delegation. This paper develops, besides a more traditional, compositional *communication* type system, a novel static *interaction* type system for global progress in dynamically interleaved and interfered multiparty sessions. The interaction type system infers causalities of channels making sure that processes do not get stuck at intermediate stages of sessions also in presence of delegation.

1. Introduction

Some important standardisation bodies for web-based business and finance protocols (Web Services Choreography Working Group, 2002; UNIFI, 2002; Savara, 2010) have recently investigated design and implementation frameworks for specifying message exchange rules and validating business logic based on a notion of *multiparty sessions*, where a global type plays as a "shared agreement" between teams of programmers developing possibly large and complex distributed protocols or software systems. Multiparty sessions, introduced in (Honda et al., 2008), provide a framework to represent messages-exchanges among concurrently running multiple peers in a distributed environment, generalising the existing dyadic sessions (Honda et al., 1998), where only two participants were assumed to interact.

A multiparty session is meant to describe the interaction of some participants on a specific subject of conversation (e.g., accessing an online e-commerce service) so that a specific goal can be achieved (e.g., buying a book). Nonetheless, it is often the case that the achievement of the goal requires distinct yet related interactions to be carried out (e.g., the communication with the e-commerce service, on one side, and the agreement between the buyers to establish the contribution of each, on the other side). We need then consider systems in which processes may be simultaneously engaged in different sessions, independently characterised by corresponding global types. When this happens, actions pertaining different sessions may interleave leading to interferences between the sessions: even if each process respects the global types of the sessions in which it participates, the system as a whole may be unable to make progress because of mutual dependencies between different sessions. Moreover, the mechanism of delegation (first introduced in (Honda et al., 1998)) may dynamically change the communication topology between the interacting processes, making the analysis of such systems even more complicated.

Previous works on (multiparty) sessions have ignored interferences among different sessions, guaranteeing a limited progress property only within a single session. More precisely, although previous type systems assure that all the participants of a session respect its global type, by checking the types of exchanged messages and the order of communications, they cannot guarantee the *global progress property*. By "global progress property" we intend, roughly, that each participant in a session, once the session has started, will always be allowed to perform its actions (regardless of whether the session will eventually terminate or not) without getting stuck in a local deadlock generated by the interaction of two (or more) different sessions. This notion of progress is different from other similar notions presented in the session type literature, and needs a careful definition and a non-trivial treatment.

To illustrate the subtleties and difficulties that arise in analysing the global progress property, let us discuss here some simple examples. For the sake of discussion, we consider only dyadic sessions written in a simplified syntax, but all examples can be easily generalised to sessions with many participants (this would require the complete syntax which is introduced later). Consider the processes

$$\begin{array}{ll} p_1 = a(y).y?(x).\mathbf{0} & p_2 = \overline{a}(y).y!\langle \mathsf{true}\rangle.\mathbf{0} \\ p_3 = b(z).z?(x).\mathbf{0} & p_4 = \overline{b}(z).z!\langle 2\rangle.\mathbf{0} \end{array}$$

where p_1 and p_2 initiate a session identified by the service name a while p_3 and p_4 initiate a session identified by the service name b. In both sessions only a single message is exchanged: the true value from p_2 to p_1 in the session identified by a and the number 2 from p_4 to p_3 in the session identified by b. The composition $p_1 \mid p_2 \mid p_3 \mid p_4$ describes a system in which both sessions are opened. Because the two sessions are totally independent (they are carried out by different processes), there is no interference and the sessions complete without problems. Consider now the processes

$$p_5 = a(y).b(z).y?(x).z!\langle 2\rangle.\mathbf{0} \qquad\qquad p_6 = \overline{a}(y).\overline{b}(z).z?(x).y!\langle \mathsf{true}\rangle.\mathbf{0}$$

which open the same two sessions. It is easy to see that the system $p_5 \mid p_6$ does not have progress since, after the sessions are initiated, each process gets stuck waiting for a message that is sent only after the other process has sent its own. Instead, if we take the system $p_5 \mid p_7$ where

$$p_7 = \overline{a}(y).\overline{b}(z).y!\langle \mathsf{true}\rangle.z?(x).\mathbf{0}$$

both sessions terminate successfully. Note that both $p_5 \mid p_6$ and $p_5 \mid p_7$ are well typed in the type system of (Honda et al., 2008) as well as in the communication type system introduced in this paper, but only the latter has progress.

The notion of progress we are seeking for is not simply deadlock-freedom. In fact, it shares many analogies with the notion of lock-freedom in (Kobayashi, 2002). For example, consider a process p in which two (or more) participants of a session are engaged in an endless but meaningful conversation (like some components in a scheduler of an operating system). Even though the process p is always able to reduce and so is the system $p_5 \mid p_6 \mid p$, we do not want to consider $p_5 \mid p_6 \mid p$ as having the global progress property because p_5 and p_6 are unable to perform their planned interaction. Notice that this process has progress according to the definition given in (Bettini et al., 2008). At the same time, an isolated process (like p_5) that is unable to make progress only because some participants of the sessions in which it is involved are missing should not be considered as lacking the global progress property *a priori*. For instance p_5 can start a successful session when composed in parallel first with p_2 and then with p_4. The assumption that session participants can be added is natural in an open ended environment in which new processes asking for interactions can join the system.

The main contributions of this article can be summarised as follows:

— We define 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. With respect to (Honda et al., 2008), we replace private communication channels within sessions with participant indexes. This choice leads to a simpler presentation of both processes and types.

— We define a notion of *global progress* (§5) which assures that all participants in openable sessions can perform all their communications, possibly with the help of suitable parallel processes. This is stronger than requiring that a system can always reduce and is weaker than requiring that all sessions can initiate.

— We develop a static *interaction type system* (§6) that assures global progress in dynamically interleaved, asynchronous, multiparty sessions.

This article is a thoroughly revised version of (Bettini et al., 2008) including a stronger notion of global progress, detailed definitions and full proofs. The new definition of progress, in particular, requires an original and non-trivial treatment. Before moving to the technical content as outlined above, we devote §2 to illustrating both the calculus and the type languages by means of an extended example involving the online e-commerce service that we hinted at earlier. §7 presents a detailed discussion of related work, while §8 concludes and discusses future work. For the sake of readability, auxiliary technical material and the proofs of the results have been postponed to the appendices.

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.

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.

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

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

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

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 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

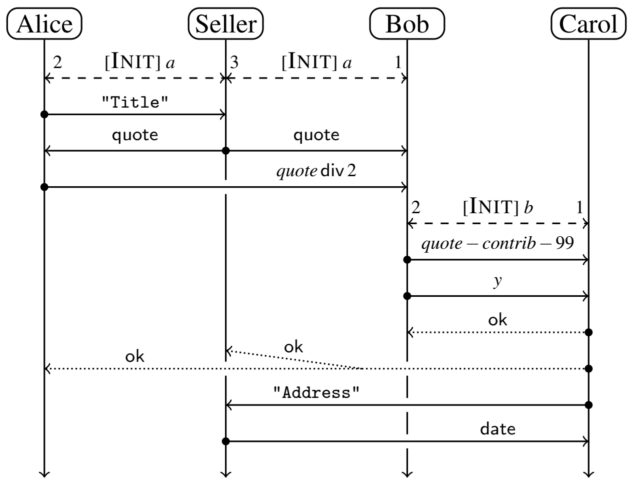

Fig. 1. An execution of the three buyer protocol.

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

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

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

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

Table 1. Implementation of the three buyer protocol.

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 it is fairly easy to see that no deadlock is possible, even if different sessions are interleaved with each other and the communication topology changes because of delegation. We formally state this in Corollary 6.5.

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.

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 §7. 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 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 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).

3.2. *Operational Semantics*

Table 3 shows the reduction rules of processes (we use $\longrightarrow^*$ and $\longrightarrow^k$ with the expected meanings).[†] 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

[†] For easiness of reading we have enclosed the tags of the reduction rules in square brackets, the tags of the communication type system rules in round brackets (see Tables 6, 10, 12) and the tags of the interaction type system rules in curly brackets (see Tables 7, 8).

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

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

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

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

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

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

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

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

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

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

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

Table 3. Reduction rules.

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

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

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

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

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

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

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

Table 4. Structural equivalence.

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.

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.

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 §. 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 $(\mathbf{t},\mathbf{t}',\dots)$ 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* $!\langle\Pi,S\rangle.T$, $!\langle\mathsf{p},\mathsf{T}\rangle.T$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 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 $\Pi=\{\mathsf{p}\}$ is a singleton we identify Π with 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* q $(G\upharpoonright\mathsf{q})$ is defined by induction on G:

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

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

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

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.

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

where T is defined at page 4.

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 $a : \mathsf{G}$, $X : S\,T$ 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 $(\textsc{Name})$ is standard: recall that u stands for x and a and S includes G.

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

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 $(\textsc{MCast})$ we can then derive $a : \mathsf{G}_a \vdash \text{Seller} \triangleright \emptyset$. Similarly by applying rule $(\textsc{MAcc})$ 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:

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

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

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

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

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

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

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

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

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

Table 6. Typing rules for expressions and pure processes.

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.

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. Progress

5.1. *The Notion of Progress for Multiparty Sessions (Informal)*

As a first approximation, we say that a process P has the progress property if all the interactions that are supposed to occur in P can (eventually) take place. Since a formal definition of progress is not straightforward, we begin by illustrating the pitfalls and the key ideas incrementally through a number of small examples.

A paradigmatic example of process without progress is the process $p_5 \mid p_6$ given in the introduction, that with the syntax of §3 becomes $P_1 \mid P_2$, where:

$$P_1 = a[1](y).b[1](z).y?(2,x_1).z!\langle 2,\mathsf{false}\rangle.\mathbf{0}$$
$$P_2 = \overline{a}[2](y).\overline{b}[2](z).z?(1,x_2).y!\langle 1,\mathsf{true}\rangle.\mathbf{0}$$

The process $P_1 \mid P_2$ reduces to:

$$(\nu s_1)(\nu s_2)(s_1[1]?(2,x_1).s_2[1]!\langle 2,\mathsf{false}\rangle.\mathbf{0} \mid s_2[2]?(1,x_2).s_1[2]!\langle 1,\mathsf{true}\rangle.\mathbf{0} \mid s_1 : \varnothing \mid s_2 : \varnothing)$$

Note that the resulting process corresponds to a configuration where two sessions have been initiated but have not terminated yet. Also, the configuration is irreducible, because it involves two input processes waiting to receive messages from two empty queues, and there is no way to induce further reductions even assuming that helper processes join the system, because the two blocked input processes are waiting on *private* session channels. This configuration is locked because the output actions of both sessions are prefixed by input actions of the other session.

On the contrary, the process $P_1 \mid P_2'$ where

$$P_2' = \overline{a}[2](y).\overline{b}[2](z).y!\langle 1,\mathsf{true}\rangle.z?(1,x_2).\mathbf{0}$$

has progress because it eventually reduces to $\mathbf{0}$.

In general, however, the technically simple definitions of progress are either too strong or too weak. For example, defining global progress as the possibility that each opened session can be successfully completed may be considered too demanding, as there are several reasonable protocols involving non terminating interactions. At the same time, the naive idea of defining progress as the possibility of reduction unless successful termination is reached, assigns progress to systems in which some processes engage an infinite chatter, while others hopelessly starve waiting for messages that are never sent. For example, the process $P_1 \mid P_2 \mid P_3 \mid P_4$ where P_1 and P_2 are as before and

$$\begin{aligned} P_3 &= b[1](t).\mathsf{def}\ X(x_3,z_3) = z_3!\langle 4,x_3\rangle.X\langle x_3,z_3\rangle\ \mathsf{in}\ X\langle \mathsf{true},t\rangle \\ P_4 &= \overline{b}[2](t).\mathsf{def}\ Y(x_4,z_4) = z_4?(3,x).Y\langle x_4,z_4\rangle\ \mathsf{in}\ Y\langle \mathsf{true},t\rangle \end{aligned}$$

leads to a configuration where P_1 and P_2 are stuck, but that can always reduce because of the interactions between processes P_3 and P_4. It would be unfortunate to say that $P_1 \mid P_2 \mid P_3 \mid P_4$ has progress only for this reason, because then *any* process, no matter how broken, could be "repaired" by coupling it with an independent, diverging subsystem. Notice that this process has progress according to the definition of (Bettini et al., 2008).

Building on Kobayashi's definition of lock-freedom (Kobayashi, 2002) and on the definition of communication safety of (Deniélou and Yoshida, 2011) we require that, in a process with the global progress property:

(1)	every input process will always (eventually) receive a message, and
(2)	every message in a queue will always (eventually) be received by an input process.

There is still one crucial aspect that we must address in some way and that affects significantly the formal definition of progress. We have seen why the compound process $P_1 \mid P_2$ does not have the progress property, but what about the processes P_1 and P_2, when they are considered in isolation? Is the fact that such processes cannot reduce enough to conclude that neither P_1 nor P_2 do have progress property? The point is that a process like P_1 is not flawed *per se*, and the reason why it cannot make any progress is just that some of the participants to initiate the sessions on a and b are missing, while the reduction rule [Init] requires *all* of the participants to be available for the session to begin. However, in an open ended scenario it is reasonable to assume that such missing participants can join in the future. Therefore, in defining the notion of progress for a process P, we consider that an incomplete service a occurring in P can always be allowed to start by composing P with some other processes containing the missing participants for a. We call such processes *catalysers* because they enable the initiation of sessions. Constructions that are similar to catalysers are given in (Dezani-Ciancaglini et al., 2008) and in (Carbone and Debois, 2010). For example, the process

$$P_5 = \overline{a}[2](y).y!\langle 1,\mathsf{true}\rangle.\mathbf{0}$$

is a catalyser that, when composed with P_1, allows the session on a to begin. Note that, by composing P_1 with the catalyser P_5, we have the reduction

$$P_1 \mid P_5 \longrightarrow^* (\nu s_1)(b[1](z).s_1[1]?(2,x).z!\langle 2,\mathsf{false}\rangle.\mathbf{0} \mid s_1 : (2,1,\mathsf{true}))$$

leading to an irreducible configuration where there there is one message $(2,1,\mathsf{true})$ in the queue associated with session s. However, unlike the irreducible configuration reachable from $P_1 \mid P_2$, in this case it is still possible to enable further reductions by adding one more catalyser

$$P_6 = \overline{b}[2](z).z?(1,x_2).\mathbf{0}$$

which initiates the session on the shared name b and ultimately allows the message in the queue to be received.

It is also important to notice that catalysers are needed to initiate sessions which produce stuck processes. For example without catalysers the process

$$P_7 = c[1](t_1).P_1 \mid P_2$$

would not reveal the deadlock.

In conclusion, we consider a notion of global progress in which a process with the progress property satisfies the conditions (1) and (2) above, possibly with the help of catalysers.

There are two important properties concerning catalysers that are consequences of the fact that we only allow catalysers to be composed in parallel with the process under exam, so as to represent missing participants of sessions. First of all, catalysers cannot be used for helping processes that can reach a locked configuration on private session channels. In particular, there is no catalyser that can prevent $P_1 \mid P_2$ from getting stuck. Second, catalysers cannot help to restart a process when the shared name on which there are missing participants is restricted. For example the process

$$P_8 = (\nu a : 1 \to 2 : \langle \mathsf{bool} \rangle.\mathsf{end})(a[1](y).y!\langle 2, \mathsf{true} \rangle.\mathbf{0})$$

behaves like $\mathbf{0}$ in any context. We can therefore say that P_7 has progress in a trivial way.

5.2. *The Notion of Progress for Multiparty Sessions (Formal)*

We will now introduce catalysers as service participants that we will use to complete a process in order to start sessions. We will build catalysers in such a way they cannot cause deadlocks. In particular, catalysers never interleave different sessions, that is only actions on the same session channel can prefix each other, with the exception of channel delegation and reception.

The formalisation of catalysers is made tricky by two aspects:

C1 the construction of a process sending a channel requires another set of catalysers which interact with the delegated channel, and

C2 the construction of a recursive process requires keeping track of the correspondence between type variables and term variables.

We build now the body of a catalyser $\mathscr{P}(T, y, \mathscr{M})$ which communicates through the channel y according to the session type T, using the mapping $\mathscr{M}$ to deal with recursion.

For delegating a channel (point C1 above), we need three mappings defined on well-formed and closed session types T. The first two mappings (denoted g and r) give respectively a well-formed, closed global type G and a participant p such that $\mathsf{T} = G \upharpoonright \mathrm{p}$, i.e. $\mathsf{T} = g(\mathsf{T}) \upharpoonright r(\mathsf{T})$. The definition of well-formed global and session types (Definition 4.2) assures that g and r can be defined. The third mapping (denoted f) gives a fresh session name.

For handling recursions (point C2 above), we devise a mapping (denoted $\mathscr{M}$) between type variables and term variables. By construction, $\mathscr{M}$ is empty when T is closed.

In order to get a deterministic construction, we make some (arbitrary) choices: for the outputs with base types, we choose a characteristic value for each sort type; for the outputs with global types, we choose fresh bound service names; for the selection types we always select the label with the minimum index. We build also characteristic environments which are needed to type catalysers.

Definition 5.1.

1 The *characteristic process* of the session type T using channel y and mapping $\mathscr{M}$, written $\mathscr{P}(T,y,\mathscr{M})$, is defined by induction on T through the following equations[‡]:

$$\begin{array}{ll}
\mathscr{P}(!\langle\Pi,\mathsf{bool}\rangle.T,y,\mathscr{M}) & = y!\langle\Pi,\mathsf{true}\rangle.\mathscr{P}(T,y,\mathscr{M}) \\
\quad\dots & = \quad\dots \\
\mathscr{P}(!\langle\Pi,\mathsf{G}\rangle.T,y,\mathscr{M}) & = (\nu a:\mathsf{G})y!\langle\Pi,a\rangle.\mathscr{P}(T,y,\mathscr{M}) \\
\mathscr{P}(!\langle\mathsf{p},\mathsf{T}\rangle.T,y,\mathscr{M}) & = f(\mathsf{T})[1](z).\mathscr{P}(g(\mathsf{T})\restriction 1,z,\emptyset)) \mid \dots \\
 & \quad \overline{f(\mathsf{T})}[r(\mathsf{T})](z).y!\langle\langle\mathsf{p},z\rangle\rangle.\mathscr{P}(T,y,\mathscr{M}) \mid \dots \\
 & \quad \overline{f(\mathsf{T})}[n](z).\mathscr{P}(g(\mathsf{T})\restriction n,z,\emptyset)) \\
 & \quad \text{where } n=\mathrm{mp}(g(\mathsf{T})) \\
\mathscr{P}(?(\mathsf{q},S).T,y,\mathscr{M}) & = y?(\mathsf{q},x).\mathscr{P}(T,y,\mathscr{M}) \\
\mathscr{P}(?(\mathsf{q},\mathsf{T}).T,y,\mathscr{M}) & = y?((\mathsf{q},z)).(\mathscr{P}(T,y,\mathscr{M}) \mid \mathscr{P}(\mathsf{T},z,\emptyset)) \\
\mathscr{P}(\oplus\langle\Pi,\{l_i:T_i\}_{i\in I}\rangle,y,\mathscr{M}) & = y\oplus\langle\Pi,l_k\rangle.\mathscr{P}(T_k,y,\mathscr{M}) \quad (k \text{ is the smallest index in } I) \\
\mathscr{P}(\&(\mathsf{q},\{l_i:T_i\}_{i\in I}),y,\mathscr{M}) & = y\&\langle\mathsf{q},\{l_i:\mathscr{P}(T_i,y,\mathscr{M})\}_{i\in I}\rangle \\
\mathscr{P}(\mu\mathbf{t}.T,y,\mathscr{M}) & = \mathsf{def}\ X(x,z)=\mathscr{P}(T,z,\mathscr{M}\cup\{\mathbf{t}\mapsto X\langle x,z\rangle\})\ \mathsf{in}\ X\langle\mathsf{true},y\rangle \\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (X \text{ fresh}) \\
\mathscr{P}(\mathbf{t},y,\mathscr{M}) & = \mathscr{M}(\mathbf{t}) \\
\mathscr{P}(\mathsf{end},y,\mathscr{M}) & = \mathbf{0}
\end{array}$$

2 The *characteristic environment* of the session type T, written $\P(T)$, is defined by induction on T through the following equations:

$$\begin{array}{c}
\P(!\langle\Pi,S\rangle.T)=\P(?(\mathsf{q},U).T)=\P(\mu\mathbf{t}.T)=\P(T) \\
\P(!\langle\mathsf{p},\mathsf{T}\rangle.T)=\{f(\mathsf{T}):g(\mathsf{T})\}\cup\P(\mathsf{T})\cup\P(T) \\
\P(\oplus\langle\Pi,\{l_i:T_i\}_{i\in I}\rangle)=\P(\&(\mathsf{q},\{l_i:T_i\}_{i\in I}))=\bigcup_{i\in I}\P(T_i) \\
\P(\mathbf{t})=\P(\mathsf{end})=\emptyset
\end{array}$$

As an example with delegation take $\mathscr{P}(!\langle 4,\mathsf{T}\rangle.\mathsf{end},y,\emptyset)$ where $\mathsf{T}=?(2,\mathsf{bool}).\mathsf{end}$. If $g(\mathsf{T})=2\to 1:\langle\mathsf{bool}\rangle.\mathsf{end}$, $r(\mathsf{T})=1$ and $f(\mathsf{T})=a$ we get

$$\begin{array}{lcl}
\mathscr{P}(!\langle 4,?(2,\mathsf{bool}).\mathsf{end}\rangle.\mathsf{end},y,\emptyset) & = & a[1](z).y!\langle\langle 4,z\rangle\rangle.\mathscr{P}(\mathsf{end},y,\emptyset) \mid \\
 & & \overline{a}[2](z).\mathscr{P}((2\to 1:\langle\mathsf{bool}\rangle.\mathsf{end})\restriction 2,z,\emptyset) \\
 & = & a[1](z).y!\langle\langle 4,z\rangle\rangle.\mathbf{0} \mid \overline{a}[2](z).z!\langle 1,\mathsf{true}\rangle.\mathbf{0} \\
 & & \\
\P(!\langle 4,?(2,\mathsf{bool}).\mathsf{end}\rangle.\mathsf{end}) & = & \{a:(2\to 1:\langle\mathsf{bool}\rangle.\mathsf{end})\}
\end{array}$$

An example of characteristic process for a recursive type is:

$$\begin{array}{lcl}
\mathscr{P}(\mu\mathbf{t}.!\langle 1,\mathsf{bool}\rangle.\mathbf{t},y,\emptyset) & = & \mathsf{def}\ X(x,z)=\mathscr{P}(!\langle 1,\mathsf{bool}\rangle.\mathbf{t},z,\{\mathbf{t}\mapsto X\langle x,z\rangle\})\ \mathsf{in}\ X\langle\mathsf{true},y\rangle \\
 & = & \mathsf{def}\ X(x,z)=z!\langle 1,\mathsf{true}\rangle.\mathscr{P}(\mathbf{t},z,\{\mathbf{t}\mapsto X\langle x,z\rangle\})\ \mathsf{in}\ X\langle\mathsf{true},y\rangle \\
 & = & \mathsf{def}\ X(x,z)=z!\langle 1,\mathsf{true}\rangle.X\langle x,z\rangle\ \mathsf{in}\ X\langle\mathsf{true},y\rangle
\end{array}$$

It is easy to verify that, whenever T is a closed type, we can derive:

$$\P(\mathsf{T})\vdash\mathscr{P}(\mathsf{T},y,\emptyset)\triangleright\{y:\mathsf{T}\}$$

[‡] For channel delegation if $r(\mathsf{T})=1$ or $r(\mathsf{T})=n$ the definition must be adapted in the obvious way. If $\mathsf{T}=\mathsf{end}$ we get $g(\mathsf{T})=\mathsf{end}$ and we assume $\mathrm{mp}(\mathsf{end})=1$.

which implies

$$\{a : \mathsf{G}\} \cup \P(\mathsf{G} \upharpoonright n) \vdash \overline{a}[n](y).\mathscr{P}(\mathsf{G} \upharpoonright n, y, \emptyset) \rhd \emptyset \quad \{a : \mathsf{G}\} \cup \P(\mathsf{G} \upharpoonright \mathrm{p}) \vdash a[\mathrm{p}](y).\mathscr{P}(\mathsf{G} \upharpoonright \mathrm{p}, y, \emptyset) \rhd \emptyset$$

where $n = \mathrm{mp}(\mathsf{G})$ and $\mathrm{p} < n$.

