Global Progress in Dynamically Merged Multiparty Sessions

Lorenzo Bettini[1], Mario Coppo[1], Loris D'Antoni[1], Marco De Luca[1], Mariangiola Dezani-Ciancaglini[1], and Nobuko Yoshida[2]

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

Abstract. A multiparty session forms a unit of structured interactions among many participants which follow a prescribed scenario specified as a global type signature. This paper develops, besides a more traditional *communication* type system, a novel static *interaction* type system for global progress in dynamically merged and interfered multiparty sessions.

1 Introduction

Widespread use of message-based communication for developing network applications to combine numerous distributed services has provoked urgent interests in structuring series of interactions to specify and program communication-safe software. The actual development of such applications still leaves to the programmer much of the responsibility in guaranteeing that communication will evolve as agreed by all the involved distributed peers. *Multiparty session type discipline* proposed in [12] offers a type-theoretic framework to validate a messages-exchange among concurrently running multiple peers in the distributed environment, generalising the existing two-party session types [10, 11]; interaction sequences are abstracted as a global type signature, which precisely declares how multiple participants communicate and synchronise with each other. Recently the two standardisation bodies for web-based business and finance protocols [19, 18] have investigated a design and implementation framework for standardising message exchange rules and validating business logic based on a notion of multiparty sessions, where a global type plays as a "shared agreement" between a team of programmers who are developing (possibly) a large size of distributed protocol or software by collaborations.

The initial multiparty session type discipline aims to retain the powerful dynamic features from the original binary sessions [11], incorporating features such as recursion and choice of interactions. Among features, *session delegation* is a key operation which permits to rely on other parties for completing specific tasks transparently in a type safe manner. A typical scenario is a web server delegating remaining interactions with a client to an application server to complete a transaction. The customer and the application server are initially unknown to each other but later communicate directly (transparently to the customer), through dynamic mobility of the session. When this mechanism is extended to multiparty interactions engaged in two or more specifications simultaneously, further complex interactions can be modelled: each multiparty session following a distinct global type can be dynamically merged and interfered by another at runtime via the channel delegation operation, grouping several structured conversations.

Previous work on multiparty session types [12] has ignored this dynamic nature, providing a limited progress property ensured only within a single session, by assuming non-interference among different sessions and by forbidding delegation. More precisely, although the previous system assures that the multiple participants respect the protocol, by checking the types of exchanged messages and the order of communications, it cannot guarantee a *global progress*, i.e, that a protocol which merges several global scenarios will not get stuck in the middle of a session. This limitation prohibits to ensure a successful termination of a transaction, making the framework practically inapplicable to a large size of dynamically reconfigured conversations.

This paper develops, besides a more traditional *communication* type system (§ 3), a novel static *interaction* type system (§ 4) for global progress in dynamically merged and interfered multiparty, asynchronous sessions. High-level session processes equipped with global signatures are translated into low-level processes which have explicit senders and receivers (§ 2.4). Type-soundness of low-level processes is guaranteed against the local, compositional communication type system (§ 3.3). The Appendix completes definitions, examples and gives proofs: it is added for referee convenience only.

The new calculus for multiparty sessions offers three technical merits without sacrifying the original simplicity and expressivity in [12]. First it avoids the overhead of global linearity-check in [12]; secondly it provides a more liberal policy in the use of variables, both in delegation and in recursive definitions; finally it implicitly provides each participant of a service with a runtime channel indexed by its role with which he can communicate with all other participants, permitting also broadcast in a natural way. The use of indexed channels, moreover, permits to define a light-weight interaction type system for global progress.

The interaction type system automatically infers causalities of channels for the low level processes, ensuring the entire protocol, starting from the high-level processes which consist of multiple sessions, does not get stuck at intermediate sessions also in the presence of delegation.

2 Syntax and Operational Semantics

2.1 Merging Two Conversations: Three Buyer Protocol

We introduce our calculus through an example, the three-buyer protocol, extending the two-buyer protocol from [12], which includes the new features, session-multicasting and dynamically merging two conversations. The overall scenario, involving a Seller (S), Alice (A), Bob (B) and Carol (C), proceeds as follows.

1. Alice sends a book title to Seller, then Seller sends back a quote to Alice and Bob. Then 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, and Seller sends back the delivery date.
3. If the price exceeds the budget, Bob asks Carol to collaborate together by establishing a new session. Then Bob sends how much Carol must pay, then *delegates* the remaining interactions with Alice and Seller to Carol.
4. If the rest of the price is within Carol's budget, Carol accepts the quote and notifies Alice, Bob and Seller, and continues the rest of the protocol with Seller and Alice transparently, *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.

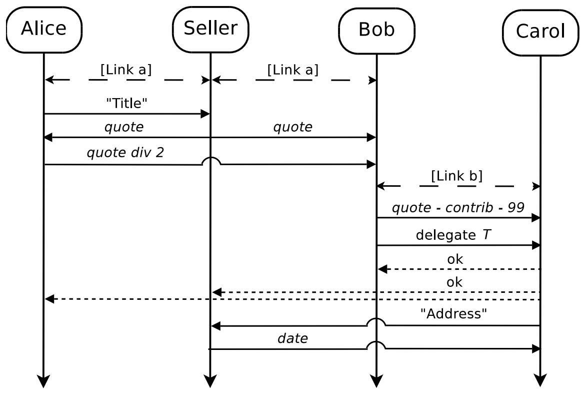

Fig. 1. The three buyer protocol interactions

Then 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 separated global types: one is G_a among Alice, Bob and Seller and the other is G_b between Bob and Carol. We write principals with legible symbols though they are actually numbers: in G_a we have $\mathtt{S} = 1$, $\mathtt{A} = 2$ and $\mathtt{B} = 3$, while in G_b we have $\mathtt{B} = 1$, $\mathtt{C} = 2$.

$$
\begin{array}{ll}
\begin{array}{l}
G_a = \\
\quad \mathtt{A} \longrightarrow \mathtt{S}: \quad \langle \mathsf{string} \rangle. \\
\quad \mathtt{S} \longrightarrow \{\mathtt{A},\mathtt{B}\}: \langle \mathsf{int} \rangle. \\
\quad \mathtt{A} \longrightarrow \mathtt{B}: \quad \langle \mathsf{int} \rangle. \\
\quad \mathtt{B} \longrightarrow \{\mathtt{S},\mathtt{A}\}: \{\mathsf{ok}: \mathtt{B} \longrightarrow \mathtt{S}: \langle \mathsf{string} \rangle. \\
\qquad\qquad\qquad\qquad\quad \mathtt{S} \longrightarrow \mathtt{B}: \langle \mathsf{date} \rangle; \mathsf{end} \\
\qquad\qquad\qquad\quad \mathsf{quit}: \mathsf{end}\}
\end{array}
&
\begin{array}{l}
G_b = \\
\quad \mathtt{B} \longrightarrow \mathtt{C}: \langle \mathsf{int} \rangle. \\
\quad \mathtt{B} \longrightarrow \mathtt{C}: \langle T \rangle. \\
\quad \mathtt{C} \longrightarrow \mathtt{B}: \{\mathsf{ok}: \mathsf{end}, \quad \mathsf{quit}: \mathsf{end}\}. \\
T = \\
\oplus(\{\mathtt{S},\mathtt{A}\}, \\
\quad \{\mathsf{ok}: !\langle \mathtt{S}, \mathsf{string} \rangle; ?\langle \mathtt{S}, \mathsf{date} \rangle; \mathsf{end}, \\
\quad \mathsf{quit}: \mathsf{end}\})
\end{array}
\end{array}
$$

The types give a global view of the two conversations, directly abstracting the scenario given by the diagram. In G_a, line 1 denotes A sends a string value to S. Line 2 says S multicasts the same integer value to A and B. In lines 4-6 B sends either ok or quit to S and A. In the first case B sends a string to and receives a date from S, in the second case there are no further communications.

Line 2 in G_b represents the delegation of the capability specified by the action type T of channels (formally defined in § 3.1) from B to C (note that S and A in T concern the session on a).

We now give the code, associated to G_a and G_b, for S, A, B and C as:

$$\mathtt{S} = \bar{a}[3](y_1).y_1?(\mathit{title}); y_1!\langle \mathit{quote} \rangle; y_1\&\{\mathsf{ok}: y_1?(\mathit{address}); y_1!\langle \mathit{date} \rangle; \mathbf{0},\ \mathsf{quit}: \mathbf{0}\}$$

$$\mathtt{A} = a[2](y_2).y_2!\langle \texttt{"Title"} \rangle; y_2?(\mathit{quote}); y_2!\langle \mathit{quote}\ \mathsf{div}\ 2 \rangle; y_2\&\{\mathsf{ok}: \mathbf{0},\ \mathsf{quit}: \mathbf{0}\}$$

$$
\begin{array}{l}
\mathtt{B} = a[3](y_3).y_3?(\mathit{quote}); y_3?(\mathit{contrib}); \\
\quad \mathsf{if}\ (\mathit{quote}\ \text{-}\ \mathit{contrib} < 100)\ \mathsf{then}\ \ y_3 \oplus \mathsf{ok}; y_3!\langle \texttt{"Address"} \rangle; y_3?(\mathit{date}); \mathbf{0} \\
\quad \mathsf{else}\ \bar{b}[2](z_1).z_1!\langle \mathit{quote}\ \text{-}\ \mathit{contrib}\ \text{-}\ 99 \rangle; z_1!\langle\langle y_3 \rangle\rangle; z_1\&\{\mathsf{ok}: \mathbf{0},\ \mathsf{quit}: \mathbf{0}\}
\end{array}
$$

$$
\begin{array}{l}
\mathtt{C} = b[2](z_2).z_2?(x); z_2?((t)); \\
\quad \mathsf{if}\ (x < 100)\ \mathsf{then}\ z_2 \oplus \mathsf{ok}; t \oplus \mathsf{ok}; t!\langle \texttt{"Address"} \rangle; t?(\mathit{date}); \mathbf{0} \\
\quad \mathsf{else}\ z_2 \oplus \mathsf{quit}; t \oplus \mathsf{quit}; \mathbf{0}
\end{array}
$$

Session name a establishes the session corresponding to G_a. S initiates a session involving three bodies as first participant by $\bar{a}[3](y_1)$: A and B participate as second and third participants by $a[2](y_2)$ and $a[3](y_3)$, respectively. Then S, A and B communicate using the channels y_1, y_2 and y_3, respectively. Each channel $y_{\mathtt{p}}$ connects participant p with all other ones; the receivers of the data sent on $y_{\mathtt{p}}$ are specified by the global type (this information will be included in the runtime code). The first line of G_a is implemented by the input and output actions $y_1?(\mathit{title})$ and $y_2!\langle\texttt{"Title"}\rangle$. The last line of G_b is implemented by the branching and selection actions $z_1\&\{\mathsf{ok} : \mathbf{0},\ \mathsf{quit} : \mathbf{0}\}$ and $z_2 \oplus \mathsf{ok}$, $z_2 \oplus \mathsf{quit}$.

In B, if the quote minus A's contribution exceeds 100€ (i.e. *quote* - *contrib* ≥ 100), another session between B and C is established dynamically through shared name b. The delegation is performed by passing the channel y_3 from B to C (actions $z_1!\langle\langle y_3\rangle\rangle$ and $z_2?((t))$), and so the rest of the session is carried out by C with S and A. We can further enrich this protocol with recursive-branching behaviours in merged sessions (for example, C can repeatedly negotiate the quote with S as if she were B). What we want to guarantee by static type-checking is that the whole merged conversation between the four parties preserves progress as if it were a single conversation.

2.2 Syntax for Multiparty Sessions

The syntax for processes initially written by the user, called *user-defined processes*, is based on [12]. We start from the following sets: *service names*, ranged over by $a, b, \ldots$ (representing public names of endpoints), *value variables*, ranged over by $x, x', \ldots$, *identifiers* , i.e., service names and variables, ranged over by $u, w, \ldots$, *channel variables*, ranged over by $y, z, t \ldots$, *labels*, ranged over by $l, l', \ldots$ (functioning like method names or labels in labelled records); *process variables*, ranged over by $X, Y, \ldots$ (used for representing recursive behaviour). Then *processes*, ranged over by $P, Q \ldots$, and *expressions*, ranged over by $e, e', \ldots$, are given by the grammar in Table 1.

$P ::= \bar{u}[n](y).P$	Multicast Request		$\mid$ $\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q$	Conditional
$\mid\ u[\mathtt{p}](y).P$	Accept		$\mid$ $P \mid Q$	Parallel
$\mid\ y!\langle e\rangle;P$	Value sending		$\mid$ $\mathbf{0}$	Inaction
$\mid\ y?(x);P$	Value reception		$\mid$ $(\nu a)P$	Hiding
$\mid\ y!\langle\langle z\rangle\rangle;P$	Session delegation		$\mid$ $\mathsf{def}\ D\ \mathsf{in}\ P$	Recursion
$\mid\ y?((z));P$	Session reception		$\mid$ $X\langle e, y\rangle$	Process call
$\mid\ y \oplus l;P$	Selection			
$\mid\ y\&\{l_i : P_i\}_{i\in I}$	Branching		$e ::= v \mid x$	
$u ::= x \mid a$	Identifier		$\mid$ $e\ \mathsf{and}\ e' \mid \mathsf{not}\ e \ldots$	Expression
$v ::= a \mid \mathsf{true} \mid \mathsf{false}$	Value		$D ::= X(x, y) = P$	Declaration

Table 1. Syntax for user-defined processes

For the primitives for session initiation, $\bar{u}[n](y).P$ initiates a new session through an identifier u (which represents a shared interaction point) with the other multiple participants, each of shape $u[\mathtt{p}](y).Q_{\mathtt{p}}$ where $2 \leq \mathtt{p} \leq n$. The (bound) variable y is the channel used to do the communications. We call p, q,... (ranging over natural numbers) the *participants* of a session. Session communications (communications that take place inside an established session) are performed using the next three pairs of primitives: the sending and receiving of a value; the session delegation and reception (where the former delegates to the latter the capability to participate in a session by passing a channel associated with the session); and the selection and branching (where the former chooses one of the branches offered by the latter). The rest of the syntax is standard from [11].

2.3 Global Types

A *global type*, ranged over by $G, G', ..$ describes the whole conversation scenario of a multiparty session as a type signature. Its grammar is given below:

$$\begin{array}{llll}
\text{Global} & G ::= \mathtt{p} \rightarrow \{\mathtt{p}_k\}_{k\in K} : \langle U\rangle.G' & \text{Exchange} & U ::= S \mid T \\
 & \mid \ \mathtt{p} \rightarrow \{\mathtt{p}_k\}_{k\in K} : \{l_i : G_i\}_{i\in I} & \text{Sorts} & S ::= \mathsf{bool} \mid \ldots \mid G \\
 & \mid \ \mu\mathbf{t}.G \ \mid \mathbf{t} \ \mid \mathsf{end} & &
\end{array}$$

We simplify the syntax in [12] by eliminating channels and parallel compositions, while preserving the original expressivity (see § 5).

The global type $\mathtt{p} \rightarrow \{\mathtt{p}_k\}_{k\in K} : \langle U\rangle.G'$ says that participant $\mathtt{p}$ multicasts a message of type U to participants $\mathtt{p}_k$ ($k \in K$): and then interactions described in G' take place. *Exchange types* $U, U', ...$ consist of *sorts* types $S, S', \ldots$ for values (either base type or global types), and *action* types $T, T', \ldots$ for channels (discussed in §3.1). Type $\mathtt{p} \rightarrow \{\mathtt{p}_k\}_{k\in K} : \{l_i : G_i\}_{i\in I}$ says participant $\mathtt{p}$ multicasts one of the labels to participants $\mathtt{p}_k$ ($k \in K$). If l_j is sent, interactions described in G_j take place. Type $\mu\mathbf{t}.G$ is a recursive type for recursive protocols, assuming type variables ($\mathbf{t}, \mathbf{t}', \ldots$) are guarded in the standard way, i.e. type variables only appear under the prefixes. We take an *equi-recursive* view of recursive types, not distinguishing between $\mu\mathbf{t}.G$ and its unfolding $G\{\mu\mathbf{t}.G/\mathbf{t}\}$ [16] (§21.8). We assume that G in the grammar of sorts is closed, i.e., without type variables. Type end represents the termination of the session. We often write $\mathtt{p} \rightarrow \mathtt{p}'$ for $\mathtt{p} \rightarrow \{\mathtt{p}'\}$.

2.4 Runtime Syntax

User defined processes equipped with global types are executed through a translation into runtime processes. The runtime syntax (Table 2) differs from the syntax of Table 1 since the input/output operations (including the delegation ones) specify the sender and the receiver, respectively. Thus, $c!\langle\{\mathtt{p}_k\}_{k\in K}, e\rangle$ sends a value to all the participants in $\{\mathtt{p}_k\}_{k\in K}$; accordingly, $c?(\mathtt{p}, x)$ denotes the intention of receiving a value from the participant $\mathtt{p}$. The same holds for delegation/reception (but the receiver is only one) and selection/branching.

$P ::= c!\langle\{\mathtt{p}_k\}_{k\in K}, e\rangle;P$	Value sending	$\mid\ c \oplus \langle\{\mathtt{p}_k\}_{k\in K}, l\rangle;P$	Selection
$\mid\ c?(\mathtt{p}, x);P$	Value reception	$\mid\ c\&(\mathtt{p}, \{l_i : P_i\}_{i\in I})$	Branching
$\mid\ c!\langle\langle \mathtt{p}, c'\rangle\rangle;P$	Session delegation	$\mid\ (\nu s)P$	Hiding session
$\mid\ c?((\mathtt{q}, y));P$	Session reception	$\mid\ s : h$	Named queue
		$\mid\ \ldots$	
$c ::= y \ \mid s[\mathtt{p}]$			Channel
$m ::= (\mathtt{q}, \{\mathtt{p}_k\}_{k\in K}, v) \ \mid (\mathtt{q}, \mathtt{p}, s[\mathtt{p}']) \ \mid (\mathtt{q}, \{\mathtt{p}_k\}_{k\in K}, l)$			Message in transit
$h ::= m\cdot h \ \mid \varnothing$			Queue

Table 2. Runtime syntax: the other syntactic forms are as in Table 1

We call $s[\mathtt{p}]$ a *channel with role*: it represents the channel of the participant $\mathtt{p}$ in the session s. We use c to range over variables and channels with roles. As in [12], in order to model TCP-like asynchronous communications (message order preservation and sender-non-blocking), we use the queues of messages in a session, denoted by h; a message in a queue can be a value message, $(\mathtt{q}, \{\mathtt{p}_k\}_{k\in K}, v)$, indicating that the value v was sent by the participant $\mathtt{q}$ and the recipients are all the participants in $\{\mathtt{p}_k\}_{k\in K}$; 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, $(\mathtt{q}, \{\mathtt{p}_k\}_{k\in K}, l)$ (similar to a value message). The empty queue is denoted by $\varnothing$. With some abuse of notation we will write $h \cdot m$ to denote that m is the last element included in h and $m \cdot h$ to denote that m

is the head of h. By $s : h$ we denote the queue h of the session s. In $(\nu s)P$ all occurrences of $s[\mathsf{p}]$ and the queue s are bound. Queues and the channel with roles $s[\mathsf{p}]$ are generated by the operational semantics (§ 2.5).

We present the translation of Bob (B) in the three buyer protocol of § 2.1 with the runtime syntax: the only difference is that all input/output operations specify also the sender and the receiver, respectively.

$$\begin{array}{l} \mathrm{B} = a[3](y_3).y_3?(1, \mathit{quote}); y_3?(2, \mathit{contrib}); \\ \quad \mathsf{if}\ (\mathit{quote}\ \text{-}\ \mathit{contrib} < 100)\ \mathsf{then}\ \ y_3 \oplus \langle\{1,2\}, \mathsf{ok}\rangle; y_3!\langle\{1\}, \texttt{"Address"}\rangle; y_3?(1, \mathit{date}); \mathbf{0} \\ \quad \mathsf{else}\ b[2](z_1).z_1!\langle\{2\}, \mathit{quote}\ \text{-}\ \mathit{contrib}\ \text{-}\ 99\rangle; z_1!\langle\langle 2, y_3\rangle\rangle; z_1\&(2, \{\mathsf{ok} : \mathbf{0},\ \mathsf{quit} : \mathbf{0}\}) \end{array}$$

It should be clear from this example that starting from a global type and user-defined processes it is possible to add sender and receivers to each communication obtaining in this way processes written in the runtime syntax.

