A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming

Alceste Scalas[1], Ornela Dardha[2], Raymond Hu[1], and Nobuko Yoshida[1]

1 **Imperial College London, UK**
{alceste.scalas, raymond.hu, n.yoshida}@imperial.ac.uk
2 **University of Glasgow, UK**
ornela.dardha@glasgow.ac.uk

Abstract

Multiparty Session Types (MPST) is a typing discipline for message-passing distributed processes that can ensure properties such as absence of communication errors and deadlocks, and protocol conformance. Can MPST provide a theoretical foundation for concurrent and distributed programming in "mainstream" languages?

We address this problem by *(1)* developing the first encoding of a *full-fledged* multiparty session π-calculus into standard linear π-calculus, and *(2)* using the encoding as the foundation of a practical toolchain for safe multiparty programming in Scala.

Our encoding is type-preserving and operationally sound and complete. Importantly for distributed applications, it preserves the *choreographic* nature of MPST and illuminates that multiparty sessions (and their safety properties) can be precisely represented with a decomposition into *binary linear channels*. Previous works have only studied the relation between (limited) multiparty sessions and binary sessions by *orchestration* means.

We exploit these results to implement an automated generation of Scala APIs for multiparty sessions. These APIs act as a layer on top of existing libraries for binary communication channels: this allows distributed multiparty systems to be safely implemented over binary transports, as commonly found in practice. Our implementation is also the first to support *distributed multiparty delegation*: our encoding yields it for free, via existing mechanisms for binary delegation.

1 Introduction

Correct design and implementation of concurrent and distributed applications is notoriously difficult. Programmers have to deal with many challenges, pertaining to both *protocol conformance* (do the messages being sent/received respect a given specification?) and the *communication mechanics* (how are the interactions actually performed?). These difficulties are exacerbated by the potential complexity of interactions between *multiple* participants, and in settings where the *communication topology* is not fixed.

As an example, consider a common scenario for a peer-to-peer multiplayer game: the clients, initially unknown to each other, first connect to a "matchmaking" server, whose task is to group players and setup a game session in which they can interact directly. Fig. 1 depicts this scenario: Q is the server, expected to set the game for

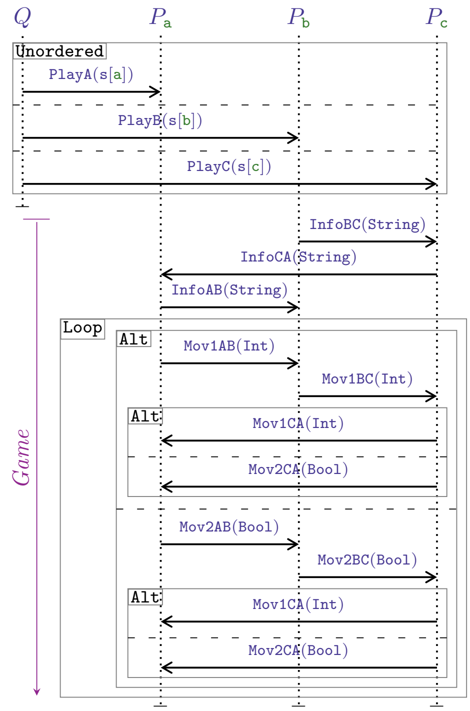


Figure 1 Game server with 3 clients.

three players, P_{a}, P_{b} and P_{c}. To set up a game, the server sends to each client some networking information (denoted by the $s[\mathsf{a}]$, $s[\mathsf{b}]$, $s[\mathsf{c}]$ payloads of the `PlayA`/`B`/`C` messages) needed to "introduce" the clients to each other and allow them to communicate. The clients then proceed according to the main game protocol (annotated as "*Game*"): it consists of some initial message exchanges (`Info`), and a main game loop, where P_{a} selects between two possible messages to send to P_{b} (`Mob1AB` or `Mov2AB`) followed by a message from P_{b} to P_{c}, who can choose which message send back to P_{a}.

As Fig. 1 illustrates, this applications can involve richly structured protocols, with non-trivial message dependencies between multiple roles, and a changing communication topology (initially client-to-server, eventually becoming client-to-client). Turning such a high level specification into an actual implementation is not straightforward—programmers would greatly benefit from tools and programming aids to *statically* assist the detection of protocol violations in source code, and correctly implement the communication topology dynamics.

Multiparty Session Types (MPST) [26] is a theoretical framework allowing for the precise modelling of such applications. In MPST, participants are abstracted as *roles* (e.g., game clients a, b, c) and implemented as *session π-calculus processes*, that model server/client programs. In the MPST framework, the "networking information payloads" $s[\mathsf{a}]$, $s[\mathsf{b}]$, $s[\mathsf{c}]$ can be naturally modelled as *multiparty channel endpoints* for the game *session* s. Notably, channel endpoints can *themselves* be sent/received: formally, this allows to *delegate* part of a multiparty interaction to another process, resulting in a change of the communicating topology. In our example, the server Q delegates (i.e., sends) the channel endpoint $s[\mathsf{b}]$ to P_{b}; the latter can then use $s[\mathsf{b}]$ to interact with the two processes that own the endpoints $s[\mathsf{a}]$ and $s[\mathsf{c}]$ (i.e., P_{a} and P_{c} after the other two delegations).

The MPST framework ensures safe interaction via *session types*: they formalise *protocols*, as structured sequences of inputs, outputs and choices. The session typing system assigns such types to channel endpoints, and *type-checks the processes* that use them. In our example, the channel endpoint $s[\mathsf{b}]$ could be typed as:

$$S_{\mathsf{b}} \;=\; \mathsf{c}!\mathtt{InfoBC}(\mathtt{String}) \,.\, \mathsf{a}?\mathtt{InfoAB}(\mathtt{String}) \,.\, \mu\mathbf{t}.\big(\mathsf{a}\;\&\;\{\,?\mathtt{Mov1AB}(\mathtt{Int}).\mathsf{c}!\mathtt{Mov1BC}(\mathtt{Int}).\mathbf{t}\;,\;?\mathtt{Mov2AB}(\mathtt{Bool}).\mathsf{c}!\mathtt{Mov2BC}(\mathtt{Bool}).\mathbf{t}\;\}\big) \tag{1}$$

S_{b} says that $s[\mathsf{b}]$ must be used to realise the *Game* interactions of P_{b} in Fig. 1: first to send `InfoBC(String)` to c, then receive `InfoAB` from a, then enter the recursive game "loop" $\mu\mathbf{t}.(\ldots)$. Inside the recursion, $\mathsf{a}\,\&\,\{\ldots\}$ is a *branching from* a: depending on a's choice, the channel will deliver either `Mov1AB(Int)` (in which case, it must be used to send `Mov1BC(Int)` to c, and loop), or `Mov2AB` (then, it must be used to send `Mov2BC` to c, and loop). Analogous types can be assigned to $s[\mathsf{a}]$ and $s[\mathsf{c}]$. The delegation actions are represented by session types like $\mathsf{q}?\mathtt{PlayB}(S_{\mathsf{b}}).\mathbf{end}$, which means: from role q, receive a message `PlayB` carrying a *channel endpoint* that must be used according to S_{b} above; then, **end** the session. Session type checking ensures that, e.g., the process P_{b} uses its channels as prescribed by the types above—thus safely implementing the expected channel dynamics and fulfilling the role of b in the game.

Finally, the MPST framework allows to formalise, e.g., the *whole Game* protocol in Fig. 1 as a *global type*, and validate that it is *deadlock-free*; then, via typing, check whether an ensemble of processes interacts according to the global type (and is, thus, deadlock-free).

MPSTs in practice: challenges The above suggests that MPSTs offer a promising formal foundation for *safe distributed programming*, helping to develop concurrent applications whose interactions are type-safe and deadlock-free. However, bridging the gap between the abstract theory and a concrete implementation raises several challenges:

C1. Multiparty session types allow 2, 3 *or more* roles to interact—but in practice, communication occurs over *binary* channels (e.g., TCP sockets). Can multiparty channels be implemented as compositions of binary channels, while preserving their *safety* properties?

C2. Multiparty session types are far from the types found in "mainstream" programming languages, as demonstrated by S_{b} in (1). Can they be represented, e.g., as objects? If so, what is their programming interface? And what are the API internals?

C3. How should *multiparty delegation* be realised, especially in *distributed* settings?

Unfortunately, the current state-of-the-art in session types has not addressed these challenges. On one hand, existing theoretical works on encoding multiparty sessions into binary sessions [7, 8] rely on a workaround by introducing centralised *medium* (or *arbiter*) processes to *orchestrate* the interactions between the multiparty session roles: hence, they depart from the choreographic (i.e., decentralised) nature of the MPST framework [26], and preclude examples such as our peer-to-peer game in Fig. 1. On the other hand, there are *no* existing implementations of full-fledged MPST; e.g., [52, 30, 31, 40, 48, 56, 51] only support *binary* sessions, while none of [27, 58, 16, 19] support session delegation.

Our approach In this work, we tackle the three challenges above with a two-step strategy:

S1. we develop the first *choreographic* encoding of a *"full-fledged" multiparty session π-calculus* into *standard linear π-calculus*;

S2. we implement a *multiparty session API generation* for Scala, based on our encoding.

By step **S1**, we formally address challenge **C1**. Linear π-calculus provides a theoretical framework with channels and types that cater only for *binary* communication, and each channel may only be used *once* for input/output. These "limitations" are key to the practicality of our approach. In fact, they force us to figure out whether *multiparty channels can be represented by a decomposition into binary linear channels*—and whether *multiparty session types can be represented by a decomposition into linear types*. The practical payoff is that linear π-calculus channels/types are amenable for an (almost) direct object-based representation, as demonstrated in [56]: this tackles challenge **C2**. Moreover, linear π-calculus allows to *prove* whether such a decomposition is "correct"—i.e., whether it *preserves type safety*, and whether MPST processes can be encoded so that they only interact on *binary* channels, while *preserving their original behaviour* (thus "inheriting" deadlock-freedom).

In step **S2**, we generate high-level typed APIs for multiparty session programming, ensuring their "correctness" by reflecting the types and process behaviours formalised in step **S1**. Following the binary decomposition in step **S1**, we can implement such APIs as a layer over *existing* libraries for binary sessions (available for Java [28], Haskell [52, 30, 40], Links [42], Rust [31], Scala [56], ML [51]), in a way that solves challenge **C3** "for free".

Contributions We present the *first* encoding (§ 5) of a full multiparty session π-calculus (§ 2) into standard π-calculus with linear, labelled tuple and variant types (§ 3).

- We present a novel, streamlined formulation of MPSTs that clearly separates the global/local typing levels. This allows us to "close the gaps" between the intricacies of the MPST theory and the (much simpler) π-calculus, while staying faithful to standard MPST literature. Via our MPST formulation, we also spot a longstanding issue with *type merging* [17] (Def. 2.11; § 2.1 "On Consistency") and fix it, obtaining a *revised subject reduction* for MPSTs (Theorem 2.16).

- At the heart of our encoding there is the discovery that the *type safety* property of MPST is *precisely* characterised as a *decomposition* into linear π-calculus types (Theorem 6.4). Our encoding of *types* preserves *duality* and *subtyping* (Theorems 6.1 and 6.2); our encoding of *processes* is *type-preserving* and *operationally sound and complete* (Theorems 6.3 and 6.6).
- We subsume the encodings of *binary* sessions into π-calculus [13, 14], and support *recursion* (§ 4), which was not properly handled in [12]. Further, we show that multiparty sessions can be encoded into binary sessions *choreographically*, i.e., while *preserving process distribution* (homomorphically w.r.t. parallel composition), in contrast to [7, 8].

In § 7, we use our encoding as formal basis for the *first implementation of multiparty sessions* supporting *distributed multiparty delegation*, over existing Scala libraries. Our implementation is available (as Open Source software) in [55].

Conventions

In derivations, we use a *single*/*double* line for *inductive*/*coinductive* rules. Recursive types $\mu\mathbf{t}.T$ are *guarded*, i.e., $\mathbf{t}$ can only appear in T under a type constructor different from μ. As usual, we define $\mathrm{unf}(\mu\mathbf{t}.T) = \mathrm{unf}(T\{^{\mu\mathbf{t}.T}/_{\mathbf{t}}\})$, and $\mathrm{unf}(T) = T$ when $T \neq \mu\mathbf{t}.T'$. We adopt *syntactic* type equality, and thus distinguish a recursive type from its unfolding. Types are always *closed*. We write $P \to P'$ for process reductions, $\to^*$ for the reflexive+transitive closure of $\to$, and $P \not\to$ iff $\nexists P'$ such that $P \to P'$. We assume a *basic subtyping* $\leqslant_{\mathsf{B}}$ capturing e.g. $\mathtt{Int} \leqslant_{\mathsf{B}} \mathtt{Real}$. For readability, we use **blue**/**red** for **multiparty**/**standard** π-calculus.

2 Multiparty Session π-Calculus

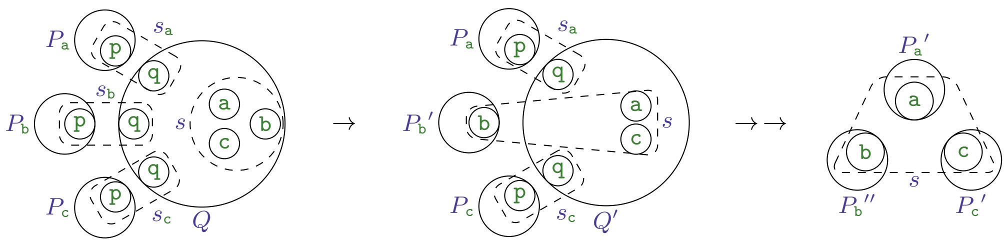
In this section we illustrate a multiparty session π-calculus [26] complete with recursion, subtyping [18] and type merging [61, 17]. We adopt a notation based on [10].

▶ **Definition 2.1.** *The* syntax *of multiparty session π-calculus* processes *and* values *is:*

Processes	P, Q	$::=$	$\mathbf{0} \mid P \mid Q \mid (\boldsymbol{\nu} s)P$	*(inaction, composition, restriction)*
			$c[\mathtt{p}] \oplus \langle l(v) \rangle.P$	*(selection towards role* p*)*
			$c[\mathtt{p}] \,\&_{i \in I} \{l_i(x_i).P_i\}$	*(branching from role* p *— with $I \neq \emptyset$)*
			$\mathbf{def}\ D\ \mathbf{in}\ Q \mid X\langle\tilde{x}\rangle$	*(process definition, process call)*
Declarations	D	$::=$	$X(\tilde{x}) = P$	*(process declaration)*
Channels	c	$::=$	$x \mid s[\mathtt{p}]$	*(variable, channel with role* p*)*
Values	v	$::=$	$c \mid \mathtt{false} \mid \mathtt{true} \mid 42 \mid \ldots$	*(channel, base value)*

$\mathrm{fc}(P)$ *is the set of* free channels with roles *in P, and* $\mathrm{fv}(P)$ *is the set of* free variables *in P.*

The **inaction** $\mathbf{0}$ represents a terminated process. The **parallel composition** $P \mid Q$ represents two processes that can execute concurrently (and possibly communicate). The **session restriction** $(\boldsymbol{\nu} s)P$ delimits the scope of a session s to P. Process $c[\mathtt{p}] \oplus \langle l(v) \rangle.P$ performs a **selection (internal choice)** on the channel c towards role $\mathtt{p}$: the labelled value $l(v)$ is sent, and the execution continues as process P. Dually, process $c[\mathtt{p}] \,\&_{i \in I} \{l_i(x_i).P_i\}$ waits for a **branching (external choice)** on the channel c from role $\mathtt{p}$. If the labelled value $l_k(v)$ is received (with $k \in I$), then the execution continues as P_k (with x_k holding value v). Note that for all $i \in I$, variable x_i is bound with scope P_i. In both branching and selection, the labels l_i $(i \in I)$ are all different and their order is irrelevant. **Process definition** $\mathbf{def}\ D\ \mathbf{in}\ Q$ and **process call** $X\langle\tilde{x}\rangle$ model recursion, with D being a **process declaration**: the call invokes X by replacing its formal parameters with the actual ones. We postulate that process declarations are *closed*, i.e., in $X(\tilde{x}) = P$, we have $\mathrm{fv}(P) \subseteq \tilde{x}$ and $\mathrm{fc}(P) = \emptyset$. A **channel**



(a) **(b)** **(c)**

Figure 2 Multiparty peer-to-peer game. Dashed lines represent session scopes, and circled roles represent channels with roles. *(a)* initial configuration; *(b)* delegation of channel with role $s[\mathtt{b}]$ (and end of session $s_{\mathtt{b}}$); *(c)* clients directly interacting on session s, after "complete" delegation.

c can be either a variable or a **channel with role** $s[\mathtt{p}]$, i.e., a multiparty communication endpoint whose user impersonates role $\mathtt{p}$ in the session s. **Values** v can be variables, or channels with roles, or base values. Note that our syntax is simplified in the style of [18]: it does not have dedicated input/output prefixes, but they can be easily encoded using $\&$ (with *one* branch) and $\oplus$.

▶ **Example 2.2.** *The following MPST π-calculus process implements the scenario in Fig. 1:*

$$
\begin{array}{l}
\textbf{def } \mathit{Loop}_{\mathtt{b}}(x) = x[\mathtt{a}] \,\&\, \big\{\mathtt{Mov1AB}(y).x[\mathtt{c}] \oplus \langle \mathtt{Mov1BC}(y)\rangle.\mathit{Loop}_{\mathtt{b}}\langle x\rangle \;,\; \mathtt{Mov2AB}(z).x[\mathtt{c}] \oplus \langle \mathtt{Mov2BC}(z)\rangle.\mathit{Loop}_{\mathtt{b}}\langle x\rangle\big\} \textbf{ in}\\
\quad \textbf{def } \mathit{Client}_{\mathtt{b}}(y) = y[\mathtt{q}] \,\&\, \mathtt{PlayB}(z) \,.\, z[\mathtt{c}] \oplus \langle \mathtt{InfoBC}(\text{“...”})\rangle \,.\, z[\mathtt{a}] \,\&\, \mathtt{InfoBA}(y) \,.\, \mathit{Loop}_{\mathtt{b}}\langle z\rangle \textbf{ in}\\
\qquad (\boldsymbol{\nu} s_{\mathtt{a}}, s_{\mathtt{b}}, s_{\mathtt{c}})\big(Q \mid P_{\mathtt{a}} \mid P_{\mathtt{b}} \mid P_{\mathtt{c}}\big)
\end{array}
$$

$$
\begin{array}{lrcl}
\textit{where:} & P_{\mathtt{b}} & = & \mathit{Client}_{\mathtt{b}}\langle s_{\mathtt{b}}[\mathtt{p}]\rangle \quad \textit{(for brevity, we omit the definitions of } P_{\mathtt{a}} \textit{ and } P_{\mathtt{c}}\textit{)}\\
 & Q & = & (\boldsymbol{\nu} s)\Big(s_{\mathtt{a}}[\mathtt{q}][\mathtt{p}] \oplus \langle \mathtt{PlayA}(s[\mathtt{a}])\rangle \mid s_{\mathtt{b}}[\mathtt{q}][\mathtt{p}] \oplus \langle \mathtt{PlayB}(s[\mathtt{b}])\rangle \mid s_{\mathtt{c}}[\mathtt{q}][\mathtt{p}] \oplus \langle \mathtt{PlayC}(s[\mathtt{c}])\rangle\Big)
\end{array}
$$

In the 3rd line, $s_{\mathtt{a}}, s_{\mathtt{b}}, s_{\mathtt{c}}$ are the sessions between the server process Q and the clients $P_{\mathtt{a}}, P_{\mathtt{b}}, P_{\mathtt{c}}$, which are composed in parallel. Each sessions has 2 roles: $\mathtt{q}$ (server) and $\mathtt{p}$ (client); e.g., $s_{\mathtt{b}}$ is accessed by the server (through the channel with role $s_{\mathtt{b}}[\mathtt{q}]$) and by the client $P_{\mathtt{b}}$ (through $s_{\mathtt{b}}[\mathtt{p}]$); similarly, $s_{\mathtt{a}}$ (resp. $s_{\mathtt{c}}$) is accessed by $P_{\mathtt{a}}$ (resp. $P_{\mathtt{c}}$) through $s_{\mathtt{a}}[\mathtt{p}]$ (resp. $s_{\mathtt{c}}[\mathtt{p}]$), while the server owns $s_{\mathtt{a}}[\mathtt{q}]$ (resp. $s_{\mathtt{c}}[\mathtt{q}]$). In the body of Q, the server declares a session s (with 3 roles $\mathtt{a}, \mathtt{b}, \mathtt{c}$) for playing the game. Note that the scope of s does not *include $P_{\mathtt{a}}, P_{\mathtt{b}}, P_{\mathtt{c}}$: see Fig. 2(a) for a schema of processes and sessions.*

The server Q uses the channel with role $s_{\mathtt{b}}[\mathtt{q}]$ (resp. $s_{\mathtt{a}}[\mathtt{q}], s_{\mathtt{c}}[\mathtt{q}]$) to concurrently send the message $\mathtt{PlayB}$ (resp. $\mathtt{PlayA}, \mathtt{PlayC}$) and the channel with role $s[\mathtt{b}]$ (resp. $s[\mathtt{a}], s[\mathtt{c}]$) to $\mathtt{p}$: i.e., the server performs a delegation *to the client process $P_{\mathtt{b}}$ (resp. $P_{\mathtt{a}}, P_{\mathtt{c}}$). This way, the client obtains a channel endpoint to interact in the game session s, interpreting role $\mathtt{b}$ (resp. $\mathtt{a}, \mathtt{c}$).*

The client $P_{\mathtt{b}}$ is implemented by invoking $\mathit{Client}_{\mathtt{b}}\langle s_{\mathtt{b}}[\mathtt{p}]\rangle$ (defined in the 2nd line). Here, $y[\mathtt{q}] \,\&\, \mathtt{PlayB}(z)$ means that y (that becomes $s_{\mathtt{b}}[\mathtt{p}]$ after the invocation) is used to receive $\mathtt{PlayB}(z)$ from $\mathtt{q}$, while $z[\mathtt{c}] \oplus \langle \mathtt{InfoBC}(\text{“...”})\rangle$ means that z (that becomes $s[\mathtt{b}]$ after the delegation is received) is used to send $\mathtt{InfoBC}(\text{“...”})$ to $\mathtt{c}$. The game loop is implemented with the recursive process call $\mathit{Loop}_{\mathtt{b}}\langle z\rangle$ (defined in the 1st line) — which becomes $\mathit{Loop}_{\mathtt{b}}\langle s[\mathtt{b}]\rangle$ after delegation.

▶ **Definition 2.3.** *The* operational semantics *of multiparty session processes is:*

$$
\begin{array}{rl}
(\textsc{R-Comm}) & s[\mathtt{p}][\mathtt{q}]\,\&_{i\in I}\,\{l_i(x_i).P_i\} \mid s[\mathtt{q}][\mathtt{p}] \oplus \langle l_j(v)\rangle.Q \;\;\to\;\; P_j\{^v/_{x_j}\} \mid Q \quad (\textit{if } j \in I \textit{ and } \mathrm{fv}(v) = \emptyset)\\
(\textsc{R-Call}) & \mathbf{def}\; X(\widetilde{x}) = P \;\mathbf{in}\; (X\langle\widetilde{v}\rangle \mid Q) \;\to\; \mathbf{def}\; X(\widetilde{x}) = P \;\mathbf{in}\; (P\{\widetilde{v}/\widetilde{x}\} \mid Q)\\
& \qquad\qquad\qquad (\textit{if } \widetilde{x} = x_1,\ldots,x_n,\ \widetilde{v} = v_1,\ldots,v_n,\ \mathrm{fv}(\widetilde{v}) = \emptyset)\\
(\textsc{R-Par}) & P \to Q \;\textit{ implies }\; P \mid R \to Q \mid R \qquad (\textsc{R-Res}) \quad P \to Q \;\textit{ implies }\; (\boldsymbol{\nu} s)P \to (\boldsymbol{\nu} s)Q\\
(\textsc{R-Def}) & P \to Q \;\textit{ implies }\; \mathbf{def}\; D \;\mathbf{in}\; P \to \mathbf{def}\; D \;\mathbf{in}\; Q\\
(\textsc{R-Struct}) & P \equiv P' \textit{ and } P \to Q \textit{ and } Q' \equiv Q \textit{ implies } P' \to Q' \quad (\textit{with} \equiv \textit{standard — see § A})
\end{array}
$$

(R-Comm) says that the parallel composition of a branching and a selection process, operating on the same session s respectively as roles $\mathtt{p}$ and $\mathtt{q}$ (i.e., via $s[\mathtt{p}]$ and $s[\mathtt{q}]$) and targeting each other (i.e., $s[\mathtt{p}]$ is used to branch from $\mathtt{q}$, and $s[\mathtt{q}]$ is used to select towards $\mathtt{p}$) reduces to the corresponding continuations, with a value substitution on the receiver side. (R-Call) says that a process call $X\langle\widetilde{v}\rangle$ in the scope of $\mathbf{def}\; X(\widetilde{x}) = P \;\mathbf{in}$ reduces by replacing $X\langle\widetilde{v}\rangle$ with P, and replacing the formal parameters ($\widetilde{x}$) with the actual ones ($\widetilde{v}$). The rest of the rules are standard: reduction can happen under parallel composition, restriction and process definition, and the reduction relation is closed under structural congruence.

▶ **Example 2.4.** *The process in Ex. 2.2 reduces as (see also Fig. 2(b), noting the scope of s):*

$$(\boldsymbol{\nu} s_{\mathtt{a}}, s_{\mathtt{b}}, s_{\mathtt{c}})\big(Q \mid P_{\mathtt{a}} \mid P_{\mathtt{b}} \mid P_{\mathtt{c}}\big) \to$$

$$(\boldsymbol{\nu} s_{\mathtt{a}}, s_{\mathtt{c}})\Big((\boldsymbol{\nu} s)\Big(\big(s_{\mathtt{a}}[\mathtt{q}][\mathtt{p}] \oplus \langle \mathtt{PlayA}(s[\mathtt{a}])\rangle \mid s_{\mathtt{c}}[\mathtt{q}][\mathtt{p}] \oplus \langle \mathtt{PlayC}(s[\mathtt{c}])\rangle\big) \mid s[\mathtt{b}][\mathtt{c}] \oplus \langle \mathtt{InfoBC}(\text{“...”})\rangle \ldots\Big) \mid P_{\mathtt{a}} \mid P_{\mathtt{c}}\Big)$$

2.1 Multiparty Session Typing

We now illustrate the typing system for the MPST π-calculus, and its properties. We adopt standard definitions from literature—except for some crucial (and duly noted) adaptations.

The MPST framework fosters a *top-down* approach where a *global type* G describes a protocol involving various *roles* — such as the game with roles $\mathtt{a}, \mathtt{b}, \mathtt{c}$ in § 1. G is *projected* into a set of *local types* $S_{\mathtt{a}}$, $S_{\mathtt{b}}$, $S_{\mathtt{c}}, \ldots$ (one per role) that specify how each role is expected to use its channel endpoint. Local types, in turn, are assigned to communication channels, and type-check the processes using them. Session typing ensures that processes *(1) never go wrong* (i.e., use their channels in a type-safe way), and *(2)* interact obeying the protocol in G, by respecting its local projections — thus realising a *multiparty, deadlock-free session.*

In the following, we provide a revised and streamlined presentation that clearly outlines the *interplay between the global/local typing levels.* For this reason, unlike most papers, we discuss *local types first*, and *global types later*, at the end of the section.

Types: Local and Partial Multiparty session types describe the expected usage of a channel, as a communication protocol involving two or more *roles.* They allow to declare structured sequences of input/output actions, specifying who is the source/target role of interaction.

▶ **Definition 2.5** (Types and roles). *The syntax of* (local) session types *is:*

$$
\begin{array}{rcll}
S & ::= & \mathtt{p}\,\&_{i\in I}\,?l_i(U_i).S_i & (\textit{branching from role } \mathtt{p} \textit{ — with } I \neq \emptyset)\\
& & \mathtt{p} \oplus_{i\in I} !l_i(U_i).S_i & (\textit{selection towards role } \mathtt{p} \textit{ — with } I \neq \emptyset)\\
& & \mu\mathbf{t}.S \mid \mathbf{t} \mid \mathbf{end} & (\textit{recursive type, type variable, termination})
\end{array}
$$

$$B ::= \mathtt{Bool} \mid \mathtt{Int} \mid \ldots \quad (\textit{base type}) \qquad U ::= B \mid S\ (\textit{closed}) \quad (\textit{payload type})$$

We omit $\&/\oplus$ when I is a singleton: $\mathtt{p}!l_1(\mathtt{Int}).S_1$ stands for $\mathtt{p} \oplus_{i\in\{1\}} !l_i(\mathtt{Int}).S_i$. The set of roles in S*, denoted as* $\mathrm{roles}(S)$*, is defined as follows:*

$$\mathrm{roles}(\mathtt{p} \oplus_{i\in I} !l_i(U_i).S_i) \triangleq \mathrm{roles}(\mathtt{p}\,\&_{i\in I}\,?l_i(U_i).S_i) \triangleq \{\mathtt{p}\} \cup \textstyle\bigcup_{i\in I} \mathrm{roles}(S_i)$$

$$\mathrm{roles}(\mathbf{end}) \triangleq \emptyset \qquad \mathrm{roles}(\mathbf{t}) \triangleq \emptyset \qquad \mathrm{roles}(\mu\mathbf{t}.S) \triangleq \mathrm{roles}(S)$$

We will write $\mathtt{p} \in S$ *for* $\mathtt{p} \in \mathrm{roles}(S)$*, and* $\mathtt{p} \in S \backslash \mathtt{q}$ *for* $\mathtt{p} \in \mathrm{roles}(S) \setminus \{\mathtt{q}\}$*.*

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** $\mathtt{p} \oplus_{i\in I} !l_i(U_i).S_i$, 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. The labels of branch/select types are all distinct and their order is irrelevant. The **recursive type** $\mu\mathbf{t}.S$ and **type variable** $\mathbf{t}$ model infinite behaviours. $\mathbf{end}$ is the type of a **terminated channel** (often omitted). **Base types** $B, B', \ldots$ can be types like `Bool`, `Int`, *etc.* **Payload types** $U, U', \ldots$ are either base types, or *closed* session types.

▶ **Example 2.6.** *See the definition and description of session type* $S_{\mathtt{b}}$ *in § 1 (equation* (1)*).*

To define session typing contexts later on, we also need *partial* session types.

▶ **Definition 2.7.** Partial session types*, denoted by* H*, are:*

$$\begin{array}{rcll} H & ::= & \&_{i\in I}\,?l_i(U_i).H_i \;\mid\; \oplus_{i\in I}\,!l_i(U_i).H_i & \textit{(branching, selection)}\ \textit{(with } I \neq \emptyset,\ U_i \textit{ closed)} \\ & & \mu\mathbf{t}.H \;\mid\; \mathbf{t} \;\mid\; \mathbf{end} & \textit{(recursive type, type variable, termination)} \end{array}$$

A partial session type H is either a branching, a selection, a recursion, a type variable, or a terminated channel type. Unlike Def. 2.5, partial types have *no role annotations*: they are similar to *binary* session types (but the payloads U_i can be *multiparty*)—and similarly, they endow a notion of *duality*: the outputs of a type match the inputs of its dual, and *vice versa*.

▶ **Definition 2.8.** $\overline{H}$ *is the* dual of H*, defined as:*

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

The dual of a select type is a branch type with dualised continuations, *and vice versa*. The payloads U_i are the same. Duality is the identity on $\mathbf{end}$ and on a type variable $\mathbf{t}$, and it is homomorphic on a recursive partial session type $\mu\mathbf{t}.H$.

Multiparty session types can be *projected* onto a role $\mathtt{q}$ (Def. 2.9 below): this yields a partial type that only describes the communications where $\mathtt{q}$ is involved. This is technically necessary for typing rules, as we will see in Def. 2.11 later on.

▶ **Definition 2.9.** $S \upharpoonright \mathtt{q}$ *is the* partial projection of S onto $\mathtt{q}$*:*

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

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

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

where $\bigsqcap$ *is the* merge operator for partial session types*:*

$$\mathbf{end} \sqcap \mathbf{end} \triangleq \mathbf{end} \qquad \mathbf{t} \sqcap \mathbf{t} \triangleq \mathbf{t} \qquad \mu\mathbf{t}.H \sqcap \mu\mathbf{t}.H' \triangleq \mu\mathbf{t}.(H \sqcap H')$$
$$\&_{i\in I}\,?l_i(U_i).H_i \;\sqcap\; \&_{i\in I}\,?l_i(U_i).H'_i \;\triangleq\; \&_{i\in I}\,?l_i(U_i).(H_i \sqcap H'_i)$$
$$\oplus_{i\in I}\,!l_i(U_i).H_i \;\sqcap\; \oplus_{j\in J}\,!l_j(U_j).H'_j \;\triangleq$$
$$\left(\oplus_{k\in I\cap J}\,!l_k(U_k).(H_k \sqcap H'_k)\right) \oplus \left(\oplus_{i\in I\setminus J}\,!l_i(U_i).H_i\right) \oplus \left(\oplus_{j\in J\setminus I}\,!l_j(U_j).H'_j\right)$$

The projection of $\mathbf{end}$ or a type variable $\mathbf{t}$ onto any role is the identity. Projecting a recursive type $\mu\mathbf{t}.S$ onto $\mathtt{q}$, means projecting S onto $\mathtt{q}$, if $S\upharpoonright\mathtt{q}$ is *not* some $\mathbf{t}'$; otherwise, the projection is

$$\dfrac{\forall i \in I \quad U_i \leqslant_{\mathsf{S}} U_i' \quad S_i \leqslant_{\mathsf{S}} S_i'}{\mathtt{p} \,\&_{i \in I}\, ?l_i(U_i).S_i \;\leqslant_{\mathsf{S}}\; \mathtt{p} \,\&_{i \in I \cup J}\, ?l_i(U_i').S_i'} \;(\text{S-Brch}) \qquad \dfrac{\forall i \in I \quad U_i' \leqslant_{\mathsf{S}} U_i \quad S_i \leqslant_{\mathsf{S}} S_i'}{\mathtt{p} \oplus_{i \in I \cup J} !l_i(U_i).S_i \leqslant_{\mathsf{S}} \mathtt{p} \oplus_{i \in I} !l_i(U_i').S_i'} \;(\text{S-Sel})$$

$$\dfrac{B \leqslant_{\mathsf{B}} B'}{B \leqslant_{\mathsf{S}} B'} \;(\text{S-B}) \qquad \dfrac{}{\mathbf{end} \leqslant_{\mathsf{S}} \mathbf{end}} \;(\text{S-End}) \qquad \dfrac{S\{\mu\mathbf{t}.S/\mathbf{t}\} \leqslant_{\mathsf{S}} S'}{\mu\mathbf{t}.S \leqslant_{\mathsf{S}} S'} \;(\text{S-}\mu\text{L}) \qquad \dfrac{S \leqslant_{\mathsf{S}} S'\{\mu\mathbf{t}.S'/\mathbf{t}\}}{S \leqslant_{\mathsf{S}} \mu\mathbf{t}.S'} \;(\text{S-}\mu\text{R})$$

$$\dfrac{\forall i \in I \quad U_i \leqslant_{\mathsf{S}} U_i' \quad H_i \leqslant_{\mathsf{P}} H_i'}{\&_{i \in I}\, ?l_i(U_i).H_i \leqslant_{\mathsf{P}} \&_{i \in I \cup J}\, ?l_i(U_i').H_i'} \;(\text{S-ParBrch}) \qquad \dfrac{\forall i \in I \quad U_i' \leqslant_{\mathsf{S}} U_i \quad H_i \leqslant_{\mathsf{P}} H_i'}{\oplus_{i \in I \cup J} !l_i(U_i).H_i \leqslant_{\mathsf{P}} \oplus_{i \in I} !l_i(U_i').H_i'} \;(\text{S-ParSel})$$

$$\dfrac{}{\mathbf{end} \leqslant_{\mathsf{P}} \mathbf{end}} \;(\text{S-ParEnd}) \qquad \dfrac{H\{\mu\mathbf{t}.H/\mathbf{t}\} \leqslant_{\mathsf{P}} H'}{\mu\mathbf{t}.H \leqslant_{\mathsf{P}} H'} \;(\text{S-Par}\mu\text{L}) \qquad \dfrac{H \leqslant_{\mathsf{P}} H'\{\mu\mathbf{t}.H'/\mathbf{t}\}}{H \leqslant_{\mathsf{P}} \mu\mathbf{t}.H'} \;(\text{S-Par}\mu\text{R})$$

Figure 3 Subtyping for session types (top) and partial session types (bottom).

$\mathbf{end}$. The projection of a selection $\mathtt{p} \oplus_{i \in I} !l_i(U_i).S_i$ (resp. branching $\mathtt{p} \,\&_{i \in I}\, ?l_i(U_i).S_i$) on role $\mathtt{p}$, produces a partial selection type $\oplus_{i \in I} !l_i(U_i).(S_i \restriction \mathtt{p})$ (resp. branching $\&_{i \in I}\, ?l_i(U_i).S_i \restriction \mathtt{p}$) with the continuations projected on $\mathtt{p}$. Otherwise, if projecting on $\mathtt{q} \neq \mathtt{p}$, the select/branch is "skipped", and the projection is the *merging of the continuations*, i.e., $\bigsqcap_{i \in I} (S_i \restriction \mathtt{q})$.

The $\sqcap$ operator (introduced in [61, 17]) expands the set of session types whose partial projections are defined, which allows to type more processes (as we will see in Def. 2.11 and Ex. 2.14 later on). Crucially, $\sqcap$ can compose different *internal* choices, but *not* external choices (because this could break type safety).

Subtyping *Session subtyping*, intuitively, says that a "smaller" type is "less demanding": it types channels that allow for more internal (and impose less external) choices.

▶ **Definition 2.10** (Subtyping). *The* subtyping $\leqslant_{\mathsf{S}}$ on multiparty session types *is the largest relation such that (i) if* $S \leqslant_{\mathsf{S}} S'$*, then* $\forall \mathtt{p} \in (\text{roles}(S) \cup \text{roles}(S'))$ $S \restriction \mathtt{p} \leqslant_{\mathsf{P}} S' \restriction \mathtt{p}$*, and (ii) is closed backwards under coinductive rules at the top of Fig. 3. The* subtyping $\leqslant_{\mathsf{P}}$ on partial session types *is coinductively defined by the rules at the bottom of Fig. 3.*

Clause (i) of Def. 2.10 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. 2.11). The gist of Def. 2.10 lies in clause (ii). Rules (S-Brch) and (S-Sel) define subtyping on branch and select types, respectively. Both rules are covariant in the continuation types, i.e., $S_i \leqslant_{\mathsf{S}} S_i'$. (S-Brch) is covariant also in the number of branches offered, whereas (S-Sel) is contravariant. (S-B) relates base types, if they are related by $\leqslant_{\mathsf{B}}$. (S-End) relates terminated channel types. (S-μL) and (S-μR) are standard: they say that a recursive session type $\mu\mathbf{t}.S$ is related to S', iff its unfolding is related, too. The subtyping $\leqslant_{\mathsf{P}}$ for partial types is similar, except for the lack of role annotations (thus resembling the *binary* session subtyping [21]).

Multiparty Session Typing System Before delving into the session typing rules, we need to formalise the notions of *typing context* and *typing judgement*, defined below.

▶ **Definition 2.11.** *A* session typing context Γ *is a partial mapping defined as:*

$$\Gamma \;::=\; \varnothing \;\mid\; \Gamma, x{:}U \;\mid\; \Gamma, s[\mathtt{p}]{:}S \;\; (\textit{with } \mathtt{p} \notin S)$$

We say that Γ is consistent *iff for all* $s[\mathtt{p}]{:}S_{\mathtt{p}}, s[\mathtt{q}]{:}S_{\mathtt{q}} \in \Gamma$ *with* $\mathtt{p} \neq \mathtt{q}$*, we have* $\overline{S_{\mathtt{p}} \restriction \mathtt{q}} \leqslant_{\mathsf{P}} S_{\mathtt{q}} \restriction \mathtt{p}$*. We say that* Γ is complete *iff for all* $s[\mathtt{p}]{:}S_{\mathtt{p}} \in \Gamma$*,* $\mathtt{q} \in S_{\mathtt{p}}$ *implies* $s[\mathtt{q}] \in \text{dom}(\Gamma)$*. We say*

$$\text{(T-Name)}\ \frac{\text{un}(\Gamma)}{\Gamma, c{:}S \vdash c{:}S} \qquad \text{(T-Basic)}\ \frac{\text{un}(\Gamma) \quad v \in B}{\Gamma \vdash v{:}B} \qquad \text{(T-DefCtx)}\ \frac{}{\Theta, X{:}\widetilde{U} \vdash X{:}\widetilde{U}} \qquad \text{(T-Sub)}\ \frac{\Theta \cdot \Gamma, c{:}S \vdash P \quad S' \leqslant_{\mathsf{S}} S}{\Theta \cdot \Gamma, c{:}S' \vdash P}$$

$$\text{(T-Nil)}\ \frac{\text{un}(\Gamma)}{\Theta \cdot \Gamma \vdash \mathbf{0}} \qquad \text{(T-Par)}\ \frac{\Theta \cdot \Gamma_1 \vdash P \quad \Theta \cdot \Gamma_2 \vdash Q}{\Theta \cdot \Gamma_1 \circ \Gamma_2 \vdash P \mid Q} \qquad \text{(T-Res)}\ \frac{\Theta \cdot \Gamma, \Gamma' \vdash P \quad \Gamma' = \{s[\mathsf{p}]{:}S_{\mathsf{p}}\}_{\mathsf{p} \in I} \text{ complete}}{\Theta \cdot \Gamma \vdash (\boldsymbol{\nu} s{:}\Gamma')P}$$

$$\text{(T-Brch)}\ \frac{\forall i \in I \quad \Theta \cdot \Gamma, x_i{:}U_i, c{:}S_i \vdash P_i}{\Theta \cdot \Gamma, c{:}\mathsf{p}\ \&_{i \in I}\ ?l_i(U_i).S_i \vdash c[\mathsf{p}]\ \&_{i \in I}\ \{l_i(x_i).P_i\}} \qquad \text{(T-Sel)}\ \frac{\Gamma_1 \vdash v{:}U \quad \Theta \cdot \Gamma_2, c{:}S \vdash P}{\Theta \cdot \Gamma_1 \circ \Gamma_2, c{:}\mathsf{p} \oplus !l(U).S \vdash c[\mathsf{p}] \oplus \langle l(v) \rangle.P}$$

$$\text{(T-Def)}\ \frac{\Theta, X{:}\widetilde{U} \cdot \widetilde{x}{:}\widetilde{U} \vdash P \quad \Theta, X{:}\widetilde{U} \cdot \Gamma \vdash Q}{\Theta \cdot \Gamma \vdash \mathbf{def}\ X(\widetilde{x}{:}\widetilde{U}) = P\ \mathbf{in}\ Q} \qquad \text{(T-Call)}\ \frac{\forall i \in \{1..n\} \quad \Gamma_i \vdash v_i{:}U_i \quad \text{un}(\Gamma)}{\Theta, X{:}U_1, \ldots, U_n \cdot \Gamma_1 \circ \ldots \circ \Gamma_n \circ \Gamma \vdash X\langle v_1, \ldots, v_n \rangle}$$

Figure 4 Typing rules for the multiparty session π-calculus.

that Γ is unrestricted, $\text{un}(\Gamma)$, *iff for all* $c \in \text{dom}(\Gamma)$, $\Gamma(c)$ *is either a base type or* $\mathbf{end}$. *The* typing contexts composition $\circ$ *is the commutative operator with* $\varnothing$ *as neutral element:*

$$\Gamma_1, c{:}U \ \circ\ \Gamma_2, c'{:}U' \ \triangleq\ (\Gamma_1 \circ \Gamma_2), c{:}U, c'{:}U' \quad (\textit{if}\ \text{dom}(\Gamma_2) \not\ni c \neq c' \notin \text{dom}(\Gamma_1))$$
$$\Gamma_1, x{:}B \ \circ\ \Gamma_2, x{:}B \ \triangleq\ (\Gamma_1 \circ \Gamma_2), x{:}B$$

Note that a typing context can map a channel with role $s[\mathsf{p}]$ to a session type S (that cannot refer to p itself, ruling out "self-interactions"), but *not* to a base type. Variables, instead, can be mapped to either session or base types. The clause "$\forall c{:}S \in \Gamma : S \upharpoonright \mathsf{p}$ is defined" clause is discussed below.

On Consistency In Def. 2.11, and in the rest of this work, we emphasise the importance of *consistency* of the context Γ for session typing: this condition is, in fact, *necessary to prove subject reduction*, and will be central for our encoding (§ 5 and § 6). As an example of *non*-consistent typing context, consider $s[\mathsf{p}]{:}\mathbf{end}, s[\mathsf{q}]{:}\mathsf{p}?l(U).S$: we have $\overline{\mathbf{end} \upharpoonright \mathsf{q}} = \mathbf{end} \not\leqslant_{\mathsf{P}} ?l(U).S = (\mathsf{p}?l(U).S) \upharpoonright \mathsf{p}$.

Note that our consistency in Def. 2.11 is *weaker* than the one in previous papers (where it is sometimes called *coherency*): we use $\leqslant_{\mathsf{P}}$, instead of (syntactic) type equality $=$, to relate dual partial projections. The reason being: if we use $=$, *and* allow partial projections with type merging (Def. 2.9), subject reduction does *not* hold. Hence, by relaxing our definition, and proving Theorem 2.16 later on, we fix a longstanding mistake appearing e.g., in [61, 17].

▶ **Definition 2.12** (Session typing judgements). *The* process declaration typing context Θ *maps process variables* X *to* n*-tuples of types* $\widetilde{U}$ *(one per argument of* X*), and is defined as:*

$$\Theta \ ::=\ \varnothing \mid \Theta, X{:}\widetilde{U}$$

Typing judgements *are inductively defined by the rules in Fig. 4, and have the forms:*

for processes: $\Theta \cdot \Gamma \vdash P$ *(with* Γ *consistent, and* $\forall c{:}S \in \Gamma,\ S \upharpoonright \mathsf{p}$ *is defined* $\forall \mathsf{p} \in S$*)*

for values: $\Gamma \vdash v{:}U$ *for process variables:* $\Theta \vdash X{:}\widetilde{U}$

(T-Name) says that a channel has the type assumed in the session typing context. (T-Basic) relates base values to their type. (T-DefCtx) says that a process name has the type assumed in the process declaration typing context. (T-Sub) is the standard subsumption rule, using $\leqslant_{\mathsf{S}}$ (Def. 2.10). (T-Nil) says that the terminated process is well typed in any unrestricted typing context. (T-Par) says that the parallel composition of P and Q is well typed under the composition of the corresponding typing contexts, as per Def. 2.11. (T-Res) says that $(\boldsymbol{\nu} s)P$ is well typed in Γ, if s occurs in a *complete* set of typed channels with roles (denoted with Γ'), and the open process P is well typed in the "full" context Γ, Γ'. For convenience, we annotate the restricted s with Γ' in the process, giving $(\boldsymbol{\nu} s{:}\Gamma')P$. (T-Brch) (resp. (T-Sel)) state that branching (resp. selection) process on $c[\mathtt{p}]$ is well typed if $c[\mathtt{p}]$ is of compatible branching (resp. selection) type, and the continuations P_i, for all $i \in I$, are well typed with the continuation session types. (T-Def) says that a process definition $\mathbf{def}\ X(\widetilde{x}) = P\ \mathbf{in}\ Q$ is well typed if both P and Q are well typed in their typing contexts enriched with $\widetilde{x}{:}\widetilde{U}$. For convenience, we annotate $\widetilde{x}$ with types $\widetilde{U}$. (T-Call) says that process call $X\langle v_1, \ldots, v_n\rangle$ is well typed if the actual parameters $v_1, \ldots, v_n$ have compatible types w.r.t. X.

As mentioned above, we emphasise the importance of consistency by restricting our process typing judgements to *consistent* typing contexts—i.e., those that allow to prove subject reduction. The clause "$\forall c{:}S \in \Gamma : S \upharpoonright \mathtt{p}$ is defined" is not usual in MPST papers, but stems naturally: by requiring the existence of partial projections, the clause rejects processes that *(a)* use some channel with role $s[\mathtt{p}]{:}S$ that, for some $\mathtt{q} \in S$, cannot be (consistently) paired with $s[\mathtt{q}]$, or *(b)* contain some variable $x{:}S$ that, in a consistent and complete Γ, cannot be substituted by any $s[\mathtt{p}]{:}S$. Hence, such rejected processes cannot participate in any complete session (case *(a)*), or are never-executed "dead code" (case *(b)*).

▶ **Remark 2.13.** *Unlike most MPST papers (e.g., [18, 10]), our rule (*T-Res*) does* not *directly map a session s to a global type: this is explained in the next section, "Global Types".*

▶ **Example 2.14.** *Consider the session type $S_{\mathtt{b}}$ in § 1 (equation* (1)*), and the client process $P_{\mathtt{b}} = \mathit{Client}_{\mathtt{b}}\langle s_{\mathtt{b}}[\mathtt{p}]\rangle$ from Ex. 2.2. By Def. 2.12, the following typing judgement holds:*

$$\mathit{Client}_{\mathtt{b}}{:}\mathtt{q}?\mathtt{PlayB}(S_{\mathtt{b}}),\ \mathit{Loop}_{\mathtt{b}}{:}\mu\mathbf{t}.\,\mathtt{a}\,\&\left\{\begin{array}{l}?\mathtt{Mov1AB}(\mathtt{Int}).\mathtt{c}!\mathtt{Mov1BC}(\mathtt{Int}).\mathbf{t}\,,\\ ?\mathtt{Mov2AB}(\mathtt{Bool}).\mathtt{c}!\mathtt{Mov2BC}(\mathtt{Bool}).\mathbf{t}\end{array}\right\}\cdot s_{\mathtt{b}}[\mathtt{p}]{:}\mathtt{q}?\mathtt{PlayB}(S_{\mathtt{b}})\ \vdash\ \mathit{Client}_{\mathtt{b}}\langle s_{\mathtt{b}}[\mathtt{p}]\rangle$$

It says that the channel with role $s_{\mathtt{b}}[\mathtt{p}]$ is used following type $\mathtt{q}?\mathtt{PlayB}(S_{\mathtt{b}}).\mathbf{end}$ (with a delegation of a $S_{\mathtt{b}}$-typed channel); the argument of $\mathit{Client}_{\mathtt{b}}$ has the same type; the argument of $\mathit{Loop}_{\mathtt{b}}$ is used following the game loop. This example cannot be typed *without merging $\sqcap$ (Def. 2.9): its derivation requires to compute $S_{\mathtt{b}}\upharpoonright\mathtt{c} = !\mathtt{InfoBC}(\mathtt{String}).\mu\mathbf{t}.(!\mathtt{Mov1BC}(\mathtt{Int}).\mathbf{t} \sqcap !\mathtt{Mov2BC}(\mathtt{Bool}).\mathbf{t}) = !\mathtt{InfoBC}(\mathtt{String}).\mu\mathbf{t}.(!\mathtt{Mov1BC}(\mathtt{Int}).\mathbf{t} \oplus !\mathtt{Mov2BC}(\mathtt{Bool}).\mathbf{t})$, which is undefined without merging.*

The typing rules in Fig. 4 satisfy a subject reduction property (Theorem 2.16) based on *typing context reductions*: they reflect the communications required by the types in Γ.

▶ **Definition 2.15.** *The* typing context reduction $\Gamma \to \Gamma'$ *is:*

$$\begin{array}{rcll} s[\mathtt{p}]{:}S_{\mathtt{p}},\ s[\mathtt{q}]{:}S_{\mathtt{q}} & \to & s[\mathtt{p}]{:}S_k,\ s[\mathtt{q}]{:}S'_k & \text{if} \left\{\begin{array}{ll} \mathrm{unf}(S_{\mathtt{p}}) = \mathtt{q} \oplus_{i\in I} !l_i(U_i).S_i & k \in I \\ \mathrm{unf}(S_{\mathtt{q}}) = \mathtt{p}\ \&_{i\in I\cup J}\ ?l_i(U'_i).S'_i & U_k \leqslant_{\mathsf{S}} U'_k \end{array}\right. \\ \Gamma,\ c{:}U & \to & \Gamma',\ c{:}U' & \text{if}\ \Gamma \to \Gamma'\ \text{and}\ U \leqslant_{\mathsf{S}} U' \end{array}$$

Our Def. 2.15 is a bit less straightforward than the ones in literature: it accommodates subtyping (hence, uses $\leqslant_{\mathsf{S}}$) and our iso-recursive type equality (hence, unfolds types explicitly).

▶ **Theorem 2.16** (Subject reduction). *If $\Theta \cdot \Gamma \vdash P$ and $P \to P'$, then there exists Γ' such that $\Gamma \to^* \Gamma'$ and $\Theta \cdot \Gamma' \vdash P'$.*

Global Types We conclude this section by discussing *global types*, that we mentioned in the opening of § 2.1 and Remark 2.13.

▶ **Definition 2.17.** *The syntax of global types, ranged over by G, is:*

$$\begin{array}{rcll} G & ::= & \mathtt{p} \to \mathtt{q}{:}\{l_i(U_i).G_i\}_{i\in I} & \textit{(interaction — with } U_i \textit{ closed)} \\ & & \mu\mathbf{t}.G \ \mid\ \mathbf{t}\ \mid\ \mathbf{end} & \textit{(recursive type, type variable, termination)} \end{array}$$

Type $\mathtt{p} \to \mathtt{q}{:}\{l_i(U_i).G_i\}_{i\in I}$ states that role $\mathtt{p}$ sends to role $\mathtt{q}$ one of the (pairwise distinct) labels l_i for $i \in I$, together with a payload U_i (Def. 2.5). If the chosen label is l_j, then the interaction proceeds as G_j. Type $\mu\mathbf{t}.G$ and type variable $\mathbf{t}$ model recursion. Type $\mathbf{end}$ states the termination of a protocol. We omit the braces $\{...\}$ from interactions when I is a singleton: e.g., $\mathtt{a}\to\mathtt{b}{:}l_1(U_1).G_1$ stands for $\mathtt{a}\to\mathtt{b}{:}\{l_i(U_i).G_i\}_{i\in\{1\}}$.