Catalysers are parallel compositions of processes obtained by prefixing characteristic processes of session types with requests and accepts:

Definition 5.2 (Catalyser). The *characteristic request* of the closed global type G for the service a is the process $\overline{a}[n](y).\mathscr{P}(\mathsf{G} \upharpoonright n, y, \emptyset)$ where $n = \mathrm{mp}(\mathsf{G})$. The *characteristic accept* of the closed global type G for the service a and participant $\mathrm{p} < \mathrm{mp}(\mathsf{G})$ is the process $a[\mathrm{p}](y).\mathscr{P}(\mathsf{G} \upharpoonright \mathrm{p}, y, \emptyset)$. A *catalyser* is a parallel composition of characteristic requests and accepts, which can be typed in the communication type system.

Notice that the process $\mathbf{0}$ is a catalyser, being the parallel composition of no process.

The last auxiliary notion we need before defining progress is a duality between input processes and message queues which only takes into account top inputs and top messages.

Definition 5.3. The *duality* $\bar{\bowtie}$ between input processes and message queues is the least symmetric relation defined by:

$$s[\mathrm{p}]?(\mathrm{q}, x).P \;\bar{\bowtie}\; s : (\mathrm{q}, \mathrm{p}, v) \cdot h \qquad s[\mathrm{p}]?((\mathrm{q}, y)).P \;\bar{\bowtie}\; s : (\mathrm{q}, \mathrm{p}, s'[\mathrm{p}']) \cdot h$$

$$s[\mathrm{p}]\&(\mathrm{q}, \{l_i : P_i\}_{i \in I}) \;\bar{\bowtie}\; s : (\mathrm{q}, \mathrm{p}, l_k) \cdot h \quad (k \in I)$$

Recall that $\mathscr{E}$ ranges over evaluation contexts (see Table 2). The main definition follows:

Definition 5.4 (Progress). A process P *has the progress property* if for all catalysers Q such that $P \mid Q$ is well typed in the communication system, if $P \mid Q \longrightarrow^* \mathscr{E}[R]$, where R is an input process or a not empty message queue, then there are a catalyser Q', and $\mathscr{E}', R'$ such that $P \mid Q \mid Q'$ is well typed in the communication system and $\mathscr{E}[R] \mid Q' \longrightarrow^* \mathscr{E}'[R][R']$ and $R \;\bar{\bowtie}\; R$'.

The well-typing of $P \mid Q \mid Q'$ in the communication system assures that in building the catalyser $Q \mid Q'$ we use global types for the free service names in P which allow to type P.

Thanks to the universal quantification over all catalysers Q we do not need to require that $\mathscr{E}[R] \mid Q'$ has the progress property. This is because $Q \mid Q'$ is a catalyser and we could just start from $P \mid Q \mid Q'$.

Notice that in the definition of progress catalysers play two different roles. The universally quantified catalyser Q allows to initiate sessions which can produce deadlocks. Without this catalyser the process P_7 defined at page 15 would have trivially progress. Instead the existentially quantified catalyser Q' allows to to initiate sessions in order to avoid deadlocks. For example the process $P_1 \mid P_5$ has progress thanks to the catalyser P_6 (see page 14).

It is interesting to remark that adding catalysers avoids the standard problems of racing for session initiations, since catalysers assure there are always enough participants to start sessions.

6. Interaction Type System

The interaction type system ensures that the typable processes always have the progress property. The basic ideas of this system are discussed in §6.1 and the typing rules are given in §6.2.

6.1. *Channel/Service Dependency and Sets of Service Names*

The progress property will be analysed via:

- an *irreflexive and transitive pre-order relation* $\mathscr{D}$ (called the channel/service dependency) among channel and service names and
- *three finite disjoint sets of service names* $\mathscr{R}$, $\mathscr{N}$ and $\mathscr{B}$.

Below we illustrate the relation and the sets, explaining their roles by examples.

Channel/Service Dependency and Relative Services. The channel/service dependency (*csd* for short) is our basic tool to analyse the dependencies between sessions. We write $a \prec b$ to denote that a precedes b in a csd. The meaning of $a \prec b$ is that some input action of a session on service a must be performed before some communication action of a session on service b. Csds are then transitive. We will see that we can assure progress of a process P if the csd of P does not contain loops, i.e. if do not have $a \prec b \prec a$ for any a, b occurring in P. In case of loop, for example, two input actions on a and b can mutually block the corresponding output actions on the other session producing a deadlock.

Take for instance the process $P_1 \mid P_2$, where P_1, P_2 are the processes defined in §5.1 (page 13). In process P_1 we have that an input action on service a blocks an output action on service b and this determines $a \prec b$. In process P_2 the situation is inverted, determining $b \prec a$. In $P_1 \mid P_2$ we then have the loop $a \prec b \prec a$. As remarked in §5.1 this process produces a deadlock. On the other hand if we take instead $P_1 \mid P_2'$, where P_2' is as in §5.1 (page 13), the csd of the whole process is $a \prec b$, since in P_2' the input action does not precede any other action, and we can so assure progress.

If we replace service b by service a in a slight modification of P_1 and P_2 we obtain:

$$P_9 = a[1](y).\overline{a}[2](z).y?(2,x).z!\langle 1,\mathsf{false}\rangle.\mathbf{0}$$
$$P_{10} = \overline{a}[2](y).a[1](z).z?(2,x').y!\langle 1,\mathsf{true}\rangle.\mathbf{0}$$

with two instances of service a and the dependency $a \prec a$. Also $P_9 \mid P_{10}$ reduces to a stuck process. Hence we must also forbid loops on single service names. This implies that csds cannot be reflexive.

Note that the csd of a single session turns out to be empty. Therefore a well-typed process with a single session has always progress as an immediate consequence of the Progress Theorem (Theorem 6.4).

The dependency-based mechanism captured by csds is quite conservative, in the sense that there exist practically relevant session patterns that yield reflexive csds but do not generate deadlocks. Moreover we need to take special care when we restrict service names. For this reason we introduce the three sets $\mathscr{R}$, $\mathscr{N}$, and $\mathscr{B}$ to distinguish between three different ways to type session initiations. The set $\mathscr{R}$ contains all service names which occur in $\mathscr{D}$: these services have the relative property of not being involved in circular dependencies with respect to $\mathscr{D}$. The sets

$\mathscr{N}$ and $\mathscr{B}$ contain service names which *may* be involved in circular dependencies, but which cannot cause deadlocks because of the peculiar, albeit recurrent, patterns in which they are related with other services. In particular the services in $\mathscr{B}$ can be safely restricted. We devote the next paragraphs to illustrate these patterns and how they are addressed with respect to the sets $\mathscr{N}$ and $\mathscr{B}$.

Nested Services. Let us consider the following process:

$$R_1 = \overline{a}[2](y).y?(1,x).\overline{a}[2](z).z?(1,x').z!\langle 1,\mathsf{true}\rangle.y!\langle 1,\mathsf{false}\rangle.\mathbf{0}$$

In this process we have an input action on z which blocks an action on y. Therefore reasoning as before we would get $a \prec a$ between the corresponding service names. But note that all actions on z are nested between the actions on y. More generally, there is no blocking action of the outermost invocation of a that is interleaved with actions of the innermost invocation of a. In fact, this interaction structure closely resembles an ordinary function call of a sequential programming language, where a caller function is suspended until the callee has terminated. If R_1 is put in parallel with another process in which the service a (and its associated channel variable) has the same behaviour we can assure progress despite the loop in the csd. For instance the process:

$$R_2 = a[1](y).y!\langle 2,\mathsf{false}\rangle.a[1](z).z!\langle 2,\mathsf{true}\rangle.z?(2,x').y?(2,x).\mathbf{0}$$

is such that $R_1 \mid R_2$ reduces to $\mathbf{0}$. This will be proved to be a general property. To take it into account we allow to put service names like a in $\mathscr{N}$ instead of putting them in the csd, avoiding then the loops they could produce.

We say that a service satisfies the *nesting condition* if all the communication actions on the channel bound by the service are interleaved only with outputs on different free channels and with services satisfying the same condition. This condition will be formalised in §6.2.

Note that we can put a service name in $\mathscr{N}$ only if *all* the occurrences of the service (in both multicast requests and accepts) respect the nesting condition. Take for instance the process:

$$R_3 = a[1](y).y!\langle 2,\mathsf{false}\rangle.a[1](z).y?(2,x).z!\langle 2,\mathsf{true}\rangle.z?(2,x').\mathbf{0}$$

It does not respect the nesting condition since an input action on y precedes an action on z. Indeed reducing $R_1 \mid R_3$ we obtain the stuck process:

$$\begin{array}{l}(\nu s_1)(\nu s_2)\ \ (s_2[2]?(1,x').s_2[2]!\langle 1,\mathsf{true}\rangle.s_1[2]!\langle 1,\mathsf{false}\rangle.\mathbf{0} \mid \\ \quad s_1[1]?(2,x).s_2[1]!\langle 2,\mathsf{true}\rangle.s_2[1]?(2,x').\mathbf{0} \mid s_1 : \varnothing \mid s_2 : \varnothing)\end{array}$$

Nesting is also useful for dealing with different services. As an example, consider the processes:

$$\begin{array}{l}R_4 = \overline{a}[2](y).\overline{b}[2](z).z?(1,x).y?(1,x').\mathbf{0}\\ R_5 = \overline{b}[2](z).\overline{a}[2](y).y?(1,x).z?(1,x').\mathbf{0}\end{array}$$

representing two clients which, for unspecified reasons, request the two services a and b in different orders. For $R_4 \mid R_5$ we get the circular csd $a \prec b \prec a$, but a and b are both nested, so putting them in $\mathscr{N}$ we can avoid this loop. For example if

$$R_6 = a[1](y).y!\langle 2,\mathsf{false}\rangle.\mathbf{0} \mid b[1](z).z!\langle 2,\mathsf{true}\rangle.\mathbf{0} \mid a[1](y).y!\langle 2,\mathsf{false}\rangle.\mathbf{0} \mid b[1](z).z!\langle 2,\mathsf{true}\rangle.\mathbf{0}$$

then $R_4 \mid R_5 \mid R_6 \longrightarrow^* \mathbf{0}$.

Considering again the processes P_1 and P_2 of page 13 we notice that the services a,b are nested in P_2 but they are not nested in P_1. So by putting a,b in $\mathscr{R}$ for P_1 we get the csd $a \prec b$ and by putting a,b in $\mathscr{N}$ for P_2 we get the empty csd. For this reason we always require that the same service cannot occur both in $\mathscr{R}$ and in $\mathscr{N}$.

Boundable Services. If we want to allow restrictions of service names the nesting condition is not enough. For example the process:

$$R_7 = a[1](y).(\nu b : \mathsf{end})(b[1](z).y!\langle 2, \mathsf{false}\rangle.\mathbf{0})$$

reduces to a deadlock when put in parallel with the catalyser $\overline{a}[2](y).y?(2,x).\mathbf{0}$. We need then ask also that all initiations of restricted services do not block any channel of another service, together with the requirement of being nested. We call this the *boundable service condition*. Also this condition will be formalised in §6.2. On the other hand we can allow that other services are requested or accepted after the actions of a boundable service are ended. For example, take the following process:

$$R_8 = (\nu a : 1 \to 2 : \langle \mathsf{bool}\rangle.\mathsf{end})(a[1](y).y!\langle 2, \mathsf{false}\rangle.R' \mid \overline{a}[2](y).y?(1,x).R'')$$

We observe that the process R_8 reduces to $R' \mid R''$, without interacting with the context. If processes R', R'' do not contain free channels (either channel variables or channels with roles) we can assure progress, provided $R' \mid R''$ assures it. In fact R_8 reduces to $R' \mid R''$ from which the computation can go on. On the other hand the process

$$R_9 = (\nu a : 1 \to 2 : \langle \mathsf{bool}\rangle.\mathsf{end})(a[1](y).y!\langle 2, \mathsf{false}\rangle.R')$$

where participant 2 is missing and R' does not contain free channels has trivially progress. Moreover R_8 and R_9 cannot cause deadlocks in any context, since they cannot participate to any interaction.

We can then put the service names satisfying both the nesting and the boundable service condition in the set $\mathscr{B}$ which is disjoint from $\mathscr{R}$ and $\mathscr{N}$.

The nesting condition is not enough also for service initiations on variables, as shown by the process:

$$a[1](y).y?(2,x).x[1](z).y!\langle 2, \mathsf{false}\rangle.\mathbf{0} \mid (\nu b : 1 \to 2 : \langle \mathsf{bool}\rangle.\mathsf{end})(\overline{a}[2](y).y!\langle 1, b\rangle.y?(1,x').\mathbf{0})$$

which reduces to the stuck process

$$(\nu s)(\nu b : 1 \to 2 : \langle \mathsf{bool}\rangle.\mathsf{end})(b[1](z).s[1]!\langle 2, \mathsf{false}\rangle.\mathbf{0} \mid s[2]?(1,x').\mathbf{0} \mid s : \varnothing)$$

Therefore we require that all service initiations on variables satisfy both the nesting and the boundable service condition.

First-class services. Finally we consider that service names are first-class entities and can be sent as messages. In this case, the dependency analysis for preventing deadlocks turns out to be too weak, because as the system evolves – and service names are passed around – the actual dependencies between services may dynamically change. To illustrate the issue, consider the processes

$$R_{10} = c[1](t).t?(2,x).x[1](y).b[1](z).y?(2,x').z!\langle 2,\mathsf{true}\rangle.\mathbf{0}$$
$$R_{11} = \overline{c}[2](t).t!\langle 1,a\rangle.\mathbf{0}$$

and observe that R_{11} sends to R_{10} the name of service a. The analysis of process R_{10} may determine the relation $x \prec b$, because there is an action pertaining service x that blocks another action pertaining service b. However, since x is a *bound variable* in R_{10}, there is no obvious way to associate this dependency with R_{10}. On the other hand, the analysis of process R_{11} yields no apparent dependencies for a. Overall, no dependency is inferred for $R_{10} \mid R_{11}$, even though at run-time the system will reduce to a configuration that yields the relation $a \prec b$. Then, if $R_{10} \mid R_{11}$ is composed with a process that yields the inverse dependency $b \prec a$, a deadlock can occur. Indeed $R_{10} \mid R_{11} \mid P_2$ reduces to $P_1 \mid P_2$ which leads to a deadlock, as we have seen at page 13.

To overcome this problem, we ask that a free service name which is sent complies with the nesting condition that can safely deal with circular dependencies. Therefore a sent service name must belong to the union of $\mathscr{N}$ and $\mathscr{B}$. This strategy is conservative, because it may happen that a service is never actually involved in circular dependencies with other services throughout the whole evolution of a system.

6.2. *Typing Rules*

We define the channel qualifier of a channel as either a channel variable or a session name.

Definition 6.1. Let c be a channel variable or a channel with role, its *channel qualifier* $\curlywedge(c)$ is given by:

$$\curlywedge(c) = \begin{cases} y & \text{if } c = y\,, \\ s & \text{if } c = s[\mathsf{p}]. \end{cases}$$

We consider csds over the set of all service names and all channel qualifiers, which we call Λ (ranged over by λ). We denote by $\lambda \prec \lambda'$ the elements of the Cartesian product $\Lambda \times \Lambda$. The meaning of $\lambda \prec \lambda'$, roughly, is that an input action or a delegation on a channel (qualified by) λ or bound by service λ can block a communication action on a channel (qualified by) λ' or bound by service λ'.

The progress property will be analysed, using the notions described above, via the *interaction* typing system, whose rules are given in Tables 7 and 8. The judgements are of the shape:

$$\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P \blacktriangleright \mathscr{D}$$

where:

- $\mathscr{D}$ is a csd,
- $\mathscr{R},\mathscr{N},\mathscr{B}$ are sets of service names and
- Θ is a set of *assumptions* of the shape $X[y] \blacktriangleright \mathscr{D}$ (for recursive definitions) with the variable y representing the channel parameter of X. We require that y is the only channel which may occur in $\mathscr{D}$. This agrees with rule (VAR), which allows only y to get a type different from end.

The sets $\mathscr{D},\mathscr{R},\mathscr{N},\mathscr{B}$ have the following meanings:

$$\frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D} \quad a\in\mathscr{R}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash \tilde{a}[\mathsf{p}](y).P \blacktriangleright \mathscr{D}\{a/y\}^{+}}\ \{\textsc{InitR}\} \qquad \frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D} \quad a\in\mathscr{N}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash \tilde{a}[\mathsf{p}](y).P \blacktriangleright \mathscr{D}\backslash\backslash y}\ \{\textsc{InitN}\}$$

$$\frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D} \quad a\in\mathscr{B} \quad \mathsf{fc}(P)\subseteq\{y\}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash \tilde{a}[\mathsf{p}](y).P \blacktriangleright \mathscr{D}\backslash\backslash y}\ \{\textsc{InitB}\} \qquad \frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D} \quad \mathsf{fc}(P)\subseteq\{y\}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash \tilde{x}[\mathsf{p}](y).P \blacktriangleright \mathscr{D}\backslash\backslash y}\ \{\textsc{InitV}\}$$

$$\frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D} \quad e\in\mathscr{S}\Rightarrow e\in\mathscr{N}\cup\mathscr{B}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash c!\langle\Pi,e\rangle.P \blacktriangleright \mathscr{D}}\ \{\textsc{Send}\} \qquad \frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash c?(\mathsf{q},x).P \blacktriangleright (\mathsf{pre}(c,\mathsf{fc}(P))\cup\mathscr{D})^{+}}\ \{\textsc{Rcv}\}$$

$$\frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash c!\langle\langle\mathsf{p},c'\rangle\rangle.P \blacktriangleright (\{\curlywedge(c)\prec\curlywedge(c')\}\cup\mathscr{D})^{+}}\ \{\textsc{Deleg}\} \qquad \frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D} \quad \mathscr{D}\setminus\mathscr{S}\subseteq\{\curlywedge(c)\prec y\}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash c?((\mathsf{q},y)).P \blacktriangleright \mathscr{D}\setminus\{y\}}\ \{\textsc{SRcv}\}$$

$$\frac{}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash \mathbf{0} \blacktriangleright \emptyset}\ \{\textsc{Inact}\} \qquad \frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D} \quad a\in\mathscr{B}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\setminus\{a\}\vdash (\nu a:\mathsf{G})P \blacktriangleright \mathscr{D}}\ \{\textsc{NRes}\}$$

$$\frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P_1 \blacktriangleright \mathscr{D}_1 \quad \Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P_2 \blacktriangleright \mathscr{D}_2}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P_1 \mid P_2 \blacktriangleright (\mathscr{D}_1\cup\mathscr{D}_2)^{+}}\ \{\textsc{Par}\} \qquad \frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P_1 \blacktriangleright \mathscr{D}_1 \quad \Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P_2 \blacktriangleright \mathscr{D}_2}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash \mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \blacktriangleright (\mathscr{D}_1\cup\mathscr{D}_2)^{+}}\ \{\textsc{If}\}$$

$$\frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash c\oplus\langle\Pi,l\rangle.P \blacktriangleright \mathscr{D}}\ \{\textsc{Sel}\} \qquad \frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P_i \blacktriangleright \mathscr{D}_i \quad \forall i\in I}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash c\&(\mathsf{p},\{l_i:P_i\}_{i\in I}) \blacktriangleright (\mathsf{pre}(c,\bigcup_{i\in I}\mathsf{fc}(P_i))\cup\bigcup_{i\in I}\mathscr{D}_i)^{+}}\ \{\textsc{Branch}\}$$

$$\frac{e\in\mathscr{S}\Rightarrow e\in\mathscr{N}\cup\mathscr{B}}{\Theta,(X[y]\blacktriangleright\mathscr{D});\mathscr{R};\mathscr{N};\mathscr{B}\vdash X\langle e,c\rangle \blacktriangleright \mathscr{D}\{\curlywedge(c)/y\}}\ \{\textsc{Var}\}$$

$$\frac{\Theta,(X[y]\blacktriangleright\mathscr{D});\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D} \qquad \Theta,(X[y]\blacktriangleright\mathscr{D});\mathscr{R};\mathscr{N};\mathscr{B}\vdash Q \blacktriangleright \mathscr{D}'}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash \mathsf{def}\ X(x,y)=P\ \mathsf{in}\ Q \blacktriangleright \mathscr{D}'}\ \{\textsc{Def}\}$$

Table 7. Interaction typing rules I.

$$\frac{}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash s:\varnothing \blacktriangleright \emptyset}\ \{\textsc{QInit}\} \qquad \frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash s:h \blacktriangleright \mathscr{D} \quad v\in\mathscr{S}\Rightarrow v\in\mathscr{N}\cup\mathscr{B}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash s:h\cdot(\mathsf{q},\Pi,v) \blacktriangleright \mathscr{D}}\ \{\textsc{QAddval}\}$$

$$\frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash s:h \blacktriangleright \mathscr{D}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash s:h\cdot(\mathsf{q},\mathsf{p},s'[\mathsf{p}']) \blacktriangleright \{s\prec s'\}\cup\mathscr{D}}\ \{\textsc{QAddsess}\}$$

$$\frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash s:h \blacktriangleright \mathscr{D}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash s:h\cdot(\mathsf{q},\Pi,l) \blacktriangleright \mathscr{D}}\ \{\textsc{QSel}\} \qquad \frac{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash P \blacktriangleright \mathscr{D}}{\Theta;\mathscr{R};\mathscr{N};\mathscr{B}\vdash (\nu s)P \blacktriangleright \mathscr{D}\setminus s}\ \{\textsc{SRes}\}$$

Table 8. Interaction typing rules II.

- $\mathscr{D}$ (*csd*) is an irreflexive pre-order between channel, session and service names;
- $\mathscr{R}$ (*relative service set*) contains the service names which may occur in $\mathscr{D}$;
- $\mathscr{N}$ (*nested service set*) is a set of services that satisfy the nesting condition in all their occurrences;
- $\mathscr{B}$ (*boundable service set*) is a set of services that satisfy the boundable service condition in all their occurrences.

Note that the typing rules are applied following the structure of the analysed process. The sets $\mathscr{R}$, $\mathscr{N}$, and $\mathscr{B}$ are not synthesised by the rules. The interaction type system simply checks that services are used correctly depending on the set in which they occur. For this reason, these sets are not changed by the rules of the interaction type system, except obviously for the case of service name restriction. The csd $\mathscr{D}$ instead accumulates the dependencies relation on channel variables, session and service names.

We convene that we can apply a typing rule only if the judgment is well formed, i.e. the obtained csd is defined and irreflexive.

