A Gentle Introduction to Multiparty Asynchronous Session Types

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

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

Abstract. This article provides a gentle introduction to multiparty session types, a class of behavioural types specifically targeted at describing protocols in distributed systems based on asynchronous communication. The type system ensures well-typed processes to enjoy non-trivial properties, including communication safety, protocol fidelity, as well as progress. The adoption of multiparty session types can positively affect the whole software lifecycle, from design to deployment, improving software reliability and reducing its development costs.

1 Introduction

In modelling distributed systems where processes interact by means of message passing, one soon realises that many interactions are meant to occur within the scope of private channels following disciplined protocols. We call such private interactions *sessions* and the protocols that describe them *session types*. In its simplest form, a session is established between *two* peers, such as a client connecting with a server. In these cases, the sessions are "binary" or "dyadic" [39, 42]. In general, a session may involve any (usually fixed, but sometimes variable) number of peers. In these cases, we speak of *multiparty sessions* and of their protocol descriptions as of *multiparty session types* [43].

The ability to describe complex interaction protocols by means of a formal, simple and yet expressive type language can have a profound impact on the way distributed systems are designed and developed. This is witnessed by the fact that some important standardisation bodies for web-based business and finance protocols [73, 72, 68] have recently investigated design and implementation frameworks for specifying message exchange rules and validating business logic based on the notion of *multiparty sessions*, where multiparty session types are "shared agreements" between teams of programmers developing possibly large and complex distributed protocols or software systems.

A multiparty session type theory consists of three main ingredients:

- At the most abstract level is the *global type*, which describes a communication protocol from a neutral viewpoint in terms of the interactions that are supposed to occur between the protocol peers, of the order of these interactions, and of the kind of messages exchanged during these interactions.
- At the most concrete level are *processes*, which describe the behaviour of the peers involved in the session using a formal language (usually, a dialect of the π-calculus).

– Somehow in between these two levels are *local types*, one for each peer, which describe the same communication protocol as the global type, but from the viewpoint of each peer.

These ingredients are strictly related: a *projection operation* extracts the local type of each peer from the global type, and a *type system* makes sure that a process uses the communication channels it owns according to their local type. Once these relations are established, a number of properties can be proved, among which:

– *communication safety*, namely the fact that there is never a mismatch between the types of sent and expected messages, despite the same communication channel is used for exchanging messages of different types;
– *protocol fidelity*, namely the fact that the interactions that occur are accounted for by the global type and therefore are allowed by the protocol;
– *progress*, namely the fact that every message sent is eventually received, and every process waiting for a message eventually receives one.

Remarkably, these properties are guaranteed by means of purely local checks on the single peers that participate in the protocol, despite the fact that they will run independently once the session has been established. The ability to prove relevant global properties by means of local checks is one of the key features of session type theories.

The present article formalises these concepts and provides a gentle introduction to multiparty session type theory. The process calculus and the type system we use have been first introduced in [3] and then developed in [25]. Notably, the focus of these two papers was the design of a type system assuring progress even in presence of session interleaving. In this article we solely describe the so-called *communication type system*, which assures communication safety, protocol fidelity and, when no sessions are interleaved, progress.

Outline. We start illustrating our calculus with simple yet comprehensive examples in §2. The calculus of asynchronous, multiparty sessions is the content of §3. The communication type system assuring that processes behave correctly with respect to the sessions in which they are involved is illustrated with examples in §4. §5 discusses related work and further readings. To ease reading and accessibility of the content, proofs of the properties enjoyed by well-typed processes and additional technical material have been collected in the Appendix.

2 Examples

In this section we present two versions of a simple but non-trivial example that illustrates the basic functionalities and features of the process calculus that we work with. This example comes from a Web service usecase in Web Service Choreography Description Language (WS-CDL) Primer 1.0 [73], capturing a collaboration pattern typical to many business and distributed protocols [62, 72, 69].

2.1 Example 1: the three buyer protocol

The setting is that of a system involving Alice, Bob, and Carol that cooperate in order to buy a book from a Seller. The participants follow a protocol that is described informally below:

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

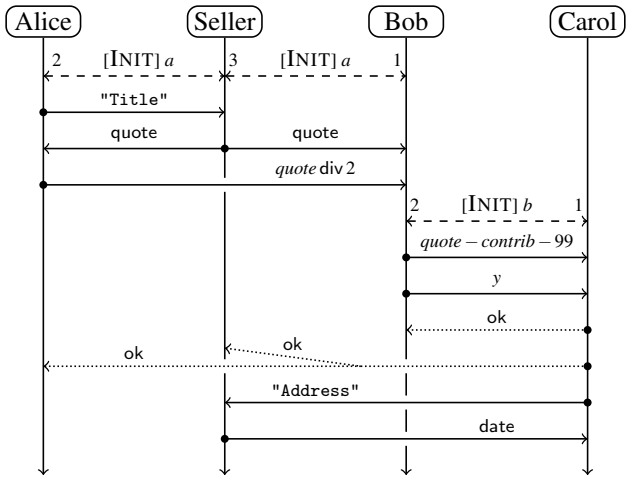

Fig. 1. An execution of the three buyer protocol.

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

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

(i) so that we can easily refer to the actions in them.

$\mathsf{G}_a =$

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

$\mathsf{G}_b =$

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

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

Global types provide an overall description of the two conversations, directly abstracting the scenario of the diagram. In G_a, line (1) denotes Alice sending a string value to Seller. Line (2) says that Seller sends the same integer value to Alice and Bob and line (3) says that Alice sends an integer to Bob. In lines (4–6) Bob sends either ok or quit to Seller and Alice. In the first case Bob sends a string to Seller and receives a date from Seller, in the second case there are no further communications.

Line (2) in G_b represents the delegation of a channel with the communication behaviour specified by the session type T from Bob to Carol (note that Seller and Alice in T concern the session on a). Then Carol terminates the interaction as if she were Bob in session a. Note that in this case the Seller do not know if he is talking with Bob or Alice.

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

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

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

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

Table 1. Implementation of the three buyer protocol.

Table 1 shows an implementation of the three buyer protocol conforming to G_a and G_b for the processes Seller, Alice, Bob, and Carol in the calculus that we will formally define in §3.1. The service name a is used for initiating sessions corresponding to the global type G_a. Seller initiates a three party session by means of the session request operation $\overline{a}[3](y)$, where the index 3 identifies Seller. Since 3 is also the overall number of participants in this session, a occurs with an over-bar. Alice and Bob get involved in the session by means of the session accept operations $a[1](y)$ and $a[2](y)$ and the indexes 2 and 1 identify them as Alice and Bob, respectively. Once the session has

started, Seller, Alice and Bob communicate using their private channels represented by y. Each channel y can be interpreted as a session endpoint connecting a participant with all the others in the same session; the receivers of the data sent on y are specified by giving the participant numbers. Line (1) of G_a is implemented by the matching output and input actions $y!\langle \mathtt{p}, \mathtt{"Title"} \rangle$ of Alice and $y?(2, \mathit{title})$ of the Seller. Line (2) of G_a is implemented by the output action $y!\langle \{1,2\}, \mathsf{quote} \rangle$ of the Seller which is matched by the input actions $y?(3, \mathit{quote})$ of both Bob and Alice. Line (3) of G_b is implemented by the selection and branching actions $z \oplus \langle 2, \mathsf{ok} \rangle$, $z \oplus \langle 2, \mathsf{quit} \rangle$ and $z \& (1, \{\mathsf{ok} : \mathbf{0}, \mathsf{quit} : \mathbf{0}\})$.

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

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

2.2 Example 2: the three buyer protocol with recursion

We now describe a variant of the above example that uses recursion. The scenario is basically the same, the only part that changes is that, if the price exceeds the budget, Bob initiates a negotiation with Carol to collaborate together by establishing a new session: Bob starts asking a first proposal of contribution to Carol. At each step Carol answers with a new offer. Bob can accept the offer, try with a new proposal or give up. When Bob decides to end the negotiation (accepting the offer or giving up) he communicates the exit to Carol and, as before, Carol concludes the protocol with Seller.

Figure 2 depicts the part of the protocol involving recursion.

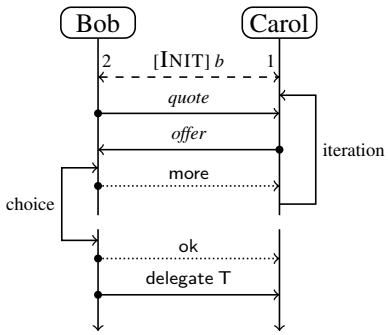

Fig. 2. The three buyer protocol with recursion: additional interactions between Bob and Carol.