▶ **Example 2.18.** *The following global type formalises the Game described in § 1 and Fig. 1:*

$$G_{Game} \;=\; \mathtt{b}\to\mathtt{c}{:}\mathtt{InfoBC}(\mathtt{String})\,.\,\mathtt{c}\to\mathtt{a}{:}\mathtt{InfoCA}(\mathtt{String})\,.\,\mathtt{a}\to\mathtt{b}{:}\mathtt{InfoAB}(\mathtt{String})\,.$$
$$\mu\mathbf{t}.\mathtt{a}\to\mathtt{b}{:}\left\{\begin{array}{l}\mathtt{Mov1AB}(\mathtt{Int}).\mathtt{b}\to\mathtt{c}{:}\mathtt{Mov1BC}(\mathtt{Int}).\mathtt{c}\to\mathtt{a}{:}\left\{\begin{array}{l}\mathtt{Mov1CA}(\mathtt{Int}).\mathbf{t}\,,\\ \mathtt{Mov2CA}(\mathtt{Bool}).\mathbf{t}\end{array}\right\},\\ \mathtt{Mov2AB}(\mathtt{Bool}).\mathtt{b}\to\mathtt{c}{:}\mathtt{Mov2BC}(\mathtt{Bool}).\mathtt{c}\to\mathtt{a}{:}\left\{\begin{array}{l}\mathtt{Mov1CA}(\mathtt{Int}).\mathbf{t}\,,\\ \mathtt{Mov2CA}(\mathtt{Bool}).\mathbf{t}\end{array}\right\}\end{array}\right\}$$

In MPST theory, a global type G with roles $\mathtt{p}_i$ $(i \in I)$ is used to *project*[1] a set of session types S_i (one per role). E.g., projecting G_{Game} in Ex. 2.18 onto $\mathtt{b}$ yields the session type $S_{\mathtt{b}}$ (see (1)). When *all* such projections S_i are defined, *and* all partial projections of each S_i are defined (as per Def. 2.9), then we can define the *projected typing context of* G:

$$\Gamma_G \;=\; \{s[\mathtt{p}_i]{:}S_i\}_{i\in I} \qquad \text{where } \forall i \in I:\; S_i \text{ is the projection of } G \text{ onto } \mathtt{p}_i$$

and Γ_G can be shown to be: (a) *consistent and complete*, i.e., can be used to type the session s by rule (T-Res) (Fig. 4), and (b) *deadlock-free*, i.e.: $\Gamma_G \to^* \Gamma'_G \not\to$ implies $\forall i \in I : \Gamma'_G(s[\mathtt{p}_i]) = \mathbf{end}$. Similarly, it can be shown that Γ_G reduces as prescribed by G.

Now, from observation (a) above, we can easily define a "strict" version of rule (T-Res) (Fig. 4) in the style of [18, 10], where (1) the clause "Γ' *complete*" is replaced with "Γ' *is the projected typing context of some* G", and (2) in the conclusion, the annotation $(\boldsymbol{\nu}s{:}\Gamma')$ is replaced with $(\boldsymbol{\nu}s{:}G)$. Further, observation (b) allows to prove Theorem 2.19 below, as shown e.g. in [4]: a typed ensemble of processes interacting on a single G-typed session is deadlock-free (note: with our rules in Fig. 4, the annotation $(\boldsymbol{\nu}s{:}G)$ would be $(\boldsymbol{\nu}s{:}\Gamma_G)$).

▶ **Theorem 2.19** (Deadlock freedom). *Let* $\varnothing\cdot\varnothing \vdash P$*, where* $P \equiv (\boldsymbol{\nu}s{:}G)\big|_{i\in I}P_i$ *and each* P_i *only interacts on* $s[\mathtt{p}_i]$*. Then,* P *is* deadlock-free*: i.e.,* $P \to^* P' \not\to$ *implies* $P' \equiv \mathbf{0}$.

Note that the properties above emerge by placing suitable session types S_i in the premises of (T-Res)—but our streamlined typing rules in Fig. 4 do *not* require it, *nor* mention G. The main property of such rules is ensuring *type safety* (Theorem 2.16). We will exploit this insight (obtained by our separation of global/local typing) in our encoding (§ 5), preserving semantics and types (and thus, Theorem 2.19) *without* explicit references to global types.

3 Linear π-Calculus

The π-calculus is the canonical model for communication and concurrency based on message-passing and *channel mobility*. It was created towards the end of 1980's, with the first paper published in 1992 [44], followed by various proposals for types and type systems. In this section we summarise the standard π-calculus with linear types [35]. The contents of this section are standard, and based on [54]; we present new π-calculus-related results in § 4.

▶ **Definition 3.1.** *The* syntax *of* π*-calculus* processes *and* values *is:*

[1] We use a *standard* projection with merging [61, 17]: for its definition (not crucial here), see § A.2.

$$
\begin{array}{rcll}
P, Q & ::= & \mathbf{0} \mid P \mid Q \mid (\boldsymbol{\nu} x)P & \textit{(inaction, parallel composition, restriction)} \\
& & {*P} \mid \overline{x}\langle v\rangle.P \mid x(y).P & \textit{(process replication, output, input)} \\
& & \mathbf{case}\, v\, \mathbf{of}\, \{l_i(x_i) \triangleright P_i\}_{i\in I} & \textit{(variant destruct)} \\
& & \mathbf{with}\, [l_i : x_i]_{i\in I} = v\, \mathbf{do}\, P & \textit{(labelled tuple destruct)} \\
u, v & ::= & x, y, w, z \mid l(v) \mid [l_i : v_i]_{i\in I} & \textit{(name, variant value, labelled tuple value)} \\
& & \mathtt{false} \mid \mathtt{true} \mid 42 \mid \ldots & \textit{(base value)}
\end{array}
$$

The inaction process $\mathbf{0}$, and the parallel composition $P \mid Q$ are straightforward, and similar to Def. 2.1. The restriction process $(\boldsymbol{\nu} x)P$ creates a new name x and binds it with scope P. The replicated process $*P$ represents infinite replicas of P, composed in parallel. The output process $\overline{x}\langle v\rangle.P$ uses the name x to send a value v, and proceeds as P; the input process $x(y).P$ uses x to receive a value that will substitute y in the continuation P. Process $\mathbf{case}\, v\, \mathbf{of}\, \{l_i(x_i) \triangleright P_i\}_{i\in I}$ pattern matches a variant value v, and if it has label l_i, substitutes x_i and continues as P_i. Process $\mathbf{with}\, [l_i : x_i]_{i\in I} = v\, \mathbf{do}\, P$ destructs a labelled tuple v, substituting each x_i in P. Values include names, which can be thought of as communication channels names, base values like $\mathtt{false}$ or 42, variant values $l(v)$ and labelled tuples $[l_i : v_i]_{i\in I}$. For brevity, we will often write "record" instead of "labelled tuple".

▶ **Definition 3.2.** *The* π-calculus operational semantics *is the relation* $\to$ *defined as:*

$$
\begin{array}{rl}
(\text{R}\pi\text{-Com}) & \overline{x}\langle v\rangle.P \mid x(y).Q \;\to\; P \mid Q\{^{v}/_{y}\} \\
(\text{R}\pi\text{-Case}) & \mathbf{case}\, l_j(v)\, \mathbf{of}\, \{l_i(x_i) \triangleright P_i\}_{i\in I} \;\to\; P_j\{^{v}/_{x_j}\} \quad (j \in I) \\
(\text{R}\pi\text{-With}) & \mathbf{with}\, [l_i : x_i]_{i\in I} = [l_i : v_i]_{i\in I}\, \mathbf{do}\, P \;\to\; P\{^{v_i}/_{x_i}\}_{i\in I} \\
(\text{R}\pi\text{-Res}) & P \to Q \;\; \textit{implies} \;\; (\boldsymbol{\nu} x)P \to (\boldsymbol{\nu} x)Q \\
(\text{R}\pi\text{-Par}) & P \to Q \;\; \textit{implies} \;\; P \mid R \to Q \mid R \\
(\text{R}\pi\text{-Struct}) & P \equiv P' \;\wedge\; P \to Q \;\wedge\; Q' \equiv Q \;\; \textit{implies} \;\; P' \to Q'
\end{array}
$$

(Rπ-Com) models communication between output and input on a name x: it reduces to the corresponding continuations, with a value substitution on the receiver process. (Rπ-Case) says that **case** applied on a variant value $l_j(v)$ reduces to P_j, with v in place of x_j. (Rπ-With) says that **with** reduces to its continuation P with v_i in place of each x_i, for all $i \in I$. By (Rπ-Res), (Rπ-Par), reductions can happen under restriction and parallel composition. By (Rπ-Struct), reduction is closed under structural congruence $\equiv$: its definition is standard (see § A).

π-Calculus Typing We now summarise the π-calculus types and typing rules.

▶ **Definition 3.3** (π-types)**.** *The syntax of a* π-calculus type T *is given by:*

$$
\begin{array}{rcll}
T & ::= & \mathsf{Li}(T) \mid \mathsf{Lo}(T) \mid \mathsf{L}\sharp(T) & \textit{(linear input, linear output, linear connection)} \\
& & \sharp(T) \mid \bullet & \textit{(unrestricted connection, no capability)} \\
& & \langle l_i\_T_i\rangle_{i\in I} \mid [l_i : T_i]_{i\in I} & \textit{(variant, labelled tuple a.k.a. "record")} \\
& & \mu\mathbf{t}.T \mid \mathbf{t} \mid \mathtt{Bool} \mid \mathtt{Int} \mid \ldots & \textit{(recursive type, type variable, base type)}
\end{array}
$$

Linear types $\mathsf{Li}(T)$, $\mathsf{Lo}(T)$ denote, respectively, names used *exactly once* to input/output a value of type T. $\mathsf{L}\sharp(T)$ denotes a name used once for sending, and once for receiving, a message of type T. $\sharp(T)$ denotes an *unrestricted connection*, i.e., a name that can be used both for input/output any number of times. $\bullet$ is assigned to names that cannot be used for input/output. $\langle l_i\_T_i\rangle_{i\in I}$ is a labelled disjoint union of types, while $[l_i : T_i]_{i\in I}$ (that we will often call "record") is a labelled product type; for both, labels l_i are all distinct, and their order is irrelevant. As syntactic sugar, we write $(T_i)_{i\in 1..n}$ for a record with integer labels $[i : T_i]_{i\in\{1,\ldots,n\}}$. Recursive types and variables, and base types like $\mathtt{Bool}$, are standard.

The predicate $\mathrm{lin}(T)$ (Def. 3.4 below) holds iff T has some linear input/output component.

▶ **Definition 3.4** (Linear/unrestricted types)**.** *The predicate* lin *is inductively defined as:*

$$(\text{T}\pi\text{-Name})\ \frac{\text{un}(\Gamma)}{\Gamma, x:T \vdash x:T} \qquad (\text{T}\pi\text{-Basic})\ \frac{\text{un}(\Gamma) \qquad v \in B}{\Gamma \vdash v:B} \qquad (\text{T}\pi\text{-LVal})\ \frac{\Gamma \vdash v:T}{\Gamma \vdash l(v):\langle l\_T\rangle}$$

$$(\text{T}\pi\text{-LTup})\ \frac{\text{un}(\Gamma) \qquad \forall i \in I \qquad \Gamma_i \vdash v_i:T_i}{\left(\biguplus_{i\in I}\Gamma_i\right) \uplus \Gamma \vdash [l_i:v_i]_{i\in I}:[l_i:T_i]_{i\in I}} \qquad (\text{T}\pi\text{-Sub})\ \frac{\Gamma \vdash x:T \qquad T \leqslant_\pi T'}{\Gamma \vdash x:T'} \qquad (\text{T}\pi\text{-Nil})\ \frac{\text{un}(\Gamma)}{\Gamma \vdash \mathbf{0}}$$

$$(\text{T}\pi\text{-Par})\ \frac{\Gamma_1 \vdash P \qquad \Gamma_2 \vdash Q}{\Gamma_1 \uplus \Gamma_2 \vdash P \mid Q} \qquad (\text{T}\pi\text{-Res1})\ \frac{\Gamma, x:\dagger(T) \vdash P \qquad \dagger \in \{\mathsf{L}\sharp, \sharp\}}{\Gamma \vdash (\boldsymbol{\nu}x)P} \qquad (\text{T}\pi\text{-Res2})\ \frac{\Gamma, x:\bullet \vdash P}{\Gamma \vdash (\boldsymbol{\nu}x)P}$$

$$(\text{T}\pi\text{-Inp})\ \frac{\Gamma_1 \vdash x:\dagger(T) \qquad \dagger \in \{\mathsf{Li}, \sharp\} \qquad \Gamma_2, y:T \vdash P}{\Gamma_1 \uplus \Gamma_2 \vdash x(y).P} \qquad (\text{T}\pi\text{-Out})\ \frac{\Gamma_1 \vdash x:\dagger(T) \qquad \dagger \in \{\mathsf{Lo}, \sharp\} \qquad \Gamma_2 \vdash v:T \qquad \Gamma_3 \vdash P}{\Gamma_1 \uplus \Gamma_2 \uplus \Gamma_3 \vdash \overline{x}\langle v\rangle.P} \qquad (\text{T}\pi\text{-Repl})\ \frac{\Gamma \vdash P \qquad \text{un}(\Gamma)}{\Gamma \vdash *P}$$

$$(\text{T}\pi\text{-Case})\ \frac{\Gamma_1 \vdash v:\langle l_i\_T_i\rangle_{i\in I} \qquad \forall i \in I \quad \Gamma_2, x_i:T_i \vdash P_i}{\Gamma_1 \uplus \Gamma_2 \vdash \textbf{case}\ v\ \textbf{of}\ \{l_i(x_i) \triangleright P_i\}_{i\in I}} \qquad (\text{T}\pi\text{-With})\ \frac{\Gamma_1 \vdash v:[l_i:T_i]_{i\in I} \qquad \Gamma_2, \{x_i:T_i\}_{i\in I} \vdash P}{\Gamma_1 \uplus \Gamma_2 \vdash \textbf{with}\ [l_i:x_i]_{i\in I} = v\ \textbf{do}\ P}$$

Figure 5 Typing rules for the linear π-calculus.

$$\text{lin}(\mathsf{Li}(T)) \qquad \text{lin}(\mathsf{Lo}(T)) \qquad \frac{\exists j \in I : \text{lin}(T_j)}{\text{lin}(\langle l_i\_T_i\rangle_{i\in I})} \qquad \frac{\exists j \in I : \text{lin}(T_j)}{\text{lin}([l_i:T_i]_{i\in I})} \qquad \frac{\text{lin}\,(T)}{\text{lin}\,(\mu\mathbf{t}.T)}$$

We write $\text{un}(T)$ *iff* $\neg\,\text{lin}(T)$ *(i.e.,* T *is unrestricted iff is not linear).*

▶ **Definition 3.5.** Subtyping $\leqslant_\pi$ *for* π*-types is coinductively defined as:*

$$\frac{B \leqslant_{\mathsf{B}} B'}{B \leqslant_\pi B'}\ (\text{S-LB}) \qquad \frac{}{\bullet \leqslant_\pi \bullet}\ (\text{S-LEnd}) \qquad \frac{T \leqslant_\pi T'}{\mathsf{Li}(T) \leqslant_\pi \mathsf{Li}(T')}\ (\text{S-Li}) \qquad \frac{T' \leqslant_\pi T}{\mathsf{Lo}(T) \leqslant_\pi \mathsf{Lo}(T')}\ (\text{S-Lo})$$

$$\frac{\forall i \in I \quad T_i \leqslant_\pi T_i'}{\langle l_i\_T_i\rangle_{i\in I} \leqslant_\pi \langle l_i\_T_i'\rangle_{i\in I\cup J}}\ (\text{S-Variant}) \qquad \frac{\forall i \in I \quad T_i \leqslant_\pi T_i'}{[l_i:T_i]_{i\in I} \leqslant_\pi [l_i:T_i']_{i\in I}}\ (\text{S-LTuple})$$

$$\frac{T\{^{\mu\mathbf{t}.T}/_{\mathbf{t}}\} \leqslant_\pi T'}{\mu\mathbf{t}.T \leqslant_\pi T'}\ (\text{S-L}\mu\text{L}) \qquad \frac{T \leqslant_\pi T'\{^{\mu\mathbf{t}.T'}/_{\mathbf{t}}\}}{T \leqslant_\pi \mu\mathbf{t}.T'}\ (\text{S-L}\mu\text{R})$$

Rule (S-LB) says that $\leqslant_\pi$ includes subtyping $\leqslant_{\mathsf{B}}$ on base types. (S-LEnd) relates types without I/O capabilities. Rule (S-Li) (resp. (S-Lo)) says that linear input (resp. output) subtyping is *covariant* (resp. *contravariant*) in the carried type. (S-Variant) says that subtyping for variant types is *covariant* in *both* carried types *and* number of components. (S-LTuple) says that subtyping for labelled tuples, a.k.a records, is *covariant* in the carried types[2]. Rules (S-LμL)/(S-LμR) relate a recursive type $\mu\mathbf{t}.T$ to T' iff its unfolding is related to T'.

▶ **Definition 3.6** (Typing context, type combination)**.** *The* linear π-calculus typing context Γ *is a partial mapping defined as:*

$$\Gamma \ ::= \ \emptyset \mid \Gamma, x:T$$

We write $\text{lin}(\Gamma)$ *iff* $\exists x:T \in \Gamma : \text{lin}(T)$*, and* $\text{un}(\Gamma)$ *iff* $\neg\,\text{lin}(\Gamma)$*. The* type combinator $\uplus$ *is defined on* π*-types as follows (and undefined in other cases), and is extended to typing contexts as expected.*

$$\mathsf{Li}(T) \uplus \mathsf{Lo}(T) \triangleq \mathsf{L}\sharp(T) \qquad \mathsf{Lo}(T) \uplus \mathsf{Li}(T) \triangleq \mathsf{L}\sharp(T) \qquad T \uplus T \triangleq T \quad \textit{if}\ \text{un}(T)$$

$$(\Gamma_1 \uplus \Gamma_2)(x) \triangleq \begin{cases} \Gamma_1(x) \uplus \Gamma_2(x) & \textit{if}\ x \in \text{dom}(\Gamma_1) \cap \text{dom}(\Gamma_2) \\ \Gamma_i(x) & \textit{if}\ x \in \text{dom}(\Gamma_i) \setminus \text{dom}(\Gamma_j) \end{cases}$$

[2] Subtyping on "full" records allows to add/remove entries [54, §7.3]; but here, "record"="labelled tuple".

$$\textbf{let}\, x = v \,\textbf{in}\, P \;\triangleq\; (\boldsymbol{\nu} z)\,(\overline{z}\langle v\rangle.\mathbf{0} \mid z(x).P) \quad \text{(where } z \notin \{x\} \cup \mathrm{fn}(v) \cup \mathrm{fn}(P))$$

$$(\textsc{R}\pi\text{-Let})\;\; \textbf{let}\, x = v \,\textbf{in}\, P \;\to\; P\{v/x\} \qquad (\textsc{T}\pi\text{-Let})\;\frac{\Gamma_1 \vdash v : T \qquad \Gamma_2, x : T \vdash P}{\Gamma_1 \uplus \Gamma_2 \vdash \textbf{let}\, x = v \,\textbf{in}\, P}$$

$$(\textsc{T}\pi\text{-Narrow})\;\frac{\Gamma, x : T \vdash P \quad T' \leqslant_\pi T}{\Gamma, x : T' \vdash P} \qquad (\textsc{T}\pi\text{-MSubst})\;\frac{\forall i \in I \quad \Gamma_i \vdash v_i : T_i \quad \Gamma, \{x_i : T_i\}_{i \in I} \vdash P}{\left(\biguplus_{i \in I} \Gamma_i\right) \uplus \Gamma \vdash P\{v_i/x_i\}_{i \in I}}$$

Figure 6 "Let" binder (definition, reduction, typing), and narrowing / substitution rules.

The typing rules for the linear π-calculus are given in Fig. 5. Typing judgements have two forms: $\Gamma \vdash v : T$ and $\Gamma \vdash P$. (Tπ-Name) says that a name has the type assumed in the typing context; (Tπ-Basic) relates base values to their types; both rules require unrestricted typing contexts. (Tπ-LVal) says that a variant value $l(v)$ is of type $\langle l\_T\rangle$ if value v is of type T. (Tπ-LTup) says that a record value $[l_i : v_i]_{i \in I}$ is of type $[l_i : T_i]_{i \in I}$ if for all $i \in I$, v_i is of type T_i. (Tπ-Sub) is the *subsumption rule*: if x has type T in Γ, then it also has any *super*type of T. (Tπ-Nil) says that $\mathbf{0}$ is well typed in every unrestricted typing context. (Tπ-Par) says that the parallel composition of two processes is typed by combining the respective typing contexts. (Tπ-Res1) says that the restriction process $(\boldsymbol{\nu} x)P$ is well typed if P is well typed by augmenting the context with $x : \mathsf{L}\sharp(T)$. By applying Def. 3.6 ($\uplus$), we have $x : \mathsf{L}\sharp(T) = x : \mathsf{Li}(T) \uplus \mathsf{Lo}(T)$: this implies that P owns *both* capabilities of linear input/output of x. Rule (Tπ-Res2) says that the restriction $(\boldsymbol{\nu} x)P$ is well typed if P is well typed and x has no capabilities. (Tπ-Inp) (resp. (Tπ-Out)) say that the input and output processes are well typed if x is a (possibly linear) name used in input (resp. output), and the carried types are compatible with the type of y (resp. value v). The typing context used to type the input and output process is obtained by applying $\uplus$ on the premises. (Tπ-Repl) says that a replicated process $*P$ is typed in the same unrestricted context that types P. (Tπ-Case) says that $\textbf{case}\, v \,\textbf{of}\, \{l_i(x_i) \triangleright P_i\}_{i \in I}$ is well typed if the guard value v has variant type, and every P_i is typed assuming $x_i : T_i$, for all $i \in I$. (Tπ-With) says that process $\textbf{with}\, [l_i : x_i]_{i \in I} = v \,\textbf{do}\, P$ is well typed if v is of record type such that for all $i \in I$, each v_i has the same type as x_i, i.e., T_i.

4 Some Typed π-Calculus Extensions and Results

We introduce some definitions and results on typed π-calculus: we will need them in § 5 and § 6, to state our encoding and its properties. As we target *standard* typed π-calculus (§ 3), all our extensions are *conservative*, so to preserve standard results (e.g., subject reduction).

"Let" binder, narrowing, substitution Fig. 6 shows several auxiliary definitions and typing rules. $\textbf{let}\, x = v \,\textbf{in}\, P$ binds x in P, and reduces by replacing x with v in P. It is a macro on other π-calculus contructs: hence, rules (Rπ-Let)/(Tπ-Let) are based on the reduction/typing of its expansion (see § A). Rule (Tπ-Narrow) derives from the narrowing lemma [54, 7.2.5]. Rule (Tπ-MSubst) represents zero or more applications of the substitution lemma [54, 8.1.4].

Duality and Recursive π-Types The *duality* for linear π-types relates opposite but compatible input/output capabilities. Intuitively, the dual of a $\mathsf{Li}(T)$ is $\mathsf{Lo}(T)$ (and *vice versa*) [14]. Note that the carried type T *is the same*: i.e., dual types can be combined with $\uplus$ (Def. 3.6), yielding $\mathsf{L}\sharp(T)$. However, defining duality for *recursive* π-types is not straightforward: what is the dual of $T = \mu\mathbf{t}.\mathsf{Lo}(\mathbf{t})$? Is it maybe $T' = \mu\mathbf{t}.\mathsf{Li}(\mathbf{t})$? Since $\uplus$ is *not* defined for μ-types, we can check whether it is defined for the *unfoldings* of our hypothetical duals T and T'. Unfortunately, we have $\mathrm{unf}(T) = \mathsf{Lo}(\mu\mathbf{t}.\mathsf{Lo}(\mathbf{t}))$ and $\mathrm{unf}(T') = \mathsf{Li}(\mu\mathbf{t}.\mathsf{Li}(\mathbf{t}))$: i.e., $\uplus$ is again undefined, so T,T' cannot be considered duals. Solving this issue is crucial: in § 5, we will need to encode recursive partial types, preserving their duality (Def. 2.8) in linear π-types.

What we want is a notion of duality that *commutes with unfolding*, so that if two recursive types are dual, and we unfold them, we get a dual pair $\mathsf{Lo}(T)/\mathsf{Li}(T)$ that can be combined with $\uplus$ (since they carry the same T). We address this issue by extending the π-calculus type variables (Def. 3.3) with their *dualised* counterpart, denoted with $\overline{\mathbf{t}}$. We allow recursive types such as $\mu\mathbf{t}.\mathsf{Li}(\overline{\mathbf{t}})$ (but *not* $\mu\overline{\mathbf{t}}.\ldots$), and postulate that when unfolding, $\overline{\mathbf{t}}$ is substituted by a "dual" type $\mu\mathbf{t}.\mathsf{Lo}(\mathbf{t})$, as formalised in Def. 4.1 below. Quite interestingly, our approach "mirrors" (on π-calculus) the "logical duality" for session types [41] (we will discuss it in § 8).

▶ **Definition 4.1.** $\overline{T}$ *is the* dual of T*, and is defined as follows:*

$$\overline{\mathsf{Li}(T)} \triangleq \mathsf{Lo}(T) \qquad \overline{\mathsf{Lo}(T)} \triangleq \mathsf{Li}(T) \qquad \overline{\bullet} \triangleq \bullet \qquad \overline{(\mathbf{t})} \triangleq \overline{\mathbf{t}} \qquad \overline{(\overline{\mathbf{t}})} \triangleq \mathbf{t} \qquad \overline{\mu\mathbf{t}.T} \triangleq \mu\mathbf{t}.\overline{T}\{\overline{\mathbf{t}}/\mathbf{t}\}$$

The substitution *of* T *for a type variable* $\mathbf{t}$ *or* $\overline{\mathbf{t}}$ *is:* $\mathbf{t}\{T/\mathbf{t}\} \triangleq T \quad \overline{\mathbf{t}}\{T/\mathbf{t}\} \triangleq \overline{T}$

The dual of a linear input type $\mathsf{Li}(T)$ is a linear output type $\mathsf{Lo}(T)$, and *vice versa*, with the payload type T unchanged, as expected. The dual of a terminated channel type $\bullet$ is itself. The dual of a type variable $\mathbf{t}$ is $\overline{\mathbf{t}}$, and the dual of a dualised type variable $\overline{\mathbf{t}}$ is $\mathbf{t}$, implying that duality on linear π-types is convolutive. The dual of $\mu\mathbf{t}.T$ is $\mu\mathbf{t}.\overline{T}\{\overline{\mathbf{t}}/\mathbf{t}\}$, where type T is dualised to $\overline{T}$, and every occurrence of $\mathbf{t}$ is replaced by its dual $\overline{\mathbf{t}}$ by Def. 4.1. Now, the desired commutativity between duality and unfolding holds, as per Lemma 4.2 below.

▶ **Lemma 4.2.** $\mathrm{unf}(\overline{T}) = \overline{\mathrm{unf}(T)}$.

▶ **Example 4.3.** *Let* $T = \mu\mathbf{t}.\mathsf{Li}((\mathbf{t},\overline{\mathbf{t}}))$. *Then:*

$\mathrm{unf}(T) = \mathsf{Li}\Big(\big(\mu\mathbf{t}.\mathsf{Li}((\mathbf{t},\overline{\mathbf{t}})), \overline{\mu\mathbf{t}.\mathsf{Li}((\mathbf{t},\overline{\mathbf{t}}))}\big)\Big) = \mathsf{Li}\big((\mu\mathbf{t}.\mathsf{Li}((\mathbf{t},\overline{\mathbf{t}})), \mu\mathbf{t}.\mathsf{Lo}((\overline{\mathbf{t}},\mathbf{t})))\big)$; *and*

$\mathrm{unf}(\overline{T}) = \mathrm{unf}\big(\mu\mathbf{t}.\mathsf{Lo}((\overline{\mathbf{t}},\mathbf{t}))\big) = \mathsf{Lo}\big((\mu\mathbf{t}.\mathsf{Li}((\mathbf{t},\overline{\mathbf{t}})), \mu\mathbf{t}.\mathsf{Lo}((\overline{\mathbf{t}},\mathbf{t})))\big) = \overline{\mathrm{unf}(T)}$

By adding dualised type variables in Def. 3.3, we naturally extend the definition of $\mathrm{fv}(T)$ (treating $\mu\mathbf{t}.\ldots$ as a binder for both $\mathbf{t}$ and $\overline{\mathbf{t}}$), the subtyping relation $\leqslant_\pi$ in Def. 3.5 (by letting rules (S-Lμl) and (S-Lμr) use the substitution in Def. 4.1) and ultimately the typing system in Def. 3.6. This will allow us to obtain a rather simple encoding of recursive session types (Def. 5.1), and solve a subtle issue involving duality, recursion and continuations (Ex. 5.3).

The reader might be puzzled about the impact of dualised variables in the π-calculus theory. We show that dualised variables *do not increase the expressiveness of linear π-types*, and *do not unsafely enlarge subtyping* $\leqslant_\pi$: this is proved in Lemma 4.4, that allows to erase dualised variables from recursive π-types. It uses *(1)* a substitution that *only* replaces dualised variables, i.e.: $\overline{\mathbf{t}}\{\mathbf{t}'/\overline{\mathbf{t}}\} = \mathbf{t}'$; *(2)* the equivalence $=_\pi$ defined as: $\leqslant_\pi \cap \leqslant_\pi^{-1}$ (see Def. C.1).

▶ **Lemma 4.4** (Erasure of $\overline{\mathbf{t}}$). $\mu\mathbf{t}.T =_\pi \mu\mathbf{t}.T\big\{{\mu\mathbf{t}'.\overline{T}\{\mathbf{t}'/\overline{\mathbf{t}}\}}/\overline{\mathbf{t}}\big\}$*, for all* $\mathbf{t}' \notin \mathrm{fv}(T)$.

▶ **Example 4.5** (Application of erasure). *Take T from Ex. 4.3. By Lemma 4.4, we have:*
$T =_\pi \mu\mathbf{t}.\mathsf{Li}\Big(\big(\mathbf{t}, \mu\mathbf{t}'.\overline{\mathsf{Li}((\mathbf{t},\overline{\mathbf{t}}))}\{\mathbf{t}'/\overline{\mathbf{t}}\}\big)\Big) = \mu\mathbf{t}.\mathsf{Li}((\mathbf{t}, \mu\mathbf{t}'.\mathsf{Lo}((\mathbf{t},\mathbf{t}'))))$.

Since $T =_\pi T'$ implies $T \leqslant_\pi T'$ and $T' \leqslant_\pi T$, Lemma 4.4 says that any $\mu\mathbf{t}.T$ is equivalent to a μ-type *without occurrences of* $\overline{\mathbf{t}}$: i.e., any typing relation with instances of $\overline{\mathbf{t}}$ corresponds to a $\overline{\mathbf{t}}$-free one. As a consequence, any typing derivation using $\overline{\mathbf{t}}$ can be turned into a $\overline{\mathbf{t}}$-free one. Summing up: adding dualised variables preserves the standard results of typed π-calculus.

Type Combinator $\boxplus$ Def. 4.6 introduces a type combinator that is a "relaxed" version of $\uplus$ (Def. 3.6) allowing for subtyping. We will use it to encode MPST typing contexts (Def. 5.6).

▶ **Definition 4.6.** *The* π-calculus type combinator $\Cap$ *is defined on* π*-types as follows (and undefined in other cases), and naturally extended to typing contexts:*

$$\left.\begin{array}{rcl}\mathsf{Lo}(T) \Cap \mathsf{Li}(T') & \triangleq & \mathsf{Li}(T) \uplus \mathsf{Lo}(T) \\ \mathsf{Li}(T') \Cap \mathsf{Lo}(T) & \triangleq & \mathsf{Li}(T) \uplus \mathsf{Lo}(T)\end{array}\right\} \text{if } T \leqslant_\pi T' \qquad T \Cap T \triangleq T \quad \text{if } \mathrm{un}(T)$$

$$(\Gamma_1 \Cap \Gamma_2)(x) \triangleq \begin{cases}\Gamma_1(x) \Cap \Gamma_2(x) & \text{if } x \in \mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2) \\ \Gamma_i(x) & \text{if } x \in \mathrm{dom}(\Gamma_i) \setminus \mathrm{dom}(\Gamma_j)\end{cases}$$

The difference between $\uplus$ and $\Cap$ is that the former combines linear inputs/outputs with *the same carried type*, while $\Cap$ is a more relaxed relation and allows one carried type to be subtype of the other, and (when defined) yields a linear connection allowing transmission of values of *both* carried types. This is shown in Lemma 4.7 and Ex. 4.8 below.

▶ **Lemma 4.7.** *If* $T = T_1 \Cap T_2$*, and* $T_1' \uplus T_2' = T$*, then either (a)* $T_1' \leqslant_\pi T_1$ *and* $T_2' \leqslant_\pi T_2$*, or (b)* $T_1' \leqslant_\pi T_2$ *and* $T_2' \leqslant_\pi T_1$*.*

Lemma 4.7 says that $T_1 \Cap T_2$ (when defined) is a type that, when split using $\uplus$, yields linear I/O types that are subtypes of the originating T_1, T_2.

▶ **Example 4.8.** *Let* $T_1 = \mathsf{Li}(\mathtt{Real})$*,* $T_2 = \mathsf{Lo}(\mathtt{Int})$*, and* $T = T_1 \Cap T_2$*. We have* $T = \mathsf{L}\sharp(\mathtt{Int})$*; if we let* $T_1' \uplus T_2' = T$*, then we get either* (a) $T_1' = \mathsf{Li}(\mathtt{Int}) \leqslant_\pi T_1$ *and* $T_2' = \mathsf{Lo}(\mathtt{Int}) \leqslant_\pi T_2$*, or* (b) $T_1' = \mathsf{Lo}(\mathtt{Int}) \leqslant_\pi T_2$ *and* $T_2' = \mathsf{Li}(\mathtt{Int}) \leqslant_\pi T_1$*.*

5 Encoding Multiparty Session-π into Linear π-Calculus

We now present our encoding of MPST π-calculus into linear π-calculus. It consists of an *encoding of types* and an *encoding of processes*: combined, they preserve the safety properties of MPST communications, both w.r.t. typing and process behaviour.

Encoding of Types Our goal is to decompose multiparty session channel endpoints into point-to-point π-calculus channels. One intuitive way to achieve this is to *encode MPST channel endpoints as labelled tuples*, such that *each role involved in a session maps to a* π*-calculus name*: i.e., if the labelled tuple has an entry for $\mathtt{p}$, it should map to a name that allows to send/receive messages to/from some other process, which in turn should be interpreting the role of $\mathtt{p}$ in the originating session. This suggests that an encoded MPST channel endpoint must be typed as a π-calculus labelled tuple; and since each name appearing in such tuple is used for communication, it should be typed with a linear input/output type.

▶ **Definition 5.1.** *The* encoding of S into linear π-types *is:*

$$[\![S]\!] \triangleq [\mathtt{p} : [\![S \upharpoonright \mathtt{p}]\!]]_{\mathtt{p} \in S}$$

where the encoding of the partial projections $[\![S \upharpoonright \mathtt{p}]\!]$ *is:*

$$\begin{array}{rcl}[\![\oplus_{i \in I} \,!l_i(U_i).H_i]\!] & \triangleq & \mathsf{Lo}\big(\langle l_i\_([\![U_i]\!], \overline{[\![H_i]\!]})\rangle_{i \in I}\big) \qquad [\![B]\!] \triangleq B \qquad [\![\mathbf{end}]\!] \triangleq \bullet \\ [\![\&_{i \in I} \,?l_i(U_i).H_i]\!] & \triangleq & \mathsf{Li}\big(\langle l_i\_([\![U_i]\!], [\![H_i]\!])\rangle_{i \in I}\big) \qquad [\![\mathbf{t}]\!] \triangleq \mathbf{t} \qquad [\![\mu\mathbf{t}.H]\!] \triangleq \mu\mathbf{t}.[\![H]\!]\end{array}$$

The encoding of a session type S, namely $[\![S]\!]$, is a record that maps each role $\mathtt{p} \in S$ to the encoding of the *partial projection* $[\![S \upharpoonright \mathtt{p}]\!]$. The encoding of partial projections, in turn, adopts the basic idea of the encoding of *binary, non-recursive* session types [34, 14]: it is the identity on a base type B, while a terminated channel type $\mathbf{end}$ becomes $\bullet$, with no capabilities. Selection $\oplus_{i \in I} \,!l_i(U_i).H_i$ and branching $\&_{i \in I} \,?l_i(U_i).H_i$ are encoded as linear output and input types, respectively, adopting a *continuation-passing style (CPS)*. In both cases, the carried types are variants: $\langle l_i\_([\![U_i]\!], \overline{[\![H_i]\!]})\rangle_{i \in I}$ for select and $\langle l_i\_([\![U_i]\!], [\![H_i]\!])\rangle_{i \in I}$

for branch, with the same labels as the originating partial projections. Such variants carry tuples $(\llbracket U_i \rrbracket, \overline{\llbracket H_i \rrbracket})$ and $(\llbracket U_i \rrbracket, \llbracket H_i \rrbracket)$: the first element is the encoded payload type, and the second (i.e., the encoding of the continuation H_i) is the type of a *continuation name*: it is sent together with the encoded payload, and will be used to send/receive the *next* message (unless H_i is **end**). Note that selection sends the *dual* of $\llbracket H_i \rrbracket$: this is because the *sender* is expected to keep interacting according to $\llbracket H_i \rrbracket$, while the *recipient* must operate *dually* (cf. Def. 4.1). E.g., if $\llbracket H_i \rrbracket$ requires to send a message, the recipient of $\overline{\llbracket H_i \rrbracket}$ must receive it. The encodings of a type variable and a recursive type are homomorphic.

Notice that by encoding session types as labelled tuples, we untangle the order of the interactions among different roles. This order will be, however, recovered by the encoding of processes, presented later on.

▶ **Example 5.2.** *Consider the session type* $S \triangleq \mathtt{p}!l_1(\mathtt{Int}).\mathtt{q}?l_2(S').\mathbf{end}$*, where* $S' \triangleq \mathtt{r}!l_3(\mathtt{Bool}).\mathtt{q}?l_4(\mathtt{String}).\mathbf{end}$*. By Def. 5.1, the encoding of* S *is:*

$$\begin{array}{rcl}\llbracket S \rrbracket & = & [\mathtt{p}\colon \llbracket S \restriction \mathtt{p} \rrbracket, \mathtt{q}\colon \llbracket S \restriction \mathtt{q} \rrbracket] \;=\; [\mathtt{p}\colon \llbracket !l_1(\mathtt{Int}) \rrbracket, \mathtt{q}\colon \llbracket ?l_2(S') \rrbracket] \\ & = & [\mathtt{p}\colon \mathsf{Lo}(\langle l_1\_(\mathtt{Int}, \bullet)\rangle), \mathtt{q}\colon \mathsf{Li}(\langle l_2\_([\mathtt{r}\colon \mathsf{Lo}(\langle l_3\_(\mathtt{Bool}, \bullet)\rangle), \mathtt{q}\colon \mathsf{Li}(\langle l_4\_(\mathtt{String}, \bullet)\rangle)], \bullet)\rangle)]\end{array}$$

Recursion, Continuations and Duality We now point out a subtle (but crucial) difference between Def. 5.1 and the encoding of *binary, non-recursive* session types in [14]. When encoding partial selections, our continuation type is the *dual of the encoding of* H_i, i.e., $\overline{\llbracket H_i \rrbracket}$; in [14], instead, it is the *encoding of the dual of* H_i, i.e., $\llbracket \overline{H_i} \rrbracket$. This difference is irrelevant for *non-recursive* types (Ex. 5.2); but for *recursive* types, using $\llbracket \overline{H_i} \rrbracket$ would yield the wrong continuations. Using $\overline{\llbracket H_i \rrbracket}$, instead, gives the expected result, by generating *dualised recursion variables* (cf. Def. 4.1). We explain it in Ex. 5.3 below.

▶ **Example 5.3.** *Let* $H = \mu \mathbf{t}.!l(\mathtt{Bool}).\mathbf{t}$*. By Def. 5.1, we have:*

$$\llbracket H \rrbracket \;=\; \llbracket \mu \mathbf{t}.!l(\mathtt{Bool}).\mathbf{t} \rrbracket = \mu \mathbf{t}.\mathsf{Lo}\Big(\langle l\_(\llbracket \mathtt{Bool} \rrbracket, \overline{\llbracket \mathbf{t} \rrbracket})\rangle\Big) = \mu \mathbf{t}.\mathsf{Lo}\big(\langle l\_(\mathtt{Bool}, \overline{\mathbf{t}})\rangle\big)$$

Let us now unfold the encoding of H*. By Def. 4.1, we have:*

$$\begin{array}{rcl}\mathrm{unf}(\llbracket H \rrbracket) & = & \mathrm{unf}\big(\mu \mathbf{t}.\mathsf{Lo}(\langle l\_(\mathtt{Bool}, \overline{\mathbf{t}})\rangle)\big) \;=\; \mathsf{Lo}\Big(\langle l\_(\mathtt{Bool}, \overline{\mu \mathbf{t}.\mathsf{Lo}\big(\langle l\_(\mathtt{Bool}, \overline{\mathbf{t}})\rangle\big)}\{\overline{\mathbf{t}}/\mathbf{t}\})\rangle\Big) \\ & = & \mathsf{Lo}\Big(\langle l\_\big(\mathtt{Bool},\ \mu \mathbf{t}.\mathsf{Li}(\langle l\_(\mathtt{Bool}, \mathbf{t})\rangle)\big)\rangle\Big)\end{array}$$

This is what we want: since H *requires a recursive output of* $\mathtt{Bool}$*eans, its encoding should output a* $\mathtt{Bool}$*ean, together with a* recursive input name *as continuation. Hence, the recipient will receive the first* $\mathtt{Bool}$*ean together with a continuation name, whose type mandates to recursively input more* $\mathtt{Bool}$*s. If encoding continuations as in [14], instead, we would have:*

$$\begin{array}{rcl}\llbracket H \rrbracket & = & \mu \mathbf{t}.\mathsf{Lo}(\langle l\_(\llbracket \mathtt{Bool} \rrbracket, \llbracket \overline{\mathbf{t}} \rrbracket)\rangle) = \mu \mathbf{t}.\mathsf{Lo}(\langle l\_(\mathtt{Bool}, \mathbf{t})\rangle) \;\; (\mathbf{t} \textit{ is } \text{not } \textit{dualised}) \\ \mathrm{unf}(\llbracket H \rrbracket) & = & \mathsf{Lo}(\langle l\_(\mathtt{Bool}, \mu \mathbf{t}.\mathsf{Lo}(\langle l\_(\mathtt{Bool}, \mathbf{t})\rangle))\rangle)\end{array}$$

which is wrong: the recipient is required to recursively output $\mathtt{Bool}$*s. This wrong encoding would also prevent us from obtaining Theorem 6.1 later on.*

Encoding of Typing Contexts In order to preserve type safety, we want to *encode a session judgement (Fig. 4) into a* π*-calculus typing judgement (Fig. 5).* For this reason, we now use the encoding of session types (Def. 5.1) to formalise the encoding of session typing contexts.

▶ **Definition 5.4.** *The* encoding of a session typing context *is:*

$$\begin{array}{cccc}\llbracket \varnothing \rrbracket \triangleq \varnothing & \llbracket \Theta \cdot \Gamma \rrbracket \triangleq \llbracket \Theta \rrbracket, \llbracket \Gamma \rrbracket & \llbracket c\!:\!U \rrbracket \triangleq \llbracket c \rrbracket : \llbracket U \rrbracket & \llbracket s[\mathtt{p}] \rrbracket \triangleq z_{s[\mathtt{p}]} \\ \Big\llbracket \Theta, X\!:\!\widetilde{U} \Big\rrbracket \triangleq \llbracket \Theta \rrbracket, \Big\llbracket X\!:\!\widetilde{U} \Big\rrbracket & \llbracket \Gamma, c\!:\!U \rrbracket \triangleq \llbracket \Gamma \rrbracket, \llbracket c\!:\!U \rrbracket & \llbracket x \rrbracket \triangleq x & \llbracket X \rrbracket \triangleq z_X \\ \llbracket \Gamma_1 \circ \Gamma_2 \rrbracket \triangleq \llbracket \Gamma_1 \rrbracket \uplus \llbracket \Gamma_2 \rrbracket & \multicolumn{3}{c}{\llbracket X\!:\!U_1, \ldots, U_n \rrbracket \triangleq \llbracket X \rrbracket : \sharp\big((\llbracket U_i \rrbracket)_{i \in 1..n}\big)}\end{array}$$

When encoding typing contexts, variables (x) keep their name, while process variables (X) and channels with roles ($s[\mathtt{p}]$) are turned into distinguished names with a subscript: e.g., X becomes z_X. The typing context composition $\Gamma_1 \circ \Gamma_2$ (Def. 2.11) is encoded using $\uplus$ (Def. 3.6): such an operation is always defined, since the domains of $[\![\Gamma_1]\!]$,$[\![\Gamma_2]\!]$ can only overlap on basic types. Note that encoded process variables have an *unrestricted* connection type, carrying an n-tuple of encoded argument types; encoded sessions, instead, are always linearly-typed.

Encoding Typing Judgements: Overview We can now have a first look at the encoding of session typing judgements in Fig. 7 (but we postpone the formal statement to Def. 5.7 later on, as it requires some more technical developments).

Terminated processes are encoded homomorphically. **Parallel composition** is also encoded homomorphically — i.e., our encoding preserves the choreographic distribution of the originating processes. Note that $[\![P]\!]_{\Theta \cdot \Gamma_1}$ and $[\![Q]\!]_{\Theta \cdot \Gamma_2}$ are the encoded processes yielded respectively by $[\![\Theta \cdot \Gamma_1 \vdash P]\!]$ and $[\![\Theta \cdot \Gamma_2 \vdash Q]\!]$: they exist because such typing judgements hold, by inversion of (T-Par) (Fig. 4). Similar uses of sub-processes encoded w.r.t. their typing occur in the other cases. **Process declaration** $\mathbf{def}\ X(\widetilde{x}{:}U) = P\ \mathbf{in}\ Q$ is encoded as a replicated π-calculus process that inputs a value z on a name $[\![X]\!] = z_X$ (matching Def. 5.4), deconstructs it into $x_1, \ldots, x_n$ (using **with**, and hence assuming that z is an n-tuple), and then continues as the encoding of the body P; meanwhile, the encoding of Q runs in parallel, enclosed by a delimitation on z_X (that matches the scope of the original declaration). Correspondingly, a **process call** $X\langle\widetilde{v}\rangle$ is encoded as a process that sends the encoded values $[\![\widetilde{v}]\!]$ on z_X and ends (in MPST π-calculus, process calls are in tail position).

Selection on $c[\mathtt{p}]$ is encoded using information from the session typing context: the fact that c has type $S = \mathtt{p} \oplus !l(U).S'$ — i.e., $[\![S]\!]$ is a record type with one entry $\mathtt{q}{:}z_{\mathtt{q}}$ for each $\mathtt{q} \in S$. Therefore, the encoding first deconstructs $[\![c]\!]$ (using **with**), an then uses the (linear) name in its $\mathtt{p}$-entry to output on $z_{\mathtt{p}}$. Before performing the output, however, a new name z is created: it is the *continuation* of the interaction with $\mathtt{p}$. Then, one endpoint of z is sent through $z_{\mathtt{p}}$ as part of $l([\![v]\!], z)$, which is a variant value carrying a tuple. The other endpoint of z is kept, and used to rebind $[\![c]\!]$ (using **let**) with a "new" record, consisting in *all* the entries of the "original" $[\![c]\!]$, *except* $z_{\mathtt{p}}$ (which has been used for output). More in detail, the "new" $[\![c]\!]$ has an entry for $\mathtt{p}$ (mapping $\mathtt{p}$ to z) iff S' still involves $\mathtt{p}$ (otherwise, if $\mathtt{p} \notin S'$, then z is discarded, since it has type $[\![S' \upharpoonright \mathtt{p}]\!] = [\![\mathbf{end}]\!] = \bullet$). After **let**, the encoding continues as $[\![P]\!]$.

Symmetrically, **branching** on $c[\mathtt{p}]$ is also encoded using information from the typing context, i.e., that c has type $S = \mathtt{p}\,\&_{i \in I}\, ?l_i(U_i).S'_i$ — and therefore, $[\![S]\!]$ is a record type with one entry $\mathtt{q}{:}z_{\mathtt{q}}$ for each $\mathtt{q} \in S$. As above, the encoded process deconstructs $[\![c]\!]$ (using **with**), an then uses the (linear) name in its $\mathtt{p}$-entry to perform an input $z_{\mathtt{p}}(y)$; y is assumed to be a variant, and is pattern matched to determine the continuation. If y matches l_i (for some $i \in I$), *and* it carries a tuple $z_i = (x_i, z)$ (where z is a continuation name), then $[\![c]\!]$ is rebound (using **let**) and the process continues as $[\![P_i]\!]$. The rebinding of $[\![c]\!]$ depends on l_i and the continuation type S'_i: the "new" $[\![c]\!]$ is a record with *all* the linear names of the "original" $[\![c]\!]$, *except* $z_{\mathtt{p}}$ (which has been used for input); as above, an entry for $\mathtt{p}$ will exist (and map $\mathtt{p}$ to z) iff S'_i still involves $\mathtt{p}$ (otherwise, if $\mathtt{p} \notin S'_i$, then z has type $\bullet$ and is discarded).

We will explain the encoding of **session restriction** $(\boldsymbol{\nu} s)P$ later, after Def. 5.7, as it requires some technicalities: namely, the substitution $\boldsymbol{\sigma}(\Gamma')$. We can, however, have an intuition about the role of $\boldsymbol{\sigma}(\Gamma')$ by considering an obvious discrepancy. Consider the following session π-calculus process, that reduces by communication (cf. Def. 2.3):

$$\Gamma, s[\mathtt{p}]{:}S, s[\mathtt{q}]{:}S' \vdash s[\mathtt{p}][\mathtt{q}]\&\{l(x).P\} \mid s[\mathtt{q}][\mathtt{p}] \oplus \langle l(v)\rangle.Q \quad \to \quad P\{^{v}/_{x}\} \mid Q \tag{2}$$

$$\llbracket \Gamma \vdash \mathbf{0} \rrbracket \triangleq \llbracket \Gamma \rrbracket \vdash \mathbf{0} \qquad \llbracket \Theta \cdot \Gamma_1 \circ \Gamma_2 \vdash P \mid Q \rrbracket \triangleq \llbracket \Theta \cdot \Gamma_1 \circ \Gamma_2 \rrbracket \vdash \llbracket P \rrbracket_{\Theta \cdot \Gamma_1} \mid \llbracket Q \rrbracket_{\Theta \cdot \Gamma_2}$$

$$\underbrace{\left\llbracket \Theta \cdot \Gamma \vdash \mathbf{def}\ X(\widetilde{x}{:}\widetilde{U}) = P\ \mathbf{in}\ Q \right\rrbracket}_{\text{where } \widetilde{x} = x_1, \ldots, x_n \text{ and } \widetilde{U} = U_1, \ldots, U_n} \triangleq \begin{array}{l} \llbracket \Theta \cdot \Gamma \rrbracket \vdash \\ (\boldsymbol{\nu} \llbracket X \rrbracket) \left(* \left(\llbracket X \rrbracket(z).\mathbf{with}\ (x_i)_{i \in \{1..n\}} = z\ \mathbf{do}\ \llbracket P \rrbracket_{\Theta, X : \widetilde{U} \cdot \widetilde{x} : \widetilde{U}} \right) \mid \llbracket Q \rrbracket_{\Theta, X : \widetilde{U} \cdot \Gamma} \right) \end{array}$$

$$\overbrace{\left\llbracket \Theta, X : \widetilde{U} \cdot \Gamma_1 \circ \ldots \circ \Gamma_n \circ \Gamma \vdash X \langle \widetilde{v} \rangle \right\rrbracket} \triangleq \left\llbracket \Theta, X : \widetilde{U} \cdot \Gamma_1 \circ \ldots \circ \Gamma_n \circ \Gamma \right\rrbracket \vdash \overline{\llbracket X \rrbracket} \langle (\llbracket v_i \rrbracket)_{i \in \{1..n\}} \rangle.\mathbf{0}$$

$$\underbrace{\llbracket \Theta \cdot c{:}S, \Gamma_1 \circ \Gamma_2 \vdash c[\mathtt{p}] \oplus \langle l(v) \rangle.P \rrbracket}_{\text{where } S = \mathtt{p} \oplus !l(U).S'} \triangleq \underbrace{\begin{array}{l} \llbracket \Theta, c{:}S, \Gamma_1 \circ \Gamma_2 \rrbracket \vdash \\ \mathbf{with}\ \left[\mathtt{q} : z_{\mathtt{q}}\right]_{\mathtt{q} \in S} = \llbracket c \rrbracket\ \mathbf{do}\ (\boldsymbol{\nu} z) \overline{z_{\mathtt{p}}} \langle l(\llbracket v \rrbracket, z) \rangle.\mathbf{let}\ \llbracket c \rrbracket = \maltese\ \mathbf{in}\ \llbracket P \rrbracket_{\Theta \cdot \Gamma_2, c : S'} \end{array}}_{\text{where } \maltese = \begin{cases} [\mathtt{p} : z,\ \mathtt{q} : z_{\mathtt{q}}]_{\mathtt{q} \in S' \setminus \mathtt{p}} & \text{if } \mathtt{p} \in S' \\ \left[\mathtt{q} : z_{\mathtt{q}}\right]_{\mathtt{q} \in S'} & \text{otherwise} \end{cases}}$$

$$\underbrace{\llbracket \Theta \cdot c{:}S, \Gamma \vdash c[\mathtt{p}]\ \&_{i \in I}\ \{l_i(x_i).P_i\} \rrbracket}_{\text{where } S = \mathtt{p}\ \&_{i \in I}\ ?l_i(U_i).S'_i} \triangleq \underbrace{\begin{array}{l} \llbracket \Theta, c{:}S, \Gamma \rrbracket \vdash \mathbf{with}\ \left[\mathtt{q} : z_{\mathtt{q}}\right]_{\mathtt{q} \in S} = \llbracket c \rrbracket\ \mathbf{do}\ z_{\mathtt{p}}(y).\mathbf{case}\ y\ \mathbf{of}\ \Big\{ \\ \qquad l_i(z_i) \triangleright \mathbf{with}\ (x_i, z) = z_i\ \mathbf{do}\ \mathbf{let}\ \llbracket c \rrbracket = \maltese_i\ \mathbf{in}\ \llbracket P_i \rrbracket_{\Theta \cdot \Gamma'} \Big\}_{i \in I} \end{array}}$$

$$\text{where } \Gamma' = \Gamma, x_i{:}U_i, c{:}S'_i \text{ and } \maltese_i = \begin{cases} [\mathtt{p} : z,\ \mathtt{q} : z_{\mathtt{q}}]_{\mathtt{q} \in S'_i \setminus \mathtt{p}} & \text{if } \mathtt{p} \in S'_i \\ \left[\mathtt{q} : z_{\mathtt{q}}\right]_{\mathtt{q} \in S'_i} & \text{otherwise} \end{cases}$$

$$\underbrace{\llbracket \Theta \cdot \Gamma \vdash (\boldsymbol{\nu} s{:}\Gamma')P \rrbracket}_{\text{where } \operatorname{conn}(s, \Gamma') = \{\{\mathtt{p}_1, \mathtt{q}_1\}, \ldots, \{\mathtt{p}_n, \mathtt{q}_n\}\}} \triangleq \llbracket \Theta \cdot \Gamma \rrbracket \vdash \underbrace{\llbracket (\boldsymbol{\nu} s) \rrbracket \llbracket P \rrbracket_{\Theta \cdot \Gamma, \Gamma'} \boldsymbol{\sigma}(\Gamma')}_{\text{where } \llbracket (\boldsymbol{\nu} s) \rrbracket = (\boldsymbol{\nu} z_{\{s, \mathtt{p}_i, \mathtt{q}_i\}})_{i \in \{1..n\}}}$$

Figure 7 Encoding of typing judgements. Here, $\llbracket P \rrbracket_{\Theta \cdot \Gamma} = Q$ iff $\llbracket \Theta \cdot \Gamma \vdash P \rrbracket = \llbracket \Theta \cdot \Gamma \rrbracket \vdash Q$.