Initiation. When a channel is bound by a session initiation the channel is removed from $\mathscr{D}$ by:

- replacing it in $\mathscr{D}$ by the service name if the service belongs to $\mathscr{R}$ and this does not create a loop in the transitive closure of the obtained csd;
- erasing it from $\mathscr{D}$ if the service belongs to $\mathscr{N}$ and the channel is minimal in $\mathscr{D}$;
- erasing it from $\mathscr{D}$ if the service belongs to $\mathscr{B}$ and the channel is minimal in $\mathscr{D}$ and the process does not contain other free channels;
- erasing it from $\mathscr{D}$ in case of a service variable if the channel is minimal in $\mathscr{D}$ and the process does not contain other free channels.

We get therefore four rules for service initiation, where we use $\tilde{a}$ for either a or $\overline{a}$.

(1): $\{\textsc{InitR}\}$. This rule requires $a \in \mathscr{R}$ and it corresponds to the more liberal policy with respect to the occurrences of the channel y in P. The service name a replaces y in $\mathscr{D}$ if this replacement does not generate a loop in the transitive closure (denoted by $^+$) of the obtained csd, otherwise the initiation cannot be typed.

(2). $\{\textsc{InitN}\}$. This rule can be applied only if $a \in \mathscr{N}$ and the channel y bound by the request or accept on a is minimal in $\mathscr{D}$, i.e. for no λ we have $\lambda \prec y \in \mathscr{D}$. To this aim we define $\mathscr{D} \setminus\!\!\setminus y$ by:

$$\mathscr{D} \setminus\!\!\setminus y = \begin{cases} \{\lambda \prec \lambda' \in \mathscr{D} \mid \lambda \neq y\} & \text{if } y \text{ is minimal in } \mathscr{D} \\ \text{undefined} & \text{otherwise} \end{cases}$$

This formalises the nesting condition.

(3) $\{\textsc{InitB}\}$. This rule requires $a \in \mathscr{B}$. Moreover we do not only ask that y is minimal (since $\mathscr{D} \setminus\!\!\setminus y$ must defined), but also that y is the only free channel in the process P. This condition is assured by the premise $\mathsf{fc}(P) \subseteq \{y\}$, since we denote by $\mathsf{fc}(P)$ the set of free channels which occur in P. This formalises the boundable condition.

(4) $\{\textsc{InitV}\}$ This rule requires both the nesting and the boundable conditions for typing an initiation on a service variable.

Sending and Receiving. Rule $\{\textsc{Rcv}\}$ asserts that the input action can block all other actions in P. This is obtained by prefixing the channel qualifier of the input action to all qualifiers of free channels in P but itself, since we define:

$$\mathsf{pre}(c, \mathsf{C}) \quad = \quad \{\curlywedge(c) \prec \curlywedge(c') \mid c' \in \mathsf{C} \wedge \curlywedge(c') \neq \curlywedge(c)\}$$

where C is a set of channels.

Rule $\{\textsc{Send}\}$ simply checks that if the sent value belongs to the set $\mathscr{S}$ of service names, then it occurs in $\mathscr{N}$ or $\mathscr{B}$, see the discussion at page 20.

Delegation. Rule {DELEG} is similar to {SEND} but it asserts that a use of $\curlywedge(c)$ must precede a use of $\curlywedge(c')$: the dependency $\curlywedge(c) \prec \curlywedge(c')$ needs to be registered since an action blocking $\curlywedge(c)$ also blocks $\curlywedge(c')$. Rule {SRC} forbids to create a process where two different roles in the same session are put in sequence (Dezani-Ciancaglini et al., 2006; Yoshida and Vasconcelos, 2007). As an example consider the processes

$$P_{11} = b[1](z).a[1](y).y!\langle\langle 2,z\rangle\rangle.\mathbf{0}$$
$$P_{12} = \overline{b}[2](z).\overline{a}[2](y).y?((1,t)).t?(2,w).z!\langle 1,\mathsf{false}\rangle.\mathbf{0}$$

and note that $P_{11} \mid P_{12}$ reduces to $(\nu s)(s[1]?(2,w).s[2]!\langle 1,\mathsf{false}\rangle.\mathbf{0})$ which is stuck. The process $P_{11} \mid P_{12}$ is typable in the communication type system, but P_{12} is not typable in the interaction type system, since by typing $y?((1,t)).t?(2,w).z!\langle 1,\mathsf{false}\rangle.\mathbf{0}$ we get $y \prec z$ which is forbidden by rule {SRC}.

Inaction and Restriction. Rule {INACT} asserts that process $\mathbf{0}$ has empty csd starting from arbitrary sets of service names.

Rule {NRES} checks that a occurs in the boundable service set.

Parallel and Conditional Compositions. Rule {PAR} is the key to calculate the csds of parallel processes. The resulting process can exhibit the behaviour of both its constituents and then we take the transitive closure of the union of all the csds of the composed processes. Rule {IF} is similar, even if we could consider more permissive conditions, at the price of having hereditarily finite sets of csds. We cannot type, for example, the process:

$$P_{13} = a[1](y).b[1](z).c[1](t_1).t_1?(2,x).\mathsf{if}\ x\ \mathsf{then}\ y?(2,x_1).z?(2,x'_1).\mathbf{0}\ \mathsf{else}\ z?(2,x'_2).y?(2,x_2).\mathbf{0}$$

since the csd of the conditional is $\{y,z,y \prec z, z \prec y\}$. Note that this process in parallel with a process where requests on a and b are in parallel, like in the process

$$P_{14} = \overline{a}[2](y).y!\langle 1,\mathsf{true}\rangle.\mathbf{0} \mid \overline{b}[2](z).z!\langle 1,\mathsf{false}\rangle.\mathbf{0} \mid \overline{c}[2](t_2).t_2!\langle 1,\mathsf{bv}\rangle.\mathbf{0}$$

with $\mathsf{bv} \in \{\mathsf{true},\mathsf{false}\}$, reduces with no deadlocks.

Branching and Selection. Rule {SEL} is similar to rule {SEND}, while rule {BRANCH} needs to record the causality as a union of all csds as in {PAR} and {IF} rules.

Process Declarations. Rule {VAR} replaces $\curlywedge(c)$ to y in the csd and if the expression is a service name checks that it belongs to $\mathscr{N} \cup \mathscr{B}$ (like {SEND}).

Rule {DEF} requires that:

- the typing of a term variable coincides with the typing of the process which will replace the variable;
- the body of the definition is typable.

Rules for Queues and Session Restriction. (See Table 8). The first four rules can be understood by comparing them to the rules {INACT}, {SEND}, {DELEG} and {SEL}, respectively. Rule {SRES} deletes the session name from the csd.

Reducing a process which is well typed in both type systems we get a process which is well typed too in the interaction type system with respect to the same sets of assumptions and service names and we get a smaller or equal csd. Note in fact that only free service or session names can occur in $\mathscr{D}$ and they cannot be created by reduction. Instead, for instance, in a session initiation a (possibly) free service name is replaced by a restricted session name which is removed from $\mathscr{D}$ by rule $\{\textsc{SRes}\}$.

Theorem 6.2 (Subject Reduction for the interaction type system).
If P is well typed in the communication type system and $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P \;\blacktriangleright\; \mathscr{D}$ and $P \longrightarrow^* P'$, then $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P' \;\blacktriangleright\; \mathscr{D}'$ for some $\mathscr{D}' \subseteq \mathscr{D}$.

Appendix B contains the proof of this theorem.

For example the processes P_1, P_2, P_2', P_3 and P_4 defined at pages 13 and 14 can be typed in the interaction type system with $\mathscr{R} = \{a, b\}$. Moreover the process P_2 can also be typed with $\mathscr{R} = \{a\}$ and $\mathscr{N} = \{b\}$, and clearly the processes P_3, P_4 can be typed with $\mathscr{R} = \{b\}$ or $\mathscr{N} = \{b\}$. Choosing $\mathscr{R} = \{a, b\}$ the csd obtained by typing P_1, P_2' is $a \prec b$, while the csd obtained by typing P_2 is $b \prec a$. Therefore $P_1 \mid P_2'$ is typable, while $P_1 \mid P_2$ and $P_1 \mid P_2 \mid P_3 \mid P_4$ are not typable. Typability is clearly not compositional.

The process P_9 at page 18 cannot be typed, since the service a is not nested and it gives $a \prec a$. Instead the process P_{10} can be typed with $\mathscr{N} = \{a\}$. The processes R_1, R_2 (see page 19) can be typed with $\mathscr{N} = \{a\}$ getting the empty csd, and so also $R_1 \mid R_2$ is typable. Instead process R_3 (page 19) cannot be typed, in fact by putting a in $\mathscr{R}$ we get $a \prec a$, and we cannot put a in $\mathscr{N}$ since it does not satisfy the nesting condition. The processes R_4, R_5, R_6 of page 19 can be typed with $\mathscr{N} = \{a, b\}$.

Lastly a must be in $\mathscr{B}$ for typing the bodies of the restriction on a in the processes R_7, R_8, R_9 of page 20.

Three Buyer Protocol. As an example we consider again the process Bob of the three buyer protocol (see §2), which we report here for convenience:

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

Table 9 shows in a schematic way the typing derivation for Bob by reporting the csds, and the applied rules, under the choice $\mathscr{R} = \{a\}$, $\mathscr{N} = \{b\}$, $\mathscr{B} = \emptyset$. It is easy to verify that this process is typable for any choice of disjoint sets $\mathscr{R}, \mathscr{N}, \mathscr{B}$ such that $a \in \mathscr{R} \cup \mathscr{N} \cup \mathscr{B}$, $b \in \mathscr{R} \cup \mathscr{N}$ and $b \in \mathscr{N}$ whenever $a \in \mathscr{N} \cup \mathscr{B}$.

6.3. *Progress Theorem*

Definition 6.3. A closed user process P is *initial* if it is typable both in the communication and in the interaction type systems, i.e. we can find $\Gamma, \mathscr{R}, \mathscr{N}, \mathscr{D}$ such that $\Gamma \vdash P \triangleright \emptyset$ and $\emptyset;\mathscr{R};\mathscr{N};\emptyset \vdash P \;\blacktriangleright\; \mathscr{D}$.

Observe that the set of boundable services can be empty, since the process P is never put in a context. The set of assumptions on process variables can be empty since P is closed.

Process	$\mathscr{D}$	Rule
$z\&(1,\{\mathsf{ok}:\mathbf{0},\ \mathsf{quit}:\mathbf{0}\})$	$\emptyset$	$\{\text{BRANCH}\}$
$z!\langle\langle 1,y\rangle\rangle$	$\{z \prec y\}$	$\{\text{DELEG}\}$
$z!\langle\{1\},\mathit{quote}\text{ - }\mathit{contrib}\text{ - }99\rangle$	$\{z \prec y\}$	$\{\text{SEND}\}$
$\overline{b}[2](z)$	$\emptyset$	$\{\text{INITN}\}$
$y?(3,\mathit{date});\mathbf{0}$	$\emptyset$	$\{\text{RCV}\}$
$y!\langle\{3\},\texttt{"Address"}\rangle$	$\emptyset$	$\{\text{SEND}\}$
$y\oplus\langle\{1,3\},\mathsf{ok}\rangle$	$\emptyset$	$\{\text{SEL}\}$
$\mathsf{if}\ (\ldots)$	$\emptyset$	$\{\text{IF}\}$
$\ldots$	$\ldots$	$\ldots$
$a[2](y)$	$\emptyset$	$\{\text{INITR}\}$

Table 9. Typing of process Bob.

The progress property is assured for all computations that are generated from initial processes, where all variable are bound but service names can be free. We claim that considering only processes with bound service names would limit the applicability of this approach to open-ended scenarios.

Theorem 6.4 (Progress). All initial processes have the progress property.

The proof of this Theorem is the content of Appendix C. To show that a process P has the progress property we must assure, roughly, that each *specific* input or output action an a channel with role occurring in some process P' obtained by reducing $P \mid Q$, where Q is an arbitrary catalyser, can always be executed finding a reduct of $P' \mid Q'$ for some catalyser Q' which adds receipt/accept processes if needed. This could be proved in a (relatively) easy way from the Subject Reduction Theorem 6.2 if all computations starting from $P \mid Q$ were finite. In this case in fact it would be enough to guarantee that a process containing a channel with role always reduces, possibly in parallel with a suitable catalyser. But note that in the interaction type system the information about specific participants is lost, so it is not straightforward to follow the moves of a process along a specific channel with role, as it would be necessary in processes like $P_1 \mid P_2 \mid P_3 \mid P_4$ at page 14. To overcome this difficulty we will consider only finite computations by introducing a notion of "approximate" typed reduction, in which recursive processes are frozen (i.e. they cannot be further reduced) after a finite number of applications of the [ProcCall] rule. It is always possible to consider a number of recursion unfoldings large enough to reach the input process or the message queue considered in the definition of progress.

As an immediate corollary of Theorem 6.4 we get that progress inside a single service is assured by the communication typing rules in §4.

It is easy to verify that the three buyer protocol can be typed in the interaction type system. Therefore we obtain:

Corollary 6.5. The three buyer protocol has the progress property.

It is clear that given two sets $\mathscr{R}$, $\mathscr{N}$, typability in the interaction system of a closed user process without recursion and whose free session names are included in $\mathscr{R} \cup \mathscr{N}$ can be easily checked. Note that the bound service names are put in $\mathscr{B}$. For recursive definitions one can compute the csd associated to the process variables by iteration, starting from the empty set and stopping when a fixed-point is reached.

Typability of a process in the interaction type system depends if the free service names of the process belong to $\mathscr{R}$ or $\mathscr{N}$, since we can safely put in $\mathscr{B}$ all and only the service names which are bound in the process. A naive type inference algorithm based directly on the rules of the type system would require backtracking, resulting in an exponential explosion of the search space. In (Coppo et al., 2013) we define a deterministic, compositional inference algorithm which is proved to be sound and complete with respect to the present interaction type system. The algorithm is given in a "natural deduction" style, as a set of inference rules that can be evaluated in a single-pass analysis according to the structure of processes. The basic idea is to devise a suitable data structure that stores the information about all the possible ways a service initiation can be typed in the interaction type system, postponing the commitment to a specific typing rule as long as possible. The inference algorithm refines the information in this data structure discarding the typing rules of service initiations that are found to be incompatible with the structure of the processes being analysed. We are working toward an implementation of this algorithm for experimenting applicability of our inference system to show progress.

7. 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.

Progress. Our notion of progress is strongly related to, and partly inspired from, the notion of *lock-freedom* in (Kobayashi, 2002), where the author develops a type system to assure it. Intuitively, a process is lock-free if, no matter how it reduces, every top-level prefix can be eventually consumed. In our case this roughly corresponds to the property that no process gets stuck on an input action and that every message in a queue can be received. Kobayashi's type system seems capable of a much more fine-grained analysis than our type system. However, despite the similarities between progress and lock-freedom, the two type systems are difficult to compare, because of several major differences in both processes and types. In addition to the fact that we consider progress modulo the availability of catalysers, our type system is given for an asynchronous language with a native notion of (multiparty) session, while Kobayashi's type system is defined for a basic variant of the synchronous, pure π-calculus. A natural way for comparing these analysis techniques would require compiling a session-based process into the π-calculus (Dardha et al., 2012), and then using Kobayashi's type system for reasoning on progress of the original process in terms of lock-freedom of the one resulting from the compilation. Using this technique we have been able to prove progress for some processes that are ill-typed according to the interaction type system. In general, however, the compilation may also produce processes that are ill-typed according to (Kobayashi, 2002) and, in some cases, Kobayashi's type system is unable to prove progress even for careful encodings of some session-based processes. For example, the process

$$\begin{array}{l}\mathsf{def}\ X(y) = y!\langle 1,75\rangle.\overline{a}[2](z).z!\langle 1,74\rangle.z?(1,x).X\langle y\rangle\ \mathsf{in}\\ \mathsf{def}\ Y(y) = y?(2,x).a[1](z).z?(2,x').z!\langle 2,x\rangle.Y\langle y\rangle\ \mathsf{in}\\ \overline{b}[2](y).X\langle y\rangle \mid b[1](y).Y\langle y\rangle\end{array}$$

which is well typed in both the communication and interaction type systems can be encoded as

$$\begin{array}{l}\ast X?(y).(y!\langle 75\rangle \mid (\nu z)a!\langle z\rangle.z!\langle 74\rangle.z?(x).X!\langle y\rangle)\\ \mid \ast Y?(y).y?(x).a?(z).z?(x').(z!\langle x\rangle \mid Y!\langle y\rangle)\\ \mid (\nu y)(b!\langle y\rangle \mid X!\langle y\rangle) \mid b?(y).Y!\langle y\rangle\end{array}$$

where we represent recursive process definitions with replications, session initiations with bound outputs, and asynchronous communication with output actions without continuations. Yet Kobayashi's type system is unable to prove that this process has the progress property.

A strategy that is alternative to compiling/encoding session-based processes is to lift the technique underlying Kobayashi's type system to a session type system for reasoning directly on the progress properties of processes. Although a formal investigation pursuing this strategy has not been published yet, some preliminary experiments are very promising (Padovani, 2013): not only the obtained type system is simpler than the one defined in the present paper, but it is also capable of proving progress for processes that are ill-typed according to the interaction type system. For example, the process

$$\begin{array}{l}\overline{a}[2](y).\overline{b}[2](z).y?(1,x).z!\langle 1,x\rangle.z?(1,x').y!\langle 1,x'\rangle.y?(1,x'').z!\langle 1,x''\rangle\\ \mid a[1](y).b[1](z).y!\langle 2,74\rangle.z?(2,x).z!\langle 2,75\rangle.y?(2,x').y!\langle 2,x'\rangle.z?(2,x'')\end{array}$$

is not typable in the interaction type system because of the mutual dependencies between the a and b service names, but can be typed using Kobayashi's technique because in that case dependencies are associated with the single actions of a session type, instead of service names. Interestingly, the structure given by sessions seems capable of simplifying some technical aspects of Kobayashi's original type system as well.

The main obstacle to assure progress for the calculus CaSPiS (Calculus of Sessions and Pipelines) (Boreale et al., 2008) is the presence of pipes, since sessions are nested and there is no delegation. For this calculus both (Bruni and Mezzina, 2008) and (Acciai and Boreale, 2008) propose type systems guaranteeing deadlock-freeness: the types in (Acciai and Boreale, 2008) are CCS-like terms and a large set of processes turns out to be typable.

Some ideas for assuring deadlock-freeness for the calculus SSCC (Stream-based Service Centred Calculus) are discussed in (Lanese et al., 2007). The problem in this case is to avoid a service waiting for a value from a stream, which can be produced only after the service execution has been completed.

(Caires and Vieira, 2010) proposes a sophisticated proof system which builds a well-founded ordering on events (similar to the line of (Yoshida, 1996)) to enforce progress for processes of the Conversation Calculus (Vieira et al., 2008), also in presence of dynamic join and leave of participants. Their progress is guaranteed under the assumption that all communications are matched with sufficient joiners.

Formal theories of contracts using multiparty interaction structures are studied in (Castagna and Padovani, 2009). Contracts record the overall behaviour of a process, and typable processes themselves may not always satisfy properties such as progress: it is proved *later* by checking whether a whole contract satisfies a certain form. Proving properties with contracts requires an exploration of all possible interleaved or non-deterministic paths of a protocol.

Most papers on service-oriented calculi only assure that clients are never stuck inside a *single* session (Honda et al., 2008; Dezani-Ciancaglini and de' Liguoro, 2010; Deniélou and Yoshida, 2011). The first papers considering progress for interleaved sessions required the nesting of sessions in Java (Dezani-Ciancaglini et al., 2006; Coppo et al., 2007). These systems can guarantee progress for only one single active binary session. The papers more related to the present one are (Dezani-Ciancaglini et al., 2008) and (Carbone and Debois, 2010). In both these papers there are constructions of processes providing missing participants, which are simpler than our catalysers since the sessions are dyadic.

The definition of progress in (Dezani-Ciancaglini et al., 2008) is the same as that in (Bettini et al., 2008), so it has the problem shown by the process $P_1 \mid P_2 \mid P_3 \mid P_4$ at page 14. The calculus of (Dezani-Ciancaglini et al., 2008) has synchronous communications, a reduction rule for delegation which spawns a new thread with the received channel, and permanent services. Progress is assured by a type system based on a channel/service dependency similar to the present one, but which does not consider nested service sets. Therefore the set of processes typable in (Dezani-Ciancaglini et al., 2008) is strictly included in the set of processes typable by both the communication and the interaction systems, also when restricting to binary sessions. For example the process $R_1 \mid R_2$ shown at page 19 is not typable in the system of (Dezani-Ciancaglini et al., 2008).

In (Carbone and Debois, 2010) like in (Dezani-Ciancaglini et al., 2008) communication is synchronous, there are no recursive definitions of processes, and services can be replicated. Services must be mutually independent. Thanks to these restrictions all closed user processes typable in a (almost standard) session type system enjoy progress according to a definition similar to ours. The proof uses an interesting graphical representation of session invocation interdependency. This methodology allows to assure the progress for the process P_{13} shown at page 24, which cannot be typed in the interaction type system. On the other hand the process $P_1 \mid P_2'$ of page 13 does not satisfy the requirement of mutual independence between services, which is enforced by the type system of (Carbone and Debois, 2010).

Lastly we mention the paper (Buscemi et al., 2012), where progress is assured by taking advantage of constraints, whose solutions correspond to the executions of logic programs.

8. Conclusions

The programming framework presented in this paper relies on the concept of global types that can be seen as the language to describe the model of the distributed communications, i.e., an abstract high-level view of the protocol that all the participants will have to respect in order to communicate in a multiparty session. The programmer will then write the program to implement some communication protocols with possible interleavings; our communication and interaction type systems check the types of the exchanged messages and the progress of the program.

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 (UNIFI, 2002; Honda et al., 2011; Honda et al., 2013). This consists of three layers: the first layer is a global type which corresponds to a signature of class models in UML; the second one is for conversation models where signatures and variables for multiple conversations are integrated; and the third layer includes extensions of the existing languages (such as Java (Hu et al., 2008; Hu et al., 2010)) which implement conversation models. We are currently considering to enrich this modelling framework with our type discipline so that we can specify and ensure progress for executable conversations. The framework of multiparty session types is effectively applicable to dynamic monitoring (Chen et al., 2012; Bocchi et al., 2013) in Python (Hu et al., 2013) for controlling messaging in a large-scale cyberinfrustructure for observing oceans (OOI, 2010). Finally multiparty session types can guide writing safe, deadlock-free message-passing parallel programs: we implemented several languages and built tool-chains (Ng et al., 2012a; Ng et al., 2011; Yoshida et al., 2008; Ng et al., 2012b; Neykova et al., 2013) and a verification framework (Honda et al., 2012; Honda et al., 2013) for high-performance computing which uses extensions of Scribble.

We plan to apply our approach to the multirole calculus of (Deniélou and Yoshida, 2011). We conjecture that for this calculus catalysers could be avoided, since sessions are not stuck when there are no participants in some role.

Acknowledgements We are grateful to Kohei Honda for his comments on an early version of this paper. For session calculi the definition of progress itself is delicate and we acknowledge Marco Carbone, Pierre-Malo Deniélou, Søren Debois and Fabrizio Montesi for enlightening discussions. We are indebted to Naoki Kobayashi for helping us in the comparisons between his and our methodologies and to Gary Brown for his collaboration on an implementation of multiparty session types. Lastly we thanks the Concur reviewers and the reviewers of the present submission for their careful reading, since we deeply revised this article following their suggestions. The third author is partially supported by NSF Ocean Observatiries Initiative and EPSRC EP/G015635/01, EP/K011715/01 and EP/K034413. The first, second and fourth author are partially supported by MIUR PRIN Project CINA Prot. 2010LHT4KM and Torino University/Compagnia San Paolo Project SALT.