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

2.5 Operational Semantics

$$\bar{a}[n](y_1).P_1 \mid a[2](y_2).P_2 \mid .. \mid a[n](y_n).P_n \longrightarrow (\nu s)(P_1\{s[1]/y_1\} \mid .. \mid P_n\{s[n]/y_n\} \mid s : \varnothing) \quad \text{[Link]}$$

$$s[\mathsf{p}]!\langle\{\mathsf{p}_k\}_{k\in K}, e\rangle; P \mid s : h \longrightarrow P \mid s : h \cdot (\mathsf{p}, \{\mathsf{p}_k\}_{k\in K}, v) \quad (e{\downarrow}v) \quad \text{[Send]}$$

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

$$s[\mathsf{p}] \oplus \langle\{\mathsf{p}_k\}_{k\in K}, l\rangle; P \mid s : h \longrightarrow P \mid s : h \cdot (\mathsf{p}, \{\mathsf{p}_k\}_{k\in K}, l) \quad \text{[Label]}$$

$$s[\mathsf{p}_j]?(\mathsf{q}, x); P \mid s : (\mathsf{q}, \{\mathsf{p}_k\}_{k\in K}, v) \cdot h \longrightarrow P\{v/x\} \mid s : (\mathsf{q}, \{\mathsf{p}_k\}_{k\in K\setminus j}, v) \cdot h \quad (j \in K) \quad \text{[Recv]}$$

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

$$s[\mathsf{p}_j]\&(\mathsf{q}, \{l_i : P_i\}_{i\in I}) \ \mid s : (\mathsf{q}, \{\mathsf{p}_k\}_{k\in K}, l_{i_0}) \cdot h \longrightarrow P_{i_0} \mid s : (\mathsf{q}, \{\mathsf{p}_k\}_{k\in K\setminus j}, l_{i_0}) \cdot h \quad (j \in K) \quad (i_0 \in I) \quad \text{[Branch]}$$

Table 3. Selected reduction rules

Table 3, shows the basic rules of the reduction relation $P \longrightarrow P'$.

Rule [Link] describes the initiation of a new session among n participants that synchronise over the service name a. The service provider $\bar{a}[n](y_1).P_1$ is considered as the participant 1 and specifies that it will accept $n-1$ clients. After the connection, the participants will share the private session name s, and the queue associated to s, which is initialised as empty. The variables y_p in each participant will then be replaced with the corresponding channel with role, $s[\mathsf{p}]$. The output rules [Send], [Deleg] and [Label] push 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 rules [Recv], [Srec] 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 (in particular, for [Recv], we need to remove the receiving participant from the set of the receivers in order to avoid reading the same message more than once).

Processes are considered modulo structural equivalence, denoted by $\equiv$, and defined by adding the following rules for queues to the standard ones [15].

- $s : (\mathsf{q}, \{\mathsf{p}_k\}_{k\in K}, z) \cdot (\mathsf{q}', \{\mathsf{p}_k\}_{k\in K'}, z') \cdot h \equiv s : (\mathsf{q}', \{\mathsf{p}_k\}_{k\in K'}, z') \cdot (\mathsf{q}, \{\mathsf{p}_k\}_{k\in K}, z) \cdot h$ if $K \cap K' = \emptyset$ or $\mathsf{q} \neq \mathsf{q}'$
- $s : (\mathsf{q}, \{\mathsf{p}_k\}_{k\in K}, z) \cdot h \equiv s : (\mathsf{q}, \{\mathsf{p}_k\}_{k\in K'}, z) \cdot (\mathsf{q}, \{\mathsf{p}_k\}_{k\in K''}, z) \cdot h$ with $K = K' \cup K''$ and $K' \cap K'' = \emptyset$
- $s : (\mathsf{q}, \emptyset, v) \cdot h \equiv s : h$, and $s : (\mathsf{q}, \emptyset, l) \cdot h \equiv s : h$

where z ranges over v, $s[\mathsf{p}]$ and l. The first and second rules permit rearranging messages when the senders or the receivers are not the same, and also splitting a message for multiple recipients. The last two rules garbage-collect messages that have already been read by all the intended recipients. We use $\longrightarrow^*$ and $\not\longrightarrow$ with the expected meanings.

3 Communication Type System

The previous section defines the syntax and the global types. This section introduces the communication type system, by which we can check type soundness of the communications which take place inside single sessions.

3.1 Types and Typing Rules for Pure Runtime Processes

We first define the local types of pure processes, called *action types*. While global types represent the whole protocol, action types correspond to the communication actions, representing sessions from the view-points of single participants.

$$
\begin{array}{llll}
\text{Action} & T ::= \ !\langle\{\mathsf{p}_k\}_{k\in K},U\rangle;T & \textit{send} & \mid \ \mu\mathbf{t}.T \ \textit{recursive} \\
& \mid \ ?(\mathsf{p},U);T & \textit{receive} & \mid \ \mathbf{t} \qquad \textit{variable} \\
& \mid \ \oplus\langle\{\mathsf{p}_k\}_{k\in K},\{l_i:T_i\}_{i\in I}\rangle & \textit{selection} & \mid \ \mathsf{end} \qquad \textit{end} \\
& \mid \ \&(\mathsf{p},\{l_i:T_i\}_{i\in I}) & \textit{branching} &
\end{array}
$$

The *send type* $!\langle\{\mathsf{p}_k\}_{k\in K},U\rangle;T$ expresses the sending to all p_k for $k\in K$ of a value or of a channel of type U, followed by the communications of T. The *selection type* $\oplus\langle\{\mathsf{p}_k\}_{k\in K},\{l_i:T_i\}_{i\in I}\rangle$ represents the transmission to all p_k for $k\in K$ 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* are dual and only need a sender. Other types are standard.

The relation between action and global types is formalised by the notion of projection as in [12]. The *projection of G onto* q ($G\restriction\mathsf{q}$) is defined by induction on G:

$$
(\mathsf{p}\to\{\mathsf{p}_k\}_{k\in K}:\langle U\rangle.G')\restriction\mathsf{q} =
\begin{cases}
!\langle\{\mathsf{p}_k\}_{k\in K},U\rangle;(G'\restriction\mathsf{q}) & \text{if } \mathsf{q}=\mathsf{p},\\
?(\mathsf{p},U);(G'\restriction\mathsf{q}) & \text{if } \mathsf{q}=\mathsf{p}_k \text{ for some } k\in K,\\
G'\restriction\mathsf{q} & \text{otherwise.}
\end{cases}
$$

$$
(\mathsf{p}\to\{\mathsf{p}_k\}_{k\in K}:\{l_i:G_i\}_{i\in I})\restriction\mathsf{q} =
$$
$$
\begin{cases}
\oplus(\{\mathsf{p}_k\}_{k\in K},\{l_i:G_i\restriction\mathsf{q}\}_{i\in I}) & \text{if } \mathsf{q}=\mathsf{p}\\
\&(\mathsf{p},\{l_i:G_i\restriction\mathsf{q}\}_{i\in I}) & \text{if } \mathsf{q}=\mathsf{p}_k \text{ for some } k\in K\\
G_1\restriction\mathsf{q} & \text{if } \mathsf{q}\neq\mathsf{p},\mathsf{q}\neq\mathsf{p}_k\forall k\in K \text{ and}\\
 & G_i\restriction\mathsf{q}=G_j\restriction\mathsf{q} \text{ for all } i,j\in I.
\end{cases}
$$

$$
(\mu\mathbf{t}.G)\restriction\mathsf{q}=\mu\mathbf{t}.(G\restriction\mathsf{q}) \quad \mathbf{t}\restriction\mathsf{q}=\mathbf{t} \quad \mathsf{end}\restriction\mathsf{q}=\mathsf{end}
$$

As an example, we list the projections of the global types G_a and G_b in § 2.1:

$$
\begin{array}{l}
G_a\restriction 1 = ?\langle 2,\mathsf{string}\rangle;!\langle\{2,3\},\mathsf{int}\rangle;\&(3,\{\mathsf{ok}:?\langle 3,\mathsf{string}\rangle;!\langle\{3\},\mathsf{date}\rangle;\mathsf{end},\mathsf{quit}:\mathsf{end}\})\\
G_a\restriction 2 = !\langle\{1\},\mathsf{string}\rangle;?\langle 1,\mathsf{int}\rangle;!\langle\{3\},\mathsf{int}\rangle;\&(3,\{\mathsf{ok}:\mathsf{end},\mathsf{quit}:\mathsf{end}\})\\
G_a\restriction 3 = ?\langle 1,\mathsf{int}\rangle;?\langle 2,\mathsf{int}\rangle;T\\
G_b\restriction 1 = ?\langle 2,\mathsf{int}\rangle;?\langle 1,T\rangle;\oplus\langle\{2\},\{\mathsf{ok}:\mathsf{end},\mathsf{quit}:\mathsf{end}\}\rangle\\
G_b\restriction 2 = !\langle\{1\},\mathsf{int}\rangle;!\langle\{1\},T\rangle;\&(1,\{\mathsf{ok}:\mathsf{end},\mathsf{quit}:\mathsf{end}\})
\end{array}
$$

where $T=\oplus\langle\{1,2\},\{\mathsf{ok}:!\langle\{1\},\mathsf{string}\rangle;?\langle 1,\mathsf{date}\rangle;\mathsf{end},\ \mathsf{quit}:\mathsf{end}\}\rangle$.

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

$$
\Gamma\vdash e:S \qquad \Gamma\vdash P\triangleright\Delta
$$

$$\frac{\Gamma \vdash u : \langle G\rangle \quad \Gamma \vdash P \triangleright \Delta, y : G \upharpoonright 1 \quad \mathrm{pn}(G) \le n}{\Gamma \vdash \bar{u}[n](y).P \triangleright \Delta}\lfloor\text{MCAST}\rfloor \qquad \frac{\Gamma \vdash u : \langle G\rangle \quad \Gamma \vdash P \triangleright \Delta, y : G \upharpoonright \mathrm{p}}{\Gamma \vdash u[\mathrm{p}](y).P \triangleright \Delta}\lfloor\text{MACC}\rfloor$$

$$\frac{\Gamma \vdash e : S \quad \Gamma \vdash P \triangleright \Delta, c : T}{\Gamma \vdash c!\langle\{\mathrm{p}_k\}_{k\in K}, e\rangle; P \triangleright \Delta, c : !\langle\{\mathrm{p}_k\}_{k\in K}, S\rangle; T}\lfloor\text{SEND}\rfloor \qquad \frac{\Gamma, x : S \vdash P \triangleright \Delta, c : T}{\Gamma \vdash c?(\mathrm{q}, x); P \triangleright \Delta, c : ?(\mathrm{q}, S); T}\lfloor\text{RCV}\rfloor$$

$$\frac{\Gamma \vdash P \triangleright \Delta, c : T}{\Gamma \vdash c!\langle\langle \mathrm{p}, c'\rangle\rangle; P \triangleright \Delta, c : !\langle \mathrm{p}, T'\rangle; T, c' : T'}\lfloor\text{DELEG}\rfloor \qquad \frac{\Gamma \vdash P \triangleright \Delta, c : T, y : T'}{\Gamma \vdash c?((\mathrm{q}, y)); P \triangleright \Delta, c : ?(\mathrm{q}, T'); T}\lfloor\text{SREC}\rfloor$$

$$\frac{\Gamma \vdash P \triangleright \Delta, c : T_j \quad j \in I}{\Gamma \vdash c \oplus \langle\{\mathrm{p}_k\}_{k\in K}, l_j\rangle; P \triangleright \Delta, c : \oplus\langle\{\mathrm{p}_k\}_{k\in K}, \{l_i : T_i\}_{i\in I}\rangle}\lfloor\text{SEL}\rfloor$$

$$\frac{\Gamma \vdash P_i \triangleright \Delta, c : T_i \quad \forall i \in I}{\Gamma \vdash c\&(\mathrm{p}, \{l_i : P_i\}_{i\in I}) \triangleright \Delta, c : \&(\mathrm{p}, \{l_i : T_i\}_{i\in I})}\lfloor\text{BRANCH}\rfloor$$

$$\frac{\Gamma \vdash P \triangleright \Delta \quad \Gamma \vdash Q \triangleright \Delta' \quad dom(\Delta) \cap dom(\Delta') = \emptyset}{\Gamma \vdash P \mid Q \triangleright \Delta \cup \Delta'}\lfloor\text{CONC}\rfloor$$

Table 4. Selected typing rules for pure processes

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

Formally we define:

$$\Gamma ::= \emptyset \mid \Gamma, u : S \mid \Gamma, X : S\,T \qquad \Delta ::= \emptyset \mid \Delta, c : T$$

assuming that we can write $\Gamma, u : S$ only if u does not occur in Γ, briefly $u \notin dom(\Gamma)$ ($dom(\Gamma)$ denotes the domain of Γ, i.e. the set of identifiers which occur in Γ). We use the same conventions for $X : S\,T$ and Δ.

Table 4 presents the interesting typing rules for pure processes. Rule $\lfloor$MCAST$\rfloor$ permits to type a service provider identified by u, which uses the channel y and which requires n participants, if the type of y is the first projection of the global type G of u and the number of participants in G (denoted by $\mathrm{pn}(G)$) is at least n. Note that an activation of service a can include more than $\mathrm{pn}(G_a)$ participants. The extra participants obviously cannot have interactions with the ones specified by G_a, but this feature could be useful for synchronisation purposes. Rule $\lfloor$MACC$\rfloor$ 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. The successive six rules associate the input/output processes to the input/output types in the expected way. Note that in rule $\lfloor$DELEG$\rfloor$ the channel which is sent cannot appear in the session environment of the premise, i.e. $c' \notin \Delta$: this is necessary in order to write the session environment in the conclusion. Rule $\lfloor$CONC$\rfloor$ permits to put in parallel two processes only if their sessions environments have disjoint domains.

For example we can derive:

$$\vdash t \oplus \langle\{1,2\}, \mathsf{ok}\rangle; t!\langle\{1\}, \texttt{"Address"}\rangle; t?(1, date); \mathbf{0} \triangleright \{t : T\}$$

where $T = \oplus\langle\{1,2\}, \{\mathsf{ok} : !\langle\{1\}, \mathsf{string}\rangle; ?\langle 1, \mathsf{date}\rangle; \mathsf{end},\ \mathsf{quit} : \mathsf{end}\}\rangle$. In the typing of the example of § 2.1 the types of the channels y_1, y_2, y_3, z_1, z_2 are (in order) the projections of G_a and G_b.

3.2 Types and Typing Rules for Runtime Processes

This subsection extends the communication type system to processes containing queues.

$$\begin{array}{lll@{\quad}lll}
\text{Message } \mathtt{T} ::= & !\langle\{\mathrm{p}_k\}_{k\in K},U\rangle & \textit{message send} & \text{Generalised } \mathsf{T} ::= & T & \textit{action} \\
\mid & \oplus\langle\{\mathrm{p}_k\}_{k\in K},l\rangle & \textit{message selection} & \mid & \mathtt{T} & \textit{message} \\
\mid & \mathtt{T};\mathtt{T}' & \textit{message sequence} & \mid & \mathtt{T};T & \textit{continuation}
\end{array}$$

Message types are the types for queues: they represent the messages contained in the queues. The *message send type* $!\langle\{\mathrm{p}_k\}_{k\in K},U\rangle$ expresses the communication to all p_k for $k \in K$ of a value or of a channel of type U. The *message selection type* $\oplus\langle\{\mathrm{p}_k\}_{k\in K},l\rangle$ represents the communication to all p_k for $k \in K$ of the label l and $\mathtt{T};\mathtt{T}'$ represents sequencing of message types. For example $\oplus\langle\{1,2\},\mathsf{ok}\rangle$ is the message type for the message $(3,\{1,2\},\mathsf{ok})$. A *generalised type* is either an action type, or a message type, or a message type followed by an action type. Type $\mathtt{T};T$ represents the continuation of the type $\mathtt{T}$ associated to a queue with the type T associated to a pure process. An example of generalised type is $\oplus\langle\{1,2\},\mathsf{ok}\rangle;!\langle\{1\},\mathsf{string}\rangle;?\langle 1,\mathsf{date}\rangle;\mathsf{end}$.

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

- $\natural\langle\emptyset,Z\rangle;T \approx T$
- $\natural\langle\{p_k\}_{k\in K},Z\rangle;\ \natural'\langle\{p_k\}_{k\in K'},Z\rangle;T \approx\ \natural'\langle\{p_k\}_{k\in K'},Z\rangle;\ \natural\langle\{p_k\}_{k\in K},Z\rangle;T$ if $K \cap K' = \emptyset$
- $\natural\langle\{p_k\}_{k\in K},Z\rangle;T \approx\ \natural\langle\{p_k\}_{k\in K'},Z\rangle;\ \natural\langle\{p_k\}_{k\in K''},Z\rangle;T$ if $K = K' \cup K'', K' \cap K'' = \emptyset$

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 empty session environment. Each message adds an output type to the current type of the channel which has the role of the message sender. Being all these rules similar, we only show the rule for message selection:

$$\frac{\Gamma \vdash_{\{s\}} s : h \triangleright \Delta}{\Gamma \vdash_{\{s\}} s : h \cdot (\mathrm{q},\{\mathrm{p}_k\}_{k\in K},l) \triangleright \Delta;\{s[\mathrm{q}] : \oplus\langle\{\mathrm{p}_k\}_{k\in K},l\rangle\}}\ \lfloor\text{QSEL}\rfloor$$

where ; is defined by:

$$\Delta;\{s[\mathrm{q}]:\mathtt{T}\} = \begin{cases} \Delta',s[\mathrm{q}]:\mathtt{T}';\mathtt{T} & \text{if } \Delta = \Delta',s[\mathrm{q}]:\mathtt{T}', \\ \Delta,s[\mathrm{q}]:\mathtt{T} & \text{otherwise.} \end{cases}$$

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

In order to type pure processes in parallel with queues, we need to use generalised types in session environments and further typing rules. The more interesting rules are:

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

Rule $\lfloor$GINIT$\rfloor$ promotes the typing of a pure process to the typing of an arbitrary process decorating the turnstile with the empty set, since a pure process does not contain queues. When two arbitrary processes are put in parallel (rule $\lfloor$GPAR$\rfloor$) we need to require that a queue associated to the same session name does not appear twice (condition $\Sigma \cap \Sigma' = \emptyset$). In composing the two session environments we want to put in sequence a message type and an action type for the same channel with role. For this reason we define the composition $*$ between local types as:

$$\mathsf{T} * \mathsf{T}' = \begin{cases} \mathsf{T};\mathsf{T}' & \text{if } \mathsf{T} \text{ is a message type,} \\ \mathsf{T}';\mathsf{T} & \text{if } \mathsf{T}' \text{ is a message type,} \\ \bot & \text{otherwise.} \end{cases}$$

and we extend $*$ to session environments as expected:

$$\Delta * \Delta' = \Delta \backslash dom(\Delta') \cup \Delta' \backslash dom(\Delta) \cup \{c : \mathsf{T} * \mathsf{T}' \mid c : \mathsf{T} \in \Delta \ \& \ c : \mathsf{T}' \in \Delta'\}$$

Note that $*$ is commutative, i.e. $\Delta * \Delta' = \Delta' * \Delta$. Also if we can derive message types only for channels with roles, we consider the channel variables in the definition of $*$ for session environments since we want to get for example $\{y : \mathsf{end}\} * \{y : \mathsf{end}\} = \bot$. An example of derivable judgement is:

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

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

3.3 Subject Reduction

Since session environments represent the communications the channels have to do, by reducing processes we get different session environments. This can be formalised as in [12] by introducing the notion of reduction of session environments, whose rules are:

- $\{s[\mathtt{p}] : !\langle\{\mathtt{p}_k\}_{k\in K}, U\rangle;\ T, s[\mathtt{p}_j] : ?(\mathtt{p}, U); T'\} \ \Rightarrow\ \{s[\mathtt{p}] : !\langle\{\mathtt{p}_k\}_{k\in K\backslash j}, U\rangle;\ T, s[\mathtt{p}_j] : T'\}$ if $j \in K$
- $\{s[\mathtt{p}] : T; \oplus\langle\{\mathtt{p}_k\}_{k\in K}, \{l_i : T_i\}_{i\in I}\rangle\} \ \Rightarrow\ \{s[\mathtt{p}] : T; \oplus\langle\{\mathtt{p}_k\}_{k\in K}, l_i\rangle; T_i\}$
- $\{s[\mathtt{p}] : \oplus\langle\{\mathtt{p}_k\}_{k\in K}, l\rangle; T, s[\mathtt{p}_j] : \&(\mathtt{p}, \{l_i : T_i\}_{i\in I})\} \ \Rightarrow\ \{s[\mathtt{p}] : \oplus\langle\{\mathtt{p}_k\}_{k\in K\backslash j}, l\rangle; T, s[\mathtt{p}_j] : T_i\}$ if $j \in K$ and $l = l_i$
- $\Delta \cup \Delta'' \ \Rightarrow\ \Delta' \cup \Delta''$ if $\Delta \ \Rightarrow\ \Delta'$

The first rule corresponds to the reception of a value or channel by the participant $\mathtt{p}_j$, the second rule corresponds to the choice of the label l_i and the third rule corresponds to the reception of the label l by the participant $\mathtt{p}_j$.

