Resource Sharing via Capability-Based Multiparty Session Types

A. Laura Voinea, Ornela Dardha, and Simon J. Gay

School of Computing Science, University of Glasgow, United Kingdom
a.voinea.1@research.gla.ac.uk
{Ornela.Dardha, Simon.Gay}@glasgow.ac.uk

Abstract. *Multiparty Session Types (MPST)* are a type formalism used to model communication protocols among components in distributed systems, by specifying *type* and *direction* of data transmitted. It is standard for multiparty session type systems to use access control based on *linear* or *affine* types. While useful in offering strong guarantees of communication safety and session fidelity, linearity and affinity run into the well-known problem of inflexible programming, excluding scenarios that make use of shared channels or need to store channels into shared data structures.
In this paper, we develop *capability-based resource sharing* for multiparty session types. In this setting, channels are split into two entities, the channel itself and the capability of using it. This gives rise to a more flexible session type system, which allows channel references to be shared and stored in persistent data structures. We illustrate our type system through a producer-consumer case study. Finally, we prove that the resulting language satisfies type safety.

1 Introduction

In the present era of communication-centric software systems, it is increasingly recognised that the structure of communication is an essential aspect of system design. *(Multiparty) session types* [18,19,32] allow communication structures to be codified as type definitions in programming languages, which can be exploited by compilers, development environments and runtime systems, for compile-time analysis or runtime monitoring. A substantial and ever-growing literature on session types and, more generally, behavioural types [20] provides a rich theoretical foundation, now being applied to a range of programming languages [1,16].

Session type systems must control *aliasing* of the endpoints of communication channels, in order to avoid race conditions. If agents A and B both think they are running the client side of a protocol with the same server S, then a message sent by A advances the session state without B's knowledge, which interferes with B's attempt to run the protocol.

In order to eliminate aliasing and guarantee unique ownership of channel endpoints, most session type systems use strict *linear typing*. For more flexibility, some others use *affine typing*, which allows channels to be discarded, but they

still forbid aliasing. It is possible to allow a session-typed channel to become sharable in the special case in which the session type reaches a point which is essentially stateless. However, in such systems, channels are *linearly typed* for the most interesting parts of their lifetimes—we discuss these possibilities in § 6.

This leads us to our research questions:

Q1 *Are session types intrinsically related to linearity or affinity?*
Q2 *Can we define session type systems without linear types?*
Q3 *How can we check resource (channel) sharing and aliasing, to guarantee communication safety and session fidelity, i.e., type safety?*

The goal of this paper is to investigate questions **Q1**–**Q3**. To give a more flexible approach to resource sharing and access control, we propose a system of multiparty session types (MPST) that includes techniques from the Capability Calculus [11], and from alias types [36]. The key idea is to split a communication channel into two entities: (1) the channel itself, and (2) its usage *capability*. Both entities are first-class and can be referred to separately. Channels can now be shared, or stored in shared data structure, and aliasing is allowed. However, in order to guarantee communication safety and session fidelity, i.e., type safety, capabilities are used linearly.

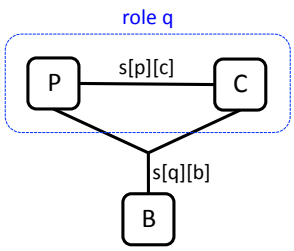

Fig. 1. A producer-consumer system.

This approach has several benefits, and improves on the state of the art: (i) for the first time, it is now possible for a system to have a communication structure defined by shared channels, with the capabilities being transferred from process to process as required; (ii) a capability can be implemented as a simple token, whereas delegation of channels requires a relatively complex implementation, thus making linearity of capabilities more lightweight than linearity of channels.

Example 1 (Producer-Consumer). Producer P and consumer C communicate via buffer B in session $\mathtt{s}$, in Fig. 1. P and C implement *role* $\mathtt{q}$, and B implements role $\mathtt{b}$. *Shared access* to B is captured by the fact that both P and C implement the same role $\mathtt{q}$ and use the same channel $\mathtt{s}[\mathtt{q}]$ to communicate with B.

Following MPST theory, we start by defining a *global type*, describing communications among *all* participants:

$$G_0 = \mathtt{q} \to \mathtt{b} : \mathsf{add}(\mathtt{Int}) \;.\; \mathtt{q} \to \mathtt{b} : \mathsf{request}() \;.\; \mathtt{b} \to \mathtt{q} : \mathsf{send}(\mathtt{Int}) \;.\; G_0$$

In G_0, protocol proceeds as follows: P (playing $\mathtt{q}$) sends an add message to B (playing $\mathtt{b}$), to add data. In sequence, C (playing $\mathtt{q}$) sends a request message to B, to ask for data. B replies with a send message, sending data to C (playing $\mathtt{q}$), and the protocol repeats as G_0.

Projecting the global protocol to each role gives us a local session type. In particular, for B (role $\mathtt{b}$) we obtain:

$$S_b = \mathtt{q}?\mathsf{add}\,(\mathtt{Int})\,.\;\mathtt{q}?\mathsf{request}\,()\,.\;\mathtt{q}!\mathsf{send}\,(\mathtt{Int})\,.\;S_b$$

and for the shared access by P and C (role $\mathtt{q}$), we obtain:

$$S_q = \mathtt{b}!\mathsf{add}\,(\mathtt{Int})\,.\;S'_q \qquad\qquad S'_q = \mathtt{b}!\mathsf{request}\,()\,.\;\mathtt{b}?\mathsf{send}\,(\mathtt{Int})\,.\;S_q$$

Finally, the definitions of processes are as follows—we will detail the syntax in § 2

$$\begin{aligned}\mathsf{P}\langle v\rangle &= \mathtt{s}[\mathtt{q}][\mathtt{b}]\oplus\langle\mathsf{add}(v)\rangle.\mathsf{P}\langle v+1\rangle\\ \mathsf{C}\langle\rangle &= \mathtt{s}[\mathtt{q}][\mathtt{b}]\oplus\langle\mathsf{request}(r)\rangle.\mathtt{s}[\mathtt{b}][\mathtt{q}]\&\{\mathsf{send(i)}\}.\mathsf{C}\langle\rangle\\ \mathsf{B}\langle\rangle &= \mathtt{s}[\mathtt{q}][\mathtt{b}]\&\{\mathsf{add}(x)\}.\mathtt{s}[\mathtt{q}][\mathtt{b}]\&\{\mathsf{request}()\}.\mathtt{s}[\mathtt{b}][\mathtt{q}]\oplus\langle\mathsf{send}(x)\rangle.\mathsf{B}\langle\rangle\end{aligned}$$

Unfortunately, the above system of processes is not typable using standard multiparty session type systems because role $\mathtt{q}$ is shared by P and C, thus violating linearity of channel $\mathtt{s}[\mathtt{q}]$.

To solve this issue and still allow sharing and aliasing, in our work instead of associating a channel c with a session type S, we associate separately c with a *tracked* type $\mathtt{tr}\,(\rho)$, and S with *capability* ρ, $\{\rho \mapsto S\}$. The capability can be passed between P and C as they take turns in using the channel. As a first attempt, we now define the following global type, getting us closer to our framework.

$$\begin{aligned}G_1 = \;&\mathtt{q} \to \mathtt{b} : \mathsf{add}(\mathtt{Int})\;.\;\mathtt{p} \to \mathtt{c} : \mathsf{turn}(\mathtt{tr}\,(\rho_q)).\\ &\mathtt{q} \to \mathtt{b} : \mathsf{request}()\;.\;\mathtt{b} \to \mathtt{q} : \mathsf{send}(\mathtt{Int})\;.\\ &\mathtt{c} \to \mathtt{p} : \mathsf{turn}(\mathtt{tr}\,(\rho_q)).\;G_1\end{aligned}$$

However, a type such as $\mathtt{tr}\,(\rho_q)$ is usually too specific because it refers to a particular channel's capability. It is preferable to be able to give definitions that abstract away from specific channels. We therefore introduce *existential types* which package a channel with its capability, in the form $\exists[\rho|\{\rho \mapsto S\}].\mathtt{tr}\,(\rho)$.

Finally, we can define our global type in the following way. It now includes an extra initial message from P to C containing the channel used with the buffer. The session types S_q and S'_q are the same as before.

$$\begin{aligned}G = \;&\mathtt{p} \to \mathtt{c} : \mathsf{buffer}\big(\exists[\rho_q|\{\rho_q \mapsto S'_q\}].\mathtt{tr}\,(\rho_q)\big).\mu\,\mathtt{t}.\mathtt{q} \to \mathtt{b} : \mathsf{add}(\mathtt{Int}).\\ &\mathtt{p} \to \mathtt{c} : \mathsf{turn}(\{\rho_q \mapsto S'_q\}).\mathtt{q} \to \mathtt{b} : \mathsf{request}().\mathtt{b} \to \mathtt{q} : \mathsf{send}(\mathtt{Int}).\\ &\mathtt{c} \to \mathtt{p} : \mathsf{turn}(\{\rho_q \mapsto S_q\}).\mathtt{t}\end{aligned}$$

In § 4 we complete this example by showing the projections to a local type for each role, and the definitions of processes that implement each role.

1.1 Contributions of the paper

(i) **MPST with capabilities**: we present a new version of MPST theory without linear typing for channels, but with linearly-typed capabilities. In § 2 we define a multiparty session π-calculus with capabilities, and its operational semantics, and in § 3 we define a MPST system for it.

$$
\begin{array}{lll}
P & ::= \mathbf{0} \mid P \mid Q \mid (\nu\, s)P & \text{inaction, parallel composition, restriction} \\
& \mid \mathsf{c}[\mathsf{p}] \oplus \langle l(v) \rangle . P \mid \mathsf{c}[\mathsf{p}] \& _{i \in I} \{ l_i(x_i).\, P_i \} & \text{select, branch} \\
& \mid \mathsf{c}[\mathsf{p}] \oplus \langle l(\mathsf{pack}(\rho, \mathsf{s}[\mathsf{q}])) \rangle . P & \text{select pack} \\
& \mid \mathsf{c}[\mathsf{p}] \& _{i \in I} \{ l_i(\mathsf{pack}(\rho_i, \mathsf{s}_\mathsf{i}[\mathsf{q}])).\, P_i \} & \text{branch pack} \\
& \mid \mathsf{def}\, D \,\mathsf{in}\, P \mid X\langle \tilde{x} \rangle & \text{recursion, process call} \\
\\
D & ::= X\langle \tilde{x} \rangle = P & \text{process declaration} \\
\\
\mathsf{c} & ::= x \mid \mathsf{s}[\mathsf{p}] & \text{variable, channel with role p} \\
v & ::= \mathsf{c} \mid \rho \mid \mathsf{true} \mid \mathsf{false} \mid 0 \mid 1 \mid \ldots & \text{channel, capability, base value}
\end{array}
$$

Fig. 2. Multiparty session π-calculus

(ii) **Producer-Consumer Case Study**: in § 4 we present a detailed account of the producer-consumer case study, capturing the core of resource sharing and use of capabilities.

(iii) **Type Safety**: in § 5 we state the type safety property, and outline its proof.

2 Multiparty Session π-Calculus with Capabilities

Our π-calculus with multiparty session types is based on the language defined by Scalas *et al.* [29]. The syntax is defined in Fig. 2. We assume infinite sets of identifiers for variables (x), sessions ($\mathtt{s}$), capabilities (ρ) and roles ($\mathtt{p}$).

The calculus combines branch (resp., select) with input (resp., output), and a message $l(v)$ consists of a label l and a payload v, which is a value. A message in session $\mathtt{s}$ from role $\mathtt{p}$ to role $\mathtt{q}$ has the prefix $\mathtt{s}[\mathtt{p}][\mathtt{q}]$, where $\mathtt{s}[\mathtt{p}]$ is represented by $\mathtt{c}$ in the grammar. The select and branch operations come in two forms. The first form is standard, and the second form handles *packages*, which are the novel feature of our type system. A package consists of a capability ρ and a channel of type $\mathtt{tr}\,(\rho)$. We will see in § 3 in the typing rules, the capability is existentially quantified. This enables a channel to be delegated, with the information that it is linked to *some* capability, which will be transmitted in a second instance.

As usual, we define structural congruence to compensate for the limitations of textual syntax. It is the smallest congruence satisfying the axioms in Fig. 3. The definition uses the concepts of *free channels* of a process, $fc(P)$; *free process variables* of a process, $fpv(P)$; and *defined process variables* of a process declaration, $dpv(D)$. We omit the definitions of these concepts, which are standard and can be found in [29].

We define a reduction-based operational semantics by the rules in Fig. 4. Rule RCom is a standard communication between roles $\mathtt{p}$ and $\mathtt{q}$. Rule RComP is communication of an existential package. Rule RCall defines a standard approach to handling process definitions. The rest are standard contextual rules.

$$P \mid \mathbf{0} \equiv P \qquad P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P(Q \mid R)$$
$$(\nu\, s)\mathbf{0} \equiv \mathbf{0} \qquad (\nu\, s)(\nu\, s')P \equiv (\nu\, s')(\nu\, s)P$$
$$(\nu\, s)P \mid Q \equiv (\nu\, s)(P \mid Q) \quad \text{if } s \notin fc(Q)$$
$$\mathsf{def}\ D\ \mathsf{in}\ \mathbf{0} \equiv \mathbf{0} \qquad \mathsf{def}\ D\ \mathsf{in}\ (\nu\, s)P \equiv (\nu\, s)(\mathsf{def}\ D\ \mathsf{in}\ P) \quad \text{if } s \notin fc(Q)$$
$$\mathsf{def}\ D\ \mathsf{in}\ (P \mid Q) \equiv (\mathsf{def}\ D\ \mathsf{in}\ P) \mid Q \quad \text{if } dpv(D) \cap fpv(Q) = \emptyset$$
$$\mathsf{def}\ D\ \mathsf{in}\ \mathsf{def}\ D'\ \mathsf{in}\ P \equiv \mathsf{def}\ D'\ \mathsf{in}\ \mathsf{def}\ D\ \mathsf{in}\ P$$
$$\text{if } (dpv(D) \cup fpv(D)) \cap dpv(D') = dpv(D') \cup fpv(D')) \cap dpv(D) = \emptyset$$

Fig. 3. Structural congruence (processes)

$$\frac{j \in I \text{ and } fv(v) = \emptyset}{\mathtt{s}[\mathtt{p}][\mathtt{q}]\&_{i\in I}\{l_i(x_i).\, P_i\} \mid \mathtt{s}[\mathtt{q}][\mathtt{p}]\oplus\langle l_j(v)\rangle.P \longrightarrow P_j\{v/x_j\} \mid P}\ \textsc{RCom}$$
$$\frac{j \in I}{\mathtt{s}[\mathtt{p}][\mathtt{q}]\&_{i\in I}\{l_i(\mathsf{pack}(\rho_i, \mathtt{v}_i)).\, P_i\} \mid \mathtt{s}[\mathtt{q}][\mathtt{p}]\oplus\langle l_j(\mathsf{pack}(\rho, \mathtt{v}))\rangle.P \longrightarrow P_j\{\mathtt{v}/\mathtt{v}_i\} \mid P}\ \textsc{RComP}$$
$$\frac{\widetilde{x} = x_1, \ldots, x_n \qquad \widetilde{v} = v_1, \ldots, v_n \qquad fv(\widetilde{v}) = \emptyset}{\mathsf{def}\ X\langle\widetilde{x}\rangle = P\ \mathsf{in}\ (X\langle\widetilde{x}\rangle \mid Q) \longrightarrow \mathsf{def}\ X\langle\widetilde{x}\rangle = P\ \mathsf{in}\ (P\{\widetilde{v}/\widetilde{x}\} \mid Q)}\ \textsc{RCall}$$
$$\frac{P \longrightarrow Q}{(\nu\, s)P \longrightarrow (\nu\, s)Q}\ \textsc{RRes} \qquad \frac{P \longrightarrow Q}{P \mid R \longrightarrow Q \mid R}\ \textsc{RPar}$$
$$\frac{P \longrightarrow Q}{\mathsf{def}\ D\ \mathsf{in}\ P \longrightarrow \mathsf{def}\ D\ \mathsf{in}\ Q}\ \textsc{RDef}$$

Fig. 4. Reduction (processes)

3 Multiparty Session Types

We now introduce a type system for the multiparty session π-calculus. The general methodology of multiparty session types is that system design begins with a *global type*, which specifies all of the communication among various *roles*. Given a global type G and a role $\mathtt{p}$, *projection* yields a *session type* or *local type* $G \upharpoonright \mathtt{p}$ that describes all of the communication involving $\mathtt{p}$. This local type can be further projected for another role $\mathtt{q}$, to give a *partial session type* that describes communication between $\mathtt{p}$ and $\mathtt{q}$.