References

Acciai, L. and Boreale, M. (2008). A Type System for Client Progress in a Service-Oriented Calculus. In *Concurrency, Graphs and Models*, volume 5065 of *LNCS*, pages 642–658. Springer.

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'13*, 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.

Boreale, M., Bruni, R., De Nicola, R., and Loreti, M. (2008). Sessions and Pipelines for Structured Service Programming. In *FMOODS'08*, volume 5051 of *LNCS*, pages 19–38. Springer.

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

Bruni, R. and Mezzina, L. G. (2008). Types and Deadlock Freedom in a Calculus of Services, Sessions and Pipelines. In *AMAST'08*, volume 5140 of *LNCS*, pages 100–115. Springer.

Buscemi, M. G., Coppo, M., Dezani-Ciancaglini, M., and Montanari, U. (2012). Constraints for Service Contracts. In *TGC'11*, volume 7173 of *LNCS*, pages 104–120. Springer.

Caires, L. and Vieira, H. T. (2010). Conversation Types. *Theoretical Computer Science*, 411(51-52):4399–4440.

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. and Debois, S. (2010). A Graphical Approach to Progress for Structured Communication in Web Services. In *ICE'10*, volume 38 of *EPTCS*, pages 13–27.

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.

Castagna, G. and Padovani, L. (2009). Contracts for Mobile Processes. In *CONCUR'09*, volume 5710 of *LNCS*, pages 211–228. Springer.

Chen, T.-C., Bocchi, L., Deniélou, P.-M., Honda, K., and Yoshida, N. (2012). Asynchronous Distributed Monitoring for Multiparty Session Enforcement. In *TGC'11*, volume 7173 of *LNCS*, pages 25–45. Springer.

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., Padovani, L., and Yoshida, N. (2013). Inference of Global Progress Properties for Dynamically Interleaved Multiparty Sessions. In *COORDINATION'13*, volume 7890 of *LNCS*, pages 45–59. Springer.

Coppo, M., Dezani-Ciancaglini, M., and Yoshida, N. (2007). Asynchronous Session Types and Progress for Object-Oriented Languages. In *FMOODS'07*, volume 4468 of *LNCS*, pages 1–31. Springer.

Dardha, O., Giachino, E., and Sangiorgi, D. (2012). Session Types Revisited. In *PPDP'12*, pages 139–150. ACM Press.

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).

Dezani-Ciancaglini, M. and de' Liguoro, U. (2010). Sessions and Session Types: an Overview. In *WS-FM'09*, volume 6194 of *LNCS*, pages 1–28. Springer.

Dezani-Ciancaglini, M., de' Liguoro, U., and Yoshida, N. (2008). On Progress for Structured Communications. In *TGC'07*, volume 4912 of *LNCS*, pages 257–275. Springer.

Dezani-Ciancaglini, M., Mostrous, D., Yoshida, N., and Drossopoulou, S. (2006). Session Types for Object-Oriented Languages. In *ECOOP'06*, volume 4067 of *LNCS*, pages 328–352. Springer.

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., Marques, E. R. B., Martins, F., Ng, N., Vasconcelos, V. T., and Yoshida, N. (2012). Verification of MPI Programs Using Session Types. In *EuroMPI*, volume 7490 of *LNCS*, pages 291–293. Springer.

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., Vasconcelos, V. T., and Kubo, M. (1998). Language Primitives and Type Disciplines for Structured Communication-based Programming. In *ESOP'98*, volume 1381 of *LNCS*, pages 22–138. Springer.

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

Hu, R., Kouzapas, D., Pernet, O., Yoshida, N., and Honda, K. (2010). Type-Safe Eventful Sessions in Java. In *ECOOP'10*, volume 6183 of *LNCS*, pages 329–353. Springer.

Hu, R., Neykova, R., Yoshida, N., Demangeon, R., and Honda, K. (2013). Practical Interruptible Conversations: Distributed Dynamic Verification with Session Types and Python. In *ICRV'13*, LNCS. Springer. To appear.

Hu, R., Yoshida, N., and Honda, K. (2008). Session-Based Distributed Programming in Java. In *ECOOP'08*, volume 5142 of *LNCS*, pages 516–541. Springer.

Kobayashi, N. (2002). A Type System for Lock-Free Processes. *Information and Computation*, 177:122–159.

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

Lanese, I., Vasconcelos, V. T., Martins, F., and Ravara, A. (2007). Disciplining Orchestration and Conversation in Service-Oriented Computing. In *SEFM'07*, pages 305–314. IEEE Computer Society Press.

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., Yoshida, N., and Hu, R. (2013). SPY:Local Verification of Global Protocols. In *ICRV'13*, LNCS. Springer. To appear.

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

Ng, N., Yoshida, N., Niu, X., Tsoi, K. H., and Luke, W. (2012b). Session Types: towards Safe and Fast Reconfigurable Programming. *SIGARCH Computer Architecture News*, 40(5):22–27.

Ng, N., Yoshida, N., Pernet, O., Hu, R., and Kryftis, Y. (2011). Safe Parallel Programming with Session Java. In *COORDINATION'11*, volume 6721 of *LNCS*, pages 110–126. 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/`.

Padovani, L. (2013). From Lock Freedom to Progress Using Session Types. In *PLACES'13*, EPTCS. to appear.

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`.

Vieira, H. T., Caires, L., and Seco, J. (2008). The Conversation Calculus: A Model of Service-Oriented Computation. In *ESOP'08*, volume 4960 of *LNCS*, pages 269–283. Springer.

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

Yoshida, N. (1996). Graph Types for Monadic Mobile Processes. In *FSTTCS'96*, volume 1180 of *LNCS*, pages 371–386. Springer.

Yoshida, N. and Vasconcelos, V. T. (2007). Language Primitives and Type Disciplines for Structured Communication-based Programming Revisited. In *SecRet'06*, volume 171 of *ENTCS*, pages 73–93. Elsevier.

Yoshida, N., Vasconcelos, V. T., Paulino, H., and Honda, K. (2008). Session-Based Compilation Framework for Multicore Programming. In *FMCO'08*, volume 5751 of *LNCS*, pages 226–246. 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.

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

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

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

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

Table 10. Typing rules for queues.

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 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 10 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 11 (with $\natural \in \{!, \oplus\}$ and $Z \in \{U, l\}$).

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

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

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

Table 11. Equivalence relation on message types.

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 $(\textsc{Inact})$ and $(\textsc{Var})$ 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:

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

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).

To give the rules for typing processes with queues we 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.1 and A.2. Notice that projection is not defined for message types.

Definition A.1. The *partial* projection of the generalised type τ onto q, denoted by $\tau \upharpoonright \mathsf{q}$, is defined by:

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

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

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

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

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

Definition A.2. 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.3. A session environment Δ is *consistent for the session s* (notation $\mathsf{co}(\Delta,s)$) if $s[\mathsf{p}] : \tau \in \Delta$ and $s[\mathsf{q}] : \tau' \in \Delta$ imply $\tau \upharpoonright \mathsf{q} \bowtie \tau' \upharpoonright \mathsf{p}$. A session environment is *consistent* if it is consistent for all sessions which occur in it.

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

Proposition A.4. Let G be a global type and $\mathsf{p} \neq \mathsf{q}$. Then $(G \upharpoonright \mathsf{p}) \upharpoonright \mathsf{q} \bowtie (G \upharpoonright \mathsf{q}) \upharpoonright \mathsf{p}$.

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.

Table 12 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 $(\textsc{GInit})$ 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 $(\textsc{GPar})$) we need to require that each session name is associated to at most one queue (condition $\Sigma \cap \Sigma' = \emptyset$). In rule $(\textsc{GSRes})$ we require the consistency of the session environment Δ with respect to the session name s to be restricted (condition $\mathsf{co}(\Delta,s)$).

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

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

$$\frac{\Gamma \vdash P \triangleright \Delta}{\Gamma \vdash_\emptyset P \triangleright \Delta}\,(\textsc{GInit}) \quad \frac{\Gamma \vdash_\Sigma P \triangleright \Delta \quad \Delta \approx \Delta'}{\Gamma \vdash_\Sigma P \triangleright \Delta'}\,(\textsc{Equiv}) \quad \frac{\Gamma \vdash_\Sigma P \triangleright \Delta \quad \Gamma \vdash_{\Sigma'} Q \triangleright \Delta' \quad \Sigma \cap \Sigma' = \emptyset}{\Gamma \vdash_{\Sigma\cup\Sigma'} P \mid Q \triangleright \Delta * \Delta'}\,(\textsc{GPar})$$

$$\frac{\Gamma \vdash_\Sigma P \triangleright \Delta \quad \mathsf{co}(\Delta,s)}{\Gamma \vdash_{\Sigma\setminus s} (\nu s)P \triangleright \Delta \setminus s}\,(\textsc{GSRes}) \qquad \frac{\Gamma, a : \mathsf{G} \vdash_\Sigma P \triangleright \Delta}{\Gamma \vdash_\Sigma (\nu a : \mathsf{G})P \triangleright \Delta}\,(\textsc{GNRes})$$

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

Table 12. Typing rules for processes.

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 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.

A.2. *Auxiliary Lemmas*

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

Lemma A.5 (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 $\Gamma \vdash e_1 \;\mathsf{and}\; e_2 : S$, then $\Gamma \vdash e_1 : \mathsf{bool}$ and $\Gamma \vdash e_2 : \mathsf{bool}$ and $S = \mathsf{bool}$.
5 If $\Gamma \vdash \overline{a}[\mathrm{p}](y).P \triangleright \Delta$, then $\Gamma \vdash a : \mathsf{G}$ and $\Gamma \vdash P \triangleright \Delta, y : \mathsf{G} \upharpoonright \mathrm{p}$ and $\mathrm{p} = \mathrm{mp}(\mathsf{G})$.
6 If $\Gamma \vdash a[\mathrm{p}](y).P \triangleright \Delta$, then $\Gamma \vdash a : \mathsf{G}$ and $\Gamma \vdash P \triangleright \Delta, y : \mathsf{G} \upharpoonright \mathrm{p}$ and $\mathrm{p} < \mathrm{mp}(\mathsf{G})$.
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 $\Gamma \vdash c?(\mathrm{q}, x).P \triangleright \Delta$, then $\Delta = \Delta', c : ?(\mathrm{q}, S).T$ and $\Gamma, x : S \vdash P \triangleright \Delta', c : T$.
9 If $\Gamma \vdash c!\langle\langle \mathrm{p}, c'\rangle\rangle.P \triangleright \Delta$, then $\Delta = \Delta', c : !\langle \mathrm{p}, \mathsf{T}\rangle.T, c' : \mathsf{T}$ and $\Gamma \vdash P \triangleright \Delta', c : T$.
10 If $\Gamma \vdash c?((\mathrm{q}, y)).P \triangleright \Delta$, then $\Delta = \Delta', c : ?(\mathrm{q}, \mathsf{T}).T$ 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 $\Gamma \vdash c\&(\mathrm{p}, \{l_i : P_i\}_{i \in I}) \triangleright \Delta$, then $\Delta = \Delta', c : \&(\mathrm{p}, \{l_i : T_i\}_{i \in I})$ 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.6 (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 $\Gamma \vdash_{\{s\}} s : h \cdot (\mathrm{q}, \Pi, v) \triangleright \Delta$, then $\Delta \approx \Delta'; \{s[\mathrm{q}] : !\langle \Pi, S\rangle\}$ and $\Gamma \vdash_{\{s\}} s : h \triangleright \Delta'$ and $\Gamma \vdash v : S$.
5 If $\Gamma \vdash_{\{s\}} s : h \cdot (\mathrm{q}, \mathrm{p}, s'[\mathrm{p}']) \triangleright \Delta$, then $\Delta \approx (\Delta'; \{s[\mathrm{q}] : !\langle \mathrm{p}, \mathsf{T}\rangle\}), s'[\mathrm{p}'] : \mathsf{T}$ and $\Gamma \vdash_{\{s\}} s : h \triangleright \Delta'$.

6 If $\Gamma \vdash_{\{s\}} s : h \cdot (\mathsf{q}, \Pi, l) \triangleright \Delta$, then $\Delta \approx \Delta'; \{s[\mathsf{q}] : \oplus\langle \Pi, l\rangle\}$ 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.7.

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

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

Lemma A.8 (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 $\Gamma \vdash P\{s[\mathsf{p}]/y\} \triangleright \Delta, s[\mathsf{p}] : T$.

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

Theorem A.9 (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.6(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.6(1) we get $\Sigma_2 = \emptyset$, which implies $\Sigma = \Sigma_1$, and $\Gamma \vdash \mathbf{0} \triangleright \Delta_2$. Using A.5(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.6(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.6(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.6(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.

— $s : h_1 \cdot (\mathsf{q}, \Pi, v) \cdot (\mathsf{q}', \Pi', v') \cdot h_2 \equiv s : h_1 \cdot (\mathsf{q}', \Pi', v') \cdot (\mathsf{q}, \Pi, v) \cdot h_2$ where $\Pi \cap \Pi' = \emptyset$ or $\mathsf{q} \neq \mathsf{q}'$. We assume $\Pi \cap \Pi' = \emptyset$ and $\mathsf{q} = \mathsf{q}'$, the proof in the case $\mathsf{q} \neq \mathsf{q}'$ being similar and simpler. If $\Gamma \vdash_\Sigma s : h_1 \cdot (\mathsf{q}, \Pi, v) \cdot (\mathsf{q}, \Pi', v') \cdot h_2 \triangleright \Delta$, then $\Sigma = \{s\}$ by Lemma A.6(2). This implies $\Delta = \Delta_1 * \{s[\mathsf{q}] : !\langle \Pi, S\rangle;\ !\langle \Pi', S'\rangle\} * \Delta_2$ 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.7(1). By the same lemma we can derive

$$\Gamma \vdash_{\{s\}} s : h_1 \cdot (\mathsf{q}, \Pi', v') \cdot (\mathsf{q}, \Pi, v) \cdot h_2 \triangleright \Delta_1 * \{s[\mathsf{q}] : !\langle \Pi', S'\rangle;\ !\langle \Pi, S\rangle\} * \Delta_2,$$

and we conclude using rule (EQUIV), since by definition $\Delta_1 * \{s[\mathsf{q}] : !\langle \Pi', S'\rangle;\ !\langle \Pi, S\rangle\} * \Delta_2 \approx \Delta$.

— $s : h_1 \cdot (\mathsf{q}, \Pi, v) \cdot h_2 \equiv s : h_1 \cdot (\mathsf{q}, \Pi', v) \cdot (\mathsf{q}, \Pi'', v) \cdot h_2$ where $\Pi = \Pi' \cup \Pi''$ and $\Pi' \cap \Pi'' = \emptyset$. If $\Gamma \vdash_\Sigma s : h_1 \cdot (\mathsf{q}, \Pi, v) \cdot h_2 \triangleright \Delta$, then $\Sigma = \{s\}$ by Lemma A.6(2). This implies

$$\Delta = \Delta_1 * \{s[\mathsf{q}] : !\langle \Pi, S\rangle\} * \Delta_2 \text{ and } \Gamma \vdash_{\{s\}} s : h_i \triangleright \Delta_i\ (i = 1, 2) \text{ and } \Gamma \vdash v : S$$

by Lemma A.7(1). By the same lemma we can derive

$$\Gamma \vdash_{\{s\}} s : h_1 \cdot (\mathsf{q}, \Pi', v) \cdot (\mathsf{q}, \Pi'', v) \cdot h_2 \triangleright \Delta_1 * \{s[\mathsf{q}] : !\langle \Pi', S\rangle;\ !\langle \Pi'', S\rangle\} * \Delta_2,$$

and we conclude using rule (EQUIV), since by definition $\Delta_1 * \{s[\mathsf{q}] : !\langle \Pi', S\rangle;\ !\langle \Pi'', S\rangle\} * \Delta_2 \approx \Delta$.

$\square$

A.3. *Subject Reduction*

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:

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

where M and τ can be missing and message types are considered modulo the equivalence relation of Table 11.
The first rule corresponds to putting in a queue a message with sender p, set of receivers Π and content of type U. The second rule corresponds to reading from a queue a message with sender p, receiver q and content of type U. The third and fourth rules are similar, but a label is transmitted.

Notice that not all the left-hand-sides of the reduction rules for processes are typed by consistent session environments. For example,

$$\Gamma \vdash_\Sigma s[1]?(2,x).s[1]?(2,y).\mathbf{0} \mid s : (2, \{1\}, \mathsf{true}) \triangleright \{s[1] : ?(2, \mathsf{bool}).?(2, \mathsf{nat}).\mathsf{end}, s : [2] : !\langle \mathsf{bool}, 1\rangle\}$$

Observe that $s[1]?(2,x).s[1]?(2,y).\mathbf{0} \mid s:(2,\{1\},\mathsf{true})$ matches the left-hand-side of the reduction rule [Rcv] and $\{s[1]:?(2,\mathsf{bool}).?(2,\mathsf{nat}).\mathsf{end}, s:[2]:!\langle \mathsf{bool}, 1\rangle\}$ is not consistent. The process obtained by putting this network in parallel with $s[2]!\langle 1,7\rangle.\mathbf{0}$ has a consistent session environment. It is then crucial to show that if the left-hand-side of a reduction rule is typed by a session environment, which is consistent when composed with some other session environment, then the same property holds for the right-hand-side too. It is sufficient to consider the reduction rules which do not contain process reductions as premises, i.e. which are the leaves in the reduction trees. This is formalised in the following lemma, which is the key step for proving the Subject Reduction Theorem.

Lemma A.10. (Key Lemma) Let $\Gamma \vdash_\Sigma P \triangleright \Delta$, and $P \longrightarrow P'$ be obtained by any reduction rule different from [Ctxt], [Str], and $\Delta * \Delta_0$ be consistent, for some Δ_0. Then there is Δ' such that $\Gamma \vdash_\Sigma P' \triangleright \Delta'$ and $\Delta \Rightarrow^* \Delta'$ and $\Delta' * \Delta_0$ is consistent.

Proof. The proof is by cases on process reduction rules. 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.6(1). Using Lemma A.5(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.5(6) on (1) we have

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

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

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

and $\mathrm{mp}(\mathsf{G}) = n$. Using Lemma A.8(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] $s[\mathrm{p}]!\langle \Pi, e\rangle.P \mid s : h \longrightarrow P \mid s : h \cdot (\mathrm{p}, \Pi, v) \;\; (e \downarrow v)$.
By hypothesis, $\Gamma \vdash_\Sigma s[\mathrm{p}]!\langle \Pi, e\rangle.P \mid s : h \triangleright \Delta$. Using Lemma A.6(7), (1), and (2) we have $\Sigma = \{s\}$ and

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

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

where $\Delta = \Delta_2 * \Delta_1$. Using A.5(7) on (8) we have

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

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

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

From (10) by subject reduction on expressions we have

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

Using $(\text{QS}\textsc{end})$ on (9) and (12) we derive

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

Using $(\text{GI}\textsc{nit})$ on (11) we derive

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

and then using $(\text{GP}\textsc{ar})$ on (14), (13) we conclude

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

Note that $\Delta_2 * (\Delta_1', s[\mathrm{p}] : !\langle \Pi, S\rangle.T) \Rightarrow (\Delta_2; \{s[\mathrm{p}] : !\langle \Pi, S\rangle\}) * (\Delta_1', s[\mathrm{p}] : T)$ and the consistency of $(\Delta_2 * (\Delta_1', s[\mathrm{p}] : !\langle \Pi, S\rangle.T)) * \Delta_0$ implies the consistency of $((\Delta_2; \{s[\mathrm{p}] : !\langle \Pi, S\rangle\}) * (\Delta_1', s[\mathrm{p}] : T)) * \Delta_0$.

— [Rcv] $s[\mathrm{p}]?(\mathrm{q}, x).P \mid s : (\mathrm{q}, \{\mathrm{p}\}, v) \cdot h \longrightarrow P\{v/x\} \mid s : h$.
By hypothesis, $\Gamma \vdash_\Sigma s[\mathrm{p}]?(\mathrm{q}, x).P \mid s : (\mathrm{q}, \{\mathrm{p}\}, v) \cdot h \triangleright \Delta$. By Lemma A.6(7), (1), and (2) we have $\Sigma = \{s\}$ and

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

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

where $\Delta = \Delta_2 * \Delta_1$. Using Lemma A.5(8) on (15) we have

$$\Delta_1 = \Delta_1', s[\mathrm{p}] : ?(\mathrm{q}, S).T$$

$$\Gamma, x : S \vdash P \triangleright \Delta_1', s[\mathrm{p}] : T \tag{17}$$

Using Lemma A.7(1) on (16) we have

$$\Delta_2 = \{s[\mathrm{q}] : !\langle \{\mathrm{p}\}, S'\rangle\} * \Delta_2'$$

$$\Gamma \vdash_{\{s\}} s : h \triangleright \Delta_2' \tag{18}$$

$$\Gamma \vdash v : S'. \tag{19}$$

The consistency of $\Delta * \Delta_0$ implies $S = S'$. Using Lemma A.8(1) from (17) and (19) we get $\Gamma \vdash P\{v/x\} \triangleright \Delta_1', s[\mathrm{p}] : T$, which implies by rule (GINIT)

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

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

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

Note that $(\{s[\mathrm{q}] : !\langle\{\mathrm{p}\}, S\rangle\} * \Delta_2') * (\Delta_1', s[\mathrm{p}] : ?(\mathrm{q}, S); T) \ \Rightarrow \Delta_2' * (\Delta_1', s[\mathrm{p}] : T)$ and the consistency of $((\{s[\mathrm{q}] : !\langle\{\mathrm{p}\}, S\rangle\} * \Delta_2') * (\Delta_1', s[\mathrm{p}] : ?(\mathrm{q}, S); T)) * \Delta_0$ implies the consistency of $(\Delta_2' * (\Delta_1', s[\mathrm{p}] : T)) * \Delta_0$.

— [Sel] $s[\mathrm{p}] \oplus \langle \Pi, l \rangle.P \mid s : h \longrightarrow P \mid s : h \cdot (\mathrm{p}, \Pi, l)$.
By hypothesis, $\Gamma \vdash_\Sigma s[\mathrm{p}] \oplus \langle \Pi, l \rangle.P \mid s : h \triangleright \Delta$. Using Lemma A.6(7), (1), and (2) we have $\Sigma = \{s\}$ and

$$\Gamma \vdash s[\mathrm{p}] \oplus \langle \Pi, l \rangle.P \triangleright \Delta_1 \tag{21}$$

$$\Gamma \vdash_{\{s\}} s : h \triangleright \Delta_2 \tag{22}$$

where $\Delta = \Delta_2 * \Delta_1$. Using Lemma A.5(11) on (21) we have for $l = l_j \ (j \in I)$:

$$\Delta_1 = \Delta_1', s[\mathrm{p}] : \oplus\langle \Pi, \{l_i : T_i\}_{i \in I} \rangle$$

$$\Gamma \vdash P \triangleright \Delta_1', s[\mathrm{p}] : T_j. \tag{23}$$

Using rule (QSEL) on (22) we derive

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

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

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

Note that $\Delta_2 * (\Delta_1', s[\mathrm{p}] : \oplus\langle \Pi, \{l_i : T_i\}_{i \in I} \rangle) \ \Rightarrow (\Delta_2; \{s[\mathrm{p}] : \oplus\langle \Pi, l \rangle\}) * (\Delta_1', s[\mathrm{p}] : T_j)$ and the consistency of $(\Delta_2 * (\Delta_1', s[\mathrm{p}] : \oplus\langle \Pi, \{l_i : T_i\}_{i \in I} \rangle)) * \Delta_0$ implies the consistency of $((\Delta_2; \{s[\mathrm{p}] : \oplus\langle \Pi, l \rangle\}) * (\Delta_1', s[\mathrm{p}] : T_j)) * \Delta_0$.