The communication protocols are described by the following global types; these are similar to the ones of the previous example. In particular G_a is exactly the same (since the server does not notice the further interactions among the buyers). Instead, G_b is now

$$\texttt{Seller} = \overline{a}[3](y).y_3?(2,\mathit{title}).y!\langle\{1,2\},\mathsf{quote}\rangle.y\&(1,\{\mathsf{ok} : y?(1,\mathit{address}).y!\langle 1,\mathsf{date}\rangle.\mathbf{0},\ \mathsf{quit} : \mathbf{0}\})$$

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

$$\begin{array}{l}\texttt{Bob} = a[1](y).y?(3,\mathit{quote}).y?(2,\mathit{contrib}).\\ \quad \mathsf{if}\ (\mathit{quote}\text{ - }\mathit{contrib} < 100)\ \mathsf{then}\ \ y \oplus \langle\{2,3\},\mathsf{ok}\rangle.y!\langle 3,\texttt{"Address"}\rangle.y?(3,\mathit{date}).\mathbf{0}\\ \quad \mathsf{else}\ \overline{b}[2](z).\\ \qquad \mathsf{def}\ X(x',z',y') =\\ \qquad\quad z'!\langle 1,x'\rangle.z?(1,w).\\ \qquad\quad \mathsf{if}\ \mathit{good}(w)\ \mathsf{then}\ z' \oplus \langle 1,\mathsf{ok}\rangle.z'!\langle\langle 1,y'\rangle\rangle.\mathbf{0}\\ \qquad\quad \mathsf{else\ if}\ \mathit{negotiable}(w)\ \mathsf{then}\ z' \oplus \langle 1,\mathsf{more}\rangle.X\langle \mathit{newproposal}(w),z',y'\rangle\\ \qquad\qquad \mathsf{else}\ z' \oplus \langle 1,\mathsf{quit}\rangle.y' \oplus \langle\{2,3\},\mathsf{quit}\rangle.\mathbf{0}\\ \qquad \mathsf{in}\ X\langle \mathit{firstproposal}(\mathit{quote}),z,y\rangle\end{array}$$

$$\begin{array}{l}\texttt{Carol} = b[1](z).\ \mathsf{def}\ Y(z') = z'?(2,x).z'!\langle 2,\mathit{offer}(x)\rangle.\\ \qquad z'\&(2,\{\mathsf{ok} : z'?((2,t)).t \oplus \langle\{2,3\},\mathsf{ok}\rangle.t!\langle 3,\texttt{"Address"}\rangle.t?(3,\mathit{date}).\mathbf{0}\\ \qquad\qquad\quad \mathsf{more} : Y\langle z'\rangle,\\ \qquad\qquad\quad \mathsf{quit} : \mathbf{0}\})\\ \qquad \mathsf{in}\ Y\langle z'\rangle\end{array}$$

Fig. 3. The three buyer example with recursion.

more involved since we have a recursive part which represents the (possibly) recursive negotiation between Bob and Carol.

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

$$\begin{array}{ll}G_b = & \\ (1)\ \mu\mathbf{t}.2 \longrightarrow 1 : & \langle\mathsf{int}\rangle.\\ (2)\ \quad 1 \longrightarrow 2 : & \langle\mathsf{int}\rangle.\\ (3)\ \quad 2 \longrightarrow 1 : & \{\mathsf{ok} : 2 \longrightarrow 1 : \langle\mathsf{T}\rangle.\mathsf{end},\\ & \qquad \mathsf{more} : \mathbf{t},\\ & \qquad \mathsf{quit} : \mathsf{end}\}\end{array}$$

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

The code of the example is in Figure 3. Again, it is similar to the previous one, but for the recursive definitions in the processes `Bob` and `Carol`. Note that the recursive process X in Bob's code has a data parameter (x') and two channel parameters (y' and z'), while the process Y in Carol's code has only one channel parameter (z').

3 The Calculus for Multiparty Sessions

In this section we formalise syntax and operational semantics of the calculus of multiparty asynchronous sessions. To ease the presentation and limit some technicalities, with respect to the previous section we consider a slightly simpler calculus in which communication actions always specify exactly one receiver instead of a non-empty set of receivers and we assume that recursive definitions have exactly one data parameter and one channel parameter. Allowing multiple receivers is mostly a matter of syntactic

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

Table 2. Process syntax and naming conventions.

sugar, since a communication action involving multiple receivers can be canonically encoded as a sequence of actions involving single receivers only. Some notions, however, such as the projection operator, are affected by this design choice and should be adjusted accordingly. The interested reader may refer to [25] for the presentation of the calculus with native support for multiple receivers, and to [34] for the definition of a projection operator that can handle actions with multiple receivers encoded as sequences of actions with single receivers only.

3.1 Syntax

The present calculus is a variant of the calculus in [43], as explained in §5. The syntax of *processes*, ranged over by $P, Q \ldots$, and that of *expressions*, ranged over by $e, e', \ldots$, is given by the grammar in Table 2, which shows also naming conventions. The operational semantics is defined by a set of reduction rules. In the reduction of processes it is handy to introduce elements, like queues of messages and runtime channels, which are not expected to occur in the source code written by users (*user processes*). These elements, which are referred as *runtime syntax*, appear shaded.

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

by a *channel with role* $s[\mathtt{p}]$, which represents the runtime channel of the participant $\mathtt{p}$ in the session s.

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

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

As usual evaluation contexts are processes with some holes.

As in [43], we use message queues in order to model TCP-like asynchronous communications (where message order is preserved and sending is non-blocking). A message in a queue can be a value message, $(\mathtt{q}, \mathtt{p}, v)$, indicating that the value v was sent by the participant $\mathtt{q}$ and the recipients is the participant $\mathtt{p}$; 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, $(\mathtt{q}, \mathtt{p}, l)$ (similar to a value message). The empty queue is denoted by $\varnothing$. By $h \cdot m$ we denote the queue obtained by concatenating m to the queue h. With some abuse of notation we will also write $m \cdot h$ to denote the queue with head element m. By $s : h$ we denote the queue h of the session s. Queues and channels with role are generated by the operational semantics (described later).

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

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

3.2 Operational Semantics

Processes are considered modulo structural equivalence, denoted by $\equiv$, and defined adding α-conversion to the rules in Table 3. We denote by $\mathrm{fn}(Q)$ ($\mathrm{fn}(D)$) the set of free names in Q (D), by $\mathrm{dpv}(D)$ the set of process variables declared in D and by $\mathrm{fpv}(Q)$ the set of process variables which occur free in Q. Besides the standard rules [50], we have a rule for rearranging messages in a queue when the senders or the receivers are not the same.

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

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

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

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

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

$$s : h \cdot (\mathtt{q}, \mathtt{p}, \zeta) \cdot (\mathtt{q}', \mathtt{p}', \zeta') \cdot h' \equiv s : h \cdot (\mathtt{q}', \mathtt{p}', \zeta') \cdot (\mathtt{q}, \mathtt{p}, \zeta) \cdot h' \quad \text{if } \mathtt{p} \neq \mathtt{p}' \text{ or } \mathtt{q} \neq \mathtt{q}'$$

Table 3. Structural equivalence.

Table 4 shows the reduction rules of processes (we use $\longrightarrow^*$ and $\longrightarrow^k$ with the expected meaning). Rule [Init] describes the initiation of a new session among n participants that synchronise over the service name a. The last participant $\overline{a}[n](y).P_n$, distinguished by the overbar on the service name, specifies the number n of participants. After the initiation, the participants will share the private session name s, and the queue associated to s, which is initially empty. The variable y in each participant $\mathtt{p}$ will be replaced by the corresponding channel with role $s[\mathtt{p}]$. The output rules [Send], [Deleg] and [Sel] enqueue values, channels and labels, respectively, into the queue of the session s (in rule [Send], $e \downarrow v$ denotes the evaluation of the expression e to the value v). The input rules [Rcv], [SRcv] and [Branch] perform the corresponding complementary operations. Note that these operations check that the sender matches, and also that the message is actually meant for the receiver.

4 Communication Type System

This section introduces the communication type system, by which we can check type soundness of the communications and protocol fidelity. This type system is the one introduced in [25], but the proof of subject reduction is cleaned up by the use of the property stated in Lemma 1. As we have done in §3.1, here too we only consider communication actions with single receivers, even though the examples make use of a slightly general syntax.

4.1 Global and Session Types

Global types describe the whole conversation scenarios of multiparty session. *Session types* correspond to projections of global types on the individual participants: they are types of pure processes. The grammar for global and session types is given in Table 5.

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

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

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

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

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

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

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

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

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

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

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

Table 4. Reduction rules.

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

The global type $\mathtt{p} \to \mathtt{q} : \langle S\rangle.G$ says that participant $\mathtt{p}$ multicasts a value of sort S to the participant $\mathtt{q} \neq \mathtt{p}$ and then the interactions described in G take place. Similarly, the global type $\mathtt{p} \to \mathtt{q} : \langle \mathsf{T}\rangle.G$ says that participant $\mathtt{p} \neq \mathtt{q}$ delegates a channel of type T to participant $\mathtt{q}$ and the interaction continues according to G. When it does not matter we use $\mathtt{p} \to \mathtt{q} : \langle U\rangle.G$ to refer both to $\mathtt{p} \to \mathtt{q} : \langle S\rangle.G$ and $\mathtt{p} \to \mathtt{q} : \langle \mathsf{T}\rangle.G$.
Type $\mathtt{p} \to \mathtt{q} : \{l_i : G_i\}_{i\in I}$ says that participant $\mathtt{p}$ send one of the labels l_i to participants $\mathtt{q}$. If l_j is sent, interactions described in G_j take place. Type $\mu\mathbf{t}.G$ is a recursive type, assuming type variables $(\mathbf{t}, \mathbf{t}', \ldots)$ are guarded in the standard way, i.e., type variables only appear under some prefix. We take an *equi-recursive* view of recursive types, not distinguishing between $\mu\mathbf{t}.G$ and its unfolding $G\{\mu\mathbf{t}.G/\mathbf{t}\}$ [66, §21.8]. Type end represents the termination of the session.

Session types represent the input-output actions performed by single participants. The *send types* $!\langle \mathtt{p}, S\rangle.T$, $!\langle \mathtt{p}, \mathsf{T}\rangle.T$express, respectively, the sending of a value of sort S to participant $\mathtt{p}$ or the sending of a channel of type T to participant $\mathtt{p}$ followed by the communications described by T. The *selection type* $\oplus\langle \mathtt{p}, \{l_i : T_i\}_{i\in I}\rangle$ represents the transmission to participant $\mathtt{p}$ of a label l_i chosen in the set $\{l_i \mid i \in I\}$ followed by the communications described by T_i. The *receive* and *branching* types are dual of send and selection types. Recursion is guarded also in session types, and we consider them modulo fold/unfold as done for global types.