Using the above notion we can state type preservation under reduction as follows:

Theorem 1 (Type Preservation). *If* $\Gamma \vdash_\Sigma P \triangleright \Delta$ *and* $P \longrightarrow^* P'$, *then* $\Gamma \vdash_\Sigma P' \triangleright \Delta'$ *for some* Δ' *such that* $\Delta \ \Rightarrow\ \Delta'$.

Note that the communication safety [12, Theorem 5.5] is a corollary of this theorem. Thus the user-defined processes with the global types can safely communicate since their runtime translation is typable by the communication type system described in this section.

4 Progress

This section studies progress: informally, we say that a process has the progress property if in all configurations where it is provided a suitable context (represented by another inactive process running in parallel), either (1) it does not contain channels with roles (i.e., all communications on open sessions ended) or (2) it can be further reduced.

Definition 1 (Progress). *A process P has the* progress property *if $P \longrightarrow^* P'$ implies that either P' does not contain channels with roles or $P' \mid Q \longrightarrow$ for some Q such that $P' \mid Q$ is well typed and $Q \not\longrightarrow$.*

We will give an interaction type system which ensures that the typable processes always have the progress property.

Let us say that a *channel qualifier* is either a session name or a channel variable. Let c be a channel, its channel qualifier is defined by: (1) if $c = y$, then $\ell(c) = y$; (2) else if $c = s[\mathtt{p}]$, then $\ell(c) = s$. Let Λ, ranged over by λ, denote the set of all service names and all channel qualifiers. The progress properties will be analysed via three finite sets:

two sets $\mathscr{N}$ and $\mathscr{B}$ of service names and a set $\mathscr{R} \subseteq \Lambda \cup (\Lambda \times \Lambda)$. The Cartesian product $\Lambda \times \Lambda$, whose elements are denoted $\lambda \prec \lambda'$, represents a transitive relation. The meaning of $\lambda \prec \lambda'$ is that an input action involving a channel (qualified by) λ or belonging to service λ could block a communication action involving a channel (qualified by) λ' or belonging to service λ'. Moreover $\mathscr{R}$ includes all channel qualifiers and all service names which do not belong to $\mathscr{N}$ or $\mathscr{B}$ and which occur free in the current process. This will be useful to easily extend $\mathscr{R}$ in the assignment rules, as it will be pointed out below. We call $\mathscr{N}$ *nested service set*, $\mathscr{B}$ *bound service set* and $\mathscr{R}$ *channel relation* (even if only a subset of it is, strictly speaking, a relation). Let us give now some related definitions.

Definition 2. *Let* $\mathscr{R} ::= \emptyset \mid \mathscr{R},\lambda \mid \mathscr{R},\lambda \prec \lambda'$.

1. $\mathscr{B}\bar{\cup}\{e\} = \begin{cases} \mathscr{B} \cup \{v\} & \text{if } e \downarrow v \text{ and } v \text{ is a session name} \\ \mathscr{B} & \text{otherwise.} \end{cases}$
2. $\mathscr{R} \setminus \lambda = \{\lambda_1 \prec \lambda_2 \mid \lambda_1 \prec \lambda_2 \in \mathscr{R} \;\&\; \lambda_1 \neq \lambda \;\&\; \lambda_2 \neq \lambda\} \cup \{\lambda' \mid \lambda' \in \mathscr{R} \;\&\; \lambda' \neq \lambda\}$
3. $\mathscr{R} \setminus\!\setminus \lambda = \begin{cases} \mathscr{R} \setminus \lambda & \text{if } \lambda \text{ is minimal in } \mathscr{R} \\ \perp & \text{otherwise.} \end{cases}$
4. $\mathscr{R} \uplus \mathscr{R}' = (\mathscr{R} \cup \mathscr{R}')^+$
5. $\mathsf{pre}(\ell(c),\mathscr{R}) = \mathscr{R} \uplus \{\ell(c)\} \uplus \{\ell(c) \prec \lambda \mid \lambda \in \mathscr{R} \;\&\; \ell(c) \neq \lambda\}$

where $\mathscr{R}^+$ *is the transitive closure of* $\mathscr{R}$ *and* λ *is* minimal *in* $\mathscr{R}$ *if* $\nexists \lambda' \prec \lambda \in \mathscr{R}$.

Note, as it easy to prove, that $\uplus$ is associative. A channel relation is *well formed* if it is irreflexive, and does not contain cycles. A channel relation $\mathscr{R}$ is *channel free* ($\mathsf{cf}(\mathscr{R})$) if it contains only service names.

In Table 5 we introduce selected rules for the interaction type system. The judgements are of the shape:

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

where Θ is a set of *assumptions* of the shape $X[y] \blacktriangleright \mathscr{R}\ ;\ \mathscr{N}\ ;\ \mathscr{B}$ (for recursive definitions) with the variable y representing the channel parameter of X.

We say that a judgement $\Theta \vdash P \blacktriangleright \mathscr{R}; \mathscr{N}; \mathscr{B}$ is *coherent* if: (1) $\mathscr{R}$ is well formed; (2) $\mathscr{R} \cap (\mathscr{N} \cup \mathscr{B}) = \emptyset$. We assume that the typing rules are applicable if and only if *the judgements in the conclusion are coherent*.

We will give now an informal account of the interaction typing rules, through a set of examples. It is understood that all processes introduced in the examples can be typed with the communication typing rules given in the previous section.

The crucial point to prove the progress property is to assure that a process, seen as a parallel composition of single threaded processes and queues, cannot be blocked in a configuration in which

1. there is no service initialisation asking for a connection (otherwise the process could be reactivated by providing it with the right partners) and
2. all subprocesses are either non-empty queues or processes waiting to perform an input action on a channel whose associated queue does not offer an appropriate message.

Progress inside a single service is assured by the communication typing rules in § 3. This will follow as an immediate corollary of progress (Theorem 2). The channel relation is essentially defined to analyse the interactions between services: this is why in the definition of $\mathsf{pre}(\ell(c),\mathscr{R})$ we put the condition $\ell(c) \neq \lambda$. A basic point is that a loop in $\mathscr{R}$ represents the possibility of a deadlock state. For instance take the processes:

$$\frac{\Theta \vdash P \blacktriangleright \mathscr{R}; \mathscr{N}; \mathscr{B}}{\Theta \vdash \bar{a}[n](y).P \blacktriangleright \mathscr{R}\{a/y\}; \mathscr{N}; \mathscr{B}}\{\textsc{MCast}\} \qquad \frac{\Theta \vdash P \blacktriangleright \mathscr{R}; \mathscr{N}; \mathscr{B}}{\Theta \vdash a[\mathsf{p}](y).P \blacktriangleright \mathscr{R}\{a/y\}; \mathscr{N}; \mathscr{B}}\{\textsc{MAcc}\}$$

$$\frac{\Theta \vdash P \blacktriangleright \mathscr{R}; \mathscr{N}; \mathscr{B}}{\Theta \vdash \bar{a}[n](y).P \blacktriangleright \mathscr{R}\backslash\backslash y; \mathscr{N} \cup \{a\}; \mathscr{B}}\{\textsc{MCastN}\} \qquad \frac{\Theta \vdash P \blacktriangleright \mathscr{R}; \mathscr{N}; \mathscr{B}}{\Theta \vdash a[\mathsf{p}](y).P \blacktriangleright \mathscr{R}\backslash\backslash y; \mathscr{N} \cup \{a\}; \mathscr{B}}\{\textsc{MAccN}\}$$

$$\frac{\Theta \vdash P \blacktriangleright \mathscr{R}; \mathscr{N}; \mathscr{B} \quad \mathsf{cf}(\mathscr{R}\backslash\backslash y)}{\Theta \vdash \bar{u}[n](y).P \blacktriangleright \mathscr{R}\backslash\backslash y; \mathscr{N}; \mathscr{B}\dot{\cup}\{u\}}\{\textsc{MCastB}\} \qquad \frac{\Theta \vdash P \blacktriangleright \mathscr{R}; \mathscr{N}; \mathscr{B} \quad \mathsf{cf}(\mathscr{R}\backslash\backslash y)}{\Theta \vdash u[\mathsf{p}](y).P \blacktriangleright \mathscr{R}\backslash\backslash y; \mathscr{N}; \mathscr{B}\dot{\cup}\{u\}}\{\textsc{MAccB}\}$$

$$\frac{\Theta \vdash P \blacktriangleright \mathscr{R}; \mathscr{N}; \mathscr{B}}{\Theta \vdash c!\langle\{\mathsf{p}_k\}_{k\in K}, e\rangle; P \blacktriangleright \{\ell(c)\} \cup \mathscr{R}; \mathscr{N}; \mathscr{B}\dot{\cup}\{e\}}\{\textsc{Send}\}$$

$$\frac{\Theta \vdash P \blacktriangleright \mathscr{R}; \mathscr{N}; \mathscr{B}}{\Theta \vdash c?(\mathsf{q}, x); P \blacktriangleright \mathsf{pre}(\ell(c), \mathscr{R}); \mathscr{N}; \mathscr{B}}\{\textsc{Rcv}\}$$

$$\frac{\Theta \vdash P \blacktriangleright \mathscr{R}; \mathscr{N}; \mathscr{B}}{\Theta \vdash c!\langle\langle \mathsf{p}', c'\rangle\rangle; P \blacktriangleright \{\ell(c), \ell(c'), \ell(c) \prec \ell(c')\} \uplus \mathscr{R}; \mathscr{N}; \mathscr{B}}\{\textsc{Deleg}\}$$

$$\frac{\Theta \vdash P \blacktriangleright \mathscr{R}; \mathscr{N}; \mathscr{B} \qquad \mathscr{R} \subseteq \{\ell(c), y, \ell(c) \prec y\}}{\Theta \vdash c?((\mathsf{q}, y)); P \blacktriangleright \{\ell(c)\}; \mathscr{N}; \mathscr{B}}\{\textsc{SRec}\}$$

$$\frac{\Theta \vdash P \blacktriangleright \mathscr{R}; \mathscr{N}; \mathscr{B} \quad \Theta \vdash Q \blacktriangleright \mathscr{R}'; \mathscr{N}'; \mathscr{B}'}{\Theta \vdash P \mid Q \blacktriangleright \mathscr{R} \uplus \mathscr{R}'; \mathscr{N} \cup \mathscr{N}'; \mathscr{B} \cup \mathscr{B}'}\{\textsc{Conc}\} \qquad \frac{\Theta \vdash P \blacktriangleright \mathscr{R}; \mathscr{N}; \mathscr{B} \quad a \notin \mathscr{R} \cup \mathscr{N}}{\Theta \vdash (\nu a)P \blacktriangleright \mathscr{R}; \mathscr{N}; \mathscr{B} \backslash a}\{\textsc{NRes}\}$$

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

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

Table 5. Selected interaction typing rules

$$P_1 = \bar{b}[2](y_1).\bar{a}[2](z_1).z_1?(2, x'); y_1!\langle 2, \mathsf{true}\rangle; \mathbf{0}$$
$$P_2 = b[2](y_2).a[2](z_2).y_2?(1, x); z_2!\langle 1, \mathsf{false}\rangle; \mathbf{0}$$

In process P_1 we have that an input action on service a can block 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 will then have a loop $a \prec b \prec a$. In fact $P_1 \mid P_2$ reduces to

$$Q = (\nu s)(\nu r) \ (r[1]?(2, x'); s[1]!\langle 2, \mathsf{true}\rangle; \mathbf{0} \mid s[2]?(1, x); r[2]!\langle 1, \mathsf{false}\rangle;)\mathbf{0}$$

which is stuck. It is easy to see that services a and b have the same types, thus we could change b in a in P_1 and P_2 obtaining P_1' and P_2' with two instances of service a and a relation $a \prec a$. But also $P_1' \mid P_2'$ would reduce to Q. Hence we must forbid also loops on single names (i.e. the channel relation cannot be reflexive).

Rule $\{\textsc{Rcv}\}$ asserts that the input action can block all other actions in P, while rule $\{\textsc{Send}\}$ simply adds $\ell(c)$ in $\mathscr{R}$ to register the presence of a communication action in P. In fact output is asynchronous, thus it can be always performed. Rule $\{\textsc{Deleg}\}$ is similar to $\{\textsc{Send}\}$ but asserts that a use of $\ell(c)$ must precede a use of $\ell(c')$: the relation $\ell(c) \prec \ell(c')$ needs to be registered since an action blocking $\ell(c)$ also blocks $\ell(c')$.

Three different sets of rules handle service initialisations. In rules {MCast}-{MAcc} the service name a replaces y in $\mathscr{R}$. Rules {MCastN}-{MAccN} can be applied only if the channel y associated to a is minimal in $\mathscr{R}$ .This implies that once a is initialised in P all communication actions on the channel with role instantiating y are performed before any input communication action on a different channel in P. The name a is added to the nested service set. Remarkably, via rules {MCastN}-{MAccM} we can prove progress when services are nested, generalising the typing strategy of [6]. The rules {MCastB} and {MAccB} add u to the bound service set whenever u is a service name. These rules are much more restrictive: they require that y is the only free channel in P and that it is minimal. Thus no interaction with other channels or services is possible. This safely allows u to be a variable (since nothing is known about it before execution except its type) or a restricted name (since no channel with role can be made inaccessible at runtime by a restriction on u). Note that rule {NRes} requires that a occurs neither in $\mathscr{R}$ nor in $\mathscr{N}$.

The sets $\mathscr{N}$ and $\mathscr{B}$ include all service names of a process P whose initialisations has been typed with {MCastN}-{MAccN}, {MCastB}-{MAccB}, respectively. Note that for a service name which will replace a variable this is assured by the (conditional) addition of e to $\mathscr{B}$ in the conclusion of rule {SEND}. The sets $\mathscr{N}$ and $\mathscr{B}$ are used to assure, via the coherence condition $\mathscr{R} \cap (\mathscr{N} \cup \mathscr{B}) = \emptyset$, that *all* participants to the same service are typed either by the first two rules or by the remaining four. This is crucial to assure progress. Take for instance the processes P_1 and P_2 above. If we type the session initialisation on b using rule {MCast} in P_1 and {MAccN} or {MAccB} in P_2 no inconsistency would be detected. But rule {Conc} does not type $P_1 \mid P_2$ owing to the coherence condition. Instead if we use {MAcc} in P_2, we detect the loop $a \prec b \prec a$. Note that we could not use {MCastN} or {MCastB} in P_1 since y_1 is not minimal.

Rule {SRec} avoids to create a process where two different roles in the same session are put in sequence. Following [21] we call this phenomenon self-delegation. As an example consider the processes

$$P_1 = \bar{b}[2](z_1).\bar{a}[2](y_1).y_1?((2,x));x?(1,w);z_1!\langle 2,\mathsf{false}\rangle;\mathbf{0}$$
$$P_2 = b[2](z_2).a[2](y_2).y_2!\langle\langle 1,z_2\rangle\rangle;\mathbf{0}$$

and note that $P_1 \mid P_2$ reduces to $(\nu s)(\nu r)(s[2]?(1,w);s[1]!\langle 2,\mathsf{false}\rangle;\mathbf{0})$ which is stuck. Note that $P_1 \mid P_2$ is typable by the communication type system but it is not typable by the interaction type system, since we get $y_1 \prec z_1$ which is forbidden by rule {SRec}.

A closed runtime process P is *initial* if it is typable both in the communication and in the interaction type systems. The progress property is assured for all computations that are generated from an initial process.

Theorem 2 (Progress). *All initial processes have the progress property.*

It is easy to verify that the (runtime) version of the three buyer protocol can be typed in the interaction type system with $\{a\};\{b\};\emptyset$ and $\emptyset;\{a,b\};\emptyset$ according to which typing rules we use for the initialisation actions on service names a, b. Therefore we get

Corollary 1. *The three buyer protocol has the progress property.*

5 Conclusions and Related Work

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 communication. The programmer will then write the program to implement this communication protocol; the system will use the global types (abstract model) and the program (implementation) to generate a runtime representation of the program which consists of the input/output operations decorated with explicit senders and receivers, according to the information provided in the global types. An alternative way could be that the programmer directly specifies the senders and the receivers in the communication operations as our low-level processes; the system could then infer the global types from the program. Our communication and interaction type systems will work as before in order to check the correctness and the progress of the program. Thus the programmer can choose between a top-down and a bottom-up style of programming, while relying on the same properties checked and guaranteed by the system.

We are currently designing and implementing a modelling and specification language with multiparty session types [17] for the standards of business and financial protocols with our industry collaborators [18, 19, 5]. 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 [13]) which implement conversation models. We are currently considering to extend this modelling framework with our type discipline so that we can specify and ensure progress for executable conversations.

Multiparty sessions The first papers on multiparty session types are [2] and [12]. The work [2] uses a distributed calculus where each channel connects a master end-point and one or more slave endpoints; instead of global types, they solely use (recursion-free) local types. In type checking, local types are projected to binary sessions, so that type safety is ensured using duality, but it loses sequencing information: hence a progress property in a session is not guaranteed.

The present calculus is an essential improvement from [12]; both processes and types in [12] share a vector of channels and each communication uses one of these channels, while our user processes and global types are simpler and user-friendly without these channels. The global types in [12] have a parallel composition operator, but its projectability from global to local types limits to disjoint senders and receivers; hence it does not increase expressivity.

The present calculus is more liberal than the calculus of [12] in the use of declarations, since the definition and the call of recursive processes are obliged to use the same channel variable in [12]. Similarly the delegation in [12] requires that the same channel is sent and received for ensuring subject reduction, as analysed in [21]. Our calculus solves this issue by having channels with roles, as in [9] (see the example at page 13). As a consequence some recursive processes, which are stuck in [12], are type-sound and reducible in our calculus, satisfying the interaction type system.

Different approaches to the description of service-oriented multiparty communications are taken in [3, 4]. In [3], the global and local views of protocols are described in two different calculi and the agreement between these views becomes a bisimulation between processes; [4] proposes a distributed calculus which provides communications either inside sessions or inside locations, modelling merging running sessions. The type-safety and progress in merged sessions are left as an open problem in [4].

Progress The majority of papers on service-oriented calculi only assure that clients are never stuck inside a *single* session, see [1, 7, 12] for detailed discussions, including comparisons between the session-based and the traditional behavioural type systems of mobile processes, e.g. [20, 14]. In [1, 7, 12], structured session primitives help to give simpler typing systems for progress.

The first papers considering progress for *interleaved* sessions required the nesting of overlapping sessions in Java [8, 6]. The present approach significantly improves the binary session system for progress in [7] by treating the following points: (1) asynchrony of the communication with queues, which enhances progress; (2) a general mechanism of process recursion instead of the limited permanent accepts; (3) a more liberal treatment of the channels which can be sent; and (4) the standard semantics for the reception of channels with roles, which permits to get rid of process sequencing. None of the previous work had treated progress across interfered, dynamically merged multiparty sessions.