Global types Fig. 5. Each interaction has a source role $\mathtt{p}$ and a target role $\mathtt{q}$. We combine branching and message transmission, so an interaction has a label l_i, a payload of type U_i, and a continuation type G_i. If there is only one branch then we usually abbreviate the syntax to $\mathtt{p} \to \mathtt{q} : l(U).\ G$. The second form of interaction deals with existential packages. Recursive types are allowed, with the assumption that they are guarded. **Base types** $B, B', \ldots$ can be types like `Bool`, `Int`, etc. **Payload types** $U, U_i, \ldots$ are either base types, tracked types, capability types or *closed* session types.

S	$::=$		*local session type*
	$\mathtt{end}$		terminated session
$\mid$	$\mathtt{p}\oplus_{i\in I}!l_i\,(U_i)\,.S_i$		selection towards role p
$\mid$	$\mathtt{p}\&_{i\in I}?l_i\,(U_i)\,.S_i$		branching from role p
$\mid$	$\mathtt{p}\,\oplus_{i\in I}\,!l_i(\exists[\rho_i\|\{\rho_i\mapsto U_i\}].\mathtt{tr}\,(\rho_i)).S_i$		pack selection towards role p
$\mid$	$\mathtt{p}\,\&_{i\in I}\,?l_i(\exists[\rho_i\|\{\rho_i\mapsto U_i\}].\mathtt{tr}\,(\rho_i)).S_i$		pack branching from role p
$\mid$	$\mathtt{t}$		type variable
$\mid$	$\mu\,\mathtt{t}.S$		recursive session type
G	$::=$		*global type*
	$\mathtt{end}$		termination
$\mid$	$\mathtt{p}\to\mathtt{q}:\{l_i(U_i).G_i\}_{i\in I}$		interaction
$\mid$	$\mathtt{p}\to\mathtt{q}:\{l_i(\exists[\rho_i\|\{\rho_i\mapsto U_i\}].tr(\rho_i)).G_i\}_{i\in I}$		pack interaction
$\mid$	$\mathtt{t}$		type variable
$\mid$	$\mu\,\mathtt{t}.G$		recursive type
H	$::=$		*partial session type*
	$\mathtt{end}$		terminated session
$\mid$	$\oplus_{i\in I}\,!l_i(U_i).H_i$		selection
$\mid$	$\&_{i\in I}\,?l_i(U_i).H_i$		branching (with $I\neq\emptyset$, U_i closed)
$\mid$	$\oplus_{i\in I}\,!l_i(\exists[\rho_i\|\{\rho_i\mapsto U_i\}].\mathtt{tr}\,(\rho_i)).H_i$		pack selection
$\mid$	$\&_{i\in I}\,?l_i(\exists[\rho_i\|\{\rho_i\mapsto U_i\}].\mathtt{tr}\,(\rho_i)).H_i$		pack branching (with $I\neq\emptyset$, U_i closed)
$\mid$	$\mathtt{t}$		type variable
$\mid$	$\mu\,\mathtt{t}.H$		recursive type
C	$::=\ \emptyset\ \mid\ C\otimes\{\rho\mapsto S\}$		*capabilities*
B	$::=\ \mathtt{Int}\ \mid\ \mathtt{Bool}$		*ground type*
U	$::=$		*payload type*
	B		ground type
$\mid$	$\mathtt{tr}\,(\rho)$		tracked type
$\mid$	$\{\rho\mapsto S\}$		capability type
$\mid$	S closed		session type
Γ	$::=\ \emptyset\ \mid\ \Gamma, x:U\ \mid\ \Gamma,\mathtt{s}[\mathtt{p}]:\mathtt{tr}\,(\rho)$		*environment*
Δ	$::=\ \emptyset\ \mid\ \Delta, X:\tilde{U}$		*process names*

Fig. 5. Local, global and partial session types

Local (session) types Fig. 5. The single form of interaction from global types splits into *select* (internal choice) and *branch* (external choice). The *branching type* $\mathtt{p}\&_{i\in I}?l_i\,(U_i)\,.S_i$ describes a channel that can receive a label l_i from role $\mathtt{p}$ (for some $i \in I$, chosen by $\mathtt{p}$), together with a *payload* of type U_i; then, the channel must be used as S_i. The *selection type* $\mathtt{p}\oplus_{i\in I}!l_i\,(U_i)\,.S_i$, describes a channel that can choose a label l_i (for any $i \in I$), and send it to $\mathtt{p}$ together with a payload of type U_i; then, the channel must be used as S_i. Session types also allow guarded recursion.

The relationship between global types and session types is formalised by the notion of projection.

Definition 1. *The projection of G onto a role $\mathtt{q}$, written $G \upharpoonright \mathtt{q}$, is:*

$$\mathtt{end} \upharpoonright \mathtt{q} \triangleq \mathtt{end} \qquad \mathtt{t} \upharpoonright \mathtt{q} \triangleq \mathtt{t} \qquad (\mu\,\mathtt{t}.G) \upharpoonright \mathtt{q} \triangleq \begin{cases} \mu\,\mathtt{t}.(G \upharpoonright \mathtt{q}) & \text{if } G \upharpoonright \mathtt{q} \neq \mathtt{t}'\,(\forall \mathtt{t}') \\ \mathtt{end} & \text{otherwise} \end{cases}$$

$$(\mathtt{p} \to \mathtt{p}' : \{l_i(U_i).G_i\}_{i\in I}) \upharpoonright \mathtt{q} \triangleq \begin{cases} \mathtt{p}' \oplus_{i\in I} !l_i(U_i).(G_i \upharpoonright \mathtt{q}) & \text{if } \mathtt{q} = \mathtt{p}, \\ \mathtt{p}\,\&_{i\in I}\,?l_i(U_i).(G_i \upharpoonright \mathtt{q}) & \text{if } \mathtt{q} = \mathtt{p}', \\ \bigsqcap_{i\in I}\,(G_i \upharpoonright \mathtt{q}) & \text{if } \mathtt{p} \neq \mathtt{q} \neq \mathtt{p}' \end{cases}$$

$$(\mathtt{p} \to \mathtt{p}' : \{l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}\,(\rho_i)).G_i\}_{i\in I}) \upharpoonright \mathtt{q} \triangleq$$
$$\begin{cases} \mathtt{p}' \oplus_{i\in I} !l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}\,(\rho_i)).(G_i \upharpoonright \mathtt{q}) & \text{if } \mathtt{q} = \mathtt{p}, \\ \mathtt{p}\,\&_{i\in I}\,?l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}\,(\rho_i)).(G_i \upharpoonright \mathtt{q}) & \text{if } \mathtt{q} = \mathtt{p}', \\ \bigsqcap_{i\in I}\,(G_i \upharpoonright \mathtt{q}) & \text{if } \mathtt{p} \neq \mathtt{q} \neq \mathtt{p}' \end{cases}$$

Definition 2. *The* merge operator for session types, $\bigsqcap$, *is defined by*

$$\mathtt{end} \sqcap \mathtt{end} \triangleq \mathtt{end} \qquad \mathtt{t} \sqcap \mathtt{t} \triangleq \mathtt{t} \qquad \mu\,t.S \sqcap \mu\,t.S' \triangleq \mu\,t.(S \sqcap S')$$

$$\mathtt{p}\,\&_{i\in I}\,?l_i(U_i).S_i \sqcap \mathtt{p}\,\&_{j\in J}\,?l_j(U_j).S'_j \triangleq \mathtt{p}\,\&_{k\in I\cap J}\,?l_k(U_k).(S_k \sqcap S'_k)\ \&\ \mathtt{p}\,\&_{j\in I\setminus J}\,?l_i(U_i).S_i\ \&\ \mathtt{p}\,\&_{j\in J\setminus I}\,?l_j(U_j).S'_j$$

$$\mathtt{p} \oplus_{i\in I} !l_i(U_i).S_i \sqcap \mathtt{p} \oplus_{i\in I} !l_i(U_i).S_i \triangleq \mathtt{p} \oplus_{i\in I} !l_i(U_i).S_i$$

$$\mathtt{p}\,\&_{i\in I}\,?l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}\,(\rho_i)).S_i \sqcap \mathtt{p}\,\&_{j\in J}\,?l_j(\exists[\rho_j|\{\rho_j \mapsto U_j\}].\mathtt{tr}\,(\rho_j)).S'_j \triangleq \mathtt{p}\,\&_{k\in I\cap J}\,?l_k(\exists[\rho_k|\{\rho_k \mapsto U_k\}].\mathtt{tr}\,(\rho_k)).(S_k \sqcap S'_k)\ \&\ \mathtt{p}\,\&_{j\in I\setminus J}\,?l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}\,(\rho_i)).S_i\ \&\ \mathtt{p}\,\&_{j\in J\setminus I}\,?l_j(\exists[\rho_j|\{\rho_j \mapsto U_j\}].\mathtt{tr}\,(\rho_j)).S'_j$$

$$\mathtt{p} \oplus_{i\in I} !l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}\,(\rho_i)).S_i \sqcap \mathtt{p} \oplus_{i\in I} !l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}\,(\rho_i)).S_i \triangleq \mathtt{p} \oplus_{i\in I} !l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}\,(\rho_i)).S_i$$

Projecting $\mathtt{end}$ or a type variable $\mathtt{t}$ onto any role does not change it. Projecting a recursive type $\mu\,t.G$ onto $\mathtt{q}$ means projecting G onto $\mathtt{q}$. However, if G does not involve $\mathtt{q}$ then $G \upharpoonright \mathtt{q}$ is a type variable and it must be replaced by $\mathtt{end}$ to avoid introducing an unguarded recursive type. Projecting an interaction between $\mathtt{p}$ and $\mathtt{p}'$ onto either $\mathtt{p}$ or $\mathtt{p}'$ produces a select or a branch. Projecting onto a different role $\mathtt{q}$ ignores the interaction and merges [13,37] the projections of the continuations.

Crucially, $\sqcap$ can combine different *internal* choices, but *not* external choices because that could violate type safety.

Definition 3. *For a session type S, $\mathtt{roles}(S)$ denotes the set of roles occurring in S. We write $\mathtt{p} \in S$ for $\mathtt{p} \in \mathtt{roles}(S)$, and $\mathtt{p} \in S \backslash \mathtt{q}$ for $\mathtt{p} \in \mathtt{roles}(S) \setminus \{\mathtt{q}\}$.*

Partial session types Fig. 5. They have the same cases as local types, without role annotations. Partial types have a notion of *duality* which exchanges branch and select but preserves payload types.

Definition 4. $\overline{H}$ *is the* dual of H*, defined by:*

$$\overline{\oplus_{i\in I}!l_i\,(U_i)\,.H_i} \triangleq \&_{i\in I}\,?l_i(U_i).\overline{H_i} \qquad \overline{\&_{i\in I}?l_i\,(U_i)\,.H_i} \triangleq \oplus_{i\in I}\,!l_i(U_i).\overline{H_i}$$

$$\overline{\oplus_{i\in I}\,!l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}\,(\rho_i)).\overline{H_i}} \triangleq \&_{i\in I}\,?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}\,(\rho_i)).\overline{H_i}$$

$$\overline{\&_{i\in I}\,?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}\,(\rho_i)).\overline{H_i}} \triangleq \oplus_{i\in I}\,!l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}\,(\rho_i)).\overline{H_i}$$

$$\overline{\mathtt{end}} \triangleq \mathtt{end} \qquad \overline{\mathtt{t}} \triangleq \mathtt{t} \qquad \overline{\mu\,t.H} \triangleq \mu\,t.\overline{H}$$

Similarly to the projection of global types to local types, a local type can be projected onto a role $\mathtt{q}$ to give a partial type. This yields a partial type that only describes the communications involving $\mathtt{q}$. The definition follows the same principles as the previous definition (Def. 1).

Definition 5. $S \upharpoonright \mathtt{q}$ *is the* partial projection of S onto $\mathtt{q}$*:*

$$\mathtt{end}\upharpoonright\mathtt{q} \triangleq \mathtt{end} \qquad \mathtt{t}\upharpoonright\mathtt{q} \triangleq \mathtt{t} \qquad (\mu\,\mathtt{t}.S)\upharpoonright\mathtt{q} \triangleq \begin{cases}\mu\,\mathtt{t}.(S\upharpoonright\mathtt{q}) & \text{if } S\upharpoonright\mathtt{q} \neq \mathtt{t}'\,(\forall\mathtt{t}')\\ \mathtt{end} & \text{otherwise}\end{cases}$$

$$(\mathtt{p}\oplus_{i\in I}!l_i\,(U_i)\,.S_i)\upharpoonright\mathtt{q} \triangleq \begin{cases}\oplus_{i\in I}\,!l_i(U_i).(S_i\upharpoonright\mathtt{q}) & \text{if } \mathtt{q}=\mathtt{p},\\ \bigsqcap_{i\in I}\,(S_i\upharpoonright\mathtt{q}) & \text{if } \mathtt{p}\neq\mathtt{q}\end{cases}$$

$$(\mathtt{p}\&_{i\in I}?l_i\,(U_i)\,.S_i)\upharpoonright\mathtt{q} \triangleq \begin{cases}\&_{i\in I}\,?l_i(U_i).S_i\upharpoonright\mathtt{q} & \text{if } \mathtt{q}=\mathtt{p},\\ \bigsqcap_{i\in I}\,(S_i\upharpoonright\mathtt{q}) & \text{if } \mathtt{p}\neq\mathtt{q}\end{cases}$$

$$(\mathtt{p}\,\oplus_{i\in I}\,!l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}\,(\rho_i)).S_i)\upharpoonright\mathtt{q} \triangleq \begin{cases}\oplus_{i\in I}\,!l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}\,(\rho_i)).(S_i\upharpoonright\mathtt{q}) & \text{if } \mathtt{q}=\mathtt{p},\\ \bigsqcap_{i\in I}\,(S_i\upharpoonright\mathtt{q}) & \text{if } \mathtt{p}\neq\mathtt{q}\end{cases}$$

$$(\mathtt{p}\,\&_{i\in I}\,?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}\,(\rho_i)).S_i)\upharpoonright\mathtt{q} \triangleq \begin{cases}\&_{i\in I}\,?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}\,(\rho_i)).S_i\upharpoonright\mathtt{q} & \text{if } \mathtt{q}=\mathtt{p},\\ \bigsqcap_{i\in I}\,(S_i\upharpoonright\mathtt{q}) & \text{if } \mathtt{p}\neq\mathtt{q}\end{cases}$$