The relation between global and session types is formalised by the notion of projection as in [43].

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

Table 5. Global and session types.

Definition 1. *The* projection of a global type G onto a participant $\mathtt{q}$ *($G \upharpoonright \mathtt{q}$) is defined by induction on G:*

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

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

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

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

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

where T is defined at page 4.

Hereafter we assume all global types are *well formed*, i.e. $G \upharpoonright \mathtt{q}$ is defined for all $\mathtt{q}$ which occur in G.

4.2 Typing Rules for Pure Processes

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

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

where

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

Formally we define:

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

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

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

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

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

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

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

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

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

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

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

The present formulation of rule (DEF) forces to type process variables only with μ-types, while the formulation in [3, 43]:

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

allows to type unguarded process variables with arbitrary types, which can be meaningless. For example with the more permissive rule we can derive

$$\vdash \mathsf{def}\ X(x,y) = X(x,y)\ \mathsf{in}\ X\langle \mathsf{true}, z \rangle \triangleright \{z : \mathsf{T}\}$$

for an arbitrary closed T, while in our system we cannot type this process since its only possible type would be $\mu \mathbf{t}.\mathbf{t}$, which is not guarded and then forbidden.

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

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

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

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

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

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

$$\frac{\Gamma \vdash P_i \triangleright \Delta, c : T_i \quad \forall i \in I}{\Gamma \vdash c \& (\mathtt{p}, \{l_i : P_i\}_{i \in I}) \triangleright \Delta, c : \&(\mathtt{p}, \{l_i : T_i\}_{i \in I})} \ (\textsc{Branch})$$

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

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

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

Table 6. Typing rules for expressions and pure processes.

4.3 Types and Typing Rules for Runtime Processes

In this subsection we extend the communication type system to processes containing queues. We start by defining the types of queues.

$$\begin{array}{rrll}
\text{Message Types } M & ::= & !\langle \mathtt{p}, U \rangle & \textit{message send} \\
& \mid & \oplus\langle \mathtt{p}, l \rangle & \textit{message selection} \\
& \mid & M; M & \textit{message sequence}
\end{array}$$

$$\begin{array}{rrll}
\text{Generalised} & \tau & ::= & T & \textit{session} \\
& & \mid & M & \textit{message} \\
& & \mid & M; T & \textit{continuation}
\end{array}$$

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

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

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

Table 7. Typing rules for queues.

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

A *generalised type* is either a session type, or a message type, or a message type followed by a session type. Type $M;T$ represents the continuation of the type M associated to a queue with the type T associated to a pure process. Examples of generalised types are

$$!\langle 3, \mathsf{string}\rangle.?(3, \mathsf{date}).\mathsf{end} \ \text{ and } \ !\langle 3, \mathsf{string}\rangle; ?(3, \mathsf{date}).\mathsf{end},$$

which only differ for the replacement of the leftmost "." by ";". In the first the type $!\langle 3, \mathsf{string}\rangle$ corresponds to an output action sending a string to participant 3, while in the second type $!\langle 3, \mathsf{string}\rangle$ corresponds to a message for participant 3 with a value of type string. See the examples of typing judgements at the end of this subsection.

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

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

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

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

In order to take into account the structural congruence between queues (see Table 3) we consider message types modulo the equivalence relation $\approx$ induced by the rule:

$$M;\ \natural\langle \mathtt{p}, Z\rangle;\ \natural'\langle \mathtt{p}', Z\rangle; M' \approx M;\ \natural'\langle \mathtt{p}', Z\rangle;\ \natural\langle \mathtt{p}, Z\rangle; M' \quad \text{if } \mathtt{p} \neq \mathtt{p}'$$

where $\natural \in \{!, \oplus\}$ and $Z \in \{U, l\}$).

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

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

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

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

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

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

We extend $*$ to session environments as expected:

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

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

To give the rules for typing processes with queues we introduce consistency of session environments, which assures that each pair of participants in a multiparty conversation performs their mutual communications in a consistent way. Consistency is defined using the notions of projection of generalised types and of duality, given respectively in Definitions 2 and 3. Notice that projection is not defined for message types.

Definition 2. *The* partial *projection of the generalised type τ onto $\mathtt{q}$, denoted by $\tau \upharpoonright \mathtt{q}$, is defined by:*

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

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

$$(\oplus\langle \mathtt{p}, \{l_i : T_i\}_{i\in I}\rangle) \upharpoonright \mathtt{q} = \begin{cases} \oplus\{l_i : T_i \upharpoonright \mathtt{q}\}_{i\in I} & \textit{if } \mathtt{q} = \mathtt{p}, \\ T_{i_0} \upharpoonright \mathtt{q} & \textit{where } i_0 \in I \textit{ if } \mathtt{q} \neq \mathtt{p} \textit{ and } T_i \upharpoonright \mathtt{q} = T_j \upharpoonright \mathtt{q} \textit{ for all } i, j \in I. \end{cases}$$

$$(\&(\mathtt{p}, \{l_i : T_i\}_{i\in I})) \upharpoonright \mathtt{q} = \begin{cases} \&\{l_i : T_i \upharpoonright \mathtt{q}\}_{i\in I} & \textit{if } \mathtt{q} = \mathtt{p}, \\ T_{i_0} \upharpoonright \mathtt{q} & \textit{where } i_0 \in I \textit{ if } \mathtt{q} \neq \mathtt{p} \textit{ and } T_i \upharpoonright \mathtt{q} = T_j \upharpoonright \mathtt{q} \textit{ for all } i, j \in I. \end{cases}$$

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

Definition 3. *The* duality relation *between projections of generalised types ($\bowtie$) is the minimal symmetric relation which satisfies:*

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

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

Definition 4. *A session environment Δ is* consistent for the session s *(notation* $\mathsf{co}(\Delta, s)$*) if $s[\mathtt{p}] : \tau \in \Delta$ and $s[\mathtt{q}] : \tau' \in \Delta$ imply $\tau \upharpoonright \mathtt{q} \bowtie \tau' \upharpoonright \mathtt{p}$. A session environment is* consistent *if it is consistent for all sessions which occur in it.*

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

Proposition 1. *Let G be a global type and $\mathtt{p} \neq \mathtt{q}$. Then $(G \upharpoonright \mathtt{p}) \upharpoonright \mathtt{q} \bowtie (G \upharpoonright \mathtt{q}) \upharpoonright \mathtt{p}$.*

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

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

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

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

$$\frac{\Gamma \vdash P \triangleright \Delta}{\Gamma \vdash_{\emptyset} P \triangleright \Delta}\ (\textsc{GInit}) \qquad \frac{\Gamma \vdash_{\Sigma} P \triangleright \Delta \quad \Delta \approx \Delta'}{\Gamma \vdash_{\Sigma} P \triangleright \Delta'}\ (\textsc{Equiv})$$

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

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

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

Table 8. Typing rules for processes.

Table 8 lists the typing rules for processes containing queues. The judgement

$$\Gamma \vdash_{\Sigma} P \triangleright \Delta$$

means that P contains the queues whose session names are in Σ. Rule (GInit) promotes the typing of a pure process to the typing of an arbitrary process without session names, since a pure process does not contain queues. When two arbitrary processes are put in

parallel (rule (GPAR)) we need to require that each session name is associated to at most one queue (condition $\Sigma \cap \Sigma' = \emptyset$).

Examples of derivable judgements are:

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

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

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

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

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

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

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

which can be typed with the non consistent session environment

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

In fact P reduces to the process

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

which cannot be typed and it is stuck.

4.4 Subject Reduction

Since session environments represent the forthcoming communications, by reducing processes session environments can change. This can be formalised as in [43] by introducing the notion of reduction of session environments, whose rules are:

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

where M can be missing and message types are considered modulo the equivalence relation $\approx$ defined at page 14.
The first rule corresponds to putting in a queue a message with sender $\mathtt{p}$, receiver $\mathtt{q}$ and content of type U. The second rule corresponds to reading from a queue a message with sender $\mathtt{p}$, receiver $\mathtt{q}$ and content of type U. The third and fourth rules are similar, but a label is transmitted.

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

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

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

Lemma 1 (Main Lemma). *Let $\Gamma \vdash_\Sigma P \triangleright \Delta$, and $P \longrightarrow P'$ be obtained by any reduction rule different from [Ctxt], [Str], and $\Delta * \Delta_0$ be consistent, for some Δ_0. Then there is Δ' such that $\Gamma \vdash_\Sigma P' \triangleright \Delta'$ and $\Delta \Rightarrow^* \Delta'$ and $\Delta' * \Delta_0$ is consistent.*

We end this section by formulating subject reduction.

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

Appendix A proves subject reduction. Note that communication safety and protocol fidelity easily follow from Theorem 1.

5 Related Work

5.1 Multiparty Session Types

The first theoretical works on multiparty session types are [10] and [43]. The paper [10] uses a distributed calculus where each channel connects a master endpoint to one or more slave endpoints; instead of global types, they solely use (recursion-free) local types. For type checking, local types are projected to binary sessions, so that type safety is ensured using duality, but it loses sequencing information: hence progress in a session interleaved with other sessions is not guaranteed.

In this article we have presented the calculus of [25], which is an essential improvement and simplification of the calculus in [43]. Both processes and types in [43] share a vector of channels and each communication uses one of these channels. In the present work, instead, processes and types use indexes for identifying the participants of a session.