We would like its encoding to reduce and communicate, too — but it is *not* the case:

$$\mathbf{with}\ [\mathtt{r} : z_{\mathtt{r}}]_{\mathtt{r} \in S} = \llbracket s[\mathtt{p}] \rrbracket\ \mathbf{do} \ldots \mid \mathbf{with}\ [\mathtt{r} : z_{\mathtt{r}}]_{\mathtt{r} \in S'} = \llbracket s[\mathtt{q}] \rrbracket\ \mathbf{do} \ldots \ \not\to \tag{3}$$

and the reason is that $\llbracket s[\mathtt{p}] \rrbracket, \llbracket s[\mathtt{q}] \rrbracket$ are "just" record-typed *names* (respectively $z_{s[\mathtt{p}]}, z_{s[\mathtt{q}]}$, as per Def. 5.4), whereas **with**-prefixes only reduce when applied to *record values* (cf. Def. 3.2). Hence, to let our encoded terms reduce, we must first substitute $\llbracket s[\mathtt{p}] \rrbracket, \llbracket s[\mathtt{q}] \rrbracket$ with two records; moreover, to let the two encoded processes synchronise and exchange $\llbracket v \rrbracket$, such records must be suitably defined: we must ensure that the entries for $\mathtt{q}$ (in one record) and $\mathtt{p}$ (in the other) map to *the same (linear) name.* In the following, we show how $\boldsymbol{\sigma}(\Gamma')$ handles this issue.

Reification of Multiparty Sessions By simply translating a channel with role $s[\mathtt{p}]$ into a π-calculus name $z_{s[\mathtt{p}]}$, we have not yet captured the insight behind our approach, i.e., the idea that a multiparty session can be decomposed into a labelled tuple of linear channels (i.e., π-calculus names), connecting *pairs of roles.* We can formalise "connections" as follows.

▶ **Definition 5.5.** *The* connections of s in Γ *are:* $\operatorname{conn}(s, \Gamma) \triangleq \big\{ \{\mathtt{p}, \mathtt{q}\} \ \big|\ s[\mathtt{p}] : S_{\mathtt{p}} \in \Gamma \wedge \mathtt{q} \in S_{\mathtt{p}} \big\}$

Intuitively, two roles $\mathtt{p}, \mathtt{q}$ are connected by s in Γ if $\mathtt{p}$ occurs in the type $\Gamma(s[\mathtt{q}])$ (but $\mathtt{q}$ *might not* occur in $\Gamma(s[\mathtt{p}])$; note, however, that $\mathtt{q}$ will always occur if Γ is consistent).

Now, as anticipated above, we want to substitute each $\llbracket s[\mathtt{p}] \rrbracket$ with a suitably defined record, composed by π-calculus names; correspondingly, such names must be typed, i.e., appear in the typing context: this is addressed in Def. 5.6.

▶ **Definition 5.6** (Reification and decomposition of MPST contexts)**.** *The* reification of a session typing context Γ_{S} *is the substitution:*

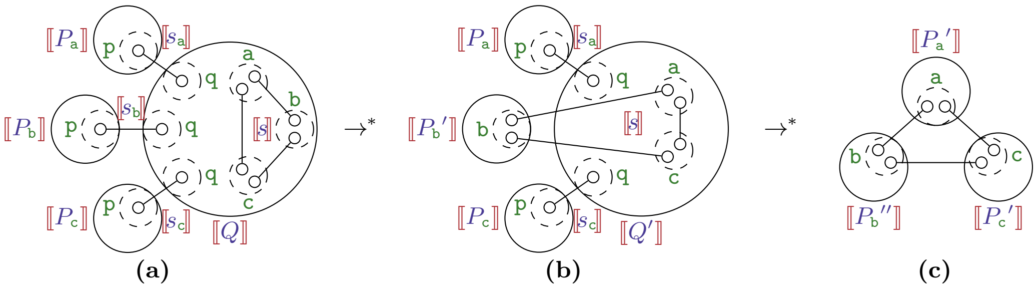


Figure 8 Multiparty peer-to-peer game: encoded version of Fig. 2. Lines are binary channels.

$$\boldsymbol{\sigma}(\Gamma_{\mathsf{S}}) \;=\; \left\{ [\mathsf{q}\!:\!z_{\{s,\mathsf{p},\mathsf{q}\}}]_{\mathsf{q}\in S_{\mathsf{p}}} / [\![s[\mathsf{p}]]\!] \right\}_{s[\mathsf{p}]:S_{\mathsf{p}}\in\Gamma_{\mathsf{S}}}$$

The linear decomposition of Γ_{S} *is the* π*-calculus typing context* $\boldsymbol{\delta}(\Gamma_{\mathsf{S}})$*, defined as:*

$$\boldsymbol{\delta}(\Gamma_{\mathsf{S}}) \;=\; \biguplus_{s[\mathsf{p}]:S_{\mathsf{p}}\in\Gamma_{\mathsf{S}}} \left\{ z_{\{s,\mathsf{p},\mathsf{q}\}} : [\![\mathrm{unf}(S_{\mathsf{p}} \restriction \mathsf{q})]\!] \right\}_{\{\mathsf{p},\mathsf{q}\}\in\mathrm{conn}(s,\Gamma_{\mathsf{S}})}$$

The π*-calculus* reification typing rule *is (note that* $\Gamma_{\mathsf{S}}, \Gamma'_{\mathsf{S}}$ *are* MPST *typing contexts):*

$$\frac{[\![\Theta \cdot \Gamma_{\mathsf{S}}]\!], [\![\Gamma'_{\mathsf{S}}]\!] \vdash P}{[\![\Theta \cdot \Gamma_{\mathsf{S}}]\!], \boldsymbol{\delta}(\Gamma'_{\mathsf{S}}) \vdash P\boldsymbol{\sigma}(\Gamma'_{\mathsf{S}})}\;(\text{T}\pi\text{-Reify})$$

The simplest part of Def. 5.6 is $\boldsymbol{\sigma}(\Gamma_{\mathsf{S}})$: it is a substitution that, for each $s[\mathsf{p}]\!:\!S_{\mathsf{p}} \in \Gamma_{\mathsf{S}}$, replaces $[\![s[\mathsf{p}]]\!]$ with a record containing one entry $\mathsf{q}\!:\!z_{\{s,\mathsf{p},\mathsf{q}\}}$ for each $\mathsf{q}\!\in\!S_{\mathsf{p}}$. Note that if there is also some $s[\mathsf{q}]\!:\!S_{\mathsf{q}} \in \Gamma_{\mathsf{S}}$ with $\mathsf{p}\!\in\!S_{\mathsf{q}}$, then the corresponding record (that replaces $[\![s[\mathsf{q}]]\!]$) has an entry $\mathsf{p}\!:\!z_{\{s,\mathsf{q},\mathsf{p}\}} = z_{\{s,\mathsf{p},\mathsf{q}\}}$ — i.e., p (in one record) and q (in the other) *map to the same name*: this realises the intuition of "multiparty sessions as interconnected binary channels".

The definition of $\boldsymbol{\sigma}(\Gamma_{\mathsf{S}})$ was the missing ingredient to formalise our encoding, presented in Def. 5.7 below. The rest of Def. 5.6 will be used to prove its correctness (Theorem 6.3): hence, we postpone its explanation to page 21.

▶ **Definition 5.7** (Encoding). *The* encoding of session typing judgements *is given in Fig. 7. We define* $[\![P]\!]_{\Theta\cdot\Gamma} = Q$ *iff* $[\![\Theta \cdot \Gamma \vdash P]\!] = [\![\Theta \cdot \Gamma]\!] \vdash Q$*. Sometimes, we write* $[\![P]\!]$ *for* $[\![P]\!]_{\Theta\cdot\Gamma}$ *when* Θ, Γ *are empty, or clear from the context.*

We conclude by explaining the last case in Fig. 7, which was not addressed on p.18. The process $(\boldsymbol{\nu} s\!:\!\Gamma')P$ is encoded by generating one delimitation for each $z_{\{s,\mathsf{p}_i,\mathsf{q}_i\}}$ whenever $\{\mathsf{p}_i, \mathsf{q}_i\}$ is a connection of s in Γ' (Def. 5.5). Then, P is encoded, and the substitution $\boldsymbol{\sigma}(\Gamma')$ is applied: it replaces each $[\![s[\mathsf{p}_i]]\!]$, $[\![s[\mathsf{q}_i]]\!]$ in $[\![P]\!]$ with records based on the delimited $z_{\{s,\mathsf{p}_i,\mathsf{q}_i\}}$.

▶ **Example 5.8.** *Consider* (2)*. If we delimit* s *and encode the resulting process, we obtain a* π*-calculus process based on* (3)*, enclosed by the delimitations yielded by* $[\![(\boldsymbol{\nu} s)]\!]$*, and the substitution* $\boldsymbol{\sigma}(s[\mathsf{p}]\!:\!S, s[\mathsf{q}]\!:\!S', \ldots)$*. Since the latter replaces* $[\![s[\mathsf{p}]]\!]$*,* $[\![s[\mathsf{q}]]\!]$ *with records whose entries reflect* roles(S) *and* roles(S')*, the encoding can now reduce, firing the two* **with***s.*

▶ **Example 5.9.** *Consider the main server/clients parallel composition in Ex. 2.2:*

$$(\boldsymbol{\nu} s_{\mathsf{a}}, s_{\mathsf{b}}, s_{\mathsf{c}})\big(Q \mid P_{\mathsf{a}} \mid P_{\mathsf{b}} \mid P_{\mathsf{c}}\big) \quad \textit{where}$$
$$Q \;=\; (\boldsymbol{\nu} s)\Big(s_{\mathsf{a}}[\mathsf{q}][\mathsf{p}]\oplus\langle\texttt{PlayA}(s[\mathsf{a}])\rangle \mid s_{\mathsf{b}}[\mathsf{q}][\mathsf{p}]\oplus\langle\texttt{PlayB}(s[\mathsf{b}])\rangle \mid s_{\mathsf{c}}[\mathsf{q}][\mathsf{p}]\oplus\langle\texttt{PlayC}(s[\mathsf{c}])\rangle\Big)$$

Its encoding is the following process, with s *decomposed in 3 linear channels (see also Fig. 8):*

$$(\boldsymbol{\nu} z_{\{s_{\mathsf{a}},\mathsf{p},\mathsf{q}\}}, z_{\{s_{\mathsf{b}},\mathsf{p},\mathsf{q}\}}, z_{\{s_{\mathsf{c}},\mathsf{p},\mathsf{q}\}})\big([\![Q]\!] \mid [\![P_{\mathsf{a}}]\!] \mid [\![P_{\mathsf{b}}]\!] \mid [\![P_{\mathsf{c}}]\!]\big) \quad \textit{where}$$
$$[\![Q]\!] \;=\; (\boldsymbol{\nu} z_{\{s,\mathsf{a},\mathsf{b}\}}, z_{\{s,\mathsf{b},\mathsf{c}\}}, z_{\{s,\mathsf{a},\mathsf{c}\}})\Big([\![s_{\mathsf{a}}[\mathsf{q}][\mathsf{p}]\oplus\langle\texttt{PlayA}(s[\mathsf{a}])\rangle]\!] \mid [\![s_{\mathsf{b}}[\mathsf{q}][\mathsf{p}]\oplus\langle\texttt{PlayB}(s[\mathsf{b}])\rangle]\!] \mid [\![s_{\mathsf{c}}[\mathsf{q}][\mathsf{p}]\oplus\langle\texttt{PlayC}(s[\mathsf{c}])\rangle]\!]\Big)$$

6 Properties of the Encoding

In this section we present some crucial properties ensuring the correctness of our encoding.

Encoding of Types Theorem 6.1 below says that our encoding commutes the duality between session types (Def. 2.8) and π-types (Def. 4.1); Theorem 6.2 shows that it also preserves subtyping.

▶ **Theorem 6.1** (Encoding preserves duality). $[\![\overline{H}]\!] = \overline{[\![H]\!]}$.

▶ **Theorem 6.2** (Encoding preserves subtyping). *If* $S \leqslant_{\mathsf{S}} S'$*, then* $[\![S]\!] \leqslant_{\pi} [\![S']\!]$.

Encoding of Typing Judgements Theorem 6.3 shows that the encoding of session typing judgements into π-calculus typing judgements is valid. As a consequence, a well-typed MPST process also enjoys the type safety guarantees that can be expressed in standard π-calculus.

▶ **Theorem 6.3** (Correctness of encoding). $\Gamma \vdash v : U$ *implies* $[\![\Gamma]\!] \vdash [\![v]\!] : [\![U]\!]$*,* $\Theta \vdash X : \widetilde{U}$ *implies* $[\![\Theta]\!] \vdash [\![X]\!] : \widetilde{[\![U]\!]}$*, and* $\Theta \cdot \Gamma \vdash P$ *implies* $[\![\Theta \cdot \Gamma \vdash P]\!]$.

The proof is by induction on the MPST typing derivation, which yields a corresponding π-calculus typing derivation. One simple case is the following, that relates subtyping:

$$\text{(T-Sub)}\ \frac{\Theta \cdot \Gamma, c{:}S \vdash P \quad S' \leqslant_{\mathsf{S}} S}{\Theta \cdot \Gamma, c{:}S' \vdash P} \quad \text{implies} \quad \frac{[\![\Theta \cdot \Gamma, c{:}S \vdash P]\!] \quad [\![S']\!] \leqslant_{\pi} [\![S]\!]}{[\![\Theta \cdot \Gamma, c{:}S']\!] \vdash [\![P]\!]_{\Theta \cdot \Gamma, c:S}}\ \begin{array}{c}(\text{T}\pi\text{-Narrow})\\ (\text{Fig. 6})\end{array}$$

that holds by the induction hypothesis and Theorem 6.2. The most delicate case is the encoding of session restriction $\Theta \cdot \Gamma \vdash (\boldsymbol{\nu} s{:}\Gamma')P$ (Fig. 7): its encoding turns $(\boldsymbol{\nu} s)$ into a set of delimited names, used in the substitution $\boldsymbol{\sigma}(\Gamma')$ applied to $[\![P]\!]_{\Theta \cdot \Gamma, \Gamma'}$; hence, to prove the theorem, we need to type such names, i.e., find a context that types $[\![P]\!]_{\Theta \cdot \Gamma, \Gamma'} \boldsymbol{\sigma}(\Gamma')$. This is where $\boldsymbol{\delta}(\Gamma')$ and (Tπ-Reify) (Def. 5.6) come into play, as we now explain.

More on Def. 5.6 and decomposition By Def. 5.6, The typing context $\boldsymbol{\delta}(\Gamma_{\mathsf{S}})$, *when defined*, has an entry $z_{\{s,\mathsf{p},\mathsf{q}\}}$ for each $s[\mathsf{p}] : S_{\mathsf{p}} \in \Gamma_{\mathsf{S}}$ and $\mathsf{q} \in S_{\mathsf{p}}$. Such entries are used to type the records yielded by $\boldsymbol{\sigma}(\Gamma_{\mathsf{S}})$. The type of $z_{\{s,\mathsf{p},\mathsf{q}\}}$ is based on the encoding of the unfolded partial projection $S_{\mathsf{p}} \restriction \mathsf{q}$, that can can be either $\bullet$, or $\mathsf{Li}(T)/\mathsf{Lo}(T)$ (for some T). Note that if there is also some $s[\mathsf{q}] : S_{\mathsf{q}} \in \Gamma_{\mathsf{S}}$ with $\mathsf{q} \neq \mathsf{p}$, the type of $z_{\{s,\mathsf{q},\mathsf{p}\}} = z_{\{s,\mathsf{p},\mathsf{q}\}}$ *(when defined)* is $[\![\mathrm{unf}(S_{\mathsf{p}} \restriction \mathsf{q})]\!] \uplus [\![\mathrm{unf}(S_{\mathsf{q}} \restriction \mathsf{p})]\!]$. This creates a deep correspondence between the consistency of Γ_{S} and the existence of $\boldsymbol{\delta}(\Gamma_{\mathsf{S}})$, as shown in Theorem 6.4 below: it says that *the precondition for type safety in MPSTs (i.e., the consistency of* Γ_{S}*) can be precisely expressed in* π*-calculus*, and *this is captured by the linear decomposition at the roots of our encoding.*

▶ **Theorem 6.4** (Precise decomposition). Γ_{S} *is consistent* if and only if $\boldsymbol{\delta}(\Gamma_{\mathsf{S}})$ *is defined.*

The final part of Def. 5.6 is the π-calculus typing rule (Tπ-Reify), that uses $\boldsymbol{\delta}(\Gamma'_{\mathsf{S}})$ to type a process on which $\boldsymbol{\sigma}(\Gamma'_{\mathsf{S}})$ has been applied. We explain the rule with a slight simplification. If we have $\Gamma'_{\mathsf{S}} = \left\{ s[\mathsf{p}] : S_{\mathsf{p}} \right\}_{\mathsf{p} \in I}$, then:

$$\boldsymbol{\delta}(\Gamma'_{\mathsf{S}}) \;=\; \biguplus_{\mathsf{p} \in I} \left\{ z_{\{s,\mathsf{p},\mathsf{q}\}} : [\![\mathrm{unf}(S_{\mathsf{p}} \restriction \mathsf{q})]\!] \right\}_{\{\mathsf{p},\mathsf{q}\} \in \mathrm{conn}(s,\Gamma_{\mathsf{S}})} \qquad \boldsymbol{\sigma}(\Gamma'_{\mathsf{S}}) \;=\; \left\{ {}^{\left[\mathsf{q} : z_{\{s,\mathsf{p},\mathsf{q}\}}\right]_{\mathsf{q} \in S_{\mathsf{p}}}} / {}_{[\![s[\mathsf{p}]]\!]} \right\}_{\mathsf{p} \in I}$$

(Note: $\boldsymbol{\delta}(\Gamma'_{\mathsf{S}})$ is defined *iff* Γ'_{S} is consistent, by Theorem 6.4). Now, take a set of types $\left\{ T_{(s,\mathsf{p},\mathsf{q})} \right\}_{\{\mathsf{p},\mathsf{q}\} \in \mathrm{conn}(s,\Gamma_{\mathsf{S}})}$ such that $\biguplus_{\mathsf{p} \in I} \left\{ z_{\{s,\mathsf{p},\mathsf{q}\}} : T_{(s,\mathsf{p},\mathsf{q})} \right\}_{\{\mathsf{p},\mathsf{q}\} \in \mathrm{conn}(s,\Gamma_{\mathsf{S}})} = \boldsymbol{\delta}(\Gamma'_{\mathsf{S}})$ (note $T_{(s,\mathsf{p},\mathsf{q})}$, $T_{(s,\mathsf{q},\mathsf{p})}$ are distinct) and assume the premise of (Tπ-Reify). The following derivation holds:

$$\dfrac{\left\{\forall \mathsf{q} \in S_{\mathsf{p}} \quad \dfrac{\dfrac{}{z_{\{s,\mathsf{p},\mathsf{q}\}}:T_{(s,\mathsf{p},\mathsf{q})} \vdash z_{\{s,\mathsf{p},\mathsf{q}\}}:T_{(s,\mathsf{p},\mathsf{q})}}\,(\text{T}\pi\text{-Name}) \qquad T_{(s,\mathsf{p},\mathsf{q})} \leqslant_\pi [\![S_{\mathsf{p}} \upharpoonright \mathsf{q}]\!]}{\dfrac{z_{\{s,\mathsf{p},\mathsf{q}\}}:T_{(s,\mathsf{p},\mathsf{q})} \vdash z_{\{s,\mathsf{p},\mathsf{q}\}}:[\![S_{\mathsf{p}} \upharpoonright \mathsf{q}]\!]}{\left\{z_{\{s,\mathsf{p},\mathsf{q}\}}:[\![S_{\mathsf{p}} \upharpoonright \mathsf{q}]\!]\right\}_{\mathsf{q}\in S_{\mathsf{p}}} \vdash \left[\mathsf{q}:z_{\{s,\mathsf{p},\mathsf{q}\}}\right]_{\mathsf{q}\in S_{\mathsf{p}}}}\,(\text{T}\pi\text{-Rec})}\,(\text{T}\pi\text{-Sub})\right\}_{\mathsf{p}\in I} \quad [\![\Theta\cdot\Gamma_{\mathsf{S}}]\!],[\![\Gamma'_{\mathsf{S}}]\!] \vdash P}{[\![\Theta\cdot\Gamma_{\mathsf{S}}]\!],\boldsymbol{\delta}(\Gamma'_{\mathsf{S}}) = [\![\Theta\cdot\Gamma]\!] \uplus \boldsymbol{\delta}(\Gamma'_{\mathsf{S}}) \vdash P\boldsymbol{\sigma}(\Gamma'_{\mathsf{S}})}\,(\text{T}\pi\text{-MSubst - Fig. 6})$$

In particular, the assumptions $T_{(s,\mathsf{p},\mathsf{q})} \leqslant_\pi [\![S_{\mathsf{p}} \upharpoonright \mathsf{q}]\!]$ hold by Lemma 4.7, since each $T_{(s,\mathsf{p},\mathsf{q})}$ has been obtained by splitting $\boldsymbol{\delta}(\Gamma'_{\mathsf{S}})$ (that combines types with $\pitchfork$) using $\uplus$. The equivalence in the conclusion holds since $\operatorname{dom}([\![\Theta\cdot\Gamma_{\mathsf{S}}]\!]) \cap \operatorname{dom}(\boldsymbol{\delta}(\Gamma'_{\mathsf{S}})) = \emptyset$. Summing up: if the (Tπ-Reify) premise holds, then the above derivation holds, which proves the conclusion of (Tπ-Reify).

Now, we can finish the proof of Theorem 6.3 for the case $\Theta\cdot\Gamma \vdash (\boldsymbol{\nu} s\!:\!\Gamma')P$. Assuming that the judgement holds, we also have $\Theta\cdot\Gamma,\Gamma' \vdash P$ and Γ' complete (by the premise of (T-Res), Fig. 4): hence, Γ' is consistent, and $\boldsymbol{\delta}(\Gamma')$ is defined (by Theorem 6.4). Assuming that $[\![\Theta\cdot\Gamma,\Gamma' \vdash P]\!]$ holds (by the induction hypothesis), we obtain:

$$\dfrac{[\![\Theta\cdot\Gamma]\!],[\![\Gamma']\!] \vdash [\![P]\!]_{\Theta\cdot\Gamma,\Gamma'}}{[\![\Theta\cdot\Gamma]\!],\boldsymbol{\delta}(\Gamma') \vdash [\![P]\!]_{\Theta\cdot\Gamma,\Gamma'}\boldsymbol{\sigma}(\Gamma')}\,(\text{T}\pi\text{-Reify})$$

where $\boldsymbol{\delta}(\Gamma')$ types all the (delimited) names $z_{\{s,\mathsf{p},\mathsf{q}\}}$ given by $[\![(\boldsymbol{\nu} s)]\!]$. We can now conclude by applying (Tπ-Res1) to delimit such names (cf. Fig. 5 — note that this is allowed by the completeness of Γ'): we get $[\![\Theta\cdot\Gamma]\!] \vdash [\![(\boldsymbol{\nu} s)]\!][\![P]\!]_{\Theta\cdot\Gamma,\Gamma'}\boldsymbol{\sigma}(\Gamma')$, i.e., we match Fig. 7.

Finally, notice (from Fig. 7) that our encoding of processes uses some typing information. In principle, a process could be typed by applying the rules in multiple ways (especially (T-Sub) in Fig. 4), and one might wonder whether an MPST process could have multiple encodings. Proposition 6.5 says that this is *not* the case: the reason is that the only typing information being used is the set of roles in each session type, which does not depend on the typing rule — and is constant w.r.t. subtyping (i.e., $S \leqslant_{\mathsf{S}} S'$ implies $\operatorname{roles}(S) = \operatorname{roles}(S')$).

▶ **Proposition 6.5.** *If* $\Theta\cdot\Gamma \vdash P$ *and* $\Theta'\cdot\Gamma' \vdash P$, *then* $[\![P]\!]_{\Theta\cdot\Gamma} = [\![P]\!]_{\Theta'\cdot\Gamma'}$.

Encoding and Reduction One usual way to assess that an encoding is "behaviourally correct" (i.e., a process and its encoding behave "in the same way") consists in proving *operational correspondence*. Roughly, it says that the encoding is: *(1) complete*, i.e., any reduction of the original process is simulated by its encoding; and *(2) sound*, i.e., any reduction of the encoded process matches some reduction of the original process. This is formalised in Theorem 6.6, where $\xrightarrow{\textbf{with}}$ denotes a reduction induced by (Rπ-With) (Def. 3.2).

▶ **Theorem 6.6** (Operational correspondence)**.** *If* $\varnothing\cdot\varnothing \vdash P$, *then:*

1. (Completeness) $P \to^* P'$ *implies* $\exists \widetilde{x}, P''$ *such that* $[\![P]\!] \to^* (\boldsymbol{\nu}\widetilde{x})P''$ *and* $P'' = [\![P']\!]$;
2. (Soundness) $[\![P]\!] \to^* P_*$ *implies* $\exists \widetilde{x}, P'', P'$ *s.t.* $P_* \to^* (\boldsymbol{\nu}\widetilde{x})P''$, $P \to^* P'$ *and* $[\![P']\!] \xrightarrow{\textbf{with}}{}^* P''$.

The statement of Theorem 6.6 is standard [22, §5.1.3]. Item 1 says that if P reduces to P', then the encoding of the former can reduce to the encoding of the latter. Item 2 says (roughly) that no matter how the encoding of P reduces, it can always further reduce to the encoding of some P', such that P reduces to P'. Note that when we write $[\![P']\!]$, we mean $[\![P']\!]_{\varnothing\cdot\varnothing}$, which implies $\varnothing\cdot\varnothing \vdash P'$ (cf. Def. 5.7). The restricted variables $\widetilde{x}$ in items 1-2 are generated by the encoding of selection (Fig. 7): it creates a (delimited) linear name to continue the session. To see why item 2 uses $\xrightarrow{\textbf{with}}{}^*$, consider the following MPST process:

$$\varnothing\cdot\Gamma, s[\mathsf{p}]\!:\!S \vdash s[\mathsf{p}][\mathsf{q}]\&\{l(x).P\} \quad \not\to \qquad \text{(the process is stuck)}$$

If we encode it (and apply $\boldsymbol{\sigma}(\Gamma, s[\mathsf{p}]\!:\!S)$ as per Ex. 5.8), we get a π-calculus process that gets stuck, too — but *only after firing one internal* **with***-reduction*:

$$\textbf{with}\,[\mathtt{r}:z_{\mathtt{r}}]_{\mathtt{r}\in S} = [\mathtt{r}:z_{\{s,\mathtt{p},\mathtt{r}\}}]_{\mathtt{r}\in S}\,\textbf{do}\,z_{\mathtt{q}}(y).\dots \xrightarrow{\text{with}} z_{\{s,\mathtt{p},\mathtt{q}\}}(y).\dots \not\rightarrow$$

This happens whenever a process is deadlocked, and even if it is closed (as in item 2 of Theorem 6.6). This is because in Fig. 7, the "atomic" branch/select operations of MPSTs are encoded with multiple steps in linear π-calculus: first **with** for deconstructing the tuple of linear channels, and then input/output. In general, if an MPST process is stuck, its encoding fires *one* **with** for each stuck branch/select, then blocks on a corresponding input/output.

Theorem 6.6 yields an immediate corollary pertaining deadlock freedom:

▶ **Corollary 6.7.** *P is deadlock-free* if and only if *$[\![P]\!]$ is deadlock-free, i.e.: $[\![P]\!] \to^* P' \not\rightarrow$ implies $\exists Q \equiv \mathbf{0}$ such that $P' = [\![Q]\!]$.*

As a consequence, our encoding allows to transfer Theorem 2.19 to π-calculus processes.

▶ **Corollary 6.8.** *Let $\varnothing \cdot \varnothing \vdash P$, where $P \equiv (\boldsymbol{\nu} s : G)\big|_{i \in I} P_i$ and each P_i* only interacts on *$s[\mathtt{p_i}]$. Then, $[\![P]\!]$ is* deadlock-free*.*

7 From Theory to Implementation

We can now show how our encoding directly guides the implementation of a toolchain for generating safe multiparty session APIs in Scala, including *distributed delegation.* We continue our Game example from § 1, focusing on player `b`: we sketch the API generation and an implementation of a client, following the results in § 6. Our approach is to: (1) exploit *type safety and distribution* provided by an existing library for *binary* session channels, and then (2) treat the *ordering* of communications *across separate channels* in the API generation.

Scala and `lchannels` Our Scala toolchain is built upon the `lchannels` library [56]. `lchannels` provides two key classes, `Out[T]` and `In[T]`, whose instances must be used *linearly* (i.e., *once*) to send/receive (by method calls) a `T`-typed message: i.e., they represent channel endpoints with π-calculus types $\mathsf{Lo}(T)$ and $\mathsf{Li}(T)$ (Def. 3.3). This approach enforces the typing of I/O actions via *static* Scala typing; the *linear usage of channels*, instead, goes beyond the capabilities of the Scala typing system, and is therefore enforced with *run-time* checks.

`lchannels` delivers messages by abstracting over various transports: local memory, TCP sockets, Akka actors [39]. Notably, `lchannels` promotes session type-safety through a *continuation-passing-style* encoding of *binary* session types [56] that is close to our encoding of partial projections (formalised in Def. 5.1). Further, `lchannels` allows to send/receive `In[T]`/`Out[T]` instances for *binary session delegation* [56, Ex. 4.3]; on *distributed* message transports, instances of `In[T]`/`Out[T]` can be sent remotely (e.g., via the Akka-based transport).

Type-safe, distributed multiparty delegation By Theorem 6.3 and Def. 5.1 and Theorem 6.4, we know that the game player session type $S_{\mathtt{b}}$ in our example (see (1)) provides the type safety guarantees of a tuple of (linear) channels, whose types are given by the encoded partial projections of $S_{\mathtt{b}}$ onto `a`,`c` (Def. 2.9). This suggests that, using `lchannels`, the delegation of an $S_{\mathtt{b}}$-typed channel (as in § 1) could be rendered in Scala as:

`In[PlayB]` with definitions:
```
case class PlayB(payload: Sb)
case class Sb(a: In[InfoAB],c: Out[InfoBC])
```

i.e., as a linear input channel carrying a message of type `PlayB`, whose `payload` has type `Sb`; `Sb`, in turn, is a Scala `case class`, which can be seen as a labelled tuple, that maps `a`,`c` to I/O channels—whose types derive from $[\![S_{\mathtt{b}}\!\upharpoonright\!\mathtt{a}]\!]$ and $[\![S_{\mathtt{b}}\!\upharpoonright\!\mathtt{c}]\!]$ (in fact, they carry messages of type `InfoAB`/`InfoBC`). In this view, `Sb` is our Scala rendering of the encoded session type $[\![S_{\mathtt{b}}]\!]$. As said above, `lchannels` allows to send channels remotely—hence, also allows to remotely send *tuples* of channels (e.g., instances of `Sb`); thus, with this simple approach, we obtain *type-safe distributed multiparty delegation* of an $[\![S_{\mathtt{b}}]\!]$-typed channel tuple "for free".

Multiparty API Generation Corresponding to the π-calculus labelled tuple type yielded by the *type* encoding $[\![S_b]\!]$, the $\mathtt{S_b}$ class outlined above can ensure communication safety, i.e., no unexpected message will be sent or received on any of its binary channels. Like $[\![S_b]\!]$, however, $\mathtt{S_b}$, so far, does not convey *ordering* of communications *across* channels, i.e., the order in which its fields, `a` and `c`, should be used. (Indeed, $[\![S_b]\!]$ may type π-processes using its separate channels in *any* order while preserving basic safety.) To recover the "desired" ordering of communications, and implement it *correctly*, we can refine our classes so that:

(1) each multiparty channel class (e.g., $\mathtt{S_b}$) exposes a `send()` or `receive()` method, according to the I/O action expected by the multiparty *type* (S_b);
(2) the implementation of such method uses the binary channels as per our *process encoding*.

E.g., consider again S_b and $\mathtt{S_b}$. S_b requires to *send* towards `c`, so $\mathtt{S_b}$ could provide the API:

```
case class Sb(a: In[InfoAB], c: Out[InfoBC]) {
  def send(v: String) = {   // v is the payload of InfoBC message
    val c′ = c !! InfoBC(v)_ // lchannels method: send v, and return continuation
    S′b(a, c′) } }           // return a "continuation object"
```

Now, $\mathtt{S_b}$`.send()` behaves *exactly* as our process encoding in Fig. 7 (case for selection $\oplus$): it picks the correct channel from the tuple (in this case, `c`), creates a new tuple $\mathtt{s'_b}$ where `c` maps to a continuation channel, and returns it — so that the caller can use it to continue the multiparty session interaction. The class $\mathtt{s'_b}$ should be similar, with a `receive()` method that uses `a` for input (by following the encoding of $\&$). This way, a programmer is correctly led to write, e.g., `val x = s.send(...).receive()` (using method call chaining)—whereas attempting, e.g., `s.receive()` is rejected by the Scala compiler (method undefined). These `send()`/`receive()` APIs are mechanical, and can be automatically generated: we did it by extending Scribble.

Scribble-Scala Toolchain Scribble is a practical MPST-based language and tool for describing global protocols [57, 62]. To implement our results, we have extended Scribble (both the language and the tool) to support the full MPST theory in § 2, including, e.g., projection, type merging and delegation (not previously supported). Our extension allows protocols with the syntax in Fig. 9 (left), by augmenting Scribble with a *projection operator* `@`; then, it computes the projections/encodings explained in § 5, and automates the Scala API generation as outlined above (producing, e.g., the $\mathtt{S_b}$, $\mathtt{S'_b}$,... classes and their `send`/`receive` methods). This approach reminds the Java API generation in [27] — but we follow a formal foundation and target the type-safe binary channels provided by **`lchannels`** (that, as shown above, takes care of most irksome aspects — e.g., delegation). As a result, the P_b client in Fig. 1 can be written as in Fig. 9 (right); and although conceptually programmed as Fig. 2, the networking mechanisms of the game will concretely follow Fig. 8.

Our implementation is Open Source, and is available in [55].

8 Conclusion and Related Works

We presented the *first* encoding of a full-fledged multiparty session π-calculus into standard π-calculus (§ 5), and used it as the foundation of the *first* implementation of multiparty sessions (based on Scala API generation) with support for *distributed multiparty delegation*, on top of existing libraries (§ 7). We proved that a *consistent* session typing context is characterised by a *decomposition* into linear π-calculus types (Theorem 6.4): i.e., the type safety property of MPSTs is precisely captured by standard π-calculus. We encode types by preserving duality and subtyping (Theorems 6.1 and 6.2); our encoding of processes is type-preserving, and operationally sound and complete (Theorems 6.3 and 6.6); hence, our encoding preserves the type-safety and deadlock-freedom properties of MPST (Cor. 6.8). These results ensure the

```
global protocol ClientA(role p, role q) {
  PlayA(Game@a) from q to p; } // Delegation payload
global protocol ClientB(role p, role q) {
  PlayB(Game@b) from q to p; }
global protocol ClientC(role p, role q) {
  PlayC(Game@c) from q to p; }

global protocol Game(role a, role b, role c) {
  InfoBC(String) from b to c;
  InfoCA(String) from c to a;
  InfoAB(String) from a to b;
  rec t { choice at a {
    Mov1AB(Int) from a to b;
    Mov1BC(Int) from b to c;
    choice at c { Mov1CA(Int) from c to a; continue t; }
             or { Mov2CA(Bool) from c to a; continue t; }
   } or {
    Mov2AB(Bool) from a to b;
    Mov2BC(Bool) from b to c;
    choice at c { Mov1CA(Int) from c to a; continue t; }
             or { Mov2CA(Bool) from c to a; continue t; }
} } }
```

```
def P_b(c_bin: In[binary.PlayB]) = { // Cf. Ex. 2.2
  // Wrap binary chan in generated multiparty API
  Client_b(MPPlayB(c_bin))
}

def Client_b(y: MPPlayB): Unit = {
  // Receive Game chan (wraps binary chans to a/c)
  val z = y.receive().p // p is the payload field
  // Send info to c; wait info from a; enter loop
  Loop_b(z.send(InfoBC("...")).receive())
}

def Loop_b(x: MPMov1ABOrMov2AB): Unit = {
  x.receive() match { // Check a's move
    case Mov1AB(p, cont) => {
      // cont only allows to send Mov1BC
      Loop_b(cont.send(Mov1BC(p)))
    }
    case Mov2AB(p, cont) => {
      // cont only allows to send Mov2BC
      Loop_b(cont.send(Mov2BC(p)))
}}} // If e.g. case Mov2AB missing: compiler warn
```

Figure 9 Game example (from § 1). Left: Scribble protocols for client/server setup sessions, and main *Game* (matching Ex. 2.18). Right: Scala client for player b, using Scribble-generated APIs, and mimicking the processes in Ex. 2.2 (for a more natural implementation on the same API, see § A.5).

correctness of our (encoding-based) Scala implementation. Moreover, our encoding *preserves process distribution* (i.e., is homomorphic w.r.t. parallel composition); correspondingly, our implementation of multiparty sessions is decentralised and *choreographic*.

Implementations of Session Types (for Mainstream Languages) We mentioned the implementations of *binary* sessions for a range of "mainstream" languages in § 1. Notably, [52, 30, 31, 40, 48, 56, 51] sought to realise benefits from session types in the *native* host language, *without* language extensions, to avoid hindering their use in practice. To do so, one approach (employed e.g. in [56, 51]) is the combination of *static* typing of I/O actions on channels, and *run-time* checking of linear channel usage. We adopt this idea in our implementation (§ 7). The Haskell-based works, instead, exploit its richer typing facilities to statically enforce linearity—but incur various expressiveness/usability compromises according to the particular strategy for embedding session types.

By contrast, implementations of *multiparty* sessions are, to date, limited, in part due to the intricacies of the multiparty theory (e.g., the interplay between *projections*, *mergability* and *consistency*), and practical issues (e.g., realising the multiparty session abstraction over binary transports, including distributed delegation), as discussed in § 1. [27] proposes MPST-based API generation for Java based on communicating FSMs and has no formalisation, unlike our implementation—which follows directly from our formal encoding. [58] was the first implementation of MPST, based on *extending* Java with special-purpose session primitives. [16, 19] developed MPST-influenced networking APIs in Python and Erlang, respectively; [46] implemented recovery strategies in Erlang (based on Scribble). [16, 19, 46] focus on *purely dynamic* MPST verification by run-time monitoring. [47, 45] extended [16] with actors and timed specifications, respectively. [43] uses a dependent MPST theory for verifications of MPI programs. Crucially, *none* of these MPST-based implementations support delegation (nor merging of choice projections, as needed by our running Game example—cf. Ex. 2.14).

Encodings of Session Types and Processes [15] encodes binary session π-calculus into an *augmented* π-calculus with branch/select constructs. [14], by following [34], and [20] encode *non-recursive, binary* session π-calculus, respectively into linear π-calculus, and the Generic Type System for π-calculus [29], and prove correctness w.r.t. typing and reduction. All the above works investigate *binary* and (except [15]) *non-recursive* session types, while in this paper we study the encoding of *multiparty* session types, subsuming binary ones; and unlike [15], we target *standard* π-calculus. We encode branching/selection using variants as in [14, 12], but our treatment of *recursion*, and the rest of the MPST theory, is novel.

The only works studying an encoding of *multiparty* sessions into binary sessions are [8, 7]: they adopt an *orchestrated* approach, by adding centralised *medium*/*arbiter* processes. Moreover, they target session calculi and *not* π-calculus, with a wider gap towards implementation. In [49] a restricted class of global types is used to extract "characteristic", deadlock-free π-calculus processes—without addressing session calculi, nor proving operational properties.

Recursion and Duality The interplay between recursion and duality has been a thorny issue in session types literature, thus requiring our careful treatment in § 4. [6] and [1] noticed that the *standard duality* in [25] does *not* commute with the unfolding of recursion when type variables occur as payload, e.g., $\mu \mathbf{t}.!\mathbf{t}.\mathbf{end}$. To solve this issue, [6, 1] define a new notion of duality, called *complement* in [1], that is used in the encoding of *recursive binary* session types into linear π-types [12]. Unfortunately, [2] later noticed that even complement does *not* commute, e.g., when unfolding $\mu \mathbf{t}.\mu \mathbf{t}'.!\mathbf{t}.\mathbf{t}'$. As said in § 4, to encode *recursive* session types we encounter similar issues in the π-types. The reason seems quite natural: in π-calculus, types do not distinguish between "payload" and "continuation", and, in the case of recursive linear inputs/outputs, type variables necessarily occur as "payload", e.g. $\mu \mathbf{t}.\mathsf{Lo}(\mathbf{t})$. Since, in the light of [2], we could not adopt the approach of [12], we proposed a solution similar to [41]: introduce *dualised type variables* $\overline{\mathbf{t}}$. [41] also sketches a property similar to our Lemma 4.4. The main difference is that, we add dualised variables to π-types (while [41] adds $\overline{\mathbf{t}}$ to session types). An alternative approach is given in [56]: recursive session types are encoded as *non-recursive* linear I/O types with *recursive payloads*. This avoids dualised variables (e.g., $\mathsf{Lo}(\mu \mathbf{t}.\mathsf{Li}(\mathbf{t}))$ instead of $\mu \mathbf{t}.\mathsf{Lo}(\overline{\mathbf{t}})$), but at the price of complicating Def. 5.1. Most importantly, [56] tackles *only* the encoding of recursive types and *not* processes.

Future work On the practical side, we plan to study whether Scala language extensions could provide stronger *static* channel usage checks E.g., [24, 23] (capabilities) could allow to ensure that a channel endpoint is not used after being sent; [53, 59] (effects) could allow to ensure that a channel endpoint is actually used in a given method. We also plan to extend our multiparty API generation approach beyond Scala and `lchannels`, targeting other languages and implementations of binary sessions/channels [52, 30, 31, 40, 48, 51].

On the theoretical side, our encoding provides a basis for reusing theoretical results and tools between MPST π-calculus and standard π-calculus. E.g., by leveraging Cor. 6.7, we could now study deadlock-freedom of processes with interleaved multiparty sessions (studied in [3, 9, 11]) by applying π-calculus deadlock detection methods to their encodings [36, 33, 60]. Moreover, we can prove that our encoding is *barb-preserving*: hence, we plan to study its *full abstraction* properties w.r.t. *barbed congruence* in session π-calculus [38, 37] and π-calculus.

References

1 Giovanni Bernardi and Matthew Hennessy. Using higher-order contracts to model session types (extended abstract). In *CONCUR*, volume 8704 of *LNCS*, pages 387–401. Springer, 2014. doi:10.1007/978-3-662-44584-6_27.

2 Giovanni Bernardi and Matthew Hennessy. Using higher-order contracts to model session types. *Logical Methods in Computer Science*, 12(2), 2016. doi:10.2168/LMCS-12(2:10)2016.

3 Lorenzo Bettini, Mario Coppo, Loris D'Antoni, Marco De Luca, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. Global progress in dynamically interleaved multiparty sessions. In *CONCUR*, volume 5201 of *LNCS*, pages 418–433. Springer, 2008. doi:10.1007/978-3-540-85361-9_33.

4 Laura Bocchi, Julien Lange, and Nobuko Yoshida. Meeting Deadlines Together (long version). Technical report. Long version of [5]. URL: http://mrg.doc.ic.ac.uk/publications/meeting-deadlines-together/long.pdf.

5 Laura Bocchi, Julien Lange, and Nobuko Yoshida. Meeting Deadlines Together. In *CONCUR*, 2015. doi:http://dx.doi.org/10.4230/LIPIcs.CONCUR.2015.283.

6 Viviana Bono and Luca Padovani. Typing copyless message passing. *Logical Methods in Computer Science*, 8(1), 2012. doi:10.2168/LMCS-8(1:17)2012.

7 Luís Caires and Jorge A. Pérez. Multiparty session types within a canonical binary theory, and beyond. In *FORTE*, volume 9688 of *LNCS*, pages 74–95. Springer, 2016. doi:10.1007/978-3-319-39570-8_6.

8 Marco Carbone, Sam Lindley, Fabrizio Montesi, Carsten Schürmann, and Philip Wadler. Coherence generalises duality: A logical explanation of multiparty session types. In *CONCUR*, 2016. doi:10.4230/LIPIcs.CONCUR.2016.33.

9 Mario Coppo, Mariangiola Dezani-Ciancaglini, Luca Padovani, and Nobuko Yoshida. Inference of global progress properties for dynamically interleaved multiparty sessions. In *COORDINATION*, volume 7890 of *LNCS*, pages 45–59. Springer, 2013. doi:10.1007/978-3-642-38493-6_4.

10 Mario Coppo, Mariangiola Dezani-Ciancaglini, Luca Padovani, and Nobuko Yoshida. *A Gentle Introduction to Multiparty Asynchronous Session Types*. Springer, 2015. doi:10.1007/978-3-319-18941-3_4.

11 Mario Coppo, Mariangiola Dezani-Ciancaglini, Nobuko Yoshida, and Luca Padovani. Global Progress for Dynamically Interleaved Multiparty Sessions. *Mathematical Structures in Computer Science*, 760:1–65, 2015.

12 Ornela Dardha. Recursive session types revisited. In *BEAT*, volume 162 of *EPTCS*, pages 27–34, 2014. doi:10.4204/EPTCS.162.4.

13 Ornela Dardha. *Type Systems for Distributed Programs: Components and Sessions*, volume 7 of *Atlantis Studies in Computing*. Atlantis Press, July 2016. URL: http://dx.doi.org/10.2991/978-94-6239-204-5, doi:10.2991/978-94-6239-204-5.

14 Ornela Dardha, Elena Giachino, and Davide Sangiorgi. Session types revisited. In *PPDP*, pages 139–150, New York, NY, USA, 2012. ACM.

15 Romain Demangeon and Kohei Honda. Full abstraction in a subtyped pi-calculus with linear types. In *CONCUR*, volume 6901 of *LNCS*, pages 280–296. Springer, 2011. doi:10.1007/978-3-642-23217-6_19.

16 Romain Demangeon, Kohei Honda, Raymond Hu, Rumyana Neykova, and Nobuko Yoshida. Practical interruptible conversations: Distributed dynamic verification with multiparty session types and python. *Formal Methods in System Design*, pages 1–29, 2015.

17 Pierre-Malo Deniélou, Nobuko Yoshida, Andi Bejleri, and Raymond Hu. Parameterised multiparty session types. *Logical Methods in Computer Science*, 8(4), 2012. doi:10.2168/LMCS-8(4:6)2012.

18 Mariangiola Dezani-Ciancaglini, Silvia Ghilezan, Svetlana Jaksic, Jovanka Pantovic, and Nobuko Yoshida. Precise subtyping for synchronous multiparty sessions. In *PLACES*, pages 29–43, 2015. doi:10.4204/EPTCS.203.3.

19 Simon Fowler. An Erlang implementation of multiparty session actors. In *ICE '16*, volume 223 of *EPTCS*, pages 36–50, 2016. URL: http://dx.doi.org/10.4204/EPTCS.223.3, doi:10.4204/EPTCS.223.3.

20 Simon J. Gay, Nils Gesbert, and António Ravara. Session types as generic process types. In *EXPRESS/SOS*, volume 160 of *EPTCS*, pages 94–110, 2014. doi:10.4204/EPTCS.160.9.

21 Simon J. Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Informatica*, 42(2-3):191–225, 2005.

22 Daniele Gorla. Towards a unified approach to encodability and separation results for process calculi. *Inf. Comput.*, 208(9):1031–1053, 2010. doi:10.1016/j.ic.2010.05.002.

23 Philipp Haller and Alexander Loiko. Lacasa: lightweight affinity and object capabilities in Scala. In *OOPSLA*.

24 Philipp Haller and Martin Odersky. Capabilities for uniqueness and borrowing. In *ECOOP*, pages 354–378, Berlin, Heidelberg, 2010. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=1883978.1884002, doi:10.1007/978-3-642-14107-2_17.

25 Kohei Honda, Vasco Vasconcelos, and Makoto Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP*, volume 1381 of *LNCS*, pages 22–138. Springer, 1998.

26 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *POPL*, volume 43(1), pages 273–284. ACM, 2008. A full version: Volume 63, Issue 1, March 2016 (9), pages 1-67, *JACM*.

27 Raymond Hu and Nobuko Yoshida. Hybrid session verification through endpoint API generation. In *FASE*, volume 9633 of *LNCS*, pages 401–418. Springer, 2016. doi:10.1007/978-3-662-49665-7_24.

28 Raymond Hu, Nobuko Yoshida, and Kohei Honda. Session-based distributed programming in java. In *ECOOP*, volume 5142 of *LNCS*, pages 516–541. Springer, 2008. doi:10.1007/978-3-540-70592-5_22.

29 Atsushi Igarashi and Naoki Kobayashi. A generic type system for the pi-calculus. *Theo. Comput. Sci.*, 311(1-3):121–163, 2004. doi:10.1016/S0304-3975(03)00325-6.

30 Keigo Imai, Shoji Yuen, and Kiyoshi Agusa. Session type inference in haskell. In *PLACES*, volume 69 of *EPTCS*, page 74, 2010.

31 Thomas Bracht Laumann Jespersen, Philip Munksgaard, and Ken Friis Larsen. Session types for rust. In *WGP@ICFP*, pages 13–22. ACM, 2015. doi:10.1145/2808098.2808100.

32 Naoki Kobayashi. Type systems for concurrent programs. In *10th Anniversary Colloquium of UNU/IIST*, pages 439–453, 2002.

33 Naoki Kobayashi. A new type system for deadlock-free processes. In *CONCUR*, volume 4137 of *LNCS*, pages 233–247. Springer, 2006.

34 Naoki Kobayashi. Type systems for concurrent programs. Extended version of [32], Tohoku University, 2007.

35 Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the pi-calculus. *ACM Trans. Program. Lang. Syst.*, 21(5):914–947, September 1999. doi:10.1145/330249.330251.

36 Naoki Kobayashi and Davide Sangiorgi. A hybrid type system for lock-freedom of mobile processes. *ACM Trans. Program. Lang. Syst.*, 32(5), 2010.

37 Dimitrios Kouzapas and Nobuko Yoshida. Globally governed session semantics. In *CONCUR*, pages 395–409, 2013. doi:10.1007/978-3-642-40184-8_28.

38 Dimitrios Kouzapas and Nobuko Yoshida. Globally governed session semantics. *Logical Methods in Computer Science*, 10(4), 2014. doi:10.2168/LMCS-10(4:20)2014.

39 Inc. Lightbend. The Akka framework, 2016. URL: `http://akka.io/`.

40 Sam Lindley and J. Garrett Morris. Embedding session types in haskell. In *Proceedings of the 9th International Symposium on Haskell, Haskell 2016, Nara, Japan, September 22-23, 2016*, pages 133–145. ACM, 2016. URL: `http://doi.acm.org/10.1145/2976002.2976018`, `doi:10.1145/2976002.2976018`.

41 Sam Lindley and J. Garrett Morris. Talking bananas: Structural recursion for session types. In *ICFP*, pages 434–447, 2016. `doi:10.1145/2951913.2951921`.

42 Links homepage. `http://links-lang.org/`. S. Fowler and D. Hillerström and S. Lindley and G. Morris and P. Wadler.

43 Hugo A. Lopez, Eduardo R. B. Marques, Francisco Martins, Nicholas Ng, Casar Santos, Vasco Thudichum Vasconcelos, and Nobuko Yoshida. Protocol-based verification of message-passing parallel programs. In *OOPSLA*, pages 280–298. ACM, 2015.