Definition 6. *The* merge operator for partial session types, $\bigsqcap$*, is defined by*

$$\mathtt{end}\sqcap\mathtt{end} \triangleq \mathtt{end} \qquad \mathtt{t}\sqcap\mathtt{t} \triangleq \mathtt{t} \qquad \mu\,t.H\sqcap\mu\,t.H' \triangleq \mu\,t.(H\sqcap H')$$

$$\&_{i\in I}\,?l_i(U_i).H_i \sqcap \&_{i\in I}\,?l_i(U_i).H'_i \triangleq \&_{i\in I}\,?l_i(U_i).(H_i\sqcap H'_i)$$

$$\oplus_{i\in I}\,!l_i(U_i).H_i \sqcap \oplus_{j\in J}\,!l_j(U_j).H'_j \triangleq \left(\oplus_{k\in I\cap J}\,!l_k(U_k).(H_k\sqcap H'_k)\right) \oplus \left(\oplus_{i\in I\setminus J}\,!l_i(U_i).H_i\right) \oplus \left(\oplus_{j\in J\setminus I}\,!l_j(U_j).H'_j\right)$$

$$\&_{i\in I}\,?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}\,(\rho_i)).H_i \sqcap \&_{i\in I}\,?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}\,(\rho_i)).H'_i \triangleq \&_{i\in I}\,?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}\,(\rho_i)).(H_i\sqcap H'_i)$$

$$\oplus_{i\in I}\,!l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}\,(\rho_i)).H_i \sqcap \oplus_{j\in J}\,!l_j(\exists[\rho_j|\{\rho_j\mapsto U_j\}].\mathtt{tr}\,(\rho_j)).H'_j \triangleq \left(\oplus_{k\in I\cap J}\,!l_k(\exists[\rho_k|\{\rho_k\mapsto U_k\}].\mathtt{tr}\,(\rho_k)).(H_k\sqcap H'_k)\right) \oplus \left(\oplus_{i\in I\setminus J}\,!l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}\,(\rho_i)).H_i\right) \oplus \left(\oplus_{j\in J\setminus I}\,!l_j(\exists[\rho_j|\{\rho_j\mapsto U_j\}].\mathtt{tr}\,(\rho_j)).H'_j\right)$$

$$\frac{\forall i \in I \quad U_i \leqslant_{\mathsf{S}} U_i' \quad S_i \leqslant_{\mathsf{S}} S_i'}{\mathtt{p}\&_{i\in I} ?l_i\,(U_i)\,.S_i \;\leqslant_{\mathsf{S}}\; \mathtt{p}\,\&_{i\in I\cup J}\,?l_i(U_i').S_i'}\ \textsc{SBr} \qquad \frac{\forall i \in I \quad U_i' \leqslant_{\mathsf{S}} U_i \quad S_i \leqslant_{\mathsf{S}} S_i'}{\mathtt{p}\oplus_{i\in I\cup J}\,!l_i(U_i).S_i \leqslant_{\mathsf{S}} \mathtt{p}\oplus_{i\in I}\,!l_i(U_i').S_i'}\ \textsc{SSel}$$

$$\frac{\forall i \in I \quad U_i \leqslant_{\mathsf{S}} U_i' \quad S_i \leqslant_{\mathsf{S}} S_i'}{\mathtt{p}\,\&_{i\in I}\,?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}\,(\rho_i)).S_i \;\leqslant_{\mathsf{S}}\; \mathtt{p}\,\&_{i\in I\cup J}\,?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i'\}].\mathtt{tr}\,(\rho_i)).S_i'}\ \textsc{SBrP}$$

$$\frac{\forall i \in I \quad U_i' \leqslant_{\mathsf{S}} U_i \quad S_i \leqslant_{\mathsf{S}} S_i'}{\mathtt{p}\oplus_{i\in I\cup J}\,!l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}\,(\rho_i)).S_i \leqslant_{\mathsf{S}} \mathtt{p}\oplus_{i\in I}\,!l_i(\exists[\rho_i|\{\rho_i\mapsto U_i'\}].\mathtt{tr}\,(\rho_i)).S_i'}\ \textsc{SSel}$$

$$\frac{B \leqslant_{\mathsf{B}} B'}{B \leqslant_{\mathsf{S}} B'}\ \textsc{SB} \qquad \frac{}{\mathtt{end} \leqslant_{\mathsf{S}} \mathtt{end}}\ \textsc{SEnd} \qquad \frac{S\{\mu\,\mathtt{t}.S/\mathtt{t}\} \leqslant_{\mathsf{S}} S'}{\mu\,\mathtt{t}.S \leqslant_{\mathsf{S}} S'}\ \textsc{S}\mu\textsc{L} \qquad \frac{S \leqslant_{\mathsf{S}} S'\{\mu\,\mathtt{t}.S'/\mathtt{t}\}}{S \leqslant_{\mathsf{S}} \mu\,\mathtt{t}.S'}\ \textsc{S}\mu\textsc{R}$$

Fig. 6. Subtyping for local session types.

Example 2 (Projections of Global and Local Types). Consider the global type G of the producer-consumer example from the introduction, which captures the interaction between the producer and consumer entities through roles $\mathtt{p}$, $\mathtt{c}$, and between producer, consumer and buffer through roles $\mathtt{q}$ (shared between producer and consumer) and $\mathtt{b}$. Projecting onto $\mathtt{p}$ gives the session type

$$S = G \upharpoonright \mathtt{p} = \mathtt{c}\oplus!\mathsf{buffer}\left(\exists[\rho_q|\{\rho_q\mapsto S_q'\}].\mathtt{tr}\,(\rho_q)\right).\mu\,\mathtt{t}.\mathtt{c}\oplus!\mathsf{turn}\left(\{\rho_q\mapsto S_q'\}\right).\\ \mathtt{c}\&?\mathsf{turn}\left(\{\rho_q\mapsto S_q\}\right).\mathtt{t}$$

and further projecting onto $\mathtt{c}$ gives the partial session type:

$$H = S \upharpoonright \mathtt{c} = \oplus!\mathsf{buffer}\left(\exists[\rho_q|\{\rho_q\mapsto S_q'\}].\mathtt{tr}\,(\rho_q)\right).\mu\,\mathtt{t}.\oplus!\mathsf{turn}\left(\{\rho_q\mapsto S_q'\}\right).\\ \&?\mathsf{turn}\left(\{\rho_q\mapsto S_q\}\right).\mathtt{t}$$

Subtyping The *subtyping relation*, Def. 7, says that a session type S is "smaller" than S' when S is "less demanding" than S' — i.e., when S permits more internal choices, and imposes fewer external choices, than S'. Subtyping is defined on both local types and partial session types.

Definition 7 (Subtyping). Subtyping $\leqslant_{\mathsf{S}}$ on session types *is the largest relation such that* (i) *if* $S \leqslant_{\mathsf{S}} S'$, *then* $\forall \mathtt{p} \in (\mathtt{roles}(S)\cup\mathtt{roles}(S'))\ S\upharpoonright\mathtt{p} \leqslant_{\mathsf{P}} S'\upharpoonright\mathtt{p}$, *and* (ii) *is closed backwards under the coinductive rules in Fig. 6.* Subtyping $\leqslant_{\mathsf{P}}$ on partial session types *is defined coinductively by the rules in Fig. 7.*

Def. 7 uses coinduction to support recursive types [27, § 20 and § 21]. Clause *(i)* links local and partial subtyping, and ensures that if two types are related, then their partial projections exist: this will be necessary later, for typing contexts (Def. 10). The essence of Def. 7 lies in clause *(ii)*. Rules SBr, SSel define subtyping on branch/select types, and SBrP, SSelP define subtyping on branch pack/select pack types. These rules are covariant in the continuation types, i.e., they require $S_i \leqslant_{\mathsf{S}} S_i'$. SBr, and SBrP are covariant also in the number of branches offered, whereas SSel, and SSelP are contravariant. SB relates base types, if they are

$$\frac{\forall i \in I \quad U_i \leqslant_{\mathsf{S}} U_i' \quad H_i \leqslant_{\mathsf{P}} H_i'}{\&_{i\in I} ?l_i\,(U_i)\,.H_i \leqslant_{\mathsf{P}} \&_{i\in I\cup J}\, ?l_i(U_i').H_i'}\ \textsc{SParBr} \qquad \frac{\forall i \in I \quad U_i' \leqslant_{\mathsf{S}} U_i \quad H_i \leqslant_{\mathsf{P}} H_i'}{\oplus_{i\in I\cup J}\, !l_i(U_i).H_i \leqslant_{\mathsf{P}} \oplus_{i\in I}\, !l_i(U_i').H_i'}\ \textsc{SParSel}$$

$$\frac{\forall i \in I \quad U_i \leqslant_{\mathsf{S}} U_i' \quad H_i \leqslant_{\mathsf{P}} H_i'}{\&_{i\in I}\, ?l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}\,(\rho_i)).H \leqslant_{\mathsf{P}} \&_{i\in I\cup J}\, ?l_i(\exists[\rho_i|\{\rho_i \mapsto U_i'\}].\mathtt{tr}\,(\rho_i)).H_i'}\ \textsc{SParBrP}$$

$$\frac{\forall i \in I \quad U_i' \leqslant_{\mathsf{S}} U_i \quad H_i \leqslant_{\mathsf{P}} H_i'}{\oplus_{i\in I\cup J}\, !l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}\,(\rho_i)).H_i \leqslant_{\mathsf{P}} \oplus_{i\in I}\, !l_i(\exists[\rho_i|\{\rho_i \mapsto U_i'\}].\mathtt{tr}\,(\rho_i)).H_i'}\ \textsc{SParSelP}$$

$$\frac{}{\mathtt{end} \leqslant_{\mathsf{P}} \mathtt{end}}\ \textsc{SParEnd} \qquad \frac{H\{\mu\,\mathtt{t}.H/\mathtt{t}\} \leqslant_{\mathsf{P}} H'}{\mu\,\mathtt{t}.H \leqslant_{\mathsf{P}} H'}\ \textsc{SPar}\mu\textsc{L} \qquad \frac{H \leqslant_{\mathsf{P}} H'\{\mu\,\mathtt{t}.H'/\mathtt{t}\}}{H \leqslant_{\mathsf{P}} \mu\,\mathtt{t}.H'}\ \textsc{SPar}\mu\textsc{R}$$

Fig. 7. Subtyping for partial session types.

related by $\leqslant_{\mathsf{B}}$. SEnd relates terminated channel types. SμL and SμR are standard under coinduction: they say that a recursive session type $\mu\,t.S$ is related to S' if its unfolding is related.

Capabilities In our type system linearity is enforced via capabilities, rather than via environment splitting as in most session type systems. Each process has a capability collection associated with it, allowing it to communicate on the associated channels. The tracked type $\mathtt{tr}\,(\rho)$ is a singleton type associating a channel to capability ρ and to no other, which in turn maps to the channel's session type $\{\rho \mapsto S\}$. Hence two variables with the same capability ρ are aliases for the same channel. Individual capabilities are joined together using the $\otimes$ operator: $C = \{\rho_1 \mapsto S_1\} \otimes ... \otimes \{\rho_n \mapsto S_n\}$. The ordering is insignificant. The type system maintains the invariant that $\rho_1, \ldots, \rho_n$ are distinct.

Definition 8 (Terminated capabilities). *A capability C is* terminated *if for every $\rho \in \mathrm{dom}\,(C)$, $C(\rho) = \mathtt{end}$.*

Definition 9 (Substitution of capabilities).

$$\{\rho \mapsto S\}[\rho'/\rho_2] = \{\rho \mapsto S\} \qquad \{\rho \mapsto S\}[\rho'/\rho] = \{\rho' \mapsto S\}$$
$$\emptyset[\rho'/\rho] = \emptyset \qquad (C_1 \otimes C_2)[\rho'/\rho] = C_1[\rho'/\rho] \otimes C_2[\rho'/\rho]$$

Definition 10 (Completeness and consistency).
(Γ, C) *is* complete *iff for all* $\mathtt{s}[\mathtt{p}]:\mathtt{tr}\,(\rho_p)\,, \rho_p:\{\rho_p \mapsto S_p\} \in \Gamma$ *with* $\{\rho_p \mapsto S_p\} \in C$, $\mathtt{q} \in S_p$ *implies* $\mathtt{s}[\mathtt{q}], \rho_q : \{\rho_q \mapsto S_q\} \in \mathrm{dom}\,(\Gamma)$ *and* $\{\rho_q \mapsto S_q\} \in \mathrm{dom}\,(C)$.
(Γ, C) *is* consistent *iff for all* $\mathtt{s}[\mathtt{p}] : \mathtt{tr}\,(\rho_p)\,, \mathtt{s}[\mathtt{q}] : \mathtt{tr}\,(\rho_q)\,, \rho_p : \{\rho_p \mapsto S_p\}, \rho_q : \{\rho_q \mapsto S_q\} \in \Gamma$ *with* $\{\rho_p \mapsto S_p, \rho_q \mapsto S_q\} \in C$ *we have* $\overline{S_p \upharpoonright \mathtt{q}} \leqslant_{\mathsf{S}} S_q \upharpoonright \mathtt{p}$.

Definition 11. *Typing judgements are inductively defined by the rules in Fig. 8, and have the form:*
– $\Gamma \vdash v : T; C$ *for values, and*

$$\textsc{TCap}\ \frac{}{\Gamma \vdash \rho : \{\rho \mapsto S\}; \{\rho \mapsto S\}} \qquad \textsc{TVar}\ \frac{\mathtt{c} : \mathtt{tr}\,(\rho)\,, \rho : \{\rho \mapsto S\} \in \Gamma}{\Gamma \vdash \mathtt{c} : \mathtt{tr}\,(\rho)\,; \emptyset} \qquad \textsc{TVal}\ \frac{v \in B}{\Gamma \vdash v : B; \emptyset} \qquad \textsc{TInact}\ \frac{C \text{ terminated}}{\Delta; \Gamma \vdash \mathbf{0}; C}$$

$$\textsc{TPar}\ \frac{\Delta; \Gamma \vdash P; C_1 \quad \Delta; \Gamma \vdash Q; C_2}{\Delta; \Gamma \vdash P \mid Q; C_1 \otimes C_2} \qquad \textsc{TSub}\ \frac{\Delta; \Gamma' \vdash P; C \otimes \{\rho \mapsto U\} \qquad U' \leqslant_{\mathsf{S}} U}{\Delta; \Gamma \vdash P; C \otimes \{\rho \mapsto U'\}}$$

$$\textsc{TSel}\ \frac{\Gamma \vdash v : U; C \quad \Delta; \Gamma \vdash P; C' \otimes \{\rho \mapsto S_j\} \quad \mathtt{c} : \mathtt{tr}\,(\rho)\,, \rho : \{\rho \mapsto S_j\} \in \Gamma \quad j \in I}{\Delta; \Gamma \vdash \mathtt{c}[\mathtt{p}] \oplus \langle l_j(v) \rangle . P; C \otimes C' \otimes \{\rho \mapsto \mathtt{p} \oplus_{i \in I} ! l_i\,(U_i)\,. S_i\}}$$

$$\textsc{TBr}\ \frac{\Delta; \Gamma, x_i : U_i \vdash P_i; C \otimes C_i \otimes \{\rho \mapsto S_i\} \quad \mathtt{c} : \mathtt{tr}\,(\rho)\,, \rho : \{\rho \mapsto S_i\} \in \Gamma \quad \forall i \in I}{\Delta; \Gamma, \mathtt{c} : \mathtt{tr}\,(\rho) \vdash \mathtt{c}[\mathtt{p}] \&_{i \in I} \{l_i(x_i).\, P_i\}; C \otimes \{\rho \mapsto \mathtt{p} \&_{i \in I} ? l_i\,(U_i)\,. S_i\}}$$

$$\textsc{TSelP}\ \frac{\begin{array}{c}\Gamma \vdash v : \mathtt{tr}\left(\rho'\right); \emptyset \\ \Delta; \Gamma \vdash P; C \otimes \{\rho \mapsto S\} \otimes \{\rho' \mapsto U\} \qquad \mathtt{c} : \mathtt{tr}\,(\rho)\,, \rho : \{\rho \mapsto S\} \in \Gamma \qquad j \in I\end{array}}{\Delta; \Gamma \vdash \mathtt{c}[\mathtt{p}] \oplus \langle l_j(\mathsf{pack}(\rho', v)) \rangle . P; C \otimes \{\rho \mapsto \mathtt{p} \oplus_{j \in I} ! l_j\,\left(\exists [\rho' | \{\rho' \mapsto U\}] . \mathtt{tr}\left(\rho'\right)\right) . S\} \otimes \{\rho' \mapsto U\}}$$

$$\textsc{TBrP}\ \frac{\Delta; \Gamma, \mathtt{v_i} : \mathtt{tr}\,(\rho_i)\,, \rho_i : \{\rho_i \mapsto U_i\} \vdash P_i; C \otimes \{\rho \mapsto S_i\} \quad \forall i \in I \quad \mathtt{c} : \mathtt{tr}\,(\rho)\,, \rho : \{\rho \mapsto S_i\} \in \Gamma}{\Delta; \Gamma \vdash \mathtt{c}[\mathtt{p}] \&_{i \in I} \{l_i(\mathsf{pack}(\rho_i, \mathtt{v_i}))\,.\, P_i\}; C \otimes \{\rho \mapsto \mathtt{p} \&_{i \in I} ? l_i\,(\exists [\rho_i | \{\rho_i \mapsto U_i\}] . \mathtt{tr}\,(\rho_i))\,. S_i\}}$$

$$\textsc{TRes}\ \frac{\begin{array}{c}\Delta; \Gamma, \Gamma' \vdash P; C \otimes C' \\ (\Gamma' = \{\mathtt{s}[\mathtt{p}] : \mathtt{tr}\,(\rho_p)\,, \rho_p : \{\rho_p \mapsto S_p\}\}_{p \in I}, C' = \otimes_{p \in I} \{\rho_p \mapsto S_p\}) \text{ complete}\end{array}}{\Delta; \Gamma \vdash (\nu\, s : \Gamma') P; C}$$

$$\textsc{TDef}\ \frac{\Delta, X : \tilde{U}; \tilde{x} : \tilde{U} \vdash P; \tilde{C} \qquad \Delta, X : \tilde{U}; \Gamma \vdash Q; C}{\Delta; \Gamma \vdash \mathsf{def}\, X \langle \tilde{x} : \tilde{U} \rangle = P; \tilde{C}\, \mathsf{in}\, Q; C} \qquad \textsc{TCall}\ \frac{\forall i \in \{1..n\} \qquad \Gamma \vdash v_i : U_i; C_i}{\Delta, X : U_1, \ldots, U_n; \Gamma \vdash X \langle v_1, \ldots, v_n \rangle; C_1 \otimes \ldots \otimes C_n}$$

Fig. 8. Typing rules

– $\Delta; \Gamma \vdash P; C$ *for processes (with* (Γ, C) *consistent, and* $\forall(\mathtt{c} : \mathtt{tr}\,(\rho) \in \Gamma; \{\rho \mapsto S\} \in C), S \upharpoonright \mathtt{p}$ *is defined* $\forall \mathtt{p} \in S$*).*

Γ is an environment of typed variables and channels together with their capability typing. Δ, defined in Fig. 5 is an environment of typed process names, used in rules TDef and TCall for recursive process definitions and calls.