The communication type system in this article improves the one of [43] in two main technical points without sacrificing expressiveness. First, it avoids the overhead of global linearity-check in [43] because our global types automatically satisfy the linearity condition in [43] due to the limitation to bi-directional channel communications. Second, it provides a more liberal policy in the use of variables in delegation, since we do not require to delegate a set of session channels. The global types in [43] have a parallel composition operator, but its projectability from global to local types limits to disjoint senders and receivers; hence our global types do not affect the expressivity.

5.2 Theoretical Studies on Multiparty Session Types

Extensions of the original multiparty session types [43] and of the communication type system in this article have been proposed, often motivated by use cases resulting from industry applications (§ 5.8). Such extensions include: a subtyping for asynchronous multiparty session types enhancing efficiency [52], motivated by financial protocols and multicore algorithms; parametrised global types for parallel programming and Web service descriptions [34]; communication buffered analysis [30]; extensions to the sum-type and its encoding [61] for describing Healthcare workflows; and exception handling for multiparty conversations [15] for Web services and financial protocols; a liveness-preserving refinement for multiparty session types [64].

Multiparty session types can be extended with logical assertions following the design by contract framework [7]. This framework is enriched in [6] to handle stateful logical assertions, while [21] offers more fine-grained property analysis for multiparty session types with these stateful assertions.

In [31] roles are inhabited by an arbitrary number of participants which can dynamically join and leave a session. The paper [71] shows that the multirole session types [31] can be naturally represented in a dependent-typed language.

To enhance expressivity and flexibility of multiparty session types, the work [28] proposes nested, higher-order multiparty session types and the work [18] studies a generalisation of choices and parallelism. The paper [17] directly types a global description language [16] by multiparty session types without using local types. This direct approach can type processes which are untypable in the original multiparty session typing (i.e. the communication type system in this article). The paper [51] extends the work in [17] to compositional global description languages.

As another line of the study, we extend the multiparty session types to express temporal properties [9]. In this work, the global times are enriched with time constraints, in a way similar to timed automata.

A type system enforcing a stronger correspondence between nondeterministic choices expressed in multiparty session types and the behaviour of processes involved in multiparty sessions has been investigated in [8].

An overview of the recent developments in these studies is the survey in the state-of-the art report produced by the Foundations Working Group of the IC COST Action BETTY, entitled "Foundations of Behavioural Types" [45].

5.3 Progress and Session Interleaving

Multiparty session types are a convenient methodology for ensuring progress of systems of communicating processes. However, progress is only guaranteed within a *single* session [43, 35, 31], but not when multiple sessions are interleaved. The first papers considering progress for interleaved sessions required the nesting of sessions in Java [36, 24]. These systems can guarantee progress for only one single active binary session. The work [25] develops a static interaction type system for global progress in dynamically interleaved and interfered multiparty sessions. A type inference algorithm for this system has been studied in [22], although for finite types only. The work [63, technical report] presents a type system for the linear π-calculus that can ensure progress even in

presence of session interleaving, exploiting an encoding similar to that described in [27] of sessions into the linear π-calculus. However, not *all* multiparty sessions can be encoded into well-typed linear π-calculus processes. In this respect, the richer structure of multiparty session types increases the range of systems for which non-trivial properties such as progress can be guaranteed.

5.4 Security

Enforcement of *integrity* properties in multiparty sessions, using session types, has been studied in [4, 67]. These papers propose a compiler which, given a multiparty session description, implements cryptographic protocols that guarantee session execution integrity.

The work [14] and in its extended version [12] propose a session type system for a calculus of multiparty sessions enriched with security levels, adding access control and secure information flow requirements in the typing rules, and show that this type system guarantees preservation of data confidentiality during session execution. In [13] this calculus is equipped with a monitored semantics, which blocks the execution of processes as soon as they attempt to leak information, raising an error.

Various approaches for enforcing security into calculi and languages for structured communications have been recently surveyed in the state-of-the art report produced by the Security Working Group of the IC COST Action BETTY, entitled "Combining Behavioural Types with Security Analysis" [2].

5.5 Behavioural Semantics

Typed behavioural theory has been one of the central topics in the study of the π-calculus throughout its history, for example, reasoning about various encodings into the typed π-calculi [65, 74, 47]. In the context of typed bisimulations and reduction-closed theories, the work [46] shows that unique behavioural theories can be constructed based on the multiparty session types. The behavioural theory in [46] treats the mutual effects of multiple choreographic sessions which are shared among distributed participants as their common knowledge or agreements, reflecting the origin of choreographic frameworks [73]. These features related to multiparty session type discipline make the theory distinct from any type-based bisimulations in the literature and also applicable to a real choreographic usecase from a large-scale distributed system. This bisimulation is called *globally governed*, since it uses global multiparty specifications to regulate the conversational behaviour of distributed processes.

5.6 Runtime Monitoring and Adaptation

Multiparty session types were originally developed to be used for static type checking of communicating processes. Via collaborations with Ocean Observatories Initiative [62], it was discovered that the framework of multiparty session types can be naturally extended to runtime type checking (monitoring). A formulation of the runtime monitoring (dynamic or runtime type checking) is firstly proposed in [20]. Later the work [5] has

formally proved its correctness and properties guaranteed by the runtime monitoring based on multiparty session types. See § 5.8.

Works addressing adaptation for multiparty communications include [26], [23] and [19]. The paper [26] proposes a choreographic language for distributed applications. Adaptation follows a rule-based approach, in which all interactions, under all possible changes produced by the adaptation rules, proceed as prescribed by an abstract model. In [23] a calculus based on global types, monitors and processes is introduced and adaptation is triggered after the execution of the communications prescribed by a global type, in reaction to changes of the global state. In contrast, in [19] adaptation is triggered by security violations, and assures access control and secure information flow.

5.7 Linkages with Other Frameworks

The work [32] gives a linkage between communicating automata [11] and a general graphical version of multiparty session types, proving a correspondence between the safety properties of communicating automata and multiparty session types. The paper [33] studies the sound and complete characterisation of the multiparty session types in communicating automata and applies the result to the synthesis of the multiparty session types. The inference of global types from a set of local types is also studied in [48]. The techniques developed in [33, 48] are extended to a synthesis of general graphical multiparty session types in [49].

The recent work [37] studies the relationship of multiparty session types with Petri Nets. It proposes a conformance relation between global session nets and endpoint programs, and proves its safety.

5.8 Implementations based on Multiparty Session Types

The research group led by the last author is currently designing and implementing a modelling and specification language with multiparty session types [68, 69] in collaboration with some industrial partners [41, 40]. This protocol language is called Scribble. An article [75] also explains the origin and recent development on Scribble.

Java protocol optimisation [70] based on multiparty session types and generation of multiparty cryptographic protocols [4] are also studied. The multiparty session type theory is applied to Healthcare workflows [38]. Its prototype implementation (the multiparty session π-processes with sumtypes) is available from [1].

Based on the runtime type checking theory, we are implementing a runtime monitoring [29, 44, 55] under collaborations with Ocean Observatories Initiative [62]. The work [29, 44] allows interruptions in Scribble and proves the correctness of this extension. Further we generalise the Python implementation to the Actor framework [54]. In order to express temporal properties studied in timed multiparty session types [9], the work [53] extends Scribble with timed constrains and implements the runtime monitoring in Python.

We also apply the multiparty session types to high-performance parallel programming in C [58, 60] and MPI [57]. A parametrised version of Scribble [57, 59] based

on the theory of parametrised multiparty session types [34] is developed. This extension, called Pabble, is used for automatically generating MPI parallel programs from sequential C code in [56].

Acknowledgements. The research reported in this chapter has been partially supported by COST IC1201. The first three authors have been partially supported by MIUR PRIN Project CINA Prot. 2010LHT4KM and Torino University/Compagnia San Paolo Project SALT. The last author has been partially supported by EPSRC EP/K011715/01, EP/K034413/01 and EP/L00058X/1 and the EU project FP7-612985 UpScale.

References

1. Apims, 2014. `http://thelas.dk/index.php?title=Apims`.
2. Massimo Bartoletti, Ilaria Castellani, Pierre-Malo Deniélou, Mariangiola Dezani-Ciancaglini, Silvia Ghilezan, Jovanka Pantovic, Jorge A. Pérez, Peter Thiemann, Bernardo Toninho, and Hugo Torres Vieira. Combining Behavioural Types with Security Analysis, 2014. Submitted for journal publication.
3. Lorenzo Bettini, Mario Coppo, Loris D'Antoni, Marco De Luca, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. Global Progress in Dynamically Interleaved Multiparty Sessions. In Franck van Breugel and Marsha Chechik, editors, *CONCUR'08*, volume 5201 of *LNCS*, pages 418–433. Springer, 2008.
4. Karthikeyan Bhargavan, Ricardo Corin, Pierre-Malo Deniélou, Cédric Fournet, and James J. Leifer. Cryptographic Protocol Synthesis and Verification for Multiparty Sessions. In John C. Mitchell, editor, *CSF'09*, pages 124–140. IEEE Computer Society Press, 2009.
5. Laura Bocchi, Tzu-Chun Chen, Romain Demangeon, Kohei Honda, and Nobuko Yoshida. Monitoring Networks through Multiparty Session Types. In Dirk Beyer and Michele Boreale, editors, *FMOODS/FORTE'13*, volume 7892 of *LNCS*, pages 50–65. Springer, 2013.
6. Laura Bocchi, Romain Demangeon, and Nobuko Yoshida. A Multiparty Multi-Session Logic. In Catuscia Palamidessi and Mark Dermot Ryan, editors, *TGC'12*, volume 8191 of *LNCS*, pages 111–97. Springer, 2012.
7. Laura Bocchi, Kohei Honda, Emilio Tuosto, and Nobuko Yoshida. A Theory of Design-by-Contract for Distributed Multiparty Interactions. In Paul Gastin and François Laroussinie, editors, *CONCUR'10*, volume 6269 of *LNCS*, pages 162–176. Springer, 2010.
8. Laura Bocchi, Hernán C. Melgratti, and Emilio Tuosto. Resolving Non-determinism in Choreographies. In Zhong Shao, editor, *ESOP'14*, volume 8410 of *LNCS*, pages 493–512. Springer, 2014.
9. Laura Bocchi, Weizhen Yang, and Nobuko Yoshida. Timed Multiparty Session Types. In Paolo Baldan and Daniele Gorla, editors, *CONCUR'14*, volume 8704 of *LNCS*, pages 419–434. Springer, 2014.
10. Eduardo Bonelli and Adriana Compagnoni. Multipoint Session Types for a Distributed Calculus. In Gilles Barthe and Cédric Fournet, editors, *TGC'07*, volume 4912 of *LNCS*, pages 240–256. Springer, 2008.
11. Daniel Brand and Pitro Zafiropulo. On Communicating Finite-State Machines. *Journal of the ACM*, 30:323–342, April 1983.
12. Sara Capecchi, Ilaria Castellani, and Mariangiola Dezani-Ciancaglini. Typing Access Control and Secure Information Flow in Sessions. *Information and Computation*, 238:68–105, 2014.