References

1. L. Acciai and M. Boreale. A Type System for Client Progress in a Service-Oriented Calculus. In *Festschrift in honor of Ugo Montanari*, LNCS. Springer, 2008. To appear.
2. E. Bonelli and A. Compagnoni. Multipoint Session Types for a Distributed Calculus. In *TGC'07*, volume 4912 of *LNCS*, pages 240–256, 2008.
3. M. Bravetti and G. Zavattaro. Towards a Unifying Theory for Choreography Conformance and Contract Compliance. In *Software Composition*, volume 4829 of *LNCS*, pages 34–50, 2007.
4. R. Bruni, I. Lanese, H. Melgratti, and E. Tuosto. Multiparty Sessions in SOC. In *COORDINATION'08*, LNCS. Springer, 2008. To appear.
5. M. Carbone, K. Honda, and N. Yoshida. Structured Communication-Centred Programming for Web Services. In *ESOP'07*, volume 4421 of *LNCS*, pages 2–17, 2007.
6. M. Coppo, M. Dezani-Ciancaglini, and N. Yoshida. Asynchronous Session Types and Progress for Object-Oriented Languages. In *FMOODS'07*, volume 4468 of *LNCS*, pages 1–31, 2007.
7. M. Dezani-Ciancaglini, U. de' Liguoro, and N. Yoshida. On Progress for Structured Communications. In *TGC '07*, volume 4912 of *LNCS*, pages 257–275, 2008.
8. M. Dezani-Ciancaglini, D. Mostrous, N. Yoshida, and S. Drossopoulou. Session Types for Object-Oriented Languages. In *ECOOP'06*, volume 4067 of *LNCS*, pages 328–352, 2006.
9. S. Gay and M. Hole. Subtyping for Session Types in the Pi-Calculus. *Acta Informatica*, 42(2/3):191–225, 2005.
10. K. Honda. Types for Dyadic Interaction. In *CONCUR'93*, volume 715 of *LNCS*, pages 509–523. Springer, 1993.
11. K. Honda, V. T. Vasconcelos, and M. Kubo. Language Primitives and Type Disciplines for Structured Communication-based Programming. In *ESOP'98*, volume 1381 of *LNCS*, pages 22–138. Springer, 1998.
12. K. Honda, N. Yoshida, and M. Carbone. Multiparty Asynchronous Session Types. In *POPL'08*, pages 273–284. ACM, 2008.
13. R. Hu, N. Yoshida, and K. Honda. Session-Based Distributed Programming in Java. In *ECOOP'08*, LNCS. Springer, 2008. To appear.
14. N. Kobayashi. A New Type System for Deadlock-Free Processes. In *CONCUR'06*, volume 4137 of *LNCS*, pages 233–247. Springer-Verlag, 2006.
15. R. Milner. *Communicating and Mobile Systems: the π-Calculus*. CUP, 1999.
16. B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002.
17. Scribble Project. www.scribble.org.
18. UNIFI. International Organization for Standardization ISO 20022 UNIversal Financial Industry message scheme. http://www.iso20022.org, 2002.
19. Web Services Choreography Working Group. Web Services Choreography Description Language. http://www.w3.org/2002/ws/chor/.
20. N. Yoshida. Graph types for monadic mobile processes. In *FSTTCS*, volume 1180 of *LNCS*, pages 371–386, 1996.
21. N. Yoshida and V. T. Vasconcelos. Language Primitives and Type Disciplines for Structured Communication-based Programming Revisited. In *SecRet'06*, volume 171 of *ENTCS*, pages 73–93. Elsevier, 2007.

A More on Operational Semantics

Table 6 gives the full structural equivalence and Table 7 gives all the reduction rules.

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

$$(\nu r)P \mid Q \equiv (\nu r)(P \mid Q) \quad \text{if } r \notin \mathrm{fn}(Q)$$

$$(\nu r r')P \equiv (\nu r' r)P \quad (\nu r)\mathbf{0} \equiv \mathbf{0} \quad \mathsf{def}\ D\ \mathsf{in}\ \mathbf{0} \equiv \mathbf{0}$$

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

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

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

$$s : (\mathtt{q}, \emptyset, v) \cdot h \equiv s : h \qquad s : (\mathtt{q}, \emptyset, l) \cdot h \equiv s : h$$

$$s : (\mathtt{q}, \{\mathtt{p}_k\}_{k \in K}, z) \cdot (\mathtt{q}', \{\mathtt{p}_k\}_{k \in K'}, z') \cdot h \equiv s : (\mathtt{q}', \{\mathtt{p}_k\}_{k \in K'}, z') \cdot (\mathtt{q}, \{\mathtt{p}_k\}_{k \in K}, z) \cdot h$$
$$\text{if } K \cap K' = \emptyset \text{ or } \mathtt{q} \neq \mathtt{q}'$$

$$s : (\mathtt{q}, \{\mathtt{p}_k\}_{k \in K}, z) \cdot h \equiv s : (\mathtt{q}, \{\mathtt{p}_k\}_{k \in K'}, z) \cdot (\mathtt{q}, \{\mathtt{p}_k\}_{k \in K''}, z) \cdot h$$
$$\text{where } K = K' \cup K'' \text{ and } K' \cap K'' = \emptyset$$

Table 6. Structural equivalence (r ranges over a,s and z ranges over v, $s[\mathtt{p}]$ and l.)

B More on the Three Buyer Example

This appendix shows some reduction steps using the example of the three buyer protocol of § 2.4. First we list the full definition of the runtime syntax.

$\mathrm{S} = \bar{a}[3](y_1).y_1?(2,\mathit{title});y_1!\langle\{2,3\},\mathit{quote}\rangle;y_1\&(\{\mathsf{ok}: y_1?(3,\mathit{address});y_1!\langle\{3\},\mathit{date}\rangle;\mathbf{0},\ \mathsf{quit}:\mathbf{0}\},3)$

$\mathrm{A} = a[2](y_2).y_2!\langle\{1\},\texttt{"Title"}\rangle;y_2?(1,\mathit{quote});y_2!\langle\{3\},\mathit{quote}\ \mathsf{div}\ 2\rangle;y_2\&(3,\{\mathsf{ok}:\mathbf{0},\ \mathsf{quit}:\mathbf{0}\})$

$\mathrm{B} = a[3](y_3).y_3?(1,\mathit{quote});y_3?(2,\mathit{contrib});$
$\quad \mathsf{if}\ (\mathit{quote}\ \text{-}\ \mathit{contrib} < 100)\ \mathsf{then}\ \ y_3 \oplus \langle\{1,2\},\mathsf{ok}\rangle;y_3!\langle\{1\},\texttt{"Address"}\rangle;y_3?(1,\mathit{date});\mathbf{0}$
$\quad \mathsf{else}\ \bar{b}[2](z_1).z_1!\langle\{2\},\mathit{quote}\ \text{-}\ \mathit{contrib}\ \text{-}\ 99\rangle;z_1!\langle\langle 2,y_3\rangle\rangle;z_1\&(2,\{\mathsf{ok}:\mathbf{0},\ \mathsf{quit}:\mathbf{0}\})$

$\mathrm{C} = b[2](z_2).z_2?(1,x);z_2?((1,t));$
$\quad \mathsf{if}\ (x < 100)\ \mathsf{then}\ z_2 \oplus \langle\{1\},\mathsf{ok}\rangle;t \oplus \langle\{1,2\},\mathsf{ok}\rangle;t!\langle\{1\},\texttt{"Address"}\rangle;t?(1,\mathit{date});\mathbf{0}$
$\quad \mathsf{else}\ z_2 \oplus \langle\{1\},\mathsf{quit}\rangle;t \oplus \langle\{1,2\},\mathsf{quit}\rangle;\mathbf{0}$

We will consider a simplified version of the example (i.e., the Buyer3 always selects the ok label, without the if ... then ... else ...) and we will concentrate on the part involving delegation. Thus, we assume that the seller and the first two buyers have already established a connection (the session name is s^a) and that the Buyer2 is about to establish a connection with Buyer3; the first line represents the server that is waiting to conclude the transaction with participant 3. We give some reduction steps in Table 8. In the computation, Buyer3 plays the role of Buyer2 (participant 3 in the session s^a) transparently to the seller.

C More on Communication Type System

Tables 9, 10 and 11 list the typing rules for pure processes, for single queues and for processes containing queues, respectively. Note that in rules $\lfloor$DELEG$\rfloor$ and $\lfloor$QDELEG$\rfloor$ the

$$\bar{a}[n](y_1).P_1 \mid a[2](y_2).P_2 \mid .. \mid a[n](y_n).P_n \longrightarrow (\nu s)(P_1\{s[1]/y_1\} \mid .. \mid P_n\{s[n]/y_n\} \mid s : \varnothing) \quad \text{[Link]}$$

$$s[\mathtt{p}]!\langle\{\mathtt{p}_k\}_{k\in K}, e\rangle; P \mid s : h \longrightarrow P \mid s : h\cdot(\mathtt{p},\{\mathtt{p}_k\}_{k\in K}, v) \quad (e{\downarrow}v) \quad \text{[Send]}$$

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

$$s[\mathtt{p}] \oplus \langle\{\mathtt{p}_k\}_{k\in K}, l\rangle; P \mid s : h \longrightarrow P \mid s : h\cdot(\mathtt{p},\{\mathtt{p}_k\}_{k\in K}, l) \quad \text{[Label]}$$

$$s[\mathtt{p}_j]?(\mathtt{q},x); P \mid s : (\mathtt{q},\{\mathtt{p}_k\}_{k\in K}, v)\cdot h \longrightarrow P\{v/x\} \mid s : (\mathtt{q},\{\mathtt{p}_k\}_{k\in K\setminus j}, v)\cdot h \quad (j \in K) \quad \text{[Recv]}$$

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

$$s[\mathtt{p}_j]\&(\mathtt{q},\{l_i : P_i\}_{i\in I}) \mid s : (\mathtt{q},\{\mathtt{p}_k\}_{k\in K}, l_{i_0})\cdot h \longrightarrow P_{i_0} \mid s : (\mathtt{q},\{\mathtt{p}_k\}_{k\in K\setminus j}, l_{i_0})\cdot h \quad (j\in K) \quad (i_0 \in I) \quad \text{[Branch]}$$

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

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

$$P \longrightarrow P' \quad\Rightarrow\quad (\nu r)P \longrightarrow (\nu r)P' \qquad P \longrightarrow P' \quad\Rightarrow\quad P \mid Q \longrightarrow P' \mid Q \quad \text{[Scop,Par]}$$

$$P \longrightarrow P' \quad\Rightarrow\quad \mathsf{def}\ D\ \mathsf{in}\ P \longrightarrow \mathsf{def}\ D\ \mathsf{in}\ P' \quad \text{[Defin]}$$

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

Table 7. Reduction rules (r ranges over a,s)

channel with role which is sent cannot appear in the session environment of the premise. Note that the session environments of single queues can contain both action types and message types: the action types are the types of the channels with roles which are sent, while the message types are the types of the channels with roles which send values, channels or labels.

Definition 3. *The* projection of the generalised local type T onto $\mathtt{q}$, *denoted by* $T \upharpoonright \mathtt{q}$*, is defined by:*

$$(\,!\langle\{\mathtt{p}_k\}_{k\in K}, U\rangle; T') \upharpoonright \mathtt{q} = \begin{cases} !U; T' \upharpoonright \mathtt{q} & \textit{if } \mathtt{q} = \mathtt{p}_k \textit{ for some } k \in K, \\ T' \upharpoonright \mathtt{q} & \textit{otherwise.} \end{cases}$$

$$(?(\{\mathtt{p}_k\}_{k\in K}, U); T') \upharpoonright \mathtt{q} = \begin{cases} ?U; T' \upharpoonright \mathtt{q} & \textit{if } \mathtt{q} = \mathtt{p}_k \textit{ for some } k \in K, \\ T' \upharpoonright \mathtt{q} & \textit{otherwise.} \end{cases}$$

$$(\oplus\langle\{\mathtt{p}_k\}_{k\in K}, \{l_i : T_i\}_{i\in I}\rangle) \upharpoonright \mathtt{q} = \begin{cases} \oplus\{l_i : T_i \upharpoonright \mathtt{q}\}_{i\in I} & \textit{if } \mathtt{q} = \mathtt{p}_k \textit{ for some } k \in K, \\ T_1 \upharpoonright \mathtt{q} & \textit{if } \mathtt{q} \neq \mathtt{p}_k\ \forall k \in K \textit{ and} \\ & T_i \upharpoonright \mathtt{q} = T_j \upharpoonright \mathtt{q} \\ & \textit{for all } i,j \in I. \end{cases}$$

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

$$(\oplus\langle\{\mathtt{p}_k\}_{k\in K}, l\rangle; T') \upharpoonright \mathtt{q} = \begin{cases} \oplus l; T' \upharpoonright \mathtt{q} & \textit{if } \mathtt{q} = \mathtt{p}_k \textit{ for some } k \in K, \\ T' \upharpoonright \mathtt{q} & \textit{otherwise.} \end{cases}$$

$$(\mu \mathbf{t}.T) \upharpoonright \mathtt{q} = \mu \mathbf{t}.(T \upharpoonright \mathtt{q}) \qquad \mathbf{t} \upharpoonright \mathtt{q} = \mathbf{t} \qquad \mathsf{end} \upharpoonright \mathtt{q} = \mathsf{end}$$

$$(\nu s^a)(s^a[1]\&(3,\{\mathsf{ok} : s^a[1]?(3,\mathit{address});s^a[1]!\langle\{3\},\mathit{date}\rangle;\mathbf{0},\ \mathsf{quit} : \mathbf{0}\}) \mid$$
$$b[2](z_1).z_1!\langle\{2\},\mathit{quote}\text{ - }\mathit{contrib}\text{ - }99\rangle;z_1!\langle\langle 2,s^a[3]\rangle\rangle;\ldots) \mid$$
$$\bar{b}[2](z_2).z_2?(1,x);z_2?((1,t));z_2 \oplus \langle\{1\},\mathsf{ok}\rangle;t \oplus \langle\{1,2\},\mathsf{ok}\rangle;t!\langle\{1\},\ldots\rangle;t?(1,\mathit{date})$$

$\longrightarrow$ by using [Link] (and the structural congruence for scope extrusion)

$$(\nu s^a s^b)(\ldots\text{as above}\ldots \mid s^b[1]!\langle\{2\},\mathit{quote}\text{ - }\mathit{contrib}\text{ - }99\rangle;s^b[1]!\langle\langle 2,s^a[3]\rangle\rangle;\ldots \mid$$
$$s^b[2]?(1,x);s^b[2]?((1,t));s^b[2] \oplus \langle\{1\},\mathsf{ok}\rangle;t \oplus \langle\{1,2\},\mathsf{ok}\rangle;t!\langle\ldots,\{1\}\rangle;t?(1,\mathit{date}))$$

$\longrightarrow^*$ by using [Send] and [Recv] the result of *quote - contrib* - 99 is communicated

$$(\nu s^a s^b)(\ldots\text{as above}\ldots \mid s^b[1]!\langle\langle 2,s^a[3]\rangle\rangle;s^b[1]\&(2,\{\mathsf{ok} : \mathbf{0},\ \mathsf{quit} : \mathbf{0}\}) \mid$$
$$s^b[2]?((1,t));s^b[2] \oplus \langle\{1\},\mathsf{ok}\rangle;t \oplus \langle\{1,2\},\mathsf{ok}\rangle;t!\langle\ldots,\{1\}\rangle;t?(1,\mathit{date}))$$

$\longrightarrow^*$ by using [Deleg] and [Srec]

$$(\nu s^a s^b)(\ldots\text{as above}\ldots \mid s^b[1]\&(2,\{\mathsf{ok} : \mathbf{0},\ \mathsf{quit} : \mathbf{0}\}) \mid$$
$$s^b[2] \oplus \langle\{1\},\mathsf{ok}\rangle;s^a[3] \oplus \langle\{1,2\},\mathsf{ok}\rangle;s^a[3]!\langle\{1\},\ldots\rangle;s^a[3]?(1,\mathit{date}))$$

$\longrightarrow^*$ by using [Label] and [Branch]

$$(\nu s^a s^b)(s^a[1]\&(3,\{\mathsf{ok} : s^a[1]?(3,\mathit{address});s^a[1]!\langle\{3\},\mathit{date}\rangle;\mathbf{0},\ \mathsf{quit} : \mathbf{0}\}) \mid \mathbf{0} \mid$$
$$s^a[3] \oplus \langle\{1,2\},\mathsf{ok}\rangle;s^a[3]!\langle\{1\},\ldots\rangle;s^a[3]?(1,\mathit{date}))$$

Table 8. Example of reduction

Definition 4. *The* duality relation *between projections of generalised local types is the minimal symmetric relation which satisfies:*

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

Definition 5. *A session environment* Δ *is* coherent for the session s *(notation* $\mathsf{co}(\Delta,s)$*) if* $s[\mathsf{p}] : T \in \Delta$ *and* $T \upharpoonright \mathsf{q} \neq \mathsf{end}$ *imply* $s[\mathsf{q}] : T' \in \Delta$ *and* $T \upharpoonright \mathsf{q} \bowtie T' \upharpoonright \mathsf{p}$. *A session environment* Δ *is* coherent *if it is coherent for all sessions which occur in it.*

D From User Syntax to Runtime Syntax via Types

Given a user process P and the set of global types associated to the service identifiers which occur free or bound in P we can add the sender and the receivers to each communication, by getting in this way a process in the runtime syntax. We define two mappings with domain the set of user processes: the first one (denote by $\lfloor G \dagger u \rfloor$) depends on a global type G and on a service identifier u, while the second one (denote by $\lfloor T \ddagger y \rfloor$) depends on an action type T and on a channel variable y. The mapping $\lfloor G \dagger u \rfloor$ (Table 12) calls the other mapping with the appropriate projection and channel variable when it is applied to a session initiation on the identifier u, and leaves the process unchanged otherwise. The mapping $\lfloor T \ddagger y \rfloor$ (Table 13) adds the sender or the receiver to the communications which use the channel y and it does not affect the other processes. An interesting clause is the fifth one, in which $\lfloor T' \ddagger y' \rfloor$ is applied to the body of the channel reception y' (T' is the action type of y'). In the last but one clause T' is the unique type such that $\lfloor T' \ddagger y \rfloor(X(e\,y))$ occurs in (the evaluation of) $\lfloor T \ddagger y \rfloor(Q)$. More precisely we evaluate this type by applying to Q the mapping $\lfloor T \natural y \natural X \rfloor()$ defined in Table 14.

In order to get the runtime version of an user process P we need to apply to P the mapping $\lfloor G \dagger u \rfloor$, for each service identifier u which occurs free or bound in P, where G

$$\Gamma, u : S \vdash u : S \ \lfloor\text{NAME}\rfloor \qquad \Gamma \vdash \text{ true, false : bool} \qquad \frac{\Gamma \vdash e_i : \text{bool}}{\Gamma \vdash e_1 \text{ and } e_2 : \text{bool}} \ \lfloor\text{BOOL}\rfloor, \lfloor\text{AND}\rfloor$$

$$\frac{\Gamma \vdash u : \langle G \rangle \quad \Gamma \vdash P \triangleright \Delta, y : G \upharpoonright 1 \quad \text{pn}(G) \leq n}{\Gamma \vdash \bar{u}[n](y).P \triangleright \Delta} \lfloor\text{MCAST}\rfloor \quad \frac{\Gamma \vdash u : \langle G \rangle \quad \Gamma \vdash P \triangleright \Delta, y : G \upharpoonright \text{p}}{\Gamma \vdash u[\text{p}](y).P \triangleright \Delta} \lfloor\text{MACC}\rfloor$$

$$\frac{\Gamma \vdash e : S \quad \Gamma \vdash P \triangleright \Delta, c : T}{\Gamma \vdash c!\langle\{\text{p}_k\}_{k \in K}, e\rangle; P \triangleright \Delta, c : !\langle\{\text{p}_k\}_{k \in K}, S\rangle; T} \lfloor\text{SEND}\rfloor \quad \frac{\Gamma, x : S \vdash P \triangleright \Delta, c : T}{\Gamma \vdash c?(\text{q}, x); P \triangleright \Delta, c : ?(\text{q}, S); T} \lfloor\text{RCV}\rfloor$$

$$\frac{\Gamma \vdash P \triangleright \Delta, c : T}{\Gamma \vdash c!\langle\langle \text{p}, c' \rangle\rangle; P \triangleright \Delta, c : !\langle \text{p}, T' \rangle; T, c' : T'} \lfloor\text{DELEG}\rfloor \quad \frac{\Gamma \vdash P \triangleright \Delta, c : T, y : T'}{\Gamma \vdash c?((\text{q}, y)); P \triangleright \Delta, c : ?(\text{q}, T'); T} \lfloor\text{SREC}\rfloor$$

$$\frac{\Gamma \vdash P \triangleright \Delta, c : T_j \quad j \in I}{\Gamma \vdash c \oplus \langle\{\text{p}_k\}_{k \in K}, l_j\rangle; P \triangleright \Delta, c : \oplus\langle\{\text{p}_k\}_{k \in K}, \{l_i : T_i\}_{i \in I}\rangle} \lfloor\text{SEL}\rfloor$$

$$\frac{\Gamma \vdash P_i \triangleright \Delta, c : T_i \quad \forall i \in I}{\Gamma \vdash c\&(\text{p}, \{l_i : P_i\}_{i \in I}) \triangleright \Delta, c : \&(\text{p}, \{l_i : T_i\}_{i \in I})} \lfloor\text{BRANCH}\rfloor$$

$$\frac{\Gamma \vdash P \triangleright \Delta \quad \Gamma \vdash Q \triangleright \Delta' \quad dom(\Delta) \cap dom(\Delta') = \emptyset}{\Gamma \vdash P \mid Q \triangleright \Delta \cup \Delta'} \lfloor\text{CONC}\rfloor$$

$$\frac{\Gamma \vdash e : \text{bool} \quad \Gamma \vdash P \triangleright \Delta \quad \Gamma \vdash Q \triangleright \Delta}{\Gamma \vdash \text{if } e \text{ then } P \text{ else } Q \triangleright \Delta} \lfloor\text{IF}\rfloor \quad \frac{\Delta \text{ end only}}{\Gamma \vdash \mathbf{0} \triangleright \Delta} \lfloor\text{INACT}\rfloor \quad \frac{\Gamma, a : \langle G \rangle \vdash P \triangleright \Delta}{\Gamma \vdash (\nu a) P \triangleright \Delta} \lfloor\text{NRES}\rfloor$$

$$\frac{\Gamma \vdash e : S \quad \Delta \text{ end only}}{\Gamma, X : S\,T \vdash X\langle e, c \rangle \triangleright \Delta, c : T} \lfloor\text{VAR}\rfloor \quad \frac{\Gamma, X : S\,T, x : S \vdash P \triangleright y : T \quad \Gamma, X : S\,T \vdash Q \triangleright \Delta}{\Gamma \vdash \text{def } X(x, y) = P \text{ in } Q \triangleright \Delta} \lfloor\text{DEF}\rfloor$$

Table 9. Typing rules for pure processes

is the global type of u. Note that when u is a bound variable we need to apply $\lfloor G \dagger x \rfloor$ only to the scope of x.