Rule TCap takes the type for a capability ρ from the capability set. TVar and TVal are standard. TInact has a standard condition that all session types have reached `end`, expressed as the capability set being terminated. TPar combines the capability sets in a parallel composition. TSub is a standard subsumption rule using $\leqslant_{\mathsf{S}}$ (Def. 7), the difference being the type in the capability set. TSel (resp. TBr) states that the selection (resp. branching) on channel $\mathtt{c}[\mathtt{p}]$ is well typed if the capability associated with it is of compatible selection (resp. branching)

type and the continuations $P_i, \forall i \in I$ are well-typed with the continuation session types. TSelP is similar to TSel, with the notable difference that an existential package is created for the channel being sent, containing the channel and its abstracted capability. Note that the actual capability to use the endpoint remains with process P. TBrP is similar to TBr, with the difference that it unpackages the channel received and binds its capability type in the continuation session type (used to identify the correct capability when received later). TRes requires the restricted environment Γ' and the associated capability set C' to be *complete* (Def. 10). TDef takes account of capability sets as well as parameters, and TCall similarly requires capability sets. The parameters of a defined process include any necessary capabilities, which then also appear in the corresponding C_i, because not all capabilities associated with the channel parameters need to be present when the call is made.

4 Case study: Producer-Consumer

We now expand on the producer-consumer scenario from § 1 by discussing the process definitions and showing part of the typing derivation. To lighten the notation, we present a set of mutually recursive definitions, instead of using the formal syntax of $\mathsf{def} \ldots \mathsf{in}$.

Recall that the example consists of three processes: the producer, the consumer, and a one-place buffer (Fig. 1). The producer and the consumer communicate with the buffer on a single shared channel. Each of the two must wait to receive the *capability* to communicate on the channel before doing so.

The buffer B is parameterised by channel $\mathtt{x}$ and by the capability for it, ρ_x, and alternately responds to add and $\mathsf{request}$ messages. At the end of the definition, $\{\rho_x \mapsto S_b\}$ shows the held capability and its session type.

$$\begin{aligned}\mathsf{B}\langle \mathtt{x} : \mathtt{tr}\,(\rho_x)\,, \rho_x : \{\rho_x \mapsto S_b\}\rangle = {} & \mathtt{x}[\mathsf{q}]\&\mathsf{add}(\mathsf{i}).\ \mathtt{x}[\mathsf{q}]\&\mathsf{request}(\mathtt{r}).\\ & \mathtt{x}[\mathsf{p}]\oplus\mathsf{send}(\mathsf{i}).\ \mathsf{B}\langle \mathtt{x}, \rho_x\rangle; \{\rho_x \mapsto S_b\}\end{aligned}$$

The producer is represented by two process definitions: $\mathsf{Produce}$ and P. $\mathsf{Produce}$ is a recursive process with several parameters. Channels $\mathtt{x}$ and $\mathtt{y}$ are used to communicate with the consumer and the buffer, respectively. Their capabilities are ρ_x and ρ_y. Finally, i is the value to be sent to the buffer. The process sends a value to the buffer ($\mathsf{add}(\mathsf{i})$), transfers the capability for the shared channel y ($\mathsf{turn}(\rho_y)$) and receives it back from the consumer. Process P is the entry point for the producer. It has the same parameters as $\mathsf{Produce}$, except for i. The only action of P is to send the consumer a shared reference to the channel used for communication with the buffer ($\mathtt{x}[\mathsf{c}]\oplus\mathsf{buffer}(\mathsf{pack}(\rho_y, \mathtt{y}[\mathsf{b}]))$.).

$$\begin{aligned}&\mathsf{Produce}\langle \mathtt{x} : \mathtt{tr}\,(\rho_x)\,, \mathtt{y} : \mathtt{tr}\,(\rho_y)\,, \mathsf{i} : \mathtt{Int}, \rho_x : \{\rho_x \mapsto S'_p\}, \rho_y : \{\rho_y \mapsto S_q\}\rangle =\\ &\quad \mathtt{y}[\mathsf{b}]\oplus\mathsf{add}(\mathsf{i}).\ \mathtt{x}[\mathsf{c}]\oplus\mathsf{turn}(\rho_y).\ \mathtt{x}[\mathsf{c}]\&\mathsf{turn}(\rho_y).\ \mathsf{Produce}\langle \mathtt{x}, \mathtt{y}, \mathsf{i}{+}1, \rho_x, \rho_y\rangle;\\ &\quad \{\rho_x \mapsto S'_p\} \otimes \{\rho_y \mapsto S_q\}\\ &\mathsf{P}\langle \mathtt{x} : \mathtt{tr}\,(\rho_x)\,, \mathtt{y} : \mathtt{tr}\,(\rho_y)\,, \rho_x : \{\rho_x \mapsto S_p\}, \rho_y : \{\rho_y \mapsto S_q\}\rangle =\\ &\quad \mathtt{x}[\mathsf{c}]\oplus\mathsf{buffer}(\mathsf{pack}(\rho_y, \mathtt{y}[\mathsf{b}])).\ \mathsf{Produce}\langle \mathtt{x}, \mathtt{y}, 0, \rho_x, \rho_y\rangle; \{\rho_x \mapsto S_p\} \otimes \{\rho_y \mapsto S_q\}\end{aligned}$$

In a similar way, the consumer is represented by $\mathsf{Consume}$ and C. The parameters, however, are different. C has $\mathtt{x}$ and its capability ρ_x, for communication with the producer, but it doesn't have $\mathtt{y}$ or ρ_y for communication with the buffer. It receives $\mathtt{y}$ from the producer, as part of $\mathsf{pack}(\rho_y, \mathtt{y}[\mathsf{b}])$, and $\mathtt{y}$ is passed as a parameter to $\mathsf{Consume}$. The capability ρ_y is not a parameter of $\mathsf{Consume}$, but it is received in a turn message from the producer.

$$
\begin{aligned}
&\mathsf{Consume}\langle \mathtt{x} : \mathtt{tr}\,(\rho_x)\,, \mathtt{y} : \mathtt{tr}\,(\rho_y)\,, \rho_x : \{\rho_x \mapsto S_c'\}\rangle = \\
&\quad \mathtt{x}[\mathsf{p}]\&\mathsf{turn}(\rho_y).\ \mathtt{y}[\mathsf{b}]\oplus\mathsf{request}(r).\ \mathtt{y}[\mathsf{b}]\&\mathsf{send}(\mathsf{i}).\ \mathtt{x}[\mathsf{p}]\oplus\mathsf{turn}(\rho_y). \\
&\quad \mathsf{Consume}\langle \mathtt{x}, \mathtt{y}, \rho_x\rangle; \{\rho_x \mapsto S_c'\} \\
&\mathsf{C}\langle \mathtt{x} : \mathtt{tr}\,(\rho_x)\,, \rho_x : \{\rho_x \mapsto S_c\}\rangle = \\
&\quad \mathtt{x}[\mathsf{p}]\&\mathsf{buffer}(\mathsf{pack}(\rho_y, \mathtt{y}[\mathsf{b}])).\ \mathsf{Consume}\langle \mathtt{x}, \mathtt{y}, \rho_x\rangle; \{\rho_x \mapsto S_c\}
\end{aligned}
$$

The complete system consists of the producer, the consumer and the buffer in parallel, with sessions s_1 (roles $\mathtt{p}$ and $\mathtt{c}$) and s_2 (roles $\mathtt{q}$ and $\mathtt{b}$) scoped to construct a closed process.

$$(\nu\, s_1)((\nu\, s_2)(\mathsf{P}\langle \mathtt{s}_1[\mathsf{p}], \mathtt{s}_2[\mathsf{q}], \rho_p, \rho_q\rangle \mid \mathsf{B}\langle \mathtt{s}_2[\mathsf{b}], \rho_b\rangle) \mid \mathsf{C}\langle \mathtt{s}_1[\mathsf{c}], \rho_c\rangle$$

The session types involved in these processes are projections of the global type G (§3). They specify how each role is expected to use its channel endpoint. The roles are $\mathtt{b}$ for the buffer, $\mathtt{q}$ for the combined role of the producer and the consumer as they interact with the buffer, $\mathtt{p}$ for the producer, and $\mathtt{c}$ for the consumer.

$$
\begin{aligned}
S_b &= G \upharpoonright \mathtt{b} = \mu\, \mathtt{t}.\mathtt{q}\&?\mathsf{add}\,(\mathtt{Int})\,.\mathtt{q}\&?\mathsf{request}\,(\mathtt{Str})\,.\mathtt{q}\oplus!\mathsf{send}\,(\mathtt{Int})\,.t \\
S_q &= G \upharpoonright \mathtt{q} = \mu\, \mathtt{t}.\mathtt{b}\oplus!\mathsf{add}\,(\mathtt{Int})\,.\mathtt{b}\oplus!\mathsf{request}\,(\mathtt{Str})\,.\mathtt{b}\&?\mathsf{send}\,(\mathtt{Int})\,.t \\
S_p &= G \upharpoonright \mathtt{p} = \mathtt{c}\oplus!\mathsf{buffer}\left(\exists[\rho_q|\{\rho_q \mapsto S_q'\}].\mathtt{tr}\,(\rho_q)\right).\mu\, \mathtt{t}. \\
&\qquad\qquad \mathtt{c}\oplus!\mathsf{turn}\left(\{\rho_q \mapsto S_q'\}\right).\mathtt{c}\&?\mathsf{turn}\,(\{\rho_q \mapsto S_q\})\,.t \\
S_c &= G \upharpoonright \mathtt{c} = \mathtt{p}\&?\mathsf{buffer}\left(\exists[\rho_q|\{\rho_q \mapsto S_q'\}].\mathtt{tr}\,(\rho_q)\right).\mu\, \mathtt{t}. \\
&\qquad\qquad \mathtt{p}\&?\mathsf{turn}\left(\{\rho_q \mapsto S_q'\}\right).\mathtt{p}\oplus!\mathsf{turn}\,(\{\rho_q \mapsto S_q\})\,.t
\end{aligned}
$$

These types occur in the capabilities associated with each process. For example process $\mathsf{P}\langle \mathtt{s}_1[\mathsf{p}], \mathtt{s}_2[\mathsf{q}], \rho_p, \rho_q\rangle$ has $\{\rho_q \mapsto S_q\} \otimes \{\rho_p \mapsto S_p\}$, process $\mathsf{B}\langle \mathtt{s}_2[\mathsf{b}], \rho_b\rangle$ has $\{\rho_b \mapsto S_b\}$, and process $\mathsf{C}\langle \mathtt{s}_1[\mathsf{c}], \rho_c\rangle$ has $\{\rho_c \mapsto S_c\}$.

To illustrate the typing rules, we show the typing derivation for the producer, i.e. processes $\mathsf{Produce}$ (Fig. 9) and P (Fig. 10). Full derivations for all of the processes are in the technical report. The derivations use the following definitions.

$$
\begin{aligned}
S_p' &= \mu\, \mathtt{t}.\mathtt{c}\oplus!\mathsf{turn}\left(\{\rho_q \mapsto S_q'\}\right).\mathtt{c}\&?\mathsf{turn}\,(\{\rho_q \mapsto S_q\})\,.t \\
S_q' &= \mathtt{b}\oplus!\mathsf{request}\,(\mathtt{Str})\,.\mathtt{b}\&?\mathsf{send}\,(\mathtt{Int})\,.S_q \\
\Delta &= \mathsf{Produce} : (\mathtt{tr}\,(\rho_p)\,, \mathtt{tr}\,(\rho_q)\,, \mathtt{Int}, \{\rho_p \mapsto S_p\}, \{\rho_q \mapsto S_q\}) \\
\Gamma &= \mathtt{x} : \mathtt{tr}\,(\rho_p)\,, \mathtt{y} : \mathtt{tr}\,(\rho_q)\,, \mathsf{i} : \mathtt{Int}, \rho_p : \{\rho_p \mapsto S_p\}, \rho_q : \{\rho_q \mapsto S_q\}
\end{aligned}
$$

Scenarios with multiple producers/consumers can be represented in a similar way, the capabilities acting as a form of lock for the resource being shared. The typing derivation for producer consumer case study can be found in [34]

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash \mathtt{x} : \mathtt{tr}\,(\rho_x)\,, \mathtt{y} : \mathtt{tr}\,(\rho_y)\,, \mathtt{i} : \mathtt{Int}, \rho_x : \{\rho_x \mapsto S'_p\}, \rho_y : \{\rho_y \mapsto S_q\}; \{\rho_x \mapsto S'_p, \rho_y \mapsto S_q\}}{\Delta; \Gamma \vdash \mathsf{Produce}\langle x, y, \mathtt{i}{+}1, \rho_x, \rho_y\rangle; \{\rho_x \mapsto S'_p, \rho_y \mapsto S_q\}}\ \textsc{TCall}}{\Delta; \Gamma \vdash \mathtt{x}[\mathtt{c}]\&\{\mathsf{turn}(\rho_y).\mathsf{Produce}\langle x, y, \mathtt{i}{+}1, \rho_x, \rho_y\rangle\}; \{\rho_x \mapsto \mathtt{c}\&?\mathsf{turn}\,(\{\rho_q \mapsto S_q\})\,.S'_p\}}\ \textsc{TBr}$$

$$\dfrac{\vdots\ \textsc{TCap}\quad \dfrac{}{\Gamma \vdash \rho_q : \{\rho_q \mapsto S'_q\}; \{\rho_q \mapsto S'_q\}} \qquad \mathtt{x} : \mathtt{tr}\,(\rho_x)\,, \rho_x : \{\rho_x \mapsto \mathtt{c}\&?\mathsf{turn}\,(\{\rho_q \mapsto S_q\})\,.S'_p\} \in \Gamma}{\Delta; \Gamma \vdash \mathtt{x}[\mathtt{c}]\oplus\langle\mathsf{turn}(\rho_q)\rangle.\mathtt{x}[\mathtt{c}]\&\{\mathsf{turn}(\rho_y).\mathsf{Produce}\langle x, y, \mathtt{i}{+}1, \rho_x, \rho_y\rangle\}; \{\rho_x \mapsto S'_p, \rho_y \mapsto S'_q\}}\ \textsc{TSel}$$

$$\dfrac{\vdots \qquad \dfrac{\mathtt{i} \in \mathtt{Int}}{\Gamma \vdash \mathtt{i} : \mathtt{Int}; \emptyset}\ \textsc{TVal} \qquad \mathtt{y} : \mathtt{tr}\,(\rho_y)\,, \rho_y : \{\rho_y \mapsto S'_q\} \in \Gamma}{\Delta; \Gamma \vdash \mathtt{y}[\mathtt{b}]\oplus\langle\mathsf{add}(\mathtt{i})\rangle.\mathtt{x}[\mathtt{c}]\oplus\langle\mathsf{turn}(\rho_q)\rangle.\mathtt{x}[\mathtt{c}]\&\{\mathsf{turn}(\rho_q).\mathsf{Produce}\langle \mathtt{x}, \mathtt{y}, \mathtt{i}{+}1, \rho_x, \rho_y\rangle\}; \{\rho_x \mapsto S'_p, \rho_y \mapsto S_q\}}\ \textsc{TSel}$$

Fig. 9. Typing derivation for Produce.

$$\dfrac{\dfrac{\Gamma \vdash \mathtt{x} : \mathtt{tr}\,(\rho_x)\,, \mathtt{y} : \mathtt{tr}\,(\rho_y)\,, \rho_x : \{\rho_x \mapsto S'_p\}, \rho_y : \{\rho_y \mapsto S_q\}; \{\rho_x \mapsto S'_p, \rho_y \mapsto S_q\}}{\Delta; \Gamma \vdash \mathsf{Produce}\langle x, y, \mathtt{i}, \rho_p, \rho_q\rangle; \{\rho_x \mapsto S'_p, \rho_y \mapsto S_q\}}\ \textsc{TCall}}{\Delta; \Gamma \vdash \mathtt{x}[\mathtt{c}]\oplus\langle\mathsf{buffer}(\mathsf{pack}(\rho_q, \mathtt{y}[\mathtt{b}]))\rangle.\mathsf{Produce}\langle \mathtt{x}, \mathtt{y}, \mathtt{i}, \rho_x, \rho_y\rangle; \{\rho_y \mapsto \mathtt{p}\oplus!l\,(\exists[\rho_y|\{\rho_y \mapsto S_q\}].\mathtt{tr}\,(\rho_y))\,.S'_p, \rho_y \mapsto S_q\}}\ \textsc{TSelP}$$

$$\vdots\ \textsc{TVar}\quad \dfrac{\mathtt{y} : \mathtt{tr}\,(\rho_y)\,, \rho_y : \{\rho_y \mapsto S_q\}}{\Gamma \vdash \mathtt{y} : \mathtt{tr}\,(\rho_y)\,; \emptyset} \qquad \mathtt{x}, \rho_x : \{\rho_x \mapsto S_p\} \in \Gamma$$

Fig. 10. Typing derivation for P.