13. Sara Capecchi, Ilaria Castellani, and Mariangiola Dezani-Ciancaglini. Information Flow Safety in Multiparty Sessions. *Mathematical Structures in Computer Science*, 2015. To appear.
14. Sara Capecchi, Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Tamara Rezk. Session Types for Access and Information Flow Control. In Paul Gastin and François Laroussinie, editors, *CONCUR'10*, volume 6269 of *LNCS*, pages 237–252. Springer, 2010.
15. Sara Capecchi, Elena Giachino, and Nobuko Yoshida. Global Escape in Multiparty Sessions. *Mathematical Structures in Computer Science*, 2015. To appear.
16. Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured Communication-Centered Programming for Web Services. *ACM Transactions on Programming Languages and Systems*, 34(2):8, 2012.
17. Marco Carbone and Fabrizio Montesi. Deadlock-freedom-by-design: Multiparty Asynchronous Global Programming. In Roberto Giacobazzi and Radhia Cousot, editors, *POPL'13*, pages 263–274. ACM, 2013.
18. Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, and Luca Padovani. On Global Types and Multi-Party Session. *Logical Methods in Computer Science*, 8(1):24, 2012.
19. Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Jorge A. Pérez. Self-Adaptation and Secure Information Flow in Multiparty Structured Communications: A Unified Perspective. In Marco Carbone, editor, *BEAT'14*, volume 162 of *EPTCS*, pages 9–18, 2014.
20. Tzu-Chun Chen, Laura Bocchi, Pierre-Malo Deniélou, Kohei Honda, and Nobuko Yoshida. Asynchronous Distributed Monitoring for Multiparty Session Enforcement. In Roberto Bruni and Vladimiro Sassone, editors, *TGC'11*, volume 7173 of *LNCS*, pages 25–45. Springer, 2012.
21. Tzu-Chun Chen and Kohei Honda. Specifying Stateful Asynchronous Properties for Distributed Programs. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR'12*, volume 7454 of *LNCS*, pages 209–224. Springer, 2012.
22. Mario Coppo, Mariangiola Dezani-Ciancaglini, Luca Padovani, and Nobuko Yoshida. Inference of Global Progress Properties for Dynamically Interleaved Multiparty Sessions. In Rocco De Nicola and Christine Julien, editors, *COORDINATION'13*, volume 7890 of *LNCS*, pages 45–59. Springer, 2013.
23. Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Self-Adaptive Multiparty Sessions. *Service Oriented Computing and Applications*, pages 1–20, 2014.
24. Mario Coppo, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. Asynchronous Session Types and Progress for Object-Oriented Languages. In Marcello Bonsangue and Einar Broch Johnsen, editors, *FMOODS'07*, volume 4468 of *LNCS*, pages 1–31. Springer, 2007.
25. Mario Coppo, Mariangiola Dezani-Ciancaglini, Nobuko Yoshida, and Luca Padovani. Global Progress for Dynamically Interleaved Multiparty Sessions. *Mathematical Structures in Computer Science*, 2015. To appear.
26. Mila Dalla Preda, Saverio Giallorenzo, Ivan Lanese, Jacopo Mauro, and Maurizio Gabbrielli. AIOCJ: A Choreographic Framework for Safe Adaptive Distributed Applications. In Benoît Combemale, David J. Pearce, Olivier Barais, and Jurgen J. Vinju, editors, *SLE'14*, volume 8706 of *LNCS*, pages 161–170. Springer, 2014.
27. Ornela Dardha, Elena Giachino, and Davide Sangiorgi. Session Types Revisited. In Danny De Schreye, Gerda Janssens, and Andy King, editors, *PPDP'12*, pages 139–150. ACM Press, 2012.
28. Romain Demangeon and Kohei Honda. Nested Protocols in Session Types. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR'12*, volume 7454 of *LNCS*, pages 272–286. Springer, 2012.

29. Romain Demangeon, Kohei Honda, Raymond Hu, Rumyana Neykova, and Nobuko Yoshida. Practical Interruptible Conversations: Distributed Dynamic Verification with Multiparty Session Types and Python. *Formal Methods in System Design*, pages 1–29, 2015.
30. Pierre-Malo Deniélou and Nobuko Yoshida. Buffered Communication Analysis in Distributed Multiparty Sessions. In Paul Gastin and François Laroussinie, editors, *CONCUR'10*, volume 6269 of *LNCS*, pages 343–357. Springer, 2010.
31. Pierre-Malo Deniélou and Nobuko Yoshida. Dynamic Multirole Session Types. In Thomas Ball and Mooly Sagiv, editors, *POPL'11*, pages 435–446. ACM Press, 2011.
32. Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty Session Types Meet Communicating Automata. In Helmut Seidl, editor, *ESOP'12*, volume 7211 of *LNCS*, pages 194–213. Springer, 2012.
33. Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *ICALP'13*, volume 7966 of *LNCS*, pages 174–186. Springer, 2013.
34. Pierre-Malo Deniélou, Nobuko Yoshida, Andi Bejleri, and Raymond Hu. Parameterised Multiparty Session Types. *Logical Methods in Computer Science*, 8(4), 2012.
35. Mariangiola Dezani-Ciancaglini and Ugo de' Liguoro. Sessions and Session Types: an Overview. In Cosimo Laneve and Jianwen Su, editors, *WS-FM'09*, volume 6194 of *LNCS*, pages 1–28. Springer, 2010.
36. Mariangiola Dezani-Ciancaglini, Dimitris Mostrous, Nobuko Yoshida, and Sophia Drossopoulou. Session Types for Object-Oriented Languages. In Dave Thomas, editor, *ECOOP'06*, volume 4067 of *LNCS*, pages 328–352. Springer, 2006.
37. Luca Fossati, Raymond Hu, and Nobuko Yoshida. Multiparty Session Nets. In Matteo Maffei and Emilio Tuosto, editors, *TGC'14*, volume 8902 of *LNCS*, pages 112–127. Springer, 2014.
38. Anders Henriksen, Lasse Nielsen, Thomas Hildebrandt, Nobuko Yoshida, , and Fritz Henglein. Trustworthy Pervasive Healthcare Services via Multi-party Session Type. In Jens Weber and Isabelle Perseil, editors, *FHIES'12*, volume 7789 of *LNCS*, pages 124–141, 2013.
39. Kohei Honda. Types for Dyadic Interaction. In Eike Best, editor, *CONCUR'93*, volume 715 of *LNCS*, pages 509–523. Springer, 1993.
40. Kohei Honda, Raymond Hu, Rumyana Neykova, Tzu-Chun Chen, Romain Demangeon, Pierre-Malo Deniélou, and Nobuko Yoshida. Structuring Communication with Session Types. In Gul A. Agha, Atsushi Igarashi, Naoki Kobayashi, Hidehiko Masuhara, Satoshi Matsuoka, Etsuya Shibayama, and Kenjiro Taura, editors, *COB'14*, volume 8665 of *LNCS*, pages 105–127. Springer, 2014.
41. Kohei Honda, Aybek Mukhamedov, Gary Brown, Tzu-Chun Chen, and Nobuko Yoshida. Scribbling Interactions with a Formal Foundation. In Raja Natarajan and Adegboyega K. Ojo, editors, *ICDCIT'11*, volume 6536 of *LNCS*, pages 55–75. Springer, 2011.
42. Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language Primitives and Type Disciplines for Structured Communication-based Programming. In Chris Hankin, editor, *ESOP'98*, volume 1381 of *LNCS*, pages 22–138. Springer, 1998.
43. Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty Asynchronous Session Types. In George C. Necula and Philip Wadler, editors, *POPL'08*, pages 273–284. ACM Press, 2008.
44. Raymond Hu, Rumyana Neykova, Nobuko Yoshida, and Romain Demangeon. Practical Interruptible Conversations: Distributed Dynamic Verification with Session Types and Python. In Axel Legay and Saddek Bensalem, editors, *RV'13*, volume 8174 of *LNCS*, pages 148–130. Springer, 2013.
45. Hans Hüttel, Ivan Lanese, Vasco T. Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniélou, Dimitris Mostrous, Luca Padovani, António Ravara, Emilio Tuosto, Hugo Torres