We say that a closed user process $P = \mathscr{C}[y_1?(x_1); Q_1] \ldots [y_m?(x_m); Q_m]$ with bound service identifiers $x_1, \ldots, x_m$ and service names a_ℓ with $\ell \in L$ is a correct implementation of the protocols described by $G_1, \ldots, G_m$ and G'_ℓ for $\ell \in L$ if we can derive

$$\lfloor G'_\ell \dagger a_\ell \rfloor_{\ell \in L}(\mathscr{C}[y_1?(x_1); \lfloor G_1 \dagger x_1 \rfloor(Q_1)] \ldots [y_m?(x_m); \lfloor G_m \dagger x_m \rfloor(Q_m)]) \triangleright \emptyset$$

from $\{a_\ell : G'_\ell \mid \ell \in L\}$.

For example by applying $\lfloor G_a \dagger a \rfloor$ and $\lfloor G_b \dagger b \rfloor$ to the three buyer protocol of Subsection 2.1 we get the three buyer protocol of Subsection 2.4.

E More on Progress

Tables 15 and 16 give the interaction typing rules.

$$\frac{}{\Gamma \vdash_{\{s\}} s : \varnothing \triangleright \emptyset}\ \lfloor\text{QINIT}\rfloor \qquad \frac{\Gamma \vdash_{\{s\}} s : h \triangleright \Delta \qquad \Gamma \vdash v : S}{\Gamma \vdash_{\{s\}} s : h \cdot (\mathsf{q}, \{\mathsf{p}_k\}_{k \in K}, v) \triangleright \Delta ; \{s[\mathsf{q}] : !\langle \{\mathsf{p}_k\}_{k \in K}, S\rangle\}}\ \lfloor\text{QSEND}\rfloor$$

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

$$\frac{\Gamma \vdash_{\{s\}} s : h \triangleright \Delta}{\Gamma \vdash_{\{s\}} s : h \cdot (\mathsf{q}, \{\mathsf{p}_k\}_{k \in K}, l) \triangleright \Delta ; \{s[\mathsf{q}] : \oplus\langle \{\mathsf{p}_k\}_{k \in K}, l\rangle\}}\ \lfloor\text{QSEL}\rfloor$$

Table 10. Typing rules for queues

$$\frac{\Gamma \vdash P \triangleright \Delta}{\Gamma \vdash_{\emptyset} P \triangleright \Delta}\ \lfloor\text{GINIT}\rfloor \qquad \frac{\Gamma \vdash_{\Sigma} P \triangleright \Delta \quad \Delta' \text{end only}}{\Gamma \vdash_{\Sigma} P \triangleright \Delta * \Delta'}\ \lfloor\text{WEAK}\rfloor$$

$$\frac{\Gamma \vdash_{\Sigma} P \triangleright \Delta \quad \Gamma \vdash_{\Sigma'} Q \triangleright \Delta' \quad \Sigma \cap \Sigma' = \emptyset}{\Gamma \vdash_{\Sigma \cup \Sigma'} P \mid Q \triangleright \Delta * \Delta'}\ \lfloor\text{GPAR}\rfloor$$

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

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

Table 11. Typing rules for processes

F Proofs

F.1 Proof of Subject Reduction for the Communication Type System

Lemma 1 (Inversion Lemma for Pure Processes).

1. *If* $\Gamma \vdash u : S$*, then* $u : S \in \Gamma$.
2. *If* $\Gamma \vdash \mathsf{true} : S$*, then* $S = \mathsf{bool}$.
3. *If* $\Gamma \vdash \mathsf{false} : S$*, then* $S = \mathsf{bool}$.
4. *If* $\Gamma \vdash e_1 \ \mathsf{and}\ e_2 : S$*, then* $\Gamma \vdash e_1, e_2 : \mathsf{bool}, S = \mathsf{bool}$.
5. *If* $\Gamma \vdash \bar{a}[n](y).P \triangleright \Delta$*, then* $\Gamma \vdash a : \langle G\rangle$ *and* $\Gamma \vdash P \triangleright \Delta, y : G \upharpoonright 1$ *and* $\mathrm{pn}(G) \leq n$.
6. *If* $\Gamma \vdash a[\mathsf{p}](y).P \triangleright \Delta$*, then* $\Gamma \vdash a : \langle G\rangle$ *and* $\Gamma \vdash P \triangleright \Delta, y : G \upharpoonright \mathsf{p}$.

$$\begin{array}{ll}
\lfloor G \dagger u \rfloor (\bar{u}[n](y).P) = \bar{u}[n](y).\lfloor G \upharpoonright 1 \ddagger y \rfloor (P) & \\
\lfloor G \dagger u \rfloor (u[\mathsf{p}](y).P) = u[\mathsf{p}](y).\lfloor G \upharpoonright \mathsf{p} \ddagger y \rfloor (P) & \\
\lfloor G \dagger u \rfloor (\mathsf{pref}; P) = \mathsf{pref}; \lfloor G \dagger u \rfloor (P) & u \notin \mathsf{pref} \\
\lfloor G \dagger u \rfloor (\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q) = \mathsf{if}\ e\ \mathsf{then}\ \lfloor G \dagger u \rfloor (P)\ \mathsf{else}\ \lfloor G \dagger u \rfloor (Q) & \\
\lfloor G \dagger u \rfloor (P \mid Q) = \lfloor G \dagger u \rfloor (P) \mid \lfloor G \dagger u \rfloor (Q) & \\
\lfloor G \dagger u \rfloor (\mathbf{0}) = \mathbf{0} & \\
\lfloor G \dagger u \rfloor ((\nu a) P) = (\nu a) \lfloor G \dagger u \rfloor (P) & \\
\lfloor G \dagger u \rfloor (\mathsf{def}\ X(x\, y) = P\ \mathsf{in}\ Q) = \mathsf{def}\ X(x\, y) = \lfloor G \dagger u \rfloor (P)\ \mathsf{in}\ \lfloor G \dagger u \rfloor (Q) & \\
\lfloor G \dagger u \rfloor (X\langle e\, y\rangle) = X\langle e\, y\rangle &
\end{array}$$

where pref is any session initialization or communication command.

Table 12. Application of a global type and a service identifier to a user process.

$$\begin{array}{ll}
\lfloor\,!\langle\{\mathtt{p}_k\}_{k\in K},S\rangle;T \ddagger y\rfloor(y!\langle e\rangle;P) = y!\langle\{\mathtt{p}_k\}_{k\in K},e\rangle;\lfloor T \ddagger y\rfloor(P) & \\
\lfloor ?(\mathtt{p},S);T \ddagger y\rfloor(y?(x);P) = y?(\mathtt{p},x);\lfloor T \ddagger y\rfloor(P) & \\
\lfloor\,!\langle\{\mathtt{p}_k\}_{k\in K},T'\rangle;T \ddagger y\rfloor(y!\langle\langle y'\rangle\rangle;P) = y!\langle\langle\{\mathtt{p}_k\}_{k\in K},y'\rangle\rangle;\lfloor T \ddagger y\rfloor(P) & \\
\lfloor T \ddagger y\rfloor(y'!\langle\langle y\rangle\rangle;P) = y'!\langle\langle y\rangle\rangle;P & \\
\lfloor ?(\mathtt{p},T');T \ddagger y\rfloor(y?((y'));P) = y?((\mathtt{p},y'));\lfloor T \ddagger y\rfloor(\lfloor T' \ddagger y'\rfloor(P)) & \\
\lfloor \oplus\langle\{\mathtt{p}_k\}_{k\in K},\{l_i:T_i\}_{i\in I}\rangle \ddagger y\rfloor(y\oplus l_j;P) = y\oplus\langle\mathtt{p},l_j\rangle;\lfloor T_j \ddagger y\rfloor(P) & j\in I \\
\lfloor \&(\mathtt{p},\{l_i:T_i\}_{i\in I}) \ddagger y\rfloor(y\&\{l_i:P_i\}_{i\in I}) = y\&(\mathtt{p},\{l_i:\lfloor T \ddagger y\rfloor(P_i)\}_{i\in I}) & \\
\lfloor T \ddagger y\rfloor(\mathsf{pref};P) = \mathsf{pref};\lfloor T \ddagger y\rfloor(P) & y\notin \mathsf{pref} \\
\lfloor T \ddagger y\rfloor(\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q) = \mathsf{if}\ e\ \mathsf{then}\ \lfloor T \ddagger y\rfloor(P)\ \mathsf{else}\ \lfloor T \ddagger y\rfloor(Q) & \\
\lfloor T \ddagger y\rfloor(P \mid Q) = \lfloor T \ddagger y\rfloor(P) \mid Q & y\notin Q \\
\lfloor T \ddagger y\rfloor(P \mid Q) = P \mid \lfloor T \ddagger y\rfloor(Q) & y\notin P \\
\lfloor \mathsf{end} \ddagger y\rfloor(\mathbf{0}) = \mathbf{0} & \\
\lfloor T \ddagger y\rfloor((\nu a)P) = (\nu a)\lfloor T \ddagger y\rfloor(P) & \\
\lfloor T \ddagger y\rfloor(\mathsf{def}\ X(x\,y') = P\ \mathsf{in}\ Q) = \mathsf{def}\ X(x\,y') = \lfloor T' \ddagger y'\rfloor(P)\ \mathsf{in}\ \lfloor T \ddagger y\rfloor(Q) & \\
\quad \text{where } T' = \lfloor T \natural\, y \natural\, X\rfloor(Q) & \\
\lfloor T \ddagger y\rfloor(X\langle e\,y'\rangle) = X\langle e\,y'\rangle &
\end{array}$$

Table 13. Application of a local type and a channel variable to a user process.

$$\begin{array}{ll}
\lfloor\,!\langle\{\mathtt{p}_k\}_{k\in K},S\rangle;T \natural\, y \natural\, X\rfloor(y!\langle e\rangle;P) = \lfloor T \natural\, y \natural\, X\rfloor(P) & \\
\lfloor ?(\mathtt{p},S);T \natural\, y \natural\, X\rfloor(y?(x);P) = \lfloor T \natural\, y \natural\, X\rfloor(P) & \\
\lfloor\,!\langle\{\mathtt{p}_k\}_{k\in K},T'\rangle;T \natural\, y \natural\, X\rfloor(y!\langle\langle y'\rangle\rangle;P) = \lfloor T \natural\, y \natural\, X\rfloor(P) & \\
\lfloor ?(\mathtt{p},T');T \natural\, y \natural\, X\rfloor(y?((y'));P) = \lfloor T \natural\, y \natural\, X\rfloor(P) & \\
\lfloor \oplus\langle\{\mathtt{p}_k\}_{k\in K},\{l_i:T_i\}_{i\in I}\rangle \natural\, y \natural\, X\rfloor(y\oplus l_j;P) = \lfloor T_j \natural\, y \natural\, X\rfloor(P) & j\in I \\
\lfloor \&(\mathtt{p},\{l_i:T_i\}_{i\in I}) \natural\, y \natural\, X\rfloor(y\&\{l_i:P_i\}_{i\in I}) = \lfloor T_j \natural\, y \natural\, X\rfloor(P_j) & j\in I\ \&\ X\in P_j \\
\lfloor T \natural\, y \natural\, X\rfloor(\mathsf{pref};P) = \lfloor T \natural\, y \natural\, X\rfloor(P) & y\notin \mathsf{pref} \\
\lfloor T \natural\, y \natural\, X\rfloor(\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q) = \lfloor T \natural\, y \natural\, X\rfloor(P) & X\in P \\
\lfloor T \natural\, y \natural\, X\rfloor(\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q) = \lfloor T \natural\, y \natural\, X\rfloor(Q) & X\in Q \\
\lfloor T \natural\, y \natural\, X\rfloor(P \mid Q) = \lfloor T \natural\, y \natural\, X\rfloor(P) & X\in P \\
\lfloor T \natural\, y \natural\, X\rfloor(P \mid Q) = \lfloor T \natural\, y \natural\, X\rfloor(Q) & X\in Q \\
\lfloor T \natural\, y \natural\, X\rfloor((\nu a)P) = \lfloor T \natural\, y \natural\, X\rfloor(P) & \\
\lfloor T \natural\, y \natural\, X\rfloor(\mathsf{def}\ X'(x\,y') = P\ \mathsf{in}\ Q) = \lfloor T \natural\, y \natural\, X\rfloor(Q) & X\neq X' \\
\lfloor T \natural\, y \natural\, X\rfloor(X\langle e\,y'\rangle) = T &
\end{array}$$

Table 14. Application of a local type and a channel variable and a process variable to a user process.

7. *If* $\Gamma \vdash c!\langle\{\mathtt{p}_k\}_{k\in K},e\rangle;P \triangleright \Delta$*, then* $\Delta = \Delta',c:\,!\langle\{\mathtt{p}_k\}_{k\in K},S\rangle;T$ *and* $\Gamma \vdash e:S$ *and* $\Gamma \vdash P \triangleright \Delta',c:T$.
8. *If* $\Gamma \vdash c?(\mathtt{q},x);P \triangleright \Delta$*, then* $\Delta = \Delta',c:?(\mathtt{q},S);T$ *and* $\Gamma,x:S \vdash P \triangleright \Delta',c:T$.
9. *If* $\Gamma \vdash c!\langle\langle \mathtt{p},c'\rangle\rangle;P \triangleright \Delta$*, then* $\Delta = \Delta',c:\,!\langle \mathtt{p},T'\rangle;T,c':T'$ *and* $\Gamma \vdash P \triangleright \Delta',c:T$.
10. *If* $\Gamma \vdash c?((\mathtt{q},y));P \triangleright \Delta$*, then* $\Delta = \Delta',c:?(\mathtt{q},T');T$ *and* $\Gamma \vdash P \triangleright \Delta',c:T,y:T'$.
11. *If* $\Gamma \vdash c \oplus \langle\{\mathtt{p}_k\}_{k\in K},l_j\rangle;P \triangleright \Delta$*, then* $\Delta = \Delta',c:\oplus\langle\{\mathtt{p}_k\}_{k\in K},\{l_i:T_i\}_{i\in I}\rangle$ *and* $\Gamma \vdash P \triangleright \Delta',c:T_j$ *and* $j\in I$.
12. *If* $\Gamma \vdash c\&(\mathtt{p},\{l_i:P_i\}_{i\in I}) \triangleright \Delta$*, then* $\Delta = \Delta',c:\&(\mathtt{p},\{l_i:T_i\}_{i\in I})$ *and* $\Gamma \vdash P_i \triangleright \Delta',c:T_i \quad \forall i\in I$.
13. *If* $\Gamma \vdash P \mid Q \triangleright \Delta$*, then* $\Delta = \Delta' \cup \Delta''$ *and* $\Gamma \vdash P \triangleright \Delta'$ *and* $\Gamma \vdash Q \triangleright \Delta''$ *where* $dom(\Delta') \cap dom(\Delta'') = \emptyset$.
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* $\Gamma \vdash (\nu a)P \triangleright \Delta$*, then* $\Gamma,a:\langle G\rangle \vdash P \triangleright \Delta$.
17. *If* $\Gamma,X:S\,T \vdash X\langle e,c\rangle \triangleright \Delta$*, then* $\Delta = \Delta',c:T$ *and* $\Gamma \vdash e:S$ *and* Δ' end *only.*

$$\frac{\Theta \vdash P \blacktriangleright \mathscr{R};\mathscr{N};\mathscr{B}}{\Theta \vdash \bar{a}[n](y).P \blacktriangleright \mathscr{R}\{a/y\};\mathscr{N};\mathscr{B}}\;\{\textsc{MCast}\} \qquad \frac{\Theta \vdash P \blacktriangleright \mathscr{R};\mathscr{N};\mathscr{B}}{\Theta \vdash a[\mathsf{p}](y).P \blacktriangleright \mathscr{R}\{a/y\};\mathscr{N};\mathscr{B}}\;\{\textsc{MAcc}\}$$

$$\frac{\Theta \vdash P \blacktriangleright \mathscr{R};\mathscr{N};\mathscr{B}}{\Theta \vdash \bar{a}[n](y).P \blacktriangleright \mathscr{R}\backslash\backslash y;\mathscr{N}\cup\{a\};\mathscr{B}}\;\{\textsc{MCastN}\} \quad \frac{\Theta \vdash P \blacktriangleright \mathscr{R};\mathscr{N};\mathscr{B}}{\Theta \vdash a[\mathsf{p}](y).P \blacktriangleright \mathscr{R}\backslash\backslash y;\mathscr{N}\cup\{a\};\mathscr{B}}\;\{\textsc{MAccN}\}$$

$$\frac{\Theta \vdash P \blacktriangleright \mathscr{R};\mathscr{N};\mathscr{B} \quad \mathsf{cf}(\mathscr{R}\backslash\backslash y)}{\Theta \vdash \bar{u}[n](y).P \blacktriangleright \mathscr{R}\backslash\backslash y;\mathscr{N};\mathscr{B}\bar{\cup}\{u\}}\;\{\textsc{MCastB}\} \quad \frac{\Theta \vdash P \blacktriangleright \mathscr{R};\mathscr{N};\mathscr{B} \quad \mathsf{cf}(\mathscr{R}\backslash\backslash y)}{\Theta \vdash u[\mathsf{p}](y).P \blacktriangleright \mathscr{R}\backslash\backslash y;\mathscr{N};\mathscr{B}\bar{\cup}\{u\}}\;\{\textsc{MAccB}\}$$

$$\frac{\Theta \vdash P \blacktriangleright \mathscr{R};\mathscr{N};\mathscr{B}}{\Theta \vdash c!\langle\{\mathsf{p}_k\}_{k\in K},e\rangle;P \blacktriangleright \{\ell(c)\}\cup\mathscr{R};\mathscr{N};\mathscr{B}\bar{\cup}\{e\}}\;\{\textsc{Send}\}$$

$$\frac{\Theta \vdash P \blacktriangleright \mathscr{R};\mathscr{N};\mathscr{B}}{\Theta \vdash c?(\mathsf{q},x);P \blacktriangleright \mathsf{pre}(\ell(c),\mathscr{R});\mathscr{N};\mathscr{B}}\;\{\textsc{Rcv}\}$$

$$\frac{\Theta \vdash P \blacktriangleright \mathscr{R};\mathscr{N};\mathscr{B}}{\Theta \vdash c!\langle\langle \mathsf{p}',c'\rangle\rangle;P \blacktriangleright \{\ell(c),\ell(c'),\ell(c)\prec\ell(c')\}\uplus\mathscr{R};\mathscr{N};\mathscr{B}}\;\{\textsc{Deleg}\}$$

$$\frac{\Theta \vdash P \blacktriangleright \mathscr{R};\mathscr{N};\mathscr{B} \qquad \mathscr{R}\subseteq\{\ell(c),y,\ell(c)\prec y\}}{\Theta \vdash c?((\mathsf{q},y));P \blacktriangleright \{\ell(c)\};\mathscr{N};\mathscr{B}}\;\{\textsc{SRec}\}$$

$$\frac{\Theta \vdash P \blacktriangleright \mathscr{R};\mathscr{N};\mathscr{B}}{\Theta \vdash c\oplus\langle\{\mathsf{p}_k\}_{k\in K},l\rangle;P \blacktriangleright \{\ell(c)\}\cup\mathscr{R};\mathscr{N};\mathscr{B}}\;\{\textsc{Sel}\}$$

$$\frac{\Theta \vdash P_i \blacktriangleright \mathscr{R}_i;\mathscr{N}_i;\mathscr{B}_i \quad \forall i\in I}{\Theta \vdash c\&(\mathsf{p},\{l_i:P_i\}_{i\in I}) \blacktriangleright \mathsf{pre}(\ell(c),\biguplus_{i\in I}\mathscr{R}_i);\bigcup_{i\in I}\mathscr{N}_i;\bigcup_{i\in I}\mathscr{B}_i}\;\{\textsc{Branch}\}$$

Table 15. Interaction typing rules I

18. *If* $\Gamma \vdash \mathsf{def}\,X(x,y)=P\ \mathsf{in}\ Q\triangleright\Delta$*, then* $\Gamma,X:S\,T,x:S\vdash P\triangleright\{y:T\}$ *and* $\Gamma,X:S\,T\vdash Q\triangleright\Delta$.