5 Technical Results

Following standard practice in the MPST literature, we show type safety and hence communication safety by proving a subject reduction theorem (Thm. 1). In the usual way, session types evolve during reduction—in our system, this is seen in both the Γ environment and the capability set C.

Definition 12 (Typing context reduction). *The* reduction $(\Gamma; C) \longrightarrow (\Gamma'; C')$ *is:*

$$(\mathtt{s}[\mathtt{p}] : \mathtt{tr}\,(\rho_p)\,,\ \mathtt{s}[\mathtt{q}] : \mathtt{tr}\,(\rho_q)\,, \rho_p : \{\rho_p \mapsto S_p\}, \rho_q : \{\rho_q \mapsto S_q\}; \{\rho_p \mapsto S_p, \rho_q \mapsto S_q\}) \longrightarrow$$
$$(\mathtt{s}[\mathtt{p}] : \mathtt{tr}\,(\rho_p)\,,\ \mathtt{s}[\mathtt{q}] : \mathtt{tr}\,(\rho_q)\,, \rho_p : \{\rho_p \mapsto S_k\}, \rho_q : \{\rho_q \mapsto S'_k\}; \{\rho_p \mapsto S_k, \rho_q \mapsto S'_k\})$$

$$\textit{if}\ \begin{cases} \mathtt{unf}(S_p) = \mathtt{q}\oplus_{i\in I}!l_i\,(U_i)\,.S_i & k\ \in\ I \\ \mathtt{unf}(S_q) = \mathtt{p}\ \&_{i\in I\cup J}\ ?l_i(U'_i).S'_i \quad U_k \leqslant_{\mathsf{S}} U'_k \end{cases}$$

$$\textit{or if}\ \begin{cases} \mathtt{unf}(S_p) = \mathtt{q}\oplus_{i\in I}!l_i\,(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}\,(\rho_i))\,.S_i & k\ \in\ I \\ \mathtt{unf}(S_q) = \mathtt{p}\ \&_{i\in I\cup J}\ ?l_i(\exists[\rho_i|\{\rho_i \mapsto U'_i\}].\mathtt{tr}\,(\rho_i)).S'_i \quad U_k \leqslant_{\mathsf{S}} U'_k \end{cases}$$

$$(\Gamma,\ \mathtt{c} : \mathtt{tr}\,(\rho)\,, \rho : \{\rho \mapsto U\}; C \otimes \{\rho \mapsto U\}) \longrightarrow$$
$$(\Gamma',\ \mathtt{c} : \mathtt{tr}\,(\rho)\,, \rho : \{\rho \mapsto U'\}; C' \otimes \{\rho \mapsto U'\})$$
$$\textit{if}\ \ (\Gamma; C) \to (\Gamma'; C')\ \ \textit{and}\ \ U \leqslant_{\mathsf{S}} U'$$

Following [29] our Def. 12 also accommodates subtyping ($\leqslant_{\mathsf{S}}$) and our iso-recursive type equivalence (hence, unfolds types explicitly).

Theorem 1 (Subject reduction). *If* $\Delta; \Gamma \vdash P; C$ *and* $P \longrightarrow P'$*, then there exist* Γ' *and* C' *such that* $\Delta; \Gamma' \vdash P'; C'$ *and* $(\Gamma; C) \longrightarrow^* (\Gamma'; C')$.

The proof is by induction on the derivation of $P \longrightarrow P'$, with an analysis of the derivation of $\Delta; \Gamma \vdash P; C$. A key case is RRes, which requires preservation of the condition in TRes that (Γ, C) is consistent. This is because a communication reduction consumes matching prefixes from a pair of dual partial session types, which therefore remain dual. All the technical result are proved correct in [34].

6 Conclusions, Related and Future Work

From the beginning of session types, channel endpoints were treated as linear resources so that each role in a protocol could be implemented by a unique agent. This approach is reinforced by several connections between session types and other linear type theories: the encodings of binary session types and multiparty session types into linear π-calculus types [12,29]; the Curry-Howard correspondence between binary session types and linear logic [6,35]; the connection between multiparty session types and linear logic [8,7].

Some session type systems generalise linearity. Vasconcelos [33] allows a session type to become non-linear, and sharable, when it reaches a state that is invariant with every subsequent message. Mostrous and Vasconcelos define *affine* session types, in which each endpoint must be used at most once and can be discarded with an explicit operator. In Fowler *et al.*'s [15] implementation of session types for the Links web programming language, affine typing allows sessions to be cancelled when exceptions (including dropped connections) occur. Caires and Pérez [5] use monadic types to describe cancellation (i.e. affine sessions) and non-determinism. Pruiksma and Pfenning [28] use adjoint logic to describe session cancellation and other behaviours including multicast and replication.

Usually linearity spreads, because a data structure containing linear values must also be linear. In the standard π-calculus, exceptions to this nature of linearity have been studied by Kobayashi in his work on deadlock-freedom [22] or on quasi-linearity [21]. Padovani [26] extends the linear π-calculus with composite regular types in such a way that data containing linear values can be shared among several processes. However, this sharing can occur only if there is no overlapping access to such values, which differs from our work where we have full sharing of values. On the other hand, we work directly with (multiparty) session types, whereas Padovani works with linear π-calculus and obtains his results via the encoding of session types into linear π-types [12].

Session types are related to the concept of *typestate* [30], especially in the work of Kouzapas *et al.* [23,24] which defines a typestate system for Java based on multiparty session types. Typestate systems require linear typing or some other form of alias control, to avoid conflicting state changes via multiple references. Approaches include the permission-based systems used in the Plural and Plaid languages [4,31] and the fine-grained approach of Militão *et al.* [25]. Crafa and Padovani [10] develop a "chemical" approach to concurrent typestate oriented

programming, allowing objects to be accessed and modified concurrently by several processes, each potentially changing only part of their state. Our approach is partly inspired by Fähndrich and DeLine's "adoption and focus" system [14], in which a shared stateful resource (in our case, a session channel) is separated from the linear key that enables it to be used. In this way the state changes of channels follow the standard session operations, channels can be shared (for example, stored in shared data structures), and access can be controlled by passing the key around the system.

Balzer *et al.* [2,3] support sharing of binary session channels by allowing locks to be acquired and released at points that are explicitly specified in the session type. Our approach with multiparty sessions is not based on locks, so it doesn't require runtime mechanisms for managing blocked processes and notifying them when locks are released.

We have presented a new system of multiparty session types which allows sharing of resources in a way that generalises the strictly linear or affine access control typical of session type systems. The key technical idea is to separate a channel from the capability of using the channel. This allows channels to be shared, while capabilities are linearly controlled. We use a form of existential typing to maintain the link between a channel and its capability while both are transmitted in messages.

We have proved communication safety, which is the basic property required of a session type system. An area of future work is to prove progress and deadlock-freedom properties along the lines of, for example, Coppo *et al.* [9]. Another possibility is to apply our techniques to functional languages with session types [17].

References

1. Ancona, D., et al.: Behavioral types in programming languages. Foundations and Trends in Programming Languages **3**(2–3), 95–230 (2016)
2. Balzer, S., Pfenning, F.: Manifest sharing with session types. PACMPL **1**(ICFP), 37:1–37:29 (2017)
3. Balzer, S., Toninho, B., Pfenning, F.: Manifest deadlock-freedom for shared session types. In: ESOP. LNCS, vol. 11423, pp. 611–639. Springer (2019)
4. Bierhoff, K., Aldrich, J.: PLURAL: checking protocol compliance under aliasing. In: ICSE Companion. pp. 971–972. ACM Press (2008)
5. Caires, L., Pérez, J.A.: Linearity, control effects, and behavioral types. In: ESOP. LNCS, vol. 10201, pp. 229–259. Springer (2017)
6. Caires, L., Pfenning, F.: Session types as intuitionistic linear propositions. In: CONCUR. LNCS, vol. 6269, pp. 222–236. Springer (2010)
7. Carbone, M., Lindley, S., Montesi, F., Schürmann, C., Wadler, P.: Coherence generalises duality: A logical explanation of multiparty session types. In: CONCUR. LIPIcs, vol. 59, pp. 33:1–33:15. Schloss Dagstuhl — Leibniz-Zentrum für Informatik (2016)
8. Carbone, M., Montesi, F., Schürmann, C., Yoshida, N.: Multiparty session types as coherence proofs. In: CONCUR. LIPIcs, vol. 42. Schloss Dagstuhl — Leibniz-Zentrum für Informatik (2015)

9. Coppo, M., Dezani-Ciancaglini, M., Yoshida, N., Padovani, L.: Global progress for dynamically interleaved multiparty sessions. Mathematical Structures in Computer Science **26**(2), 238–302 (2016)
10. Crafa, S., Padovani, L.: The chemical approach to typestate-oriented programming. ACM Transactions on Programming Languages and Systems **39**(3), 13:1–13:45 (2017)
11. Crary, K., Walker, D., Morrisett, G.: Typed memory management in a calculus of capabilities. In: POPL. pp. 262–275. ACM (1999)
12. Dardha, O., Giachino, E., Sangiorgi, D.: Session types revisited. In: PPDP. ACM (2012)
13. Deniélou, P., Yoshida, N., Bejleri, A., Hu, R.: Parameterised multiparty session types. Logical Methods in Computer Science **8**(4) (2012)
14. Fähndrich, M., DeLine, R.: Adoption and focus: Practical linear types for imperative programming. In: PLDI. pp. 13–24. ACM (2002)
15. Fowler, S., Lindley, S., Morris, J.G., Decova, S.: Exceptional asynchronous session types: session types without tiers. PACMPL **3**(POPL), 28:1–28:29 (2019)
16. Gay, S.J., Ravara, A. (eds.): Behavioural Types: From Theory to Tools. River Publishers (2017)
17. Gay, S.J., Vasconcelos, V.T.: Linear type theory for asynchronous session types. Journal of Functional Programming **20**(1), 19–50 (2010)
18. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: POPL. pp. 273–284. ACM Press (2008)
19. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: ESOP. LNCS, vol. 1381, pp. 122–138. Springer (1998)
20. Hüttel, H., et al.: Foundations of session types and behavioural contracts. ACM Computing Surveys **49**(1) (2016)
21. Kobayashi, N.: Quasi-linear types. In: POPL. pp. 29–42. ACM (1999)
22. Kobayashi, N.: A new type system for deadlock-free processes. In: CONCUR. LNCS, vol. 4137, pp. 233–247. Springer (2006)
23. Kouzapas, D., Dardha, O., Perera, R., Gay, S.J.: Typechecking protocols with Mungo and StMungo. In: PPDP. pp. 146–159. ACM (2016)
24. Kouzapas, D., Dardha, O., Perera, R., Gay, S.J.: Typechecking protocols with Mungo and StMungo: a session type toolchain for Java. Science of Computer Programming **155**, 52–75 (2018)
25. Militão, F., Aldrich, J., Caires, L.: Aliasing control with view-based typestate. In: FTFJP. pp. 7:1–7:7. ACM (2010)
26. Padovani, L.: Type reconstruction for the linear π-calculus with composite regular types. Logical Methods in Computer Science **11**(4) (2015)
27. Pierce, B.C.: Types and programming languages. MIT Press (2002)
28. Pruiksma, K., Pfenning, F.: A message-passing interpretation of adjoint logic. In: PLACES. Electronic Proceedings in Theoretical Computer Science, vol. 291, pp. 60–79. Open Publishing Association (2019)
29. Scalas, A., Dardha, O., Hu, R., Yoshida, N.: A linear decomposition of multiparty sessions for safe distributed programming. In: ECOOP. LIPIcs, vol. 74, pp. 24:1–24:31. Schloss Dagstuhl — Leibniz-Zentrum für Informatik (2017)
30. Strom, R.E., Yemini, S.: Typestate: A programming language concept for enhancing software reliability. IEEE Trans. Softw. Eng. **12**(1), 157–171 (1986)
31. Sunshine, J., Naden, K., Stork, S., Aldrich, J., Tanter, É.: First-class state change in Plaid. In: OOPSLA. pp. 713–732. ACM (2011)

32. Takeuchi, K., Honda, K., Kubo, M.: An interaction-based language and its typing system. In: PARLE. Springer LNCS, vol. 817, pp. 398–413 (1994)
33. Vasconcelos, V.T.: Fundamentals of session types. Inf. Comput. **217**, 52–70 (2012)
34. Voinea, A.L., Dardha, O., Gay, S.J.: Resource sharing via capability-based multiparty session types. Tech. rep., School of Computing Science, University of Glasgow (2019), http://www.dcs.gla.ac.uk/~ornela/publications/VDG19-Extended.pdf
35. Wadler, P.: Propositions as sessions. In: ICFP. pp. 273–286. ACM (2012)
36. Walker, D., Morrisett, J.G.: Alias types for recursive data structures. In: TIC. pp. 177–206. LNCS, Springer (2000)
37. Yoshida, N., Deniélou, P., Bejleri, A., Hu, R.: Parameterised multiparty session types. In: FOSSACS (2010)

A Producer Consumer Typing derivaton

We show the typing for the process definition of Produce:

$$S = \mu\, \mathtt{t}.\mathtt{b}\oplus!\mathsf{add}\,(\mathtt{Int})\,.\mathtt{c}\oplus!\mathsf{turn}\left(\{\rho_y \mapsto \overline{S}_b\}\right).\mathtt{c}\&?\mathsf{turn}\left(\{\rho_y \mapsto \overline{S}_b\}\right).t$$

$$\Delta = \mathsf{Produce} : (\mathtt{tr}\,(\rho_p)\,, \mathtt{tr}\,(\rho_y)\,, \mathtt{Int}, \{\rho_p \mapsto S_p\}, \{\rho_y \mapsto S_q\})$$

$$\Gamma = \mathtt{x} : \mathtt{tr}\,(\rho_p)\,, \mathtt{y} : \mathtt{tr}\,(\rho_y)\,, \mathsf{i} : \mathtt{Int}, \rho_p : \{\rho_p \mapsto S_p\}, \rho_y : \{\rho_y \mapsto S_q\}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash \mathtt{x} : \mathtt{tr}\,(\rho_x)\,, \mathtt{y} : \mathtt{tr}\,(\rho_y)\,, \mathsf{i} : \mathtt{Int}, \rho_x : \{\rho_x \mapsto S'_p\}, \rho_y : \{\rho_y \mapsto S_q\}; \{\rho_x \mapsto S'_p, \rho_y \mapsto S_q\}}{\Delta; \Gamma \vdash \mathsf{Produce}\langle x, y, \mathsf{i}{+}1, \rho_x, \rho_y\rangle; \{\rho_x \mapsto S'_p, \rho_y \mapsto S_q\}}\ \textsc{TCall}}{\Delta; \Gamma \vdash \mathtt{x}[\mathsf{c}]\&\{\mathsf{turn}(\rho_y).\mathsf{Produce}\langle x, y, \mathsf{i}{+}1, \rho_x, \rho_y\rangle\}; \{\rho_x \mapsto \mathsf{c}\&?\mathsf{turn}\,(\{\rho_y \mapsto S_q\})\,.S'_p\}}\ \textsc{TBr}}{\vdots}\ \textsc{TCap}}{\Delta; \Gamma \vdash \mathtt{x}[\mathsf{c}]\oplus\langle\mathsf{turn}(\rho_y)\rangle.\mathtt{x}[\mathsf{c}]\&\{\mathsf{turn}(\rho_y).\mathsf{Produce}\langle x, y, \mathsf{i}{+}1, \rho_x, \rho_y\rangle\}; \{\rho_x \mapsto S'_p, \rho_y \mapsto S'_q\}}\ \textsc{TSel}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash \mathtt{x} : \mathtt{tr}\,(\rho_x)\,, \mathtt{y} : \mathtt{tr}\,(\rho_y)\,, \rho_x : \{\rho_x \mapsto S'_p\}, \rho_y : \{\rho_y \mapsto S_q\}; \{\rho_x \mapsto S'_p, \rho_y \mapsto S_q\}}{\Delta; \Gamma \vdash \mathsf{Produce}\langle x, y, \mathsf{i}, \rho_p, \rho_y\rangle; \{\rho_x \mapsto S'_p, \rho_y \mapsto S_q\}}\ \textsc{TCall}}{\vdots}\quad \dfrac{\mathtt{y} : \mathtt{tr}\,(\rho_y)\,, \rho_y : \{\rho_y \mapsto S_q\}}{\Gamma \vdash \mathtt{y} : \mathtt{tr}\,(\rho_y)\,; \emptyset}\ \textsc{TVar}\quad \mathtt{x}, \rho_x : \{\rho_x \mapsto S_p\} \in \Gamma}{\Delta; \Gamma \vdash \mathtt{x}[\mathsf{c}]\oplus\langle\mathsf{buffer}(\mathsf{pack}(\rho_y, \mathtt{y}[\mathsf{b}]))\rangle.\mathsf{Produce}\langle \mathtt{x}, \mathtt{y}, \mathsf{i}, \rho_x, \rho_y\rangle;\ \{\rho_y \mapsto \mathtt{p}\oplus!l\,(\exists[\rho_y|\{\rho_y \mapsto S_q\}].\mathtt{tr}\,(\rho_y))\,.S'_p, \rho_y \mapsto S_q\}}\ \textsc{TSelP}$$