44 Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, parts I and II. *Inf. Comput.*, 100(1):1–77, 1992. `doi:10.1016/0890-5401(92)90008-4`.

45 Rumyana Neykova, Laura Bocchi, and Nobuko Yoshida. Timed Runtime Monitoring for Multiparty Conversations. In *3rd International Workshop on Behavioural Types*, volume 162. EPTCS, 2014. A full version in *Formal Aspects of Computing*. `doi:10.4204/EPTCS.162.3`.

46 Rumyana Neykova and Nobuko Yoshida. Let It Recover: Multiparty Protocol-Induced Recovery. In *26th International Conference on Compiler Construction*. ACM, 2017.

47 Rumyana Neykova and Nobuko Yoshida. Multiparty Session Actors. *Logical Methods in Computer Science*, 2017.

48 Dominic A. Orchard and Nobuko Yoshida. Effects as sessions, sessions as effects. In *POPL*, pages 568–581. ACM, 2016. `doi:10.1145/2837614.2837634`.

49 Luca Padovani. Deadlock and lock freedom in the linear π-calculus. online version of [50], January 2014. URL: `https://hal.inria.fr/hal-00932356`.

50 Luca Padovani. Deadlock and lock freedom in the linear π-calculus. In *CSL-LICS*, pages 72:1–72:10. ACM, 2014. `doi:10.1145/2603088.2603116`.

51 Luca Padovani. Fuse - a simple library implementation of binary sessions, 2016. URL: `http://www.di.unito.it/~padovani/Software/FuSe/FuSe.html`.

52 Riccardo Pucella and Jesse A. Tov. Haskell session types with (almost) no class. In *Haskell*, pages 25–36. ACM, 2008. `doi:10.1145/1411286.1411290`.

53 Lukas Rytz, Martin Odersky, and Philipp Haller. Lightweight polymorphic effects. In James Noble, editor, *ECOOP*, pages 258–282, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. URL: `http://dx.doi.org/10.1007/978-3-642-31057-7_13`, `doi:10.1007/978-3-642-31057-7_13`.

54 Davide Sangiorgi and David Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

55 Alceste Scalas, Ornela Dardha, Raymond Hu, and Nobuko Yoshida. A linear decomposition of multiparty sessions, 2016. Technical report and source code. URL: `https://www.doc.ic.ac.uk/~ascalas/mpst-linear/`.

56 Alceste Scalas and Nobuko Yoshida. Lightweight session programming in scala. In *ECOOP*, volume 56 of *LIPIcs*, pages 21:1–21:28, 2016. `doi:10.4230/LIPIcs.ECOOP.2016.21`.

57 Scribble homepage. `http://www.scribble.org`.

58 K. C. Sivaramakrishnan, Karthik Nagaraj, Lukasz Ziarek, and Patrick Eugster. Efficient session type guided distributed interaction. In *Coordination'10*, volume 6116 of *LNCS*, pages 152–167, 2010.

59 Matías Toro and Éric Tanter. Customizable gradual polymorphic effects for Scala. In *OOPSLA*, pages 935–953, New York, NY, USA, 2015. ACM. URL: `http://doi.acm.org/10.1145/2814270.2814315`, `doi:10.1145/2814270.2814315`.

60 TYPICAL. Type-based static analyzer for the pi-calculus. http://www-kb.is.s.u-tokyo.ac.jp/~koba/typical/.

61 Nobuko Yoshida, Pierre-Malo Deniélou, Andi Bejleri, and Raymond Hu. Parameterised multiparty session types. In *FOSSACS*, volume 6014 of *LNCS*, pages 128–145. Springer, 2010. doi:10.1007/978-3-642-12032-9_10.

62 Nobuko Yoshida, Raymond Hu, Rumyana Neykova, and Nicholas Ng. The Scribble protocol language. In *TGC'13*, volume 8358 of *LNCS*, pages 22–41. Springer, 2013.

Appendices

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

$$(\boldsymbol{\nu} s)(\boldsymbol{\nu} s')P \equiv (\boldsymbol{\nu} s')(\boldsymbol{\nu} s)P \qquad (\boldsymbol{\nu} s)P \mid Q \equiv (\boldsymbol{\nu} s)(P \mid Q) \;\; (\text{if } \; s \notin \text{fc}(Q)) \qquad \mathbf{def}\, D \,\mathbf{in}\, \mathbf{0} \equiv \mathbf{0}$$

$$\mathbf{def}\, D \,\mathbf{in}\, (\boldsymbol{\nu} s)P \equiv (\boldsymbol{\nu} s)\mathbf{def}\, D \,\mathbf{in}\, P \;\; (\text{if } \; s \notin \text{fc}(P))$$

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

$$\mathbf{def}\, D \,\mathbf{in}\, \mathbf{def}\, D' \,\mathbf{in}\, P \equiv \mathbf{def}\, D' \,\mathbf{in}\, \mathbf{def}\, D \,\mathbf{in}\, P$$

$$(\text{if } \; (\text{dpv}(D) \cup \text{fpv}(D)) \cap \text{dpv}(D') = (\text{dpv}(D') \cup \text{fpv}(D')) \cap \text{dpv}(D) = \emptyset)$$

Figure 10 Structural congruence for the multiparty session π-calculus.

A Auxiliary Definitions

A.1 Structural Congruence for Multiparty Session π-Calculus

The operational semantics of multiparty session processes is based on the notion of structural congruence $\equiv$, given in Fig. 12. We write "$s \notin \text{fc}(P)$" to mean that there does not exist a p such that $s[\mathsf{p}] \in \text{fc}(P)$. We use $\text{fv}(D)$ to denote the set of *free variables* in D. We use $\text{dpv}(D)$ to denote the set of process variables declared in D, and $\text{fpv}(P)$ for the set of process variables which occur free in P.

Most of the rules of structural congruence are standard. The first two lines in Fig. 12 show the commutativity and associativity of the relation w.r.t. parallel composition and restriction and $\mathbf{0}$ used as the neutral element w.r.t. parallel composition, restriction and process definition. The last three lines in Fig. 12 describe how a process definition can be rearranged w.r.t. restriction, parallel composition and process definition, respectively. These rules make use of well-formedness criteria on the free names and variables in the process definition, which are given as side conditions.

A.2 Global Types

We now provide the formal definition of projection of a global type onto a role.

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

$$\left(\mathsf{p} \to \mathsf{p}' : \{l_i(U_i).G_i\}_{i \in I}\right) \upharpoonright \mathsf{q} \triangleq \begin{cases} \mathsf{p}' \oplus_{i \in I} !l_i(U_i).(G_i \upharpoonright \mathsf{q}) & \textit{if } \mathsf{q} = \mathsf{p}' \\ \mathsf{p}' \&_{i \in I} ?l_i(U_i).(G_i \upharpoonright \mathsf{q}) & \textit{if } \mathsf{q} = \mathsf{p}' \\ \prod_{i \in I} (G_i \upharpoonright \mathsf{q}) & \textit{if } \mathsf{p} \neq \mathsf{q} \neq \mathsf{p}' \end{cases}$$

$$(\mu \mathbf{t}.G) \upharpoonright \mathsf{q} \triangleq \begin{cases} \mu \mathbf{t}.(G \upharpoonright \mathsf{q}) & \textit{if } G \upharpoonright \mathsf{q} \neq \mathbf{t}' \; (\forall \mathbf{t}') \\ \mathbf{end} & \textit{otherwise} \end{cases} \qquad \mathbf{t} \upharpoonright \mathsf{q} \triangleq \mathbf{t} \qquad \mathbf{end} \upharpoonright \mathsf{q} \triangleq \mathbf{end}$$

where $\sqcap$ *is the merge operator on* session types, *defined as:*

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

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

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

The projection of global types onto session types gives branch and select types, as well as recursion and termination, which are the multiparty session types given in Def. 2.5. The intuition behind it follows the same lines as Def. 2.9.

The merge operation [61, 17], as for partial projections, makes local projections defined in more cases.

A.3 Structural Congruence for Standard π-Calculus

In order to complete the operational semantics for the π-calculus, we need the structural congruence relation, $\equiv$; it is defined as the smallest congruence relation on processes that satisfies the axioms given in Fig. 11.

$$
\begin{array}{rcll}
P \mid Q & \equiv & Q \mid P & \\
(P \mid Q) \mid R & \equiv & P \mid (Q \mid R) & \\
P \mid \mathbf{0} & \equiv & P & \\
(\boldsymbol{\nu} x)\mathbf{0} & \equiv & \mathbf{0} & \\
(\boldsymbol{\nu} x)(\boldsymbol{\nu} y)P & \equiv & (\boldsymbol{\nu} y)(\boldsymbol{\nu} x)P & \\
(\boldsymbol{\nu} x)P \mid Q & \equiv & (\boldsymbol{\nu} x)(P \mid Q) & (x \notin \mathrm{fv}(Q)) \\
*P & \equiv & P \mid *P &
\end{array}
$$

Figure 11 Structural congruence for the standard π-calculus.

The first three axioms say that the parallel composition of processes is commutative, associative and uses process **0** as the neutral element. The next three axioms involve restriction: the first of the sequence is used to collect vacuous restrictions, by saying that restriction can be removed from the terminated process, the second says that restriction is commutative and the third is the main one, *scope extrusion*, saying that the scope of a restriction can be extended to other parallel processes provided that no free names are captured. The last axiom states that replication can be "decomposed" into a parallel composition of a copy of the process itself and the persistent replicated process.

A.4 "Let" binder reduction and typing

The "let" binder is just a macro based on standard π-calculus constructs. Hence, its reduction and typing follow the expansion of its definition (Fig. 6):

$$
\begin{array}{rcll}
\mathbf{let}\, x = v \,\mathbf{in}\, P & = & (\boldsymbol{\nu} z)\,(\overline{z}\langle v\rangle.\mathbf{0} \mid z(x).P) & \text{(where } z \notin \{x\} \cup \mathrm{fn}(v) \cup \mathrm{fn}(P)\text{)} \\
 & \to & (\boldsymbol{\nu} z)\,(\mathbf{0} \mid P\{^{v}/_{x}\}) \;\equiv\; (\boldsymbol{\nu} z)\mathbf{0} \mid P\{^{v}/_{x}\} \;\equiv\; P\{^{v}/_{x}\} &
\end{array}
$$

$$
\dfrac{
\dfrac{
\dfrac{\Gamma_1 \vdash v : T \qquad \dfrac{}{\vdash \mathbf{0}}\,(\text{T}\pi\text{-Nil})}{\Gamma_1, z : \mathsf{Lo}(T) \vdash \overline{z}\langle v\rangle.\mathbf{0}}\,(\text{T}\pi\text{-Out})
\qquad
\dfrac{\Gamma_2, x : T \vdash P}{\Gamma_2, z : \mathsf{Li}(T) \vdash z(x).P}\,(\text{T}\pi\text{-Inp})
}{\Gamma_1 \uplus \Gamma_2, z : \mathsf{Lo}(T) \uplus \mathsf{Li}(T) \vdash \overline{z}\langle v\rangle.\mathbf{0} \mid z(x).P}\,(\text{T}\pi\text{-Par})
}{\Gamma_1 \uplus \Gamma_2 \vdash \mathbf{let}\, x = v \,\mathbf{in}\, P = (\boldsymbol{\nu} z)\,(\overline{z}\langle v\rangle.\mathbf{0} \mid z(x).P)}\,(\text{T}\pi\text{-Res1})
$$

A.5 Multiparty API Generation for Scala

The following code shows an alternative (and more natural) implementation of the b-playing game client in Fig. 9 (right): albeit using the same Scribble-generated APIs, it does *not* try to mimic the processes in Ex. 2.2.

```
def client(s: In[binary.PlayB]) = {
  // Wrap binary chan in multiparty session obj
  val c = MPPlayB(s)
  // Receive multiparty game channel
  val g = c.receive().p
  // Send info to C; wait info from a
  val i = g.send(InfoBC("...")).receive()
  loop(i.cont) // Game loop
}

def loop(g: MPMov1ABOrMov2AB): Unit = {
  g.receive() match { // Check a's move
    case Mov1AB(p, cont) => {
      // cont only allows to send Mov1BC
      val g2 = cont.send(Mov1BC(p))
      loop(g2) // Keep playing
    }
    case Mov2AB(p, cont) => {
      // cont only allows to send Mov2BC
      val g2 = cont.send(Mov2BC(p))
      loop(g2) // Keep playing
} } } // If case Mov1AB or Mov2AB is missing: compiler warn
```

B Multiparty Session Types

B.1 Structural Congruence for Multiparty Session π-Calculus

The operational semantics of multiparty session processes is based on the notion of structural congruence $\equiv$, given in Fig. 12. We write "$s \notin \mathrm{fc}(P)$" to mean that there does not exist a $\mathtt{p}$ such that $s[\mathtt{p}] \in \mathrm{fc}(P)$. We use $\mathrm{fv}(D)$ to denote the set of *free variables* in D. We use $\mathrm{dpv}(D)$ to denote the set of process variables declared in D, and $\mathrm{fpv}(P)$ for the set of process variables which occur free in P.

Most of the rules of structural congruence are standard. The first two lines in Fig. 12 show the commutativity and associativity of the relation w.r.t. parallel composition and restriction and $\mathbf{0}$ used as the neutral element w.r.t. parallel composition, restriction and process definition. The last three lines in Fig. 12 describe how a process definition can be rearranged w.r.t. restriction, parallel composition and process definition, respectively. These rules make use of well-formedness criteria on the free names and variables in the process definition, which are given as side conditions.

B.2 Global Types

We will now formally introduce global types and give the definition of projection onto a role.

▶ **Definition B.1.** *The syntax of global types, ranged over by G, is:*

$$\begin{array}{rcll} G & ::= & \mathtt{p} \to \mathtt{q}\colon\{l_i(U_i).G_i\}_{i\in I} & \textit{(interaction — with } U_i \textit{ closed)} \\ & & \mu\mathbf{t}.G \;\mid\; \mathbf{t} \;\mid\; \mathbf{end} & \textit{(recursive type, type variable, termination)} \end{array}$$

Type $\mathtt{p} \to \mathtt{q}\colon\{l_i(U_i).G_i\}_{i\in I}$ states that role $\mathtt{p}$ sends one of the labels l_i for $i \in I$, together with a payload, to role $\mathtt{q}$. Labels are pairwise distinct. If such label is l_j, then the continuation proceeds as G_j. Type $\mu\mathbf{t}.G$ is a recursive type, where type variables $\mathbf{t}, \mathbf{t}', \ldots$ are guarded, namely they appear only under type prefixes. Finally, type $\mathbf{end}$ states the termination of a session. We may omit the braces $\{\ldots\}$ from an interaction when I is a singleton, e.g., to write $\mathtt{a}\to\mathtt{b}\colon l_1(U_1).G_1$ instead of $\mathtt{a}\to\mathtt{b}\colon\{l_i(U_i).G_i\}_{i\in\{1\}}$.

The relation between global types and session types is formalised by the notion of projection, given below.

▶ **Definition B.2.** *The* projection of G onto a role $\mathtt{q}$, *written* $G \upharpoonright \mathtt{q}$, *is:*

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

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

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

$$(\nu s)(\nu s')P \equiv (\nu s')(\nu s)P \qquad (\nu s)P \mid Q \equiv (\nu s)(P \mid Q) \;\; (\text{if } s \notin \mathrm{fc}(Q)) \qquad \mathbf{def}\; D \;\mathbf{in}\; \mathbf{0} \equiv \mathbf{0}$$

$$\mathbf{def}\; D \;\mathbf{in}\; (\nu s)P \equiv (\nu s)\mathbf{def}\; D \;\mathbf{in}\; P \;\; (\text{if } s \notin \mathrm{fc}(P))$$

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

$$\mathbf{def}\; D \;\mathbf{in}\; \mathbf{def}\; D' \;\mathbf{in}\; P \equiv \mathbf{def}\; D' \;\mathbf{in}\; \mathbf{def}\; D \;\mathbf{in}\; P$$

$$(\text{if } (\mathrm{dpv}(D) \cup \mathrm{fpv}(D)) \cap \mathrm{dpv}(D') = (\mathrm{dpv}(D') \cup \mathrm{fpv}(D')) \cap \mathrm{dpv}(D) = \emptyset)$$

Figure 12 Structural congruence for the multiparty session π-calculus.

where $\sqcap$ *is the merge operator on* session types*, defined as:*

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

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

$$\mathbf{end} \sqcap \mathbf{end} \;\triangleq\; \mathbf{end} \qquad \mu\mathbf{t}.S \sqcap \mu\mathbf{t}.S' \;\triangleq\; \mu\mathbf{t}.(S \sqcap S') \qquad \mathbf{t} \sqcap \mathbf{t} \;\triangleq\; \mathbf{t}$$

The projection of global types onto session types gives branch and select types, as well as recursion and termination, which are the multiparty session types given in Def. 2.5. The intuition behind it follows the same lines as Def. 2.9.

The merge operation [61, 17], as for partial projections, makes local projections defined in more cases.

B.3 Properties of Partial Session Types

▶ **Definition B.3** (Open Subtyping for Partial Session Types)**.** *The relation* $\leqslant_{\mathsf{OP}}$ *between partial session types is inductively defined by the following rules:*

$$\frac{\forall i \in I \qquad H_i \leqslant_{\mathsf{OP}} H'_i}{\&_{i\in I}\,?l_i(U_i).H_i \leqslant_{\mathsf{OP}} \&_{i\in I\cup J}\,?l_i(U_i).H'_i}\ (\text{S-OPARBRCH}) \qquad \frac{\forall i \in I \qquad H_i \leqslant_{\mathsf{OP}} H'_i}{\oplus_{i\in I\cup J}\,!l_i(U_i).H_i \leqslant_{\mathsf{OP}} \oplus_{i\in I}\,!l_i(U_i).H'_i}\ (\text{S-OPARSEL})$$

$$\frac{H \leqslant_{\mathsf{OP}} H'}{\mu\mathbf{t}.H \leqslant_{\mathsf{OP}} \mu\mathbf{t}.H'}\ (\text{S-OPAR}\mu) \qquad \frac{}{\mathbf{t} \leqslant_{\mathsf{OP}} \mathbf{t}}\ (\text{S-OPAR}\mathbf{t}) \qquad \frac{}{\mathbf{end} \leqslant_{\mathsf{OP}} \mathbf{end}}\ (\text{S-OPAREND})$$

▶ **Corollary B.4.** $\leqslant_{\mathsf{OP}}$ *is reflexive.*

Proof. For all H, we can prove $H \leqslant_{\mathsf{OP}} H$ by easy structural induction on H. ◀

▶ **Proposition B.5.** *If* $\mu\mathbf{t}.H_1 \leqslant_{\mathsf{OP}} \mu\mathbf{t}.H_2$*, then* $H_1\{^{\mu\mathbf{t}.H_1}/_{\mathbf{t}}\} \leqslant_{\mathsf{OP}} H_2\{^{\mu\mathbf{t}.H_2}/_{\mathbf{t}}\}$.

Proof. Assume $\mu\mathbf{t}.H_1 \leqslant_{\mathsf{OP}} \mu\mathbf{t}.H_2$. Without loss of generality, assume that all bound variables in $\mu\mathbf{t}.H_1$ are pairwise distinct, and similarly for $\mu\mathbf{t}.H_2$ (otherwise, the requirement can be met via α-conversion — i.e., this is a form of Barendregt convention). Such a relation can only hold by (S-OPARμ), and therefore we have some derivation $\mathcal{D}$ such that:

$$\frac{\mathcal{D}\left\{\dfrac{\vdots}{H_1 \leqslant_{\mathsf{OP}} H_2}\right.}{\mu\mathbf{t}.H_1 \leqslant_{\mathsf{OP}} \mu\mathbf{t}.H_2}\ (\text{S-OPAR}\mu)$$

We can inductively rewrite $\mathcal{D}$ by replacing each occurrence of $H \leqslant_{\mathsf{OP}} H'$ with $H\{^{\mu\mathbf{t}.H_1}/_{\mathbf{t}}\} \leqslant_{\mathsf{OP}} H'\{^{\mu\mathbf{t}.H_2}/_{\mathbf{t}}\}$. This way, we obtain a new derivation $\mathcal{D}'$ where:

1. each instance of the axiom (S-OPAR$\mathbf{t}$) with $\mathbf{t} \leqslant_{\mathsf{OP}} \mathbf{t}$ in $\mathcal{D}$ becomes $\mu\mathbf{t}.H_1 \leqslant_{\mathsf{OP}} \mu\mathbf{t}.H_2$ (which holds by hypothesis) in $\mathcal{D}'$;
2. the conclusion of $\mathcal{D}'$ is $H_1\{^{\mu\mathbf{t}.H_1}/_{\mathbf{t}}\} \leqslant_{\mathsf{OP}} H_2\{^{\mu\mathbf{t}.H_2}/_{\mathbf{t}}\}$.

Hence, $\mathcal{D}'$ proves the thesis. ◀

▶ **Lemma B.6.** *Let* H, H' *be closed partial session types. Then,* $H \leqslant_{\mathsf{OP}} H'$ *implies* $H \leqslant_{\mathsf{P}} H'$.

Proof. Consider the following relation:

$$\begin{array}{rcl} \mathcal{R} & = & \mathcal{R}_1 \cup \mathcal{R}_2 \\ \mathcal{R}_1 & = & \left\{(H, H') \,\middle|\, H, H' \text{ closed and } H \leqslant_{\mathsf{OP}} H'\right\} \\ \mathcal{R}_2 & = & \left\{(H_1\{^{\mu\mathbf{t}.H_1}/_{\mathbf{t}}\}, \mu\mathbf{t}.H_2),\ (\mu\mathbf{t}.H_1, H_2\{^{\mu\mathbf{t}.H_2}/_{\mathbf{t}}\}) \,\middle|\, \mu\mathbf{t}.H_1, \mu\mathbf{t}.H_2 \text{ closed and } \mu\mathbf{t}.H_1 \leqslant_{\mathsf{OP}} \mu\mathbf{t}.H_2\right\} \end{array}$$

We first prove that $\mathcal{R}$ is closed backwards under the rules obtained from Def. 2.10, by replacing each occurrence of $\leqslant_{\mathsf{P}}$ with $\mathcal{R}$. For each $(H, H') \in \mathcal{R}$, we have two cases:

- $(H, H') \in \mathcal{R}_1$. We know that H, H' are closed and $H \leqslant_{\mathsf{OP}} H'$. Therefore, we proceed by cases on the rule in Def. B.3 that concludes $H \leqslant_{\mathsf{OP}} H'$:
  - (S-OPAR**t**). This case is absurd: it would imply $H = H' = \mathbf{t}$, which contradicts the hypothesis that H, H' are closed;
  - (S-OPAREND). We have $H = H' = \mathbf{end}$, which satisfies rule (S-PAREND);
  - (S-OPARBRCH). We have $H = \&_{i \in I}\, ?l_i(U_i).H_i \leqslant_{\mathsf{OP}} \&_{i \in I \cup J}\, ?l_i(U_i).H'_i = H'$, and we need to show that (H, H') satisfies rule (S-PARBRCH). We observe, for all $i \in I$:

$$H_i, H'_i \text{ are closed} \qquad \text{(otherwise, } H \text{ or } H' \text{ would } \textit{not} \text{ be closed)} \tag{4}$$
$$H_i \leqslant_{\mathsf{OP}} H'_i \qquad \text{(from the premise of (S-OPARBRCH))} \tag{5}$$
$$H_i \,\mathcal{R}\, H'_i \qquad \text{(from (4) and (5), by definition of } \mathcal{R}_1\text{)} \tag{6}$$
$$U_i \leqslant_{\mathsf{S}} U_i \qquad \text{(by reflexivity of } \leqslant_{\mathsf{S}}\text{)} \tag{7}$$

    Hence, from (6) and (7) we conclude that (H, H') satisfies rule (S-PARBRCH);
  - (S-OPARSEL). We have $H = \oplus_{i \in I \cup J}\, !l_i(U_i).H_i \leqslant_{\mathsf{OP}} \oplus_{i \in I}\, !l_i(U_i).H'_i = H'$, and we need to show that (H, H') satisfies rule (S-PARSEL). We observe that, for all $i \in I$:

$$H_i, H'_i \text{ are closed} \qquad \text{(otherwise, } H \text{ or } H' \text{ would } \textit{not} \text{ be closed)} \tag{8}$$
$$H_i \leqslant_{\mathsf{OP}} H'_i \qquad \text{(from the premise of (S-OPARSEL))} \tag{9}$$
$$H_i \,\mathcal{R}\, H'_i \qquad \text{(from (8) and (9), by definition of } \mathcal{R}_1\text{)} \tag{10}$$
$$U_i \leqslant_{\mathsf{S}} U_i \qquad \text{(by reflexivity of } \leqslant_{\mathsf{S}}\text{)} \tag{11}$$

    Hence, from (10) and (11) we conclude that (H, H') satisfies rule (S-PARSEL);
  - (S-OPARμ). We have $H = \mu\mathbf{t}.H_1 \leqslant_{\mathsf{OP}} \mu\mathbf{t}.H_2 = H'$, and we need to show that (H, H') satisfies both rules (S-PARμL) and (S-PARμR). We observe that:

$$(H_1\{^{\mu\mathbf{t}.H_1}/_{\mathbf{t}}\}, \mu\mathbf{t}.H_2) \in \mathcal{R}_2 \subseteq \mathcal{R} \qquad \text{(by definition of } \mathcal{R}_2 \text{ and } \mathcal{R}\text{)} \tag{12}$$
$$(\mu\mathbf{t}.H_1, H_2\{^{\mu\mathbf{t}.H_2}/_{\mathbf{t}}\}) \in \mathcal{R}_2 \subseteq \mathcal{R} \qquad \text{(by definition of } \mathcal{R}_2 \text{ and } \mathcal{R}\text{)} \tag{13}$$

    Therefore, we conclude that (H, H') satisfies both (S-PARμL) (by (12)) and (S-PARμR) (by (13)).
- $(H, H') \in \mathcal{R}_2$. We know that H, H' are closed, and either:
  - $H = H_1\{^{\mu\mathbf{t}.H_1}/_{\mathbf{t}}\}$, $H' = \mu\mathbf{t}.H_2$, and $\mu\mathbf{t}.H_1 \leqslant_{\mathsf{OP}} \mu\mathbf{t}.H_2$. In this case, we need to show that (H, H') satisfies rule (S-PARμR). We observe that:

$$H_1\{^{\mu\mathbf{t}.H_1}/_{\mathbf{t}}\}, H_2\{^{\mu\mathbf{t}.H_2}/_{\mathbf{t}}\} \text{ are closed} \qquad \text{(otherwise, } H \text{ or } H' \text{ would } \textit{not} \text{ be closed)} \tag{14}$$
$$H_1\{^{\mu\mathbf{t}.H_1}/_{\mathbf{t}}\} \leqslant_{\mathsf{OP}} H_2\{^{\mu\mathbf{t}.H_2}/_{\mathbf{t}}\} \qquad \text{(by } \mu\mathbf{t}.H_1 \leqslant_{\mathsf{OP}} \mu\mathbf{t}.H_2 \text{ and Proposition B.5)} \tag{15}$$
$$(H_1\{^{\mu\mathbf{t}.H_1}/_{\mathbf{t}}\}, H_2\{^{\mu\mathbf{t}.H_2}/_{\mathbf{t}}\}) \in \mathcal{R}_1 \subseteq \mathcal{R} \qquad \text{(from (14), (15), and by definition of } \mathcal{R}_1 \text{ and } \mathcal{R}\text{)} \tag{16}$$

    Therefore, we conclude that (H, H') satisfies rule (S-PARμR);
  - $H = \mu\mathbf{t}.H_1$, $H' = H_2\{^{\mu\mathbf{t}.H_2}/_{\mathbf{t}}\}$, and $\mu\mathbf{t}.H_1 \leqslant_{\mathsf{OP}} \mu\mathbf{t}.H_2$. In this case, we need to show that (H, H') satisfies rule (S-PARμL): the proof is symmetric w.r.t. the previous case.

We have shown that $\mathcal{R}$ is closed backwards under the rules obtained from Def. 2.10. Therefore, since $\leqslant_{\mathsf{P}}$ is the *largest* relation closed backwards under such rules, we have $\mathcal{R} \subseteq \leqslant_{\mathsf{P}}$. We also know that for all closed H, H' such that $H \leqslant_{\mathsf{OP}} H'$, we have $(H, H') \subseteq \mathcal{R}_1 \subseteq \mathcal{R} \subseteq \leqslant_{\mathsf{P}}$: we conclude $H \leqslant_{\mathsf{P}} H'$. ◀

▶ **Lemma B.7.** *For any finite set of partial session types $\{H_i\}_{i \in I}$, if $H^* = \bigsqcap_{i \in I} H_i$ is defined, then $\forall k \in I : H^* \leqslant_{\mathsf{P}} H_k$.*

Proof. Assuming that H^* is defined, we choose any H_k (with $k \in I$) and proceed by structural induction on H_k:

- base case $H_k = \mathbf{end}$. By Def. 2.9, we must have $H^* = \mathbf{end}$ and $\forall i \in I : H_i = \mathbf{end}$ (otherwise, H^* would be undefined). We conclude $H^* \leqslant_{\mathsf{OP}} H_k$, by (S-OParEnd);
- base case $H_k = \mathbf{t}$. By Def. 2.9, we must have $H^* = \mathbf{t}$ and $\forall i \in I : H_i = \mathbf{t}$ (otherwise, H^* would be undefined). We conclude $H^* \leqslant_{\mathsf{OP}} H_k$, by (S-OPar$\mathbf{t}$);
- inductive case $H_k = \&_{j\in J}\, ?l_j(U_j).H_{kj}$. By Def. 2.9, we must have $H^* = \&_{j\in J}\, ?l_j(U_j).\big(\bigsqcap_{i\in I} H_{ij}\big)$ and $\forall i \in I : H_i = \&_{j\in J}\, ?l_j(U_j).H_{ij}$ (otherwise, H^* would be undefined). By the induction hypothesis, $\forall j \in J : \bigsqcap_{i\in I} H_{ij} \leqslant_{\mathsf{OP}} H_{kj}$: we conclude $H^* \leqslant_{\mathsf{OP}} H_k$, by (S-OParBrch);
- inductive case $H_k = \oplus_{j\in J_k}\, !l_j(U_j).H_{kj}$. By Def. 2.9, we must have:

  $$H^* \quad = \quad \oplus_{j\in J^*}\, !l_j(U_j).\big(\bigsqcap_{i\in I} H_{ij}\big) \;\oplus\; \bigoplus_{i\in I}\big(\oplus_{j\in J_i\setminus J^*}\, !l_j(U_j).H_{ij}\big) \qquad \text{where } J^* = \bigcap_{i\in I} J_i$$

  and $\forall i \in I : H_i = \oplus_{j\in J_i}\, !l_j(U_j).H_{ij}$ (otherwise, H^* would be undefined). By the induction hypothesis, $\forall j \in J^* \cap J_k : \bigsqcap_{i\in I} H_{ij} \leqslant_{\mathsf{OP}} H_{kj}$; moreover, $\forall j \in J_k \setminus J^* : H_{kj} \leqslant_{\mathsf{OP}} H_{kj}$ (by Cor. B.4). We conclude $H^* \leqslant_{\mathsf{OP}} H_k$, by (S-OParSel);
- inductive case $H_k = \mu\mathbf{t}.H'_k$. By Def. 2.9, we must have $H^* = \mu\mathbf{t}.(\bigsqcap_{i\in I} H'_i)$ and $\forall i \in I : H_i = \mu\mathbf{t}.H'_i$ (otherwise, H^* would be undefined). By the induction hypothesis, $\bigsqcap_{i\in I} H'_i \leqslant_{\mathsf{OP}} H'_k$: we conclude $H^* \leqslant_{\mathsf{OP}} H_k$, by (S-OPar$\mu$).

◀

▶ **Proposition B.8.** *For all partial types H, H', $H \leqslant_{\mathsf{P}} H'$ iff $\operatorname{unf}(H) \leqslant_{\mathsf{P}} H'$ iff $H \leqslant_{\mathsf{P}} \operatorname{unf}(H')$ iff $\operatorname{unf}(H) \leqslant_{\mathsf{P}} \operatorname{unf}(H')$.*

Proof. We split the statement in three parts, and prove them separately:

- ($H \leqslant_{\mathsf{P}} H'$ iff $\operatorname{unf}(H) \leqslant_{\mathsf{P}} H'$) Let $H = \mu\mathbf{t_1}.\ldots\mu\mathbf{t_m}.H_\diamond$, with $H_\diamond \neq \mu\mathbf{t}'.\ldots$. We first prove the following statement:

$$\forall n \in 0..m : \quad H \leqslant_{\mathsf{P}} H' \quad \text{iff} \quad H_*\{^{\mu\mathbf{t_1}.\ldots\mu\mathbf{t_n}.H_*}/_{\mathbf{t}_1}\}\ldots\{^{\mu\mathbf{t_n}.H_*}/_{\mathbf{t}_n}\} \leqslant_{\mathsf{P}} H' \quad \text{where } H_* = \mu\mathbf{t_{n+1}}.\ldots\mu\mathbf{t_m}.H_\diamond \tag{17}$$

  The proof proceeds by induction on n. The base case $n = 0$ is trivial, and holds by reflexivity of $\leqslant_{\mathsf{P}}$. In the inductive case $n = n' + 1$, we have (by the induction hypothesis):

  $$H \leqslant_{\mathsf{P}} H' \quad \text{iff} \quad \mu\mathbf{t_n}.H_*\{^{\mu\mathbf{t_1}.\ldots\mu\mathbf{t_{n'}}.\mu\mathbf{t_n}.H_*}/_{\mathbf{t}_1}\}\ldots\{^{\mu\mathbf{t_{n'}}.\mu\mathbf{t_n}.H_*}/_{\mathbf{t}_{n'}}\} \leqslant_{\mathsf{P}} H' \tag{18}$$

  We can notice that, by the coinductive rule (S-ParμL) in Def. 2.10, the RHS of the "iff" in (18) holds *if and only if*:

  $$H_*\{^{\mu\mathbf{t_1}.\ldots\mu\mathbf{t_{n'}}.\mu\mathbf{t_n}.H_*}/_{\mathbf{t}_1}\}\ldots\{^{\mu\mathbf{t_{n'}}.\mu\mathbf{t_n}.H_*}/_{\mathbf{t}_{n'}}\}\{^{\mu\mathbf{t_n}.H_*}/_{\mathbf{t}_n}\} \leqslant_{\mathsf{P}} H'$$

  which is the thesis.

  We conclude observing that, when $n = m$ (i.e., H is completely unfolded, bringing at the top-level $H_\diamond \neq \mu\mathbf{t}'\ldots$) we have proved $H \leqslant_{\mathsf{P}} H'$ iff $\operatorname{unf}(H) \leqslant_{\mathsf{P}} H'$.
- ($H \leqslant_{\mathsf{P}} H'$ iff $H \leqslant_{\mathsf{P}} \operatorname{unf}(H')$) The proof is symmetric w.r.t. the case above, and uses (S-ParμR);
- ($H \leqslant_{\mathsf{P}} H'$ iff $\operatorname{unf}(H) \leqslant_{\mathsf{P}} \operatorname{unf}(H')$) From the previous cases we have:

  $$\operatorname{unf}(H) \leqslant_{\mathsf{P}} H \quad \text{iff} \quad H \leqslant_{\mathsf{P}} H \qquad \text{and} \qquad H' \leqslant_{\mathsf{P}} H' \quad \text{iff} \quad H' \leqslant_{\mathsf{P}} \operatorname{unf}\big(H'\big)$$

  and by transitivity of $\leqslant_{\mathsf{P}}$, we conclude $H \leqslant_{\mathsf{P}} H'$ iff $\operatorname{unf}(H) \leqslant_{\mathsf{P}} \operatorname{unf}(H')$.

◀

▶ **Proposition B.9.** $\operatorname{unf}\big(\overline{H}\big) = \overline{\operatorname{unf}(H)}$.

Proof. Let $H = \mu\mathbf{t_1}\ldots\mu\mathbf{t_m}.H_\diamond$, with $H_\diamond \neq \mu\mathbf{t'}\ldots$. We first prove the following statement:

$$\forall n \in 0..m: \quad \overline{H_*}\left\{{}^{\mu\mathbf{t_1}\ldots\mu\mathbf{t_n}.\overline{H_*}}/_{\mathbf{t}_1}\right\}\ldots\left\{{}^{\mu\mathbf{t_n}.\overline{H_*}}/_{\mathbf{t}_n}\right\} = \overline{H_*\{{}^{\mu\mathbf{t_1}\ldots\mu\mathbf{t_n}.H_*}/_{\mathbf{t_1}}\}\ldots\{{}^{\mu\mathbf{t_n}.H_*}/_{\mathbf{t}_n}\}}$$
$$\text{where } H_* = \mu\mathbf{t_{n+1}}\ldots\mu\mathbf{t_m}.H_\diamond$$

The proof proceeds by induction on n. The base case $n = 0$ is trivial, while in the inductive case $n = n' + 1$ we have:

$$\overline{\mu\mathbf{t_n}.H_*}\left\{{}^{\mu\mathbf{t_1}\ldots\mu\mathbf{t_{n'}}.\overline{\mu\mathbf{t_n}.H_*}}/_{\mathbf{t}_1}\right\}\ldots\left\{{}^{\mu\mathbf{t_{n'}}.\overline{\mu\mathbf{t_n}.H_*}}/_{\mathbf{t}_{n'}}\right\} = \overline{\mu\mathbf{t_n}.H_*\{{}^{\mu\mathbf{t_1}\ldots\mu\mathbf{t_{n'}}.\mu\mathbf{t_n}.H_*}/_{\mathbf{t_1}}\}\ldots\{{}^{\mu\mathbf{t_{n'}}.\mu\mathbf{t_n}.H_*}/_{\mathbf{t}_{n'}}\}} \quad \text{(by the i.h.)}$$
$$\mu\mathbf{t_n}.\overline{H_*}\left\{{}^{\mu\mathbf{t_1}\ldots\mu\mathbf{t_{n'}}.\mu\mathbf{t_n}.\overline{H_*}}/_{\mathbf{t}_1}\right\}\ldots\left\{{}^{\mu\mathbf{t_{n'}}.\mu\mathbf{t_n}.\overline{H_*}}/_{\mathbf{t}_{n'}}\right\} = \mu\mathbf{t_n}.\overline{H_*}\left\{{}^{\mu\mathbf{t_1}\ldots\mu\mathbf{t_{n'}}.\mu\mathbf{t_n}.\overline{H_*}}/_{\mathbf{t}_1}\right\}\ldots\left\{{}^{\mu\mathbf{t_{n'}}.\mu\mathbf{t_n}.\overline{H_*}}/_{\mathbf{t}_{n'}}\right\} \quad \text{(by Def. 2.8)}$$

and we obtain the thesis by further unfolding $\mu\mathbf{t_n}\ldots$ in the LHS and RHS above.

We conclude observing that, when $n = m$ (i.e., H is completely unfolded, bringing at the top-level $H_\diamond \neq \mu\mathbf{t'}\ldots$) we have proved $\mathrm{unf}\left(\overline{H}\right) = \overline{\mathrm{unf}(H)}$. ◀

▶ **Proposition B.10.** *For all session types S and roles p, $\mathrm{unf}(S)\restriction\mathrm{p} = \mathrm{unf}(S\restriction\mathrm{p})$.*

Proof. Let $S = \mu\mathbf{t_1}\ldots\mu\mathbf{t_m}.S_\diamond$, with $S_\diamond \neq \mu\mathbf{t'}\ldots$. We first prove the following statement:

$$\forall n \in 0..m: \quad (S_*\{{}^{\mu\mathbf{t_1}\ldots\mu\mathbf{t_n}.S_*}/_{\mathbf{t}_1}\}\ldots\{{}^{\mu\mathbf{t_n}.S_*}/_{\mathbf{t}_n}\})\restriction\mathrm{p} = (S_*\restriction\mathrm{p})\{{}^{\mu\mathbf{t_1}\ldots\mu\mathbf{t_n}.(S_*\restriction\mathrm{p})}/_{\mathbf{t}_1}\}\ldots\{{}^{\mu\mathbf{t_n}.(S_*\restriction\mathrm{p})}/_{\mathbf{t}_n}\}$$
$$\text{where } S_* = \mu\mathbf{t_{n+1}}\ldots\mu\mathbf{t_m}.S_\diamond$$

We proceed by induction on n. The base case $n = 0$ is trivial, while in the inductive case $n = n' + 1$ we have:

$$\begin{aligned}&(\mu\mathbf{t_n}.S_*\{{}^{\mu\mathbf{t_1}\ldots\mu\mathbf{t_{n'}}.\mu\mathbf{t_n}.S_*}/_{\mathbf{t}_1}\}\ldots\{{}^{\mu\mathbf{t_{n'}}.\mu\mathbf{t_n}.S_*}/_{\mathbf{t}_{n'}}\})\restriction\mathrm{p} = \\ &(\mu\mathbf{t_n}.S_*\restriction\mathrm{p})\{{}^{\mu\mathbf{t_1}\ldots\mu\mathbf{t_{n'}}.(\mu\mathbf{t_n}.S_*\restriction\mathrm{p})}/_{\mathbf{t}_1}\}\ldots\{{}^{\mu\mathbf{t_{n'}}.(\mu\mathbf{t_n}.S_*\restriction\mathrm{p})}/_{\mathbf{t}_{n'}}\}\end{aligned} \quad \text{(by the i.h.)} \tag{19}$$

At this point we have two cases, based on the partial projection of recursive types in Def. 2.9. If $S_*\restriction\mathrm{p} \neq \mathbf{t'}$ (for all $\mathbf{t'}$), then $(\mu\mathbf{t_n}.S_*\restriction\mathrm{p}) = \mu\mathbf{t_n}.(S_*\restriction\mathrm{p})$, and we get:

$$\begin{aligned}&\mu\mathbf{t_n}.\left((S_*\{{}^{\mu\mathbf{t_1}\ldots\mu\mathbf{t_{n'}}.\mu\mathbf{t_n}.S_*}/_{\mathbf{t}_1}\}\ldots\{{}^{\mu\mathbf{t_{n'}}.\mu\mathbf{t_n}.S_*}/_{\mathbf{t}_{n'}}\})\restriction\mathrm{p}\right) = \\ &(\mu\mathbf{t_n}.S_*\restriction\mathrm{p})\{{}^{\mu\mathbf{t_1}\ldots\mu\mathbf{t_{n'}}.(\mu\mathbf{t_n}.S_*\restriction\mathrm{p})}/_{\mathbf{t}_1}\}\ldots\{{}^{\mu\mathbf{t_{n'}}.(\mu\mathbf{t_n}.S_*\restriction\mathrm{p})}/_{\mathbf{t}_{n'}}\}\end{aligned} \quad \text{(by Def. 2.9)}$$

and we obtain the thesis by further unfolding $\mu\mathbf{t_n}\ldots$ in the LHS and RHS above.

Otherwise, if $S_*\restriction\mathrm{p} = \mathbf{t'}$ (for some $\mathbf{t'}$), then $(\mu\mathbf{t_n}.S_*\restriction\mathrm{p}) = \mathbf{end}$. Therefore, on the RHS of (19) we have:

$$(\mu\mathbf{t_n}.S_*\restriction\mathrm{p})\{{}^{\mu\mathbf{t_1}\ldots\mu\mathbf{t_{n'}}.(\mu\mathbf{t_n}.S_*\restriction\mathrm{p})}/_{\mathbf{t}_1}\}\ldots\{{}^{\mu\mathbf{t_{n'}}.(\mu\mathbf{t_n}.S_*\restriction\mathrm{p})}/_{\mathbf{t}_{n'}}\} = \mathbf{end}\{{}^{\mu\mathbf{t_1}\ldots\mu\mathbf{t_{n'}}.\mathbf{end}}/_{\mathbf{t}_1}\}\ldots\{{}^{\mu\mathbf{t_{n'}}.\mathbf{end}}/_{\mathbf{t}_{n'}}\} = \mathbf{end} \tag{20}$$

Moreover, since in this case we must have $\mathbf{t'} \in \mathrm{fv}(S_*)$, then we also have one of the following:

$$(S_*\{{}^{\mu\mathbf{t_1}\ldots\mu\mathbf{t_{n'}}.\mu\mathbf{t_n}.S_*}/_{\mathbf{t}_1}\}\ldots\{{}^{\mu\mathbf{t_{n'}}.\mu\mathbf{t_n}.S_*}/_{\mathbf{t}_{n'}}\})\restriction\mathrm{p} = \mathbf{t}_n \quad \text{or} \tag{21}$$
$$(S_*\{{}^{\mu\mathbf{t_1}\ldots\mu\mathbf{t_{n'}}.\mu\mathbf{t_n}.S_*}/_{\mathbf{t}_1}\}\ldots\{{}^{\mu\mathbf{t_{n'}}.\mu\mathbf{t_n}.S_*}/_{\mathbf{t}_{n'}}\})\restriction\mathrm{p} = \mathbf{end} \quad (\text{if } \mathbf{t'} \neq \mathbf{t}_n, \text{ i.e., } \mathbf{t'} = \mathbf{t}_i \text{ for some } i \in 1..n') \tag{22}$$

Now, if (21) holds, we get:

$$(\mu\mathbf{t_n}.S_*\{{}^{\mu\mathbf{t_1}\ldots\mu\mathbf{t_{n'}}.\mu\mathbf{t_n}.S_*}/_{\mathbf{t}_1}\}\ldots\{{}^{\mu\mathbf{t_{n'}}.\mu\mathbf{t_n}.S_*}/_{\mathbf{t}_{n'}}\})\restriction\mathrm{p} = \mathbf{end} \quad \text{(by Def. 2.9)}$$

that, together with (20), by further (vacuously) unfolding both terms once, gives us $\mathbf{end} = \mathbf{end}$ (which is our thesis).

Otherwise, if (22) holds, we get:

$$(\mu\mathbf{t_n}.S_*\{{}^{\mu\mathbf{t_1}\ldots\mu\mathbf{t_{n'}}.\mu\mathbf{t_n}.S_*}/_{\mathbf{t}_1}\}\ldots\{{}^{\mu\mathbf{t_{n'}}.\mu\mathbf{t_n}.S_*}/_{\mathbf{t}_{n'}}\})\restriction\mathrm{p} = \mu\mathbf{t_n}.\mathbf{end} \quad \text{(by Def. 2.9)}$$

that, together with (20), by further unfolding both terms once, gives us $\mathbf{end} = \mathbf{end}$ (which is our thesis).

We conclude observing that, when $n = m$ (i.e., H is completely unfolded, bringing at the top-level $H_\diamond \neq \mu\mathbf{t'}\ldots$) we have proved $\mathrm{unf}(S)\restriction\mathrm{p} = \mathrm{unf}(S\restriction\mathrm{p})$. ◀

▶ **Proposition B.11.** *$S \leqslant_{\mathsf{S}} S'$ implies* $\text{roles}(S) = \text{roles}(S')$.

Proof. Assume $S \leqslant_{\mathsf{S}} S'$. By contradiction, assume that $\text{roles}(S) \neq \text{roles}(S')$, i.e., $\exists \mathsf{q} \in S'$ but $\mathsf{q} \notin S$ (the proof for $S \ni \mathsf{p} \notin S'$ is similar, but uses (T-Sel) in the following). We can observe that S, S' cannot be related by (S-End) (otherwise we would have $S = S' = \mathbf{end}$, and thus $\mathsf{q} \notin \text{roles}(S') = \emptyset$). Moreover, q cannot be the top-level role of *any* rule applied in the derivation of $S \leqslant_{\mathsf{S}} S'$ (otherwise we would have $S' \ni \mathsf{q} \in S$). Hence, the derivation for $S \leqslant_{\mathsf{S}} S'$ must have at least one occurrence of (T-Brch) with some (but not all) branches on the RHS containing q, i.e.:

- $\mathsf{p} \&_{i \in I} ?l_i(U_i).S_i \leqslant_{\mathsf{S}} \mathsf{p} \&_{i \in I \cup J} ?l_i(U'_i).S'_i$, with $\mathsf{q} \in S'_k$ for some $k \in J$, and $\mathsf{q} \notin S'_i$ for some $i \in I$ (otherwise, q would necessarily be the top-level role at some point in the derivation). Then, we have two cases: either $S'_k \upharpoonright \mathsf{q}$ is *not* defined, or $S'_k \upharpoonright \mathsf{q}$ is defined, but $S'_k \upharpoonright \mathsf{q} \neq \mathbf{end}$ — which implies that it cannot be merged with $S'_i \upharpoonright \mathsf{q} = \mathbf{end}$. In both cases, we obtain that $S' \upharpoonright \mathsf{q}$ is *not* defined, which violates clause (i) of Def. 2.10, and therefore implies $S \not\leqslant_{\mathsf{S}} S'$ — contradiction.

◀

▶ **Proposition B.12.** *Let S, S' be closed session types. If $S \leqslant_{\mathsf{S}} S'$, then for all p also $S \upharpoonright \mathsf{p} \leqslant_{\mathsf{P}} S' \upharpoonright \mathsf{p}$.*

Proof. By Def. 2.10 (clause (i)), we already know that $\forall \mathsf{p} \in (\text{roles}(S) \cup \text{roles}(S'))$, we have $S \upharpoonright \mathsf{p} \leqslant_{\mathsf{P}} S' \upharpoonright \mathsf{p}$. Since p in the statement is universally quantified, we are left to prove it for all $\mathsf{p} \notin (\text{roles}(S) \cup \text{roles}(S'))$. By Proposition B.11, we know that $\mathsf{p} \in S$ iff $\mathsf{p} \in S'$: hence, by Def. 2.9, we obtain that for all $\mathsf{p} \notin (\text{roles}(S) \cup \text{roles}(S'))$, $S \upharpoonright \mathsf{p} = \mathbf{end} \leqslant_{\mathsf{P}} \mathbf{end} = S' \upharpoonright \mathsf{p}$. ◀

▶ **Proposition B.13.** *If $S \upharpoonright \mathsf{q}$ is defined and closed, and either* $\text{unf}(S) = \mathsf{p} \&_{i \in I} ?l_i(U_i).S_i$ *or* $\text{unf}(S) = \mathsf{p} \oplus_{i \in I} !l_i(U_i).S_i$ *with $\mathsf{p} \neq \mathsf{q}$, then $\forall k \in I$: $S \upharpoonright \mathsf{q} \leqslant_{\mathsf{P}} S_k \upharpoonright \mathsf{q}$.*

Proof. Assume that $S \upharpoonright \mathsf{q}$ is defined and closed. We have two cases:

- $\text{unf}(S) = \mathsf{p} \&_{i \in I} ?l_i(U_i).S_i$. By Def. 2.9, $\text{unf}(S) \upharpoonright \mathsf{q} = \bigsqcap_{i \in I} (S_i \upharpoonright \mathsf{q})$; moreover, by Lemma B.7, $\forall k \in I : \bigsqcap_{i \in I} (S_i \upharpoonright \mathsf{q}) \leqslant_{\mathsf{OP}} S_k \upharpoonright \mathsf{q}$. Noticing that $\forall k \in I : S_k \upharpoonright \mathsf{q}$ is closed (otherwise, $S \upharpoonright \mathsf{q}$ would not be closed), by Lemma B.6 we get $\forall k \in I : \text{unf}(S) \upharpoonright \mathsf{q} \leqslant_{\mathsf{P}} S_k \upharpoonright \mathsf{q}$, and therefore (by Proposition B.10) $\text{unf}(S \upharpoonright \mathsf{q}) \leqslant_{\mathsf{P}} S_k \upharpoonright \mathsf{q}$; then, by Proposition B.8, we conclude $S \upharpoonright \mathsf{q} \leqslant_{\mathsf{P}} S_k \upharpoonright \mathsf{q}$;
- $S = \mathsf{p} \oplus_{i \in I} !l_i(U_i).S_i$. By Def. 2.9, $S \upharpoonright \mathsf{q} = \bigsqcap_{i \in I} (S_i \upharpoonright \mathsf{q})$: the proof is similar to the previous case.