Lemma 2 (Inversion Lemma for Processes).

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

$$\frac{\Theta \vdash P \blacktriangleright \mathscr{R};\mathscr{N};\mathscr{B} \quad \Theta \vdash Q \blacktriangleright \mathscr{R}';\mathscr{N}';\mathscr{B}'}{\Theta \vdash P \mid Q \blacktriangleright \mathscr{R}\uplus\mathscr{R}';\, \mathscr{N}\cup\mathscr{N}';\, \mathscr{B}\cup\mathscr{B}'}\{\textsc{Conc}\} \qquad \frac{\Theta \vdash P \blacktriangleright \mathscr{R};\mathscr{N};\mathscr{B} \quad a \notin \mathscr{R}\cup\mathscr{N}}{\Theta \vdash (\nu a)P \blacktriangleright \mathscr{R};\, \mathscr{N};\, \mathscr{B}\setminus a}\{\textsc{NRes}\}$$

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

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

$$\frac{\Theta \vdash P \blacktriangleright \mathscr{R};\, \mathscr{N};\, \mathscr{B} \quad \Theta \vdash Q \blacktriangleright \mathscr{R}';\, \mathscr{N}';\, \mathscr{B}'}{\Theta \vdash \mathsf{if}\, e \,\mathsf{then}\, P \,\mathsf{else}\, Q \blacktriangleright \mathscr{R}\uplus\mathscr{R}';\, \mathscr{N}\cup\mathscr{N}';\, \mathscr{B}\cup\mathscr{B}'}\{\textsc{If}\}$$

$$\frac{}{\Theta \vdash \mathbf{0} \blacktriangleright \emptyset;\, \emptyset;\, \emptyset}\{\textsc{Inact}\} \qquad \frac{}{\Theta \vdash s : \varnothing \blacktriangleright \emptyset;\, \emptyset;\, \emptyset}\{\textsc{QInit}\}$$

$$\frac{\Theta \vdash s : h \blacktriangleright \mathscr{R};\, \emptyset;\, \mathscr{B}}{\Theta \vdash s : h \cdot (\mathsf{q}, \{\mathsf{p}_k\}_{k\in K}, v) \blacktriangleright \mathscr{R};\, \emptyset;\, \mathscr{B}\bar{\cup}\{v\}}\{\textsc{QAddval}\}$$

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

$$\frac{\Theta \vdash s : h \blacktriangleright \mathscr{R};\, \emptyset;\, \mathscr{B}}{\Theta \vdash s : h \cdot (\mathsf{q}, \{\mathsf{p}_k\}_{k\in K}, l) \blacktriangleright \mathscr{R};\, \emptyset;\, \mathscr{B}}\{\textsc{QSel}\} \qquad \frac{\Theta \vdash P \blacktriangleright \mathscr{R};\, \mathscr{N};\, \mathscr{B}}{\Theta \vdash (\nu s)P \blacktriangleright \mathscr{R}\setminus s;\, \mathscr{N};\, \mathscr{B}}\{\textsc{SRes}\}$$

Table 16. Interaction typing rules II

Lemma 3. *1. If* $\Gamma \vdash_{\{s\}} s : (\mathsf{q}, \{\mathsf{p}_k\}_{k\in K}, v) \cdot h \triangleright \Delta$, *then* $\Delta = \{s[\mathsf{q}] : !\langle \{\mathsf{p}_k\}_{k\in K}, S\rangle\} * \Delta'$ *and* $\Gamma \vdash_{\{s\}} s : h \triangleright \Delta'$ *and* $\Gamma \vdash v : S$.

2. If $\Gamma \vdash_{\{s\}} s : (\mathsf{q}, \mathsf{p}, s'[\mathsf{p}']) \cdot h \triangleright \Delta$, *then* $\Delta = \{s[\mathsf{q}] : !\langle \mathsf{p}, T'\rangle\} * \Delta'$ *and* $\Gamma \vdash_{\{s\}} s : h \triangleright \Delta'$ *and* $s'[\mathsf{p}'] : T' \in \Delta$.

3. If $\Gamma \vdash_{\{s\}} s : (\mathsf{q}, \{\mathsf{p}_k\}_{k\in K}, l) \cdot h \triangleright \Delta$, *then* $\Delta = \{s[\mathsf{q}] : \oplus\langle \{\mathsf{p}_k\}_{k\in K}, l\rangle\} * \Delta'$ *and* $\Gamma \vdash_{\{s\}} s : h \triangleright \Delta'$.

Theorem 3 (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.

Case 1 $(P \mid \mathbf{0} \equiv P)$*.* First we assume $\Gamma \vdash_\Sigma P \triangleright \Delta$. By $\Gamma \vdash_\emptyset \mathbf{0} \triangleright \emptyset$ and by applying $\lfloor \text{GPar} \rfloor$ 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 2(6) we obtain: $\Gamma \vdash_{\Sigma'} P \triangleright \Delta_1$, $\Gamma \vdash_{\Sigma''} \mathbf{0} \triangleright \Delta_2$ where $\Delta = \Delta_1 * \Delta_2$, $\Sigma = \Sigma' \cup \Sigma''$ and $\Sigma' \cap \Sigma'' = \emptyset$. Using 2(1) we get $\Sigma'' = \emptyset$, which implies $\Sigma = \Sigma'$, and $\Gamma \vdash \mathbf{0} \triangleright \Delta_2$. Using 1(15) we get Δ_2 end only and we conclude $\Gamma \vdash_\Sigma P \triangleright \Delta_1 * \Delta_2$ by applying $\lfloor \text{QWeak} \rfloor$.

Case 2 $(P \mid Q \equiv Q \mid P)$*.* By the symmetry of the rule we have only to show one direction. Suppose $\Gamma \vdash_\Sigma P \mid Q \triangleright \Delta$. Using 2(6) we obtain $\Gamma \vdash_{\Sigma'} P \triangleright \Delta_1$, $\Gamma \vdash_{\Sigma''} Q \triangleright \Delta_2$ where $\Delta = \Delta_1 * \Delta_2$, $\Sigma = \Sigma' \cup \Sigma''$ and $\Sigma' \cap \Sigma'' = \emptyset$. Using $\lfloor \text{GPar} \rfloor$ 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.

Case 3 *($P \mid (Q \mid R) \equiv (P \mid Q) \mid R$).* Suppose $\Gamma \vdash_\Sigma P \mid (Q \mid R) \triangleright \Delta$. Using 2(6) we obtain $\Gamma \vdash_{\Sigma'} P \triangleright \Delta_1$, $\Gamma \vdash_{\Sigma''} Q \mid R \triangleright \Delta_2$ where $\Delta = \Delta_1 * \Delta_2$, $\Sigma = \Sigma' \cup \Sigma''$ and $\Sigma' \cap \Sigma'' = \emptyset$. Using 2(6) we obtain $\Gamma \vdash_{\Sigma''_1} Q \triangleright \Delta_{21}$, $\Gamma \vdash_{\Sigma''_2} R \triangleright \Delta_{22}$ where $\Delta_2 = \Delta_{21} * \Delta_{22}$, $\Sigma'' = \Sigma''_1 \cap \Sigma''_2$ and $\Sigma''_1 \cup \Sigma''_2 = \emptyset$. Using [GPar] we get $\Gamma \vdash_{\Sigma' \cup \Sigma''_1} P \mid Q \triangleright \Delta_1 * \Delta_{21}$. Using $\lfloor$GPAR$\rfloor$ 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.

Case 4 *($s : (\mathrm{q}, \emptyset, v) \cdot h \equiv s : h$).* Using 3(1) we obtain $\Gamma \vdash_s (\mathrm{q}, \emptyset, v) \cdot h \triangleright \Delta$, where $\Delta = \{s[\mathrm{q}] : !\langle \emptyset, S \rangle\} * \Delta'$ and $\Gamma \vdash_{\{s\}} s : h \triangleright \Delta'$ and $\Gamma \vdash v : S$. Using the equivalence relation on Δ we get $\{s[\mathrm{q}] : !\langle \emptyset, S \rangle\} * \Delta' \approx \Delta'$.

Lemma 4 (Substitution lemma).

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

Proof. Standard induction on P.

Theorem 4 (Type Preservation under Reduction). *If* $\Gamma \vdash_\Sigma P \triangleright \Delta$ *and* $P \longrightarrow^* P'$, *then* $\Gamma \vdash_\Sigma P' \triangleright \Delta'$ *for some* Δ' *such that* $\Delta \Rightarrow \Delta'$. *Moreover* Δ *coherent implies* Δ' *coherent and* Δ *closed implies* Δ' *closed.*

Proof. - Case [Link]

$$\bar{a}[n](y_1).P_1 \mid a[2](y_2).P_2 \mid \ldots \mid a[n](y_n).P_n \longrightarrow (\nu s)(P_1\{s[1]/y_1\}|...|P_n\{s[n]/y_n\}|s : \varnothing).$$

Assume $\Gamma \vdash_\Sigma \bar{a}[n](y_1).P_1 \mid a[2](y_2).P_2 \mid \ldots \mid a[n](y_n).P_n \triangleright \Delta$, then $\Sigma = \emptyset$ and

$$\Gamma \vdash \bar{a}[n](y_1).P_1 \mid a[2](y_2).P_2 \mid \ldots \mid a[n](y_n).P_n \triangleright \Delta$$

by Lemma 2(1). Using Lemma 1(13) more times we have

$$\Gamma \vdash \bar{a}[n](y_1).P_1 \triangleright \Delta_1 \quad (1)$$

$$\Gamma \vdash a[i](y_i).P_i \triangleright \Delta_i \quad (2 \leq i \leq n) \quad (2)$$

where $\Delta = \bigcup_{i=1}^n \Delta_i$. Using Lemma 1(5) on (1) we have

$$\Gamma \vdash a : \langle G \rangle$$

$$\Gamma \vdash P_1 \triangleright \Delta_1, y_1 : G \upharpoonright 1 \quad (3)$$

and $\mathrm{pn}(G) \leq n$. Using Lemma 1(6) on (2) we have

$$\Gamma \vdash a : \langle G \rangle$$

$$\Gamma \vdash P_i \triangleright \Delta_i, y_i : G \upharpoonright i \quad (2 \leq i \leq n). \quad (4)$$

Using Lemma 4(2) on (3) and (4)

$$\Gamma \vdash_{\{s\}} P_i\{s[i]/y_i\} \triangleright \Delta_i, s[i] : G \upharpoonright i \quad (1 \leq i \leq n). \quad (5)$$

Using $\lfloor$CONC$\rfloor$ more times on (5) we have

$$\Gamma \vdash P_1\{s[1]/y_1\}|...|P_n\{s[n]/y_n\} \triangleright \bigcup_{i=1}^{n}(\Delta_i, s[i] : G \upharpoonright i). \quad (6)$$

Note that

$$\bigcup_{i=1}^{n}(\Delta_i, s[i] : G \upharpoonright i) = \Delta, s[1] : G \upharpoonright 1, \ldots, s[n] : G \upharpoonright n$$

Using $\lfloor \text{GINIT} \rfloor$, $\lfloor \text{QINIT} \rfloor$ and $\lfloor \text{GPAR} \rfloor$ on (6) we have

$$\Gamma \vdash_{\{s\}} P_1\{s[1]/y_1\}|...|P_n\{s[n]/y_n\} \mid s : \varnothing \triangleright \Delta, s[1] : G \upharpoonright 1, \ldots, s[n] : G \upharpoonright n. \tag{7}$$

Using $\lfloor \text{QSCOPE} \rfloor$ on (7) we have

$$\Gamma \vdash_{\emptyset} (\nu s)(P_1\{s[1]/y_1\}|...|P_n\{s[n]/y_n\} \mid s : \varnothing) \triangleright \Delta \tag{8}$$

since

$$(\Delta, s[1] : G \upharpoonright 1, \ldots, s[n] : G \upharpoonright n) \setminus s = \Delta.$$

- Case [Send]

$$s[\mathrm{p}]!\langle\{\mathrm{p}_k\}_{k\in K}, e\rangle; P \mid s : h \longrightarrow P \mid s : h \cdot (\mathrm{p}, \{\mathrm{p}_k\}_{k\in K}, v) \;\; (e \downarrow v).$$

Assume

$$\Gamma \vdash_{\Sigma} s[\mathrm{p}]!\langle\{\mathrm{p}_k\}_{k\in K}, e\rangle; P \mid s : h \triangleright \Delta.$$

Using Lemma 2(1) and 2(6) we have $\Sigma = \{s\}$ and

$$\Gamma \vdash s[\mathrm{p}]!\langle\{\mathrm{p}_k\}_{k\in K}, e\rangle; P \triangleright \Delta_1 \tag{9}$$

$$\Gamma \vdash_{\{s\}} s : h \triangleright \Delta_2 \tag{10}$$

where $\Delta = \Delta_2 * \Delta_1$. Using 1(7) on (9) we have

$$\Delta_1 = \Delta_1', s[\mathrm{p}] : !\langle\{\mathrm{p}_k\}_{k\in K}, S\rangle; T$$

$$\Gamma \vdash e : S \tag{11}$$

$$\Gamma \vdash P \triangleright \Delta_1', s[\mathrm{p}] : T. \tag{12}$$

Using $\lfloor \text{QADDVAL} \rfloor$ on (10) and (11) we have

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

Using $\lfloor \text{GINIT} \rfloor$ on (12) and then $\lfloor \text{GPAR} \rfloor$ on (12), (13) we get

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

Note that

$$(\Delta_2; \{s[\mathrm{p}] : !\langle\{\mathrm{p}_k\}_{k\in K}, S\rangle\}) * (\Delta_1', s[\mathrm{p}] : T) \;\Rightarrow \Delta_2 * (\Delta_1', s[\mathrm{p}] : !\langle\{\mathrm{p}_k\}_{k\in K}, S\rangle; T).$$

- Case [Recv]

$$s[\mathrm{p}_j]?(\mathrm{q}, x); P \mid s : (\mathrm{q}, \{\mathrm{p}_k\}_{k\in K}, v) \cdot h \longrightarrow P\{v/x\} \mid s : (\mathrm{q}, \{\mathrm{p}_k\}_{k\in K, k\neq j}, v) \cdot h \quad (j \in K).$$

Assume

$$\Gamma \vdash_{\Sigma} s[\mathrm{p}_j]?(\mathrm{q}, x); P \mid s : (\mathrm{q}, \{\mathrm{p}_k\}_{k\in K}, v) \cdot h \triangleright \Delta.$$

By 2(1) and 2(6) we have $\Sigma = \emptyset$ and

$$\Gamma \vdash s[\mathrm{p}_j]?(\mathrm{q}, x); P \triangleright \Delta_1 \tag{14}$$

$$\Gamma \vdash_{\{s\}} s : (\mathrm{q}, \{\mathrm{p}_k\}_{k\in K}, v) \cdot h \triangleright \Delta_2 \tag{15}$$

where

$$\Delta = \Delta_2 * \Delta_1.$$

Using Lemma 1(8) on (14) we have

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

$$\Gamma, x : S \vdash P \triangleright \Delta_1', s[\mathrm{p}_j] : T \tag{16}$$

Thanks to Lemma 4(1) from (16) we get $\Gamma \vdash P\{v/x\} \triangleright \Delta_1', s[\mathrm{p}_j] : T$, which implies by rule $\lfloor$GINIT$\rfloor$

$$\Gamma \vdash_{\emptyset} P\{v/x\} \triangleright \Delta_1', s[\mathrm{p}_j] : T. \tag{17}$$

Using Lemma 3(1) on (15) we have

$$\begin{aligned} \Delta_2 &= \{s[\mathrm{q}] : !\langle\{\mathrm{p}_k\}_{k\in K}, S\rangle\} * \Delta_2' \\ \Gamma &\vdash_{\{s\}} s : h \triangleright \Delta_2' \\ \Gamma &\vdash v : S. \end{aligned} \tag{18}$$

Applying rule $\lfloor$QADDVAL$\rfloor$ on (18) we get

$$\Gamma \vdash_{\{s\}} (\mathrm{q}, \{\mathrm{p}_k\}_{k\in K\setminus j}, v) \cdot h \triangleright \{s[\mathrm{q}] : !\langle\{\mathrm{p}_k\}_{k\in K\setminus j}, S\rangle\} * \Delta_2' \tag{19}$$

Using rule $\lfloor$GPAR$\rfloor$ on (17) and (19) we get

$$\Gamma \vdash_{\{s\}} P\{v/x\} \mid (\mathrm{q}, \{\mathrm{p}_k\}_{k\in K\setminus j}, v) \cdot h \triangleright (\{s[\mathrm{q}] : !\langle\{\mathrm{p}_k\}_{k\in K\setminus j}, S\rangle\} * \Delta_2') * (\Delta_1', s[\mathrm{p}_j] : T).$$

Note that

$$(\{s[\mathrm{q}] : !\langle\{\mathrm{p}_k\}_{k\in K}, S\rangle\} * \Delta_2') * (\Delta_1', s[\mathrm{p}_j] : ?(\mathrm{q}, S); T) \Rightarrow (\{s[\mathrm{q}] : !\langle\{\mathrm{p}_k\}_{k\in K\setminus j}, S\rangle\} * \Delta_2') * (\Delta_1', s[\mathrm{p}_j] : T).$$

- Case [Label]

$$s[\mathrm{p}] \oplus \langle\{\mathrm{p}_k\}_{k\in K}, l\rangle; P \mid s : h \longrightarrow P \mid s : h \cdot (\mathrm{p}, \{\mathrm{p}_k\}_{k\in K}, l)$$

Assume

$$\Gamma \vdash_{\Sigma} s[\mathrm{p}] \oplus \langle\{\mathrm{p}_k\}_{k\in K}, l\rangle; P \mid s : h \triangleright \Delta$$

Using Lemma 2(1) and 2(6) we have $\Sigma = \{s\}$ and

$$\Gamma \vdash s[\mathrm{p}] \oplus \langle\{\mathrm{p}_k\}_{k\in K}, l\rangle; P \triangleright \Delta_1 \tag{20}$$

$$\Gamma \vdash_{\{s\}} s : h \triangleright \Delta_2 \tag{21}$$

where

$$\Delta = \Delta_2 * \Delta_1$$

Using Lemma 1(11) on (20) we have for $l = l_j$ $(j \in I)$:

$$\begin{aligned} \Delta_1 &= \Delta_1', s[\mathrm{p}] : \oplus\langle\{\mathrm{p}_k\}_{k\in K}, \{l_i : T_i\}_{i\in I}\rangle \\ \Gamma &\vdash P \triangleright \Delta_1', T_j. \end{aligned} \tag{22}$$

Using rule $\lfloor$QSEL$\rfloor$ on (21) we have

$$\Gamma \vdash_{\{s\}} s : h \cdot (\mathrm{p}, \{\mathrm{p}_k\}_{k\in K}, l) \triangleright \Delta_2; \{s[\mathrm{p}] : \oplus\langle\{\mathrm{p}_k\}_{k\in K}, l\rangle\}. \tag{23}$$

Using $\lfloor$GPAR$\rfloor$ on (22) and (23) we have

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

Note that

$$\Delta_2 * (\Delta_1', s[\mathrm{p}] : \oplus\langle\{\mathrm{p}_k\}_{k\in K}, \{l_i : T_i\}_{i\in I}\rangle) \Rightarrow (\Delta_2; \{s[p] : \oplus\langle\{\mathrm{p}_k\}_{k\in K}, l\rangle\}) * (\Delta_1', s[\mathrm{p}] : T_j).$$

- Case [Branch]

$$s[\mathrm{p}_j]\&(\mathrm{q}, \{l_i : P_i\}_{i\in I}) \mid s : (\mathrm{q}, \{\mathrm{p}_k\}_{k\in K}, l_{i_0}) \cdot h \longrightarrow P_{i_0} \mid s : (\mathrm{q}, \{\mathrm{p}_k\}_{k\in K\setminus j}, l_{i_0}) \cdot h \quad (j \in K)$$
$(i_0 \in I)$

Assume

$$\Gamma \vdash_{\Sigma} s[\mathrm{p}_j]\&(\mathrm{q}, \{l_i : P_i\}_{i\in I}) \mid s : (\mathrm{q}, \{\mathrm{p}_k\}_{k\in K}, l_{i_0}) \cdot h \triangleright \Delta.$$

Using Lemma 2(1) and 2(6) we have $\Sigma = \{s\}$ and

$$\Gamma \vdash s[\mathrm{p}_j]\&(\mathrm{q}, \{l_i : P_i\}_{i\in I}) \triangleright \Delta_1 \tag{24}$$

$$\Gamma \vdash_{\{s\}} s : (\mathrm{q}, \{\mathrm{p}_k\}_{k\in K}, l_{i_0}) \cdot h \triangleright \Delta_2 \tag{25}$$

where

$$\Delta = \Delta_2 * \Delta_1 = \Delta_2 * \Delta_1.$$

Using Lemma 1(12) on (24) we have

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

$$\Gamma \vdash P_i \triangleright \Delta_1', s[\mathrm{p}_j] : T_i \quad \forall i \in I. \tag{26}$$

Using Lemma 3(3) on (25) we have

$$\Delta_2 = \{s[\mathrm{q}] : \oplus(\{\mathrm{p}_k\}_{k\in K}, l_{i'})\} * \Delta_2'$$

$$\Gamma \vdash_{\{s\}} s : h \triangleright \Delta_2'. \tag{27}$$

Using $\lfloor$QSEL$\rfloor$ on (27) we get

$$\Gamma \vdash_{\{s\}} s : (\mathrm{q}, \{\mathrm{p}_k\}_{k\in K\setminus j}, l_{i_0}) \cdot h \triangleright \{s[\mathrm{q}] : \oplus(\{\mathrm{p}_k\}_{k\in K\setminus j}, l_{i'})\} * \Delta_2'. \tag{28}$$

Using $\lfloor$GPAR$\rfloor$ on (26) and (28) we have

$$\Gamma \vdash_{\{s\}} P_{i_0} \mid s : (\mathrm{q}, \{\mathrm{p}_k\}_{k\in K\setminus j}, l_{i_0}) \cdot h \triangleright (\{s[\mathrm{q}] : \oplus(\{\mathrm{p}_k\}_{k\in K\setminus j}, l_{i'})\} * \Delta_2') * (\Delta_1', s[\mathrm{p}_j] : T_{i_0}).$$

Note that

$$(\{s[\mathrm{q}] : \oplus(\{\mathrm{p}_k\}_{k\in K}, l_{i'})\} * \Delta_2') * (\Delta_1', s[\mathrm{p}_j] : \&(\mathrm{q}, \{l_i : T_i\}_{i\in I})) \Rightarrow$$
$$(\{s[\mathrm{q}] : \oplus(\{\mathrm{p}_k\}_{k\in K\setminus j}, l_{i'})\} * \Delta_2') * (\Delta_1', s[\mathrm{p}_j] : T_{i_0}).$$

F.2 Proof of the Progress Theorem

In the following definitions and proofs we assume that all considered processes are well typed with the communication type system of Section 3.