We show the typing for the process P starting from the Produce process call:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash \mathtt{x} : \mathtt{tr}\,(\rho_x)\,, \mathtt{y} : \mathtt{tr}\,(\rho_y)\,, \rho_x : \{\rho_x \mapsto S'_p\}, \rho_y : \{\rho_y \mapsto S_q\}; \{\rho_x \mapsto S'_p, \rho_y \mapsto S_q\}}{\Delta; \Gamma \vdash \mathsf{Produce}\langle x, y, \mathsf{i}, \rho_p, \rho_y\rangle; \{\rho_x \mapsto S'_p, \rho_y \mapsto S_q\}}\ \textsc{TCall}}{\vdots}\quad \dfrac{\mathtt{y} : \mathtt{tr}\,(\rho_y)\,, \rho_y : \{\rho_y \mapsto S_q\}}{\Gamma \vdash \mathtt{y} : \mathtt{tr}\,(\rho_y)\,; \emptyset}\ \textsc{TVar}\quad \mathtt{x}, \rho_x : \{\rho_x \mapsto S_p\} \in \Gamma}{\Delta; \Gamma \vdash \mathtt{x}[\mathsf{c}]\oplus\langle\mathsf{buffer}(\mathsf{pack}(\rho_y, \mathtt{y}[\mathsf{b}]))\rangle.\mathsf{Produce}\langle \mathtt{x}, \mathtt{y}, \mathsf{i}, \rho_x, \rho_y\rangle;\ \{\rho_y \mapsto \mathtt{p}\oplus!l\,(\exists[\rho_y|\{\rho_y \mapsto S_q\}].\mathtt{tr}\,(\rho_y))\,.S'_p, \rho_y \mapsto S_q\}}\ \textsc{TSelP}$$

Typing derivation for the consumer:

$$\Delta = \mathsf{C} : (\mathtt{tr}\,(\rho_x)\,, \{\rho_x \mapsto S\}), \mathsf{Consume} : (\mathtt{tr}\,(\rho_x)\,, \mathtt{tr}\,(\rho_y)\,, \{\rho_x \mapsto S\})$$

$$\Gamma = \mathtt{x} : \mathtt{tr}\,(\rho_x)\,, \mathtt{y} : \mathtt{tr}\,(\rho_y)\,, \rho_x : \{\rho_x \mapsto S\}$$

$$\Gamma' = \Gamma, \rho_y : \{\rho_y \mapsto S'_q\}$$

$$S = \mu\, \mathtt{t}.\mathtt{p}\&?\mathsf{turn}\left(\{\rho_y \mapsto S'_q\}\right).\mathtt{p}\oplus!\mathsf{turn}\,(\{\rho_y \mapsto S_q\})\,.t$$

We show the typing for the process definition of Consume:

$$\dfrac{\dfrac{\overline{\Gamma', \mathsf{i}:\mathtt{Int} \vdash \rho_y; \{\rho_y \mapsto S_q\}}\ \textsc{TCap} \quad \mathtt{x}:\mathtt{tr}\,(\rho_x), \rho_x : \{\rho_x \mapsto S_c\} \in \Gamma' \quad \begin{array}{c}\dfrac{\Gamma', \mathsf{i}:\mathtt{Int} \vdash x:\mathtt{tr}\,(\rho_x), y:\mathtt{tr}\,(\rho_y), \rho_x : \{\rho_x \mapsto S_c\}; \{\rho_x \mapsto S_c\}}{\Delta; \Gamma', \mathsf{i}:\mathtt{Int} \vdash \mathsf{Consume}\langle x, y, \rho_x\rangle; \{\rho_x \mapsto \mathsf{p}\oplus!\mathsf{turn}\,(\{\rho_y \mapsto S_q\})\,.S_c\}}\ \textsc{TCall} \\ \vdots\end{array}}{\dfrac{\Delta; \Gamma', \mathsf{i}:\mathtt{Int} \vdash \mathsf{x[p]}\oplus\langle\mathsf{turn}(\rho_y)\rangle.\mathsf{Consume}\langle x, y, \rho_x\rangle; \{\rho_x \mapsto \mathsf{p}\oplus!\mathsf{turn}\,(\{\rho_y \mapsto S_q\})\,.S_c, \rho_y \mapsto S_q\}}{\Delta; \Gamma' \vdash \mathsf{y[q]}\&\{\mathsf{send(i)}\}.\mathsf{x[p]}\oplus\langle\mathsf{turn}(\rho_y)\rangle.\mathsf{Consume}\langle x, y, \rho_x\rangle; \{\rho_x \mapsto \mathsf{p}\oplus!\mathsf{turn}\,(\{\rho_y \mapsto S_q\})\,.S_c, \rho_y \mapsto \mathsf{b}\&?\mathsf{send}\,(\mathtt{Int})\,.S_q\}}\ \textsc{TBr}}\ \textsc{TSel}$$

$$\dfrac{\dfrac{\dfrac{r \in \mathtt{Str}}{\Gamma' \vdash r:\mathtt{Str}; \emptyset}\ \textsc{TVal} \quad \mathsf{y}, \rho_y : \{\rho_y \mapsto S_q\} \in \Gamma' \quad \vdots}{\begin{array}{c}\Delta; \Gamma' \vdash \mathsf{y[b]}\oplus\langle\mathsf{request}(r)\rangle.\mathsf{y[b]}\&\{\mathsf{send(i)}\}.\mathsf{x[p]}\oplus\langle\mathsf{turn}(\rho_y)\rangle.\mathsf{Consume}\langle x, y, \rho_x\rangle; \{\rho_x \mapsto \mathsf{p}\oplus!\mathsf{turn}\,(\{\rho_y \mapsto S_q\})\,.S_c\}\otimes \\ \{\rho_y \mapsto \mathsf{b}\oplus!\mathsf{request}\,(\mathtt{Str})\,.\mathsf{b}\&?\mathsf{send}\,(\mathtt{Int})\,.S_q\}\end{array}}\ \textsc{TSel}}{\begin{array}{c}\Delta, \Gamma \vdash \mathsf{x[p]}\&\{\mathsf{turn}(\rho_y)\}.\mathsf{y[b]}\oplus\langle\mathsf{request}(r)\rangle.\mathsf{y[b]}\&\{\mathsf{send(i)}\}.\mathsf{x[p]}\oplus\langle\mathsf{turn}(\rho_y)\rangle.\mathsf{Consume}\langle x, y, \rho_x\rangle; \\ \{\rho_x \mapsto \mathsf{p}\&?\mathsf{turn}\left(\{\rho_y \mapsto S_q'\}\right).\mathsf{p}\oplus!\mathsf{turn}\,(\{\rho_y \mapsto S_q\})\,.S\}\end{array}}\ \textsc{TBr}$$

We show the typing for the process definition of C starting with the typing for Consume:

$$\dfrac{\dfrac{\textsc{TCall} \atop \Delta; \Gamma, \mathtt{y}:\mathtt{tr}\,(\rho_y), \rho_y : \{\rho_y \mapsto S_q'\} \vdash x:\mathtt{tr}\,(\rho_x), y:\mathtt{tr}\,(\rho_y), \rho_x : \{\rho_x \mapsto S_c'\}; \{\rho_x \mapsto S_c'\}}{\Delta; \Gamma, \mathtt{y}:\mathtt{tr}\,(\rho_y), \rho_y : \{\rho_y \mapsto S_q'\} \vdash \mathsf{Consume}\langle x, y, \rho_x\rangle; \{\rho_x \mapsto S_c'\}} \quad \mathtt{x}:\mathtt{tr}\,(\rho_x), \rho_x : \{\rho_x \mapsto S_c'\} \in \Gamma}{\Delta; \Gamma \vdash \mathsf{x[p]}\oplus\langle\mathsf{buffer}(\mathsf{pack}(\rho_y, \mathsf{y}))\rangle.\mathsf{Consume}\langle x, y, \rho_x\rangle; \{\rho_c \mapsto \mathsf{c}\&?l\left(\exists[\rho_y|\{\rho_y \mapsto S_q'\}].\mathtt{tr}\,(\rho_y)\right).S_c'\}}\ \textsc{TBrP}$$

Typing derivation for the buffer process $\mathsf{B}\langle \mathtt{x}, \rho_x\rangle$:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\dfrac{\dfrac{\mathtt{x}:\mathtt{tr}\,(\rho_x), \{\rho_x \mapsto S_b\} \in \Gamma, \mathsf{i}:\mathtt{Int}, r:\mathtt{Str}}{\Gamma, \mathsf{i}:\mathtt{Int}, r:\mathtt{Str} \vdash \mathtt{x}:\mathtt{tr}\,(\rho_x); \{\rho_x \mapsto S_b\}}\ \textsc{TVar}}{\Delta; \Gamma, \mathtt{x}:\mathtt{tr}\,(\rho_x), \mathsf{i}:\mathtt{Int}, r \vdash \mathsf{B}\langle x\rangle; \{\rho_x \mapsto S_b\}}\ \textsc{TCall} \\ \vdots\end{array} \quad \overline{\Gamma, \mathsf{i}:\mathtt{Int}, r:\mathtt{Str} \vdash \mathsf{i}; \emptyset}\ \textsc{TVal} \quad x \in \Gamma, \mathtt{x}:\mathtt{tr}\,(\rho_x), \mathsf{i}:\mathtt{Int}, r:\mathtt{Str}}{\begin{array}{c}\Delta; \Gamma, \mathtt{x}:\mathtt{tr}\,(\rho_x), \mathsf{i}:\mathtt{Int}, r:\mathtt{Str} \vdash \mathsf{x[p]}\oplus\langle\mathsf{send(i)}\rangle.\mathsf{B}\langle x, \rho_x\rangle; \{\rho_x \mapsto \mathsf{q}\oplus!\mathsf{send}\,(\mathtt{Int})\,.S_b\} \\ \vdots\end{array}}\ \textsc{TSel} \quad \mathtt{x}:\mathtt{tr}\,(\rho_x), \rho_x : \{\rho_x \mapsto \mathsf{q}\oplus!\mathsf{send}\,(\mathtt{Int})\,.S_b\} \in \Gamma}{\begin{array}{c}\Delta; \Gamma, \mathsf{i}:\mathtt{Int} \vdash \mathsf{x[q]}\&\{\mathsf{request}(r)\}.\mathsf{x[p]}\oplus\langle\mathsf{send(i)}\rangle.\mathsf{B}\langle x, \rho_x\rangle; \{\rho_x \mapsto \mathsf{q}\&?\mathsf{request}\,(\mathtt{Str})\,.\mathsf{q}\oplus!\mathsf{send}\,(\mathtt{Int})\,.S_b\} \\ \vdots\end{array}}\ \textsc{TBr} \quad \mathtt{x}:\mathtt{tr}\,(\rho_x), \{\rho_x \mapsto \mathsf{q}\&?\mathsf{request}\,(\mathtt{Str})\,.\mathsf{q}\oplus!\mathsf{send}\,(\mathtt{Int})\,.S_b\} \in \Gamma}{\Delta; \Gamma \vdash \mathsf{x[q]}\&\{\mathsf{add(i)}\}.\mathsf{x[q]}\&\{\mathsf{request}(r)\}.\mathsf{x[p]}\oplus\langle\mathsf{send(i)}\rangle.\mathsf{B}\langle \mathtt{x}, \rho_x\rangle; \{\rho_x \mapsto \mathsf{q}\&?\mathsf{add}\,(\mathtt{Int})\,.\mathsf{q}\&?\mathsf{request}\,(\mathtt{Str})\,.\mathsf{q}\oplus!\mathsf{send}\,(\mathtt{Int})\,.S_b\}}\ \textsc{TBr}$$

B Proofs

Proposition 1. *If* $(\Gamma, C) \longrightarrow^* (\Gamma', C')$ *then* $\mathrm{dom}\,(\Gamma) = \mathrm{dom}\,(\Gamma')$ *and* $\mathrm{dom}\,(C) = \mathrm{dom}\,(C')$.

Lemma 1. *If* $(\Gamma; C) \longrightarrow (\Gamma'; C')$ *and* $(\Gamma; C)$ *is* consistent (resp. complete), *then so is* $(\Gamma'; C')$.

Corollary 1. *If* $(\Gamma, C_1 \otimes C_2)$ *is consistent and* $(\Gamma, C_1) \longrightarrow^* (\Gamma, C_1')$, *then* $(\Gamma, C_1' \otimes C_2)$ *is consistent.*

Proposition 2. *For all multiparty session processes P, P' if $\Delta; \Gamma \vdash P; C$ and $P \equiv P'$, then $\Delta; \Gamma \vdash P'; C$.*

Proof. The proof proceeds by induction on the structural congruence $\equiv$.

Definition 13 (Substitution of Capabilities). *Substitution of capabilities is defined as:*

$$\{\rho \mapsto S\}[\rho'/\rho_2] = \{\rho \mapsto S\} \qquad \{\rho \mapsto S\}[\rho'/\rho] = \{\rho' \mapsto S\}$$

$$\emptyset[\rho'/\rho] = \emptyset \qquad (C_1 \otimes C_2)[\rho'/\rho] = C_1[\rho'/\rho] \otimes C_2[\rho'/\rho]$$

Lemma 2 (Substitution).

1. *If $\Delta; \Gamma, \mathtt{x} : U \vdash P; C$, $\Gamma' \vdash \mathtt{v} : U; \emptyset$, then $\Delta; \Gamma, \Gamma' \vdash P\{v/x\}; C$.*
2. *If $\Delta; \Gamma, \rho : \{\rho \mapsto S\} \vdash P; C \otimes \{\rho \mapsto S\}$, $\Gamma' \vdash \rho' : \{\rho' \mapsto S\}; \{\rho' \mapsto S\}$ and $(\Gamma, \Gamma'; C \otimes \{\rho' \mapsto S\})$ consistent, then $\Delta; \Gamma, \Gamma' \vdash P\{\rho'/\rho\}; C \otimes \{\rho' \mapsto S\}$.*

Proof. By induction on the derivation of $\Delta; \Gamma, x : U \vdash P; C$, with a case analysis on the last rule applied.