◀

▶ **Proposition B.14.** *If $(\Gamma, x : U)$ is consistent, then Γ is consistent.*

Proof. The proof is straightforward, by noticing that Def. 2.11 on consistency does not depend on $x : U$. ◀

▶ **Proposition B.15.** *If $(\Gamma, s[\mathsf{p}] : S)$ is consistent, then Γ is consistent.*

Proof. Assume that $\Gamma, s[\mathsf{p}] : S$ is consistent. By Def. 2.11, it means that $\forall s[\mathsf{q}], s[\mathsf{r}] \in \text{dom}(\Gamma, s[\mathsf{p}] : S)$: $\mathsf{q} \neq \mathsf{r}$ implies $\overline{\Gamma(s[\mathsf{q}]) \upharpoonright \mathsf{r}} \leqslant_{\mathsf{P}} \Gamma(s[\mathsf{r}]) \upharpoonright \mathsf{q}$. Since $\text{dom}(\Gamma) = \text{dom}(\Gamma, s[\mathsf{p}] : S) \setminus \{s[\mathsf{p}]\}$, we also have that $\forall s[\mathsf{q}], s[\mathsf{r}] \in \text{dom}(\Gamma)$: $\mathsf{q} \neq \mathsf{r}$ implies $\overline{\Gamma(s[\mathsf{q}]) \upharpoonright \mathsf{r}} \leqslant_{\mathsf{P}} \Gamma(s[\mathsf{r}]) \upharpoonright \mathsf{q}$. Hence, by Def. 2.11, we conclude that Γ is consistent. ◀

▶ **Corollary B.16.** *If (Γ_1, Γ_2) is consistent, then Γ_1 and Γ_2 are consistent.*

Proof. By repeatedly applying Proposition B.14 and Proposition B.15 to remove all entries of Γ_1 from (Γ_1, Γ_2), we prove that Γ_2 is consistent. With the symmetric procedure, we prove that Γ_1 is consistent. ◀

▶ **Corollary B.17.** *If $(\Gamma_1 \circ \Gamma_2)$ is consistent, then Γ_1 and Γ_2 are consistent.*

Proof. Similar to the proof of Cor. B.16, except that we might have entries of the form $x : B$ (which are not relevant for consistency, as per Def. 2.11) appearing in both Γ_1 and Γ_2. ◀

▶ **Proposition B.18.** *If $\Gamma, s[\mathsf{p}] : S$ is consistent and $S \leqslant_{\mathsf{S}} S'$, then $\Gamma, s[\mathsf{p}] : S'$ is consistent.*

Proof. Assume that $\Gamma, s[\mathrm{p}]:S$ is consistent, and take any S' such that $S \leqslant_{\mathsf{S}} S'$. By Def. 2.11, we know that $\forall s[\mathrm{q}]:S_{\mathrm{q}} \in \mathrm{dom}(\Gamma) : \overline{S_{\mathrm{q}} \upharpoonright \mathrm{p}} \leqslant_{\mathsf{P}} S \upharpoonright \mathrm{q}$; moreover, by Proposition B.12, we have $\forall \mathrm{q} : S \upharpoonright \mathrm{q} \leqslant_{\mathsf{P}} S \upharpoonright \mathrm{q}$. Therefore, by transitivity of $\leqslant_{\mathsf{P}}$, we also have $\forall s[\mathrm{q}]:S_{\mathrm{q}} \in \mathrm{dom}(\Gamma) : \overline{S_{\mathrm{q}} \upharpoonright \mathrm{p}} \leqslant_{\mathsf{P}} S' \upharpoonright \mathrm{q}$, and by Def. 2.11, we conclude that $\Gamma, s[\mathrm{p}]:S'$ is consistent. ◀

▶ **Corollary B.19.** *If Γ_1, Γ_2 is consistent and $\Gamma_2 \leqslant_{\mathsf{S}} \Gamma_2'$, then Γ_1, Γ_2' is consistent.*

Proof. By induction on the size of Γ_2. The base case ($\Gamma_2 = \varnothing$) is trivial, while the inductive case is proved by the induction hypothesis, and Proposition B.18. ◀

▶ **Corollary B.20.** *If $\Gamma_1 \circ \Gamma_2$ is consistent and $\Gamma_2 \leqslant_{\mathsf{S}} \Gamma_2'$, then $\Gamma_1 \circ \Gamma_2'$ is consistent.*

Proof. Similar to the proof of Cor. B.19, except that Γ_1, Γ_2 and Γ_2' can have (possibly shared) entries mapping some x to a basic type (which are not relevant for consistency, as per Def. 2.11). ◀

▶ **Proposition B.21.** *If $\Gamma \to^* \Gamma'$, then $\mathrm{dom}(\Gamma) = \mathrm{dom}(\Gamma')$.*

Proof. We first verify the following statement, by induction on the size of $\mathrm{dom}(\Gamma)$:

$$\Gamma \to \Gamma' \qquad \text{implies} \qquad \mathrm{dom}(\Gamma) = \mathrm{dom}(\Gamma') \tag{23}$$

Then, we can prove the main statement, by induction on the length of the sequence of reductions in $\Gamma \to^* \Gamma'$. The base case is trivial (we have 0 reductions, and $\Gamma = \Gamma'$), while in the inductive case, we apply the induction hypothesis and (23). ◀

▶ **Lemma B.22.** *If $\Gamma \to \Gamma'$ and Γ is consistent (resp. complete), then so is Γ'.*

Proof. Assume that Γ is consistent. We proceed by induction on the derivation of $\Gamma \to \Gamma'$, as per Def. 2.15:

- base case $\Gamma = s[\mathrm{p}]:S_{\mathrm{p}}, s[\mathrm{q}]:S_{\mathrm{q}} \to s[\mathrm{p}]:S_k, s[\mathrm{q}]:S_k' = \Gamma'$, with $\mathrm{unf}(S_{\mathrm{p}}) = \mathrm{q} \oplus_{i \in I} !l_i(U_i).S_i$, $\mathrm{unf}(S_{\mathrm{q}}) = \mathrm{p} \,\&_{i \in I \cup J} ?l_i(U_i').S_i'$, $k \in I$ and $U_k \leqslant_{\mathsf{S}} U_k'$. We observe:

$$\begin{array}{rcll}
\overline{S_{\mathrm{p}} \upharpoonright \mathrm{q}} & \leqslant_{\mathsf{P}} & S_{\mathrm{p}} \upharpoonright \mathrm{q} & \text{(by hypothesis and Def. 2.11)}\\
\mathrm{unf}\big(\overline{S_{\mathrm{p}} \upharpoonright \mathrm{q}}\big) & \leqslant_{\mathsf{P}} & \mathrm{unf}(S_{\mathrm{q}} \upharpoonright \mathrm{p}) & \text{(by (S-Par}\mu\text{L) and (S-Par}\mu\text{R))}\\
\overline{\mathrm{unf}(S_{\mathrm{p}} \upharpoonright \mathrm{q})} & \leqslant_{\mathsf{P}} & \mathrm{unf}(S_{\mathrm{q}} \upharpoonright \mathrm{p}) & \text{(by Proposition B.9)}\\
\overline{\mathrm{unf}(S_{\mathrm{p}}) \upharpoonright \mathrm{q}} & \leqslant_{\mathsf{P}} & \mathrm{unf}(S_{\mathrm{q}}) \upharpoonright \mathrm{p} & \text{(by Proposition B.10)}\\
\overline{(\mathrm{q} \oplus_{i \in I} !l_i(U_i).S_i) \upharpoonright \mathrm{q}} & \leqslant_{\mathsf{P}} & (\mathrm{p} \,\&_{i \in I \cup J} ?l_i(U_i').S_i') \upharpoonright \mathrm{p} & \text{(by hypothesis)}\\
\overline{\oplus_{i \in I} !l_i(U_i).(S_i \upharpoonright \mathrm{q})} & \leqslant_{\mathsf{P}} & \&_{i \in I \cup J} ?l_i(U_i').(S_i' \upharpoonright \mathrm{p}) & \text{(by Def. 2.9)}\\
\&_{i \in I} ?l_i(U_i).\overline{(S_i \upharpoonright \mathrm{q})} & \leqslant_{\mathsf{P}} & \&_{i \in I \cup J} ?l_i(U_i').(S_i' \upharpoonright \mathrm{p}) & \text{(by Def. 2.8)}\\
\forall k \in I : \;\; \overline{S_k \upharpoonright \mathrm{q}} & \leqslant_{\mathsf{P}} & S_k' \upharpoonright \mathrm{p} & \text{(by (S-ParBrch))}
\end{array}$$

  and we conclude that Γ' is consistent;

- inductive case $\Gamma = \Gamma_1, c:U \to \Gamma_1', c:U' = \Gamma'$, with $U \leqslant_{\mathsf{S}} U'$. In this case, c might be either a variable x, or a channel with role $s[\mathrm{r}]$. If $c = x$, the thesis holds trivially by the induction hypothesis, since $x:U$ and $x:U'$ are not relevant for consistency (Def. 2.11). Instead, if $c = s[\mathrm{r}]$, both U and U' must be session types (by Def. 2.11). Therefore, we have $\Gamma = \Gamma_1, s[\mathrm{r}]:S_{\mathrm{r}} \to \Gamma_1', s[\mathrm{r}]:S_{\mathrm{r}}' = \Gamma'$, with:

$$\Gamma_1 \to \Gamma_1' \tag{24}$$

$$S_{\mathrm{r}} \leqslant_{\mathsf{S}} S_{\mathrm{r}}' \tag{25}$$

  From (24), we can observe that Γ_1, Γ_1' must have the form:

$$\Gamma_1 \;=\; s[\mathrm{p}]:S_{\mathrm{p}},\, s[\mathrm{q}]:S_{\mathrm{q}},\, \Gamma_0 \tag{26}$$

$$\Gamma_1' \;=\; s[\mathrm{p}]:S_{\mathrm{p}}',\, s[\mathrm{q}]:S_{\mathrm{q}}',\, \Gamma_0' \tag{27}$$

$$\text{where}\;\; s[\mathrm{p}]:S_{\mathrm{p}}, s[\mathrm{q}]:S_{\mathrm{q}} \to s[\mathrm{p}]:S_{\mathrm{p}}', s[\mathrm{q}]:S_{\mathrm{q}}' \;\;\text{and}\;\; \Gamma_0 \leqslant_{\mathsf{S}} \Gamma_0' \tag{28}$$

Therefore:

$$\Gamma \;=\; \Gamma_1, s[\mathtt{r}]:S_{\mathtt{r}} \;=\; s[\mathtt{p}]:S_{\mathtt{p}},\, s[\mathtt{q}]:S_{\mathtt{q}},\, \Gamma_0,\, s[\mathtt{r}]:S_{\mathtt{r}} \;\text{ is consistent} \quad \text{(by hypothesis and (26))} \tag{29}$$

$$s[\mathtt{p}]:S_{\mathtt{p}},\, s[\mathtt{q}]:S_{\mathtt{q}},\, \Gamma'_0,\, s[\mathtt{r}]:S'_{\mathtt{r}} \;\text{ is consistent} \quad \text{(from (29), (28), and Cor. B.19)} \tag{30}$$

$$\Gamma'_0,\, s[\mathtt{r}]:S'_{\mathtt{r}} \;\text{ is consistent} \quad \text{(by (30) and Cor. B.16)} \tag{31}$$

$$s[\mathtt{p}]:S'_{\mathtt{p}},\, s[\mathtt{q}]:S'_{\mathtt{q}},\, \Gamma'_0 \;\text{ is consistent} \quad \text{(by (27), (24) and the induction hypothesis)} \tag{32}$$

$$s[\mathtt{p}]:S'_{\mathtt{p}},\, s[\mathtt{q}]:S'_{\mathtt{q}} \;\text{ and }\; s[\mathtt{p}]:S'_{\mathtt{p}},\, \Gamma'_0 \;\text{ and }\; s[\mathtt{q}]:S'_{\mathtt{q}},\, \Gamma'_0 \;\text{ are consistent} \quad \text{(by (32) and Cor. B.16)} \tag{33}$$

Hence, to prove that $\Gamma' = s[\mathtt{p}]:S'_{\mathtt{p}}, s[\mathtt{q}]:S'_{\mathtt{q}}, \Gamma'_0, s[\mathtt{r}]:S'_{\mathtt{r}}$ is consistent, from (27) and (33) we can see that we are left to prove that both $s[\mathtt{p}]:S'_{\mathtt{p}},\, s[\mathtt{r}]:S'_{\mathtt{r}}$ and $s[\mathtt{q}]:S'_{\mathtt{q}},\, s[\mathtt{r}]:S'_{\mathtt{r}}$ are consistent. By Def. 2.11, it means that we need to prove:

$$\overline{S'_{\mathtt{p}} \upharpoonright \mathtt{r}} \leqslant_{\mathsf{P}} S'_{\mathtt{r}} \upharpoonright \mathtt{p} \qquad \text{and} \qquad \overline{S'_{\mathtt{q}} \upharpoonright \mathtt{r}} \leqslant_{\mathsf{P}} S'_{\mathtt{r}} \upharpoonright \mathtt{q} \tag{34}$$

From (28) and Def. 2.15, we have two sub-cases:

- $\mathrm{unf}(S_{\mathtt{p}}) = \mathtt{q} \oplus_{i \in I} !l_i(U_i).S_i$ and $\mathrm{unf}(S_{\mathtt{q}}) = \mathtt{p} \,\&_{i \in I \cup J}\, ?l_i(U'_i).S'_i$. Then:

$$\text{for some } k \in I,\; S'_{\mathtt{p}} = S_k \text{ and } S'_{\mathtt{q}} = S'_k \quad \text{(by Def. 2.15)} \tag{35}$$

$$\mathrm{unf}(S_{\mathtt{p}}) \upharpoonright \mathtt{r} \leqslant_{\mathsf{P}} S'_{\mathtt{p}} \upharpoonright \mathtt{r} \;\text{ and }\; \mathrm{unf}(S_{\mathtt{q}}) \upharpoonright \mathtt{r} \leqslant_{\mathsf{P}} S'_{\mathtt{q}} \upharpoonright \mathtt{r} \quad \text{(by (35) and Proposition B.13)} \tag{36}$$

$$\overline{S'_{\mathtt{p}} \upharpoonright \mathtt{r}} \leqslant_{\mathsf{P}} \overline{\mathrm{unf}(S_{\mathtt{p}}) \upharpoonright \mathtt{r}} \;\text{ and }\; \overline{S'_{\mathtt{q}} \upharpoonright \mathtt{r}} \leqslant_{\mathsf{P}} \overline{\mathrm{unf}(S_{\mathtt{q}}) \upharpoonright \mathtt{r}} \quad \text{(by (36) and Proposition D.1)} \tag{37}$$

$$\overline{\mathrm{unf}(S_{\mathtt{p}}) \upharpoonright \mathtt{r}} \leqslant_{\mathsf{P}} S_{\mathtt{r}} \upharpoonright \mathtt{p} \;\text{ and }\; \overline{\mathrm{unf}(S_{\mathtt{q}}) \upharpoonright \mathtt{r}} \leqslant_{\mathsf{P}} S_{\mathtt{r}} \upharpoonright \mathtt{q} \quad \text{(by hypothesis (consistency of } \Gamma\text{) and Def. 2.11)} \tag{38}$$

$$S_{\mathtt{r}} \upharpoonright \mathtt{p} \leqslant_{\mathsf{P}} S'_{\mathtt{r}} \upharpoonright \mathtt{p} \;\text{ and }\; S_{\mathtt{r}} \upharpoonright \mathtt{q} \leqslant_{\mathsf{P}} S'_{\mathtt{r}} \upharpoonright \mathtt{q} \quad \text{(by (25) and Proposition B.12)} \tag{39}$$

$$\overline{S'_{\mathtt{p}} \upharpoonright \mathtt{r}} \leqslant_{\mathsf{P}} S'_{\mathtt{r}} \upharpoonright \mathtt{p} \;\text{ and }\; \overline{S'_{\mathtt{q}} \upharpoonright \mathtt{r}} \leqslant_{\mathsf{P}} S'_{\mathtt{r}} \upharpoonright \mathtt{p} \quad \text{(by (37), (38), (39) and transitivity of } \leqslant_{\mathsf{P}}) \tag{40}$$

- $\mathrm{unf}(S_{\mathtt{q}}) = \mathtt{p} \,\&_{i \in I \cup J}\, ?l_i(U'_i).S'_i$ and $\mathrm{unf}(S_{\mathtt{p}}) = \mathtt{q} \oplus_{i \in I} !l_i(U_i).S_i$. The proof is symmetric w.r.t. the previous case.

Hence, we have proved (34); from (34), (31) and (32), by Def. 2.11 we conclude that Γ' is consistent.

For the second part of the statement, assume that Γ is complete: we can prove that Γ' is also complete by induction on the derivation of $\Gamma \to \Gamma'$, as per Def. 2.15. The key observation if that for each $s[\mathtt{p}] \in \mathrm{dom}(\Gamma)$, we also have $s[\mathtt{p}] \in \mathrm{dom}(\Gamma')$ (by Proposition B.21), and $\mathrm{roles}(\Gamma'(s[\mathtt{p}])) \subseteq \mathrm{roles}(\Gamma(s[\mathtt{p}]))$. ◀

▶ **Corollary B.23.** *If Γ_1, Γ_2 is consistent and $\Gamma_1 \to^* \Gamma'_1$, then Γ'_1, Γ_2 is consistent.*

Proof. Assume all the hypotheses, and let n be the length of the sequence of reductions in $\Gamma_1 \to^* \Gamma'_1$. In the base case ($n = 0$) the thesis holds trivially. In the inductive case $n = n' + 1$, we have:

$$\Gamma_1 \underbrace{\to \cdots \to}_{n' \text{ times}} \Gamma^*_1 \;\to\; \Gamma'_1$$

and by the induction hypothesis, Γ^*_1, Γ_2 is consistent. This implies that Γ'_1, Γ_2 is consistent: we prove such a fact with a further induction on the size of Γ_2. In the base case ($\Gamma_2 = \varnothing$) we conclude immediately by Lemma B.22. In the inductive case we have $\Gamma_2 = \Gamma_0, c{:}U$; by applying the induction hypothesis we get that Γ'_1, Γ_0 is consistent, and we examine the shape of the additional entry $c{:}U$ and its consistency w.r.t. Γ'_1, Γ_0, similarly to the inductive case in the proof of Lemma B.22. In all cases, we conclude that Γ'_1, Γ_2 is consistent. ◀

▶ **Corollary B.24.** *If $\Gamma_1 \circ \Gamma_2$ is consistent and $\Gamma_1 \to^* \Gamma'_1$, then $\Gamma'_1 \circ \Gamma_2$ is consistent.*

Proof. Similar to the proof of Cor. B.23, except that Γ_1, Γ'_1 and Γ_2 can have (possibly shared) entries mapping some x to a basic type (which are not relevant for consistency, Def. 2.11). ◀

▶ **Proposition B.25.** *For all multiparty session processes P, P', if $\Theta \cdot \Gamma \vdash P$ and $P \equiv P'$, then $\Theta \cdot \Gamma \vdash P'$.*

Proof. The proof proceeds by induction on the structural congruence $\equiv$, defined in Fig. 12. ◀

▶ **Lemma B.26** (Substitution lemma)**.** *If $\Theta \cdot \Gamma, x{:}U \vdash P$, $\Gamma' \vdash v{:}U$ and $\Gamma \circ \Gamma'$ is consistent, then $\Theta \cdot \Gamma \circ \Gamma' \vdash P\{^v/_x\}$.*

Proof. The proof is by induction on the typing derivations, with a case analysis on the last rule applied. ◀

▶ **Definition B.27** (Context subtyping)**.** *For all multiparty session typing contextes Γ_S, Γ'_S, the relation $\Gamma_S \leqslant_S \Gamma'_S$ holds iff $\operatorname{dom}(\Gamma_S) = \operatorname{dom}(\Gamma'_S)$ and $\forall c \in \operatorname{dom}(\Gamma_S) : \Gamma_S(c) \leqslant_S \Gamma'_S(c)$. We define the following multiparty session typing rule, corresponding to 0 or more consecutive applications of* (T-Sub)*:*

$$\frac{\Theta \cdot \Gamma_S \vdash P \qquad \Gamma'_S \leqslant_S \Gamma_S}{\Theta \cdot \Gamma'_S \vdash P}\ \text{(T-MSub)}$$

Proposition B.30 below will allow us to consider only one form of "normalised" typing derivation for multiparty session processes (the first one in the statement), where (possibly vacuous) instances of (T-MSub) appear as premises of (T-Par), but not *vice versa*. This allows rewrite a typing derivation by "pushing" (T-MSub) towards the leaves, until reaching a sub-process that cannot be further decomposed using the parallel composition $|$.

▶ **Proposition B.28.** *If $\Theta \cdot \Gamma \vdash P_1 \mid P_2$, then $\exists \Gamma_1, \Gamma'_1, \Gamma_2, \Gamma'_2$ such that $\Gamma = \Gamma_1 \circ \Gamma_2$, $\Gamma_1 \leqslant_S \Gamma'_1$, $\Gamma_2 \leqslant_S \Gamma'_2$, $\Theta \cdot \Gamma'_1 \vdash P_1$ and $\Theta \cdot \Gamma'_2 \vdash P_2$. Moreover:*

$$\text{(T-Par)}\ \frac{\text{(T-MSub)}\ \dfrac{\Theta \cdot \Gamma'_1 \vdash P_1 \qquad \Gamma_1 \leqslant_S \Gamma'_1}{\Theta \cdot \Gamma_1 \vdash P_1} \qquad \text{(T-MSub)}\ \dfrac{\Theta \cdot \Gamma'_2 \vdash P_2 \qquad \Gamma_2 \leqslant_S \Gamma'_2}{\Theta \cdot \Gamma_2 \vdash P_2}}{\Theta \cdot \Gamma_1 \circ \Gamma_2 \vdash P_1 \mid P_2}$$

$$\textit{iff}\quad \frac{\text{(T-Par)}\ \dfrac{\Theta \cdot \Gamma'_1 \vdash P_1 \qquad \Theta \cdot \Gamma'_2 \vdash P_2}{\Theta \cdot \Gamma'_1 \circ \Gamma'_2 \vdash P_1 \mid P_2} \qquad \Gamma_1 \circ \Gamma_2 \leqslant_S \Gamma'_1 \circ \Gamma'_2}{\Theta \cdot \Gamma_1 \circ \Gamma_2 \vdash P_1 \mid P_2}\ \text{(T-MSub)}$$

$$\textit{iff}\quad \frac{\text{(T-Par)}\ \dfrac{\Theta \cdot \Gamma'_1 \vdash P_1 \qquad \dfrac{\Theta \cdot \Gamma'_2 \vdash P_2 \qquad \Gamma_2 \leqslant_S \Gamma'_2}{\Theta \cdot \Gamma_2 \vdash P_2}\ \text{(T-MSub)}}{\Theta \cdot \Gamma'_1 \circ \Gamma_2 \vdash P_1 \mid P_2} \qquad \Gamma_1 \circ \Gamma_2 \leqslant_S \Gamma'_1 \circ \Gamma_2}{\Theta \cdot \Gamma_1 \circ \Gamma_2 \vdash P_1 \mid P_2}\ \text{(T-MSub)}$$

$$\textit{iff}\quad \frac{\text{(T-Par)}\ \dfrac{\text{(T-MSub)}\ \dfrac{\Theta \cdot \Gamma'_1 \vdash P_1 \qquad \Gamma_1 \leqslant_S \Gamma'_1}{\Theta \cdot \Gamma_1 \vdash P_1} \qquad \Theta \cdot \Gamma'_2 \vdash P_2}{\Theta \cdot \Gamma_1 \circ \Gamma'_2 \vdash P_1 \mid P_2} \qquad \Gamma_1 \circ \Gamma_2 \leqslant_S \Gamma_1 \circ \Gamma'_2}{\Theta \cdot \Gamma_1 \circ \Gamma_2 \vdash P_1 \mid P_2}\ \text{(T-MSub)}$$

Proof. The first part of the statement is straightforward by inversion of (T-PAR) (by Def. B.27). and adding a (possibly vacuous) instance of (T-SUB), as in the last case in the statement after "moreover". The "iff" relations among the typing derivations are also straightforward: the hypotheses of one derivation imply all the others, and if one hypothesis is falsified, none of the derivations hold. ◀

▶ **Proposition B.29.** *$\Theta \cdot \Gamma_1 \vdash (\boldsymbol{\nu} s{:}\Gamma_2)P$, then $\exists \Gamma_1', \Gamma_2'$ such that $\Gamma_1 \leqslant_{\mathsf{S}} \Gamma_1'$, $\Gamma_2 \leqslant_{\mathsf{S}} \Gamma_2'$, and $\Gamma_1' \circ \Gamma_2' \vdash P$. Moreover:*

$$\text{(T-RES)}\dfrac{\text{(T-MSUB)}\dfrac{\Theta \cdot \Gamma_1' \circ \Gamma_2' \vdash P \qquad \Gamma_1 \circ \Gamma_2 \leqslant_{\mathsf{S}} \Gamma_1' \circ \Gamma_2'}{\Theta \cdot \Gamma_1 \circ \Gamma_2 \vdash P}}{\Theta \cdot \Gamma_1 \vdash (\boldsymbol{\nu} s{:}\Gamma_2)P} \quad \textit{iff} \quad \dfrac{\Gamma_1 \leqslant_{\mathsf{S}} \Gamma_1' \qquad \dfrac{\dfrac{\Theta \cdot \Gamma_1' \circ \Gamma_2' \vdash P \qquad \Gamma_2 \leqslant_{\mathsf{S}} \Gamma_2'}{\Theta \cdot \Gamma_1' \circ \Gamma_2 \vdash P}\text{(T-MSUB)}}{\Theta \cdot \Gamma_1' \vdash (\boldsymbol{\nu} s{:}\Gamma_2)P}\text{(T-RES)}}{\Theta \cdot \Gamma_1 \vdash (\boldsymbol{\nu} s{:}\Gamma_2)P}\text{(T-MSUB)}$$

Proof. The first part of the statement is straightforward by inversion of (T-RES) (by Def. B.27). and adding a (possibly vacuous) instance of (T-MSUB), as in the first case in the statement after "moreover". The "iff" relations among the typing derivations are also straightforward: the hypotheses of one derivation implies the other, and if one hypothesis is falsified, none of the derivations hold. ◀

▶ **Proposition B.30** (Subtyping normalisation). *If $\Theta \cdot \Gamma \vdash P$, then there exist a derivation that proves the judgement by only applying rule (T-MSUB) on the conclusions of (T-BRCH), (T-SEL) and (T-CALL).*

Proof. Assume that we have a derivation $\mathcal{D}$ concluding $\Theta \cdot \Gamma \vdash P$, that does not match the thesis: if it is just an instance of (T-BRCH), (T-SEL) and (T-CALL), we conclude by simply adding a vacuous instance of (T-MSUB). Otherwise, $\mathcal{D}$ must have one of the shapes in Proposition B.28 or Proposition B.29, and we can "push" (T-MSUB) towards the leafs (where (T-BRCH), (T-SEL) and (T-CALL) occur) by recursively rewriting it in the first form of the statements. ◀

▶ **Theorem 2.16** (Subject reduction). *If $\Theta \cdot \Gamma \vdash P$ and $P \to P'$, then there exists Γ' such that $\Gamma \to^* \Gamma'$ and $\Theta \cdot \Gamma' \vdash P'$.*

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

- base case (R-COMM). We have $P = Q_1 \mid Q_2$, and:

$$\begin{aligned} Q_1 &= s[\mathsf{p}][\mathsf{q}]\&_{j\in I}\{l_j(x_j).Q_{1j}''\} \\ Q_2 &= s[\mathsf{q}][\mathsf{p}] \oplus \langle l_k(v)\rangle.Q_2'' \\ P &= Q_1 \mid Q_2 \;\to\; Q_{1k}''\{^v/_{x_k}\} \mid Q_2'' \;=\; P' \qquad (k \in I) \end{aligned} \tag{41}$$

Therefore, for some $k \in I$, by inversion of (T-PAR) and (T-BRCH)/(T-SEL), allowing (possibly vacuous) instances of (T-MSUB) as per Proposition B.30, there exist Γ_1, Γ_2 such that $\Gamma = \Gamma_1 \circ \Gamma_2$, and $\Gamma_1^\diamond, \Gamma_2^\diamond, \Gamma_1^{\diamond\prime}, \Gamma_2^{\diamond\prime}$ such that:

$$\dfrac{\text{(T-MSUB)}\dfrac{\text{(T-BRCH)}\dfrac{\forall j \in I \qquad \Theta \cdot \Gamma_1^{\diamond\prime}, x_j{:}U_j', s[\mathsf{p}]{:}S_j' \vdash Q_{1j}''}{\Theta \cdot \Gamma_1^\diamond \vdash s[\mathsf{p}][\mathsf{q}]\&_{j\in I}\{l_j(x_j).Q_{1j}''\}} \qquad \Gamma_1 \leqslant_{\mathsf{S}} \Gamma_1^\diamond}{\Theta \cdot \Gamma_1 \vdash s[\mathsf{p}][\mathsf{q}]\&_{j\in I}\{l_j(x_j).Q_{1j}''\}} \qquad \dfrac{\Gamma_2 \leqslant_{\mathsf{S}} \Gamma_2^\diamond \qquad \dfrac{\Gamma_v \vdash v{:}U'' \qquad \Theta \cdot \Gamma_2^{\diamond\prime}, s[\mathsf{q}]{:}S'' \vdash Q_2''}{\Theta \cdot \Gamma_2^\diamond \vdash s[\mathsf{q}][\mathsf{p}] \oplus \langle l_k(v)\rangle.Q_2''}\text{(T-SEL)}}{\Theta \cdot \Gamma_2 \vdash s[\mathsf{q}][\mathsf{p}] \oplus \langle l_k(v)\rangle.Q_2''}\text{(T-MSUB)}}{\Theta \cdot \Gamma_1 \circ \Gamma_2 \vdash Q_1 \mid Q_2 \;=\; s[\mathsf{p}][\mathsf{q}]\&_{j\in I}\{l_j(x_j).Q_{1j}''\} \;\mid\; s[\mathsf{q}][\mathsf{p}] \oplus \langle l_k(v)\rangle.Q_2''}\text{(T-PAR)} \tag{42}$$

$$\Gamma_1^\diamond = \Gamma_1^{\diamond\prime}, s[\mathsf{p}]{:}S_{\mathsf{p}} \quad \text{where} \quad S_{\mathsf{p}} = \mathsf{q}\,\&_{j\in I}\,?l_j(U_j').S_j' \tag{43}$$

$$\Gamma_2^\diamond = \Gamma_v \circ \Gamma_2^{\diamond\prime}, s[\mathsf{q}]{:}S_{\mathsf{q}} \quad \text{where} \quad S_{\mathsf{q}} = \mathsf{p} \oplus !l_k(U'').S'' \quad \text{for some} \quad k \in I \tag{44}$$

Notice that:

$$\Gamma = \left(\Gamma_1^*, s[\mathtt{p}]\!:\!S_{\mathtt{p}}^*\right) \circ \left(\Gamma_v^* \circ \Gamma_2^*, s[\mathtt{q}]\!:\!S_{\mathtt{q}}^*\right) \quad \text{where} \begin{cases} \Gamma_1^* \leqslant_{\mathsf{S}} \Gamma_1^{\diamond\prime},\ S_{\mathtt{p}}^* \leqslant_{\mathsf{S}} S_{\mathtt{p}}, \\ \Gamma_v^* \leqslant_{\mathsf{S}} \Gamma_v,\ \Gamma_2^* \leqslant_{\mathsf{S}} \Gamma_2^{\diamond\prime},\ S_{\mathtt{q}}^* \leqslant_{\mathsf{S}} S_{\mathtt{q}} \end{cases} \quad \text{(by (42), (43), (44))} \tag{45}$$

From the consistency of $\Gamma = \Gamma_1 \circ \Gamma_2$, and since $\Gamma_1 \leqslant_{\mathsf{S}} \Gamma_1^{\diamond}$ and $\Gamma_2 \leqslant_{\mathsf{S}} \Gamma_2^{\diamond}$ with $\Gamma_v \vdash v\!:\!U''$, we also have:

$$U'' \leqslant_{\mathsf{S}} U_k' \quad \text{(from (43) and (44))} \tag{46}$$

Now, let:

$$\Gamma' \;=\; \Gamma_1' \circ \Gamma_2' \qquad \text{where} \quad \Gamma_1' \;=\; \Gamma_1^{\diamond\prime} \circ \Gamma_v, s[\mathtt{p}]\!:\!S_k' \quad \text{and} \quad \Gamma_2' \;=\; \Gamma_2^{\diamond\prime}, s[\mathtt{q}]\!:\!S'' \tag{47}$$

Before proceeding, we prove $\Gamma \to \Gamma'$ (and therefore, $\Gamma \to^* \Gamma'$):

1. we first observe that:

$$s[\mathtt{p}]\!:\!S_{\mathtt{p}}^*, s[\mathtt{q}]\!:\!S_{\mathtt{q}}^* \;\;\to\;\; s[\mathtt{p}]\!:\!S_k', s[\mathtt{q}]\!:\!S'' \tag{48}$$

since Γ is consistent by hypothesis, and therefore $\mathrm{unf}\left(S_{\mathtt{p}}^*\right) \restriction \mathtt{q}$ and $\mathrm{unf}\left(S_{\mathtt{q}}^*\right) \restriction \mathtt{p}$ have at least l_k in common, with compatible payload types as per Def. 2.15) ;

2. then, let:

$$\Gamma_{\dagger} \;=\; \Gamma_1^* \circ \Gamma_v^* \circ \Gamma_2^* \qquad\qquad \Gamma_{\dagger}' \;=\; \Gamma_1^{\diamond\prime} \circ \Gamma_v \circ \Gamma_2^{\diamond\prime}$$

(i.e., $\Gamma_{\dagger}$ and $\Gamma_{\dagger}'$ are respectively Γ and Γ' without their entries for $s[\mathtt{p}], s[\mathtt{q}]$). We can prove the following statement:

$$s[\mathtt{p}]\!:\!S_{\mathtt{p}}^*, s[\mathtt{q}]\!:\!S_{\mathtt{q}}^*, \Gamma_{\dagger} \;\;\to\;\; s[\mathtt{p}]\!:\!S_k', s[\mathtt{q}]\!:\!S'', \Gamma_{\dagger}' \qquad \text{(and thus, } \Gamma \to^* \Gamma'\text{)} \tag{49}$$

by induction on the size of $\Gamma_{\dagger}$ (which is also the size of $\Gamma_{\dagger}'$): the base case ($\Gamma_{\dagger} = \Gamma_{\dagger}' = \varnothing$) follows by (48), while in the inductive case we apply the induction hypothesis, and use the subtyping relations in (45) to conclude by the inductive rule of Def. 2.15.

We can now continue proving the main statement, observing:

$$\begin{array}{lr} \Gamma' \;\text{ is consistent} & \text{(by (49) and Lemma B.22)} \quad (50) \\ \Theta \cdot \Gamma_1^{\diamond\prime}, x_k\!:\!U_k', s[\mathtt{p}]\!:\!S_k' \vdash Q_{1k}'' \quad (i \in k) & \text{(from (42), premise of (T-BRCH))} \quad (51) \\ \Theta \cdot \Gamma_1^{\diamond\prime}, x_k\!:\!U'', s[\mathtt{p}]\!:\!S_k' \vdash Q_{1k}'' \quad (i \in k) & \text{(by (51), (46) and (T-SUB))} \quad (52) \\ \Gamma_v \vdash v\!:\!U'' & \text{(from (42), premise of (T-SEL))} \quad (53) \\ \Gamma_1^{\diamond\prime}, s[\mathtt{p}]\!:\!S_k' \circ \Gamma_v \;\text{ is consistent} & \text{(by (50), (47) and Cor. B.17)} \quad (54) \\ \Theta \cdot \Gamma_1^{\diamond\prime}, s[\mathtt{p}]\!:\!S_k' \circ \Gamma_v \vdash Q_{1k}''\{v/x_k\} \quad (i \in k) & \text{(by (52), (53), (54) and Lemma B.26)} \quad (55) \end{array}$$

Therefore, by (47), using (55) and the remaining premise of (T-SEL) in (42), we conclude by typing the reduct in (41) as follows:

$$\Theta \cdot \Gamma \vdash P \;\to\; \dfrac{\Theta \cdot \Gamma_1^{\diamond\prime} \circ \Gamma_v, s[\mathtt{p}]\!:\!S_k' \vdash Q_{1k}''\{v/x_k\} \qquad \Theta \cdot \Gamma_2^{\diamond\prime}, s[\mathtt{q}]\!:\!S'' \vdash Q_2''}{\Theta \cdot \Gamma' \vdash Q_{1k}''\{v/x_k\} \mid Q_2'' \;=\; P'} \text{(T-PAR)}$$

- base case (R-CALL). We have:

$$P \;=\; \mathbf{def}\ X(x_1,\ldots,x_n) = Q_X\ \mathbf{in}\ (X\langle v_1,\ldots,v_n\rangle \mid Q) \;\;\to\;\; \mathbf{def}\ X(x_1,\ldots,x_n) = Q_X\ \mathbf{in}\ (Q_X\{v_i/x_i\}_{i\in\{1..n\}} \mid Q) \;=\; P'$$

Let $\widetilde{x} = x_1, \ldots, x_n$ and $\widetilde{U} = U_1, \ldots, U_n$. By inversion of (T-Res), (T-Par) and (T-Call), allowing a (possibly vacuous) instance of (T-MSub) as per Proposition B.30, we have $\Gamma = \Gamma_1 \circ \Gamma_2$ with $\Gamma_1 \leqslant_{\mathsf{S}} \Gamma_1^{\diamond} = \Gamma_{1,1}^{\diamond} \circ \cdots \circ \Gamma_{1,n}^{\diamond}$, such that:

$$\text{(T-Def)}\ \dfrac{\Theta, X{:}\widetilde{U} \cdot \widetilde{x}{:}\widetilde{U} \vdash Q_X \qquad \text{(T-Par)}\ \dfrac{\text{(T-MSub)}\ \dfrac{\text{(T-Call)}\ \dfrac{\Theta, X{:}\widetilde{U} \vdash X{:}\widetilde{U} \qquad \forall i \in \{1..n\} \quad \Gamma_{1,i}^{\diamond} \vdash v_i{:}U_i}{\Theta, X{:}\widetilde{U} \cdot \Gamma_1^{\diamond} \vdash X\langle v_1, \ldots, v_n\rangle} \qquad \Gamma_1 \leqslant_{\mathsf{S}} \Gamma_1^{\diamond}}{\Theta, X{:}\widetilde{U} \cdot \Gamma_1 \vdash X\langle v_1, \ldots, v_n\rangle} \qquad \Theta, X{:}\widetilde{U} \cdot \Gamma_2 \vdash Q}{\Theta, X{:}\widetilde{U} \cdot (\Gamma_1 \circ \Gamma_2) \vdash X\langle v_1, \ldots, v_n\rangle \mid Q}}{\Theta \cdot \Gamma \vdash \mathbf{def}\ X(x_1, \ldots, x_n) = Q_X\ \mathbf{in}\ (X\langle v_1, \ldots, v_n\rangle \mid Q)}$$

Observe that from $\Theta, X{:}\widetilde{U} \cdot \widetilde{x}{:}\widetilde{U} \vdash Q_X$, by applying Lemma B.26 n times (noticing that each time we get a consistent context) we obtain $\Theta, X{:}\widetilde{U} \cdot \Gamma_1^{\diamond} \vdash Q_X\{v_i/x_i\}_{i\in\{1..n\}}$, and thus $\Theta, X{:}\widetilde{U} \cdot \Gamma_1 \vdash Q_X\{v_i/x_i\}_{i\in\{1..n\}}$ (by $\Gamma_1 \leqslant_{\mathsf{S}} \Gamma_1^{\diamond}$ and (T-MSub)), and therefore:

$$\text{(T-Def)}\ \dfrac{\Theta, X{:}\widetilde{U} \cdot \widetilde{x}{:}\widetilde{U} \vdash Q_X \qquad \text{(T-Par)}\ \dfrac{\Theta, X{:}\widetilde{U} \cdot \Gamma_1 \vdash Q_X\{v_i/x_i\}_{i\in\{1..n\}} \qquad \Theta, X{:}\widetilde{U} \cdot \Gamma_2 \vdash Q}{\Theta, X{:}\widetilde{U} \cdot (\Gamma_1 \circ \Gamma_2) \vdash Q_X\{v_i/x_i\}_{i\in\{1..n\}} \mid Q}}{\Theta \cdot \Gamma \vdash \mathbf{def}\ X(x_1, \ldots, x_n) = Q_X\ \mathbf{in}\ (Q_X\{v_i/x_i\}_{i\in\{1..n\}} \mid Q) \ = \ P'}$$

and we conclude by letting $\Gamma' = \Gamma$;

- inductive case (R-Par). We have $P = P_1 \mid P_2 \to P_1' \mid P_2 = P'$, with $P_1 \to P_1'$ (from the rule premise). By inversion of (T-Par), we have $\Gamma = \Gamma_1 \circ \Gamma_2$ such that:

$$\text{(T-Par)}\ \dfrac{\Theta \cdot \Gamma_1 \vdash P_1 \qquad \Theta \cdot \Gamma_2 \vdash P_2}{\Theta \cdot \Gamma_1 \circ \Gamma_2 \vdash P_1 \mid P_2 = P}$$

By the induction hypothesis, $\exists \Gamma_1'$ such that $\Gamma_1 \to^* \Gamma_1'$ and $\Theta, \Gamma_1' \vdash P_1'$. By Cor. B.24, we have that $\Gamma_1' \circ \Gamma_2$ is consistent. Hence, we conclude by letting $\Gamma' = \Gamma_1' \circ \Gamma_2$, obtaining:

$$\text{(T-Par)}\ \dfrac{\Theta \cdot \Gamma_1 \vdash P_1' \qquad \Theta \cdot \Gamma_2 \vdash P_2}{\Theta \cdot \Gamma' \vdash P_1' \mid P_2 = P'}$$

- inductive case (R-Res). We have $P = (\boldsymbol{\nu} s{:}\Gamma^{\diamond})P' \to (\boldsymbol{\nu} s)P'' = P'$, with $P' \to P''$, $\Gamma^{\diamond} = \{s[\mathtt{p}]{:}S_{\mathtt{p}}\}_{\mathtt{p}\in I}$ (for some I), and $\Theta \cdot \Gamma \circ \Gamma^{\diamond} \vdash P'$ (from the rule premise). By the induction hypothesis, $\exists \Gamma''$ such that:

$$\Gamma \circ \Gamma^{\diamond} \ \to^* \ \Gamma'' \qquad \text{and} \qquad \Theta \cdot \Gamma'' \vdash P'' \tag{56}$$

By Proposition B.21, we know that $\operatorname{dom}(\Gamma'') = \operatorname{dom}(\Gamma \circ \Gamma^{\diamond}) = \operatorname{dom}(\Gamma \circ \{s[\mathtt{p}]{:}S_{\mathtt{p}}\}_{\mathtt{p}\in I})$, and therefore:

$$\text{for some } \ \Gamma', \Gamma^{\diamond\prime} \ \text{ with } \ \operatorname{dom}(\Gamma') = \operatorname{dom}(\Gamma) \ \text{ and } \ \Gamma^{\diamond\prime} = \{S_{\mathtt{p}}'\}_{\mathtt{p}\in I}, \quad \Gamma'' = \Gamma' \circ \Gamma^{\diamond\prime} \tag{57}$$

Hence, we can rewrite the typing context reduction in (56) as:

$$\Gamma \circ \Gamma^{\diamond} \ \to^* \ \Gamma' \circ \Gamma^{\diamond\prime} \tag{58}$$

and therefore,

$$\Theta \cdot \Gamma' \circ \Gamma^{\diamond\prime} \vdash P'' \qquad\qquad \text{(by (58), (57) and (56))} \tag{59}$$

By Def. 2.12, the validity of the typing judgement in (59) implies that $\Gamma' \circ \Gamma^{\diamond\prime}$ is consistent, and therefore, by Cor. B.24, Γ' is consistent. Hence, we conclude by:

$$\text{(T-Res)}\ \dfrac{\Theta \cdot \Gamma' \circ \Gamma^{\diamond\prime} \vdash P''}{\Theta \cdot \Gamma' \vdash (\boldsymbol{\nu} s{:}\Gamma^{\diamond\prime})P'' = P'}$$

- inductive case (R-Def). We have $P = \mathbf{def}\, X(\tilde{x}) = Q_X \;\mathbf{in}\, Q \to \mathbf{def}\, X(\tilde{x}) = Q_X \;\mathbf{in}\, Q' = P'$, with $Q \to Q'$ (from the rule premise). By inversion of (T-Def), we get:

$$\text{(T-Def)}\;\frac{\Theta, X\!:\!\tilde{U} \cdot \tilde{x}\!:\!\tilde{U} \vdash Q_X \qquad \Theta, X\!:\!\tilde{U} \cdot \Gamma \vdash Q}{\Theta \cdot \Gamma \vdash \mathbf{def}\, X(\tilde{x}) = Q_X \;\mathbf{in}\, Q}$$

By the induction hypothesis, $\exists\Gamma' : \Gamma \to^* \Gamma'$ and $\Theta, X\!:\!\tilde{U} \cdot \Gamma' \vdash Q'$, and we conclude by:

$$\text{(T-Def)}\;\frac{\Theta, X\!:\!\tilde{U} \cdot \tilde{x}\!:\!\tilde{U} \vdash Q_X \qquad \Theta, X\!:\!\tilde{U} \cdot \Gamma' \vdash Q'}{\Theta \cdot \Gamma' \vdash \mathbf{def}\, X(\tilde{x}) = Q_X \;\mathbf{in}\, Q' = P'}$$

- inductive case (R-Struct). We have $P \equiv Q$ and $Q' \equiv P'$, with $Q \to Q'$ (from the rule premise). By Proposition B.25, $\Theta \cdot \Gamma \vdash Q$; by the induction hypothesis, $\exists\Gamma' : \Gamma \to^* \Gamma'$ and $\Theta \cdot \Gamma' \vdash Q'$; by Proposition B.25, we conclude $\Theta \cdot \Gamma' \vdash P'$.

◀

C Proofs for § 4

▶ **Lemma 4.2.** $\mathrm{unf}\big(\overline{T}\big) = \overline{\mathrm{unf}(T)}$.

Proof. We first show that the LHS of the statement is defined iff the RHS is defined, too. Obviously, $\overline{T}$ is defined iff $\mathrm{unf}\big(\overline{T}\big)$ is defined. Moreover, by Def. 4.1, $\overline{T}$ is defined iff T is a (possibly recursive) linear input/output type, or $\bullet$: hence, $\overline{T}$ is defined iff $\mathrm{unf}(T)$ is a (non-recursive) linear input/output type, or $\bullet$: this implies that $\overline{T}$ is defined iff $\overline{\mathrm{unf}(T)}$ is defined. Summing up: $\mathrm{unf}\big(\overline{T}\big)$ is defined iff $\overline{\mathrm{unf}(T)}$ is defined.

Let us now assume that $\overline{T}$ is defined. If T is *not* a μ-type, i.e., $T \neq \mu\mathbf{t}.T'$ (for some T'), the statement holds trivially by Def. 4.1: in fact, we have $\mathrm{unf}\big(\overline{T}\big) = \overline{T} = \overline{\mathrm{unf}(T)}$. Otherwise, when $T = \mu\mathbf{t}.T'$, by Def. 4.1 we have $\overline{T} = \overline{\mu\mathbf{t}.T'} = \mu\mathbf{t}.\overline{T'}\big\{\overline{\mathbf{t}}/\mathbf{t}\big\}$. Let us examine the *one-step* unfolding of $\overline{T}$ (i.e., we do not (yet) unfold $\overline{T'}$ if it is a μ-type):

$$
\begin{aligned}
&\overline{T'}\big\{\overline{\mathbf{t}}/\mathbf{t}\big\}\Big\{\mu\mathbf{t}.\overline{T'}\{\overline{\mathbf{t}}/\mathbf{t}\}/\mathbf{t}\Big\}\\
&= \overline{T'}\Big\{\overline{\mu\mathbf{t}.\overline{T'}\{\overline{\mathbf{t}}/\mathbf{t}\}}/\mathbf{t}\Big\} \;=\; \overline{T'}\Big\{\mu\mathbf{t}.\overline{\overline{T'}\{\overline{\mathbf{t}}/\mathbf{t}\}}\{\overline{\mathbf{t}}/\mathbf{t}\}/\mathbf{t}\Big\} \;=\; \overline{T'}\Big\{\mu\mathbf{t}.\overline{\overline{T'}}\{\overline{\mathbf{t}}/\mathbf{t}\}\{\overline{\mathbf{t}}/\mathbf{t}\}/\mathbf{t}\Big\} \;=\; \Big\{\mu\mathbf{t}.T'\{\overline{\mathbf{t}}/\mathbf{t}\}\{\overline{\mathbf{t}}/\mathbf{t}\}/\mathbf{t}\Big\}\\
&= \overline{T'}\big\{\mu\mathbf{t}.T'/\mathbf{t}\big\}
\end{aligned}
$$

We can observe that if we dualise the *one-step* unfolding of $T = \mu\mathbf{t}.T'$ (i.e., if we dualise $T'\big\{\mu\mathbf{t}.T'/\mathbf{t}\big\}$), we get the same result:

$$
\overline{T'\big\{\mu\mathbf{t}.T'/\mathbf{t}\big\}} \;=\; \overline{T'}\big\{\mu\mathbf{t}.T'/\mathbf{t}\big\}
$$

Now, if we take $\overline{T'}\big\{\mu\mathbf{t}.T'/\mathbf{t}\big\}$ and its dual $\overline{\overline{T'}\big\{\mu\mathbf{t}.T'/\mathbf{t}\big\}} = \overline{\overline{T'}}\big\{\mu\mathbf{t}.T'/\mathbf{t}\big\} = T'\big\{\mu\mathbf{t}.T'/\mathbf{t}\big\}$ we can repeat the reasoning above; we can further iterate along all the successive one-step unfoldings, until we reach a non-μ-type: at each step, the one-step unfolding of the dualised type matches the dual of the one-step-unfolded type. Hence, we conclude $\mathrm{unf}\big(\overline{T}\big) = \overline{\mathrm{unf}(T)}$. ◀

▶ **Definition C.1.** *The relation $=_\pi$ for π-types is* coinductively *defined as:*

$$
\frac{}{B =_\pi B}\;(=_\pi\text{-LB}) \qquad \frac{}{\bullet =_\pi \bullet}\;(=_\pi\text{-LEND}) \qquad \frac{T =_\pi T'}{\mathsf{Li}(T) =_\pi \mathsf{Li}\big(T'\big)}\;(=_\pi\text{-}\mathsf{Li}) \qquad \frac{T' =_\pi T}{\mathsf{Lo}(T) =_\pi \mathsf{Lo}\big(T'\big)}\;(=_\pi\text{-}\mathsf{Lo})
$$

$$
\frac{\forall i \in I \quad T_i =_\pi T'_i}{\langle l_i\_T_i\rangle_{i\in I} =_\pi \big\langle l_i\_T'_i\big\rangle_{i\in I}}\;(=_\pi\text{-VARIANT}) \qquad \frac{\forall i \in I \quad T_i =_\pi T'_i}{[l_i : T_i]_{i\in I} =_\pi \big[l_i : T'_i\big]_{i\in I}}\;(=_\pi\text{-LTUPLE})
$$

$$
\frac{T\{\mu\mathbf{t}.T/\mathbf{t}\} =_\pi T'\big\{\mu\mathbf{t}'.T'/\mathbf{t}'\big\}}{\mu\mathbf{t}.T =_\pi \mu\mathbf{t}'.T'}\;(=_\pi\text{-L}\mu)
$$

▶ **Remark C.2.** *Def. C.1 is actually* stronger *than required for Lemma 4.4: it implies $\leqslant_\pi \cap \leqslant_\pi^{-1}$ (see Proposition C.4 below), but restricts unfolding of recursion so that related types can only unfold "in unison" (by rule ($=_\pi$-Lμ)).*

▶ **Proposition C.3.** *$=_\pi$ is reflexive.*

▶ **Proposition C.4.** *If $T =_\pi T'$, then $T \leqslant_\pi T'$ and $T' \leqslant_\pi T$.*

Proof. We first prove the thesis for $T \leqslant_\pi T'$. Consider the following relation:

$$
\begin{aligned}
\mathcal{R} &= \mathcal{R}_1 \cup \mathcal{R}_2\\
\mathcal{R}_1 &= \big\{\, (T, T') \;\big|\; T =_\pi T' \,\big\}\\
\mathcal{R}_2 &= \Big\{\, (T\{\mu\mathbf{t}.T/\mathbf{t}\}, \mu\mathbf{t}'.T'), (\mu\mathbf{t}.T, T'\big\{\mu\mathbf{t}'.T'/\mathbf{t}'\big\}) \;\Big|\; \mu\mathbf{t}.T =_\pi \mu\mathbf{t}'.T' \,\Big\}
\end{aligned}
$$

We can easily prove that $\mathcal{R}$ is closed backwards under the rules in Def. 3.5. For all $(T_1, T_2) \in \mathcal{R}$, we have either:

- $(T_1, T_2) \in \mathcal{R}_1$. Then, we proceed by cases on the coinductive rule in Def. C.1 concluding $T_1 =_\pi T_2$. Most cases are straighforward: we show that they satisfy a corresponding rule in Def. 3.5, and the relations in the coinductive premises involve pairs of elements (T_1', T_2') that belong to $\mathcal{R}_1 \subseteq \mathcal{R}$. The inly exception is:
  - $(=_\pi\text{-L}\mu)$. Then, $T_1 = \mu\mathbf{t}.T$ and $T_2 = \mu\mathbf{t}'.T'$, and we have to show that $\mathcal{R}$ satisfies *both* rules (S-LμL) and (S-LμR) in Def. 3.5: we conclude observing that the required pairs $(T\{\mu\mathbf{t}.T/\mathbf{t}\}, \mu\mathbf{t}'.T')$ and $(\mu\mathbf{t}.T, T'\{\mu\mathbf{t}'.T'/\mathbf{t}'\})$ belong to $\mathcal{R}_2 \subseteq \mathcal{R}$;
- $(T_1, T_2) \in \mathcal{R}_2$. We have either:
  - $T_1 = T\{\mu\mathbf{t}.T/\mathbf{t}\}$ and $T_2 = \mu\mathbf{t}'.T'$. We have to satisfy rule (S-LμR): we conclude observing that the required pair of types $(T\{\mu\mathbf{t}.T/\mathbf{t}\}, T'\{\mu\mathbf{t}'.T'/\mathbf{t}'\})$ belongs to $\mathcal{R}_1 \subseteq \mathcal{R}$, by $(=_\pi\text{-L}\mu)$;
  - $T_1 = \mu\mathbf{t}.T$ and $T_2 = T'\{\mu\mathbf{t}'.T'/\mathbf{t}'\}$. Similar to the previous case: we have to satisfy rule (S-LμL), and conclude by observing that the required pair belongs to $\mathcal{R}_1 \subseteq \mathcal{R}$, by $(=_\pi\text{-L}\mu)$.

Summing up, we have shown that $\mathcal{R}$ is closed backwards under the rules for $\leqslant_\pi$; and since $\leqslant_\pi$ is the *largest* relation closed backwards under such rules, we have $\mathcal{R} \subseteq \leqslant_\pi$. Hence, since $T =_\pi T'$ implies $(T, T') \in \mathcal{R}_1 \subseteq \mathcal{R}$, we conclude that $T =_\pi T'$ implies $T \leqslant_\pi T'$.

The proof of the statement for $T' \leqslant_\pi T$ is symmetric. ◀

▶ **Lemma 4.4** (Erasure of $\overline{\mathbf{t}}$). $\mu\mathbf{t}.T =_\pi \mu\mathbf{t}.T\{\mu\mathbf{t}'.\overline{T}\{\mathbf{t}'/\overline{\mathbf{t}}\}/\overline{\mathbf{t}}\}$, *for all* $\mathbf{t}' \notin \text{fv}(T)$.