Lemma 5. *If* $\Theta \vdash s : h \cdot m \blacktriangleright \mathscr{R}; \emptyset; \mathscr{B}$ *then* $\Theta \vdash s : m \cdot h \blacktriangleright \mathscr{R}; \emptyset; \mathscr{B}$.

Proof. By induction on h.

Lemma 6 (Substitution Lemma). *Let* $\Theta \vdash P \blacktriangleright \mathscr{R}; \mathscr{N}; \mathscr{B}$.

1. *Let* $v \notin \mathscr{R}$. *Then* $\Theta \vdash P\{v/x\} \blacktriangleright \mathscr{R}; \mathscr{N}; \mathscr{B}'$ *where* $\mathscr{B}' = \mathscr{B} \dot\cup \{v\}$*;*
2. $\Theta \vdash P\{s[\mathrm{p}]/y\} \blacktriangleright \mathscr{R}\{s/y\}; \mathscr{N}; \mathscr{B}$.

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

1. By induction on P. The only interesting case is when v is a service name a, thus, $P \equiv \bar{x}[n](y).P'$ or $P \equiv x[n](y).P'$ and the last applied rules are $\{$MCASTB$\}$ or $\{$MACCB$\}$, respectively. Let us consider $P \equiv \bar{x}[n](y).P'$ (the other case is similar). From $\{$MCASTB$\}$ we have that $\Theta \vdash P' \blacktriangleright \mathscr{R}'; \mathscr{N}; \mathscr{B}$ such that $\mathsf{cf}(\mathscr{R}' \setminus\setminus y)$ and $\mathscr{R} = \mathscr{R}' \setminus\setminus y$. Now, $P\{a/x\} = \bar{a}[n](y).P'$. Since, by hypothesis, $\mathsf{cf}(\mathscr{R}' \setminus\setminus y)$, thus we can apply $\{$MCASTB$\}$, obtaining $\Theta \vdash \bar{a}[n](y).P' \blacktriangleright \mathscr{R}; \mathscr{N}; \mathscr{B} \cup \{a\}$. Note that this judgements is coherent since by hypothesis $a \notin \mathscr{R}$.
2. Easily follows from the definition of $\ell(c)$.

Theorem 5 (Type Preservation under Equivalence). *If P is well typed and* $\Theta \vdash P \blacktriangleright \mathscr{R}; \mathscr{N}; \mathscr{B}$ *and* $P \equiv P'$, *then* $\Theta \vdash P' \blacktriangleright \mathscr{R}; \mathscr{N}; \mathscr{B}$.

Proof. Standard induction on $\equiv$.

Theorem 6 (Type Preservation under Reduction). *If P is well typed and $\Theta \vdash P \blacktriangleright \mathscr{R}\,;\mathscr{N}\,;\mathscr{B}$ and $P \longrightarrow^* P'$, then $\Theta \vdash P' \blacktriangleright \mathscr{R}'\,;\mathscr{N}'\,;\mathscr{B}'$ for some $\mathscr{R}' \subseteq \mathscr{R}$, $\mathscr{N}' \subseteq \mathscr{N}$ and $\mathscr{B}' \subseteq \mathscr{B}$.*

Proof. By induction on $\longrightarrow$ and by cases on the last applied rule.

- [Link]. By hypothesis

$$\Theta \vdash \bar{a}[n](y_1).P_1 \mid a[2](y_2).P_2 \mid \ldots \mid a[n](y_n).P_n \blacktriangleright \mathscr{R}\,;\mathscr{N}\,;\mathscr{B}.$$

  This judgement is obtained by applying rule $\{\textsc{Conc}\}$ to the subprocesses $\bar{a}[n](y_1).P_1$, $a[2](y_2).P_2,\ldots,a[n](y_n).P_n$. Then we have:
  - $\Theta \vdash \bar{a}[n](y_1).P_1 \blacktriangleright \mathscr{R}_1\,;\mathscr{N}_1\,;\mathscr{B}_1$
  - $\Theta \vdash a[2](y_2).P_2 \blacktriangleright \mathscr{R}_2\,;\mathscr{N}_2\,;\mathscr{B}_2$
  - $\ldots$
  - $\Theta \vdash a[n](y_n).P_n \blacktriangleright \mathscr{R}_n\,;\mathscr{N}_n\,;\mathscr{B}_n$

  where $\mathscr{R} = \biguplus_{1\le i\le n}\mathscr{R}_i$ and $\mathscr{N} = \bigcup_{1\le i\le n}\mathscr{N}_i$ and $\mathscr{B} = \bigcup_{1\le i\le n}\mathscr{B}_i$. Point 2. of the the coherence condition (see page 11) implies that the rules $\{\text{MC}\textsc{ast}\}$, $\{\text{MA}\textsc{cc}\}$ cannot be used for the same session name with the rules $\{\text{MC}\textsc{astn}\}$, $\{\text{MA}\textsc{ccn}\}$, $\{\text{MC}\textsc{astb}\}$, $\{\text{MA}\textsc{ccb}\}$.
  We consider the case in which P_1 has been typed with rule $\{\text{MC}\textsc{astn}\}$ or $\{\text{MC}\textsc{astb}\}$ and each P_{p} $(2 \le \mathrm{p} \le n)$ with $\{\text{MA}\textsc{ccn}\}$ or $\{\text{MA}\textsc{ccb}\}$.
  Then for each i $(1 \le i \le n)$ we must have $\Theta \vdash P_i \blacktriangleright \mathscr{R}'_i\,;\mathscr{N}_i'\,;\mathscr{B}'_i$ such that $\mathscr{R}_i = \mathscr{R}'_i \setminus\setminus y_i$, $\mathscr{N}_i' \subseteq \mathscr{N}_i$, $\mathscr{B}'_i \subseteq \mathscr{B}_i$ (y_i is minimal in $\mathscr{R}'_i$). By Lemma 6(2) we have

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

  By using $\{\textsc{Conc}\}$ (and $\{\text{QI}\textsc{nit}\}$) we have

$$\Theta \vdash P_1\{s[1]/y_1\}|...|P_n\{s[n]/y_n\}|s:\varnothing \blacktriangleright \mathscr{R}'\,;\mathscr{N}'\,;\mathscr{B}'$$

  where $\mathscr{R}' = \biguplus \mathscr{R}'_i\{s/y_i\}$,$\mathscr{N}' = \bigcup \mathscr{N}_i'$ and $\mathscr{B}' = \bigcup \mathscr{B}'_i$. Note that this judgement is coherent since s must be minimal in $\mathscr{R}'$ and $\mathscr{R}' \cap (\mathscr{N}' \cup \mathscr{B}') = \emptyset$.
  By using $\{\text{SR}\textsc{es}\}$,

$$\Theta \vdash (\nu s)(P_1\{s[1]/y_1\}|...|P_n\{s[n]/y_n\}|s:\varnothing) \blacktriangleright \mathscr{R}' \setminus s\,;\mathscr{N}'\,;\mathscr{B}'$$

  Finally it is easy to see that $\mathscr{R}' \setminus s = \mathscr{R}$ (by the minimality of the y_i in $\mathscr{R}'_i$ and of s in $\mathscr{R}'$), $\mathscr{N}' \subseteq \mathscr{N}$ and $\mathscr{B}' \subseteq \mathscr{B}$.

- [Send]. By hypothesis, $\Theta \vdash s[\mathrm{p}]!\langle\{\mathrm{p}_k\}_{k\in K},e\rangle;P \mid s:h \blacktriangleright \mathscr{R}\,;\mathscr{N}\,;\mathscr{B}$, which is obtained by applying rule $\{\textsc{Conc}\}$. Thus,

$$\Theta \vdash s[\mathrm{p}]!\langle\{\mathrm{p}_k\}_{k\in K},e\rangle;P \blacktriangleright \mathscr{R}_1\,;\mathscr{N}\,;\mathscr{B}_1 \qquad \Theta \vdash s:h \blacktriangleright \mathscr{R}_2\,;\emptyset;\mathscr{B}_2$$

  where $\mathscr{R} = \mathscr{R}_1 \uplus \mathscr{R}_2$ and $\mathscr{B} = \mathscr{B}_1 \cup \mathscr{B}_2$. The first judgement can only be obtained by $\{\textsc{Send}\}$, i.e., $\Theta \vdash P \blacktriangleright \mathscr{R}'_1\,;\mathscr{N}\,;\mathscr{B}'_1$ such that $\mathscr{R}_1 = \{s\} \cup \mathscr{R}'_1$ and $\mathscr{B}_1 = \mathscr{B}'_1 \bar{\cup} \{v\}$.
  By using rules $\{\text{QA}\textsc{ddval}\}$ and $\{\textsc{Conc}\}$ we obtain

$$\Theta \vdash P \mid s:h\cdot(\mathrm{p},\{\mathrm{p}_k\}_{k\in K},v) \blacktriangleright \mathscr{R}'_1 \uplus \mathscr{R}_2\,;\mathscr{N}\,;\mathscr{B}'_1 \cup (\mathscr{B}_2 \bar{\cup} \{v\}).$$

  Now note that $\mathscr{R}'_1 \uplus \mathscr{R}_2 \subseteq \mathscr{R}$ and $\mathscr{B}'_1 \cup (\mathscr{B}_2 \bar{\cup} \{v\}) = \mathscr{B}$.

- [Deleg]. Proceed as in the previous case, thus obtaining

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

where $\mathscr{R} = \mathscr{R}_1 \uplus \mathscr{R}_2$ and $\mathscr{B} = \mathscr{B}_1 \cup \mathscr{B}_2$. By inverting rule $\{\textsc{Deleg}\}$ we obtain $\Theta \vdash P \blacktriangleright \mathscr{R}_1' \,;\, \mathscr{N} \,;\, \mathscr{B}_1$ where $\mathscr{R}_1 = \{s, s', s \prec s'\} \uplus \mathscr{R}_1'$. By using rules $\{\textsc{QAddSess}\}$ and $\{\textsc{Conc}\}$ we have

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

- [Label]. Similar to [Send] but simpler (using rule $\{\textsc{QSel}\}$ instead of $\{\textsc{QAddVal}\}$).

- [Recv]. By hypothesis, $\Theta \vdash s[\mathsf{p}_j]?(\mathsf{q}, x);P \mid s : (\mathsf{q}, \{\mathsf{p}_k\}_{k \in K}, v) \cdot h \blacktriangleright \mathscr{R} ; \mathscr{N} ; \mathscr{B}$. Proceed as in the case of rule [Send], thus obtaining

$$\Theta \vdash s[\mathsf{p}_j]?(\mathsf{q}, x);P \blacktriangleright \mathscr{R}_1 \,;\, \mathscr{N} \,;\, \mathscr{B}_1 \qquad \Theta \vdash s : (\mathsf{q}, \{\mathsf{p}_k\}_{k \in K}, v) \cdot h \blacktriangleright \mathscr{R}_2 \,;\, \emptyset ;\, \mathscr{B}_2$$

where $\mathscr{R} = \mathscr{R}_1 \uplus \mathscr{R}_2$ and $\mathscr{B} = \mathscr{B}_1 \cup \mathscr{B}_2$. By inverting rule $\{\textsc{Recv}\}$ we obtain $\Theta \vdash P \blacktriangleright \mathscr{R}_1' \,;\, \mathscr{N} \,;\, \mathscr{B}_1$ where $\mathscr{R}_1 = \mathsf{pre}(s, \mathscr{R}_1')$. By Lemma 6(1) we obtain $\Theta \vdash P\{v/x\} \blacktriangleright \mathscr{R}_1' \,;\, \mathscr{N} \,;\, \mathscr{B}_1 \bar{\cup} \{v\}$. Moreover we have $\Theta \vdash s : h \blacktriangleright \mathscr{R}_2 \,;\, \emptyset ;\, \mathscr{B}_2'$ where $\mathscr{B}_2 = \mathscr{B}_2' \bar{\cup} \{v\}$. Applying $\{\textsc{Conc}\}$ we get

$$(1)\;\; \Theta \vdash P\{v/x\} \mid s : (\mathsf{q}, \{\mathsf{p}_k\}_{k \in K \setminus j}, v) \cdot h \blacktriangleright \mathscr{R}_1' \uplus \mathscr{R}_2 \,;\, \mathscr{N} \,;\, \mathscr{B}_1 \bar{\cup} \{v\} \cup \mathscr{B}_2'.$$

and note that $\mathscr{R}_1' \uplus \mathscr{R}_2 \subseteq \mathscr{R}_1 \uplus \mathscr{R}_2$ and $\mathscr{B}_1 \bar{\cup} \{v\} \cup \mathscr{B}_2' = \mathscr{B}$.
If $v = a$ is a service name, then $a \in \mathscr{B}_2$ implies that $a \notin \mathscr{R}_1 \uplus \mathscr{R}_2$ and so $a \notin \mathscr{R}_1' \uplus \mathscr{R}_2$. Then (1) is coherent.

- [Srec]. By hypothesis, $\Theta \vdash s[\mathsf{p}]?((\mathsf{q}, y));P \mid s : (\mathsf{q}, \mathsf{p}, s'[\mathsf{p}']) \cdot h \blacktriangleright \mathscr{R} ; \mathscr{N} ; \mathscr{B}$. Proceeding as before,

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

where $\mathscr{R} = \{s\} \uplus \mathscr{R}_2$ and $\mathscr{B} = \mathscr{B}_1 \cup \mathscr{B}_2$. In particular (inverting rule $\{\textsc{SRec}\}$) we have $\Theta \vdash P \blacktriangleright \mathscr{R}_1' \,;\, \mathscr{N} \,;\, \mathscr{B}_1$ where $\mathscr{R}_1' \subseteq \{s, y, s \prec y\}$. Moreover, by $\{\textsc{QAddSess}\}$ (and Lemma 5) we have that $\Theta \vdash s : h \blacktriangleright \mathscr{R}_2' \,;\, \emptyset ;\, \mathscr{B}_2$ such that $\mathscr{R}_2 = \{s, s', s \prec s'\} \uplus \mathscr{R}_2'$. By Lemma 6(2), we have $\Theta \vdash P\{s'[\mathsf{p}']/y\} \blacktriangleright \mathscr{R}_1'' \,;\, \mathscr{N} \,;\, \mathscr{B}_1$ where $\mathscr{R}_1'' \subseteq \{s, s', s \prec s'\}$. By applying rule $\{\textsc{Conc}\}$ we obtain

$$\Theta \vdash P\{s'[\mathsf{p}']/y\} \mid s : h \blacktriangleright \mathscr{R}_1'' \uplus \mathscr{R}_2' \,;\, \mathscr{N} \,;\, \mathscr{B}_1 \cup \mathscr{B}_2.$$

Lastly it is easy to see that this statement is coherent and that $\mathscr{R}_1'' \uplus \mathscr{R}_2' \subseteq \mathscr{R}$.

- [Branch]. By hypothesis, $\Theta \vdash s[\mathsf{p}_j]\&(\mathsf{q}, \{l_i : P_i\}_{i \in I}) \mid s : (\mathsf{q}, \{\mathsf{p}_k\}_{k \in K}, l_{i_0}) \cdot h \blacktriangleright \mathscr{R} ; \mathscr{N} ; \mathscr{B}$. By inverting the rules we have
  - $\Theta \vdash P_i \blacktriangleright \mathscr{R}_i ; \mathscr{N}_i ; \mathscr{B}_i \quad \forall i \in I$
  - $\Theta \vdash s : (\mathsf{q}, \{\mathsf{p}_k\}_{k \in K}, l_{i_0}) \cdot h \blacktriangleright \mathscr{R}' ; \emptyset ; \mathscr{B}'$
  - $\mathscr{R} = \mathsf{pre}(s, \biguplus_{i \in I} \mathscr{R}_i) \uplus \mathscr{R}'$, $\mathscr{N} = \bigcup_{i \in I} \mathscr{N}_i$, $\mathscr{B} = \bigcup_{i \in I} \mathscr{B}_i \cup \mathscr{B}'$.

  By applying rule $\{\textsc{Conc}\}$ to the reduced process we obtain

$$\Theta \vdash P_{i_0} \mid s : (\mathsf{q}, \{\mathsf{p}_k\}_{k \in K \setminus j}, l_{i_0}) \cdot h \blacktriangleright \mathscr{R}_{i_0} \uplus \mathscr{R}' \,;\, \mathscr{N}_{i_0} \,;\, \mathscr{B}_{i_0} \cup \mathscr{B}'$$

and the result follows easily.

- [If-T, If-F]. Straightforward.