— [Branch] $s[\mathrm{p}]\&(\mathrm{q}, \{l_i : P_i\}_{i \in I}) \mid s : (\mathrm{q}, \{\mathrm{p}\}, l_j) \cdot h \longrightarrow P_j \mid s : h$.
By hypothesis, $\Gamma \vdash_\Sigma s[\mathrm{p}]\&(\mathrm{q}, \{l_i : P_i\}_{i \in I}) \mid s : (\mathrm{q}, \{\mathrm{p}\}, l_j) \cdot h \triangleright \Delta$. Using Lemma A.6(7), (1), and (2) we have $\Sigma = \{s\}$ and

$$\Gamma \vdash s[\mathrm{p}]\&(\mathrm{q}, \{l_i : P_i\}_{i \in I}) \triangleright \Delta_1 \tag{25}$$

$$\Gamma \vdash_{\{s\}} s : (\mathrm{q}, \{\mathrm{p}\}, l_j) \cdot h \triangleright \Delta_2 \tag{26}$$

where $\Delta = \Delta_2 * \Delta_1$. Using Lemma A.5(12) on (25) we have

$$\Delta_1 = \Delta_1', s[\mathrm{p}] : \&(\mathrm{q}, \{l_i : T_i\}_{i \in I})$$

$$\Gamma \vdash P_i \triangleright \Delta_1', s[\mathrm{p}] : T_i \quad \forall i \in I. \tag{27}$$

Using Lemma A.7(3) on (26) we have

$$\Delta_2 = \{s[\mathrm{q}] : \oplus\langle \mathrm{p}, l_j \rangle\} * \Delta_2'$$

$$\Gamma \vdash_{\{s\}} s : h \triangleright \Delta_2'. \tag{28}$$

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

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

and the consistency of $((\{s[\mathtt{q}] : \oplus\langle \mathtt{p}, l_j\rangle\} * \Delta_2') * (\Delta_1', s[\mathtt{p}] : \&(\mathtt{q}, \{l_i : T_i\}_{i\in I}))) * \Delta_0$ implies the consistency of $(\Delta_2' * (\Delta_1', s[\mathtt{p}] : T_j)) * \Delta_0$ for $j \in I$.

□

The main result concerning the communication type system is the subject reduction theorem. The subject reduction for closed user processes (Theorem 4.3) follows immediately.

Theorem A.11 (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 $\Delta \;\Rightarrow^*\; \Delta'$.

Proof. Let $P \equiv \mathscr{E}[P_0]$ and $P' \equiv \mathscr{E}[P_0']$, where $P_0 \longrightarrow P_0'$ by one of the rules considered in Lemma A.10. By structural equivalence we can assume $\mathscr{E} = (\overrightarrow{\nu a : \mathtt{G}})(\overrightarrow{\mathsf{def}\ D\ \mathsf{in}}\ (\overrightarrow{\nu s})([\ ] \mid P_1))$ without loss of generality. Theorem A.9 and Lemma A.6(9), (10) and (8) applied to $\Gamma \vdash_\Sigma P \triangleright \Delta$ give $\Gamma, \overrightarrow{a : \mathtt{G}}, \overrightarrow{X : S\,\mu\mathbf{t}.T} \vdash_{\Sigma_0} P_0 \triangleright \Delta_0$, and $\Gamma, \overrightarrow{a : \mathtt{G}}, \overrightarrow{X : S\,\mu\mathbf{t}.T} \vdash_{\Sigma_1} P_1 \triangleright \Delta_1$ and $\overrightarrow{\Gamma, \overrightarrow{a : \mathtt{G}}, X : S\,\mathbf{t} \vdash Q \triangleright \{y : T\}}$, where $\overrightarrow{D} = \overrightarrow{X(x,y) = Q}$, $\Sigma = (\Sigma_0 \cup \Sigma_1) \setminus \overrightarrow{s}$ and $\Delta = (\Delta_0 * \Delta_1) \setminus \overrightarrow{s}$. The consistency of Δ implies the consistency of $\Delta_0 * \Delta_1$ by Lemma A.6(8). By Lemma A.10 there is Δ_0' such that $\Gamma, \overrightarrow{a : \mathtt{G}}, \overrightarrow{X : S\,\mu\mathbf{t}.T} \vdash_{\Sigma_0} P_0' \triangleright \Delta_0'$ and $\Delta_0 \;\Rightarrow^*\; \Delta_0'$ and $\Delta_0' * \Delta_1$ is consistent. We derive $\Gamma \vdash_\Sigma P' \triangleright \Delta'$, where $\Delta' = (\Delta_0 * \Delta_1') \setminus \overrightarrow{s}$ by applying typing rules (GPAR), (GSRES), (GDEF) and (GNRES). Observe that $\Delta \;\Rightarrow^*\; \Delta'$ and Δ' is consistent. □

Note that communication safety (Honda et al., 2008, Theorem 5.5) is a corollary of Theorem A.11.

Appendix B. Subject Reduction for the Interaction Type System

The structure of this appendix is standard, but for the first lemma which shows that only messages containing channels contribute to csds. Therefore typing of queues is independent from the order of messages.

Lemma B.1. If $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash s : h \;\blacktriangleright\; \mathscr{D}$, then $\mathscr{D} = \{s \prec s' \mid (\mathtt{p}, \mathtt{q}, s'[\mathtt{p}']) \in h\}$.

Proof. Easy by induction on h. □

Lemma B.2 (Substitution Lemma). Let P be well typed in the communication type system and $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P \;\blacktriangleright\; \mathscr{D}$.

1 Let $v \in \mathscr{S}$ implies $v \in \mathscr{N} \cup \mathscr{B}$. Then $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P\{v/x\} \;\blacktriangleright\; \mathscr{D}$.
2 Let $s \notin \mathscr{D}$. Then $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P\{s[\mathtt{p}]/y\} \;\blacktriangleright\; \mathscr{D}\{s/y\}$.

Proof. By induction on $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P \;\blacktriangleright\; \mathscr{D}$.

1 The only interesting case is when v is a service name a. The proof is by structural induction on P. Let $P \equiv \tilde{x}[\mathrm{p}](y).P'$ and the last applied rule be $\{\text{INITV}\}$. From $\{\text{INITV}\}$ we have that $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P' \blacktriangleright \mathscr{D}'$ and $\mathsf{fc}(P') \subseteq \{y\}$ and $\mathscr{D} = \mathscr{D}' \setminus\setminus y$. Now, $P\{a/x\} = \tilde{a}[\mathrm{p}](y).P'\{a/x\}$. By structural induction $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P'\{a/x\} \blacktriangleright \mathscr{D}'$. Since, by hypothesis, $\mathsf{fc}(P') \subseteq \{y\}$ and $a \in \mathscr{N} \cup \mathscr{B}$, we can apply either $\{\text{INITN}\}$ or $\{\text{INITB}\}$, obtaining $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash \tilde{a}[\mathrm{p}](y).P' \blacktriangleright \mathscr{D}$.

2 The proof is standard using the definition of $\curlywedge(c)$.

$\square$

Theorem B.3 (Type Preservation under Equivalence). If P is well typed in the communication type system and $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P \blacktriangleright \mathscr{D}$ and $P \equiv P'$, then $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P' \blacktriangleright \mathscr{D}$.

Proof. By induction on $\equiv$ using Lemma B.1 for the equivalences on queue. $\square$

Proof of Theorem 6.2 (Subject Reduction) By induction on $\longrightarrow$ and by cases on the last applied rule.

— [Init] By hypothesis

$$\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash a[1](y).P_1 \mid \ldots \mid a[n-1](y).P_{n-1} \mid \overline{a}[n](y).P_n \blacktriangleright \mathscr{D}.$$

This judgement is obtained by applying the inference rule $\{\text{PAR}\}$ to the subprocesses $a[1](y).P_1,\ldots, a[n-1](y).P_{n-1},\overline{a}[n](y).P_n$. Then we have:

- $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash a[1](y).P_1 \blacktriangleright \mathscr{D}_1$
- $\ldots$
- $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash a[n-1](y).P_{n-1} \blacktriangleright \mathscr{D}_{n-1}$
- $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash \overline{a}[n](y).P_n \blacktriangleright \mathscr{D}_n$

where $\mathscr{D} = (\bigcup_{1\le i\le n} \mathscr{D}_i)^+$ is irreflexive. We consider the case $a \in \mathscr{N}$, the other cases being similar.
For each i ($1 \le i \le n$) we must have $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P_i \blacktriangleright \mathscr{D}'_i$ such that $\mathscr{D}_i = \mathscr{D}'_i \setminus\setminus y$. Notice that y is minimal in $\mathscr{D}'_i$. By construction s is fresh and so by Lemma B.2(2) we have

$$\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P_i\{s[i]/y\} \blacktriangleright \mathscr{D}'_i\{s/y\}.$$

By using $\{\text{QINIT}\}$ and $\{\text{PAR}\}$ we derive

$$\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P_1\{s[1]/y\}|\ldots|P_n\{s[n]/y\}|s:\varnothing \blacktriangleright \mathscr{D}'$$

where $\mathscr{D}' = (\bigcup_{1\le i\le n} \mathscr{D}'_i\{s/y\})^+$. Note that $\mathscr{D}'$ is irreflexive since $\mathscr{D}$ is irreflexive and s is minimal in $\mathscr{D}'$.
By using $\{\text{SRES}\}$ we conclude

$$\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash (\nu s)(P_1\{s[1]/y\}|\ldots|P_n\{s[n]/y\}|s:\varnothing) \blacktriangleright \mathscr{D}' \setminus s$$

Finally it is easy to see that $\mathscr{D}' \setminus s = \mathscr{D}$ by the minimality of the y in $\mathscr{D}'_i$ for all $i \in I$ and of s in $\mathscr{D}'$.

— [Send] By hypothesis, $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash s[\mathrm{p}]!\langle\Pi,e\rangle.P \mid s:h \blacktriangleright \mathscr{D}$, which is obtained by applying rule $\{\text{PAR}\}$. Thus, we get

$$\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash s[\mathrm{p}]!\langle\Pi,e\rangle.P \blacktriangleright \mathscr{D}_1 \qquad \Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash s:h \blacktriangleright \mathscr{D}_2$$

where $\mathscr{D} = (\mathscr{D}_1 \cup \mathscr{D}_2)^+$. The first judgement can only be obtained by $\{\text{SEND}\}$, i.e., $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P \blacktriangleright \mathscr{D}_1$ and $e \in \mathscr{S}$ implies $e \in \mathscr{N} \cup \mathscr{B}$. By using rules $\{\text{QADDVAL}\}$ and $\{\text{PAR}\}$ we conclude

$$\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P \mid s : h \cdot (\mathtt{p},\Pi,v) \blacktriangleright (\mathscr{D}_1 \cup \mathscr{D}_2)^+$$

where $e \downarrow v$.

— [Deleg] By reasoning as in the previous case, we get

$$\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash s[\mathtt{p}]!\langle\langle \mathtt{q}, s'[\mathtt{p}']\rangle\rangle.P \blacktriangleright \mathscr{D}_1 \qquad \Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash s : h \blacktriangleright \mathscr{D}_2$$

where $\mathscr{D} = (\mathscr{D}_1 \cup \mathscr{D}_2)^+$. By inverting rule $\{\text{DELEG}\}$ we obtain $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P \blacktriangleright \mathscr{D}'_1$ where $\mathscr{D}_1 = \{s \prec s'\} \cup \mathscr{D}'_1$. By using rules $\{\text{QADDSESS}\}$ and $\{\text{PAR}\}$ we conclude

$$\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P \mid s : h \cdot (\mathtt{q},\mathtt{p},s'[\mathtt{p}']) \blacktriangleright \mathscr{D}'_1 \cup \{s \prec s'\} \cup \mathscr{D}_2.$$

— [Sel] Similar to [Send] but simpler (using rule $\{\text{QSEL}\}$ instead of $\{\text{QADDVAL}\}$).

— [Rcv] By hypothesis, $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash s[\mathtt{p}]?(\mathtt{q},x).P \mid s : (\mathtt{q},\{\mathtt{p}\},v) \cdot h \blacktriangleright \mathscr{D}$. By reasoning as in the case of rule [Send], we get

$$\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash s[\mathtt{p}]?(\mathtt{q},x).P \blacktriangleright \mathscr{D}_1 \qquad \Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash s : (\mathtt{q},\{\mathtt{p}\},v) \cdot h \blacktriangleright \mathscr{D}_2$$

where $\mathscr{D} = (\mathscr{D}_1 \cup \mathscr{D}_2)^+$. By inverting rule $\{\text{RCV}\}$ we obtain $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P \blacktriangleright \mathscr{D}'_1$, where $\mathscr{D}_1 = (\mathsf{pre}(s[\mathtt{p}],\mathsf{fc}(P)) \cup \mathscr{D}'_1)^+$. By inverting rule $\{\text{QADDVAL}\}$ we have $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash s : h \blacktriangleright \mathscr{D}_2$ and $v \in \mathscr{S}$ implies $v \in \mathscr{N} \cup \mathscr{B}$. By Lemma B.2(1) $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P\{v/x\} \blacktriangleright \mathscr{D}'_1$. Applying $\{\text{PAR}\}$ we conclude

$$\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P\{v/x\} \mid s : h \blacktriangleright (\mathscr{D}'_1 \cup \mathscr{D}_2)^+.$$

Note that $(\mathscr{D}'_1 \cup \mathscr{D}_2)^+ \subseteq (\mathscr{D}_1 \cup \mathscr{D}_2)^+$.

— [SRcv] By hypothesis, $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash s[\mathtt{p}]?((\mathtt{q},y)).P \mid s : (\mathtt{q},\mathtt{p},s'[\mathtt{p}']) \cdot h \blacktriangleright \mathscr{D}$. As before we get

$$\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash s[\mathtt{p}]?((\mathtt{q},y)).P \blacktriangleright \mathscr{D}_1 \qquad \Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash s : (\mathtt{q},\mathtt{p},s'[\mathtt{p}']) \cdot h \blacktriangleright \mathscr{D}_2$$

where $\mathscr{D} = (\mathscr{D}_1 \cup \mathscr{D}_2)^+$. Inverting rule $\{\text{SRCV}\}$) we have $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P \blacktriangleright \mathscr{D}'_1$ where $\mathscr{D}_1 = \mathscr{D}'_1 \setminus \{y\}$ and $\mathscr{D}'_1 \setminus \mathscr{S} \subseteq \{s \prec y\}$. Moreover, by Lemma B.1 it folllows that $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash s : h \blacktriangleright \mathscr{D}'_2$ with $\mathscr{D}_2 = (\{s \prec s'\} \cup \mathscr{D}'_2)^+$. By Lemma B.2(2), it follows that $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P\{s'[\mathtt{p}']/y\} \blacktriangleright \mathscr{D}''_1$ where $\mathscr{D}''_1 = \mathscr{D}'_1\{s'/y\}$. By applying rule $\{\text{PAR}\}$ we conclude

$$\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P\{s'[\mathtt{p}']/y\} \mid s : h \blacktriangleright (\mathscr{D}''_1 \cup \mathscr{D}'_2)^+.$$

Lastly it is easy to see that that $(\mathscr{D}''_1 \cup \mathscr{D}'_2)^+ \subseteq \mathscr{D}$.

— [Branch] By hypothesis, $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash s[\mathtt{p}]\&(\mathtt{q},\{l_i : P_i\}_{i\in I}) \mid s : (\mathtt{q},\Pi,l_j) \cdot h \blacktriangleright \mathscr{D}$. By inverting the rules we have

- $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P_i \blacktriangleright \mathscr{D}_i \quad \forall i \in I$
- $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash s : (\mathtt{q},\{\mathtt{p}\},l_j) \cdot h \blacktriangleright \mathscr{D}'$

– $\mathscr{D} = (\mathsf{pre}(s[\mathrm{p}], \bigcup_{i\in I} \mathsf{fc}(P_i)) \cup \bigcup_{i\in I} \mathscr{D}_i \cup \mathscr{D}')^+$.
By Lemma B.1 we get
$$\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash s : h \blacktriangleright \mathscr{D}'.$$
By applying rule $\{\textsc{Par}\}$ to the reduced process we conclude
$$\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P_j \mid s : h \blacktriangleright \mathscr{D}_j \cup \mathscr{D}'$$
which implies the result.

— [If-T], [If-F] Straightforward.

— [ProcCall] Let's assume $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash \mathsf{def}\ X(x,y) = P \ \mathsf{in}\ (X\langle e, s[\mathrm{p}]\rangle \mid Q) \blacktriangleright \mathscr{D}$. By inspecting the inference rule, as before, we must have:

1 $\Theta';\mathscr{R};\mathscr{N};\mathscr{B} \vdash P \blacktriangleright \mathscr{D}'$;
2 $\Theta';\mathscr{R};\mathscr{N};\mathscr{B} \vdash X\langle e, s[\mathrm{p}]\rangle \blacktriangleright \mathscr{D}'\{s/y\}$;
3 $\Theta';\mathscr{R};\mathscr{N};\mathscr{B} \vdash Q \blacktriangleright \mathscr{D}''$;

where $\Theta' = \Theta, X[y] \blacktriangleright \mathscr{D}'$ and $\mathscr{D} = (\mathscr{D}'\{s/y\} \cup \mathscr{D}'')^+$ and $e \in \mathscr{S}$ implies $e \in \mathscr{N} \cup \mathscr{B}$.
Note that by rule $(\textsc{Def})$ y is the only free channel which can occur P and then $s \notin \mathscr{D}'$. Let $e \downarrow v$. By Lemma B.2(1) and (2) we have $\Theta';\mathscr{R};\mathscr{N};\mathscr{B} \vdash P\{v/x\}\{s[\mathrm{p}]/y\} \blacktriangleright \mathscr{D}'\{s/y\}$.
By rule $\{\textsc{Par}\}$ we derive $\Theta';\mathscr{R};\mathscr{N};\mathscr{B} \vdash P\{v/x\}\{s[\mathrm{p}]/y\} \mid Q \blacktriangleright \mathscr{D}$. By rule $\{\textsc{Def}\}$ we conclude
$$\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash \mathsf{def}\ X(x,y) = P \ \mathsf{in}\ (P\{v/x\}\{s[\mathrm{p}]/y\} \mid Q) \blacktriangleright \mathscr{D}.$$

— [Ctxt] The thesis follows from the induction hypothesis.
— [Str] The thesis follows from Theorem B.3 and the induction hypothesis.

□

Appendix C. Proof of the Progress Theorem

In this section we will prove that a process typable in both type systems has the progress property. Typability in the communication type system is needed as shown by the simple example $a[1](y).y!\langle 2,\mathsf{true}\rangle.y?(2,x).\mathbf{0} \mid \overline{a}[2](y).y?(1,x').y!\langle 2,x'+1\rangle.\mathbf{0}$, which reduces to a stuck process on the evaluation of $1+\mathsf{true}$.

Our proof will go through the following steps:

- We first enrich processes by adding a *"session mark"* $s : \mathsf{G}$ for each current session name s, where G is an *extended closed global type* which represents the actions that are still expected in session s. Note that recursive global types, through unfolding, can represent infinite behaviours. *"Marked" reductions* will update the session marks according to the actions performed by the process. The Inversion Lemmas (Lemmas A.5 and A.6) of the communication type system assure, essentially, that session marks follows exactly the reductions of processes (§C.1).
- We then *"approximate"* global types by allowing only a finite number of unfoldings, after which the global types are ended and the corresponding reductions (recursive calls in particular) are forbidden. We need to start from global types which are unfolded as least as the

processes they type. For this reason we consider an iso-recursive version of the communication type system with a subtyping for global types allowing unfolding. Through a suitable notion of measure we can then prove that marked reductions with approximate global types in session marks always terminate. Until their end is not reached the approximate global types follow the exact ones, assuring the correct matching between process actions and session marks (§C.2).

- Lastly we will introduce a notion of *"pseudo-progress"* as a handy tool to prove Theorem 6.4. Pseudo-progress requires that choosing suitable catalysers all approximated marked reductions terminate with only end as global type in session marks. We show that initial processes (see Definition 6.3) have pseudo-progress. The finiteness of reduction allows also to build the catalysers used in both the definitions of progress and pseudo progress. Finally the pseudo-progress will be proved to imply progress as defined in Definition 5.4 (§C.3).

C.1. *Typed Operational Semantics*

In our typed operational semantics the type information (represented by global types) is explicitly added to processes via session marks. The global type associated to a session will be "consumed" following the execution of the actions it represents. To allow this we need *extended global types* which are defined by adding to the syntax of global types (§4.1) the clauses:

$$G := \mathtt{p} \dashrightarrow \Pi : \langle U \rangle.G \;\mid\; \mathtt{p} \dashrightarrow \Pi : \langle l \rangle.G$$

The meaning of $\mathtt{p} \dashrightarrow \Pi : \langle U \rangle.G$ is that the output of a value or channel of type U has been executed by participant p and that the value or channel is currently on the associated queue waiting to be received by the participants in Π. The meaning of $\mathtt{p} \dashrightarrow \Pi : \langle l \rangle.G$ is similar.

From now on we will often use simply "global type" to refer to extended global types.

The functions γ, δ (see Table 13) register, respectively, the execution of a value, channel or label communication performed by a participant q on an extended closed global type G, so they can be undefined when no more communication actions for q are allowed by G. The function δ has also the communicated label as parameter. This is used to simplify multiple choices in the global type after that a label corresponding to that choice has been sent. Notice that both functions are undefined for the global type end. By "-" we denote either an exchange type or a label.