Proof. Let $T' = T\{\mu\mathbf{t}'.\overline{T}\{\mathbf{t}'/\overline{\mathbf{t}}\}/\overline{\mathbf{t}}\}$ (for some $\mathbf{t}' \notin \text{fv}(T)$), and consider the following relation:

$$
\begin{aligned}
\mathcal{R} &= \mathcal{R}_\mu \cup \mathcal{R}_* \cup \mathcal{R}_{\overline{\mu}} \cup \mathcal{R}_{\overline{*}}\\
\mathcal{R}_\mu &= \left\{(\mu\mathbf{t}.T, \mu\mathbf{t}.T'),\ (T\{\mu\mathbf{t}.T/\mathbf{t}\},\ T'\{\mu\mathbf{t}.T'/\mathbf{t}\})\right\}\\
\mathcal{R}_* &= \left\{ (T_A\{\mu\mathbf{t}.T/\mathbf{t}\},\ T_B\{\mu\mathbf{t}'.\overline{T}\{\mathbf{t}'/\overline{\mathbf{t}}\}/\overline{\mathbf{t}}\}\{\mu\mathbf{t}.T'/\mathbf{t}\}) \;\middle|\; T_A\{\mu\mathbf{t}.T/\mathbf{t}\} =_\pi T_B\{\mu\mathbf{t}.T/\mathbf{t}\} \right\}\\
\mathcal{R}_{\overline{\mu}} &= \left\{ \begin{array}{l} \left(\mu\mathbf{t}.\overline{T}\{\overline{\mathbf{t}}/\mathbf{t}\},\ \mu\mathbf{t}'.\overline{T}\{\mathbf{t}'/\overline{\mathbf{t}}\}\{\mu\mathbf{t}.T'/\mathbf{t}\}\right),\\ \left(\overline{T}\{\overline{\mathbf{t}}/\mathbf{t}\}\{\mu\mathbf{t}.\overline{T}\{\overline{\mathbf{t}}/\mathbf{t}\}/\mathbf{t}\},\ \overline{T}\{\mathbf{t}'/\overline{\mathbf{t}}\}\{\mu\mathbf{t}.T'/\mathbf{t}\}\{\mu\mathbf{t}'.\overline{T}\{\mathbf{t}'/\overline{\mathbf{t}}\}\{\mu\mathbf{t}.T'/\mathbf{t}\}/\mathbf{t}'\}\right) \end{array} \right\}\\
\mathcal{R}_{\overline{*}} &= \left\{ (\overline{T_A}\{\overline{\mathbf{t}}/\mathbf{t}\}\{\mu\mathbf{t}.\overline{T}\{\overline{\mathbf{t}}/\mathbf{t}\}/\mathbf{t}\},\ \overline{T_B}\{\mathbf{t}'/\overline{\mathbf{t}}\}\{\mu\mathbf{t}.T'/\mathbf{t}\}\{\mu\mathbf{t}'.\overline{T}\{\mathbf{t}'/\overline{\mathbf{t}}\}\{\mu\mathbf{t}.T'/\mathbf{t}\}/\mathbf{t}'\}) \;\middle|\; \overline{T_A}\{\mu\mathbf{t}.T/\mathbf{t}\} =_\pi \overline{T_B}\{\mu\mathbf{t}.T/\mathbf{t}\} \right\}
\end{aligned}
$$

We prove that $\mathcal{R}$ is closed backwards under the rules obtained from Def. C.1 by replacing each occurrence of $=_\pi$ with $\mathcal{R}$. For each pair of types $(T_1, T_2) \in \mathcal{R}$, we have the following cases:

- $(T_1, T_2) \in \mathcal{R}_\mu$. We have the following sub-cases:
  - $T_1 = \mu\mathbf{t}.T$ and $T_2 = \mu\mathbf{t}.T'$. We need to satisfy rule $(=_\pi\text{-L}\mu)$: we conclude by noticing that $(T\{\mu\mathbf{t}.T/\mathbf{t}\}, T'\{\mu\mathbf{t}.T'/\mathbf{t}\}) \in \mathcal{R}_\mu \subseteq \mathcal{R}$;
  - $T_1 = T\{\mu\mathbf{t}.T/\mathbf{t}\}$ and $T_2 = T'\{\mu\mathbf{t}.T'/\mathbf{t}\} = T\{\mu\mathbf{t}'.\overline{T}\{\mathbf{t}'/\overline{\mathbf{t}}\}/\overline{\mathbf{t}}\}\{\mu\mathbf{t}.T'/\mathbf{t}\}$. Since $T\{\mu\mathbf{t}.T/\mathbf{t}\} =_\pi T\{\mu\mathbf{t}.T/\mathbf{t}\}$ (by reflexivity of $=_\pi$, Proposition C.3), by definition of $\mathcal{R}_*$ we have $(T_1, T_2) \in \mathcal{R}_*$: we study this case below;
- $(T_1, T_2) \in \mathcal{R}_*$. We have $T_1 = T_A\{\mu\mathbf{t}.T/\mathbf{t}\}$ and $T_2 = T_B\{\mu\mathbf{t}'.\overline{T}\{\mathbf{t}'/\overline{\mathbf{t}}\}/\overline{\mathbf{t}}\}\{\mu\mathbf{t}.T'/\mathbf{t}\}$, for some T_A, T_B such that:

$$T_A\{\mu\mathbf{t}.T/\mathbf{t}\} =_\pi T_B\{\mu\mathbf{t}.T/\mathbf{t}\} \tag{60}$$

  We proceed by cases on the rule in Def. C.1 concluding (60), examining the possible shapes of T_A and T_B, and showing that $\mathcal{R}_*$ is closed backwards under the same rule. Case $(=_\pi\text{-LEnd})$ is trivial, and most other cases are simple: by the coinductive premises of the selected rule, we obtain one or more relations of the form $T_A'\{\mu\mathbf{t}.T/\mathbf{t}\} =_\pi T_B'\{\mu\mathbf{t}.T/\mathbf{t}\}$ (for some T_A', T_B'), and in each case we conclude that $\left(T_A'\{\mu\mathbf{t}.T/\mathbf{t}\},\ T_B'\{\mu\mathbf{t}'.\overline{T}\{\mathbf{t}'/\overline{\mathbf{t}}\}/\overline{\mathbf{t}}\}\{\mu\mathbf{t}.T'/\mathbf{t}\}\right)$ belongs to $\mathcal{R}_*$. The only exception is when (60) is the conclusion of $(=_\pi\text{-L}\mu)$, and T_A,T_B are either:
  - $T_A = T_B = \mathbf{t}$. In this case, we have $T_1 = \mu\mathbf{t}.T$ and $T_2 = \mu\mathbf{t}.T'$, and by definition of $\mathcal{R}_\mu$, $(T_1, T_2) \in \mathcal{R}_\mu$: we study this case above;

- $\overline{T_A} = \overline{T_B} = \overline{\mathbf{t}}$. In this case, we have $T_1 = \overline{\mu\mathbf{t}.T} = \mu\mathbf{t}.\overline{T}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}$ and $T_2 = \mu\mathbf{t}'.\overline{T}\left\{\mathbf{t}'/\overline{\mathbf{t}}\right\}\left\{\mu\mathbf{t}.T'/\mathbf{t}\right\}$. By definition of $\mathcal{R}_{\overline{\mu}}$, we have $(T_1, T_2) \in \mathcal{R}_{\overline{\mu}}$: we study this case below;
- $\overline{T_A} = \mathbf{t}$, $\overline{T_B} = \overline{\mathbf{t}}$, or $\overline{T_A} = \overline{\mathbf{t}}$, $\overline{T_B} = \mathbf{t}$. These cases imply $T = \bullet$, and we conclude that (T_1, T_2) satisfies rule ($=_\pi$-LEnd);

- $(T_1, T_2) \in \mathcal{R}_{\overline{\mu}}$. We have 2 sub-cases, and one is proved similarly to the forst case of $(T_1, T_2) \in \mathcal{R}_\mu$ (i.e., by satisfying rule ($=_\pi$-Lμ)). The remaining (and most interesting) case is:
  - $T_1 = \overline{T}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}\left\{\mu\mathbf{t}.\overline{T}\{\overline{\mathbf{t}}/\mathbf{t}\}/\mathbf{t}\right\}$ and $T_2 = \overline{T}\left\{\mathbf{t}'/\overline{\mathbf{t}}\right\}\left\{\mu\mathbf{t}.T'/\mathbf{t}\right\}\left\{\mu\mathbf{t}'.\overline{T}\{\mathbf{t}'/\overline{\mathbf{t}}\}\{\mu\mathbf{t}.T'/\mathbf{t}\}/\mathbf{t}'\right\}$. Since $\overline{T}\{\mu\mathbf{t}.T/\mathbf{t}\} =_\pi \overline{T}\{\mu\mathbf{t}.T/\mathbf{t}\}$ (by reflexivity of $=_\pi$, Proposition C.3), by definition of $\mathcal{R}_{\overline{*}}$ we have $(T_1, T_2) \in \mathcal{R}_{\overline{*}}$: we study this case below;

- $(T_1, T_2) \in \mathcal{R}_{\overline{*}}$. We have $T_1 = \overline{T_A}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}\left\{\mu\mathbf{t}.\overline{T}\{\overline{\mathbf{t}}/\mathbf{t}\}/\mathbf{t}\right\}$ and $T_2 = \overline{T_B}\left\{\mathbf{t}'/\overline{\mathbf{t}}\right\}\left\{\mu\mathbf{t}.T'/\mathbf{t}\right\}\left\{\mu\mathbf{t}'.\overline{T}\{\mathbf{t}'/\overline{\mathbf{t}}\}\{\mu\mathbf{t}.T'/\mathbf{t}\}/\mathbf{t}'\right\}$, for some $\overline{T_A}, \overline{T_B}$ such that:

$$\overline{T_A}\{\mu\mathbf{t}.T/\mathbf{t}\} \;=_\pi\; \overline{T_B}\{\mu\mathbf{t}.T/\mathbf{t}\} \tag{61}$$

  The proof is similar to that for $(T_1, T_2) \in \mathcal{R}_*$ above: we proceed by cases on the rule in Def. C.1 concluding (61), examining the possible shapes of $\overline{T_A}$ and $\overline{T_B}$, and showing that $\mathcal{R}_{\overline{*}}$ is closed backwards under the same rule. Case ($=_\pi$-LEnd) is trivial, and most other cases are simple: by the coinductive premises of the selected rule, we obtain one or more relations of the form $\overline{T'_A}\{\mu\mathbf{t}.T/\mathbf{t}\} =_\pi \overline{T'_B}\{\mu\mathbf{t}.T/\mathbf{t}\}$ (for some $\overline{T'_A}, \overline{T'_B}$), and in each case we conclude that $(\overline{T'_A}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}\left\{\mu\mathbf{t}.\overline{T}\{\overline{\mathbf{t}}/\mathbf{t}\}/\mathbf{t}\right\}, \overline{T'_B}\left\{\mathbf{t}'/\overline{\mathbf{t}}\right\}\left\{\mu\mathbf{t}.T'/\mathbf{t}\right\}\left\{\mu\mathbf{t}'.\overline{T}\{\mathbf{t}'/\overline{\mathbf{t}}\}\{\mu\mathbf{t}.T'/\mathbf{t}\}/\mathbf{t}'\right\})$ belongs to $\mathcal{R}_{\overline{*}}$. The only exception is when (61) is the conclusion of ($=_\pi$-Lμ), and $\overline{T_A}$,$\overline{T_B}$ are either:

  - $\overline{T_A} = \overline{T_B} = \mathbf{t}$. In this case, we have:

$$T_1 = \overline{\mathbf{t}}\left\{\mu\mathbf{t}.\overline{T}\{\overline{\mathbf{t}}/\mathbf{t}\}/\mathbf{t}\right\} = \overline{\mu\mathbf{t}.\overline{T}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}} = \mu\mathbf{t}.\overline{\overline{T}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\} = \mu\mathbf{t}.\overline{\overline{T}}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\} = \mu\mathbf{t}.T\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\} = \mu\mathbf{t}.T$$
$$T_2 = \mu\mathbf{t}.T'\left\{\mu\mathbf{t}'.\overline{T}\{\mathbf{t}'/\overline{\mathbf{t}}\}\{\mu\mathbf{t}.T'/\mathbf{t}\}/\mathbf{t}'\right\} = \mu\mathbf{t}.T' \quad (\text{since } \mathbf{t}' \notin \mathrm{fv}(T'))$$

  Hence, by definition of $\mathcal{R}_\mu$, we get $(T_1, T_2) \in \mathcal{R}_\mu$: we study this case above;
  - $\overline{T_A} = \overline{T_B} = \overline{\mathbf{t}}$. In this case, we have:

$$T_1 = \mathbf{t}\left\{\mu\mathbf{t}.\overline{T}\{\overline{\mathbf{t}}/\mathbf{t}\}/\mathbf{t}\right\} = \mu\mathbf{t}.\overline{T}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}$$
$$T_2 = \mathbf{t}'\left\{\mu\mathbf{t}.T'/\mathbf{t}\right\}\left\{\mu\mathbf{t}'.\overline{T}\{\mathbf{t}'/\overline{\mathbf{t}}\}\{\mu\mathbf{t}.T'/\mathbf{t}\}/\mathbf{t}'\right\} = \mu\mathbf{t}'.\overline{T}\left\{\mathbf{t}'/\overline{\mathbf{t}}\right\}\left\{\mu\mathbf{t}.T'/\mathbf{t}\right\}$$

  Hence, by definition of $\mathcal{R}_{\overline{\mu}}$, we get $(T_1, T_2) \in \mathcal{R}_{\overline{\mu}}$: we study this case above;
  - $\overline{T_A} = \mathbf{t}$, $\overline{T_B} = \overline{\mathbf{t}}$, or $\overline{T_A} = \overline{\mathbf{t}}$, $\overline{T_B} = \mathbf{t}$. These cases imply $T = \bullet$, and we conclude that (T_1, T_2) satisfies rule ($=_\pi$-LEnd).

We have shown that $\mathcal{R}$ is closed backwards under the rules obtained from Def. C.1. Therefore, since $=_\pi$ is the *largest* relation closed backwards under such rules, we have $\mathcal{R} \subseteq =_\pi$. We also know that $(\mu\mathbf{t}.T, \mu\mathbf{t}.T') \subseteq \mathcal{R}_\mu \subseteq \mathcal{R} \subseteq =_\pi$: we conclude $\mu\mathbf{t}.T =_\pi \mu\mathbf{t}.T'$. ◀

C.1 Quasi-Linearity and Confluence

We now characterise a confluent fragment of linear π-calculus: we will use it later on, to prove the operational correspondence of our encoding.

▶ **Definition C.5** (Quasi-linearity). *The predicate* $\mathrm{qlin}(T)$ *is defined as:*

$$\frac{T \in B \cup \{\mathsf{Li}(T'), \mathsf{Lo}(T'), \mathsf{L}\sharp(T'), \bullet\}}{\mathrm{qlin}(T)} \qquad \frac{\forall i \in I \quad \mathrm{qlin}(T_i)}{\mathrm{qlin}(\langle l_i\_T_i\rangle_{i\in I})} \qquad \frac{\forall i \in I \quad \mathrm{qlin}(T_i)}{\mathrm{qlin}([l_i : T_i]_{i\in I})} \qquad \frac{\mathrm{qlin}(T)}{\mathrm{qlin}(\mu\mathbf{t}.T)}$$

We say that T is quasi-linear *iff* $\mathrm{qlin}(T)$. *We say that a typing judgement* $\Gamma \vdash P$ is quasi-linear *iff it has a derivation such that, for each* $\Gamma', x{:}T \vdash P'$ *occurring in it, either:* (a) $\mathrm{qlin}(T)$, *or* (b) $T = \sharp(T')$ *and* $P' \in \{Q\,,\, x(y).Q\,,\, *(x(y).Q) \mid Q'\}$ *where* x *can occur in* Q, Q' *only as* $\overline{x}\langle v\rangle$ *with* $x \notin \mathrm{fn}(v)$. *We say that* P is quasi-linear *iff* $\exists\Gamma, P'$ *such that* $P' \equiv P$ *and* $\Gamma \vdash P'$ *holds and is quasi-linear.*

Intuitively, Def. C.5 says that if a type T is quasi-linear, then it does *not* harbour unrestricted communication capabilities. If a typed process P is quasi-linear, then each name x is quasi-linear (item *(a)*), or is unrestricted but used in a syntactically-constrained way (item *(b)*). The constraints of item *(b)* are quite standard, and ensure that x is *uniformly ω-receptive* [54, §8.2]: for all synchronisations on x, each transmitted value is processed immediately, and in the same way, by *one* process that spawns a new replica for each input on x — while x is only used for output elsewhere. Note that e.g. $P' = x(y).\mathbf{0} \mid *(x(y).\mathbf{0})$ violates item *(b)*, but is quasi-linear since $P' \equiv *(x(y).\mathbf{0})$ (which satisfies the definition). Quasi-linearity is preserved along reductions (Proposition C.6) and implies confluence (Lemma C.7) — intuitively, because synchronisations are deterministic, as they can only involve linear names [35, Theorem 4.4.1], or ω-receptive names.

▶ **Proposition C.6.** *If P is quasi-linear and $P \to^* P'$, then P' is quasi-linear.*

▶ **Lemma C.7** (Quasi-linear processes are confluent)**.** *If P is quasi-linear, $P \to P_1$ and $P \to P_2$, then either $P_1 \equiv P_2$ or $\exists P_3$ such that $P_1 \to P_3$ and $P_2 \to P_3$.*

▶ **Proposition C.6.** *If P is quasi-linear and $P \to^* P'$, then P' is quasi-linear.*

Proof. We first prove that:

$$P \to P' \qquad \text{implies that} \qquad P' \text{ is quasi-linear} \tag{62}$$

We first observe that, by Def. C.5 P must be typed by some context Γ, and there exist $P_0 \equiv P$ such that $\Gamma \vdash P_0$ is quasi-linear. Since $P_0 \xrightarrow{\alpha} P'$ (for some α), by standard subject reduction on linear π-calculus [35, Theorem 4.3.1], there exists some Γ' (whose definition depends on Γ and α) such that $\Gamma' \vdash P'$. We then proceed by induction on the derivation of the transition $P_0 \xrightarrow{\alpha} P'$:

- in the base case of synchronisation with $\alpha = x$ and x linear, we observe that $\Gamma'(x) = \bullet$ and $\forall y \in \operatorname{dom}(\Gamma) \setminus \{x\} : \Gamma'(y) = \Gamma(y)$ (i.e., no new unrestricted communication capabilities are introduced); moreover, P' still respects item *(b)*;
- in the base case of synchronisation with $\alpha = x$ and x unrestricted, we observe that $\Gamma' = \Gamma$ (i.e., no new unrestricted communication capabilities are introduced); then, we use item *(b)* of Def. C.5 to determine the shape of P_0, and verify that P' still respects item *(b)*;
- in the base cases with $\alpha \in \{\mathbf{case}, \mathbf{with}, \mathbf{let}\}$, we have $\Gamma' = \Gamma$ (i.e., no new unrestricted communication capabilities are introduced); then we verify that P' still respects item *(b)*;
- the other inductive cases hold by applying the induction hypothesis.

We can now prove the main statement. Let n be the length of the sequence of transitions $P \to^* P'$: in the base case ($n = 0$) the statement holds trivially, while the inductive case ($n = n' + 1$) it follows by the induction hypothesis and (62). ◀

▶ **Proposition C.8** (Linear synchronisations are confluent)**.** *If $\Gamma, x{:}\mathsf{L}\sharp(T) \vdash P$, $P \xrightarrow{x} P'$ and $P \xrightarrow{x} P''$, then $P' \equiv P''$.*

Proof. See [35, Theorem 4.4.1]. ◀

▶ **Lemma C.9** (Quasi-linear synchronisations are confluent)**.** *If $\Gamma, x{:}\sharp(T) \vdash P$ is quasi-linear, $P \xrightarrow{x} P_1$ and $P \xrightarrow{x} P_2$, then either $P_1 \equiv P_2$, or $\exists P_3$ such that $P_1 \xrightarrow{x} P_3$ and $P_2 \xrightarrow{x} P_3$.*

Proof. We can only get a synchronisation on x if we have, by Def. C.5::

$$P \;\equiv\; (\boldsymbol{\nu}\tilde{z})\left(*x(y).Q \mid \overline{x}\langle v\rangle.Q' \mid R\right) \;\equiv\; (\boldsymbol{\nu}\tilde{z})\left(x(y).Q \mid \overline{x}\langle v\rangle.Q' \mid *(x(y).Q) \mid R\right)$$

where $x \notin \tilde{z}$ and Q, Q', R can only use x for output. Considering the synchronisation $P \xrightarrow{x} P_1$, we get:

$$P_1 \;\equiv\; (\boldsymbol{\nu}\tilde{z})\left(Q\{v/y\} \mid Q' \mid *(x(y).Q) \mid R\right)$$

Let us now examine the synchronisation $P \xrightarrow{x} P_2$. We have two cases:

- if it coincides with the synchronisation $P \xrightarrow{x} P_1$, we trivially conclude $P_1 \equiv P_2$;
- otherwise, if a *different* synchronisation leads to P_2, we must have $R \equiv \overline{x}\langle v'\rangle.R' \mid R''$ (i.e., another output is enabled on x), and thus:
  - $P_1 \equiv (\boldsymbol{\nu}\tilde{z})\,(Q\{^{v}/_{y}\} \mid Q' \mid *(x(y).Q) \mid \overline{x}\langle v'\rangle.R' \mid R'') \equiv (\boldsymbol{\nu}\tilde{z})\,(Q\{^{v}/_{y}\} \mid x(y).Q \mid Q' \mid \overline{x}\langle v'\rangle.R' \mid *(x(y).Q) \mid R'')$;
  - $P_2 \equiv (\boldsymbol{\nu}\tilde{z})\,\left(Q\left\{^{v'}/_{y}\right\} \mid \overline{x}\langle v\rangle.Q' \mid R' \mid R''\right) \equiv (\boldsymbol{\nu}\tilde{z})\,\left(x(y).Q \mid Q\left\{^{v'}/_{y}\right\} \mid \overline{x}\langle v\rangle.Q' \mid R' \mid *(x(y).Q) \mid R''\right)$.

  Therefore, letting $P_3 = (\boldsymbol{\nu}\tilde{z})\,\left(Q\{^{v}/_{y}\} \mid Q\left\{^{v'}/_{y}\right\} \mid Q' \mid R' \mid *(x(y).Q) \mid R''\right)$, we conclude $P_1 \xrightarrow{x} P_3$ and $P_2 \xrightarrow{x} P_3$.

◀

▶ **Corollary C.10** (Partial confluence). *If P is quasi-linear, $P \xrightarrow{x} P_1$ and $P \xrightarrow{\alpha} P_2$ (for any α), then either $P_1 \equiv P_2$ or $\exists P_3$ such that $P_1 \xrightarrow{\alpha} P_3$ and $P_2 \xrightarrow{x} P_3$.*

Proof. *(Sketch)* The proof is similar to [35, Theorem 4.4.3], which assumes x to be linearly-typed, and depends on Proposition C.8. The only difference is that in our statement, x might be an unrestricted name; in this case, the result still holds by the quasi-linearity hypothesis, and by Lemma C.9. ◀

▶ **Proposition C.11.** *If $\alpha \in \{\mathbf{with}, \mathbf{case}, \mathbf{let}\}$, $P \xrightarrow{\alpha} P_1$ and $P \xrightarrow{\beta} P_2$ (for any β), then either $P_1 \equiv P_2$, or $\exists P_3$ such that $P_1 \xrightarrow{\beta} P_3$ and $P_2 \xrightarrow{\alpha} P_3$.*

Proof. Let $\alpha = \mathbf{with}$. We have: :

$$P \;\equiv\; (\boldsymbol{\nu}\tilde{z})\,\left(\mathbf{with}\,[l_i : x_i]_{i\in I} = [l_i : v_i]_{i\in I}\,\mathbf{do}\,P \mid Q\right) \;\xrightarrow{\mathbf{with}}\; (\boldsymbol{\nu}\tilde{z})\,(P\{^{v_i}/_{x_i}\}_{i\in I} \mid Q) \;\equiv\; P_1$$

Let us now examine the reduction $P \xrightarrow{\beta} P_2$. We have two cases:

- if it coincides with the reduction $P \xrightarrow{\alpha} P_1$, we trivially conclude $P_1 \equiv P_2$;
- otherwise, if a *different* reduction leads to P_2, we must have $Q \xrightarrow{\beta} Q'$, and:

$$P \;\xrightarrow{\beta}\; (\boldsymbol{\nu}\tilde{z})\,\left(\mathbf{with}\,[l_i : x_i]_{i\in I} = [l_i : v_i]_{i\in I}\,\mathbf{do}\,P \mid Q'\right) \;\equiv\; P_2$$

Therefore, letting $P_3 = (\boldsymbol{\nu}\tilde{z})\,(P\{^{v_i}/_{x_i}\}_{i\in I} \mid Q')$, we conclude $P_1 \xrightarrow{\beta} P_3$ and $P_2 \xrightarrow{\alpha} P_3$.

The proofs for $\alpha = \mathbf{case}$ and $\alpha = \mathbf{let}$ are similar. ◀

▶ **Lemma C.7** (Quasi-linear processes are confluent). *If P is quasi-linear, $P \to P_1$ and $P \to P_2$, then either $P_1 \equiv P_2$ or $\exists P_3$ such that $P_1 \to P_3$ and $P_2 \to P_3$.*

Proof. Assume that P is quasi-linear. The statement follows from Cor. C.10 and Proposition C.11, which cover all possible transitions of P. ◀

▶ **Corollary C.12** (Quasi-linear processes are confluent (II)). *If P is quasi-linear, $P \to^* P_1$ and $P \to P_2$, then either $P_2 \to^* P_1$, or $\exists P_3$ such that $P_1 \to P_3$ and $P_2 \to^* P_3$.*

Proof. Assume that P is quasi-linear. Let n be the length of the sequence of transitions $P \to^* P_1$. We proceed by induction on n:

- base case $n = 0$. We have $P_1 \equiv P$, and therefore conclude by letting $P_3 = P_2$;
- inductive case $n = n' + 1$. Take P_1' such that $P \to^* P_1' \to P_1$, with n' transitions in $P \to^* P_1'$. By the induction hypothesis, either:
  - $P_2 \to^* P_1'$. In this case, we conclude $P_2 \to^* P_1$;
  - $\exists P_3'$ such that $P_1' \to P_3'$ and $P_2 \to^* P_3'$. In this case, notice that P_1' is quasi-linear (from $P \to^* P_1'$ and by Proposition C.6); therefore, since $P_1' \to P_3'$ and $P_1' \to P_1$, by Lemma C.7 we have either:
    - $P_3' \equiv P_1$. In this case, from $P_2 \to^* P_3'$ we conclude $P_2 \to^* P_1$;

- $\exists P_3''$ such that $P_3' \to P_3''$ and $P_1 \to P_3''$. In this case, by letting $P_3 = P_3''$, we conclude $P_1 \to P_3$ and $P_2 \to^* P_3$.

◀

▶ **Lemma 4.7.** *If $T = T_1 \Cap T_2$, and $T_1' \uplus T_2' = T$, then either (a) $T_1' \leqslant_\pi T_1$ and $T_2' \leqslant_\pi T_2$, or (b) $T_1' \leqslant_\pi T_2$ and $T_2' \leqslant_\pi T_1$.*

Proof. By Def. 4.6, T is defined only if either:

- $T_1 = T_2 = T^*$, for some $\mathrm{un}(T^*)$. In this case, we also have $T = T_1' = T_2' = T^*$, and we trivially obtain both items (a) and (b);
- T_1, T_2 are respectively a linear input and output type, or *vice versa*. Then, again by Def. 4.6, we have two sub-cases:
  - $T_1 = \mathsf{Lo}(T_1^*)$, $T_2 = \mathsf{Li}(T_2^*)$, and $T_1^* \leqslant_\pi T_2^*$. In this case, $T = \mathsf{Li}(T_1^*) \uplus \mathsf{Lo}(T_1^*) = \mathsf{L}\sharp(T_1^*)$, and we have either:
    - $T_1' = \mathsf{Lo}(T_1^*)$ and $T_2' = \mathsf{Li}(T_1^*)$. Then, we conclude $T_1' \leqslant_\pi T_1$ and $T_2' \leqslant_\pi T_2$, i.e., case (a) of the statement;
    - $T_1' = \mathsf{Li}(T_1^*)$ and $T_2' = \mathsf{Lo}(T_1^*)$. Then, we conclude $T_1' \leqslant_\pi T_2$ and $T_2' \leqslant_\pi T_1$, i.e., case (b) of the statement;
  - $T_1 = \mathsf{Li}(T_1')$, $T_2 = \mathsf{Lo}(T_2')$, and $T = T_2' \leqslant_\pi T_1'$. The proof is similar to the previous case.

◀

D Properties of Encoding of Types

D.1 Auxiliary Results

▶ **Proposition D.1.** *Let H, H' be partial session types. Then, $H \leqslant_{\mathsf{P}} H'$ iff $\overline{H'} \leqslant_{\mathsf{P}} \overline{H}$.*

Proof. Follows by the standard properties of duality for binary session types [21]. ◀

▶ **Definition D.2.** $type(\mathsf{p}, [\mathsf{p} : T, \mathsf{q} : T_{\mathsf{q}}]_{\mathsf{q} \in I}) = T$.

D.2 Subtyping and Encoding

▶ **Theorem 6.2** (Encoding preserves subtyping). *If $S \leqslant_{\mathsf{S}} S'$, then $[\![S]\!] \leqslant_{\pi} [\![S']\!]$.*

Proof. By Proposition B.11, since $S \leqslant_{\mathsf{S}} S'$, then $\mathrm{roles}(S) = \mathrm{roles}(S')$. We construct a relation $\mathcal{R} \triangleq \mathcal{R}_S \cup \mathcal{R}_T \cup \mathcal{R}_U \cup \mathcal{R}_P \cup \mathcal{R}_i \cup \mathcal{R}_o$, where its subcomponents are defined as follows:

$$\mathcal{R}_S \triangleq \{([\![S]\!], [\![S']\!]) \mid S \leqslant_{\mathsf{S}} S'\}$$
$$\mathcal{R}_T \triangleq \{(T, T') \mid ([\mathsf{p}\!:\!T_{\mathsf{p}}]_{\mathsf{p} \in I}, [\mathsf{p}\!:\!T'_{\mathsf{p}}]_{\mathsf{p} \in I}) \in \mathcal{R}_S \text{ and } \exists \mathsf{q} \text{ such that } type(\mathsf{q}, [\mathsf{p}\!:\!T_{\mathsf{p}}]_{\mathsf{p} \in I}) = T \text{ and } type(\mathsf{q}, [\mathsf{p}\!:\!T'_{\mathsf{p}}]_{\mathsf{p} \in I}) = T'\}$$
$$\mathcal{R}_U \triangleq \{([\![U]\!], [\![U']\!]) \mid U \leqslant_{\mathsf{S}} U'\}$$
$$\mathcal{R}_P \triangleq \{([\![H]\!], [\![H']\!]) \mid H \leqslant_{\mathsf{P}} H'\}$$
$$\mathcal{R}_i \triangleq \{(T_1, T_2) \mid (\mathsf{Li}(T_1), \mathsf{Li}(T_2)) \in \mathcal{R}_P\}$$
$$\mathcal{R}_o \triangleq \{(T_2, T_1) \mid (\mathsf{Lo}(T_1), \mathsf{Lo}(T_2)) \in \mathcal{R}_P\}$$

We first prove that $\mathcal{R}$ is closed backwards under the rules of $\leqslant_{\pi}$, given by Def. 3.5. We examine all the elements of $\mathcal{R}$, by inspecting all its subsets.

For each pair $([\![U]\!], [\![U']\!]) \in \mathcal{R}_U$ we have the following cases:

- $U \leq_B U'$ meaning that types U, U' are basic types. Since the encoding of basic types is the identity function, then by subtyping $\leq_B$ we conclude that the pair $([\![U]\!], [\![U']\!])$ satisfies rule (S-LB).
- In all other cases U, U' must be closed session types and thus $([\![U]\!], [\![U']\!]) \in \mathcal{R}_S$: we study this case below:.

For each pair $([\![S]\!], [\![S']\!]) \in \mathcal{R}_S$, we know that $S \leqslant_{\mathsf{S}} S'$, and recalling that they are closed session types, we have the following cases, depending on the coinductive rule in Def. 2.10 concluding $S \leqslant_{\mathsf{S}} S'$:

- Case (S-End). We have $[\![S]\!] = [\![S']\!] = \bullet$. Hence, we conclude that the pair $([\![S]\!], [\![S']\!]) = (\bullet, \bullet)$ satisfies rule (S-LEnd).
- Case (S-μL). We have $S = \mu \mathbf{t}.S'' \leqslant_{\mathsf{S}} S'$; hence, by Def. 5.1, $[\![S]\!] = \mu \mathbf{t}.T''$, where $T'' = [\![S'']\!]$. By the premise of (S-μL) we also have $S''\{\mu \mathbf{t}.S''/\mathbf{t}\} \leqslant_{\mathsf{S}} S'$, which implies:

  $$([\![S''\{\mu \mathbf{t}.S''/\mathbf{t}\}]\!], [\![S']\!]) \in \mathcal{R}_S \subseteq \mathcal{R} \tag{63}$$

  Now, we observe:

  $$\begin{array}{rcll} [\![S''\{\mu \mathbf{t}.S''/\mathbf{t}\}]\!] & = & [\![S'']\!]\{[\![\mu \mathbf{t}.S'']\!]/\mathbf{t}\} & \text{(by Lemma D.5)} \\ & = & [\![S'']\!]\{\mu \mathbf{t}.[\![S'']\!]/\mathbf{t}\} & \text{(by Def. 5.1)} \\ & = & T''\{\mu \mathbf{t}.T''/\mathbf{t}\} & \text{(since } T'' = [\![S'']\!]) \end{array}$$

  From (63) we also have $(T''\{\mu \mathbf{t}.T''/\mathbf{t}\}, [\![S']\!]) \in \mathcal{R}_S \subseteq \mathcal{R}$. Hence, the pair $([\![S]\!], [\![S']\!]) = (\mu \mathbf{t}.T'', [\![S']\!])$ satisfies rule (S-Lμ).
- Case (S-μR). Similar to case (S-μL), except that this time we have $S' = \mu \mathbf{t}.S''$. Then, we let $T'' = [\![S'']\!]$ and we obtain the pair $([\![S]\!], [\![S']\!]) = ([\![S]\!], \mu \mathbf{t}.T'')$, which satisfies rule (S-Rμ).

- Cases (S-Brch) and (S-Sel). We have:

$$[\![S]\!] = [\mathtt{p}:T_{\mathtt{p}}]_{\mathtt{p}\in S} \qquad [\![S']\!] = \left[\mathtt{p}:T'_{\mathtt{p}}\right]_{\mathtt{p}\in S'}$$

Let $I = \text{roles}(S) = \text{roles}(S')$. For all $\mathtt{p} \in I$, we have $(T_{\mathtt{p}}, T'_{\mathtt{p}}) \in \mathcal{R}_T \subseteq \mathcal{R}$. We conclude that the pair $([\![S]\!], [\![S']\!])$ satisfies rule (S-LTuple).

For each pair $(T, T') \in \mathcal{R}_T$ there is corresponding pair $([\mathtt{p}:T_{\mathtt{p}}]_{\mathtt{p}\in I}, \left[\mathtt{p}:T'_{\mathtt{p}}\right]_{\mathtt{p}\in I}) \in \mathcal{R}_S$ and there exists $\mathtt{q}$ such that $\mathit{type}(\mathtt{q}, [\mathtt{p}:T_{\mathtt{p}}]_{\mathtt{p}\in I}) = T$ and $\mathit{type}(\mathtt{q}, \left[\mathtt{p}:T'_{\mathtt{p}}\right]_{\mathtt{p}\in I}) = T'$; hence, there are S and S' such that $S \leqslant_{\mathsf{S}} S'$ and

$$[\![S]\!] = [\mathtt{p}:T_{\mathtt{p}}]_{\mathtt{p}\in S} \qquad [\![S']\!] = \left[\mathtt{p}:T'_{\mathtt{p}}\right]_{\mathtt{p}\in S'}$$

Let $I = \text{roles}(S) = \text{roles}(S')$. By Def. 5.1 we have that for all $\mathtt{p} \in I$,

$$T_{\mathtt{p}} = [\![S \upharpoonright \mathtt{p}]\!] \qquad T'_{\mathtt{p}} = [\![S' \upharpoonright \mathtt{p}]\!]$$

By Def. 5.1 and by Def. D.2 we have that $T = [\![S \upharpoonright \mathtt{q}]\!]$ and $T' = [\![S' \upharpoonright \mathtt{q}]\!]$. By Proposition B.12, since $S \leqslant_{\mathsf{S}} S'$, then for all $\mathtt{p} \in S$ also $S \upharpoonright \mathtt{p} \leqslant_{\mathsf{P}} S' \upharpoonright \mathtt{p}$. In particular, since $\mathtt{q} \in S$, also $S \upharpoonright \mathtt{q} \leqslant_{\mathsf{P}} S' \upharpoonright \mathtt{q}$. Then, we have that $(S \upharpoonright \mathtt{q}, S \upharpoonright \mathtt{q}) \in \mathcal{R}_P$: this case is studied below.

For each pair $([\![H]\!], [\![H']\!]) \in \mathcal{R}_P$, we know that $H \leqslant_{\mathsf{P}} H'$. We proceed by cases on the coinductive rule in Def. 2.10 that concludes $H \leqslant_{\mathsf{P}} H'$:

- Case (S-ParEnd). We have $H = H' = \mathbf{end}$, and therefore, $[\![H]\!] = [\![H']\!] = \bullet$. We conclude that the pair $([\![H]\!], [\![H']\!]) = (\bullet, \bullet)$ satisfies rule (S-LEnd).
- Case (S-ParμL). We have $H = \mu\mathbf{t}.H'' \leqslant_{\mathsf{P}} H'$; hence, by Def. 5.1, $[\![H]\!] = \mu\mathbf{t}.T''$, where $T'' = [\![H'']\!]$. By the premise of (S-ParμL) we also have $H''\left\{\mu\mathbf{t}.H''/\mathbf{t}\right\} \leqslant_{\mathsf{P}} H'$, implying:

$$([\![H''\left\{\mu\mathbf{t}.H''/\mathbf{t}\right\}]\!], [\![H']\!]) \in \mathcal{R}_P \tag{64}$$

Now, we observe:

$$\begin{array}{rcll} [\![H''\left\{\mu\mathbf{t}.H''/\mathbf{t}\right\}]\!] & = & [\![H'']\!]\left\{[\![\mu\mathbf{t}.H'']\!]/\mathbf{t}\right\} & \text{(by Lemma D.6)} \\ & = & [\![H'']\!]\left\{\mu\mathbf{t}.[\![H'']\!]/\mathbf{t}\right\} & \text{(by Def. 5.1)} \\ & = & T''\left\{\mu\mathbf{t}.T''/\mathbf{t}\right\} & \text{(since } T'' = [\![H'']\!]) \end{array}$$

From Equation (64) we have $(T''\left\{\mu\mathbf{t}.T''/\mathbf{t}\right\}, [\![H']\!]) \in \mathcal{R}_P$. Hence, the pair $([\![H]\!], [\![H']\!]) = (\mu\mathbf{t}.T'', [\![H']\!])$ satisfies rule (S-LμL).
- Case (S-ParμR). Symmetrical to case (S-ParμL), except that we have $H' = \mu\mathbf{t}.H''$: we let $T'' = [\![H'']\!]$, and we obtain that the pair $([\![H]\!], [\![H']\!]) = ([\![H]\!], \mu\mathbf{t}.T'')$ satisfies rule (S-LμR).
- Cases (S-ParBrch) and (S-ParSel). In these cases, we have either:
- Case (S-ParBrch). In this case, for some T_1, T_2, we have $[\![H]\!] = \mathsf{Li}(T_1)$ and $[\![H']\!] = \mathsf{Li}(T_2)$, and therefore $(T_1, T_2) \in \mathcal{R}_i \subseteq \mathcal{R}$. We conclude that the pair $([\![H]\!], [\![H']\!])$ satisfies rule (S-Li).
- Case (S-ParSel). In this case, for some T_1, T_2, we have $[\![H]\!] = \mathsf{Lo}(T_1)$ and $[\![H']\!] = \mathsf{Lo}(T_2)$, and therefore $(T_2, T_1) \in \mathcal{R}_o \subseteq \mathcal{R}$. We conclude that the pair $([\![H]\!], [\![H']\!])$ satisfies rule (S-Lo).

For each pair $(T_1, T_2) \in \mathcal{R}_i$ there is a corresponding pair $(\mathsf{Li}(T_1), \mathsf{Li}(T_2)) \in \mathcal{R}_P$, and there exist H, H' such that

$$[\![H]\!] = \mathsf{Li}(T_1) \qquad [\![H']\!] = \mathsf{Li}(T_2) \tag{65}$$

and $H \leqslant_{\mathsf{P}} H'$. Equation (65) and Def. 5.1 imply that H and H' are partial branch types. So, the only case to consider is rule (S-ParBrch) and we have that:

$$H = \&_{i\in I}\, ?l_i(U_i).H_i \qquad H' = \&_{i\in I\cup J}\, ?l_i(U'_i).H'_i$$

By the premise of (S-ParBrch), for all $i \in I$ it is the case that $U_i \leqslant_{\mathsf{S}} U'_i$ and $H_i \leqslant_{\mathsf{P}} H'_i$. Then, for all $i \in I$:

$$(\llbracket U_i \rrbracket, \llbracket U'_i \rrbracket) \in \mathcal{R}_U \subseteq \mathcal{R} \qquad (\llbracket H_i \rrbracket, \llbracket H'_i \rrbracket) \in \mathcal{R}_P \subseteq \mathcal{R}$$

By the encoding of partial branch types:

$$\llbracket H \rrbracket = \mathsf{Li}(T_1) \text{ implies } T_1 = \langle l_i\_(\llbracket U_i \rrbracket, \llbracket H_i \rrbracket)\rangle_{i\in I} \quad \llbracket H' \rrbracket = \mathsf{Li}(T_2) \text{ implies } T_2 = \left\langle l_i\_(\llbracket U'_i \rrbracket, \llbracket H'_i \rrbracket)\right\rangle_{i\in I\cup J}$$

We can conclude that the pair (T_1, T_2) satisfies rule (S-Variant).

For each pair in $(T_2, T_1) \in \mathcal{R}_o$ there is a corresponding pair $(\mathsf{Lo}(T_1), \mathsf{Lo}(T_2)) \in \mathcal{R}_P$ and there exist H, H' such that

$$\llbracket H \rrbracket = \mathsf{Lo}(T_1) \qquad \llbracket H' \rrbracket = \mathsf{Lo}(T_2) \tag{66}$$

and $H \leqslant_{\mathsf{P}} H'$. Equation (66) and Def. 5.1 imply that H and H' are partial select types. So, the only case to consider is rule (S-ParSel) and we have that:

$$H = \oplus_{i\in I\cup J}\, !l_i(U_i).H_i \qquad H' = \oplus_{i\in I}\, !l_i(U'_i).H'_i$$

By the premise of rule (S-ParSel), for all $i \in I$. $U'_i \leqslant_{\mathsf{S}} U_i$ and $H_i \leqslant_{\mathsf{P}} H'_i$. By Proposition D.1 we have that $\overline{H'_i} \leqslant_{\mathsf{P}} \overline{H_i}$. Then, for all $i \in I$:

$$(\llbracket U'_i \rrbracket, \llbracket U_i \rrbracket) \in \mathcal{R}_U \subseteq \mathcal{R} \qquad (\llbracket \overline{H'_i} \rrbracket, \llbracket \overline{H_i} \rrbracket) \in \mathcal{R}_P \subseteq \mathcal{R}$$

By the encoding of partial select type:

$$\llbracket H \rrbracket = \mathsf{Lo}(T_1) \text{ implies that } T_1 = \left\langle l_i\_(\llbracket U_i \rrbracket, \llbracket \overline{H_i} \rrbracket)\right\rangle_{i\in I\cup J} \quad \llbracket H' \rrbracket = \mathsf{Lo}(T_2) \text{ implies that } T_2 = \left\langle l_i\_(\llbracket U'_i \rrbracket, \llbracket \overline{H'_i} \rrbracket)\right\rangle_{i\in I}$$

We can conclude that the pair (T_2, T_1) satisfies rule (S-Variant).

We have thus proved that $\mathcal{R}$ is closed backwards under the rules of $\leqslant_\pi$ — and since $\leqslant_\pi$ is the *largest* relation closed backwards under such rules, this implies $\mathcal{R} \subseteq \leqslant_\pi$. We prove the main statement observing that, since $S \leqslant_{\mathsf{S}} S'$ implies $(\llbracket S \rrbracket, \llbracket S' \rrbracket) \in \mathcal{R}_S \subseteq \mathcal{R} \subseteq \leqslant_\pi$, then $S \leqslant_{\mathsf{S}} S'$ implies $\llbracket S \rrbracket \leqslant_\pi \llbracket S' \rrbracket$. ◀

▶ **Corollary D.3.** *$H \leqslant_{\mathsf{P}} H'$, then $\llbracket H \rrbracket \leqslant_\pi \llbracket H' \rrbracket$.*

Proof. Assume $H \leqslant_{\mathsf{P}} H'$. The statement follows by the proof of Theorem 6.2, where the relation $\mathcal{R}_P$ contains the pair $(\llbracket H \rrbracket, \llbracket H' \rrbracket)$; this implies $(\llbracket H \rrbracket, \llbracket H' \rrbracket) \in \mathcal{R}_P \subseteq \mathcal{R} \subseteq \leqslant_\pi$, and therefore, $\llbracket H \rrbracket \leqslant_\pi \llbracket H' \rrbracket$. ◀

D.2.1 Duality and Encoding

▶ **Lemma D.4.** *Let H be a (possibly open) partial session type. Then, $\overline{\overline{\llbracket H \rrbracket \left\{\bar{\mathsf{t}}/\mathsf{t}\right\}_{\mathsf{t}\in \mathrm{fv}(H)}}} = \overline{\overline{\llbracket H \rrbracket}} \left\{\bar{\mathsf{t}}/\mathsf{t}\right\}_{\mathsf{t}\in \mathrm{fv}(H)} = \llbracket H \rrbracket \left\{\bar{\mathsf{t}}/\mathsf{t}\right\}_{\mathsf{t}\in \mathrm{fv}(H)}$.*

Proof. Simple induction on the structure of H. We use Def. 4.1 and the substitution of dualised variables. ◀

▶ **Theorem 6.1** (Encoding preserves duality). $\llbracket \overline{H} \rrbracket = \overline{\llbracket H \rrbracket}$.

Proof. We prove a more general statement: let H be a (possibly open) partial session type. Then $\llbracket \overline{H} \rrbracket = \overline{\llbracket H \rrbracket} \left\{\bar{\mathsf{t}}/\mathsf{t}\right\}_{\mathsf{t}\in \mathrm{fv}(H)}$. The case for a closed partial type H follows as a corollary by the fact that $\mathrm{fv}(H) = \emptyset$ (and thus, the substitution applied on $\overline{\llbracket H \rrbracket}$ is vacuous).

The proof proceeds by induction on the structure of H.

- $H = \mathbf{end}$. By Def. 2.8 we have that $\overline{\mathbf{end}} = \mathbf{end}$. We conclude by Def. 5.1 and by Def. 4.1 and the fact that $\mathrm{fv}(\mathbf{end}) = \emptyset$.

- $H = \mathbf{t}$.
By Def. 2.8 we have that $\overline{\mathbf{t}} = \mathbf{t}$. By Def. 5.1 we have $\llbracket \overline{\mathbf{t}} \rrbracket = \llbracket \mathbf{t} \rrbracket = \mathbf{t}$. By Def. 4.1 we have that $\overline{\llbracket \mathbf{t} \rrbracket} = \overline{(\mathbf{t})} = \overline{\mathbf{t}}$. Then $\overline{\mathbf{t}}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\} = \mathbf{t}$, which concludes this case.

- $H = \&_{i\in I}\, ?l_i(U_i).H_i$.
By Def. 2.5 we know that each U_i is ether a base type or a closed session type; hence $\mathrm{fv}(H) = \cup_{i\in I} \mathrm{fv}(H_i)$. By Def. 2.8 we have that $\overline{H} = \oplus_{i\in I}\, !l_i(U_i).\overline{H_i}$. By Def. 5.1 we have that $\llbracket \overline{H} \rrbracket = \llbracket \oplus_{i\in I}\, !l_i(U_i).\overline{H_i} \rrbracket = \mathsf{Lo}\left(\left\langle l_i\_(\llbracket U_i \rrbracket, \overline{\llbracket \overline{H_i} \rrbracket})\right\rangle_{i\in I}\right)$.
By induction hypothesis for all $i \in I$, $\llbracket \overline{H_i} \rrbracket = \overline{\llbracket H_i \rrbracket}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}_{\mathbf{t}\in\mathrm{fv}(H_i)}$. We rewrite the above as $\llbracket \overline{H} \rrbracket = \mathsf{Lo}\left(\left\langle l_i\_(\llbracket U_i \rrbracket, \overline{\overline{\llbracket H_i \rrbracket}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}_{\mathbf{t}\in\mathrm{fv}(H_i)}})\right\rangle_{i\in I}\right)$ By Lemma D.4 we conclude

$$\llbracket \overline{H} \rrbracket = \mathsf{Lo}\left(\left\langle l_i\_(\llbracket U_i \rrbracket, \llbracket H_i \rrbracket\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}_{\mathbf{t}\in\mathrm{fv}(H_i)})\right\rangle_{i\in I}\right) \tag{67}$$

On the other hand, by Def. 5.1 we have that $\llbracket H \rrbracket = \llbracket \&_{i\in I}\, ?l_i(U_i).H_i \rrbracket = \mathsf{Li}\left(\langle l_i\_(\llbracket U_i \rrbracket, \llbracket H_i \rrbracket)\rangle_{i\in I}\right)$. By Def. 4.1 we have $\overline{\llbracket H \rrbracket} = \overline{\mathsf{Li}\left(\langle l_i\_(\llbracket U_i \rrbracket, \llbracket H_i \rrbracket)\rangle_{i\in I}\right)} = \mathsf{Lo}\left(\langle l_i\_(\llbracket U_i \rrbracket, \llbracket H_i \rrbracket)\rangle_{i\in I}\right)$. Then, by type substitution for all $\mathbf{t} \in \mathrm{fv}(H)$, we have that

$$\overline{\llbracket H \rrbracket}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}_{\mathbf{t}\in\mathrm{fv}(H)} = \mathsf{Lo}\left(\left\langle l_i\_(\llbracket U_i \rrbracket, \llbracket H_i \rrbracket\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}_{\mathbf{t}\in\mathrm{fv}(H_i)})\right\rangle_{i\in I}\right) \tag{68}$$

By comparing Equation (67) and Equation (68) we conclude this case.

- $H = \oplus_{i\in I}\, !l_i(U_i).H_i$. By Def. 2.5 we know that each U_i is ether a base type or a closed session type; hence $\mathrm{fv}(H) = \cup_{i\in I} \mathrm{fv}(H_i)$. By Def. 2.8we have that $\overline{H} = \&_{i\in I}\, ?l_i(U_i).\overline{H_i}$ By Def. 5.1 we have that $\llbracket \overline{H} \rrbracket = \llbracket \&_{i\in I}\, ?l_i(U_i).\overline{H_i} \rrbracket = \mathsf{Li}\left(\left\langle l_i\_(\llbracket U_i \rrbracket, \llbracket \overline{H_i} \rrbracket)\right\rangle_{i\in I}\right)$.
By induction hypothesis for all $i \in I$, $\llbracket \overline{H_i} \rrbracket = \overline{\llbracket H_i \rrbracket}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}_{\mathbf{t}\in\mathrm{fv}(H_i)}$. We rewrite the above as

$$\llbracket \overline{H} \rrbracket = \mathsf{Li}\left(\left\langle l_i\_(\llbracket U_i \rrbracket, \overline{\llbracket H_i \rrbracket}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}_{\mathbf{t}\in\mathrm{fv}(H_i)})\right\rangle_{i\in I}\right) \tag{69}$$

On the other hand, by Def. 5.1 we have that $\llbracket H \rrbracket = \llbracket \oplus_{i\in I}\, !l_i(U_i).H_i \rrbracket = \mathsf{Lo}\left(\left\langle l_i\_(\llbracket U_i \rrbracket, \overline{\llbracket H_i \rrbracket})\right\rangle_{i\in I}\right)$
By Def. 4.1 we have $\overline{\llbracket H \rrbracket} = \overline{\mathsf{Lo}\left(\left\langle l_i\_(\llbracket U_i \rrbracket, \overline{\llbracket H_i \rrbracket})\right\rangle_{i\in I}\right)} = \mathsf{Li}\left(\left\langle l_i\_(\llbracket U_i \rrbracket, \overline{\llbracket H_i \rrbracket})\right\rangle_{i\in I}\right)$. Then, by type substitution for all $\mathbf{t} \in \mathrm{fv}(H)$, we have that

$$\overline{\llbracket H \rrbracket}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}_{\mathbf{t}\in\mathrm{fv}(H)} = \mathsf{Li}\left(\left\langle l_i\_(\llbracket U_i \rrbracket, , \overline{\llbracket H_i \rrbracket}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}_{\mathbf{t}\in\mathrm{fv}(H_i)})\right\rangle_{i\in I}\right) \tag{70}$$

By comparing Equation (69) and Equation (70) we conclude this case.