Vieira, and Gianluigi Zavattaro. Foundations of Behavioural Types, 2014. Submitted for journal publication.
46. Dimitrios Kouzapas and Nobuko Yoshida. Globally Governed Session Semantics. *Logical Methods in Computer Science*, 10, 2015.
47. Dimitrios Kouzapas, Nobuko Yoshida, Raymond Hu, and Kohei Honda. On Asynchronous Eventful Session Semantics. *Mathematical Structures in Computer Science*, 29:1–62, 2015.
48. Julien Lange and Emilio Tuosto. Synthesising Choreographies from Local Session Types. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR'12*, volume 7454 of *LNCS*, pages 225–239. Springer, 2012.
49. Julien Lange, Emilio Tuosto, and Nobuko Yoshida. From Communicating Machines to Graphical Choreographies. In Sriram K. Rajamani and David Walker, editors, *POPL'15*, pages 221–232. ACM Press, 2015.
50. Robin Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.
51. Fabrizio Montesi and Nobuko Yoshida. Compositional Choreographies. In Pedro R. D'Argenio and Hernán C. Melgratti, editors, *CONCUR'13*, volume 8052 of *LNCS*, pages 439–425. Springer, 2013.
52. Dimitris Mostrous, Nobuko Yoshida, and Kohei Honda. Global Principal Typing in Partially Commutative Asynchronous Sessions. In Giuseppe Castagna, editor, *ESOP'09*, volume 5502 of *LNCS*, pages 316–332. Springer, 2009.
53. Rumyana Neykova, Laura Bocchi, and Nobuko Yoshida. Timed Runtime Monitoring for Multiparty Conversations. In Marco Carbone, editor, *BEAT'14*, volume 162 of *EPTCS*, pages 19–26, 2014.
54. Rumyana Neykova and Nobuko Yoshida. Multiparty Session Actors. In Eva Kühn and Rosario Pugliese, editors, *COORDINATION'14*, volume 8459 of *LNCS*. Springer, 2014.
55. Rumyana Neykova, Nobuko Yoshida, and Raymond Hu. SPY: Local Verification of Global Protocols. In Axel Legay and Saddek Bensalem, editors, *RV'13*, volume 8174 of *LNCS*, pages 363–358. Springer, 2013.
56. Nicholas Ng, Jose G.F. Coutinho, and Nobuko Yoshida. Protocols by Default: Safe MPI Code Generation based on Session Types. In Björn Franke, editor, *CC'15*, LNCS. Springer, 2015.
57. Nicholas Ng and Nobuko Yoshida. Pabble: Parameterised Scribble. *Service Oriented Computing and Applications*, pages 1–16, 2014.
58. Nicholas Ng, Nobuko Yoshida, and Kohei Honda. Multiparty Session C: Safe Parallel Programming with Message Optimisation. In Carlo A. Furia and Sebastian Nanz, editors, *TOOLS'12*, volume 7304 of *LNCS*, pages 202–218. Springer, 2012.
59. Nicholas Ng, Nobuko Yoshida, and Wayne Luk. Scalable Session Programming for Heterogeneous High-Performance Systems. In Steve Counsell and Manuel Núñez, editors, *SEFM'13*, volume 8368 of *LNCS*, pages 82–98. Springer, 2013.
60. Nicholas Ng, Nobuko Yoshida, Xin Yu Niu, Kuen Hung Tsoi, and Wayne Luk. Session Types: Towards Safe and Fast Reconfigurable Programming. *SIGARCH CAN*, 40:22–27, 2012.
61. Lasse Nielsen, Nobuko Yoshida, and Kohei Honda. Multiparty Symmetric Sum Types. In Sibylle B. Fröschle and Frank D. Valencia, editors, *EXPRESS'10*, volume 41 of *EPTCS*, pages 121–135, 2010.
62. Ocean Observatories Initiative, 2010. `http://www.oceanleadership.org/programs-and-partnerships/ocean-observing/ooi/`.
63. Luca Padovani. Deadlock and Lock Freedom in the Linear π-Calculus. In Thomas A. Henzinger and Dale Miller, editors, *CSL-LICS'14*, pages 72:1–72:10. ACM Press, 2014. Extended technical report available at `http://hal.archives-ouvertes.fr/hal-00932356v2/document`.

64. Luca Padovani. Fair Subtyping for Multi-Party Session Types. *Mathematical Structures in Computer Science*, pages 1–41, 2014.
65. Benjamin Pierce and Davide Sangiorgi. Typing and Subtyping for Mobile Processes. *Journal of Mathematical Structures in Computer Science*, 6(5):409–454, 1996.
66. Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002.
67. Jérémy Planul, Ricardo Corin, and Cédric Fournet. Secure Enforcement for Global Process Specifications. In Mario Bravetti and Gianluigi Zavattaro, editors, *CONCUR'09*, volume 5710 of *LNCS*, pages 511–526. Springer, 2009.
68. Savara. SAVARA JBoss RedHat Project, 2010. `http://www.jboss.org/savara`.
69. Scribble. Scribble JBoss RedHat Project, 2008. `http://www.jboss.org/scribble`.
70. K. C. Sivaramakrishnan, Karthik Nagaraj, Lukasz Ziarek, and Patrick Eugster. Efficient Session Type Guided Distributed Interaction. In Dave Clarke and Gul A. Agha, editors, *COORDINATION'10*, volume 6116 of *LNCS*, pages 152–167. Springer, 2010.
71. Nikhil Swamy, Juan Chen, Cédric Fournet, Pierre-Yves Strub, Karthikeyan Bhargavan, and Jean Yang. Secure Distributed Programming with Value-Dependent Types. In Manuel M. T. Chakravarty, Zhenjiang Hu, and Olivier Danvy, editors, *ICFP'11*, pages 266–278. ACM Press, 2011.
72. UNIFI. International Organization for Standardization ISO 20022 UNIversal Financial Industry message scheme, 2002. `http://www.iso20022.org`.
73. Web Services Choreography Working Group. Web Services Choreography Description Language. http://www.w3.org/2002/ws/chor/, 2002.
74. Nobuko Yoshida. Graph Types for Monadic Mobile Processes. In Vijay Chandru and V. Vinay, editors, *FSTTCS'96*, volume 1180 of *LNCS*, pages 371–386. Springer, 1996.
75. Nobuko Yoshida, Raymond Hu, Rumyana Neykova, and Nicholas Ng. The Scribble Protocol Language. In Martín Abadi and Alberto Lluch-Lafuente, editors, *TGC'13*, volume 8358 of *LNCS*, pages 22–41. Springer, 2013.

A Properties of the Communication Type System

This appendix completes the description of the communication type system given in §4. Auxiliary lemmas, in particular inversion lemmas, are the content of §A.1. Lastly §A.2 proves subject reduction.

A.1 Auxiliary Lemmas

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

Lemma 2 (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* $\Gamma \vdash e_1 \text{ and } e_2 : S$, *then* $\Gamma \vdash e_1 : \mathsf{bool}$ *and* $\Gamma \vdash e_2 : \mathsf{bool}$ *and* $S = \mathsf{bool}$.
5. *If* $\Gamma \vdash \overline{a}[\mathtt{p}](y).P \triangleright \Delta$, *then* $\Gamma \vdash a : \mathsf{G}$ *and* $\Gamma \vdash P \triangleright \Delta, y : \mathsf{G} \upharpoonright \mathtt{p}$ *and* $\mathtt{p} = \mathrm{mp}(\mathsf{G})$.
6. *If* $\Gamma \vdash a[\mathtt{p}](y).P \triangleright \Delta$, *then* $\Gamma \vdash a : \mathsf{G}$ *and* $\Gamma \vdash P \triangleright \Delta, y : \mathsf{G} \upharpoonright \mathtt{p}$ *and* $\mathtt{p} < \mathrm{mp}(\mathsf{G})$.
7. *If* $\Gamma \vdash c!\langle \mathtt{p}, e\rangle.P \triangleright \Delta$, *then* $\Delta = \Delta', c : !\langle \mathtt{p}, S\rangle.T$ *and* $\Gamma \vdash e : S$ *and* $\Gamma \vdash P \triangleright \Delta', c : T$.
8. *If* $\Gamma \vdash c?(\mathtt{q}, x).P \triangleright \Delta$, *then* $\Delta = \Delta', c : ?(\mathtt{q}, S).T$ *and* $\Gamma, x : S \vdash P \triangleright \Delta', c : T$.
9. *If* $\Gamma \vdash c!\langle\langle \mathtt{p}, c'\rangle\rangle.P \triangleright \Delta$, *then* $\Delta = \Delta', c : !\langle \mathtt{p}, \mathsf{T}\rangle.T, c' : \mathsf{T}$ *and* $\Gamma \vdash P \triangleright \Delta', c : T$.