The syntax of *marked* processes (range over by P) is defined as in Table 2 by adding the clause:

$$\mathsf{P} ::= s : \mathsf{G} \qquad \textit{session mark}$$

where G is an extended closed global type and s is a session name. We say that G is the *mark* of s. Marks occurring in a process P are intended to correspond in a one-one way to the session names (either public of private) occurring in P. A closed user process then, having no opened sessions, can also be seen as a (trivially) marked process. The *erasure* $|\mathsf{P}|$ of a marked process P is defined as the process obtained by deleting from it all session marks. We extend evaluation contexts (see Table 2) to marked processes by adding session marks and we use $\mathscr{E}$ to range over them.

$$\gamma(\mathsf{p} \to \Pi : \langle U \rangle.\mathsf{G}, \mathsf{q}) = \begin{cases} \mathsf{p} \dashrightarrow \Pi : \langle U \rangle.\mathsf{G} & \text{if } \mathsf{p} = \mathsf{q}, \\ \mathsf{p} \to \Pi : \langle U \rangle.\gamma(\mathsf{G}, \mathsf{q}) & \text{otherwise.} \end{cases}$$

$$\gamma(\mathsf{p} \to \Pi : \{l_i : \mathsf{G}_i\}_{i \in I}, \mathsf{q}) = \begin{cases} \mathsf{p} \to \Pi : \{l_i : \gamma(\mathsf{G}_i, \mathsf{q})\}_{i \in I} & \text{if } \mathsf{q} \notin \Pi \cup \{\mathsf{p}\} \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\gamma(\mathsf{p} \dashrightarrow \Pi : \langle\text{-}\rangle.\mathsf{G}, \mathsf{q}) = \begin{cases} \mathsf{G} & \text{if } \Pi = \{\mathsf{q}\}, \\ \mathsf{p} \dashrightarrow \Pi \setminus \mathsf{q} : \langle\text{-}\rangle.\mathsf{G} & \text{if } \mathsf{q} \in \Pi \text{ and } \Pi \setminus \mathsf{q} \neq \emptyset \\ \mathsf{p} \dashrightarrow \Pi : \langle\text{-}\rangle.\gamma(\mathsf{G}, \mathsf{q}) & \text{otherwise} \end{cases}$$

$$\gamma(\mu\mathbf{t}.G, \mathsf{q}) = \gamma(G\{\mu\mathbf{t}.G/\mathbf{t}\}, \mathsf{q})$$

$$\delta(\mathsf{p} \to \Pi : \langle U \rangle.\mathsf{G}, \mathsf{q}, l) = \begin{cases} \mathsf{p} \to \Pi : \langle U \rangle.\delta(\mathsf{G}, \mathsf{q}, l) & \text{if } \mathsf{q} \notin \Pi \cup \{\mathsf{p}\} \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\delta(\mathsf{p} \to \Pi : \{l_i : \mathsf{G}_i\}_{i \in I}, \mathsf{q}, l) = \begin{cases} \mathsf{p} \dashrightarrow \Pi : \langle l_i \rangle.\mathsf{G}_i & \text{if } \mathsf{p} = \mathsf{q} \text{ and } l = l_i \\ \mathsf{p} \to \Pi : \{l_i : \delta(\mathsf{G}_i, \mathsf{q}, l)\}_{i \in I} & \text{otherwise} \end{cases}$$

$$\delta(\mathsf{p} \dashrightarrow \Pi : \langle\text{-}\rangle.\mathsf{G}, \mathsf{q}, l) = \begin{cases} \mathsf{p} \dashrightarrow \Pi : \langle\text{-}\rangle.\delta(\mathsf{G}, \mathsf{q}, l) & \text{if } \mathsf{q} \notin \Pi \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\delta(\mu\mathbf{t}.G, \mathsf{q}, l) = \delta(G\{\mu\mathbf{t}.G/\mathbf{t}\}, \mathsf{q}, l)$$

Table 13. The functions γ and δ.

Table 14 gives the reduction rules parameterised on a standard environment Γ for marked processes. The environment is used in session initiation to generate the right session mark. So the global types which occur in marked processes are types of initiated sessions, while the global types which occur in standard environments are types of services. We call *marked reductions* or Γ-*reductions* (if we want to specify the environment) such reductions. Note that in correspondence of output and input actions we update the marks of the involved sessions using the functions γ, δ. So the reduction rules give a correspondence between communication actions and applications of γ, δ. We use $\xrightarrow{\Gamma}^{*}$ and $\xrightarrow{\Gamma}^{n}$ with the standard meanings. It is understood that a reduction cannot be performed if the corresponding γ or δ function is undefined.

In the remaining of this § we will show the correspondence between marked reductions (Table 14) and the reductions defined in Table 3 (Lemma C.1 and Theorem C.4).

As expected there is no problem in getting standard reductions which mimic marked reductions.

Lemma C.1. If $\mathsf{P}_0 \xrightarrow{\Gamma}^{*} \mathsf{P}$, then $|\mathsf{P}_0| \longrightarrow^{*} |\mathsf{P}|$.

Proof. Easy since the rules of Table 14 only add conditions to the rules of Table 3. □

To show the vice versa we need to extended session types to mirror global types in which alternative choices are simplified according to the δ function. The *extended session types* are

$$a[1](y).P_1 \mid ... \mid a[n-1](y).P_n \mid \overline{a}[n](y).P_n \xrightarrow{\Gamma}$$
$$(\nu s)(P_1\{s[1]/y\} \mid ... \mid P_n\{s[n]/y\} \mid s:\varnothing \mid s:\mathsf{G}) \quad (\Gamma \vdash a:\mathsf{G}) \qquad \text{[InitM]}$$

$$s[\mathrm{p}]!\langle\Pi,e\rangle.P \mid s:h \mid s:\mathsf{G} \xrightarrow{\Gamma} P \mid s:h\cdot(\mathrm{p},\Pi,v) \mid s:\gamma(\mathsf{G},\mathrm{p}) \quad (e{\downarrow}v) \qquad \text{[SendM]}$$

$$s[\mathrm{p}]!\langle\langle \mathrm{q},s'[\mathrm{p}']\rangle\rangle.P \mid s:h \mid s:\mathsf{G} \xrightarrow{\Gamma} P \mid s:h\cdot(\mathrm{p},\mathrm{q},s'[\mathrm{p}']) \mid s:\gamma(\mathsf{G},\mathrm{p}) \qquad \text{[DelegM]}$$

$$s[\mathrm{p}]\oplus\langle\Pi,l\rangle.P \mid s:h \mid s:\mathsf{G} \xrightarrow{\Gamma} P \mid s:h\cdot(\mathrm{p},\Pi,l) \mid s:\delta(\mathsf{G},\mathrm{p},l) \qquad \text{[SelM]}$$

$$s[\mathrm{p}]?(\mathrm{q},x).P \mid s:(\mathrm{q},\mathrm{p},v)\cdot h \mid s:\mathsf{G} \xrightarrow{\Gamma} P\{v/x\} \mid s:h \mid s:\gamma(\mathsf{G},\mathrm{p}) \qquad \text{[RcvM]}$$

$$s[\mathrm{p}]?((\mathrm{q},y)).P \mid s:(\mathrm{q},\mathrm{p},s'[\mathrm{p}'])\cdot h \mid s:\mathsf{G} \xrightarrow{\Gamma} P\{s'[\mathrm{p}']/y\} \mid s:h \mid s:\gamma(\mathsf{G},\mathrm{p}) \qquad \text{[SRcvM]}$$

$$s[\mathrm{p}]\&(\mathrm{q},\{l_i:P_i\}_{i\in I}) \mid s:(\mathrm{q},\mathrm{p},l_j)\cdot h \mid s:\mathsf{G} \xrightarrow{\Gamma} P_j \mid s:h \mid s:\gamma(\mathsf{G},\mathrm{p}) \quad (j\in I) \qquad \text{[BranchM]}$$

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

$$\mathsf{def}\ X(x,y)=P\ \mathsf{in}\ (X\langle e,s[\mathrm{p}]\rangle \mid s:\mathsf{G} \mid \mathsf{P}) \xrightarrow{\Gamma} \mathsf{def}\ X(x,y)=P\ \mathsf{in}\ (P\{v/x\}\{s[\mathrm{p}]/y\} \mid s:\mathsf{G} \mid \mathsf{P}) \quad (e\downarrow v) \qquad \text{[ProcCallM]}$$

$$\mathsf{P} \xrightarrow{\Gamma,a:\mathsf{G}} \mathsf{P}' \Rightarrow (\nu a:\mathsf{G})\mathsf{P} \xrightarrow{\Gamma} (\nu a:\mathsf{G})\mathsf{P}' \qquad \mathsf{P} \xrightarrow{\Gamma} \mathsf{P}' \Rightarrow (\nu s)\mathsf{P} \xrightarrow{\Gamma} (\nu s)\mathsf{P}' \qquad \text{[ScopM]}$$

$$\mathsf{P} \xrightarrow{\Gamma} \mathsf{P}' \Rightarrow \mathsf{P} \mid \mathsf{P}_0 \xrightarrow{\Gamma} \mathsf{P}' \mid \mathsf{P}_0 \qquad \text{[ParM]}$$

$$\mathsf{P} \xrightarrow{\Gamma} \mathsf{P}' \Rightarrow \mathsf{def}\ D\ \mathsf{in}\ \mathsf{P} \xrightarrow{\Gamma} \mathsf{def}\ D\ \mathsf{in}\ \mathsf{P}' \qquad \text{[DefinM]}$$

$$\mathsf{P}_1 \equiv \mathsf{P}_1' \text{ and } \mathsf{P}_1' \xrightarrow{\Gamma} \mathsf{P}_2' \text{ and } \mathsf{P}_2' \equiv \mathsf{P}_2 \Rightarrow \mathsf{P}_1 \xrightarrow{\Gamma} \mathsf{P}_2 \qquad \text{[StrM]}$$

Table 14. Γ-reduction rules.

obtained by adding to the syntax of session types (Table 5) the clause:

$$T := \&(\mathrm{p},l).T \qquad \textit{unique branch}$$

Extended session environments are defined as expected.

The projection $G \upharpoonright \mathrm{q}$ from extended global types to extended session types is defined by adding to Definition 4.1:

$$(\mathrm{p}\dashrightarrow\Pi:\langle U\rangle.G)\upharpoonright\mathrm{q} = \begin{cases} !\langle\Pi,U\rangle;(G\upharpoonright\mathrm{q}) & \text{if } \mathrm{q}=\mathrm{p},\\ ?(\mathrm{p},U).(G\upharpoonright\mathrm{q}) & \text{if } \mathrm{q}\in\Pi,\\ G\upharpoonright\mathrm{q} & \text{otherwise.}\end{cases} \quad (\mathrm{p}\dashrightarrow\Pi:\langle l\rangle.G)\upharpoonright\mathrm{q} = \begin{cases} \oplus\langle\Pi,l\rangle;(G\upharpoonright\mathrm{q}) & \text{if } \mathrm{q}=\mathrm{p}\\ \&(\mathrm{p},l).(G\upharpoonright\mathrm{q}) & \text{if } \mathrm{q}\in\Pi\\ G\upharpoonright\mathrm{q} & \text{otherwise.}\end{cases}$$

Note that since in an extended global type a dashed arrow of the form $\mathrm{p}\dashrightarrow\Pi:\langle U\rangle.G$ represents an output action already performed (with the corresponding value on the queue), its projection onto p starts with a message type (see §A.1). Similarly the projection of $\mathrm{p}\dashrightarrow\Pi:\langle l\rangle.G$ onto p starts with a message type. Instead the projection of $\mathrm{p}\dashrightarrow\Pi:\langle l\rangle.G$ onto $\mathrm{q}\in\Pi$ is a unique branch type.

By projecting extended global types we obtain extended session environments, so to relate session marks with session environments we give a mapping from session environments to extended session environments. The mapping is defined by means of a reduction rule which replaces branching types by unique branch types when the session environment already contains the message selection type. We need some definitions. The syntax of contexts for extended session types (ranged over by $\mathscr{T}$) is given by:

$$\begin{array}{lll}\mathscr{T} & ::= & [\,] \mid !\langle\Pi,U\rangle.\mathscr{T} \mid ?(\mathtt{p},U).\mathscr{T} \mid \oplus\langle\Pi,\{l:\mathscr{T},l_i:T_i\}_{i\in I}\rangle \mid \&(\mathtt{p},\{l:\mathscr{T},l_i:T_i\}_{i\in I}) \mid \\ & & !\langle\Pi,U\rangle;\mathscr{T} \mid \oplus\langle\Pi,l\rangle;\mathscr{T} \mid \&(\mathtt{p},l).\mathscr{T} \mid \mu\mathbf{t}.\mathscr{T}\end{array}$$

The projection of contexts for extended session types is obtained by adding to the clauses of Definition A.1:

$$[\,]\restriction\mathtt{p}=[\,] \qquad \&(\mathtt{p},l).\mathscr{T}\restriction\mathtt{q}=\begin{cases}\&l.\mathscr{T}\restriction\mathtt{q} & \text{if } \mathtt{p}=\mathtt{q},\\ \mathscr{T}\restriction\mathtt{q} & \text{otherwise.}\end{cases}$$

The duality relation between projections of contexts for extended session types is obtained from Definition A.2 by erasing the last clause and by adding the clauses:

$$[\,]\bowtie[\,] \qquad \mathbb{T}\bowtie\mathbb{T}' \implies \oplus l;\mathbb{T}\bowtie\&l.\mathbb{T}'$$

where $\mathbb{T}$ ranges over projections of contexts for extended session types.

We can then define the reduction rule as:

$$\{s[\mathtt{q}]:\mathscr{T}[\&(\mathtt{p},\{l_i:T_i\}_{i\in I})],s[\mathtt{p}]:\mathscr{T}'[\oplus\langle\Pi,l_j\rangle;\tau]\}\rightsquigarrow\{s[\mathtt{q}]:\mathscr{T}[\&(\mathtt{p},l_j).T_j],s[\mathtt{p}]:\mathscr{T}'[\oplus\langle\Pi,l_j\rangle;\tau]\}$$
$$\text{if } \mathtt{q}\in\Pi \text{ and } j\in I \text{ and } \mathscr{T}\restriction\mathtt{p}\bowtie\mathscr{T}'\restriction\mathtt{q}$$

where the condition $\mathscr{T}\restriction\mathtt{p}\bowtie\mathscr{T}'\restriction\mathtt{q}$ guarantees that a message selection modifies a branching only if they "correspond to each other". In order to formalise this correspondence we introduce contexts for global types (ranged over by $\mathscr{C}$) defined by:

$$\mathscr{C} ::= [\,] \mid \mathtt{p}\to\Pi:\langle U\rangle.\mathscr{C} \mid \mathtt{p}\to\Pi:\{l:\mathscr{C},l_i:G_i\}_{i\in I} \mid \mu\mathbf{t}.\mathscr{C} \mid \mathtt{p}\dashrightarrow\Pi:\langle U\rangle.\mathscr{C} \mid \mathtt{p}\dashrightarrow\Pi:\langle l\rangle.\mathscr{C}$$

Then we can show:

Lemma C.2. If $\mathscr{T}\restriction\mathtt{p}\bowtie\mathscr{T}'\restriction\mathtt{q}$, then there is a global type context $\mathscr{C}$ such that $\mathscr{T}=\mathscr{C}\restriction\mathtt{q}$ and $\mathscr{T}'=\mathscr{C}\restriction\mathtt{p}$.

Proof. Standard by cases on the definitions of projection and duality. For example let $\mathscr{T}=\oplus\langle\Pi,l\rangle;\mathscr{T}_1$ and $\mathscr{T}'=\&(\mathtt{p},l).\mathscr{T}_2$, then $\mathscr{T}_1\restriction\mathtt{p}\bowtie\mathscr{T}_2\restriction\mathtt{q}$ by definition of duality. By induction there is a context $\mathscr{C}'$ such that $\mathscr{T}_1=\mathscr{C}'\restriction\mathtt{q}$ and $\mathscr{T}_2=\mathscr{C}'\restriction\mathtt{p}$. Therefore we can choose $\mathscr{C}=\mathtt{p}\dashrightarrow\Pi:\langle l\rangle.\mathscr{C}'$. □

We denote by $\eta(\Delta)$ the extended session environment obtained by applying the reduction rule ($\rightsquigarrow$) to subsets of the session environment Δ whenever possible.

We can now formulate the relation between session marks and extended session environments (Lemma C.3) which will be the key to show that marked reductions mimic standard reduction (Theorem C.4).

Lemma C.3. Let P_0 be a closed pure process such that $\Gamma\vdash P_0\triangleright\emptyset$. If $P_0\xrightarrow{\Gamma}{}^{*}(\nu s)(\mathsf{P}\mid s:\mathsf{G})$ and $\Gamma\vdash|\mathsf{P}|\triangleright\Delta$, then

$$\{s[\mathtt{p}]:\mathsf{G}\restriction\mathtt{p}\mid\mathsf{G}\restriction\mathtt{p}\neq\mathsf{end}\}=\eta(\{s[\mathtt{p}]:T\mid s[\mathtt{p}]:T\in\Delta\wedge T\neq\mathsf{end}\}).$$

Proof. By Lemma C.1 and the Subject Reduction Theorem $|\mathsf{P}|$ is typable from Γ in the communication type system. The proof is by induction on Γ-reductions and by cases on the last applied rule using the inversion Lemmas A.5 and A.6 and the consistency of session environments (Definition A.3).

We only consider the case in which the last applied rule is [SelM], the other cases being similar and simpler. Let $\mathsf{P} \equiv (\nu \overrightarrow{a:\mathsf{G}})\mathscr{E}[s[\mathrm{p}] \oplus \langle \Pi, l\rangle.P \mid s:h]$ where $\mathscr{E}$ does not contain service restrictions. We get:

$$s[\mathrm{p}] \oplus \langle \Pi, l\rangle.P \mid s:h \mid s:\mathsf{G} \xrightarrow{\Gamma,\overrightarrow{a:G}} P \mid s:h\cdot(\mathrm{p},\Pi,l) \mid s:\delta(\mathsf{G},\mathrm{p},l)$$

Let Δ, Δ' be the session environments obtained by typing the erasures of the processes in the left-hand-side and in the right-hand-side of the shown reduction step, respectively.

By Lemmas A.5(11) and A.6(7) the type of $s[\mathrm{p}]$ in Δ must be $\mathscr{T}[\oplus\langle\Pi,\{l_i:T_i\}_{i\in I}\rangle]$ for some $\mathscr{T}$, T_i, l_i such that $l=l_j$ and $j\in I$. By the consistency of Δ this implies that the types of $s[\mathrm{q}]$ in Δ for all $\mathrm{q}\in\Pi$ must be $\mathscr{T}_{\mathrm{q}}[\&(\mathrm{p},\{l_i:T_i^{(\mathrm{q})}\}_{i\in I})]$ for some $\mathscr{T}_{\mathrm{q}}$, $T_i^{(\mathrm{q})}$.

By induction $\mathsf{G}\restriction\mathrm{p}$ is $\hat{\mathscr{T}}[\oplus\langle\Pi,\{l_i:\hat{T}_i\}_{i\in I}\rangle]$ and $\mathsf{G}\restriction\mathrm{q}$ is $\hat{\mathscr{T}}_{\mathrm{q}}[\&(\mathrm{p},\{l_i:\hat{T}_i^{(\mathrm{q})}\}_{i\in I})]$ for all $\mathrm{q}\in\Pi$, where $\hat{}$ is the mapping induced by the reduction rule of extended session environments, if applicable. Therefore G must be of the form $\mathscr{C}[\mathrm{p}\to\Pi:\{l_i:\mathsf{G}_i\}_{i\in I}]$ for some $\mathscr{C}$, G_i such that $(\mathrm{p}\to\Pi:\{l_i:\mathsf{G}_i\}_{i\in I})\restriction\mathrm{p}=\oplus\langle\Pi,\{l_i:\hat{T}_i\}_{i\in I}\rangle$ and $(\mathrm{p}\to\Pi:\{l_i:\mathsf{G}_i\}_{i\in I})\restriction\mathrm{q}=\&(\mathrm{p},\{l_i:\hat{T}_i^{(\mathrm{q})}\}_{i\in I})$ for all $\mathrm{q}\in\Pi$. This implies $\delta(\mathsf{G},\mathrm{p},l)=\mathscr{C}[\mathrm{p}\dashrightarrow\Pi:\langle l\rangle.\mathsf{G}_j]$ and $\delta(\mathsf{G},\mathrm{p},l)\restriction\mathrm{p}=\hat{\mathscr{T}}[\oplus\langle\Pi,l\rangle;\hat{T}_j]$ and $\delta(\mathsf{G},\mathrm{p},l)\restriction\mathrm{q}=\hat{\mathscr{T}}_{\mathrm{q}}[\&(\mathrm{p},l);\hat{T}_j^{(\mathrm{q})}]$ for all $\mathrm{q}\in\Pi$.

By the proof of the Subject Reduction Theorem (Theorem 4.3) and Lemma A.6(6) the environment Δ' contains $s[\mathrm{p}]:\mathscr{T}[\oplus\langle\Pi,l\rangle;T_j]$ and $s[\mathrm{q}]:\mathscr{T}_{\mathrm{q}}[\&(\mathrm{p},\{l_i:T_i^{(\mathrm{q})}\}_{i\in I})]$ for all $\mathrm{q}\in\Pi$. This completes the proof by definition of the mapping $\hat{}$ and since:

$$\{s[\mathrm{q}]:\mathscr{T}_{\mathrm{q}}[\&(\mathrm{p},\{l_i:T_i^{(\mathrm{q})}\}_{i\in I})],s[\mathrm{p}]:\mathscr{T}[\oplus\langle\Pi,l\rangle;T_j]\}\rightsquigarrow\{s[\mathrm{q}]:\mathscr{T}_q[\&(\mathrm{p},l).T_j],s[\mathrm{p}]:\mathscr{T}[\oplus\langle\Pi,l\rangle;T_j]\}$$

□

Theorem C.4. Let P_0 be a closed pure process such that $\Gamma\vdash P_0\triangleright\emptyset$. If $P_0\longrightarrow^* P$, then $P_0 \xrightarrow{\Gamma}{}^* \mathsf{P}$ where $|\mathsf{P}|\equiv P$.

Proof. We show under the hypotheses of the theorem that:

$$\text{If } P\longrightarrow P'\text{, then } \mathsf{P}\xrightarrow{\Gamma}\mathsf{P}'\text{ where } |\mathsf{P}'|\equiv P'.$$

The proof is by a case analysis on the last applied rule using Lemmas C.3, A.5 and A.6. The interesting cases are the communication rules, for which we need to assure that γ or δ are defined. We consider as paradigmatic case that of rule [Sel]:

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

By Lemmas A.5(11) and A.6(7) the session environment for typing the left-hand-side must contain $s[\mathrm{p}]:\mathscr{T}[\oplus\langle\Pi,\{l_i:T_i\}_{i\in I}\rangle]$ for some $\mathscr{T}$, T_i, l_i such that $l=l_j$ and $j\in I$. By Lemma C.3 the global type G in the mark of s is such that $\mathsf{G}\restriction\mathrm{p}=\hat{\mathscr{T}}[\oplus\langle\Pi,\{l_i:\hat{T}_i\}_{i\in I}\rangle]$, where $\hat{}$ is the mapping induced by the reduction rule of extended session environments. Therefore G must be $\mathscr{C}[\mathrm{p}\to\Pi:\{l_i:\mathsf{G}_i\}_{i\in I}]$ for some $\mathscr{C}$, G_i. We conclude that $\delta(\mathsf{G},\mathrm{p},l)$ is defined. □

C.2. *Approximate Typed Operational Semantics*

We want to define approximate marked reductions in such a way that:

1 all computations are finite (Theorem C.11);
2 standard reductions mimic them (Lemma C.12);
3 they mimic all marked reductions with a finite number of steps (Theorem C.14).

We start by introducing an iso-recursive version of the communication type system. I.e. we do not allow fold/unfold of global and session types and we modify the rules for typing session initiation as follows:

$$\frac{\Gamma \vdash u : \mathsf{G} \quad \exists \mathsf{G}' \leq \mathsf{G} \quad \Gamma \vdash P \triangleright \Delta, y : \mathsf{G}' \upharpoonright \mathtt{p} \quad \mathtt{p} = \mathrm{mp}(\mathsf{G})}{\Gamma \vdash \overline{u}[\mathtt{p}](y).P \triangleright \Delta} \ (\text{MC}\textsc{ast}_{\leq})$$

$$\frac{\Gamma \vdash u : \mathsf{G} \quad \exists \mathsf{G}' \leq \mathsf{G} \quad \Gamma \vdash P \triangleright \Delta, y : \mathsf{G}' \upharpoonright \mathtt{p} \quad \mathtt{p} < \mathrm{mp}(\mathsf{G})}{\Gamma \vdash u[\mathtt{p}](y).P \triangleright \Delta} \ (\text{MA}\textsc{cc}_{\leq})$$

where $\leq$ is the partial order induced by the contextual closure of $\mu\mathbf{t}.G \leq G\{\mu\mathbf{t}.G/\mathbf{t}\}$ and by that of $\mu\mathbf{t}.T \leq T\{\mu\mathbf{t}.T/\mathbf{t}\}$. Notice that this means that also the global and session types which occur in exchanges can be related by $\leq$. We denote by $\vdash^{\leq}$ derivability in the obtained system. The feature of this system is to oblige the global and session types to be unfolded at least as the processes are, without losing typability with respect to the original system, as proved in the following theorem. We extend $\leq$ to environments and processes in the expected way.