- $H = \mu\mathbf{t}'.H'$. We have that $\mathrm{fv}(H) = \mathrm{fv}(H') \setminus \{\mathbf{t}'\}$. By Def. 2.8 we have that $\overline{H} = \mu\mathbf{t}'.\overline{H'}$. By Def. 5.1 we have that $\llbracket \mu\mathbf{t}'.\overline{H'} \rrbracket = \mu\mathbf{t}'.\llbracket \overline{H'} \rrbracket$. By induction hypothesis $\llbracket \overline{H'} \rrbracket = \overline{\llbracket H' \rrbracket}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}_{\mathbf{t}\in\mathrm{fv}(H')}$. This implies,

$$\llbracket \overline{H} \rrbracket = \mu\mathbf{t}'.\llbracket \overline{H'} \rrbracket = \mu\mathbf{t}'.\overline{\llbracket H' \rrbracket}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}_{\mathbf{t}\in\mathrm{fv}(H')} \tag{71}$$

On the other hand, $\llbracket H \rrbracket = \llbracket \mu\mathbf{t}'.H' \rrbracket = \mu\mathbf{t}'.\llbracket H' \rrbracket$. By Def. 4.1 we have that

$$\overline{\llbracket H \rrbracket} = \overline{\mu\mathbf{t}'.\llbracket H' \rrbracket} = \mu\mathbf{t}'.\overline{\llbracket H' \rrbracket}\left\{\overline{\mathbf{t}'}/\mathbf{t}'\right\} \tag{72}$$

We have the following:

$$\begin{array}{rcll} \overline{\llbracket H \rrbracket}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}_{\mathbf{t}\in\mathrm{fv}(H)} & = & \left(\mu\mathbf{t}'.\overline{\llbracket H' \rrbracket}\left\{\overline{\mathbf{t}'}/\mathbf{t}'\right\}\right)\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}_{\mathbf{t}\in\mathrm{fv}(H')\setminus\{\mathbf{t}'\}} & \text{(by Equation (72) and the fact that } \mathrm{fv}(H) = \mathrm{fv}(H') \setminus \{\mathbf{t}'\}) \\ & = & \mu\mathbf{t}'.\overline{\llbracket H' \rrbracket}\left\{\overline{\mathbf{t}}/\mathbf{t}\right\}_{\mathbf{t}\in\mathrm{fv}(H')} & \text{(since } \mathrm{fv}(H') \setminus \{\mathbf{t}'\} \cup \{\mathbf{t}'\} = \mathrm{fv}(H')) \\ & = & \llbracket \overline{H} \rrbracket & \text{(by Equation (71))} \end{array}$$

which concludes this case.

◀

D.2.2 Substitution and Encoding

▶ **Lemma D.5.** *Let* S, S' *be session types. Then,* $[\![S\{S'/\mathtt{t}\}]\!] = [\![S]\!]\{[\![S']\!]/\mathtt{t}\}$.

Proof. By induction on the structure of S. ◀

▶ **Lemma D.6.** *Let* H, H' *be partial session types. Then,* $[\![H\{H'/\mathtt{t}\}]\!] = [\![H]\!]\{[\![H']\!]/\mathtt{t}\}$.

Proof. By induction on the structure of H. ◀

The following lemma gives the relation between the type combinator $\uplus$ and the standard ',' operator in linear π-typing contexts.

▶ **Lemma D.7.** *If* $\Gamma_1 \uplus \Gamma_2$ *is defined and* $\mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2) = \emptyset$*, then also* Γ_1, Γ_2 *is defined.*

Proof. Immediate by the combination of typing contexts given in Def. 3.6. ◀

The following definition is an extends Def. 2.11 to accommodate the notion of *linear* session typing context.

▶ **Definition D.8** (Linear and Unrestricted Session Typing Context)**.** *We say that* Γ is unrestricted, $\mathrm{un}(\Gamma)$*, iff for all* $c \in \mathrm{dom}(\Gamma)$*,* $\Gamma(c)$ *is either a base type or* **end***, otherwise we say that* Γ is linear, $\mathrm{lin}(\Gamma)$.

▶ **Proposition D.9.** *Let* Γ_{S} *be a session typing context and* q *be either* lin *or* un*. Then,* $q(\Gamma_{\mathsf{S}})$ *if and only if* $q([\![\Gamma_{\mathsf{S}}]\!])$.

Proof. The result follows immediately by Def. D.8 and Def. 3.6. ◀

$$\llbracket \dfrac{\mathrm{un}(\Gamma) \quad v \in B}{\Gamma \vdash v:B}\ (\text{T-Basic}) \rrbracket \triangleq \dfrac{\mathrm{un}(\llbracket\Gamma\rrbracket) \quad v \in B}{\llbracket\Gamma\rrbracket \vdash v:B}\ (\text{T}\pi\text{-Basic}) \qquad \llbracket \dfrac{\mathrm{un}(\Gamma)}{\Gamma, c:S \vdash c:S}\ (\text{T-Name}) \rrbracket \triangleq \dfrac{\mathrm{un}(\llbracket\Gamma\rrbracket)}{\llbracket\Gamma, c:S\rrbracket \vdash \llbracket c:S\rrbracket}\ (\text{T}\pi\text{-Name})$$

$$\llbracket \dfrac{}{\Theta, X:\widetilde{T} \vdash X:\widetilde{T}}\ (\text{T-DefCtx}) \rrbracket \triangleq \dfrac{\mathrm{un}(\llbracket\Theta\rrbracket)}{\llbracket\Theta, X:\widetilde{T}\rrbracket \vdash \llbracket X:\widetilde{T}\rrbracket}\ (\text{T}\pi\text{-Name}) \qquad \llbracket \dfrac{\mathrm{un}(\Gamma)}{\Gamma \vdash \mathbf{0}}\ (\text{T-Nil}) \rrbracket \triangleq \dfrac{\mathrm{un}(\llbracket\Gamma\rrbracket)}{\llbracket\Gamma\rrbracket \vdash \mathbf{0}}\ (\text{T}\pi\text{-Nil})$$

$$\llbracket \dfrac{\Theta \cdot \Gamma_1 \vdash P \quad \Theta \cdot \Gamma_2 \vdash Q}{\Theta \cdot \Gamma_1 \circ \Gamma_2 \vdash P \mid Q}\ (\text{T-Par}) \rrbracket \triangleq \dfrac{\llbracket\Theta \cdot \Gamma_1\rrbracket \vdash P \quad \llbracket\Theta \cdot \Gamma_2 \vdash Q\rrbracket}{\llbracket\Theta \cdot \Gamma_1 \circ \Gamma_2\rrbracket \vdash \llbracket P\rrbracket_{\Theta\cdot\Gamma_1} \mid \llbracket Q\rrbracket_{\Theta\cdot\Gamma_2}}\ (\text{T}\pi\text{-Par})$$

where $\widetilde{x} = x_1, \ldots, x_n$ and $\widetilde{U} = U_1, \ldots, U_n$

$$\llbracket \dfrac{\Theta, X:\widetilde{U} \cdot \widetilde{x}:\widetilde{U} \vdash P \quad \Theta, X:\widetilde{U} \cdot \Gamma \vdash Q}{\Theta \cdot \Gamma \vdash \mathbf{def}\ X(\widetilde{x}:U) = P\ \mathbf{in}\ Q}\ (\text{T-Def}) \rrbracket \triangleq$$

$$\dfrac{\dfrac{\dfrac{\mathrm{un}\left(\llbracket\Theta, X:\widetilde{U}\rrbracket\right) \quad \dfrac{\dfrac{}{\llbracket X:\widetilde{U}\rrbracket \vdash \llbracket X:\widetilde{U}\rrbracket}\ (\text{T}\pi\text{-Name}) \quad \dfrac{\dfrac{}{z:(\llbracket U_i\rrbracket)_{i\in\{1..n\}} \vdash z:(\llbracket U_i\rrbracket)_{i\in\{1..n\}}}\ (\text{T}\pi\text{-Name}) \quad \llbracket\Theta, X:\widetilde{U} \cdot \widetilde{x}:\widetilde{U} \vdash P\rrbracket}{\llbracket\Theta\rrbracket, \llbracket X\rrbracket:\sharp\left((\llbracket U_i\rrbracket)_{i\in\{1..n\}}\right), z:(\llbracket U_i\rrbracket)_{i\in\{1..n\}} \vdash \mathbf{with}\,(x_i)_{i\in\{1..n\}} = z\ \mathbf{do}\ \llbracket P\rrbracket_{\Theta, X:\widetilde{U}\cdot\widetilde{x}:\widetilde{U}}}\ (\text{T}\pi\text{-With})}{\llbracket\Theta\rrbracket, \llbracket X\rrbracket:\sharp\left((\llbracket U_i\rrbracket)_{i\in\{1..n\}}\right) \vdash \llbracket X\rrbracket(z).\mathbf{with}\,(x_i)_{i\in\{1..n\}} = z\ \mathbf{do}\ \llbracket P\rrbracket_{\Theta, X:\widetilde{U}\cdot\widetilde{x}:\widetilde{U}}}\ (\text{T}\pi\text{-Inp})}{\llbracket\Theta\rrbracket, \llbracket X\rrbracket:\sharp\left((\llbracket U_i\rrbracket)_{i\in\{1..n\}}\right) \vdash *\left(\llbracket X\rrbracket(z).\mathbf{with}\,(x_i)_{i\in\{1..n\}} = z\ \mathbf{do}\ \llbracket P\rrbracket_{\Theta, X:\widetilde{U}\cdot\widetilde{x}:\widetilde{U}}\right)}\ (\text{T}\pi\text{-Repl}) \quad \llbracket\Theta, X:\widetilde{U} \cdot \Gamma \vdash Q\rrbracket}{\llbracket\Theta \cdot \Gamma\rrbracket, \llbracket X\rrbracket:\sharp\left((\llbracket U_i\rrbracket)_{i\in\{1..n\}}\right) \vdash *\left(\llbracket X\rrbracket(z).\mathbf{with}\,(x_i)_{i\in\{1..n\}} = z\ \mathbf{do}\ \llbracket P\rrbracket_{\Theta, X:\widetilde{U}\cdot\widetilde{x}:\widetilde{U}}\right) \mid \llbracket Q\rrbracket_{\Theta, X:\widetilde{U}\cdot\Gamma}}\ (\text{T}\pi\text{-Par})}{\llbracket\Theta \cdot \Gamma\rrbracket \vdash (\boldsymbol{\nu}\llbracket X\rrbracket)\left(*\left(\llbracket X\rrbracket(z).\mathbf{with}\,(x_i)_{i\in\{1..n\}} = z\ \mathbf{do}\ \llbracket P\rrbracket_{\Theta, X:\widetilde{U}\cdot\widetilde{x}:\widetilde{U}}\right) \mid \llbracket Q\rrbracket_{\Theta, X:\widetilde{U}\cdot\Gamma}\right)}\ (\text{T}\pi\text{-Res1})$$

where $\widetilde{v} = v_1, \ldots, v_n$ and $\widetilde{U} = U_1, \ldots, U_n$

$$\llbracket \dfrac{\Theta \vdash X:\widetilde{U} \quad \forall i \in \{1..n\} \quad \Gamma_i \vdash v_i:U_i \quad \mathrm{un}(\Gamma)}{\Theta, X:\widetilde{U} \cdot \Gamma_1 \circ \ldots \circ \Gamma_n \circ \Gamma \vdash X\langle\widetilde{v}\rangle}\ (\text{T-Call}) \rrbracket \triangleq \dfrac{\llbracket\Theta, X:\widetilde{U} \vdash X:\widetilde{U}\rrbracket \quad \dfrac{\forall i \in \{1..n\} \quad \llbracket\Gamma_i \vdash v_i:U_i\rrbracket}{\biguplus_{i\in\{1..n\}} \llbracket\Gamma_i\rrbracket \vdash (\llbracket v_i\rrbracket)_{i\in\{1..n\}}:(\llbracket U_i\rrbracket)_{i\in\{1..n\}}}\ (\text{T}\pi\text{-LTup}) \quad \dfrac{\mathrm{un}(\llbracket\Gamma\rrbracket)}{\llbracket\Gamma\rrbracket \vdash \mathbf{0}}\ (\text{T}\pi\text{-Nil})}{\llbracket\Theta, X:\widetilde{U} \cdot \Gamma_1 \circ \ldots \circ \Gamma_n \circ \Gamma\rrbracket \vdash \overline{\llbracket X\rrbracket}\langle(\llbracket v_i\rrbracket)_{i\in\{1..n\}}\rangle.\mathbf{0}}\ (\text{T}\pi\text{-Out})$$

$$\llbracket \dfrac{\Theta \cdot \Gamma, c:S \vdash P \quad S' \leqslant_{\mathsf{S}} S}{\Theta \cdot \Gamma, c:S' \vdash P}\ (\text{T-Sub}) \rrbracket \triangleq \dfrac{\llbracket\Theta \cdot \Gamma, c:S \vdash P\rrbracket \quad \llbracket S'\rrbracket \leqslant_{\pi} \llbracket S\rrbracket}{\llbracket\Theta \cdot \Gamma, c:S'\rrbracket \vdash \llbracket P\rrbracket_{\Theta\cdot\Gamma, c:S}}\ (\text{T}\pi\text{-Narrow})$$

$$\llbracket \dfrac{\Theta \cdot \Gamma, x_i:U_i, c:S'_i \vdash P_i \quad \forall i \in I}{\Theta \cdot c:S, \Gamma \vdash c[\mathsf{p}]\ \&_{i\in I}\ \{l_i(x_i).P_i\}}\ (\text{T-Brch}) \rrbracket \triangleq \qquad \text{where } S = \mathsf{p}\ \&_{i\in I}\ ?l_i(U_i).S'_i$$

$$\dfrac{\dfrac{}{\llbracket c:S\rrbracket \vdash \llbracket c:S\rrbracket}\ (\text{T}\pi\text{-Name}) \quad \dfrac{\dfrac{}{z_{\mathsf{p}}:\llbracket S\upharpoonright\mathsf{p}\rrbracket \vdash z_{\mathsf{p}}:\llbracket S\upharpoonright\mathsf{p}\rrbracket}\ (\text{T}\pi\text{-Name}) \quad \dfrac{\dfrac{}{y:\langle l_i\_(\llbracket U_i\rrbracket, \llbracket S'_i\upharpoonright\mathsf{p}\rrbracket)\rangle_{i\in I} \vdash y:\langle l_i\_(\llbracket U_i\rrbracket, \llbracket S'_i\upharpoonright\mathsf{p}\rrbracket)\rangle_{i\in I}}\ (\text{T}\pi\text{-Name}) \quad \forall i \in I \quad \dfrac{\dfrac{\dfrac{\left\{\dfrac{\dfrac{}{z_{\mathsf{q}}:\llbracket S\upharpoonright\mathsf{q}\rrbracket \vdash z_{\mathsf{q}}:\llbracket S\upharpoonright\mathsf{q}\rrbracket}\ (\text{T}\pi\text{-Name}) \quad \llbracket S\upharpoonright\mathsf{q}\rrbracket \leqslant_{\pi} \llbracket S'_i\upharpoonright\mathsf{q}\rrbracket}{z_{\mathsf{q}}:\llbracket S\upharpoonright\mathsf{q}\rrbracket \vdash z_{\mathsf{q}}:\llbracket S'_i\upharpoonright\mathsf{q}\rrbracket}\ (\text{T}\pi\text{-Sub})\right\}_{\mathsf{q}\in S\setminus\mathsf{p}} \quad \overbrace{\dfrac{}{z:\llbracket S'_i\upharpoonright\mathsf{p}\rrbracket \vdash z:\llbracket S'_i\upharpoonright\mathsf{p}\rrbracket}\ (\text{T}\pi\text{-Name})}^{\text{if } S'_i\upharpoonright\mathsf{p} \neq \mathbf{end}\text{; otherwise, } \mathrm{un}(z:\llbracket S'_i\upharpoonright\mathsf{p}\rrbracket)}}{\{z_{\mathsf{q}}:\llbracket S\upharpoonright\mathsf{q}\rrbracket\}_{\mathsf{q}\in S\setminus\mathsf{p}}, z:\llbracket S'\upharpoonright\mathsf{p}\rrbracket \vdash \maltese_i:\llbracket S'\rrbracket}\ (\text{T}\pi\text{-LTup}) \quad \llbracket\Theta \cdot \Gamma, x_i:U_i, c:S'_i \vdash P_i\rrbracket}{\llbracket\Theta \cdot \Gamma\rrbracket, x_i:\llbracket U_i\rrbracket, \{z_{\mathsf{q}}:\llbracket S\upharpoonright\mathsf{q}\rrbracket\}_{\mathsf{q}\in S\setminus\mathsf{p}}, z:\llbracket S'_i\upharpoonright\mathsf{p}\rrbracket \vdash \mathbf{let}\ \llbracket c\rrbracket = \maltese_i\ \mathbf{in}\ \llbracket P_i\rrbracket_{\Theta\cdot\Gamma, x_i:U_i, c:S'_i}}\ (\text{T}\pi\text{-Let})}{\llbracket\Theta \cdot \Gamma\rrbracket, z_i:(\llbracket U_i\rrbracket, \llbracket S'_i\upharpoonright\mathsf{p}\rrbracket), \{z_{\mathsf{q}}:\llbracket S\upharpoonright\mathsf{q}\rrbracket\}_{\mathsf{q}\in S\setminus\mathsf{p}} \vdash \mathbf{with}\,(x_i, z) = z_i\ \mathbf{do}\ \mathbf{let}\ \llbracket c\rrbracket = \maltese_i\ \mathbf{in}\ \llbracket P_i\rrbracket_{\Theta\cdot\Gamma, x_i:U_i, c:S'_i}}\ (\text{T}\pi\text{-With})}{\llbracket\Theta \cdot \Gamma\rrbracket, \{z_{\mathsf{q}}:\llbracket S\upharpoonright\mathsf{q}\rrbracket\}_{\mathsf{q}\in S\setminus\mathsf{p}}, y:\langle l_i\_(\llbracket U_i\rrbracket, \llbracket S'_i\upharpoonright\mathsf{p}\rrbracket)\rangle_{i\in I} \vdash \mathbf{case}\ y\ \mathbf{of}\ \left\{l_i(z_i) \triangleright \mathbf{with}\,(x_i, z) = z_i\ \mathbf{do}\ \mathbf{let}\ \llbracket c\rrbracket = \maltese_i\ \mathbf{in}\ \llbracket P_i\rrbracket_{\Theta\cdot\Gamma, x_i:U_i, c:S'_i}\right\}_{i\in I}}\ (\text{T}\pi\text{-Case})}{\llbracket\Theta \cdot \Gamma\rrbracket, \{z_{\mathsf{q}}:\llbracket S\upharpoonright\mathsf{q}\rrbracket\}_{\mathsf{q}\in S} \vdash z_{\mathsf{p}}(y).\mathbf{case}\ y\ \mathbf{of}\ \left\{l_i(z_i) \triangleright \mathbf{with}\,(x_i, z) = z_i\ \mathbf{do}\ \mathbf{let}\ \llbracket c\rrbracket = \maltese_i\ \mathbf{in}\ \llbracket P_i\rrbracket_{\Theta\cdot\Gamma, x_i:U_i, c:S'_i}\right\}_{i\in I}}\ (\text{T}\pi\text{-Inp})}{\llbracket\Theta, c:S, \Gamma\rrbracket \vdash \mathbf{with}\ [\mathsf{q}:z_{\mathsf{q}}]_{\mathsf{q}\in S} = \llbracket c\rrbracket\ \mathbf{do}\ z_{\mathsf{p}}(y).\mathbf{case}\ y\ \mathbf{of}\ \left\{l_i(z_i) \triangleright \mathbf{with}\,(x_i, z) = z_i\ \mathbf{do}\ \mathbf{let}\ \llbracket c\rrbracket = \maltese_i\ \mathbf{in}\ \llbracket P_i\rrbracket_{\Theta\cdot\Gamma, x_i:U_i, c:S'_i}\right\}_{i\in I}}\ (\text{T}\pi\text{-With})$$

where $\maltese_i = \begin{cases} [\mathsf{p}:z,\ \mathsf{q}:z_{\mathsf{q}}]_{\mathsf{q}\in S'_i\setminus\mathsf{p}} & \text{if } \mathsf{p} \in S'_i \\ [\mathsf{q}:z_{\mathsf{q}}]_{\mathsf{q}\in S'_i} & \text{otherwise} \end{cases}$

Figure 13 Encoding of multiparty session typing judgements (part 1).

$$\underbrace{\left\llbracket \dfrac{\Gamma_1 \vdash v : U \qquad \Theta \cdot \Gamma_2, c : S' \vdash P}{\Theta \cdot c : S, \Gamma_1 \circ \Gamma_2 \vdash c[\mathtt{p}] \oplus \langle l(v) \rangle . P}\ (\text{T-Sel}) \right\rrbracket}_{\text{where } S = \mathtt{p} \oplus !l(U).S'} \triangleq$$

$$\underbrace{\dfrac{\dfrac{}{\llbracket c:S \rrbracket \vdash \llbracket c:S \rrbracket}\ (\text{T}\pi\text{-Name}) \qquad \dfrac{\dfrac{\dfrac{}{z_{\mathtt{p}} : \mathsf{Lo}\left(\left\langle l\_(\llbracket U \rrbracket, \overline{\llbracket S' \restriction \mathtt{p} \rrbracket})\right\rangle\right) \vdash z_{\mathtt{p}} : \mathsf{Lo}\left(\left\langle l\_(\llbracket U \rrbracket, \overline{\llbracket S' \restriction \mathtt{p} \rrbracket})\right\rangle\right)}\ (\text{T}\pi\text{-Name}) \qquad \dfrac{\dfrac{\dfrac{}{z : \overline{\llbracket \mathrm{unf}(S' \restriction \mathtt{p}) \rrbracket} \vdash z : \overline{\llbracket \mathrm{unf}(S' \restriction \mathtt{p}) \rrbracket}}\ (\text{T}\pi\text{-Name}) \qquad \overline{\llbracket \mathrm{unf}(S' \restriction \mathtt{p}) \rrbracket} \leqslant_\pi \overline{\llbracket S' \restriction \mathtt{p} \rrbracket}}{z : \overline{\llbracket \mathrm{unf}(S' \restriction \mathtt{p}) \rrbracket} \vdash z : \overline{\llbracket S' \restriction \mathtt{p} \rrbracket}}\ (\text{T}\pi\text{-Sub}) \qquad \llbracket \Gamma_1 \vdash v : U \rrbracket}{\llbracket \Gamma_1 \rrbracket, z : \overline{\llbracket \mathrm{unf}(S' \restriction \mathtt{p}) \rrbracket} \vdash l(\llbracket v \rrbracket, z) : \left\langle l\_(\llbracket U \rrbracket, \overline{\llbracket S' \restriction \mathtt{p} \rrbracket})\right\rangle}\ (\text{T}\pi\text{-LVal}) \qquad \dfrac{\dfrac{\left\{ \dfrac{\dfrac{}{z_{\mathtt{q}} : \llbracket S \restriction \mathtt{q} \rrbracket \vdash z_{\mathtt{q}} : \llbracket S \restriction \mathtt{q} \rrbracket}\ (\text{T}\pi\text{-Name}) \qquad \llbracket S \restriction \mathtt{q} \rrbracket \leqslant_\pi \llbracket S'_i \restriction \mathtt{q} \rrbracket}{z_{\mathtt{q}} : \llbracket S \restriction \mathtt{q} \rrbracket \vdash z_{\mathtt{q}} : \llbracket S'_i \restriction \mathtt{q} \rrbracket}\ (\text{T}\pi\text{-Sub}) \right\}_{\mathtt{q} \in S \setminus \mathtt{p}} \qquad \overbrace{\dfrac{\llbracket \mathrm{unf}(S' \restriction \mathtt{p}) \rrbracket \leqslant_\pi \llbracket S' \restriction \mathtt{p} \rrbracket \qquad \dfrac{}{z : \llbracket \mathrm{unf}(S' \restriction \mathtt{p}) \rrbracket \vdash z : \llbracket \mathrm{unf}(S' \restriction \mathtt{p}) \rrbracket}\ (\text{T}\pi\text{-Name})}{z : \llbracket \mathrm{unf}(S' \restriction \mathtt{p}) \rrbracket \vdash z : \llbracket S' \restriction \mathtt{p} \rrbracket}\ (\text{T}\pi\text{-Sub})}^{\text{if } S' \restriction \mathtt{p} \neq \mathbf{end};\ \text{otherwise, } \mathrm{un}(z : \llbracket S' \restriction \mathtt{p} \rrbracket)}}{\{z_{\mathtt{q}} : \llbracket S' \restriction \mathtt{q} \rrbracket\}_{\mathtt{q} \in S \setminus \mathtt{p}}, z : \llbracket S' \restriction \mathtt{p} \rrbracket \vdash \maltese : \llbracket S' \rrbracket}\ (\text{T}\pi\text{-LTup}) \qquad \llbracket \Theta \cdot \Gamma_2, c : S' \vdash P \rrbracket}{\llbracket \Theta \cdot \Gamma_2 \rrbracket, \{z_{\mathtt{q}} : \llbracket S \restriction \mathtt{q} \rrbracket\}_{\mathtt{q} \in S \setminus \mathtt{p}}, z : \llbracket \mathrm{unf}(S' \restriction \mathtt{p}) \rrbracket \vdash \mathbf{let}\ \llbracket c \rrbracket = \maltese\ \mathbf{in}\ \llbracket P \rrbracket_{\Theta \cdot \Gamma_2, c:S'}}\ (\text{T}\pi\text{-Let})}{\llbracket \Theta \cdot \Gamma_1 \circ \Gamma_2 \rrbracket, \{z_{\mathtt{q}} : \llbracket S \restriction \mathtt{q} \rrbracket\}_{\mathtt{q} \in S \setminus \mathtt{p}}, z_{\mathtt{p}} : \mathsf{Lo}\left(\left\langle l\_(\llbracket U \rrbracket, \overline{\llbracket S' \restriction \mathtt{p} \rrbracket})\right\rangle\right), z : \llbracket \mathrm{unf}(S' \restriction \mathtt{p}) \rrbracket \uplus \overline{\llbracket \mathrm{unf}(S' \restriction \mathtt{p}) \rrbracket} \vdash \overline{z_{\mathtt{p}}} \langle l(\llbracket v \rrbracket, z) \rangle . \mathbf{let}\ \llbracket c \rrbracket = \maltese\ \mathbf{in}\ \llbracket P \rrbracket_{\Theta \cdot \Gamma_2, c:S'}}\ (\text{T}\pi\text{-Out})}{\llbracket \Theta \cdot \Gamma_1 \circ \Gamma_2 \rrbracket, \{z_{\mathtt{q}} : \llbracket S \restriction \mathtt{q} \rrbracket\}_{\mathtt{q} \in S} \vdash (\boldsymbol{\nu} z) \overline{z_{\mathtt{p}}} \langle l(\llbracket v \rrbracket, z) \rangle . \mathbf{let}\ \llbracket c \rrbracket = \maltese\ \mathbf{in}\ \llbracket P \rrbracket_{\Theta \cdot \Gamma_2, c:S'}}\ (\text{T}\pi\text{-Res*})}{\llbracket \Theta, c : S, \Gamma_1 \circ \Gamma_2 \rrbracket \vdash \mathbf{with}\ [\mathtt{q} : z_{\mathtt{q}}]_{\mathtt{q} \in S} = \llbracket c \rrbracket\ \mathbf{do}\ (\boldsymbol{\nu} z) \overline{z_{\mathtt{p}}} \langle l(\llbracket v \rrbracket, z) \rangle . \mathbf{let}\ \llbracket c \rrbracket = \maltese\ \mathbf{in}\ \llbracket P \rrbracket_{\Theta \cdot \Gamma_2, c:S'}}\ (\text{T}\pi\text{-With})}_{\text{where } \maltese = \begin{cases} [\mathtt{p} : z,\ \mathtt{q} : z_{\mathtt{q}}]_{\mathtt{q} \in S' \setminus \mathtt{p}} & \text{if } \mathtt{p} \in S' \\ [\mathtt{q} : z_{\mathtt{q}}]_{\mathtt{q} \in S'} & \text{otherwise} \end{cases}}$$

$$\left\llbracket \dfrac{\Theta \cdot \Gamma, \Gamma' \vdash P \qquad \Gamma' = \{s[\mathtt{p}] : S_{\mathtt{p}}\}_{\mathtt{p} \in I}}{\Theta \cdot \Gamma \vdash (\boldsymbol{\nu} s : \Gamma') P}\ (\text{T-Res}) \right\rrbracket \triangleq \underbrace{\dfrac{\dfrac{\vdots}{\llbracket \Theta \cdot \Gamma \rrbracket, z_{\{s,\mathtt{p}_1,\mathtt{q}_1\}} : \llbracket \mathrm{unf}(S_{\mathtt{p}_1} \restriction \mathtt{q}_1) \rrbracket \Cap \llbracket \mathrm{unf}(S_{\mathtt{q}_1} \restriction \mathtt{p}_1) \rrbracket \vdash (\boldsymbol{\nu} z_{\{s,\mathtt{p}_i,\mathtt{q}_i\}})_{i \in \{2..n\}} \llbracket P \rrbracket_{\Theta \cdot \Gamma, \Gamma'} \boldsymbol{\sigma}(\Gamma')}\ (\text{T}\pi\text{-Res1})}{\llbracket \Theta \cdot \Gamma \rrbracket \vdash \llbracket (\boldsymbol{\nu} s) \rrbracket \llbracket P \rrbracket_{\Theta \cdot \Gamma, \Gamma'} \boldsymbol{\sigma}(\Gamma')}\ (\text{T}\pi\text{-Res1})}_{\text{where } \mathrm{conn}(s, \Gamma') = \{\{\mathtt{p}_1, \mathtt{q}_1\}, \ldots, \{\mathtt{p}_n, \mathtt{q}_n\}\} \text{ and } \llbracket (\boldsymbol{\nu} s) \rrbracket = (\boldsymbol{\nu} z_{\{s,\mathtt{p}_i,\mathtt{q}_i\}})_{i \in \{1..n\}}}$$

where the elided derivation begins with

$$\dfrac{\dfrac{\llbracket \Theta \cdot \Gamma, \Gamma' \vdash P \rrbracket}{\llbracket \Theta \cdot \Gamma \rrbracket, \boldsymbol{\delta}(\Gamma') \vdash \llbracket P \rrbracket_{\Theta \cdot \Gamma, \Gamma'} \boldsymbol{\sigma}(\Gamma')}\ (\text{T}\pi\text{-Reify})}{\vdots}\ (\text{T}\pi\text{-Res1})$$

Figure 14 Encoding of multiparty session typing judgements (part 2).

▶ **Theorem 6.3** (Correctness of encoding). *$\Gamma \vdash v : U$ implies $[\![\Gamma]\!] \vdash [\![v]\!] : [\![U]\!]$, $\Theta \vdash X : \widetilde{U}$ implies $[\![\Theta]\!] \vdash [\![X]\!] : \widetilde{[\![U]\!]}$, and $\Theta \cdot \Gamma \vdash P$ implies $[\![\Theta \cdot \Gamma \vdash P]\!]$.*

Proof.(A) The thesis for $\Gamma \vdash v : U$ follows immediately by the encoding of rules (T-Basic) and (T-Name) in Fig. 13. Note that, in the premises of the encoded typing derivation, we use Proposition D.9.

(B) The thesis for $\Theta \vdash X : \widetilde{U}$ follows immediately by the encoding of rules (T-DefCtx) in Fig. 13. Note that the premise un($[\![\Theta]\!]$) holds because, by Def. 5.4, $[\![\Theta]\!]$ only contains names with unrestricted types.

The thesis for $\Theta \cdot \Gamma \vdash P$ is proved by induction on the derivation of the judgement, producing a π-calculus derivation that concludes $[\![\Theta]\!], [\![\Gamma]\!] \vdash [\![P]\!]_{\Theta' \cdot \Gamma'}$ (for some Θ', Γ' depending on the rule from Fig. 4). The possible cases are shown Fig. 13 and Fig. 14: in all cases, each encoded derivation is supported by premises that hold either by **(A)** or **(B)** above, or by the induction hypothesis. Here we discuss the crucial points of each case:

- (T-Def) and (T-Call). The derivations are self-explanatory. We just point out that the premise of (Tπ-Nil) in the latter holds by Proposition D.9;
- (T-Brch). The derivation is mostly self-explanatory, except for the topmost premises. The application of (Tπ-Sub) are needed to provide the required types to the names used to compose $\maltese_i$. Each relation holds because, for all $\mathtt{q} \in S \setminus \mathtt{p}$, we know that $S \restriction \mathtt{q} \leqslant_{\mathtt{P}} S'_i \restriction \mathtt{q}$ holds by Proposition B.13, which implies $[\![S \restriction \mathtt{q}]\!] \leqslant_\pi [\![S'_i \restriction \mathtt{q}]\!]$ by Cor. D.3. The (Tπ-Name) instance is only yielded when $S'_i \restriction \mathtt{p} \neq \mathbf{end}$, which implies that z is used to compose $\maltese_i$ (note that, in this case, to avoid cluttering the notation we are omitting a premise un($\varnothing$) required by (Tπ-LTup)); otherwise, by Def. 5.1 we have $[\![S'_i \restriction \mathtt{p}]\!] = \bullet$, and the premise is replaced by un($z : \bullet$), since z is *not* used to compose $\maltese_i$;
- (T-Sel). The derivation is, again, mostly self-explanatory. The type of the continuation name z is either $\bullet$ (and in this case, (Tπ-Res*) stands for (Tπ-Res2)) or a linear connection type $\mathsf{L}\sharp(T)$, where T is the carried type of the encoding of the *unfolded* partial projection $S' \restriction \mathtt{p}$ (and in this case, (Tπ-Res*) stands for (Tπ-Res1)). In the second case, the unfolding ensures that $[\![\mathrm{unf}(S' \restriction \mathtt{p})]\!]$ and $\overline{[\![\mathrm{unf}(S' \restriction \mathtt{p})]\!]}$ yield dual types $\mathsf{Li}(T)/\mathsf{Lo}(T)$ that can be composed with $\uplus$ (remind that $\uplus$ is *not* defined on μ-types). To correctly deal with such unfolding, the derivation has a branch with an instance of (Tπ-Sub) and premise $\overline{[\![\mathrm{unf}(S' \restriction \mathtt{p})]\!]} \leqslant_\pi \overline{[\![S' \restriction \mathtt{p}]\!]}$, that is necessary because the type of the variant being sent along $z_{\mathtt{p}}$ requires the type of the continuation to be exactly $\overline{[\![S' \restriction \mathtt{p}]\!]}$. Similarly, the derivation has another branch with (Tπ-Sub) and premise $[\![\mathrm{unf}(S' \restriction \mathtt{p})]\!] \leqslant_\pi [\![S' \restriction \mathtt{p}]\!]$: this is necessary because, when composing $\maltese$, the latter requires z to have exactly type $[\![S' \restriction \mathtt{p}]\!]$. Similarly to the encoding of (T-Brch), the instance of (Tπ-Name)/(Tπ-Sub) on the right (that types z) is only generated if $S' \restriction \mathtt{p} \neq \mathbf{end}$, which implies that z is used to compose $\maltese$ in (Tπ-LTup) (otherwise, z is *not* used and the premise of (Tπ-LTup) is replaced by un($z : \bullet$));
- (T-Sub) and (T-Res). These cases are discussed after the statement of Theorem 6.3 (page 21). In the latter, note that the instances of (Tπ-Res1) can be applied because by consistency and completeness of Γ', and by Def. 5.6, for all $i \in 1..n$, $[\![\mathrm{unf}(S_{\mathtt{p}_i} \restriction \mathtt{q}_i)]\!] \text{ ⩀ } [\![\mathrm{unf}(S_{\mathtt{q}_i} \restriction \mathtt{p}_i)]\!] = \mathsf{Li}(T) \uplus \mathsf{Lo}(T) = \mathsf{L}\sharp(T)$ (for some T).

◀

E Operational Correspondence

Lemma E.1 says that our encoding yields quasi-linear π-calculus processes (Def. C.5). In fact, sessions are encoded as (quasi-)linearly-typed π-calculus names, and the only unrestricted names are yielded by process declarations, under the constraints of Def. C.5 (item *(b)*).

▶ **Lemma E.1.** *If* $\Theta \cdot \Gamma \vdash P$, *then* $[\![\Theta \cdot \Gamma \vdash P]\!]$ *is quasi-linear.*

Lemma E.1 implies that our encoding produces *confluent* π-calculus processes, as per Theorem E.8.

In this section, we reuse and extend several results from [35]. For this purpose, we introduce slightly adapted notions of *normal form* (Def. E.2) and *annotated transition* (Def. E.5).

▶ **Definition E.2** (Guarded and normal-form processes)**.** *A π-calculus process is* guarded *iff it is either an input, output, case, record destructor, or let-binder. A process P* is in normal form *iff* $P = (\boldsymbol{\nu}\tilde{x})\,(P_1 \mid \ldots \mid P_m \mid *Q_1 \mid *Q_n)$ *where $P_1, \ldots, P_m$ are guarded.*

▶ **Proposition E.3** (Existence of normal form)**.** *For any process P there is some Q in normal form such that $P \equiv Q$.*

Proof. See [35, Lemma 4.1.3] and [54, Exercise 1.2.10]. ◀

▶ **Proposition E.4.** *For all π-calculus processes P, P', $\Gamma \vdash P$ and $P \equiv P'$ implies $\Gamma \vdash P'$.*

Proof. Standard result (see e.g. [35, Lemma 4.1.1]). ◀

▶ **Definition E.5** (Annotated transitions)**.** Transition annotations *are ranged over by $\alpha, \beta, \ldots$, and are defined as:*

$$\alpha, \beta, \ldots \quad ::= \quad x \mid \mathbf{case} \mid \mathbf{with} \mid \mathbf{let} \mid \tau$$

We define the annotated reduction relation $\xrightarrow{\alpha}$ *between π-calculus processes as follows:*

$$\begin{array}{rl}
(\text{R}\pi\text{-ComA}) & \overline{x}\langle v\rangle.P \mid x(y).Q \;\xrightarrow{x}\; P \mid Q\{^{v}/_{y}\} \\
(\text{R}\pi\text{-CaseA}) & \mathbf{case}\, l_j(v)\, \mathbf{of}\, \{l_i(x_i) \triangleright P_i\}_{i\in I} \;\xrightarrow{\mathbf{case}}\; P_j\{^{v}/_{x_j}\} \quad (j \in I) \\
(\text{R}\pi\text{-WithA}) & \mathbf{with}\, [l_i : x_i]_{i\in I} = [l_i : v_i]_{i\in I}\, \mathbf{do}\, P \;\xrightarrow{\mathbf{with}}\; P\{^{v_i}/_{x_i}\}_{i\in I} \\
(\text{R}\pi\text{-Let}) & \mathbf{let}\, x = v\, \mathbf{in}\, P \;\xrightarrow{\mathbf{let}}\; P\{^{v}/_{x}\} \\
(\text{R}\pi\text{-ResA1}) & P \xrightarrow{\alpha} Q \;\; \textit{implies} \;\; (\boldsymbol{\nu}x)P \xrightarrow{\tau} (\boldsymbol{\nu}x)Q \quad (\textit{if } \alpha = x) \\
(\text{R}\pi\text{-ResA2}) & P \xrightarrow{\alpha} Q \;\; \textit{implies} \;\; (\boldsymbol{\nu}x)P \xrightarrow{\alpha} (\boldsymbol{\nu}x)Q \quad (\textit{if } \alpha \neq x) \\
(\text{R}\pi\text{-ParA}) & P \xrightarrow{\alpha} Q \;\; \textit{implies} \;\; P \mid R \xrightarrow{\alpha} Q \mid R \\
(\text{R}\pi\text{-StructA}) & P \equiv P' \,\wedge\, P \xrightarrow{\alpha} Q \,\wedge\, Q' \equiv Q \;\; \textit{implies} \;\; P' \xrightarrow{\alpha} Q'
\end{array}$$

▶ **Lemma E.1.** *If* $\Theta \cdot \Gamma \vdash P$, *then* $[\![\Theta \cdot \Gamma \vdash P]\!]$ *is quasi-linear.*

Proof. By easy analysis of Figures 13 and 14. Note that case *(b)* of Def. C.5 covers the encoding of (T-Def) (which produces the $\sharp(T')$-typed x in $(\boldsymbol{\nu}x)\,(*(x(y).Q) \mid Q'))$ and (T-Call) (which can only occur within the scope of (T-Def), and produces the only uses of x, as outputs in Q, Q'). ◀

▶ **Remark E.6.** *Lemma E.1 implies that the encoded typing derivations for* (T-Def) *and* (T-Call) *in Fig. 13 could have been further strengthened by adopting the specialised types and typing rules for ω-receptiveness [54, §8.2.2]. However, we preferred to keep the typing rules in Fig. 5 as simple as possible.*

▶ **Corollary E.7.** *If* $\Theta \cdot \Gamma \vdash P$, *then* $[\![\Theta]\!], \boldsymbol{\delta}(\Gamma) \vdash [\![P]\!]\boldsymbol{\sigma}(\Gamma)$ *is quasi-linear.*

Proof. By Lemma E.1, $[\![\Theta \cdot \Gamma \vdash P]\!]$ is quasi-linear. By Def. 5.6, we have:

$$\frac{[\![\Theta \cdot \Gamma \vdash P]\!]}{[\![\Theta]\!], \boldsymbol{\delta}(\Gamma) \vdash [\![P]\!]\boldsymbol{\sigma}(\Gamma)}\ (\text{T}\pi\text{-Reify})$$

where the conclusion is obtained by replacing all names in $\operatorname{dom}(\llbracket\Gamma\rrbracket)$ with labelled tuples of linearly-typed names, introduced in the typing context by $\boldsymbol{\delta}(\Gamma)$. (as discussed on page 21). Therefore, by Def. C.5, all names introduced in the conclusion have a quasi-linear type; moreover, the syntactic structure of $\llbracket P\rrbracket\boldsymbol{\sigma}(\Gamma)$ is the same of $\llbracket P\rrbracket$: we conclude that $\llbracket P\rrbracket\boldsymbol{\sigma}(\Gamma)$ is quasi-linear, according to Def. C.5. ◀

▶ **Theorem E.8** (Encoding is confluent). *Whenever* $\llbracket P\rrbracket_{\Theta\cdot\Gamma}\boldsymbol{\sigma}(\Gamma) \to^* P_1$ *and* $\llbracket P\rrbracket_{\Theta\cdot\Gamma}\boldsymbol{\sigma}(\Gamma) \to^* P_2$, *then* $\exists P_3$ *such that* $P_1 \to^* P_3$ *and* $P_2 \to^* P_3$.

Proof. Note that, Cor. E.7, $\llbracket P\rrbracket\boldsymbol{\sigma}(\Gamma)$ is quasi-linear; and by Proposition C.6, all its reducts are quasi-linear. Letting n be the length of the sequence of reductions in $\llbracket P\rrbracket_{\Theta\cdot\Gamma}\boldsymbol{\sigma}(\Gamma) \to^* P_2$, we proceed by induction on n:

- base case $n = 0$. We have $P_2 = \llbracket P\rrbracket_{\Theta\cdot\Gamma}\boldsymbol{\sigma}(\Gamma)$, and we conclude by letting $P_3 = P_1$;
- inductive case $n = n' + 1$. We have $\llbracket P\rrbracket_{\Theta\cdot\Gamma}\boldsymbol{\sigma}(\Gamma) \to^* P_* \to P_2$, and by the induction hypothesis, either:
  - $P_* \equiv P_1$. We conclude by letting $P_3 = P_2$;
  - $\exists P_3'$ such that $P_1 \to^* P_3'$ and $P_* \to^* P_3'$. By $P_* \to P_2$ and Cor. C.12, we have either:
    - $P_2 \to^* P_3'$. We conclude by letting $P_3 = P_3'$;
    - $\exists P_3''$ such that $P_3' \to^* P_3''$ and $P_2 \to^* P_3''$. We conclude by letting $P_3 = P_3''$.

◀

▶ **Definition E.9** (Context narrowing). $\Gamma \leqslant_\pi \Gamma'$ *holds iff* $\operatorname{dom}(\Gamma) = \operatorname{dom}(\Gamma')$ *and* $\forall x \in \operatorname{dom}(\Gamma)$, *either: (a)* $\Gamma(x) \leqslant_\pi \Gamma'(x)$, *or (b)* $\Gamma(x) = \mathsf{Li}(T) \uplus \mathsf{Lo}(T) = \Gamma'(x)$.

▶ **Definition E.10** (Multi-narrowing). *The multi-narrowing typing rule for* π*-calculus is:*

$$\frac{\Gamma \vdash P \qquad \Gamma' \leqslant_\pi \Gamma}{\Gamma' \vdash P}\;(\text{T}\pi\text{-MNarrow})$$

▶ **Proposition E.11.** *Rule* (Tπ-MNarrow) *is sound.*

Proof. Assume $\Gamma \vdash P$ and $\Gamma' \leqslant_\pi \Gamma$. We can rewrite the proof of $\Gamma \vdash P$ into a proof concluding $\Gamma' \vdash P$, noticing that, by Def. E.9, for all $x \in \operatorname{dom}\Gamma$, we have either:

- clause *(a)*: $\Gamma'(x) \leqslant_\pi \Gamma(x)$. In this case, the proof of $\Gamma \vdash P$ can be adapted to use $\Gamma'(x)$ instead of $\Gamma(x)$, by applying of the classical narrowing lemma [54, 7.2.5];
- clause *(b)*: $\Gamma(x) = T_1 \uplus T_2 = \Gamma'(x)$. In this case, the proof proof of $\Gamma \vdash P$ can be trivially adapted to use $\Gamma'(x)$ instead of $\Gamma(x)$.

◀

▶ **Definition E.12** (Context subtyping). *We define the encoding of an instance of* (T-MSub) *(Def. B.27) as:*

$$\left\llbracket \frac{\Theta \cdot \Gamma_{\mathsf{S}} \vdash P \qquad \Gamma'_{\mathsf{S}} \leqslant_{\mathsf{S}} \Gamma_{\mathsf{S}}}{\Theta \cdot \Gamma'_{\mathsf{S}} \vdash P}\,(\text{T-MSub}) \right\rrbracket = \frac{\llbracket \Theta \cdot \Gamma_{\mathsf{S}} \vdash P\rrbracket \qquad \llbracket \Theta \cdot \Gamma'_{\mathsf{S}}\rrbracket \leqslant_\pi \llbracket \Theta \cdot \Gamma_{\mathsf{S}}\rrbracket}{\llbracket \Theta \cdot \Gamma'_{\mathsf{S}}\rrbracket \vdash \llbracket P\rrbracket_{\Theta\cdot\Gamma_{\mathsf{S}}}}\,(\text{T}\pi\text{-MNarrow})$$

▶ **Proposition E.13.** *If* $\Theta \cdot \Gamma \vdash P$ *by rule* (T-MSub)*, then* $\llbracket \Theta \cdot \Gamma \vdash P\rrbracket$ *holds.*

Proof. By induction on the typing derivation of $\Theta \cdot \Gamma \vdash P$, noticing that $\operatorname{dom}(\llbracket\Theta\cdot\Gamma\rrbracket) = \operatorname{dom}(\llbracket\Theta\cdot\Gamma'\rrbracket)$, and $\forall x \in \operatorname{dom}(\llbracket\Gamma\rrbracket) : \llbracket\Gamma\rrbracket(x) \leqslant_\pi \llbracket\Gamma'\rrbracket(x)$ by Def. B.27 and Theorem 6.2. ◀

▶ **Theorem 6.4** (Precise decomposition). Γ_{S} *is consistent* if and only if $\boldsymbol{\delta}(\Gamma_{\mathsf{S}})$ *is defined.*

Proof. ($\Longrightarrow$). Assume that Γ_S is consistent. We proceed by induction on the size of Γ_S. In the base case $\Gamma_S = \emptyset$, we simply conclude noticing that $\boldsymbol{\delta}(\Gamma_S) = \emptyset$. For the inductive case, for some Γ'_S we have $\Gamma_S = \Gamma'_S, s[\mathsf{p}]{:}S_{\mathsf{p}}$, with $\boldsymbol{\delta}(\Gamma'_S)$ defined (by the induction hypothesis), and two possibilities:

- $\not\exists \mathsf{q} \neq \mathsf{p}$ such that $s[\mathsf{q}] \in \mathrm{dom}(\Gamma'_S)$ (i.e., s does not occur in Γ'_S). This implies $z_{\{s,\mathsf{p},\mathsf{q}\}} \notin \mathrm{dom}(\boldsymbol{\delta}(\Gamma'_S))$; therefore, by Def. 5.6, we conclude that $\boldsymbol{\delta}(\Gamma_S)$ is defined as:

$$\boldsymbol{\delta}(\Gamma_S) \;=\; \boldsymbol{\delta}\big(\Gamma'_S\big) \Cap z_{\{s,\mathsf{p},\mathsf{q}\}} : [\![\mathrm{unf}(S_{\mathsf{p}} \restriction \mathsf{q})]\!] \;=\; \boldsymbol{\delta}\big(\Gamma'_S\big), z_{\{s,\mathsf{p},\mathsf{q}\}} : [\![\mathrm{unf}(S_{\mathsf{p}} \restriction \mathsf{q})]\!]$$

- $\exists \mathsf{q} \neq \mathsf{p}$ such that $s[\mathsf{q}]{:}S_{\mathsf{q}} \in \Gamma'_S$. This implies $\Gamma'_S = \Gamma''_S, s[\mathsf{q}]{:}S_{\mathsf{q}}$. By Def. 2.11, we also have $\overline{S_{\mathsf{p}} \restriction \mathsf{q}} \leqslant_{\mathsf{P}} S_{\mathsf{q}} \restriction \mathsf{p}$, and therefore:

$$\begin{array}{ll} \overline{[\![S_{\mathsf{p}} \restriction \mathsf{q}]\!]} \leqslant_\pi [\![S_{\mathsf{q}} \restriction \mathsf{p}]\!] & \text{(by Cor. D.3)} \\ \mathrm{unf}\big(\overline{[\![S_{\mathsf{p}} \restriction \mathsf{q}]\!]}\big) \leqslant_\pi \mathrm{unf}([\![S_{\mathsf{q}} \restriction \mathsf{p}]\!]) & \text{(by (S-L}\mu\text{L) and (S-L}\mu\text{R))} \end{array}$$

  This implies that we can have three cases:

  - $\mathrm{unf}\big(\overline{[\![S_{\mathsf{p}} \restriction \mathsf{q}]\!]}\big) = \mathrm{unf}([\![S_{\mathsf{q}} \restriction \mathsf{p}]\!]) = \bullet$. This implies $\mathrm{unf}([\![S_{\mathsf{p}} \restriction \mathsf{q}]\!]) = \mathrm{unf}([\![S_{\mathsf{q}} \restriction \mathsf{p}]\!]) = \bullet$;
  - $\mathrm{unf}\big(\overline{[\![S_{\mathsf{p}} \restriction \mathsf{q}]\!]}\big) = \mathsf{Li}(T)$ and $\mathrm{unf}([\![S_{\mathsf{q}} \restriction \mathsf{p}]\!]) = \mathsf{Li}(T')$ with $T \leqslant_\pi T'$. By Theorem 6.1, this implies $\mathrm{unf}([\![S_{\mathsf{p}} \restriction \mathsf{q}]\!]) = \overline{\mathsf{Li}(T)} = \mathsf{Lo}(T)$;
  - $\mathrm{unf}\big(\overline{[\![S_{\mathsf{p}} \restriction \mathsf{q}]\!]}\big) = \mathsf{Lo}(T)$ and $\mathrm{unf}([\![S_{\mathsf{q}} \restriction \mathsf{p}]\!]) = \mathsf{Lo}(T')$ with $T' \leqslant_\pi T$. By Theorem 6.1, this implies $\mathrm{unf}([\![S_{\mathsf{p}} \restriction \mathsf{q}]\!]) = \overline{\mathsf{Lo}(T)} = \mathsf{Li}(T)$.

  In all cases, we can verify that $\mathrm{unf}([\![S_{\mathsf{p}} \restriction \mathsf{q}]\!]) \Cap \mathrm{unf}([\![S_{\mathsf{q}} \restriction \mathsf{p}]\!])$ is defined, by Def. 4.6. Therefore, by Def. 5.6, noticing that $z_{\{s,\mathsf{q},\mathsf{p}\}} = z_{\{s,\mathsf{p},\mathsf{q}\}} \notin \mathrm{dom}(\boldsymbol{\delta}(\Gamma''_S))$, and that $\boldsymbol{\delta}(\Gamma''_S)$ is defined (by the induction hypothesis), we conclude that $\boldsymbol{\delta}(\Gamma_S)$ is defined as:

$$\boldsymbol{\delta}(\Gamma_S) \;=\; \boldsymbol{\delta}\big(\Gamma''_S\big),\, z_{\{s,\mathsf{p},\mathsf{q}\}} : \mathrm{unf}([\![S_{\mathsf{p}} \restriction \mathsf{q}]\!]) \Cap \mathrm{unf}([\![S_{\mathsf{q}} \restriction \mathsf{p}]\!])$$

($\Longleftarrow$). We prove the contrapositive. Assume that $\boldsymbol{\delta}(\Gamma_S)$ is *not* defined. Examining Def. 5.6, we can see that this can only occur if some application of $\Cap$ is *not* defined, i.e., there exist some $s[\mathsf{p}]{:}S_{\mathsf{p}}, s[\mathsf{q}]{:}S_{\mathsf{q}} \in \Gamma'_S$ with $\mathsf{p} \neq \mathsf{q}$ such that $\mathrm{unf}([\![S_{\mathsf{p}} \restriction \mathsf{q}]\!]) \Cap \mathrm{unf}([\![S_{\mathsf{q}} \restriction \mathsf{p}]\!])$ (i.e., the type for $z_{\{s,\mathsf{p},\mathsf{q}\}}$) is *not* defined. By Def. 4.6, we can have the following four cases:

- $\mathrm{unf}([\![S_{\mathsf{p}} \restriction \mathsf{q}]\!]) = \bullet$ and $\mathrm{unf}([\![S_{\mathsf{q}} \restriction \mathsf{p}]\!]) \neq \bullet$;
- $\mathrm{unf}([\![S_{\mathsf{p}} \restriction \mathsf{q}]\!]) \neq \bullet$ and $\mathrm{unf}([\![S_{\mathsf{q}} \restriction \mathsf{p}]\!]) = \bullet$;
- $\mathrm{unf}([\![S_{\mathsf{p}} \restriction \mathsf{q}]\!]) = \mathsf{Li}(T)$ and $\mathrm{unf}([\![S_{\mathsf{q}} \restriction \mathsf{p}]\!]) \neq \mathsf{Lo}(T')$ with $T' \leqslant_\pi T$;
- $\mathrm{unf}([\![S_{\mathsf{p}} \restriction \mathsf{q}]\!]) = \mathsf{Lo}(T)$ and $\mathrm{unf}([\![S_{\mathsf{q}} \restriction \mathsf{p}]\!]) \neq \mathsf{Li}(T')$ with $T \leqslant_\pi T'$.