10. *If* $\Gamma \vdash c?((\mathsf{q},y)).P \triangleright \Delta$, *then* $\Delta = \Delta', c:?(\mathsf{q},\mathsf{T}).T$ *and* $\Gamma \vdash P \triangleright \Delta', c:T, y:\mathsf{T}$.
11. *If* $\Gamma \vdash c \oplus \langle \mathsf{p}, l_j \rangle .P \triangleright \Delta$, *then* $\Delta = \Delta', c:\oplus\langle \mathsf{p}, \{l_i : T_i\}_{i\in I}\rangle$ *and* $\Gamma \vdash P \triangleright \Delta', c:T_j$ *and* $j \in I$.
12. *If* $\Gamma \vdash c\&(\mathsf{p}, \{l_i : P_i\}_{i\in I}) \triangleright \Delta$, *then* $\Delta = \Delta', c:\&(\mathsf{p}, \{l_i : T_i\}_{i\in I})$ *and* $\Gamma \vdash P_i \triangleright \Delta', c:T_i$ $\forall i \in I$.
13. *If* $\Gamma \vdash P \mid Q \triangleright \Delta$, *then* $\Delta = \Delta', \Delta''$ *and* $\Gamma \vdash P \triangleright \Delta'$ *and* $\Gamma \vdash Q \triangleright \Delta''$.
14. *If* $\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta$, *then* $\Gamma \vdash e : \mathsf{bool}$ *and* $\Gamma \vdash P \triangleright \Delta$ *and* $\Gamma \vdash Q \triangleright \Delta$.
15. *If* $\Gamma \vdash \mathbf{0} \triangleright \Delta$, *then* Δ end *only*.
16. *If* $\Gamma \vdash (\nu a)P \triangleright \Delta$, *then* $\Gamma, a : \mathsf{G} \vdash P \triangleright \Delta$.
17. *If* $\Gamma \vdash X\langle e, c\rangle \triangleright \Delta$, *then* $\Gamma = \Gamma', X : S\ T$ *and* $\Delta = \Delta', c : T$ *and* $\Gamma \vdash e : S$ *and* Δ' end *only*.
18. *If* $\Gamma \vdash \mathsf{def}\ X(x,y) = P\ \mathsf{in}\ Q \triangleright \Delta$, *then* $\Gamma, X : S\,\boldsymbol{t}, x : S \vdash P \triangleright \{y : T\}$ *and* $\Gamma, X : S\ \mu\boldsymbol{t}.T \vdash Q \triangleright \Delta$.

Lemma 3 (Inversion Lemma for Processes).

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

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

Lemma 4. 1. *If* $\Gamma \vdash_{\{s\}} s : h_1 \cdot (\mathsf{q},\mathsf{p},v) \cdot h_2 \triangleright \Delta$, *then* $\Delta = \Delta_1 * \{s[\mathsf{q}] : !\langle \mathsf{p}, S\rangle\} * \Delta_2$ *and* $\Gamma \vdash_{\{s\}} s : h_i \triangleright \Delta_i$ *(*$i = 1,2$*) and* $\Gamma \vdash v : S$.
Vice versa $\Gamma \vdash_{\{s\}} s : h_i \triangleright \Delta_i$ *(*$i = 1,2$*) and* $\Gamma \vdash v : S$ *imply*
$$\Gamma \vdash_{\{s\}} s : h_1 \cdot (\mathsf{q},\mathsf{p},v) \cdot h_2 \triangleright \Delta_1 * \{s[\mathsf{q}] : !\langle \mathsf{p}, S\rangle\} * \Delta_2.$$
2. *If* $\Gamma \vdash_{\{s\}} s : h_1 \cdot (\mathsf{q},\mathsf{p},s'[\mathsf{p}']) \cdot h_2 \triangleright \Delta$, *then* $\Delta = (\Delta_1 * \{s[\mathsf{q}] : !\langle \mathsf{p}, \mathsf{T}\rangle\} * \Delta_2), s'[\mathsf{p}'] : \mathsf{T}$ *and* $\Gamma \vdash_{\{s\}} s : h_i \triangleright \Delta_i$ *(*$i = 1,2$*).*
Vice versa $\Gamma \vdash_{\{s\}} s : h_i \triangleright \Delta_i$ *(*$i = 1,2$*) imply*
$$\Gamma \vdash_{\{s\}} s : h_1 \cdot (\mathsf{q},\mathsf{p},s'[\mathsf{p}']) \cdot h_2 \triangleright (\Delta_1 * \{s[\mathsf{q}] : !\langle \mathsf{p}, \mathsf{T}\rangle\} * \Delta_2), s'[\mathsf{p}'] : \mathsf{T}.$$
3. *If* $\Gamma \vdash_{\{s\}} s : h_1 \cdot (\mathsf{q},\mathsf{p},l) \cdot h_2 \triangleright \Delta$, *then* $\Delta = \Delta_1 * \{s[\mathsf{q}] : \oplus\langle \mathsf{p}, l\rangle\} * \Delta_2$ *and* $\Gamma \vdash_{\{s\}} s : h_i \triangleright \Delta_i$ *(*$i = 1,2$*).*
Vice versa $\Gamma \vdash_{\{s\}} s : h_i \triangleright \Delta_i$ *(*$i = 1,2$*) imply*
$$\Gamma \vdash_{\{s\}} s : h_1 \cdot (\mathsf{q},\mathsf{p},l) \cdot h_2 \triangleright \Delta_1 * \{s[\mathsf{q}] : \oplus\langle \mathsf{p}, l\rangle\} * \Delta_2.$$

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

Lemma 5 (Substitution lemma).

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

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

Theorem 2 (Type Preservation under Equivalence). *If* $\Gamma \vdash_\Sigma P \triangleright \Delta$ *and* $P \equiv P'$*, then* $\Gamma \vdash_\Sigma P' \triangleright \Delta$.

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

- $P \mid \mathbf{0} \equiv P$. First we assume $\Gamma \vdash_\Sigma P \triangleright \Delta$. From $\Gamma \vdash_\emptyset \mathbf{0} \triangleright \emptyset$ by applying (GPAR) to these two sequents we obtain $\Gamma \vdash_\Sigma P|\mathbf{0} \triangleright \Delta$.
  For the converse direction assume $\Gamma \vdash_\Sigma P|\mathbf{0} \triangleright \Delta$. Using 3(7) we obtain: $\Gamma \vdash_{\Sigma_1} P \triangleright \Delta_1$, $\Gamma \vdash_{\Sigma_2} \mathbf{0} \triangleright \Delta_2$, where $\Delta = \Delta_1 * \Delta_2$, $\Sigma = \Sigma_1 \cup \Sigma_2$ and $\Sigma_1 \cap \Sigma_2 = \emptyset$. Using 3(1) we get $\Sigma_2 = \emptyset$, which implies $\Sigma = \Sigma_1$, and $\Gamma \vdash \mathbf{0} \triangleright \Delta_2$. Using 2(15) we get Δ_2 end only which implies $\Delta_1 \approx \Delta_1 * \Delta_2$, so we conclude $\Gamma \vdash_\Sigma P \triangleright \Delta_1 * \Delta_2$ by applying (EQUIV).
- $P \mid Q \equiv Q \mid P$. By the symmetry of the rule we have to show only one direction. Suppose $\Gamma \vdash_\Sigma P \mid Q \triangleright \Delta$. Using 3(7) we obtain $\Gamma \vdash_{\Sigma_1} P \triangleright \Delta_1$, $\Gamma \vdash_{\Sigma_2} Q \triangleright \Delta_2$, where $\Delta = \Delta_1 * \Delta_2$, $\Sigma = \Sigma_1 \cup \Sigma_2$ and $\Sigma_1 \cap \Sigma_2 = \emptyset$. Using (GPAR) we get $\Gamma \vdash_\Sigma Q \mid P \triangleright \Delta_2 * \Delta_1$. Thanks to the commutativity of $*$, we get $\Delta_2 * \Delta_1 = \Delta$ and so we are done.
- $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$. Suppose $\Gamma \vdash_\Sigma P \mid (Q \mid R) \triangleright \Delta$. Using 3(7) we obtain $\Gamma \vdash_{\Sigma_1} P \triangleright \Delta_1$, $\Gamma \vdash_{\Sigma_2} Q \mid R \triangleright \Delta_2$, where $\Delta = \Delta_1 * \Delta_2$, $\Sigma = \Sigma_1 \cup \Sigma_2$ and $\Sigma_1 \cap \Sigma_2 = \emptyset$. Using 3(7) we obtain $\Gamma \vdash_{\Sigma_{21}} Q \triangleright \Delta_{21}$, $\Gamma \vdash_{\Sigma_{22}} R \triangleright \Delta_{22}$ where $\Delta_2 = \Delta_{21} * \Delta_{22}$, $\Sigma_2 = \Sigma_{21} \cup \Sigma_{22}$ and $\Sigma_{21} \cap \Sigma_{22} = \emptyset$. Using (GPAR) we get $\Gamma \vdash_{\Sigma_1 \cup \Sigma_{21}} P \mid Q \triangleright \Delta_1 * \Delta_{21}$. Using (GPAR) again we get $\Gamma \vdash_\Sigma (P \mid Q) \mid R \triangleright \Delta_1 * \Delta_{21} * \Delta_{22}$ and so we are done by the associativity of $*$. The proof for the other direction is similar.
- $s : h_1 \cdot (\mathtt{q}, \mathtt{p}, v) \cdot (\mathtt{q}', \mathtt{p}', v') \cdot h_2 \equiv s : h_1 \cdot (\mathtt{q}', \mathtt{p}', v') \cdot (\mathtt{q}, \mathtt{p}, v) \cdot h_2$ where $\mathtt{p} \neq \mathtt{p}'$ or $\mathtt{q} \neq \mathtt{q}'$. We assume $\mathtt{p} \neq \mathtt{p}'$ and $\mathtt{q} = \mathtt{q}'$, the proof in the case $\mathtt{q} \neq \mathtt{q}'$ being similar and simpler. If $\Gamma \vdash_\Sigma s : h_1 \cdot (\mathtt{q}, \mathtt{p}, v) \cdot (\mathtt{q}, \mathtt{p}', v') \cdot h_2 \triangleright \Delta$, then $\Sigma = \{s\}$ by Lemma 3(2). This implies $\Delta = \Delta_1 * \{s[\mathtt{q}] : !\langle \mathtt{p}, S\rangle;\ !\langle \mathtt{p}', S'\rangle\} * \Delta_2$ and $\Gamma \vdash_{\{s\}} s : h_i \triangleright \Delta_i$ $(i = 1, 2)$ and $\Gamma \vdash v : S$ and $\Gamma \vdash v' : S'$ by Lemma 4(1). By the same lemma we can derive
  $$\Gamma \vdash_{\{s\}} s : h_1 \cdot (\mathtt{q}, \mathtt{p}', v') \cdot (\mathtt{q}, \mathtt{p}, v) \cdot h_2 \triangleright \Delta_1 * \{s[\mathtt{q}] : !\langle \mathtt{p}', S'\rangle;\ !\langle \mathtt{p}, S\rangle\} * \Delta_2,$$
  and we conclude using rule (EQUIV), since by definition
  $$\Delta_1 * \{s[\mathtt{q}] : !\langle \mathtt{p}', S'\rangle;\ !\langle \mathtt{p}, S\rangle\} * \Delta_2 \approx \Delta.$$

□

A.2 Proof of Subject Reduction

We show the Main Lemma first and then the Subject Reduction Theorem.