- [Def]. Let's assume $\Theta \vdash \mathsf{def}\, X(x,y) = P \;\mathsf{in}\; (X\langle e, s[\mathtt{p}]\rangle \mid Q) \blacktriangleright \mathscr{R} ; \mathscr{N} ; \mathscr{B}$. Note that by rule $\lfloor \textsc{Def} \rfloor$ y is the only free channel which can occur P. By inspecting the inference rule, as before, we must have:
  1. $\Theta' = \Theta, X[y] \blacktriangleright \mathscr{R}' ; \mathscr{N}' ; \mathscr{B}'$;
  2. $\Theta' \vdash P \blacktriangleright \mathscr{R}' ; \mathscr{N}' ; \mathscr{B}'$;
  3. $\Theta' \vdash X\langle e, s[\mathtt{p}]\rangle \blacktriangleright \mathscr{R}'\{s/y\} ; \mathscr{N}' ; \mathscr{B}' \bar{\cup} \{e\}$;
  4. $\Theta' \vdash Q \blacktriangleright \mathscr{R}'' ; \mathscr{N}'' ; \mathscr{B}''$;

  where $\mathscr{R} = \mathscr{R}'\{s/y\} \uplus \mathscr{R}''$, $\mathscr{N} = \mathscr{N}' \cup \mathscr{N}''$, $\mathscr{B} = \mathscr{B}' \bar{\cup} \{e\} \cup \mathscr{B}''$.
  By Lemma 6 we have $\Theta' \vdash P\{v/x\}\{s[\mathtt{p}]/y\} \blacktriangleright \mathscr{R}\{s/y\} ; \mathscr{N} ; \mathscr{B}' \bar{\cup} \{v\}$ and then by rule $\{\textsc{Conc}\}$ $\Theta' \vdash (P\{v/x\}\{s[\mathtt{p}]/y\} \mid Q) \blacktriangleright \mathscr{R} ; \mathscr{N} ; \mathscr{B}$ since $e \downarrow v$ implies $\mathscr{B}' \bar{\cup} \{e\} = \mathscr{B}' \bar{\cup} \{v\}$. By rule $\{\textsc{Def}\}$ we conclude $\Theta \vdash \mathsf{def}\, X(x,y) = P \;\mathsf{in}\; (P\{v/x\}\{s[\mathtt{p}]/y\} \mid Q) \blacktriangleright \mathscr{R} ; \mathscr{N} ; \mathscr{B}$.

- [Scop, Pat, Defin, Str]. For the congruence rules the thesis follows from the induction hypothesis.

Lemma 7. *If $\Gamma \vdash_\Sigma P \triangleright \Delta$ and $\Theta \vdash P \blacktriangleright \mathscr{R} ; \mathscr{N} ; \mathscr{B}$, then:*
1. *$s[\mathtt{p}] : T \in \Delta$ and $T \neq \mathsf{end}$ imply $s \in \mathscr{R}$;*
2. *$s \in \mathscr{R}$ implies $\Delta(s[\mathtt{p}]) \neq \mathsf{end}$ for some $\mathtt{p}$.*

Proof. Standard by induction on P.

Lemma 8. *If $\Theta \vdash P \blacktriangleright \mathscr{R} ; \mathscr{N} ; \mathscr{B}$ and $a \notin \mathscr{R} \cup \mathscr{N}$ and $P \equiv \bar{a}[n](y).P'$ or $P \equiv a[\mathtt{p}](y).P'$, then no channel with role occurs in $\mathscr{R}$.*

Proof. The last applied rule must be $\{\textsc{MCastB}\}$ or $\{\textsc{MAccB}\}$ and then we must have $\Theta \vdash P' \blacktriangleright \mathscr{R}' ; \mathscr{N} ; \mathscr{B}$ and $\mathscr{R} = \mathscr{R}' \setminus\!\setminus y$. Note that the condition $\mathsf{cf}(\mathscr{R}' \setminus\!\setminus y)$ prevents channels with roles to occur in $\mathscr{R}'$.

In the following definition we use $C[\;]$ to denote a context with a hole defined in the standard way.

Definition 6 (Precedence).
1. *The channel c* precedes *c' in the process P if one of the following condition holds:*
   - $P = C[c?(\mathtt{q}, x); Q]$ *and c' occurs in Q;*
   - $P = C[c!\langle\langle \mathtt{p}, c'\rangle\rangle; Q]$;
   - $P = C[c?((\mathtt{q}, y)); Q]$ *and c' occurs in Q;*
   - $P = C[c\&(\mathtt{q}, \{l_i : P_i\}_{i\in I})]$ *and c' occurs in P_i for some $i \in I$;*
   - $P = C[s : h \cdot (\mathtt{q}, \mathtt{p}, s'[\mathtt{p}']) \cdot h']$ *and $c = s[\mathtt{p}]$ and $c' = s'[\mathtt{p}']$.*
2. *The channel c* weakly precedes *c' in the process P if either c precedes c' in P or one of the following condition holds:*
   - $P = C[c!\langle \{\mathtt{p}_k\}_{k\in K}, e\rangle; Q]$ *and c' occurs in Q;*
   - $P = C[c!\langle\langle \mathtt{p}, c_0\rangle\rangle; Q]$ *and c' occurs in Q.*

Lemma 9. *If $\Theta \vdash P \blacktriangleright \mathscr{R} ; \mathscr{N} ; \mathscr{B}$ and $s[\mathtt{p}]$ precedes $s'[\mathtt{p}']$ in P and $s \neq s'$, then $s \prec s' \in \mathscr{R}$.*

Proof. By induction on P.

Lemma 10. *Let P be initial and $P \longrightarrow^* P'$.*

1. *If* $s[\mathsf{p}]$ *weakly precedes* $s'[\mathsf{q}]$ *in* P'*, then either* $s \neq s'$ *or* $\mathsf{p} = \mathsf{q}$*;*
2. *If* $P \equiv P' \mid s : h' \cdot (\mathsf{q}, \mathsf{p}, s'[\mathsf{p}']) \cdot h$ *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 [Link] creates a new channel with a unique distinguished role for each parallel process. Both 1. and 2. follow trivially by the induction hypothesis.
- When the reduction step is obtained by rule [Srec] we must have $s : (\mathsf{q}, \mathsf{p}, s'[\mathsf{p}']) \cdot h$. By induction hypothesis we must have $s \neq s'$. By Theorem 6 we can derive a channel relation for the left hand side of the reduction rule [Srec] using the interaction typing rule $\{\text{SREC}\}$. Therefore $s[\mathsf{p}]$ and $s'[\mathsf{p}']$ are the only channels with role in $P\{s'[\mathsf{p}']/y\}$ and point 1. follows immediately.
- When the reduction step is obtained by rule [Deleg] note that the session delegation command must have been typed by rule $\lfloor\text{DELEG}\rfloor$. For this reason we get $s[\mathsf{p}] \neq s'[\mathsf{p}']$. Since $s[\mathsf{p}]$ precedes $s'[\mathsf{p}']$ in the session delegation command, then by induction $s = s'$ implies $\mathsf{p} = \mathsf{p}'$. We then conclude $s \neq s'$.

Definition 7. *Define* $\propto$ *between processes, message queues and local types, as follows:*

$$c!\langle\{\mathsf{p}_k\}_{k\in K}, e\rangle; P \propto\, !\langle\{\mathsf{p}_k\}_{k\in K}, S\rangle; T \qquad c?(\mathsf{q}, x); P \propto ?(\mathsf{q}, S); T$$
$$c!\langle\langle \mathsf{p}', c'\rangle\rangle; P \propto\, !\langle\{\mathsf{p}_k\}_{k\in K}, T\rangle; T \qquad c?((\mathsf{q}, y)); P \propto ?(\mathsf{q}, T); T$$
$$c \oplus \langle\{\mathsf{p}_k\}_{k\in K}, l_i\rangle; P \propto \oplus\langle\{\mathsf{p}_k\}_{k\in K}, \{l_i : T_i\}_{i\in I}\rangle \qquad c\&(\mathsf{q}, \{l_i : P_i\}_{i\in I}) \propto \&(\mathsf{p}, \{l_i : T_i\}_{i\in I})$$
$$(\mathsf{q}, \{\mathsf{p}_k\}_{k\in K}, v) \cdot h \propto\, !\langle\{\mathsf{p}_k\}_{k\in K}, S\rangle; T \qquad (\mathsf{q}, \mathsf{p}', s[\mathsf{p}]) \cdot h \propto\, !\langle\{\mathsf{p}_k\}_{k\in K}, T\rangle; T$$
$$(\mathsf{q}, \{\mathsf{p}_k\}_{k\in K}, l) \cdot h \propto \oplus\langle\{\mathsf{p}_k\}_{k\in K}, \{l_i : T_i\}_{i\in I}\rangle \qquad X\langle e, c\rangle \propto T$$

where $i \in I$*.*

Definition 8. *A process* P *is* ready *in a process* Q *if one of the following conditions holds:*
- $Q \equiv P$*;*
- $Q \equiv P \mid R$ *for some* R*;*
- $Q \equiv (\nu a)R$ *and* P *is ready in* R*, for some* R*,* a*;*
- $Q \equiv (\nu s)R$ *and* P *is ready in* R*, for some* R*,* s*;*
- $Q \equiv \mathsf{def}\ D\ \mathsf{in}\ R$ *and* P *is ready in* R*, for some* R*,* D*.*

Definition 9.
- *An* input process *is a value sending, session delegation or label selection.*
- *An* output process *is a value reception, session reception or label branching.*
- *The identifier* u *is the* subject *of* $\bar{u}[n](y).P$ *and* $u[\mathsf{p}](y).P$*.*
- *The channel* c *is the* subject *of* $c!\langle\{\mathsf{p}_k\}_{k\in K}, e\rangle; P$*,* $c?(\mathsf{q}, x); P$*,* $c!\langle\langle \mathsf{p}', c'\rangle\rangle; P$*,* $c?((\mathsf{q}, y)); P$*,* $c \oplus \langle\{\mathsf{p}_k\}_{k\in K}, l\rangle; P$ *and* $c\&(\mathsf{q}, \{l_i : P_i\}_{i\in I})$*.*
- *An* output *type is a type of the shape* $!\langle\{\mathsf{p}_k\}_{k\in K}, U\rangle; \mathsf{T}$*,* $\oplus\langle\{\mathsf{p}_k\}_{k\in K}, \{l_i : T_i\}_{i\in I}\rangle$*, or* $\oplus\langle\{\mathsf{p}_k\}_{k\in K}, l\rangle; \mathsf{T}$*.*
- *An* input *type is a type of the shape* $?(\{\mathsf{p}_k\}_{k\in K}, U); T$*, or* $\&(\mathsf{p}, \{l_i : T_i\}_{i\in I})$*.*

Lemma 11. *Assume that*
- $\Theta \vdash P \blacktriangleright \mathscr{R}; \mathscr{N}; \mathscr{B}$*;*
- $\mathscr{R}$ *contains service names which are not bigger than channels with roles and less than at least one channel with role;*
- *no ready process in* P *is an output or a conditional or a process call or a session initialisation on a variable.*

Then P *contains one ready session initialisation on a free service name which belongs to* $\mathscr{R} \cup \mathscr{N}$*.*

Proof. If P is a session initialisation on a free service name which belongs to $\mathscr{R} \cup \mathscr{N}$ there is nothing to prove. Otherwise the proof is by induction on P.

P cannot be a session initialisation on a free session name which does not belong to $\mathscr{R} \cup \mathscr{N}$, since otherwise $\mathscr{R}$ could not contain channels with roles by Lemma 8.

P cannot be an input process since otherwise by Lemma 9 a channel with role would be less than all channels with roles which occur in $\mathscr{R}$.

If $P \equiv P_1 \mid P_2$, then $\mathscr{R} = \mathscr{R}_1 \uplus \mathscr{R}_2$ and $\Theta \vdash P_1 \blacktriangleright \mathscr{R}_1 ; \mathscr{N}_1 ; \mathscr{B}_1$ and $\Theta \vdash P_2 \blacktriangleright \mathscr{R}_2 ; \mathscr{N}_2 ; \mathscr{B}_2$ for some $\mathscr{R}_1, \mathscr{R}_2$, since the last applied rule for deriving $\Theta \vdash P \blacktriangleright \mathscr{R} ; \mathscr{N} ; \mathscr{B}$ must be $\{\textsc{Conc}\}$. Note that at least one between $\mathscr{R}_1$ and $\mathscr{R}_2$ must contain session names which are not bigger than channels with roles and less than at least one channel with role. Therefore by induction either P_1 or P_2 contains a ready session initialisation on a free service name which belongs to $\mathscr{R} \cup \mathscr{N}$.

If $P \equiv \mathsf{def}\ X(x,y) = P'$ in Q, then $\Theta, X[y] \blacktriangleright \mathscr{R}' ; \mathscr{N}' ; \mathscr{B}' \vdash Q \blacktriangleright \mathscr{R} ; \mathscr{N} ; \mathscr{B}$ since the last applied rule for deriving $\Theta \vdash P' \blacktriangleright \mathscr{R}' ; \mathscr{N}' ; \mathscr{B}'$ must be $\{\textsc{Def}\}$. Therefore by induction Q contains a ready session initialisation on a free service name which belongs to $\mathscr{R} \cup \mathscr{N}$.

If $P \equiv (\nu a)P'$, then $\Theta \vdash P' \blacktriangleright \mathscr{R} ; \mathscr{N} ; \mathscr{B}'$ where $\mathscr{B}' = \mathscr{B} \setminus a$ and $a \notin \mathscr{R} \cup \mathscr{N}$, since the last applied rule for deriving $\Theta \vdash (\nu a)P \blacktriangleright \mathscr{R} ; \mathscr{N} ; \mathscr{B}$ must be $\{\textsc{NRes}\}$. Therefore by induction P' contains a ready session initialisation on a free service name which belongs to $\mathscr{R} \cup \mathscr{N}$.

Lemma 12. *Assume that*

- $\Gamma \vdash_\Sigma P \triangleright \Delta$*;*
- $\Theta \vdash P \blacktriangleright \mathscr{R} ; \mathscr{N} ; \mathscr{B}$ *is proved without using rule* $\{\textsc{SRes}\}$*;*
- *s is minimal in $\mathscr{R}$;*
- *no* $s[\mathsf{p}]$ *precedes* $s[\mathsf{q}]$ *with* $\mathsf{p} \neq \mathsf{q}$ *in P;*
- *no ready process in P is an output, a conditional, a process call, a session initialisation on a free channel or on a variable.*

Then:

1. *if* $\Delta(s[\mathsf{p}])$ *is an input type then P contains a ready input process Q with subject* $s[\mathsf{p}]$ *such that* $Q \propto \Delta(s[\mathsf{p}])$*;*
2. *if* $\Delta(s[\mathsf{p}])$ *is an output type then P contains the queue* $s : h$ *and* $h \propto \Delta(s[\mathsf{p}])$.

Proof. The proof of both points is by induction on P. Note that P cannot be a session initialisation on a bound channel, i.e. we cannot have $P \equiv (\nu a)Q$ where Q is a session initialisation on the channel a, since in that case the channel relation for Q should contain $a \prec s$ and this is impossible by Lemma 8.

(1). If P is an input process, then by Lemmas 9 and 10 the subject of P must be $s[\mathsf{p}]$: obviously P is ready. Note that P is a user process and then $\Gamma \vdash P \triangleright \Delta$ by Lemma 2(1). We get $P \propto \Delta(s[\mathsf{p}])$ by Lemma 1(8), (10) and (1).

If $P \equiv P_1 \mid P_2$, then by Lemma 2(6) $\Sigma = \Sigma_1 \cup \Sigma_2$ and $\Delta = \Delta_1 * \Delta_2$ and $\Gamma \vdash_{\Sigma_1} P_1 \triangleright \Delta_1$ and $\Gamma \vdash_{\Sigma_2} P_2 \triangleright \Delta_2$. Since an input type is never a message type we have either $\Delta(s[\mathsf{p}]) = \Delta_1(s[\mathsf{p}])$ or $\Delta(s[\mathsf{p}]) = \Delta_2(s[\mathsf{p}])$. Assume $\Delta(s[\mathsf{p}]) = \Delta_1(s[\mathsf{p}])$. Moreover, since the last applied rule must be $\{\textsc{Conc}\}$, $\Theta \vdash P_1 \blacktriangleright \mathscr{R}_1 ; \mathscr{N}_1 ; \mathscr{B}_1$ and $\Theta \vdash P_2 \blacktriangleright \mathscr{R}_2 ; \mathscr{N}_2 ; \mathscr{B}_2$ and $\mathscr{R} = \mathscr{R}_1 \uplus \mathscr{R}_2$. Note that by Lemma 7 $\mathscr{R}_1$ contains s. Moreover s is minimal in $\mathscr{R}_1$ since $\mathscr{R}_1 \subseteq \mathscr{R}$. Therefore by induction P_1 contains a ready input process Q with subject $s[\mathsf{p}]$ such that $Q \propto \Delta(s[\mathsf{p}])$.

If $P \equiv \mathsf{def}\ X(x,y) = P'$ in Q, then by Lemma 2(9) $\Gamma, X : S\ T, x : S \vdash P \triangleright y : T$ and $\Gamma, X : S\ T \vdash_\Sigma Q \triangleright \Delta$. Moreover $\Theta, X[y] \blacktriangleright \mathscr{R}' ; \mathscr{N}' ; \mathscr{B}' \vdash Q \blacktriangleright \mathscr{R} ; \mathscr{N} ; \mathscr{B}$, since the last applied rule for deriving $\Theta \vdash P' \blacktriangleright \mathscr{R}' ; \mathscr{N}' ; \mathscr{B}'$ must be $\{\textsc{Def}\}$. Therefore by induction Q contains a ready input process Q with subject $s[\mathrm{p}]$ such that $Q \propto \Delta(s[\mathrm{p}])$.

If $P \equiv (\nu a)P'$, then by Lemma 2(8) $\Gamma, a : \langle G \rangle \vdash_\Sigma P' \triangleright \Delta$. Moreover, since the last applied rule for deriving $\Theta \vdash (\nu a)P' \blacktriangleright \mathscr{R} ; \mathscr{N} ; \mathscr{B}$ must be $\{\textsc{NRes}\}$, $\Theta \vdash P' \blacktriangleright \mathscr{R} ; \mathscr{N} ; \mathscr{B}'$ where $\mathscr{B} = \mathscr{B}' \setminus a$ and $a \notin \mathscr{R} \cup \mathscr{N}$. Therefore by induction P' contains a ready input process Q with subject $s[\mathrm{p}]$ such that $Q \propto \Delta(s[\mathrm{p}])$.

(2). If P is a queue, then it must be the queue s and the result follows from Lemma 3.

If $P \equiv P_1 \mid P_2$, then by Lemma 2(6) $\Sigma = \Sigma_1 \cup \Sigma_2$ and $\Delta = \Delta_1 * \Delta_2$ and $\Gamma \vdash_{\Sigma_1} P_1 \triangleright \Delta_1$ and $\Gamma \vdash_{\Sigma_2} P_2 \triangleright \Delta_2$. We consider the case $\Delta(s[\mathrm{p}]) = \Delta_1(s[\mathrm{p}]);\Delta_2(s[\mathrm{p}])$, the other cases being similar or simpler. As in the proof of (1) we get $\mathscr{R} = \mathscr{R}_1 \uplus \mathscr{R}_2$ and $\Theta \vdash P_1 \blacktriangleright \mathscr{R}_1 ; \mathscr{N}_1 ; \mathscr{B}_1$. Note that by Lemma 7(1) $\mathscr{R}_1$ contain s. Therefore by induction P_1 contains the queue $s{:}h$ and $h \propto \Delta(s[\mathrm{p}])$.

If $P \equiv \mathsf{def}\ P_1\ \mathsf{in}\ P_2$ or $P \equiv (\nu a)P'$, the proof proceeds as in the case of (1).

Proof of Theorem 2 [Progress].
Let P_0 be initial and $P_0 \longrightarrow^* P$.

If P does not contain channels with roles there is nothing to prove.

If a ready sub-process of P is an output process, then P is reducible.

If a ready process in P is a conditional, then P would reduce, since P is closed (being P_0 closed) and any closed boolean value is either true or false. Similarly if a ready process of P is a process call it can be reduced.

No ready process in P is an accept/request on a variable since P is closed.

If one ready process in P is an accept/request on a free channel a, then a must be in the domain of the standard environment Γ used to type P_0 and P. Even if in P there are not enough partners to apply rule [Link], using $\Gamma(a)$ we can build a process Q containing the missing partners which are necessary in order to apply it to $P \mid Q$.

Otherwise let $P \equiv (\nu \tilde{s})Q$, where $\tilde{s}$ is the set of all session names which occur in P. By the Type Preservation Theorems 4 and 6 P is well typed both in the communication and in the interaction type systems. This implies $\vdash Q \blacktriangleright \mathscr{R} ; \mathscr{N} ; \mathscr{B}$ for some $\mathscr{R}, \mathscr{N}, \mathscr{B}$. Let Δ be the session environment of Q. Note that by construction we do not use rule $\{\textsc{SRes}\}$ for deriving $\mathscr{R}$. All minimals in $\mathscr{R}$ cannot be service names names since otherwise P would contain one ready initialisation on a free service name by Lemma 11. So there must be a session name s which is minimal. By Lemma 7(2) and the coherence of Δ there must be p, q such that $\Delta(s[\mathrm{p}]) = T$, $\Delta(s[\mathrm{q}]) = T'$ and $T \restriction \mathrm{q} \bowtie T' \restriction \mathrm{p}$. Without loss of generality we can assume that T is an input type and T' is an output type. Then Lemma 12 implies that Q contains a ready input process R such that $R \propto T$ and the queue $s : h$ with $h \propto T'$. Therefore P reduces by rule [Recv].