Definition 14. *For all pairs of multiparty session typing contexts and capability collections (Γ, C), (Γ', C'), the relation $C \leqslant_{\mathsf{S}} C'$ holds iff $\mathrm{dom}(\Gamma) = \mathrm{dom}(\Gamma')$, $\mathrm{dom}(C) = \mathrm{dom}(C')$ and $\forall \mathtt{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto U\} \in \mathrm{dom}(\Gamma)$ then $C(\rho) \leqslant_{\mathsf{S}} C'(\rho)$. The following rule is defined, corresponding to 0 or more consecutive applications of* TSub*:*

$$\dfrac{\overset{\text{TMSub}}{\dfrac{\Delta; \Gamma \vdash P; C \qquad C' \leqslant_{\mathsf{S}} C}{\Delta; \Gamma \vdash P; C'}}}{}$$

Proposition 3. *If $\Delta; \Gamma \vdash P \mid Q; C$ then there exist C_1, C_2, C_1', C_2' such that $C = C_1 \otimes C_2$, $C_1 \leqslant_{\mathsf{S}} C_1'$, $C_2 \leqslant_{\mathsf{S}} C_2'$, $\Delta; \Gamma \vdash P; C_1'$, $\Delta; \Gamma \vdash Q; C_2'$. Moreover:*

$$\dfrac{\overset{\text{TMSub}}{\dfrac{\Delta; \Gamma \vdash P; C_1' \qquad C_1 \leqslant_{\mathsf{S}} C_1'}{\Delta; \Gamma \vdash P; C_1}} \qquad \overset{\text{TMSub}}{\dfrac{\Delta; \Gamma \vdash Q; C_2' \qquad C_2 \leqslant_{\mathsf{S}} C_2'}{\Delta; \Gamma \vdash Q; C_2}}}{\Delta; \Gamma \vdash P \mid Q; C_1 \otimes C_2}\ \text{TPar}$$

$$\textit{iff}\ \dfrac{\dfrac{\Delta; \Gamma \vdash P; C_1' \quad \Delta; \Gamma \vdash Q; C_2'}{\Delta; \Gamma \vdash P \mid Q; C_1' \otimes C_2'}\ \text{TPar} \qquad C_1 \otimes C_2 \leqslant_{\mathsf{S}} C_1' \otimes C_2'}{\Delta; \Gamma \vdash P \mid Q; C_1 \otimes C_2}\ \text{TMSub}$$

$$\textit{iff}\ \dfrac{\dfrac{\Delta; \Gamma \vdash P; C_1' \qquad \dfrac{\Delta; \Gamma \vdash Q; C_2' \qquad C_2 \leqslant_{\mathsf{S}} C_2'}{\Delta; \Gamma \vdash Q; C_2}\ \text{TMSub}}{\Delta; \Gamma \vdash P \mid Q; C_1' \otimes C_2}\ \text{TPar} \qquad C_1 \otimes C_2 \leqslant_{\mathsf{S}} C_1' \otimes C_2}{\Delta; \Gamma \vdash P \mid Q; C_1 \otimes C_2}\ \text{TMSub}$$

$$\textit{iff}\ \dfrac{\dfrac{\dfrac{\Delta; \Gamma \vdash P; C_1' \qquad C_1 \leqslant_{\mathsf{S}} C_1'}{\Delta; \Gamma \vdash P; C_1}\ \text{TMSub} \qquad \Delta; \Gamma \vdash Q; C_2'}{\Delta; \Gamma \vdash P \mid Q; C_1 \otimes C_2'}\ \text{TPar} \qquad C_1 \otimes C_2 \leqslant_{\mathsf{S}} C_1 \otimes C_2'}{\Delta; \Gamma \vdash P \mid Q; C_1 \otimes C_2}\ \text{TMSub}$$

Proposition 4. *$\Delta; \Gamma_1 \vdash (\nu\, s : \Gamma_2)P; C_1$ then there exist C'_1, C'_2 such that $C_1 \leqslant_{\mathsf{S}} C'_1$, $C_2 \leqslant_{\mathsf{S}} C'_2$ and $\Delta; \Gamma_1, \Gamma_2 \vdash P; C'_1 \otimes C'_2$. Moreover:*

$$\dfrac{\dfrac{\Delta; \Gamma_1, \Gamma_2 \vdash P; C'_1 \otimes C'_2 \qquad C_1 \otimes C_2 \leqslant_{\mathsf{S}} C'_1 \otimes C'_2}{\Delta; \Gamma_1, \Gamma_2 \vdash P; C_1 \otimes C_2}\,\text{TMSub}}{\Delta; \Gamma_1 \vdash (\nu\, s : \Gamma_2)P; C_1}\,\text{TRes} \qquad \textit{iff} \qquad \dfrac{\dfrac{\dfrac{\Delta; \Gamma_1, \Gamma_2 \vdash P; C'_1 \otimes C'_2 \qquad C_2 \leqslant_{\mathsf{S}} C'_2}{\Delta; \Gamma_1, \Gamma_2 \vdash P; C'_1 \otimes C_2}\,\text{TMSub}}{\Delta; \Gamma_1 \vdash (\nu\, s : \Gamma_2)P; C'_1}\,\text{TRes} \qquad C_1 \leqslant_{\mathsf{S}} C'_1}{\Delta; \Gamma_1 \vdash (\nu\, s : \Gamma_2)P; C_1}\,\text{TMSub}$$

Proposition 5 (Subtyping normalisation). *If $\Delta; \Gamma \vdash P; C$ then there exists a derivation that proves the judgement by only applying rule* TMSub *on the conclusions of* TBr*,* TSel*,* TBrP*,* TSelP*, and* TCall*.*

Proof (??). By induction on the derivation of the reduction $P \to P'$:

– **case RCom** We have: $P = \mathtt{s[p][q]}\&_{i\in I}\{l_i(x_i).\, Q_i\} \mid \mathtt{s[q][p]}\oplus\langle l_j(v)\rangle.Q \;\longrightarrow\; Q_j\{v/x_j\} \mid Q = P'$, where if $j \in I$ and $fv(v) = \emptyset$, and $C = C_1 \otimes \{\rho_p \mapsto \mathtt{p}\&_{i\in I}?l_i\,(U_i)\,.S_i\} \otimes C_2 \otimes \{\rho_q \mapsto \mathtt{q}\oplus_{j\in I}!l_j\,(U)\,.S\}$. We distinguish two cases based on whether the payload communicated has a capability attached or not. **In the first case,** the payload has no capability attached. By inversion of TPar and TBr/TSel, allowing for (possibly vacuous) instances of TMSub as per Prop. 5, there exist C_1, C_2 such that $C = C_1 \otimes C_2$, and C'_1, C'_2 such that:

$$\dfrac{\dfrac{\dfrac{\Delta; \Gamma, x_i : U'_i \vdash Q_i; C_1 \otimes \{\rho_p \mapsto S'_i\} \qquad \mathtt{s[p]} : \mathtt{tr}\,(\rho_p)\,, \rho_p : \{\rho_p \mapsto S'_i\} \in \Gamma}{\Delta; \Gamma \vdash \mathtt{s[p][q]}\&_{i\in I}\{l_i(x_i).\, Q_i\}; C'_1 \otimes \{\rho_p \mapsto \mathtt{p}\&_{i\in I}?l_i\,(U'_i)\,.S'_i\}}\,\text{TBr} \qquad C_1 \otimes \{\rho_p \mapsto \mathtt{p}\&_{i\in I}?l_i\,(U_i)\,.S_i\} \leqslant_{\mathsf{S}} C'_1 \otimes \{\rho_p \mapsto \mathtt{p}\&_{i\in I}?l_i\,(U'_i)\,.S'_i\}}{\Delta; \Gamma \vdash \mathtt{s[p][q]}\&_{i\in I}\{l_i(x_i).\, Q_i\}; C_1 \otimes \{\rho_p \mapsto \mathtt{p}\&_{i\in I}?l_i\,(U_i)\,.S_i\}}\,\text{TMSub} \qquad \dfrac{\dfrac{\dfrac{v \in U'}{\Gamma \vdash v : U'; \emptyset}\,\text{TVal} \qquad \mathtt{s[q]} : \mathtt{tr}\,(\rho_q)\,, \rho_q : \{\rho_q \mapsto S'\} \in \Gamma \qquad \Delta; \Gamma \vdash Q; C'_2 \otimes \{\rho_q \mapsto S'\}}{\Delta; \Gamma \vdash \mathtt{s[q][p]}\oplus\langle l_j(v)\rangle.Q; C'_2 \otimes \{\rho_q \mapsto \mathtt{q}\oplus_{j\in I}!l_j\,(U')\,.S'\}}\,\text{TSel} \qquad C_2 \otimes \{\rho_q \mapsto \mathtt{q}\oplus_{j\in I}!l_j\,(U)\,.S\} \leqslant_{\mathsf{S}} C'_2 \otimes \{\rho_q \mapsto \mathtt{q}\oplus_{j\in I}!l_j\,(U')\,.S'\}}{\Delta; \Gamma \vdash \mathtt{s[q][p]}\oplus\langle l_j(v)\rangle.Q; C_2 \otimes \{\rho_q \mapsto \mathtt{q}\oplus_{j\in I}!l_j\,(U)\,.S\}}\,\text{TMSub}}{\Delta; \Gamma \vdash \mathtt{s[p][q]}\&_{i\in I}\{l_i(x_i).\, Q_i\} \mid \mathtt{s[q][p]}\oplus\langle l_j(v)\rangle.Q; C}\,\text{TPar}$$

From the consistency of $(\Gamma; C) = (\Gamma; C_1 \otimes \{\rho_p \mapsto \mathtt{p}\&_{i\in I}?l_i\,(U_i)\,.S_i\} \otimes C_2 \otimes \{\rho_q \mapsto \mathtt{q}\oplus_{j\in I}!l_j\,(U)\,.S\})$, and since $C_1 \otimes \{\rho_p \mapsto \mathtt{p}\&_{i\in I}?l_i\,(U_i)\,.S_i\} \leqslant_{\mathsf{S}} C'_1 \otimes \{\rho_p \mapsto \mathtt{p}\&_{i\in I}?l_i\,(U'_i)\,.S'_i\}$ and $C_2 \otimes \{\rho_q \mapsto \mathtt{q}\oplus_{j\in I}!l_j\,(U)\,.S\} \leqslant_{\mathsf{S}} C'_2 \otimes \{\rho_q \mapsto \mathtt{q}\oplus_{j\in I}!l_j\,(U')\,.S'\}$ we also have: $U' \leqslant_{\mathsf{S}} U'_i$.
Let $C' = C'_1 \otimes \{\rho_p \mapsto \mathtt{p}\&_{i\in I}?l_i\,(U'_i)\,.S'_i\} \otimes C'_2 \otimes \{\rho_q \mapsto \mathtt{q}\oplus_{j\in I}!l_j\,(U')\,.S'\}$.
Before proceeding, we prove $(\Gamma; C) \longrightarrow (\Gamma; C')$, and therefore, $(\Gamma; C) \longrightarrow^* (\Gamma; C')$:
We observe that:

$$(\mathtt{s[p]} : \mathtt{tr}\,(\rho_p)\,, \mathtt{s[q]} : \mathtt{tr}\,(\rho_q)\,; \{\rho_p \mapsto \mathtt{p}\&_{i\in I}?l_i\,(U_i)\,.S_i, \rho_q \mapsto \mathtt{q}\oplus_{j\in I}!l_j\,(U)\,.S\}) \;\longrightarrow$$

$$(\mathtt{s[p]} : \mathtt{tr}\,(\rho_p)\,, \mathtt{s[q]} : \mathtt{tr}\,(\rho_q)\,; \{\rho_p \mapsto S'_j, \rho_q \mapsto S'\})$$

since Γ is consistent by hypothesis, and therefore $\mathtt{unf}((\mathtt{p}\&_{i\in I}?l_i\,(U_i)\,.S_i)\upharpoonright\mathtt{q})$ and $\mathtt{unf}((\mathtt{q}\oplus_{j\in I}!l_j\,(U)\,.S)\upharpoonright\mathtt{p})$ have at least l_j in common, with compatible payload types as per Def. 12.
Then we can prove the following statement:

$$(\Gamma; C_1\otimes C_2\otimes\{\rho_p\mapsto\mathtt{p}\&_{i\in I}?l_i\,(U_i)\,.S_i,\rho_q\mapsto\mathtt{q}\oplus_{j\in I}!l_j\,(U)\,.S\})\longrightarrow(\Gamma; C_1'\otimes C_2'\otimes\{\rho_p\mapsto S_j',\rho_q\mapsto S'\})$$

by induction on the size of $C_1\otimes C_2$. The base case is given above, while for the inductive case we apply the induction hypothesis use the subtyping relations between the capability sets to conclude by the inductve rule of Def. 12.
We can now continue proving the main statement, observing:

$(\Gamma; C')$ is consistent	by Lem. 1
$\Delta;\Gamma, x_i : U_i'\vdash Q_i; C_1'\otimes\{\rho_p\mapsto S_i'\}$	from the premise of TBr
$\Delta;\Gamma, x_i : U'\vdash Q_i; C_1'\otimes\{\rho_p\mapsto S_i'\}$	from the premise of TSub
$\Gamma\vdash v : U'$	from the premise of TSel
$\Delta;\Gamma\vdash Q_i\{v/x_i\}; C_1'\otimes\{\rho_p\mapsto S_i'\}$	from the above and Lem. 2

Therefore we conclude by:

$$\frac{\Delta;\Gamma\vdash Q_i\{v/x_i\}; C_1'\otimes\{\rho_p\mapsto S_i'\}\qquad\Delta;\Gamma\vdash Q; C_2'\otimes\{\rho_q\mapsto S'\}}{\Delta;\Gamma\vdash Q_j\{v/x_j\}\mid Q; C_1'\otimes\{\rho_p\mapsto S_i'\}\otimes C_2'\otimes\{\rho_q\mapsto S'\}}\ \textsc{TPar}$$

Second case case, the payload has a capability attached, which means that the payload in itself must be a capability.
By inversion of TPar and TBr/TSel, allowing (possibly vacuous) instances of TMSub as per Prop. 5, there exist C_1, C_2 such that $C = C_1\otimes C_2$, and C_1', C_2' such that:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Delta;\Gamma,\rho_i:\{\rho_i\mapsto U_i'\}\vdash Q_i; C_1''\otimes\{\rho_p\mapsto S_i'\}\otimes\{\rho_i\mapsto U_i'\}\quad \mathtt{s}[\mathtt{p}]:\mathtt{tr}\,(\rho_p)\,,\rho_p:\{\rho_p\mapsto S_i'\}\in\Gamma}{\Delta;\Gamma\vdash\mathtt{s}[\mathtt{p}][\mathtt{q}]\&_{i\in I}\{l_i(x_i).\,Q_i\}; C_1'=C_1''\otimes\{\rho_p\mapsto\mathtt{p}\&_{i\in I}?l_i\,(\{\rho_i\mapsto U_i'\})\,.S_i'\}}\ \textsc{TBr}}{\vdots\qquad C_1\leqslant_{\mathsf{S}} C_1'}\ \textsc{TMSub}}{\Delta;\Gamma\vdash\mathtt{s}[\mathtt{p}][\mathtt{q}]\&_{i\in I}\{l_i(x_i).\,Q_i\}; C_1}\ \textsc{TPar}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathtt{s}[\mathtt{q}]:\mathtt{tr}\,(\rho_q)\,,\rho_q:\{\rho_q\mapsto S'\}\in\Gamma\quad\Delta;\Gamma\vdash Q; C_2''\otimes\{\rho_q\mapsto S'\}\quad\overline{\Gamma\vdash\rho:\{\rho\mapsto U'\};\{\rho\mapsto U'\}}^{\,\textsc{TCap}}}{\Delta;\Gamma\vdash\mathtt{s}[\mathtt{q}][\mathtt{p}]\oplus\langle l_j(v)\rangle.Q; C_2'=C_2''\otimes\{\rho_q\mapsto\mathtt{q}\oplus_{j\in I}!l_j\,(\{\rho\mapsto U'\})\,.S'\}\otimes\{\rho\mapsto U'\}}\ \textsc{TSel}}{\vdots\qquad C_2\leqslant_{\mathsf{S}} C_2'}\ \textsc{TMSub}}{\Delta;\Gamma\vdash\mathtt{s}[\mathtt{q}][\mathtt{p}]\oplus\langle l_j(v)\rangle.Q; C_2}}{\Delta;\Gamma\vdash\mathtt{s}[\mathtt{p}][\mathtt{q}]\&_{i\in I}\{l_i(x_i).\,Q_i\}\mid\mathtt{s}[\mathtt{q}][\mathtt{p}]\oplus\langle l_j(v)\rangle.Q; C}\ \textsc{TPar}$$

From the consistency of $(\Gamma; C) = (\Gamma; C_1\otimes C_2) = (\Gamma; C_1^{\diamond}\otimes\{\rho_p\mapsto\mathtt{p}\&_{i\in I}?l_i\,(U_i)\,.S_i\}\otimes C_2^{\diamond})$, and since $C_1\leqslant_{\mathsf{S}} C_1'$ and $C_2\leqslant_{\mathsf{S}} C_2'$ we also have: $U'\leqslant_{\mathsf{S}} U_i'$.
Let $C' = C_1'\otimes C_2'$.
Before proceeding, we prove $(\Gamma; C)\longrightarrow(\Gamma; C')$, and therefore, $(\Gamma; C)\longrightarrow^*(\Gamma; C')$:

We observe that:

$$(\mathtt{s}[\mathtt{p}] : \mathtt{tr}\,(\rho_p)\,, \mathtt{s}[\mathtt{q}] : \mathtt{tr}\,(\rho_q)\,; \{\rho_p \mapsto \mathtt{p}\&_{i\in I}?l_i\,(U_i)\,.S_i, \rho_q \mapsto \mathtt{q}\oplus_{j\in I}!l_j\,(U)\,.S\}) \longrightarrow$$

$$(\mathtt{s}[\mathtt{p}] : \mathtt{tr}\,(\rho_p)\,, \mathtt{s}[\mathtt{q}] : \mathtt{tr}\,(\rho_q)\,; \{\rho_p \mapsto S'_j, \rho_q \mapsto S'\})$$

since Γ is consistent by hypothesis, and therefore $\mathtt{unf}((\mathtt{p}\&_{i\in I}?l_i\,(U_i)\,.S_i) \upharpoonright \mathtt{q})$ and $\mathtt{unf}((\mathtt{q}\oplus_{j\in I}!l_j\,(U)\,.S) \upharpoonright \mathtt{p})$ have at least l_j in common, with compatible payload types as per Def. 12.
Then we can prove the following statement:

$$(\Gamma; C_1 \otimes C_2 \otimes \{\rho_p \mapsto \mathtt{p}\&_{i\in I}?l_i\,(U_i)\,.S_i, \rho_q \mapsto \mathtt{q}\oplus_{j\in I}!l_j\,(U)\,.S\}) \longrightarrow (\Gamma; C'_1 \otimes C'_2 \otimes \{\rho_p \mapsto S'_j, \rho_q \mapsto S'\})$$

by induction on the size of $C_1 \otimes C_2$. The base case is given above, while for the inductive case we apply the induction hypothesis use the subtyping relations between the capability sets to conclude by the inductve rule of Def. 12.
We have $\Delta; \Gamma, x_j : U_j \vdash Q_j; C_1 \otimes C_j \otimes \{\rho_p \mapsto S_j\}$. By applying Lem. 2 we obtain $\Delta; \Gamma \vdash Q_j\{v/x_j\}; C_1 \otimes C' \otimes \{\rho_p \mapsto S_j\}$.
We can now continue proving the main statement, observing:

$$\begin{array}{ll} (\Gamma; C') \text{ is consistent} & \text{by Lem. 1} \\ \Delta; \Gamma, x_i : U'_i \vdash Q_i; C'_1 \otimes \{\rho_p \mapsto S'_i\} & \text{from the premise of } \textsc{TBr} \\ \Delta; \Gamma, x_i : U'' \vdash Q_i; C'_1 \otimes \{\rho_p \mapsto S'_i\} & \text{from the premise of } \textsc{TSub} \end{array}$$

We have $\Delta; \Gamma, x_j : U_j \vdash Q_j; C_1 \otimes C_j \otimes \{\rho_p \mapsto S_j\}$. By applying Lem. 2 we obtain $\Delta; \Gamma \vdash Q_j\{v/x_j\}; C_1 \otimes C' \otimes \{\rho_p \mapsto S_j\}$.

– **case RComP**
We have: $P = \mathtt{s}[\mathtt{p}][\mathtt{q}]\ \&_{i\in I}\ ?l_i(\mathsf{pack}(\rho_i, \mathtt{v}_i)).Q_i \mid \mathtt{s}[\mathtt{q}][\mathtt{p}]\oplus\langle l_j(\mathsf{pack}(\rho, \mathtt{v}))\rangle.Q \longrightarrow Q_j\{\mathtt{v}/\mathtt{v}_j\} \mid Q = P'$, where: if $j \in I$ and $fv(v) = \emptyset$.
$R = \mathtt{s}[\mathtt{p}][\mathtt{q}]\ \&_{i\in I}\ ?l_i(\mathsf{pack}(\rho_i, \mathtt{v}_i)).Q_i$
$R' = \mathtt{s}[\mathtt{q}][\mathtt{p}]\oplus\langle l_j(\mathsf{pack}(\rho, \mathtt{v}))\rangle.Q$
$P = R \mid R'$.

$$\frac{\begin{array}{c}\Delta; \Gamma \vdash R; C_1 \otimes \{\rho_p \mapsto \mathtt{p}\&_{i\in I}?l_i\,(\exists[\rho'_i|\{\rho'_i \mapsto U_i\}].\mathtt{tr}\,(\rho'_i))\,.S_i\} \\ \Delta; \Gamma \vdash R'; C_2 \otimes \{\rho \mapsto U\} \otimes \{\rho_q \mapsto \mathtt{p}\oplus_{i\in I}!l_i\,(\exists[\rho|\{\rho \mapsto U\}].\mathtt{tr}\,(\rho))\,.S_i\}\end{array}}{\begin{array}{c}\Delta; \Gamma \vdash R \mid R'; C_1 \otimes \{\rho_p \mapsto \mathtt{p}\&_{i\in I}?l_i\,(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}\,(\rho_i))\,.S_i\} \otimes \\ C_2 \otimes \{\rho \mapsto U\} \otimes \{\rho_q \mapsto \mathtt{q}\oplus_{j\in I}!l_j\,(\exists[\rho|\{\rho \mapsto U\}].\mathtt{tr}\,(\rho))\,.S\}\end{array}}\ \textsc{TPar}$$

We continue with the derivation for process R:

$$\frac{\Delta; \Gamma, v_i : \mathtt{tr}\,(\rho_i) \vdash Q_i; C_1 \otimes \{\rho_p \mapsto S_i\} \quad \mathtt{s}[\mathtt{p}] : \mathtt{tr}\,(\rho_p) \in \Gamma}{\Delta; \Gamma \vdash \mathtt{s}[\mathtt{p}][\mathtt{q}]\ \&_{i\in I}\ ?l_i(\mathsf{pack}(\rho_i, \mathtt{v}_i)).Q; C_1 \otimes \{\rho_p \mapsto \mathtt{p}\&_{i\in I}?l_i\,(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}\,(\rho_i))\,.S_i\}}\ \textsc{TBrP}$$

We continue with the derivation for process R':

$$\frac{\Gamma \vdash v : \mathtt{tr}\,(\rho)\,; \emptyset \quad \mathtt{s}[\mathtt{q}] : \mathtt{tr}\,(\rho_q) \in \Gamma \quad \Delta; \Gamma \vdash Q; C_2 \otimes \{\rho_q \mapsto S\} \otimes \{\rho \mapsto U\} \quad j \in I}{\Delta; \Gamma \vdash \mathtt{s}[\mathtt{q}][\mathtt{p}]\oplus\langle l_j(\mathsf{pack}(\rho, \mathtt{v}))\rangle.Q; C_2 \otimes C' \otimes \{\rho_q \mapsto \mathtt{q}\oplus_{j\in I}!l_j\,(\exists[\rho|\{\rho \mapsto U\}].\mathtt{tr}\,(\rho))\,.S\}}\ \textsc{TSelP}$$

We have $\Delta; \Gamma, v_j : \mathtt{tr}\,(\rho_j) \vdash Q_j; C_1 \otimes \{\rho_p \mapsto S_j\}$. By applying Lem. 2 we obtain $\Delta; \Gamma \vdash Q_j\{v/v_j\}; C_1 \otimes \{\rho_p \mapsto S_j\}$.

$$\frac{\Delta; \Gamma \vdash Q_j\{v/v_j\}; C_1 \otimes \{\rho_p \mapsto S_j\} \quad \Delta; \Gamma \vdash Q; C_2 \otimes \{\rho_q \mapsto S\} \otimes \{\rho \mapsto U\}}{\Delta; \Gamma \vdash Q_j\{v/v_j\} \mid Q; C_1 \otimes C_2 \otimes \{\rho_p \mapsto S_j\} \otimes \{\rho_q \mapsto S\} \otimes \{\rho \mapsto U\}}\ \textsc{TPar}$$

By definition Def. 12: $\Gamma; \{\rho_p \mapsto \mathtt{p}\&_{i\in I} ?l_i\,(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}\,(\rho_i))\,.S_i\} \otimes \{\rho_q \mapsto \mathtt{q} \oplus_{j\in I} !l_j\,(\exists[\rho|\{\rho \mapsto U\}].\mathtt{tr}\,(\rho))\,.S\} \longrightarrow \Gamma; \{\rho_p \mapsto S_j\} \otimes \{\rho_q \mapsto S\}$.
Therefore: $C_1 \otimes \{\rho_p \mapsto \mathtt{p}\&_{i\in I} ?l_i\,(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}\,(\rho_i))\,.S_i\} \otimes C_2 \otimes \{\rho \mapsto U\} \otimes \{\rho_q \mapsto \mathtt{q} \oplus_{j\in I} !l_j\,(\exists[\rho|\{\rho \mapsto U\}].\mathtt{tr}\,(\rho))\,.S\} \longrightarrow^* C_1 \otimes C_2 \otimes \{\rho_p \mapsto S_j\} \otimes \{\rho_q \mapsto S\} \otimes \{\rho \mapsto U\}$ as required.

– **case RCall**
We have: $P = \mathsf{def}\, X\langle\tilde{x}\rangle = Q_X \,\mathsf{in}\, (X\langle\tilde{v}\rangle \mid Q) \longrightarrow \mathsf{def}\, X\langle\tilde{x}\rangle = Q_X \,\mathsf{in}\, (Q_X\{\tilde{v}/\tilde{x}\} \mid Q) = P'$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\forall i \in \{1..n\} \qquad \Gamma \vdash v_i : U_i; C'_i}{\Delta, X : \tilde{U}; \Gamma \vdash X\langle\tilde{v}\rangle; C'_X = C'_1 \otimes ... \otimes C'_n}\ \textsc{TCall} \qquad C_X \leqslant_{\mathsf{S}} C'_X}{\Delta, X : \tilde{U}; \Gamma \vdash X\langle\tilde{v}\rangle; C_X}\ \textsc{TMSub} \qquad \Delta, X : \tilde{U}; \Gamma \vdash Q; C_Q}{\Delta, X : \tilde{U}; \Gamma \vdash (X\langle\tilde{v}\rangle \mid Q); C}\ \textsc{TPar} \\ \vdots \\ \dfrac{\Delta, X : \tilde{U}; \tilde{x} : \tilde{U} \vdash Q_X; \tilde{C} \qquad \vdots}{\Delta; \Gamma \vdash \mathsf{def}\, X\langle\tilde{x}\rangle = Q_X \,\mathsf{in}\, (X\langle\tilde{v}\rangle \mid Q); C}\ \textsc{TDef}$$

Observe that from $\Delta, X : \tilde{U}; \tilde{x} : \tilde{U} \vdash Q_X; \tilde{C}$, by applying Lem. 2 n times (each time obtaining a consistent context) we obtain $\Delta, X : \tilde{U}; \Gamma \vdash Q_X\{\tilde{v}/\tilde{x}\}; C'_X$. By applying TMSub we obtain $\Delta, X : \tilde{U}; \Gamma \vdash Q_X\{\tilde{v}/\tilde{x}\}; C_X$.

$$\dfrac{\Delta, X : \tilde{U}; \tilde{x} : \tilde{U} \vdash Q_X; \tilde{C} \qquad \dfrac{\Delta, X : \tilde{U}; \Gamma \vdash Q_X\{\tilde{v}/\tilde{x}\}; C_X \quad \Delta, X : \tilde{U}; \Gamma \vdash Q; C_Q}{\Delta, X : \tilde{U}; \Gamma \vdash Q_X\{\tilde{v}/\tilde{x}\} \mid Q; C}\ \textsc{TPar}}{\Delta; \Gamma \vdash \mathsf{def}\, X\langle\tilde{x}\rangle = Q_X \,\mathsf{in}\, (Q_X\{\tilde{v}/\tilde{x}\} \mid Q); C}\ \textsc{TDef}$$

– **case RRes**
We have $P = (\nu\, s : \Gamma')Q \longrightarrow (\nu\, s)R = P'$, with $Q \longrightarrow R$ (from the rule premise).

$$\dfrac{\overset{\textsc{TRes}}{\Delta; \Gamma, \Gamma' \vdash Q; C \otimes C_1} \qquad (\Gamma' = \{\mathtt{s}[\mathtt{p}] : \mathtt{tr}\,(\rho_p)\,, \rho_p : \{\rho_p \mapsto S_p\}\}_{p\in I}, C_1 = \otimes_{p\in I}\{\rho_p \mapsto S_p\})}{\Delta; \Gamma \vdash (\nu\, s : \Gamma')Q; C}$$

By the induction hypothesis, $\Delta; \Gamma, \Gamma' \vdash R; C_2$ and $(\Gamma, \Gamma', C \otimes C_1) \longrightarrow^* (\Gamma, \Gamma', C_2)$.
By Prop. 1 we know that $\mathrm{dom}\,(C_2) = \mathrm{dom}\,(C \otimes C_1) = \mathrm{dom}\,(C \otimes_{p\in I}\{\rho_p \mapsto S_p\})$.
So we can rewrite C_2 as $C' \otimes C'_1$, where $\mathrm{dom}\,(C') = \mathrm{dom}\,(C)$, $\mathrm{dom}\,(C'_1) = \mathrm{dom}\,(C_1)$, and $C'_1 = \otimes_{p\in I}\{\rho_p \mapsto S'_p\}$.
The reduction $(\Gamma, \Gamma', C \otimes C_1) \longrightarrow^* (\Gamma, \Gamma', C_2)$ then becomes: $(\Gamma, \Gamma', C \otimes C_1) \longrightarrow^* (\Gamma, \Gamma', C' \otimes C'_1)$.
By Def. 12 $(\Gamma, \Gamma', C' \otimes C'_1)$ is consistent, and therefore by Corollary 1 (Γ', C') is consistent. Hence:

$$\frac{\Delta; \Gamma, \Gamma' \vdash R; C' \otimes C_1' \qquad (\Gamma' = \{\mathtt{s}[\mathtt{p}] : \mathtt{tr}\,(\rho_p)\,, \rho_p : \{\rho_p \mapsto S_p\}\}_{p \in I}, C_1' = \otimes_{p \in I}\{\rho_p \mapsto S_p'\})}{\Delta; \Gamma \vdash (\nu\, s : \Gamma')R; C'}\ \textsc{TRes}$$

– **case RPar**
We have $P = Q \mid R \longrightarrow Q' \mid R = P'$, with $Q \longrightarrow Q'$ from the rule premise.

$$\textsc{TPar}\ \frac{\Delta; \Gamma \vdash Q; C_1 \qquad \Delta; \Gamma \vdash R; C_2}{\Delta; \Gamma \vdash Q \mid R; C = C_1 \otimes C_2}$$

By the induction hypothesis $\Delta; \Gamma \vdash Q'; C_3$, and $(\Gamma, C_1) \longrightarrow^* (\Gamma, C_3)$.

$$\textsc{TPar}\ \frac{\Delta; \Gamma \vdash Q'; C_3 \qquad \Delta; \Gamma \vdash R; C_2}{\Delta; \Gamma \vdash Q' \mid R; C_3 \otimes C_2 = C'}$$

By Corollary 1 $(\Gamma, C_1 \otimes C_2) \longrightarrow^* (\Gamma, C_3 \otimes C_2)$, which is to say $(\Gamma, C) \longrightarrow^* (\Gamma, C')$ as required.

– **case RDef** We have $P = \mathsf{def}\, X\langle \tilde{x} : \tilde{U}\rangle = Q_X \,\mathsf{in}\, Q \longrightarrow \mathsf{def}\, X\langle \tilde{x} : \tilde{U}\rangle = Q_X \,\mathsf{in}\, Q' = P'$ from the rule premise.

$$\frac{\Delta, X : \tilde{U}; \tilde{x} : \tilde{U} \vdash Q_X; \tilde{C} \qquad \Delta, X : \tilde{U}; \Gamma \vdash Q; C}{\Delta; \Gamma \vdash \mathsf{def}\, X\langle \tilde{x} : \tilde{U}\rangle = Q_X \,\mathsf{in}\, Q; C}\ \textsc{TDef}$$

By the induction hypothesis, $\Delta, X : \tilde{U}; \Gamma \vdash Q'; C'$ and $(\Gamma, C) \longrightarrow^* (\Gamma, C')$, and we conclude by:

$$\frac{\Delta, X : \tilde{U}; \tilde{x} : \tilde{U} \vdash Q_X; \tilde{C} \qquad \Delta, X : \tilde{U}; \Gamma \vdash Q'; C'}{\Delta; \Gamma \vdash \mathsf{def}\, X\langle \tilde{x} : \tilde{U}\rangle = Q_X \,\mathsf{in}\, Q'; C'}\ \textsc{TDef}$$

– **case RCong**
We have $P \equiv Q$, $P' \equiv Q'$, and $Q \longrightarrow Q'$ from the rule premise. By Prop. 2 $\Delta; \Gamma \vdash Q; C$. By the induction hypothesis $\Delta; \Gamma \vdash Q'; C'$ and $(\Gamma, C) \longrightarrow^* (\Gamma, C')$. By Prop. 2 we conclude $\Delta; \Gamma \vdash P'; C'$.