Lemma 1 (Main Lemma). *Let $\Gamma \vdash_\Sigma P \triangleright \Delta$, and $P \longrightarrow P'$ be obtained by any reduction rule different from [Ctxt], [Str], and $\Delta * \Delta_0$ be consistent, for some Δ_0. Then there is Δ' such that $\Gamma \vdash_\Sigma P' \triangleright \Delta'$ and $\Delta \Rightarrow^* \Delta'$ and $\Delta' * \Delta_0$ is consistent.*

Proof. The proof is by cases on process reduction rules. We only consider some paradigmatic cases.

– [Init] $a[1](y).P_1 \mid ... \mid \overline{a}[n](y).P_n \longrightarrow (\nu s)(P_1\{s[1]/y_1\} \mid ... \mid P_n\{s[n]/y\} \mid s : \varnothing)$. By hypothesis $\Gamma \vdash_\Sigma a[1](y).P_1 \mid a[2](y_2).P_2 \mid \ldots \mid \overline{a}[n](y).P_n \triangleright \Delta$; then, since the redex is a pure process, $\Sigma = \emptyset$ and $\Gamma \vdash a[1](y).P_1 \mid a[2](y_2).P_2 \mid \ldots \mid \overline{a}[n](y).P_n \triangleright \Delta$ by Lemma 3(1). Using Lemma 2(13) on all the processes in parallel we have

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

where $\Delta = \bigcup_{i=1}^n \Delta_i$. Using Lemma 2(6) on (1) we have

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

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

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

and $\mathrm{mp}(\mathsf{G}) = n$. Using Lemma 5(2) on (4) and (3) we have

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

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

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

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

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

Using (GSRES) on (7) we conclude

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

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

– [Send] $s[\mathtt{p}]!\langle \mathtt{q}, e\rangle.P \mid s : h \longrightarrow P \mid s : h \cdot (\mathtt{p}, \mathtt{q}, v)$ $(e \downarrow v)$.
By hypothesis, $\Gamma \vdash_\Sigma s[\mathtt{p}]!\langle \mathtt{q}, e\rangle.P \mid s : h \triangleright \Delta$. Using Lemma 3(7), (1), and (2) we have $\Sigma = \{s\}$ and

$$\Gamma \vdash s[\mathtt{p}]!\langle \mathtt{q}, e\rangle.P \triangleright \Delta_1 \tag{8}$$
$$\Gamma \vdash_{\{s\}} s : h \triangleright \Delta_2 \tag{9}$$

where $\Delta = \Delta_2 * \Delta_1$. Using 2(7) on (8) we have

$$\Delta_1 = \Delta_1', s[\mathtt{p}] : !\langle \mathtt{q}, S\rangle.T$$
$$\Gamma \vdash e : S \tag{10}$$
$$\Gamma \vdash P \triangleright \Delta_1', s[\mathtt{p}] : T. \tag{11}$$

From (10) by subject reduction on expressions we have

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

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

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

Using (GINIT) on (11) we derive

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

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

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

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

– [Rcv] $s[\mathtt{p}]?(\mathtt{q}, x).P \mid s : (\mathtt{q}, \{\mathtt{p}\}, v) \cdot h \longrightarrow P\{v/x\} \mid s : h$.
By hypothesis, $\Gamma \vdash_\Sigma s[\mathtt{p}]?(\mathtt{q}, x).P \mid s : (\mathtt{q}, \{\mathtt{p}\}, v) \cdot h \triangleright \Delta$. By Lemma 3(7), (1), and (2) we have $\Sigma = \{s\}$ and

$$\Gamma \vdash s[\mathtt{p}]?(\mathtt{q}, x).P \triangleright \Delta_1 \tag{15}$$
$$\Gamma \vdash_{\{s\}} s : (\mathtt{q}, \{\mathtt{p}\}, v) \cdot h \triangleright \Delta_2 \tag{16}$$

where $\Delta = \Delta_2 * \Delta_1$. Using Lemma 2(8) on (15) we have

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

Using Lemma 4(1) on (16) we have

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

The consistency of $\Delta * \Delta_0$ implies $S = S'$. Using Lemma 5(1) from (17) and (19) we get
$\Gamma \vdash P\{v/x\} \triangleright \Delta_1', s[\mathtt{p}] : T$, which implies by rule (GINIT)

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

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

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

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

– [Sel] $s[\mathtt{p}] \oplus \langle \mathtt{p}, l\rangle.P \mid s : h \longrightarrow P \mid s : h \cdot (\mathtt{p}, \mathtt{q}, l)$.
By hypothesis, $\Gamma \vdash_{\Sigma} s[\mathtt{p}] \oplus \langle \mathtt{q}, l\rangle.P \mid s : h \triangleright \Delta$. Using Lemma 3(7), (1), and (2) we have $\Sigma = \{s\}$ and

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

where $\Delta = \Delta_2 * \Delta_1$. Using Lemma 2(11) on (21) we have for $l = l_j$ $(j \in I)$:

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

Using rule (QSEL) on (22) we derive

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

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

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

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

– [Branch] $s[\mathtt{p}]\&(\mathtt{q}, \{l_i : P_i\}_{i\in I}) \mid s : (\mathtt{q}, \{\mathtt{p}\}, l_j) \cdot h \longrightarrow P_j \mid s : h$.
By hypothesis, $\Gamma \vdash_{\Sigma} s[\mathtt{p}]\&(\mathtt{q}, \{l_i : P_i\}_{i\in I}) \mid s : (\mathtt{q}, \{\mathtt{p}\}, l_j) \cdot h \triangleright \Delta$. Using Lemma 3(7), (1), and (2) we have $\Sigma = \{s\}$ and

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

where $\Delta = \Delta_2 * \Delta_1$. Using Lemma 2(12) on (25) we have

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

Using Lemma 4(3) on (26) we have

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

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

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

Note that

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

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

Theorem 1 (Subject Reduction). *If $\Gamma \vdash_\Sigma P \triangleright \Delta$ with Δ consistent and $P \longrightarrow^* P'$, then $\Gamma \vdash_\Sigma P' \triangleright \Delta'$ for some consistent Δ' such that $\Delta \;\Rightarrow^*\; \Delta'$.*

Proof. Let $P \equiv \mathscr{E}[P_0]$ and $P' \equiv \mathscr{E}[P_0']$, where $P_0 \longrightarrow P_0'$ by one of the rules considered in Lemma 1. By structural equivalence we can assume $\mathscr{E} = (\overrightarrow{va})(\overrightarrow{\mathsf{def}\ D\ \mathsf{in}})(\overrightarrow{vs})([\,] \mid P_1))$ without loss of generality. Theorem 2 and Lemma 3(9), (10) and (8) applied to $\Gamma \vdash_\Sigma P \triangleright \Delta$ give $\Gamma, \overrightarrow{a : \mathsf{G}}, \overrightarrow{X : S\,\mu\mathbf{t}.T} \vdash_{\Sigma_0} P_0 \triangleright \Delta_0$, and $\Gamma, \overrightarrow{a : \mathsf{G}}, \overrightarrow{X : S\,\mu\mathbf{t}.T} \vdash_{\Sigma_1} P_1 \triangleright \Delta_1$ and $\Gamma, \overrightarrow{a : \mathsf{G}}, X : S\,\mathbf{t} \vdash \overrightarrow{Q \triangleright \{y : T\}}$, where $\overrightarrow{D} = \overrightarrow{X(x,y) = Q}$, $\Sigma = (\Sigma_0 \cup \Sigma_1) \setminus \overrightarrow{s}$ and $\Delta = (\Delta_0 * \Delta_1) \setminus \overrightarrow{s}$. The consistency of Δ implies the consistency of $\Delta_0 * \Delta_1$ by Lemma 3(8). By Lemma 1 there is Δ_0' such that $\Gamma, \overrightarrow{a : \mathsf{G}}, \overrightarrow{X : S\,\mu\mathbf{t}.T} \vdash_{\Sigma_0} P_0' \triangleright \Delta_0'$ and $\Delta_0 \;\Rightarrow^*\; \Delta_0'$ and $\Delta_0' * \Delta_1$ is consistent. We derive $\Gamma \vdash_\Sigma P' \triangleright \Delta'$, where $\Delta' = (\Delta_0 * \Delta_1') \setminus \overrightarrow{s}$ by applying typing rules (GPAR), (GSRES), (GDEF) and (GNRES). Observe that $\Delta \;\Rightarrow^*\; \Delta'$ and Δ' is consistent. □