Theorem C.5. If $\Gamma \vdash^{\leq} P \triangleright \Delta$, then $\Gamma \vdash P \triangleright \Delta$. If $\Gamma \vdash P \triangleright \Delta$, then there are $\Gamma' \geq \Gamma$ and $P' \geq P$ such that $\Gamma' \vdash^{\leq} P' \triangleright \Delta$.

Proof. Clearly when types are equi-recursive rules $(\text{MC}\textsc{ast}_{\leq})$, $(\text{MA}\textsc{cc}_{\leq})$ coincide with rules (MCast), (MAcc), and so a derivation in the new system is also a derivation in the original one. For the vice versa we need to:

- choose as recursive session types in channel exchanges exactly the types required for typing channel receptions, possibly unfolding the corresponding global types;
- unfold global types in Γ and in P until rules $(\text{MC}\textsc{ast}_{\leq})$, $(\text{MA}\textsc{cc}_{\leq})$ become applicable.

□

We can now define approximants in a rather standard way. We say that a global type is *finite* if it does not contain recursions. A standard environment is *finite* if it contains only finite global types.

Definition C.6.

1 The *direct approximant* of the global type G (notation $\alpha(\mathsf{G})$) is the finite global type defined by:

$$\alpha(\mathtt{p} \to \Pi : \langle U \rangle.\mathsf{G}) = \mathtt{p} \to \Pi : \langle \beta(U) \rangle.\alpha(\mathsf{G}) \quad \alpha(\mathtt{p} \dashrightarrow \Pi : \langle\text{-}\rangle.\mathsf{G}) = \mathtt{p} \dashrightarrow \Pi : \langle\text{-}\rangle.\alpha(\mathsf{G})$$
$$\alpha(\mathtt{p} \to \Pi : \{l_i : \mathsf{G}_i\}_{i \in I}) = \mathtt{p} \to \Pi : \{l_i : \alpha(\mathsf{G}_i)\}_{i \in I} \quad \alpha(\mu\mathbf{t}.G) = \alpha(\mathsf{end}) = \mathsf{end}$$

where

$$\beta(U) = \begin{cases} \alpha(U) & \text{if } U \text{ is a global or a session type,} \\ U & \text{if } U \text{ is a base type} \end{cases}$$

$$\alpha(!\langle\Pi, U\rangle.T) = !\langle\Pi, \beta(U)\rangle.\alpha(T) \quad \alpha(?(\mathsf{p}, U).T) = ?(\mathsf{p}, \beta(U)).\alpha(T)$$
$$\alpha(\oplus\langle\Pi, \{l_i : T_i\}_{i\in I}\rangle) = \oplus\langle\Pi, \{l_i : \alpha(T_i)\}_{i\in I}\rangle \quad \alpha(\&(\mathsf{p}, \{l_i : T_i\}_{i\in I})) = \&(\mathsf{p}, \{l_i : \alpha(T_i)\}_{i\in I})$$
$$\alpha(\mu\mathbf{t}.T) = \alpha(\mathsf{end}) = \mathsf{end}$$

2 A finite *global type* G is an *approximant* of a global type G' (notation $\mathsf{G} \sqsubseteq \mathsf{G}'$) if there is a global type $\mathsf{G}'' \geq \mathsf{G}'$ and $\mathsf{G}=\alpha(\mathsf{G}'')$.

3 A finite standard *environment* Γ is an *approximant* of a standard environment Γ' (notation $\Gamma \sqsubseteq \Gamma'$) if Γ is obtained from Γ' by replacing each global type in Γ' by one of its approximants.

4 A marked *process* P is an *approximant* of a marked process P' (notation $\mathsf{P} \sqsubseteq \mathsf{P}'$) if P is obtained from P' by replacing each global type occurring in P' by one of its approximants.

5 A process P is *well marked from* Γ if there is a closed user process P_0 such that $\Gamma \vdash^{\leq} P_0 \triangleright \emptyset$ and $P_0 \overset{\Gamma}{\longrightarrow}{}^{*} \mathsf{P}$.

6 A marked *process* P is *approximate* for a finite standard environment Γ if there are P', Γ' such that $\mathsf{P} \sqsubseteq \mathsf{P}'$ and $\Gamma \sqsubseteq \Gamma'$ and P' is well marked for Γ'.

The use of $\vdash^{\leq}$ instead of $\vdash$ in the definition of well-marked processes from a given environment (point 5 of Definition C.6) is needed to assure that recursive global types are enough unfolded. For example the process

$$\mathsf{def}\ X(x,y) = y!\langle x, 2\rangle.X(x,y)\ \mathsf{in}\ \mathsf{def}\ Y(x,y) = y?(x', 1).Y(x,y)\ \mathsf{in}$$
$$a[1](z).z!\langle \mathsf{true}, 2\rangle.X\langle \mathsf{true}, z\rangle \mid \overline{a}[2](z).Y\langle \mathsf{false}, z\rangle$$

can be typed from $\{a : \mu\mathbf{t}.1 \to 2 : \langle\mathsf{bool}\rangle.\mathbf{t}\}$ in the system $\vdash$. Instead in the system $\vdash^{\leq}$ it can be typed only from $\{a : 1 \to 2 : \langle\mathsf{bool}\rangle.\mu\mathbf{t}.1 \to 2 : \langle\mathsf{bool}\rangle.\mathbf{t}\}$ and further unfoldings. With the finite environment $\{a : \mathsf{end}\}$ (which is an approximant of $\{a : \mu\mathbf{t}.1 \to 2 : \langle\mathsf{bool}\rangle.\mathbf{t}\}$, but not of $\{a : 1 \to 2 : \langle\mathsf{bool}\rangle.\mu\mathbf{t}.1 \to 2 : \langle\mathsf{bool}\rangle.\mathbf{t}\}$), this process reduces to a process contaning an output action which cannot be executed since γ is undefined.

Notice that to build a process approximate for an environment (point 6 of Definition C.6) we can choose the approximants of global types in an arbitrary way. In fact as already noted at page 49 the global types in a process control the open sessions, while the global types in an environment are used in rule [InitM] for opening new sessions.

Obviously a recursive global type has infinitely many approximants. Note that approximate marked processes cannot be typed using as types for the restricted services the declared global types, since these global types are approximants of the original ones.

We want to define approximate marked reductions for approximate marked processes. Since approximate marked processes can contain recursive process calls we use the notion of communication number (Definition C.7) to bound the number of calls in computations. The communication number of participant p in an extended global type G represents the maximum number of input-output actions that p can do in a session whose mark is G, if this number is finite, and it is undefined otherwise. So the communication number is always defined for finite global types.

Definition C.7. The *communication number* of a participant p in an extended global type G (notation $\#(\mathsf{G},\mathsf{p})$) is defined by:

$$\#(\mathsf{p} \to \Pi : \langle U \rangle.\mathsf{G}',\mathsf{q}) = \begin{cases} 1 + \#(\mathsf{G}',\mathsf{q}) & \text{if } \mathsf{p} = \mathsf{q} \text{ or } \mathsf{q} \in \Pi, \\ \#(\mathsf{G}',\mathsf{q}) & \text{otherwise} \end{cases}$$

$$\#(\mathsf{p} \to \Pi : \{l_i : \mathsf{G}_i\}_{i\in I},\mathsf{q}) = \begin{cases} 1 + \max\{\#(\mathsf{G}_i,\mathsf{q})\}_{i\in I} & \text{if } \mathsf{p} = \mathsf{q} \text{ or } \mathsf{q} \in \Pi, \\ \max\{\#(\mathsf{G}_i,\mathsf{q})\}_{i\in I} & \text{otherwise} \end{cases}$$

$$\#(\mathsf{p} \dashrightarrow \Pi : \langle\text{-}\rangle.\mathsf{G}',\mathsf{q}) = \begin{cases} 1 + \#(\mathsf{G}',\mathsf{q}) & \text{if } \mathsf{q} \in \Pi, \\ \#(\mathsf{G}',\mathsf{q}) & \text{otherwise} \end{cases} \qquad \#(\mathsf{end},\mathsf{p}) = 0 \qquad \#(\mu\mathbf{t}.G',\mathsf{p}) = 0 \text{ if } G' \upharpoonright \mathsf{p} = \mathbf{t}$$

We are now able to define approximate marked reductions.

Definition C.8. A Γ-*reduction* is *approximate* if the marked processes are approximate for Γ and rule [ProcCallM] is applied only if $\#(\mathsf{G},\mathsf{p}) \neq 0$.

Note that by definition if a Γ-reduction is approximate, then Γ is finite.

The next lemma shows that in approximate reductions the communication actions correspond to defined applications of γ and δ. This lemma implies that approximate reductions and standard reductions can reduce the same communication actions that occur in approximate processes. The difference between approximate reductions and standard reductions is the applicability of rule [ProcCallM], since the associated mark can forbid it.

Lemma C.9. Let $\Gamma_0 \vdash^{\leq} P_0 \triangleright \emptyset$ and $P \sqsubseteq P_0$ and $\Gamma \sqsubseteq \Gamma_0$ and $P \xrightarrow{\Gamma}{}^{*} \mathsf{P}$. If P contains a communication action, then the application of the corresponding γ or δ is defined.

Proof. A communication action in P can be either an action in P_0 or an action in a process obtained by replacing a recursion variable with an application of rule [ProcCallM]. In the first case the result follows from the fact that the type system is iso-recursive. In the second case observe that an application of rule [ProcCallM] corresponds exactly to an unfolding of the global type in the current mark. ☐

To prove the termination of approximate reductions we start by defining the length of a finite extended global type G as the maximum number of input-output actions that can be performed in a session whose mark is G.

Definition C.10. The *length* of a finite global type G (notation $\ell(\mathsf{G})$) is defined by:

$$\ell(\mathsf{p} \to \Pi : \langle U \rangle.\mathsf{G}') = 1 + n + \ell(\mathsf{G}') \qquad \ell(\mathsf{p} \to \Pi : \{l_i : \mathsf{G}_i\}_{i\in I}) = 1 + n + \max\{\ell(\mathsf{G}_i)\}_{i\in I}$$
$$\ell(\mathsf{p} \dashrightarrow \Pi : \langle\text{-}\rangle.\mathsf{G}') = n + \ell(\mathsf{G}') \qquad \ell(\mathsf{end}) = 0$$

where n is the cardinality of Π.

We define, for each marked processes P which is approximate for Γ, a well funded *weight* $\mho(\mathsf{P},\Gamma)$ such that $\mathsf{P} \xrightarrow{\Gamma} \mathsf{P}'$ implies $\mho(\mathsf{P},\Gamma) > \mho(\mathsf{P}',\Gamma)$ in lexicographic order. This weight is a triple of natural numbers with the following meanings:

$$\omega(\bar{u}[\mathtt{p}](y).P,\Gamma,\mathscr{V},\mathbb{G}) = \langle 1,0\rangle + \omega(P,\Gamma,\mathscr{V},\mathbb{G})$$

$$\omega((\nu s)P,\Gamma,\mathscr{V},\mathbb{G}) = \omega(y!\langle\Pi,e\rangle.P,\Gamma,\mathscr{V},\mathbb{G}) = \omega(y?(\mathtt{p},x).P,\Gamma,\mathscr{V},\mathbb{G}) = \omega(P,\Gamma,\mathscr{V},\mathbb{G})$$

$$\omega(y!\langle\langle\mathtt{p},c\rangle\rangle.P,\Gamma,\mathscr{V},\mathbb{G}) = \omega(y?((\mathtt{q},y')).P,\Gamma,\mathscr{V},\mathbb{G}) = \omega(y\oplus\langle\Pi,l\rangle.P,\Gamma,\mathscr{V},\mathbb{G}) = \omega(P,\Gamma,\mathscr{V},\mathbb{G})$$

$$\omega(y\&(\mathtt{p},\{l_i:P_i\}_{i\in I}),\Gamma,\mathscr{V},\mathbb{G}) = \max\{\omega(P_i,\Gamma,\mathscr{V},\mathbb{G})\}_{i\in I}$$

$$\omega(s[\mathtt{p}]!\langle\Pi,e\rangle.P,\Gamma,\mathscr{V},\mathbb{G}) = \omega(s[\mathtt{p}]?(\mathtt{q},x).P,\Gamma,\mathscr{V},\mathbb{G}) = \begin{cases} \omega(P,\Gamma,\mathscr{V},\mathbb{G}'\cup\{s:\gamma(\mathsf{G},\mathtt{p})\}) & \text{if } \mathbb{G}=\mathbb{G}'\cup\{s:\mathsf{G}\} \\ & \text{and } \gamma(\mathsf{G},\mathtt{p}) \text{ is defined,} \\ \langle 0,0\rangle & \text{otherwise.} \end{cases}$$

$$\omega(s[\mathtt{p}]!\langle\langle\mathtt{q},c\rangle\rangle.P,\Gamma,\mathscr{V},\mathbb{G}) = \omega(s[\mathtt{p}]?((y,\mathtt{q})).P,\Gamma,\mathscr{V},\mathbb{G}) = \begin{cases} \omega(P,\Gamma,\mathscr{V},\mathbb{G}'\cup\{s:\gamma(\mathsf{G},\mathtt{p})\}) & \text{if } \mathbb{G}=\mathbb{G}'\cup\{s:\mathsf{G}\} \\ & \text{and } \gamma(\mathsf{G},\mathtt{p}) \text{ is defined,} \\ \langle 0,0\rangle & \text{otherwise.} \end{cases}$$

$$\omega(s[\mathtt{p}]\oplus\langle\Pi,l\rangle.P,\Gamma,\mathscr{V},\mathbb{G}) = \begin{cases} \omega(P,\Gamma,\mathscr{V},\mathbb{G}'\cup\{s:\delta(\mathsf{G},\mathtt{p},l)\}) & \text{if } \mathbb{G}=\mathbb{G}'\cup\{s:\mathsf{G}\} \\ & \text{and } \delta(\mathsf{G},\mathtt{p},l) \text{ is defined,} \\ \langle 0,0\rangle & \text{otherwise.} \end{cases}$$

$$\omega(s[\mathtt{p}]\&(\mathtt{q},\{l_i:P_i\}_{i\in I}),\Gamma,\mathscr{V},\mathbb{G}) = \begin{cases} \max\{\omega(P_i,\Gamma,\mathscr{V},\mathbb{G}'\cup\{s:\gamma(\mathsf{G},\mathtt{p})\})\}_{i\in I} & \text{if } \mathbb{G}=\mathbb{G}'\cup\{s:\mathsf{G}\} \\ & \text{and } \gamma(\mathsf{G},\mathtt{p}) \text{ is defined,} \\ \langle 0,0\rangle & \text{otherwise.} \end{cases}$$

$$\omega(\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q,\Gamma,\mathscr{V},\mathbb{G}) = \langle 0,1\rangle + \max\{\omega(P,\Gamma,\mathscr{V},\mathbb{G}),\omega(Q,\Gamma,\mathscr{V},\mathbb{G})\}$$

$$\omega(P\mid Q,\Gamma,\mathscr{V},\mathbb{G}) = \omega(P,\Gamma,\mathscr{V},\mathbb{G}) + \omega(Q,\Gamma,\mathscr{V},\mathbb{G}) \quad \omega(\mathbf{0},\Gamma,\mathscr{V},\mathbb{G}) = \omega(s:h,\Gamma,\mathscr{V},\mathbb{G}) = \langle 0,0\rangle$$

$$\omega((\nu a:\mathsf{G})P,\Gamma,\mathscr{V},\mathbb{G}) = \omega(P,\Gamma\cup\{a:\mathsf{G}\},\mathscr{V},\mathbb{G})$$

$$\omega(\mathsf{def}\ X(x,y)=P\ \mathsf{in}\ Q,\Gamma,\mathscr{V},\mathbb{G}) = \omega(Q,\Gamma,\mathscr{V}\cup\{X(x,y)\mapsto(P,k_0)\},\mathbb{G}) \text{ where } k_0=\max\{\ell(\mathsf{G})\mid u:\mathsf{G}\in\Gamma\}$$

$$\omega(X\langle e,y\rangle,\Gamma,\mathscr{V},\mathbb{G}) = \begin{cases} \langle 0,0\rangle & \text{if } \mathscr{V}=\mathscr{V}'\cup\{X(x,y')\mapsto(P,0)\} \\ \langle 0,1\rangle+\omega(P,\Gamma,\mathscr{V}'\cup\{X(x,y')\mapsto(P,k)\},\mathbb{G}) & \text{if } \mathscr{V}=\mathscr{V}'\cup\{X(x,y')\mapsto(P,k+1)\}. \end{cases}$$

$$\omega(X\langle e,s[\mathtt{p}]\rangle,\Gamma,\mathscr{V},\mathbb{G}) = \begin{cases} \langle 0,0\rangle & \text{if } \mathscr{V}=\mathscr{V}'\cup\{X(x,y)\mapsto(P,0)\} \\ \langle 0,1\rangle+\omega(P\{v/x\}\{s[\mathtt{p}]/y\},\Gamma,\mathscr{V}'\cup\{X(x,y)\mapsto(P,k')\},\mathbb{G}) & \text{otherwise, where } e\downarrow v \text{ and} \\ & s:\mathsf{G}\in\mathbb{G} \text{ and } k'=\#(\mathsf{G},\mathtt{p}). \end{cases}$$

Table 15. The mapping ω.

1 the first number bounds the number of request or accept that could start (possibly with the help of catalysers) by Γ-reducing the process;

2 the second number is the sum of the lengths of the global types which occur as marks in the process, i.e. of the global types of the sessions already started;

3 the third number bounds the number of possible applications of rules [If-M] and [ProcCallM] by Γ-reducing the process.

Let $\mathsf{P}\equiv(\nu\overrightarrow{s})(P\mid\Pi_{i\in I}(s_i:\mathsf{G}_i))$, where P does not contain session marks and session restrictions. The weight of P and Γ is the triple $\langle n,\Sigma_{i\in I}\ell(\mathsf{G}_i),m\rangle$ where $\langle n,m\rangle=\omega(P,\Gamma,\emptyset,\mathbb{G})$ and the function ω is defined in Table 15 and $\mathbb{G}=\{s_i:\mathsf{G}_i\mid i\in I\}$. In this table the sum and the maximum of natural pairs are component-wise and $\mathscr{V}$ is a mapping from term variables to pairs of processes and naturals.

Using this weight we can show the termination of all approximate marked reductions.

Theorem C.11. Every approximate marked reduction terminates.

Proof. We can show that each reduction rule decreases the weight of processes and standard environments. It is immediate to see that the weight decreases at every communication action (in the second component) and at every conditional choice (in the last component). The opening of a service increases the second component but decreases the first one thus reducing the weight. As for the recursive definitions note that the weight decreases by 1 in the third component at every process call. Moreover the condition that recursive calls are guarded (see §4.1) assures that in the recursive definition of ω the next call will be executed with a smaller natural associated to the term variable.

The more interesting case is rule [ProcCallM]. The crucial observation is that

$$\omega(P\{v/x\}\{s[\mathrm{p}]/y\},\Gamma,\{X(x,y)\mapsto(P,k_1)\},\mathbb{G})=\omega(P\{v/x\}\{s[\mathrm{p}]/y\},\Gamma,\{X(x,y)\mapsto(P,k_2)\},\mathbb{G})$$

for all positive k_1,k_2. This follows from the definition of $\omega(X\langle e,s[\mathrm{p}]\rangle,\Gamma,\mathscr{V},\mathbb{G})$, which in this case replaces $\#(\mathsf{G},\mathrm{p})$ to the (positive) values k_1,k_2, where $s:\mathsf{G}\in\mathbb{G}$.
Let $\mathsf{P}_1=\mathsf{def}\ X(x,y)=P\ \mathsf{in}\ (X\langle e,s[\mathrm{p}]\rangle\mid s:\mathsf{G}\mid\mathsf{P})$ and $\mathsf{P}_2=\mathsf{def}\ X(x,y)=P\ \mathsf{in}\ (P\{v/x\}\{s[\mathrm{p}]/y\}\mid s:\mathsf{G}\mid\mathsf{P})$ and $e\downarrow v$ and $k_0=\max\{\ell(\mathsf{G}')\mid u:\mathsf{G}'\in\Gamma\}$ and $\mathsf{P}\equiv R\mid\Pi_{i\in I}(s_i:\mathsf{G}_i)$ and $\mathbb{G}=\{s_i:\mathsf{G}_i\mid i\in I\}$. We have:

$$\begin{array}{rcl}\omega(P_1,\Gamma,\emptyset,\mathbb{G}\cup\{s:\mathsf{G}\}) &=& \omega(X\langle e,s[\mathrm{p}]\rangle\mid R,\Gamma,\{X(x,y)\mapsto(P,k_0)\},\mathbb{G}\cup\{s:\mathsf{G}\})\\ &=& \omega(X\langle e,s[\mathrm{p}]\rangle,\Gamma,\{X(x,y)\mapsto(P,k_0)\},\mathbb{G}\cup\{s:\mathsf{G}\})+\\ && \omega(R,\Gamma,\{X(x,y)\mapsto(P,k_0)\},\mathbb{G}\cup\{s:\mathsf{G}\})\\ &=& \langle 0,1\rangle+\omega(P\{v/x\}\{s[\mathrm{p}]/y\},\Gamma,\{X(x,y)\mapsto(P,\#(\mathsf{G},\mathrm{p}))\},\mathbb{G}\cup\{s:\mathsf{G}\})+\\ && \omega(R,\Gamma,\{X(x,y)\mapsto(P,k_0)\},\mathbb{G}\cup\{s:\mathsf{G}\})\end{array}$$

$$\begin{array}{rcl}\omega(P_2,\Gamma,\emptyset,\mathbb{G}\cup\{s:\mathsf{G}\}) &=& \omega(P\{v/x\}\{s[\mathrm{p}]/y\}\mid R,\Gamma,\{X(x,y)\mapsto(P,k_0)\},\mathbb{G}\cup\{s:\mathsf{G}\})\\ &=& \omega(P\{v/x\}\{s[\mathrm{p}]/y\},\Gamma,\{X(x,y)\mapsto(P,k_0)\},\mathbb{G}\cup\{s:\mathsf{G}\})+\\ && \omega(R,\Gamma,\{X(x,y)\mapsto(P,k_0)\},\mathbb{G}\cup\{s:\mathsf{G}\})\end{array}$$

where $P_1=\mathsf{def}\ X(x,y)=P\ \mathsf{in}\ (X\langle e,s[\mathrm{p}]\rangle\mid R)$ and $P_2=\mathsf{def}\ X(x,y)=P\ \mathsf{in}\ (P\{v/x\}\{s[\mathrm{p}]/y\}\mid R)$. Since the application of rule [ProcCallM] requires $\#(\mathsf{G},\mathrm{p})\neq 0$ we conclude

$$\omega(P_1,\Gamma,\emptyset,\{s:\mathsf{G}\})>\omega(P_2,\Gamma,\emptyset,\{s:\mathsf{G}\}).$$

□

It is easy to check that approximate marked reductions become standard reductions by erasing the session marks.

Lemma C.12. If $\mathsf{P}\ \xrightarrow{\Gamma}^{*}\ \mathsf{P}'$ is an approximate marked reduction, then $|\mathsf{P}|\longrightarrow^{*}|\mathsf{P}'|$.

For the vice versa it is useful to show how communication numbers decrease in approximate reductions.

Lemma C.13. If $\mathsf{P}\mid s:\mathsf{G}\ \xrightarrow{\Gamma}^{n}\ \mathsf{P}'\mid s:\mathsf{G}'$ and G, G' are finite, then $\#(\mathsf{G},\mathrm{p})\leq\#(\mathsf{G}',\mathrm{p})+n$ for all p.

Proof. The communication number of a participant decreases only if the applied reduction rule is a communication rule in which the participant sends or receives a message. □

We prove that for each finite marked reduction there is an approximate reduction producing the same final process (modulo session marks).

Theorem C.14. If P_0 is well marked from Γ and $\mathsf{P}_0 \xrightarrow{\Gamma}{}^{*} \mathsf{P}$, then there are $\mathsf{P}'_0 \sqsubseteq \mathsf{P}_0$ and $\Gamma' \sqsubseteq \Gamma$ such that $\mathsf{P}'_0 \xrightarrow{\Gamma'}{}^{*} \mathsf{P}'$ and $\mathsf{P}' \sqsubseteq \mathsf{P}$.

Proof. The key observation here is that by definition of well-marking the global types in P_0 and Γ have been unfolded at least as the corresponding processes in P_0. This is assured by typability in $\vdash^{\leq}$ of the original process. If $\mathsf{P}_0 \xrightarrow{\Gamma}{}^{n} \mathsf{P}$ we choose all approximate global types in P'_0 and Γ' as direct approximants of the given global types, where all recursions have been unfolded at least n times. Then by Lemmas C.9 and C.13 all Γ-reduction steps starting from P_0 can also be performed as Γ'-reduction steps from P'_0, i.e. $\mathsf{P}'_0 \xrightarrow{\Gamma'}{}^{*} \mathsf{P}'$ where $\mathsf{P}' \sqsubseteq \mathsf{P}$. $\square$

C.3. *Pseudo-progress*

Building on approximate reductions we introduce now pseudo-progress. The main difference between progress (Definition 5.4) and pseudo-progress is that instead of asking that a specific input or output request is satisfied we ask that an approximate process can be reduced until all marks are end.

Definition C.15. A closed user process P has the *pseudo-progress* if for all $\Gamma, \Gamma', P', \mathsf{P}$ with $\Gamma \vdash^{\leq} P \triangleright \emptyset$ and $\Gamma' \sqsubseteq \Gamma$ and $P' \sqsubseteq P$ and $P' \xrightarrow{\Gamma'}{}^{*} \mathsf{P}$, there is a catalyser Q which is approximate for an environment $\Gamma'' \supseteq \Gamma'$ such that $\mathsf{P} \mid Q \xrightarrow{\Gamma''}{}^{*} \mathsf{P}'$ and all marks in P' are end.

Notice that closed user processes are marked processes (as observed at page 48) and therefore $\sqsubseteq$ between them is defined.

To get that initial processes have progress we will show that:

1 if P is initial, then $P \mid Q$ has pseudo-progress for all catalysers Q such that $P \mid Q$ is typable in the communication system (Theorem C.22);

2 if $P \mid Q$ has pseudo-progress for all catalysers Q such that $P \mid Q$ is typable in the communication system, then P has progress (Theorem C.23).

We start with some technical definitions and lemmas which are handy for the proof of point 1. The first two lemmas consider the relation between occurrences of communication actions, and to formulate them it is useful to define when an action is at the top of another one and when a channel with role precedes another one. By *action* we mean a communication action, an accept/request or a conditional.

Definition C.16.

1 The occurrence ϕ of an action is *at the top* of the occurrence ψ of an action or process call in the process $\mathsf{P} = \mathscr{E}[P]$ if one of the following conditions holds:

(a) $P = \phi.P'$ and ψ occurs in P';

(b) $P = \phi = \text{if } e \text{ then } P_1 \text{ else } P_2$ and ψ occurs in P_1 or P_2;

(c) $P = \phi = c\&(\mathtt{p}, \{l_i : P_i\}_{i \in I})$ and ψ occurs in P_i for some $i \in I$.

2 The channel $s[\mathtt{p}]$ *precedes* the channel $s'[\mathtt{q}]$ in the process P if P contains:

(a) a communication action on channel $s[\mathtt{p}]$ which is at the top of a communication action on channel $s'[\mathtt{q}]$;

(b) a delegation $s[\mathtt{p}]!\langle\langle \mathtt{p}', s'[\mathtt{q}]\rangle\rangle$ for some $\mathtt{p}'$;

(c) a message $(\mathtt{p}, \mathtt{p}', s'[\mathtt{q}])$ for some $\mathtt{p}'$ in the queue s.

3 The channel $s[\mathtt{p}]$ *strongly precedes* the channel $s'[\mathtt{q}]$ in the process P if $s[\mathtt{p}]$ precedes $s'[\mathtt{q}]$ in P and in case (a) the communication action on $s[\mathtt{p}]$ is an input action.

It is easy to verify by structural induction on processes that strong precedence between channels is recorded in csds.

Lemma C.17. If $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash P \blacktriangleright \mathscr{D}$ and $s[\mathtt{p}]$ strongly precedes $s'[\mathtt{q}]$ in P and $s \neq s'$, then $s \prec s' \in \mathscr{D}$.

A key property assured only by typability in both type systems is that two channels of the same session with different participants can never precede each other.

Lemma C.18. Let P be initial and $P \longrightarrow^* P'$.

1 If $s[\mathtt{p}]$ precedes $s'[\mathtt{q}]$ in P', then either $s \neq s'$ or $\mathtt{p} = \mathtt{q}$;
2 If $P' \equiv P'' \mid s : h' \cdot (\mathtt{p}, \mathtt{q}, s'[\mathtt{p}']) \cdot h$, then $s' \neq s$.

Proof. We show both points simultaneously by induction on $\longrightarrow^*$. In an initial P there are no channels with roles. As for the induction step we discuss the more interesting cases.
- Rule [Init] creates a new channel with a unique distinguished role for each parallel process. Both points (1) and (2) follow trivially by the induction hypothesis.
- Let us apply rule [SRcv] to $s[\mathtt{p}]?(\mathtt{q},x).R \mid s : (\mathtt{q}, \mathtt{p}, s'[\mathtt{p}']) \cdot h$. By induction hypothesis we must have $s \neq s'$. By Theorem 6.2 we can derive a csd for $s[\mathtt{p}]?(\mathtt{q},x).R$ using the interaction typing rule $\{\textsc{SRcv}\}$. Therefore $s[\mathtt{p}]$ and $s'[\mathtt{p}']$ are the only channels with role in $R\{s'[\mathtt{p}']/y\}$ and point (1) follows. Point (2) is immediate by induction hypothesis.
- When rule [Deleg] is used note that the session delegation command must have been typed in the communication type system by rule $(\textsc{Deleg})$. For this reason we get $s[\mathtt{p}] \neq s'[\mathtt{p}']$. Since $s[\mathtt{p}]$ precedes $s'[\mathtt{p}']$ in the session delegation command, by induction hypothesis $s = s'$ implies $\mathtt{p} = \mathtt{p}'$. We then conclude $s \neq s'$ proving point (2). Point (1) is immediate by induction hypothesis. □

The next lemma assures that no free channel will occur in a process after a request/accept on a service name which can be bound.

Lemma C.19. If $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash \tilde{a}[\mathtt{p}](y).P \blacktriangleright \mathscr{D}$ and $a \in \mathscr{B}$, then y is the only free channel which occurs in P.

Proof. The last applied rule must be $\{\textsc{InitB}\}$, then the condition $\mathsf{fc}(P) \subseteq \{y\}$ assures the statement. □

The last two lemmas consider how the global types restrict csds and processes in well-marked processes.

Lemma C.20. If $\mathsf{P} \mid s : \mathsf{end}$ is a well marked process and $\Theta;\mathscr{R};\mathscr{N};\mathscr{B} \vdash |\mathsf{P}| \blacktriangleright \mathscr{D}$, then we have $s \prec s' \in \mathscr{D}$ for no s'.

$$\sigma(\tilde{u}[\mathsf{p}](y).P) = \sigma(c?(\mathsf{p},x).P)=\sigma(c!\langle\langle \mathsf{p},c'\rangle\rangle.P)=\sigma(c?((\mathsf{q},y)).P)=\sigma(y\oplus\langle\Pi,l\rangle.P)=\sigma(P)$$

$$\sigma(c!\langle\Pi,e\rangle.P)=\begin{cases}\{a\}\cup\sigma(P) & \text{if } e=a,\\ \sigma(P) & \text{otherwise.}\end{cases}\qquad \sigma(c\&(\mathsf{p},\{l_i:P_i\}_{i\in I}))=\bigcup_{i\in I}\sigma(P_i)$$

$$\sigma(\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q)=\sigma(P\mid Q)=\sigma(\mathsf{def}\ X(x,y)=P\ \mathsf{in}\ Q)=\sigma(P)\cup\sigma(Q)$$

$$\sigma(\mathbf{0})=\sigma(s:h)=\sigma(X\langle e,y\rangle)=\emptyset\qquad \sigma((\nu a:\mathsf{G})P)=\sigma((\nu s)P)=\sigma(P)$$

Table 16. The mapping σ.

Proof. It is easy to verify by induction on reduction of well-marked processes that, if the mark of s is end, then a channel $s[\mathsf{p}]$ for some p can occur only in process calls, and by definition a csd associated to a process variable can contain at most one channel. □

Lemma C.21. Let $\mathsf{P} = \mathscr{E}[s:\mathsf{G}]$ be a well marked process, where $\mathsf{G}\neq\mathsf{end}$ is finite. Then P contains at least one of the following:

- an output action on a channel $s[\mathsf{p}]$;
- an input action on a channel $s[\mathsf{p}]$ and a corresponding message on the top of the s queue;
- a process call on a channel $s[\mathsf{p}]$ and $\#(\mathsf{G},\mathsf{p})\neq 0$;

for some p.

Proof. By induction on marked reductions it is easy to show that:

- If $\mathsf{G}=\mathsf{p}\to\Pi:\langle S\rangle.\mathsf{G}'$, then the process P contains either an action of the shape $s[\mathsf{p}]!\langle\Pi,e\rangle.P$, where e is an expression of type S, or a process call on the channel $s[\mathsf{p}]$;
- If $\mathsf{G}=\mathsf{p}\to\Pi:\langle\mathsf{T}\rangle.\mathsf{G}'$, then the process P contains either an action of the shape $s[\mathsf{p}]!\langle\langle\mathsf{p},c\rangle\rangle.P$, where c is a channel of type T, or a process call on the channel $s[\mathsf{p}]$;
- If $\mathsf{G}=\mathsf{p}\to\Pi:\{l_i:\mathsf{G}_i\}_{i\in I}$, then the process P contains either an action of the shape $s[\mathsf{p}]\oplus\langle\Pi,l_j\rangle.P$ for some $j\in I$, or a process call on the channel $s[\mathsf{p}]$;
- If $\mathsf{G}=\mathsf{p}\dashrightarrow\Pi:\langle S\rangle.\mathsf{G}'$, then, for all $\mathsf{q}\in\Pi$, the process P contains either an action of the shape $s[\mathsf{q}]?(\mathsf{p},x).P$ and the queue $s:h$ is such that $h\equiv(\mathsf{p},\Pi,v)\cdot h'$ for some value v of type S and some queue h', or a process call on the channel $s[\mathsf{q}]$;
- If $\mathsf{G}=\mathsf{p}\dashrightarrow\mathsf{q}:\langle\mathsf{T}\rangle.\mathsf{G}'$, then the process P contains either an action of the shape $s[\mathsf{q}]?((\mathsf{p},y)).P$ and the queue $s:h$ is such that $h\equiv(\mathsf{p},\mathsf{q},s'[\mathsf{p}'])\cdot h'$ for some channel $s'[\mathsf{p}']$ of type T and some queue h', or a process call on the channel $s[\mathsf{q}]$;
- If $\mathsf{G}=\mathsf{p}\dashrightarrow\Pi:\langle l_j\rangle.\mathsf{G}'$, then, for all $\mathsf{q}\in\Pi$, the process P contains either an action of the shape $s[\mathsf{q}]\&(\mathsf{p},\{l_i:P_i\}_{i\in I})$ with $j\in I$ and the queue $s:h$ is such that $h\equiv(\mathsf{p},\Pi,l_j)\cdot h'$ for some queue h', or a process call on the channel $s[\mathsf{q}]$.

□

We conclude by proving points 1 and 2 as discussed at page 58.

Theorem C.22. If P is initial, then $P\mid Q$ has the pseudo-progress for all catalysers Q such that $P\mid Q$ is well typed in the communication type system.

Proof. Let Γ,Γ',P' and P be such that $\Gamma\vdash^{\leq}P\mid Q\triangleright\emptyset$ and $P'\sqsubseteq P\mid Q$ and $\Gamma'\sqsubseteq\Gamma$ and $P'\xrightarrow{\Gamma'}{}^{*}\mathsf{P}$. If all marks in P are end there is nothing to prove.

Otherwise we build a catalyser Q' and a finite environment $\Gamma''\supseteq\Gamma'$ such that Q' is approximate

$$\rho(\tilde{a}[\mathsf{p}](y).P,\Gamma,\mathscr{V},\mathscr{A}) = \{\{a\}\} \uplus \rho(P,\Gamma,\mathscr{V},\mathscr{A}) \qquad \rho(\tilde{x}[\mathsf{p}](y).P,\Gamma,\mathscr{V},\mathscr{A}) = \mathscr{A} \uplus \rho(P,\Gamma,\mathscr{V},\mathscr{A})$$

$$\rho(c!\langle\Pi,e\rangle.P,\Gamma,\mathscr{V},\mathscr{A}) = \rho(c?(\mathsf{p},x).P,\Gamma,\mathscr{V},\mathscr{A}) = \rho(c!\langle\langle\mathsf{p},c'\rangle\rangle.P,\Gamma,\mathscr{V},\mathscr{A}) = \rho(c?((\mathsf{q},y')).P,\Gamma,\mathscr{V},\mathscr{A}) = \rho(P,\Gamma,\mathscr{V},\mathscr{A})$$

$$\rho(c\oplus\langle\Pi,l\rangle.P,\Gamma,\mathscr{V},\mathscr{A}) = \rho((\nu a:\mathsf{G})P,\Gamma,\mathscr{V},\mathscr{A}) = \rho((\nu s)P,\Gamma,\mathscr{V},\mathscr{A}) = \rho(P,\Gamma,\mathscr{V},\mathscr{A})$$

$$\rho(c\&(\mathsf{p},\{l_i:P_i\}_{i\in I}),\Gamma,\mathscr{V},\mathscr{A}) = \biguplus_{i\in I}\rho(P_i,\Gamma,\mathscr{V},\mathscr{A})$$

$$\rho(\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q,\Gamma,\mathscr{V},\mathscr{A}) = \rho(P \mid Q,\Gamma,\mathscr{V},\mathscr{A}) = \rho(P,\Gamma,\mathscr{V},\mathscr{A}) \uplus \rho(Q,\Gamma,\mathscr{V},\mathscr{A})$$

$$\rho(\mathbf{0},\Gamma,\mathscr{V},\mathscr{A}) = \rho(s:h,\Gamma,\mathscr{V},\mathscr{A}) = \emptyset$$

$$\rho(\mathsf{def}\ X(x,y) = P\ \mathsf{in}\ Q,\Gamma,\mathscr{V},\mathscr{A}) = \rho(Q,\Gamma,\mathscr{V}\cup\{X(x,y)\mapsto(P,k_0)\},\mathscr{A})$$
$$\text{where } k_0 = \max\{\ell(\mathsf{G}) \mid u:\mathsf{G}\in\Gamma\}$$

$$\rho(X\langle e,c\rangle,\Gamma,\mathscr{V},\mathscr{A}) = \begin{cases} \rho(P,\Gamma,\mathscr{V}'\cup\{X(x,y)\mapsto(P,k)\},\mathscr{A}) & \text{if } \mathscr{V} = \mathscr{V}'\cup\{X(x,y)\mapsto(P,k+1)\}, \\ \emptyset & \text{if } \mathscr{V} = \mathscr{V}'\cup\{X(x,y)\mapsto(P,0)\}. \end{cases}$$

Table 17. The mapping ρ.

for Γ'' and it has all needed service participants. We show that if $\mathsf{P} \mid Q' \xrightarrow{\Gamma''}{}^{*} \mathsf{P}'$ and not all marks in P' are end, then P' can be Γ''-reduced. This is enough thanks to the termination of approximate reductions (Theorem C.11).

The definition of Q' uses the mappings σ and ρ defined in Tables 16 and 17. The mapping σ collects all service names which are sent. The mapping ρ gives an upper bound to the multiset[§]of service names which could ask for missing participants in the reducts of $\mathsf{P} \mid Q'$. It uses σ to deal with requests/accepts on variables. If $\rho(|\mathsf{P}|,\Gamma',\emptyset,\sigma(|\mathsf{P}|)) = \{\{a_i \mid i\in I\}\}$ and $a_i : \mathsf{G}_i \in \Gamma'$ for $i \in I$, then $Q' = \Pi_{i\in I}\mathscr{Q}(a_i,\mathsf{G}_i)$, where $\mathscr{Q}(a,\mathsf{G})$ is given by

$$\mathscr{Q}(a,\mathsf{G}) = a[1](y).\mathscr{P}(\mathsf{G}\restriction 1,y,\emptyset) \mid \ldots \mid a[n-1](y).\mathscr{P}(\mathsf{G}\restriction(n-1),y,\emptyset) \mid \overline{a}[n](y).\mathscr{P}(\mathsf{G}\restriction n,y,\emptyset)$$

with $n = \mathrm{mp}(\mathsf{G})$. It is easy to verify that: $\{a:\mathsf{G}\}\cup\bigcup_{1\le j\le n}\P(\mathsf{G}\restriction j) \vdash \mathscr{Q}(a,\mathsf{G}) \rhd \emptyset$.

To build Γ'' we use the length of a closed session type T (notation $\ell(\mathsf{T})$) defined by:

$$\begin{array}{ll} \ell(!\langle\Pi,U\rangle.\mathsf{T}') = 1+n+\ell(\mathsf{T}') & \ell(?(\mathsf{p},U).\mathsf{T}') = 1+\ell(\mathsf{T}') \\ \ell(\oplus\langle\Pi,\{l_i:\mathsf{T}_i\}_{i\in I}\rangle) = 1+n+\max\{\ell(\mathsf{T}_i)\}_{i\in I} & \ell(\&(\mathsf{p},\{l_i:\mathsf{T}_i\}_{i\in I})) = 1+\max\{\ell(\mathsf{T}_i)\}_{i\in I} \\ \ell(\mu\mathbf{t}.T) = 0 & \ell(\mathsf{end}) = 0 \end{array}$$

where n is the cardinality of Π.

Let $\Gamma_0 = \bigcup_{i\in I}\bigcup_{1\le j\le n_i}\P(\mathsf{G}_i\restriction j)$ where $n_i = \mathrm{mp}(\mathsf{G}_i)$ for $i \in I$. By construction $\Gamma,\Gamma_0 \vdash Q' \rhd \emptyset$. We define $\Gamma_0' \sqsubseteq \Gamma_0$ by unfolding m times each global type in Γ_0, where m is the maximum of the lengths of the session types which occur in G_i for $i \in I$. Lastly we choose $\Gamma'' = \Gamma',\Gamma_0'$. Since we typed $P \mid Q$ from Γ in the iso-recursive system $\vdash^{\le}$ we are sure that the reduction of $\mathsf{P} \mid Q'$ can execute all process calls in Q' for dealing with delegations whenever the corresponding processes in P require these calls.

We say that a process R is *ready* inside a process R if $\mathsf{R} = \mathscr{E}[R]$ for some evaluation context $\mathscr{E}$. Note that:

- if a ready process in P' is an output, then P' can be reduced by Lemma C.9;

- if a ready process in P' is a conditional, then P' can be reduced, since P' is closed (being $P \mid Q \mid Q'$ closed) and any closed boolean value is either true or false;

- no ready process in P' is an request/accept on a variable since P' is closed;

[§] We use $\{\{\ \}\}$ to denote multisets and $\uplus$ to denote multiset union.

- if a ready process in P' is a request/accept on a free service name, then we can apply rule [InitM] since Q' by construction allows to complete any service call by providing the missing participants.

Otherwise notice that if P is initial, then $P \mid Q$ is initial for all catalysers Q. Therefore by Lemma C.12 and the Subject Reduction property of the interaction system (Theorem 6.2) the process $|\mathsf{P}'|$ can be typed in the interaction system. Let $|\mathsf{P}'| = (\nu \overrightarrow{s})P''$, where $\overrightarrow{s}$ is the set of all session names which occur in P'. This implies $\emptyset;\mathscr{R};\mathscr{N};\emptyset \vdash P'' \blacktriangleright \mathscr{D}$ for some $\mathscr{R}$, $\mathscr{N}$, $\mathscr{D}$.

Let s be a session name with a mark G different from end and such that $s' \prec s \in \mathscr{D}$ for no s'. The existence of such an s is assured by the fact that $\mathscr{D}$ is loop free and by Lemma C.20.

By Lemma C.21 P' must contain:

1 an output action on a channel $s[\mathrm{p}]$ with γ/δ defined by Lemma C.9 or
2 input action on a channel $s[\mathrm{p}]$ and the corresponding message on the top of the queue s with γ defined by Lemma C.9 or
3 a process call on a channel $s[\mathrm{p}]$ with $\#(\mathsf{G},\mathrm{p}) \neq 0$

for some p. We claim that we can reduce this communication action or this process call since no other action can be at its top. In fact this action at top cannot be:

1 a communication on a channel $s[\mathrm{q}]$ for some $\mathrm{q} \neq \mathrm{p}$ by Lemma C.18;
2 an output, request/accept on a free service name or conditional action since we reduced all them already;
3 a request/accept on a bound service name by Lemma C.19;
4 an input action on a channel $s'[\mathrm{q}]$, since otherwise $s' \prec s \in \mathscr{D}$ by Lemma C.17.

We conclude that P' must contain some ready process which can be reduced. □

Theorem C.23. If $P \mid Q$ has the pseudo-progress for all catalysers Q, then P has progress.

Proof. Let $\Gamma \vdash^{\leq} P \mid Q \triangleright \emptyset$ and $P \mid Q \longrightarrow^* \mathscr{E}[R]$, where R is an input process or a message queue. This implies by Theorems C.5 and C.4 that $P \mid Q \xrightarrow{\Gamma}{}^* \mathsf{P}$ and $|\mathsf{P}|=\mathscr{E}[R]$. By Theorem C.14 there is Γ' such that $\Gamma' \sqsubseteq \Gamma$ and $P \mid Q \xrightarrow{\Gamma'}{}^* \mathsf{P}'$ with $\mathsf{P}' \sqsubseteq \mathsf{P}$. By definition of pseudo-progress there is a catalyser Q' which is approximate for some $\Gamma'' \sqsupseteq \Gamma$ such that $\mathsf{P}' \mid Q' \xrightarrow{\Gamma''}{}^* \mathsf{R}$ and R is a marked process in which all marks are end. In particular the mark of s has to be end, and this implies that R has been reduced thanks to a dual message queue or input process. □