In all cases, we obtain:

$$\begin{array}{ll} \overline{\mathrm{unf}([\![S_{\mathsf{p}} \restriction \mathsf{q}]\!])} \not\leqslant_\pi \mathrm{unf}([\![S_{\mathsf{q}} \restriction \mathsf{p}]\!]) & \\ \mathrm{unf}\big(\overline{[\![S_{\mathsf{p}} \restriction \mathsf{q}]\!]}\big) \not\leqslant_\pi \mathrm{unf}([\![S_{\mathsf{q}} \restriction \mathsf{p}]\!]) & \text{(by Lemma 4.2)} \\ \mathrm{unf}\big([\![\overline{S_{\mathsf{p}} \restriction \mathsf{q}}]\!]\big) \not\leqslant_\pi \mathrm{unf}([\![S_{\mathsf{q}} \restriction \mathsf{p}]\!]) & \text{(by the contrapositive of Cor. D.3)} \\ [\![\overline{S_{\mathsf{p}} \restriction \mathsf{q}}]\!] \not\leqslant_\pi [\![S_{\mathsf{q}} \restriction \mathsf{p}]\!] & \text{(by contrapositive of (S-L}\mu\text{L) and (S-L}\mu\text{R))} \\ \overline{S_{\mathsf{p}} \restriction \mathsf{q}} \not\leqslant_{\mathsf{P}} S_{\mathsf{q}} \restriction \mathsf{p} & \text{(by the contrapositive of Proposition B.8)} \end{array}$$

Hence, by Def. 2.11, we conclude that Γ_S is *not* consistent. ◀

▶ **Proposition E.14.** *If* $\Gamma_S \leqslant_S \Gamma'_S$*, then* $\boldsymbol{\sigma}(\Gamma_S) = \boldsymbol{\sigma}(\Gamma'_S)$.

Proof. Assume $\Gamma_S \leqslant_S \Gamma'_S$. Then:

$$\begin{array}{ll} \mathrm{dom}(\Gamma_S) = \mathrm{dom}(\Gamma'_S) \quad \text{and} \quad \forall s[\mathsf{p}] \in \mathrm{dom}(\Gamma_S) : \Gamma_S(s[\mathsf{p}]) \leqslant_S \Gamma'_S(s[\mathsf{p}]) & \text{(by Def. B.27)} \\ s[\mathsf{p}] \in \mathrm{dom}(\Gamma_S) : \mathrm{roles}(\Gamma_S(s[\mathsf{p}])) = \mathrm{roles}(\Gamma'_S(s[\mathsf{p}])) & \text{(by Proposition B.11)} \\ \mathrm{conn}(s, \Gamma_S) = \mathrm{conn}(s, \Gamma'_S) & \text{(by Def. 5.5)} \end{array}$$

Therefore, by Def. 5.6, we conclude $\boldsymbol{\sigma}(\Gamma_S) = \boldsymbol{\sigma}(\Gamma'_S)$. ◀

▶ **Proposition E.15.** *If* $\Gamma \vdash P$*, then* $\mathrm{fn}(P) \subseteq \mathrm{dom}\,(\Gamma)$ *and* $\forall x \in (\mathrm{dom}\,(\Gamma) \setminus \mathrm{fn}(P)) : \mathrm{un}(\Gamma(x))$.

Proof. By induction on the derivation of $\Gamma \vdash P$. ◀

▶ **Proposition E.16.** *If* $\Theta \cdot \Gamma \vdash P$*, then* $\mathrm{fc}(P) \subseteq \mathrm{dom}\,(\Gamma)$ *and* $\forall c \in (\mathrm{dom}\,(\Gamma) \setminus \mathrm{fc}(P)) : \Gamma(c) = \mathbf{end}$.

Proof. By induction on the derivation of $\Theta \cdot \Gamma \vdash P$. ◀

▶ **Proposition E.17.** *If* $\Theta \cdot \Gamma \vdash P$ *and* $\Theta' \cdot \Gamma' \vdash P$*, then:*

1. $\forall c \in (\mathrm{dom}\,(\Gamma) \cap \mathrm{dom}\,(\Gamma')) : \mathrm{roles}(\Gamma(c)) = \mathrm{roles}(\Gamma'(c))$*;*
2. $\forall c \in (\mathrm{dom}\,(\Gamma) \setminus \mathrm{dom}\,(\Gamma')) : \mathrm{roles}(\Gamma(c)) = \emptyset$.
3. $\forall c \in (\mathrm{dom}\,(\Gamma') \setminus \mathrm{dom}\,(\Gamma)) : \mathrm{roles}(\Gamma'(c)) = \emptyset$.

Proof. Item 1 is proved by induction on the derivation of $\Theta \cdot \Gamma \vdash P$, noticing that Θ, Θ' are irrelevant for the statement, while Γ, Γ' can only differ by adding/removing an instance of rule (T-Sub), so that $\Gamma(c) \leqslant_{\mathsf{S}} \Gamma'(c)$ or $\Gamma'(c) \leqslant_{\mathsf{S}} \Gamma(c)$; in both cases, we conclude by Proposition B.11. Items 2 and 3 are a consequence of Proposition E.16. ◀

▶ **Proposition 6.5.** *If* $\Theta \cdot \Gamma \vdash P$ *and* $\Theta' \cdot \Gamma' \vdash P$*, then* $[\![P]\!]_{\Theta \cdot \Gamma} = [\![P]\!]_{\Theta' \cdot \Gamma'}$.

Proof. By inspecting Fig. 7, we can observe that the only typing information used to generate $[\![P]\!]_{\Theta \cdot \Gamma}$ and $[\![P]\!]_{\Theta' \cdot \Gamma'}$ is the set of participants involved in each open session (for processes typed by (T-Brch), (T-Sel), (T-Res)), and this does not change between Γ and Γ', by Proposition E.17. ◀

▶ **Proposition E.18.** *If* $\Theta \cdot \Gamma \vdash P$*, (i)* $x{:}T \in [\![\Theta]\!]$ *implies* $T = \sharp(T')$ *and (ii)* $x{:}T \in [\![\Gamma]\!]$ *implies* $\mathrm{qlin}(T)$.

Proof. Straightforward by Def. 5.4. ◀

▶ **Definition E.19** (Guarded and normal-form processes)**.** *A multiparty session process is* guarded *iff it has the form* $s[\mathsf{q}][\mathsf{p}] \oplus \langle l(v) \rangle.P'$ *or* $s[\mathsf{p}][\mathsf{q}]\,\&_{i \in I}\,\{l_i(x_i).P_i\}$*. A multiparty session process* P is in normal form *iff*

$$P \;=\; \mathbf{def}\;\widetilde{D}\;\mathbf{in}\;(\boldsymbol{\nu}\widetilde{s_*})\big(Q_1 \mid \ldots \mid Q_m \;\mid\; Y_1\langle\widetilde{v_1}\rangle \mid \ldots \mid Y_{m'}\langle\widetilde{v_{m'}}\rangle\big)$$

where $Q_1, \ldots, Q_m$ *are guarded.*

▶ **Proposition E.20.** *If* $\Gamma \vdash P \xrightarrow{\mathbf{with}} P'$*, then* $\Gamma \vdash P'$.

Proof. Standard subject reduction property for π-calculus with linear types (see [54, Theorem 8.1.5]). ◀

▶ **Proposition E.21.** *If* $\Theta \cdot \Gamma \vdash P$*,*

(i) $x{:}T \in [\![\Theta]\!]$ *implies* $T = \sharp(T')$ *and*

(ii) $x{:}T \in \boldsymbol{\delta}(\Gamma)$ *implies* $\mathrm{qlin}(T) \in \{\mathsf{Li}(T'), \mathsf{Lo}(T'), \mathsf{L}\sharp(T')\}$ *(for some* T'*).*

Proof. Straightforward by Proposition E.18 and Def. 5.6. ◀

▶ **Proposition E.22.** *If* $\Theta \cdot \Gamma \vdash P \equiv P'$*, then* $[\![P]\!]_{\Theta \cdot \Gamma} \equiv [\![P']\!]_{\Theta \cdot \Gamma}$ *and* $[\![P]\!]_{\Theta \cdot \Gamma}\boldsymbol{\sigma}(\Gamma) \equiv [\![P']\!]_{\Theta \cdot \Gamma}\boldsymbol{\sigma}(\Gamma)$.

Proof. We can prove $[\![P]\!]_{\Theta \cdot \Gamma} \equiv [\![P']\!]_{\Theta \cdot \Gamma}$ by induction on the derivation of $P \equiv P'$. Then, the last part of the statement is straightforward. ◀

▶ **Lemma E.23.** *If* $\Theta \cdot \Gamma \vdash P$*:*

$$\dfrac{[\![\Theta \cdot \Gamma \vdash P]\!]}{[\![\Theta]\!], \boldsymbol{\delta}(\Gamma) \vdash [\![P]\!]\boldsymbol{\sigma}(\Gamma)}\;(\text{T}\pi\text{-Reify}) \quad P_0 \to P^*$$

$\xrightarrow{\mathbf{with}}{}^*$ *implies that* $\exists \tilde{x}, P'', \Gamma', P'$ *such that* $P_* \to^* (\boldsymbol{\nu}\tilde{x})P''$ *and*

$$\dfrac{[\![\Theta \cdot \Gamma' \vdash P']\!]}{[\![\Theta]\!], \boldsymbol{\delta}(\Gamma') \vdash [\![P']\!]\boldsymbol{\sigma}(\Gamma')}\;(\text{T}\pi\text{-Reify}) \quad P''$$

$\xrightarrow{\mathbf{with}}{}^*$

and $P \to P'$.

Proof. Assume $\Theta \cdot \Gamma \vdash P$. We first collect several facts that we will use in the proof later on. By structural equivalence [10, Proof of Theorem 1], and by Proposition B.25, we have:

$$\Theta \cdot \Gamma \vdash \mathbf{def}\ \widetilde{D}\ \mathbf{in}\ (\boldsymbol{\nu}\widetilde{s_*})(Q_1 \mid \ldots \mid Q_m \mid Q_Y) \equiv P \qquad \text{where } \widetilde{D} = X_1(\widetilde{x_1}) = P_{X_1}, \ldots, X_n(\widetilde{x_1}) = P_{X_n} \tag{73}$$

$$\text{for some} \quad X_1, \ldots, X_n, P_{X_1}, \ldots, P_{X_n}, \quad \text{guarded (Def. E.19)} Q_1, \ldots, Q_m \quad \text{and} \quad Q_Y = Y_1\langle\widetilde{v_1}\rangle \mid \ldots \mid Y_{m'}\langle\widetilde{v_{m'}}\rangle \tag{74}$$

From (73), by Proposition E.22 and Def. 5.7, we have:

$$[\![P]\!]_{\Theta\cdot\Gamma} \equiv (\boldsymbol{\nu}[\![X_1]\!]) \ldots (\boldsymbol{\nu}[\![X_n]\!])[\![(\boldsymbol{\nu}\widetilde{s_*})]\!]\ (\ [\![Q_1]\!]_{\Theta_X\cdot\Gamma_1} \mid \ldots \mid [\![Q_m]\!]_{\Theta_X\cdot\Gamma_m} \mid [\![Q_Y]\!]_{\Theta_X\cdot\Gamma_Y} \mid P_X\ ) \tag{75}$$

$$\text{where} \quad \Gamma_1 \circ \ldots \circ \Gamma_m \circ \Gamma_Y = \Gamma \quad \text{and} \quad \Theta_X = \Theta, X_1 {:} \widetilde{U_1}, \ldots, X_n {:} \widetilde{U_n} \quad \text{and} \quad \forall i \in 1..m' : Y_i \in \operatorname{dom}(\Theta) \tag{76}$$

$$\text{and} \quad \Gamma_Y = \Gamma_{Y_1} \circ \ldots \circ \Gamma_{Y_{m'}} \tag{77}$$

and where $[\![(\boldsymbol{\nu}\widetilde{s_*})]\!]$ is a sequence of restrictions yielded by the encoding of (T-Res) (Fig. 14), and

$$P_X = *\Big([\![X_1]\!](z).\mathbf{with}\ (\widetilde{x_1}) = z\ \mathbf{do}\ [\![P_{X_1}]\!]_{\Theta, X_1:\widetilde{U_1}\cdot\widetilde{x_1}:\widetilde{U_1}}\Big) \mid \ldots \mid *\Big([\![X_n]\!](z).\mathbf{with}\ (\widetilde{x_n}) = z\ \mathbf{do}\ [\![P_{X_n}]\!]_{\Theta, X_1:\widetilde{U_1},\ldots,X_n:\widetilde{U_n}\cdot\widetilde{x_n}:\widetilde{U_n}}\Big) \tag{78}$$

i.e., P_X is a parallel composition of input-guarded replicated processes, corresponding to the encodings of $P_{X_1}, \ldots, P_{X_n}$.

From (74) and Def. E.19, for all $i \in 1..m$, we have:

$$\text{for some } c_i, \mathsf{q}, I\text{:} \quad Q_i = c_i[\mathsf{q}]\&_{j\in I}\{l_j(x_j).Q''_{ij}\} \qquad \text{or} \qquad \text{for some } c'_i, \mathsf{p}\text{:} \quad Q_i = c'_i[\mathsf{p}] \oplus \langle l(v)\rangle.Q''_i \tag{79}$$

This implies that for all $i \in 1..m$ (to avoid cluttering the notation, in the following we will omit an i-index on S_c and $S_{c'}$, which will be clear from the context):

c_i in (79) must be typed by some branching type $S_c = \mathsf{q}\,\&_{j\in I_i}\,?l_j(U'_j).S'_j$
c'_i in (79) must be typed by some selection type $S_{c'} = \mathsf{p} \oplus !l(U'').S''$

Moreover, we can assume that $\Theta_X \cdot \Gamma_i \vdash Q_i$ holds by a (possibly vacuous) subtyping on c_i or c'_i, as per Proposition B.30:

$$\text{(T-MSub)}\ \frac{\Theta_X \cdot \Gamma_i^\diamond \vdash Q_i \qquad \Gamma_i \leqslant_{\mathsf{S}} \Gamma_i^\diamond}{\Theta_X \cdot \Gamma_i \vdash Q_i} \qquad \text{where (by Def. B.27) } \operatorname{dom}(\Gamma_i) = \operatorname{dom}(\Gamma_i^\diamond)\text{, and } \begin{cases} \Gamma_i(c_i) \leqslant_{\mathsf{S}} \Gamma_i^\diamond(c_i) = S_c \\ \text{or} \\ \Gamma_i(c'_i) \leqslant_{\mathsf{S}} \Gamma_i^\diamond(c'_i) = S_{c'} \end{cases} \tag{80}$$

At this point, we can observe that each **with**-reduction in $[\![P]\!]_{\Theta\cdot\Gamma}\boldsymbol{\sigma}(\Gamma) \xrightarrow{\mathbf{with}}{}^* P_0$ can only be induced by some **with**-prefix occurring in $[\![Q_1]\!]_{\Theta_X\cdot\Gamma_1}, \ldots, [\![Q_m]\!]_{\Theta_X\cdot\Gamma_m}$ in (75); moreover, since $Q_1, \ldots, Q_m$ are typed by (T-Brch)/(T-Sel), by examining Fig. 13 we can see that for all $i \in 1..m$, $[\![Q_i]\!]_{\Theta_X\cdot\Gamma_i}$ has *exactly one* top-level **with**-prefix, followed by an input/output on a linearly-typed name.

We will now focus on those $i \in 1..m$ where c_i and c'_i above are channels with roles: omitting some indexing on $i \in 1..m$, we consider $c_i = s[\mathsf{p}]$ typed by some S_{p} (for some s, p) or $c'_i = s'[\mathsf{q}]$ typed by some S_{q} (for some s', q) (the cases where c_i/c'_i are session-typed variables are similar). Note that (80) becomes:

$$\text{(T-MSub)}\ \frac{\Theta_X \cdot \Gamma_i^\diamond \vdash Q_i \qquad \Gamma_i \leqslant_{\mathsf{S}} \Gamma_i^\diamond}{\Theta_X \cdot \Gamma_i \vdash Q_i} \qquad \text{where } \begin{cases} \Gamma_i(s[\mathsf{p}]) \leqslant_{\mathsf{S}} \Gamma_i^\diamond(s[\mathsf{p}]) = S_{\mathsf{p}} \\ \text{or} \\ \Gamma_i(s'[\mathsf{q}]) \leqslant_{\mathsf{S}} \Gamma_i^\diamond(s'[\mathsf{q}]) = S_{\mathsf{q}} \end{cases} \tag{81}$$

Summing up, for all $i \in 1..m$ where c_i, c'_i in (79) are channels with roles, we have the following properties and encodings of Q_i (note that (T-MSub) in (80) is encoded as (Tπ-MNarrow), as per Def. E.12):

$$\begin{cases} \text{for some } s, \mathtt{p}, \mathtt{q}, I, S_{\mathtt{p}},\ \ Q_i = s[\mathtt{p}][\mathtt{q}]\&_{j\in I}\{l_j(x_j).Q''_{ij}\} \quad \text{with } \Gamma_i(s[\mathtt{p}]) \leqslant_{\mathsf{S}} S_{\mathtt{p}} = \mathtt{q}\,\&_{j\in I}\, ?l_j(U'_j).S'_j \text{ and} \\ \llbracket Q_i \rrbracket_{\Theta_X \Gamma_i} \boldsymbol{\sigma}(\Gamma_i) = \left(\mathbf{with}\, [\mathtt{r}{:}z_{\mathtt{r}}]_{\mathtt{r}\in S_{\mathtt{p}}} = \llbracket s[\mathtt{p}] \rrbracket \,\mathbf{do}\, z_{\mathtt{q}}(y).\mathbf{case}\, y \,\mathbf{of} \left\{ \begin{array}{l} l_j(z_j) \triangleright \mathbf{with}\,(x_j,z) = z_j \,\mathbf{do} \\ \qquad \mathbf{let}\, \llbracket s[\mathtt{p}] \rrbracket = \maltese_j \,\mathbf{in}\, \llbracket Q''_{ij} \rrbracket_{\Theta_X \cdot \Gamma_i^{\diamond}\{S'_j/s[\mathtt{p}]\}, x_j:U'_j} \end{array} \right\}_{j\in I} \right) \boldsymbol{\sigma}(\Gamma_i) \\ \qquad \text{where } \ \forall j \in I : \maltese_j = \begin{cases} [\mathtt{q}: z,\ \mathtt{r}: z_{\mathtt{r}}]_{\mathtt{r}\in S'_j \setminus \mathtt{q}} & \text{if } \mathtt{q} \in S'_j \\ [\mathtt{r}: z_{\mathtt{r}}]_{\mathtt{r}\in S'_j} & \text{otherwise} \end{cases} \\ \text{or} \\ \text{for some } s', \mathtt{p}, \mathtt{q}, S_{\mathtt{q}},\ \ Q_i = s'[\mathtt{q}][\mathtt{p}] \oplus \langle l_i(v)\rangle.Q''_i \quad \text{with } \Gamma_i(s'[\mathtt{q}]) \leqslant_{\mathsf{S}} S_{\mathtt{q}} = \mathtt{p} \oplus !l(U'').S'' \text{ and} \\ \llbracket Q_i \rrbracket_{\Theta_X \cdot \Gamma_i} \boldsymbol{\sigma}(\Gamma_i) = \left(\mathbf{with}\, [\mathtt{r}: z_{\mathtt{r}}]_{\mathtt{r}\in S_{\mathtt{q}}} = \llbracket s'[\mathtt{q}] \rrbracket \,\mathbf{do}\, (\boldsymbol{\nu} z)\overline{z_{\mathtt{p}}}\langle l_i(\llbracket v \rrbracket, z)\rangle.\mathbf{let}\, \llbracket s'[\mathtt{q}] \rrbracket = \maltese \,\mathbf{in}\, \llbracket Q''_i \rrbracket_{\Theta_X \cdot \Gamma_i^{\diamond}\{S''/s'[\mathtt{q}]\}\setminus v} \right) \boldsymbol{\sigma}(\Gamma_i) \\ \qquad \text{where } \ \maltese = \begin{cases} [\mathtt{p}: z,\ \mathtt{r}: z_{\mathtt{r}}]_{\mathtt{r}\in S''\setminus \mathtt{p}} & \text{if } \mathtt{p} \in S'' \\ [\mathtt{r}: z_{\mathtt{r}}]_{\mathtt{r}\in S''} & \text{otherwise} \end{cases} \end{cases} \tag{82}$$

By applying the substitutions $\boldsymbol{\sigma}(\Gamma_i)$ in (82), for all $i \in 1..m$ we get either (note that $\llbracket s[\mathtt{p}] \rrbracket$ and $\llbracket s'[\mathtt{q}] \rrbracket$ are rebound by **let**):

$$\begin{array}{l} \llbracket Q_i \rrbracket_{\Theta_X \cdot \Gamma_i} \boldsymbol{\sigma}(\Gamma_i) \ = \\ \mathbf{with}\, [\mathtt{r}{:}z_{\mathtt{r}}]_{\mathtt{r}\in S_{\mathtt{p}}} = \left[\mathtt{r}{:}z_{\{s,\mathtt{p},\mathtt{r}\}}\right]_{\mathtt{r}\in S_{\mathtt{p}}} \mathbf{do}\, z_{\mathtt{q}}(y).\mathbf{case}\, y \,\mathbf{of} \left\{ \begin{array}{r} l_j(z_j) \triangleright \mathbf{with}\,(x_j,z) = z_j \,\mathbf{do}\,\mathbf{let}\, \llbracket s[\mathtt{p}] \rrbracket = \maltese_j \,\mathbf{in}\, \left(\llbracket Q''_{ij} \rrbracket_{\Theta_X \cdot \Gamma_i^{\diamond}\{S'_j/s[\mathtt{p}]\}, x_j:U'_j} \right) \\ \left(\boldsymbol{\sigma}(\Gamma_i) \setminus \mathrm{dom}\,(\boldsymbol{\sigma}(s[\mathtt{p}]:S_{\mathtt{p}})) \right) \end{array} \right\}_{j\in I} \\ \text{or} \\ \llbracket Q_i \rrbracket_{\Theta_X \cdot \Gamma_i} \boldsymbol{\sigma}(\Gamma_i) \ = \\ \mathbf{with}\, [\mathtt{r}: z_{\mathtt{r}}]_{\mathtt{r}\in S_{\mathtt{q}}} = \left[\mathtt{r}: z_{\{s',\mathtt{q},\mathtt{r}\}}\right]_{\mathtt{r}\in S_{\mathtt{q}}} \mathbf{do}\, (\boldsymbol{\nu} z)\overline{z_{\mathtt{p}}}\langle l_i(\llbracket v \rrbracket, z)\rangle.\mathbf{let}\, \llbracket s'[\mathtt{q}] \rrbracket = \maltese \,\mathbf{in}\, \left(\llbracket Q''_i \rrbracket_{\Theta_X \cdot \Gamma_i^{\diamond}\{S''/s'[\mathtt{q}]\}\setminus v} \right) \left(\boldsymbol{\sigma}(\Gamma_i) \setminus \mathrm{dom}\,(\boldsymbol{\sigma}(s'[\mathtt{q}]:S_{\mathtt{q}})) \right) \end{array} \tag{83}$$

Hence, there exist $Q'_1, \ldots, Q'_m$ such that (to save some symbols, we will now redefine $\maltese_j$ and $\maltese$ by taking the corresponding definitions in (82) and applying the substitutions induced by **with**):

$$\begin{array}{rl} \llbracket P \rrbracket_{\Theta \cdot \Gamma} \boldsymbol{\sigma}(\Gamma) \equiv & (\boldsymbol{\nu}\llbracket X_1 \rrbracket) \ldots (\boldsymbol{\nu}\llbracket X_n \rrbracket)\llbracket(\boldsymbol{\nu}\widetilde{s_*})\rrbracket \left(\llbracket Q_1 \rrbracket_{\Theta_X \cdot \Gamma_1} \boldsymbol{\sigma}(\Gamma_1) \mid \ldots \mid \llbracket Q_m \rrbracket_{\Theta_X \cdot \Gamma_m} \boldsymbol{\sigma}(\Gamma_m) \mid \llbracket Q_Y \rrbracket_{\Theta_X \cdot \Gamma_Y} \boldsymbol{\sigma}(\Gamma_Y) \mid P_X \right) \\ \xrightarrow{\mathbf{with}}{}^{*} & (\boldsymbol{\nu}\llbracket X_1 \rrbracket) \ldots (\boldsymbol{\nu}\llbracket X_n \rrbracket)\llbracket(\boldsymbol{\nu}\widetilde{s_*})\rrbracket \left(Q'_1 \mid \ldots \mid Q'_m \mid \llbracket Q_Y \rrbracket_{\Theta_X \cdot \Gamma_Y} \boldsymbol{\sigma}(\Gamma_Y) \mid P_X \right) \ \equiv \ P_0 \end{array} \tag{84}$$

$$\text{where } \forall i \in 1..m \ \begin{cases} Q'_i = \llbracket Q_i \rrbracket_{\Theta_X \cdot \Gamma_i} \boldsymbol{\sigma}(\Gamma_i) \\ \text{or} \\ \llbracket Q_i \rrbracket_{\Theta_X \cdot \Gamma_i} \boldsymbol{\sigma}(\Gamma_i) \xrightarrow{\mathbf{with}} Q'_i \ \text{ with } \ \llbracket \Theta_X \rrbracket, \boldsymbol{\delta}(\Gamma_i) \vdash Q'_i \ \text{(by Proposition E.20)} \\ \text{where } \begin{cases} Q'_i = z_{\{s,\mathtt{p},\mathtt{q}\}}(y).\mathbf{case}\, y \,\mathbf{of} \left\{ l_j(z_j) \triangleright \mathbf{with}\,(x_j,z) = z_j \,\mathbf{do}\,\mathbf{let}\, \llbracket s[\mathtt{p}] \rrbracket = \maltese_j \mathbf{in} \left(\llbracket Q''_{ij} \rrbracket_{\Theta_X \cdot \Gamma_i^{\diamond}\{S'_j/s[\mathtt{p}]\}, x_j:U'_j} \right) \boldsymbol{\sigma}(\Gamma_i \setminus s[\mathtt{p}]) \right\}_{j\in I} \\ \qquad \text{where } \ \forall j \in I : \maltese_j = \begin{cases} [\mathtt{q}: z,\ \mathtt{r}: z_{\{s,\mathtt{p},\mathtt{r}\}}]_{\mathtt{r}\in S'_j \setminus \mathtt{q}} & \text{if } \mathtt{q} \in S'_j \\ \left[\mathtt{r}: z_{\{s,\mathtt{p},\mathtt{r}\}}\right]_{\mathtt{r}\in S'_j} & \text{otherwise} \end{cases} \\ \text{or} \\ Q'_i = (\boldsymbol{\nu} z)\overline{z_{\{s',\mathtt{q},\mathtt{p}\}}}\langle l_i(\llbracket v \rrbracket, z)\rangle.\mathbf{let}\, \llbracket s'[\mathtt{q}] \rrbracket = \maltese \,\mathbf{in} \left(\llbracket Q''_i \rrbracket_{\Theta_X \cdot \Gamma_i^{\diamond}\{S''/s'[\mathtt{q}]\}\setminus v} \right) \boldsymbol{\sigma}(\Gamma_i \setminus s'[\mathtt{q}]) \\ \qquad \text{where } \ \maltese = \begin{cases} [\mathtt{p}: z,\ \mathtt{r}: z_{\{s',\mathtt{q},\mathtt{r}\}}]_{\mathtt{r}\in S''\setminus \mathtt{p}} & \text{if } \mathtt{p} \in S'' \\ \left[\mathtt{r}: z_{\{s',\mathtt{q},\mathtt{r}\}}\right]_{\mathtt{r}\in S''} & \text{otherwise} \end{cases} \end{cases} \end{cases} \tag{85}$$

We can now proceed by cases on α in the annotated reduction $P_0 \xrightarrow{\alpha} P^*$ (according to Def. E.5):

- $\alpha \in \{\textbf{case}, \textbf{let}\}$. These cases are impossible, and the statement hold vacuously. In fact, these reductions can only be fired by an occurrence of **case** or **let**, and we can verify that such prefixes do *not* appear at top-level in $Q'_1, \ldots, Q'_m$, nor $[\![Q_Y]\!]_{\Theta_X \cdot \Gamma_Y} \boldsymbol{\sigma}(\Gamma_Y)$, nor P_X. Hence, for all P_0 such that $[\![P]\!]_{\Theta \cdot \Gamma} \boldsymbol{\sigma}(\Gamma) \xrightarrow{\textbf{with}}{}^* P_0$, we have $P_0 \not\xrightarrow{\textbf{case}}$ and $P_0 \not\xrightarrow{\textbf{let}}$;
- $\alpha = \textbf{with}$. We have $P_0 \xrightarrow{\textbf{with}} P_*$, and thus $[\![P]\!]_{\Theta \cdot \Gamma} \boldsymbol{\sigma}(\Gamma) \xrightarrow{\textbf{with}}{}^* P_*$: we conclude by letting $\tilde{x} = \emptyset$, $P'' = P_*$, $\Gamma = \Gamma'$, $P = P'$;
- $\alpha = x$ (for some x). Since by Proposition E.20 we have $[\![\Theta]\!], \boldsymbol{\delta}(\Gamma) \vdash P_0$, we also know that $x \in \operatorname{dom}([\![\Theta]\!], \boldsymbol{\delta}(\Gamma))$ (by Proposition E.15). By Proposition E.21 we have two sub-cases:
  - $x{:}\sharp(T) \in [\![\Theta]\!]$. This case is absurd. In fact, by (76) it would imply $x \neq [\![X_j]\!]$ (for all $j \in 1..n$); moreover, it would require an unrestricted output on x, which implies $x = [\![Y_i]\!]$ for some $i \in 1..m'$ (by (84), (85), and (74)). Finally, it would imply a synchronisation with a process guarded by an unrestricted input on $[\![Y_i]\!]$ occurring in P_0 — which contradicts (84);
  - $x{:}\mathsf{L}\sharp(T) \in \boldsymbol{\delta}(\Gamma)$. We have $x{:}\mathsf{Li}(T) \uplus \mathsf{Lo}(T) \in \boldsymbol{\delta}(\Gamma)$, and it implies that two processes Q'_i, Q'_j (from (84) and (85)) synchronise on some x, performing respectively an input and an output. Without loss of generality, let $i = 1$ and $j = 2$. By Def. 5.6, it means that $\boldsymbol{\sigma}(\Gamma)$ replaces some $z_{s[\mathrm{p}]}$ and $z_{s'[\mathrm{q}]}$ in $[\![P]\!]_{\Theta \cdot \Gamma}$ with labelled tuples of channels v_1, v_2 such that $x = v_1(\mathrm{q}) = v_2(\mathrm{p}) = z_{\{s,\mathrm{p},\mathrm{q}\}}$, which implies $s = s'$. Correspondingly, by Def. 5.6, $\boldsymbol{\delta}(\Gamma)$ combines (using $⩀$) the the encodings of two unfolded partial projections, that after being split with $\uplus$, yield $\mathsf{Li}(T)$ and $\mathsf{Lo}(T)$ above. More precisely, considering that (by (76)) Γ is split into $\Gamma_1 \circ \Gamma_2 \circ \ldots \circ \Gamma_m \circ \ldots$, we have:

$$x = \boldsymbol{\sigma}(\Gamma)(s[\mathrm{p}])(\mathrm{q}) = \boldsymbol{\sigma}(\Gamma_1)(s[\mathrm{p}])(\mathrm{q}) = \boldsymbol{\sigma}(\Gamma)(s[\mathrm{q}])(\mathrm{p}) = \boldsymbol{\sigma}(\Gamma_2)(s[\mathrm{q}])(\mathrm{p}) = z_{\{s,\mathrm{p},\mathrm{q}\}} \qquad \text{(by Def. 5.6)}$$

$$\Gamma \;\text{ is consistent} \qquad \text{(by hypothesis)} \tag{86}$$

$$\Gamma_1 \circ \Gamma_2 \;\text{ is consistent} \qquad \text{(by (86), (76) and Cor. B.17)} \tag{87}$$

$$\text{for some } s, \mathrm{p}, \mathrm{q}, S_{\mathrm{p}}, S_{\mathrm{q}}\text{:} \;\; \Gamma(s[\mathrm{p}]) = \Gamma_1(s[\mathrm{p}]) \;\text{ and }\; \Gamma(s[\mathrm{q}]) = \Gamma_2(s[\mathrm{q}])$$

$$\overline{\Gamma_1(s[\mathrm{p}]) \upharpoonright \mathrm{q}} \leqslant_{\mathsf{P}} \Gamma_2(s[\mathrm{q}]) \upharpoonright \mathrm{p} \qquad \text{(by (86) and (87))} \tag{88}$$

$$\operatorname{unf}\left(\overline{\Gamma_1(s[\mathrm{p}]) \upharpoonright \mathrm{q}}\right) \leqslant_{\mathsf{P}} \operatorname{unf}(\Gamma_2(s[\mathrm{q}]) \upharpoonright \mathrm{p}) \qquad \text{(by (88) and Proposition B.8)} \tag{89}$$

  Now, the fact that Q'_1 performs an input on $z_{\{s,\mathrm{p},\mathrm{q}\}}$ implies that Q_1 performs a branching on $s[\mathrm{p}][\mathrm{q}]$, which means (by inverting (T-BRCH)) that Q_1 is typed by some S_{p} that is a branching from q, and $\Gamma_1(s[\mathrm{p}]) \leqslant_{\mathsf{S}} S_{\mathrm{p}}$. Symmetrically, the fact that Q'_2 performs an output on $z_{\{s,\mathrm{p},\mathrm{q}\}}$ implies that Q_2 performs a selection on $s[\mathrm{q}][\mathrm{p}]$, which means (by inverting (T-SEL)) that Q_2 is typed by some S_{q} that is a selection towards p, and $\Gamma_2(s[\mathrm{q}]) \leqslant_{\mathsf{S}} S_{\mathrm{q}}$. From (89), and the observation that S_{p},S_{q} are respectively a branching and selection type, we have (for some I^*):

$$\operatorname{unf}(\Gamma_1(s[\mathrm{p}])) = \mathrm{q} \,\&_{j \in I^*}\, ?l_j(U^*_j).S^*_j \leqslant_{\mathsf{S}} S_{\mathrm{p}} \quad \text{and} \quad \operatorname{unf}(\Gamma_2(s[\mathrm{q}])) = \mathrm{p} \oplus_{j \in I^*} !l_j(U^*_j).S^{**}_j \leqslant_{\mathsf{S}} S_{\mathrm{q}} \quad \text{and} \quad \forall j \in I^* : S^*_j \upharpoonright \mathrm{q} = \overline{S^{**}_j \upharpoonright \mathrm{p}} \tag{90}$$

  Now, we can notice that $S_{\mathrm{p}}, S_{\mathrm{q}}, \Gamma_1, \Gamma_2, \Gamma^{\diamond}_1, \Gamma^{\diamond}_2, Q_1, Q_2$ match the corresponding definitions in Equation (42) on page 44 (in the proof of Theorem 2.16 — subject reduction for multiparty session typed processes). In the rest of the present proof, we will thus refer to results that follow Equation (42) (in particular, (43), (44) and (46)) applying them to (85), to study the synchronisation on $x = z_{\{s,\mathrm{p},\mathrm{q}\}}$ between Q'_1 and Q'_2.
  Let $\boldsymbol{\sigma}(v{:}U) = \emptyset$ (i.e., the empty substitution) if $U = B$ (otherwise, Def. 5.6 applies). We can see that the synchronisation on x gives the following reductions, where the linear names given by the reified instantiation of v is passed from Q'_2 to Q'_1 (we exploit the fact that $U'' \leqslant_{\mathsf{S}} U'_k$

by (46), and thus $\boldsymbol{\sigma}(v{:}U'_k) = \boldsymbol{\sigma}(v{:}U'')$ by Proposition E.14):

$$\llbracket\Theta_X\rrbracket, \boldsymbol{\delta}(\Gamma_1) \uplus \boldsymbol{\delta}(\Gamma_2) \vdash Q'_1 \mid Q'_2 \xrightarrow{x}$$

$$(\boldsymbol{\nu}z)\left(\begin{array}{l}\textbf{case}\, l_k(\llbracket v\rrbracket, z)\,\textbf{of}\,\left\{l_j(z_j) \triangleright \textbf{with}\,(x_j,z){=}z_j\,\textbf{do}\,\textbf{let}\,\llbracket s[\mathtt{p}]\rrbracket{=}\maltese_j\,\textbf{in}\left(\llbracket Q''_{1j}\rrbracket_{\Theta_X\cdot\Gamma_1^{\diamond\prime},s[\mathtt{p}]:S'_j,x_j:U'_j}\right)\boldsymbol{\sigma}(\Gamma_1\setminus s[\mathtt{p}])\right\}_{j\in I}\boldsymbol{\sigma}(v{:}U'_k)\\ \mid\ \textbf{let}\,\llbracket s[\mathtt{q}]\rrbracket{=}\maltese\,\textbf{in}\left(\llbracket Q''_2\rrbracket_{\Theta_X\cdot\Gamma_2^{\diamond\prime},s[\mathtt{q}]:S''}\right)\left(\boldsymbol{\sigma}(\Gamma_2\setminus s[\mathtt{q}])\setminus \mathrm{dom}\left(\boldsymbol{\sigma}(v{:}U'')\right)\right)\end{array}\right)\xrightarrow{\textbf{case}}\xrightarrow{\textbf{with}}$$

$$(\boldsymbol{\nu}z)\left(\begin{array}{l}\textbf{let}\,\llbracket s[\mathtt{p}]\rrbracket{=}\maltese_k\,\textbf{in}\left(\llbracket Q''_{1k}\rrbracket_{\Theta_X\cdot\Gamma_1^{\diamond\prime},s[\mathtt{p}]:S'_k,x_k:U'_k}\right)\boldsymbol{\sigma}(\Gamma_1\setminus s[\mathtt{p}])\boldsymbol{\sigma}(v{:}U'_k)\{\llbracket v\rrbracket/x_k\}\\ \mid\ \textbf{let}\,\llbracket s[\mathtt{q}]\rrbracket{=}\maltese\,\textbf{in}\left(\llbracket Q''_2\rrbracket_{\Theta_X\cdot\Gamma_2^{\diamond\prime},s[\mathtt{q}]:S''}\right)\boldsymbol{\sigma}(\Gamma_2\setminus s[\mathtt{q}]\setminus v)\end{array}\right)\xrightarrow{\textbf{let}}\xrightarrow{\textbf{let}}$$

$$(\boldsymbol{\nu}z)\left(\begin{array}{l}\left(\llbracket Q''_{1k}\rrbracket_{\Theta_X\cdot\Gamma_1^{\diamond\prime},s[\mathtt{p}]:S'_k,x_k:U'_k}\{\llbracket v\rrbracket/x_k\}\right)\boldsymbol{\sigma}(\Gamma_1\setminus s[\mathtt{p}])\boldsymbol{\sigma}(v{:}U'_k)\{\maltese_k/\llbracket s[\mathtt{p}]\rrbracket\}\\ \mid\ \left(\llbracket Q''_2\rrbracket_{\Theta_X\cdot\Gamma_2^{\diamond\prime},s[\mathtt{q}]:S''}\right)\boldsymbol{\sigma}(\Gamma_2\setminus s[\mathtt{q}]\setminus v)\{\maltese/\llbracket s[\mathtt{q}]\rrbracket\}\end{array}\right) \tag{91}$$

Now, notice that, from Equation (85), we have:

$$\maltese_k = \begin{cases}[\mathtt{q}{:}\,z,\ \mathtt{r}{:}\,z_{\{s,\mathtt{p},\mathtt{r}\}}]_{\mathtt{r}\in S'_k\setminus\mathtt{q}} & \text{if } \mathtt{q}\in S'_k\\ \left[\mathtt{r}{:}\,z_{\{s,\mathtt{p},\mathtt{r}\}}\right]_{\mathtt{r}\in S'_k} & \text{otherwise}\end{cases} \qquad \maltese = \begin{cases}[\mathtt{p}{:}\,z,\ \mathtt{r}{:}\,z_{\{s,\mathtt{q},\mathtt{r}\}}]_{\mathtt{r}\in S''\setminus\mathtt{p}} & \text{if } \mathtt{p}\in S''\\ \left[\mathtt{q}{:}\,z_{\{s,\mathtt{q},\mathtt{r}\}}\right]_{\mathtt{q}\in S''} & \text{otherwise}\end{cases} \tag{92}$$

If we replace z with $z_{\{s,\mathtt{p},\mathtt{q}\}}$ in (92), from (45), (47) and Proposition E.14 we observe that:

$$\boldsymbol{\sigma}(\Gamma_1\setminus s[\mathtt{p}])\boldsymbol{\sigma}\left(v{:}U'_k\right)\left\{\maltese_k\left\{z_{\{s,\mathtt{p},\mathtt{q}\}}/z\right\}/\llbracket s[\mathtt{p}]\rrbracket\right\} = \boldsymbol{\sigma}\left(\Gamma_1\left\{S'_k/s[\mathtt{p}]\right\}\right)\boldsymbol{\sigma}\left(v{:}U'_k\right) = \boldsymbol{\sigma}\left(\Gamma_1^{\diamond\prime}\circ\Gamma_v, s[\mathtt{p}]{:}S'_k\right) = \boldsymbol{\sigma}\left(\Gamma'_1\right) \tag{93}$$

$$\boldsymbol{\sigma}(\Gamma_2\setminus s[\mathtt{q}]\setminus v)\left\{\maltese\left\{z_{\{s,\mathtt{p},\mathtt{q}\}}/z\right\}/\llbracket s[\mathtt{q}]\rrbracket\right\} = \boldsymbol{\sigma}\left(\Gamma_2\left\{S''/s[\mathtt{q}]\right\}\setminus v\right) = \boldsymbol{\sigma}\left(\Gamma_2^{\diamond\prime}, s[\mathtt{q}]{:}S''\right) = \boldsymbol{\sigma}\left(\Gamma'_2\right) \tag{94}$$

Moreover, by applying the substitutions in the processes in (91), by (47) and (55) we get:

$$\llbracket Q''_{1k}\rrbracket_{\Theta_X\cdot\Gamma_1^{\diamond\prime},s[\mathtt{p}]:S'_k,x_k:U'_k}\{\llbracket v\rrbracket/x_k\} = \llbracket Q''_{1k}\{v/x_k\}\rrbracket_{\Theta_X\cdot\Gamma'_1} \qquad \llbracket Q''_2\rrbracket_{\Theta_X\cdot\Gamma_2^{\diamond\prime},s[\mathtt{q}]:S''} = \llbracket Q''_2\rrbracket_{\Theta_X\cdot\Gamma'_2} \tag{95}$$

Now, by α-renaming $(\boldsymbol{\nu}z)$ into $(\boldsymbol{\nu}z_{\{s,\mathtt{p},\mathtt{q}\}})$ in (91), and applying (93), (94) and (95), we get:

$$\llbracket\Theta_X\rrbracket, \boldsymbol{\delta}(\Gamma_1)\uplus\boldsymbol{\delta}(\Gamma_2)\vdash Q'_1\mid Q'_2 \ \to^*\ (\boldsymbol{\nu}z_{\{s,\mathtt{p},\mathtt{q}\}})\left(\llbracket Q''_{1k}\{v/x_k\}\rrbracket_{\Theta_X\cdot\Gamma'_1}\boldsymbol{\sigma}\left(\Gamma'_1\right)\ \mid\ \llbracket Q''_2\rrbracket_{\Theta_X\cdot\Gamma'_2}\boldsymbol{\sigma}\left(\Gamma'_2\right)\right) \tag{96}$$

$$= (\boldsymbol{\nu}z_{\{s,\mathtt{p},\mathtt{q}\}})\left(\llbracket Q''_{1k}\{v/x_k\}\ \mid\ Q''_2\rrbracket_{\Theta_X\cdot\Gamma'_1\circ\Gamma'_2}\right) \tag{97}$$

Now, from the reductions in (91), and by (96) and (84), we get:

$$P_0 \ \to^*$$

$$(\boldsymbol{\nu}\llbracket X_1\rrbracket)\ldots(\boldsymbol{\nu}\llbracket X_n\rrbracket)\llbracket(\boldsymbol{\nu}\widetilde{s_*})\rrbracket\left((\boldsymbol{\nu}z_{\{s,\mathtt{p},\mathtt{q}\}})\left(\llbracket Q''_{1k}\{v/x_k\}\rrbracket_{\Theta_X\cdot\Gamma'_1}\boldsymbol{\sigma}\left(\Gamma'_1\right)\ \mid\ \llbracket Q''_2\rrbracket_{\Theta_X\cdot\Gamma'_2}\boldsymbol{\sigma}\left(\Gamma'_2\right)\right)\mid Q'_3\mid\ldots\mid Q'_m\ \mid\ \llbracket Q_Y\rrbracket_{\Theta_X\cdot\Gamma_Y}\boldsymbol{\sigma}(\Gamma_Y)\ \mid\ P_X\right) \tag{98}$$

Hence, if we let:

$$P'' = (\boldsymbol{\nu}\llbracket X_1\rrbracket)\ldots(\boldsymbol{\nu}\llbracket X_n\rrbracket)\llbracket(\boldsymbol{\nu}\widetilde{s_*})\rrbracket\left(\llbracket Q''_{1k}\{v/x_k\}\rrbracket_{\Theta_X\cdot\Gamma'_1}\boldsymbol{\sigma}\left(\Gamma'_1\right)\ \mid\ \llbracket Q''_2\rrbracket_{\Theta_X\cdot\Gamma'_2}\boldsymbol{\sigma}\left(\Gamma'_2\right)\mid Q'_3\mid\ldots\mid Q'_m\ \mid\ \llbracket Q_Y\rrbracket_{\Theta_X\cdot\Gamma_Y}\boldsymbol{\sigma}(\Gamma_Y)\ \mid\ P_X\right) \tag{99}$$

we obtain:

$$P_0 \ \to^*\ (\boldsymbol{\nu}z_{\{s,\mathtt{p},\mathtt{q}\}})P'' \qquad \text{(by (99) and (98))}$$

Furthermore, notice that:

$$\Gamma_1 \circ \Gamma_2 \vdash Q_1 \mid Q_2 \quad \to \quad \Gamma_1' \circ \Gamma_2' \vdash Q_{1k}''\{v/x_k\} \mid Q_2'' \qquad \text{(from (97))} \tag{100}$$

$$\Theta \cdot \Gamma \vdash P \quad \to \quad \Theta \cdot \Gamma_1' \circ \Gamma_2' \circ \Gamma_3 \circ \ldots \circ \Gamma_m \vdash \mathbf{def}\ \widetilde{D}\ \mathbf{in}\ (\boldsymbol{\nu}\widetilde{s_*})\big(Q_{1k}''\{v/x_k\} \mid Q_2'' \mid Q_3 \mid \ldots \mid Q_Y\big) \quad \text{(from (73), (76) and (100))} \tag{101}$$

Hence, if we let:

$$\Gamma' \;=\; \Gamma_1' \circ \Gamma_2' \circ \Gamma_3 \circ \ldots \circ \Gamma_m \quad \text{and} \quad P' \;=\; \mathbf{def}\ \widetilde{D}\ \mathbf{in}\ (\boldsymbol{\nu}\widetilde{s_*})\big(Q_{1k}''\{v/x_k\} \mid Q_2'' \mid Q_3 \mid \ldots \mid Q_Y\big) \tag{102}$$

we obtain:

$$(\mathrm{T}\pi\text{-}\mathrm{Reify})\ \frac{[\![\Theta \cdot \Gamma' \vdash P']\!]}{[\![\Theta]\!], \boldsymbol{\delta}\big(\Gamma'\big) \vdash [\![P']\!]\boldsymbol{\sigma}\big(\Gamma'\big)} \xrightarrow{\text{with}}{}^{*} \quad P'' \qquad \text{(by (84), (85) and (99))}$$

Summing up, we prove the statement by taking:

- $\widetilde{x} \;=\; z_{\{s,\mathrm{p},\mathrm{q}\}}$;
- P'' as defined in (99);
- Γ' as defined in (102);
- P' as defined in (102);

- $\alpha = \tau$. By Def. E.5, this reduction can only be induced by a synchronisation on some delimited name x. By (75), we have two cases:
  - x is delimited in $[\![(\boldsymbol{\nu}\widetilde{s_*})]\!]$. Therefore, there is some $s \in \widetilde{s_*}$ whose encoding yields a delimitation for x. In this case, after opening $(\boldsymbol{\nu}s)$ by inverting (T-Res), we fall back into a case similar to that for $\alpha = x$ and $x{:}\mathsf{L}\sharp(T) \in \boldsymbol{\delta}(\Gamma)$ above: we get an encoded π-calculus typing derivation where x is linearly-typed, and corresponds to a reduction between some $s[\mathrm{p}][\mathrm{q}]/s[\mathrm{q}][\mathrm{p}]$. We prove this case as above, and conclude by re-applying the delimitation $(\boldsymbol{\nu}s)$, and taking:
    - $\widetilde{x} = z_{\{s,\mathrm{p},\mathrm{q}\}}$;
    - P'' as in (99);
    - $\Gamma' = \Gamma$ (since the reduction occurs on a delimited session s);
    - P' as in (102);
  - x is delimited in $(\boldsymbol{\nu}[\![X_1]\!]) \ldots (\boldsymbol{\nu}[\![X_n]\!])$, yielded by the encodings of $\mathbf{def}\ X_1(\widetilde{x_1}) = P_{X_1}\ \mathbf{in} \ldots \mathbf{def}\ X_n(\widetilde{x_1}) = P_{X_n}\ \mathbf{in} \ldots$ in (73), and thus by the encoding of (T-Def) in Fig. 13. Without loss of generality, let $x = [\![X_1]\!]$ (otherwise, the proof is similar). If we open the delimitation $(\boldsymbol{\nu}[\![X_1]\!])$ by looking at the premise of (Tπ-Res1) in the encoded derivation, we can see that $x = [\![X_1]\!]$ has an unrestricted type $\sharp\Big([\![\widetilde{U_1}]\!]\Big)$, and is used for *input* by a replicated process in P_X, as shown in (78); therefore, the τ-reduction under analysis is induced by a synchronisation on x with another process that uses $x{:}\sharp\Big([\![\widetilde{U_1}]\!]\Big)$ for *output*. By examining Fig. 13, we can see that such a process can only be produced by the encoding of (T-Call): this implies that Q_Y in (73) and (74) contains a process $X_1\langle\widetilde{v_1}\rangle$, whose arguments are typed as $\widetilde{U_1}$. Without loss of generality, let $Y_1 = X_1$ in (74), which implies:

$$\Gamma_{Y_1} = \Gamma_{X_1} \vdash \widetilde{v_1} : \widetilde{U_1} \qquad \text{(by (75), (77) and inversion of (T-Call))} \tag{103}$$

Applying these findings in (84), we obtain:

$$[\![P]\!]_{\Theta\cdot\Gamma}\boldsymbol{\sigma}(\Gamma) \xrightarrow{\text{with}}{}^{*} \tag{104}$$

$$P_0 \;\equiv\; (\boldsymbol{\nu}[\![X_1]\!]) \ldots (\boldsymbol{\nu}[\![X_n]\!])[\![(\boldsymbol{\nu}\widetilde{s_*})]\!]\ \big(\ Q_1' \mid \ldots \mid Q_m' \ \mid\ [\![X_1\langle\widetilde{v_1}\rangle]\!]_{\Theta_X\cdot\Gamma_{X_1}}\boldsymbol{\sigma}(\Gamma_{X_1}) \ \mid\ [\![Q_{Y'}]\!]_{\Theta_X\cdot\Gamma_{Y'}}\boldsymbol{\sigma}(\Gamma_{Y'}) \ \mid\ P_X\ \big) \tag{105}$$

where $Q_{Y'} = Y_2\langle\widetilde{v_2}\rangle \mid \ldots \mid Y_{m'}\langle\widetilde{v_{m'}}\rangle$ and $\Gamma_{Y'} = \Gamma_{Y_2} \circ \ldots \circ \Gamma_{Y_m'}$

Now, letting P_0 synchronise, we get:

$$P_0 \xrightarrow{\tau} (\boldsymbol{\nu}[\![X_1]\!])\dots(\boldsymbol{\nu}[\![X_n]\!])[\![(\boldsymbol{\nu}\widetilde{s_*})]\!]\left(Q'_1 \mid \dots \mid Q'_m \mid \left(\textbf{with}\,(\widetilde{x_1})=\widetilde{[\![v_1]\!]}\,\textbf{do}\,[\![P_{X_1}]\!]_{\Theta,X_1:\widetilde{U_1}\cdot\widetilde{x_1}:\widetilde{U_1}}\right)\boldsymbol{\sigma}(\Gamma_{X_1}) \mid [\![Q_{Y'}]\!]_{\Theta_X\cdot\Gamma_{Y'}}\boldsymbol{\sigma}(\Gamma_{Y'}) \mid P_X\right) \xrightarrow{\textbf{with}}$$

$$(\boldsymbol{\nu}[\![X_1]\!])\dots(\boldsymbol{\nu}[\![X_n]\!])[\![(\boldsymbol{\nu}\widetilde{s_*})]\!]\left(Q'_1 \mid \dots \mid Q'_m \mid \left([\![P_{X_1}]\!]_{\Theta,X_1:\widetilde{U_1}\cdot\widetilde{x_1}:\widetilde{U_1}}\right)\boldsymbol{\sigma}(\Gamma_{X_1})\left\{{}^{\widetilde{[\![v_1]\!]}}/_{\widetilde{x_1}}\right\} \mid [\![Q_{Y'}]\!]_{\Theta_X\cdot\Gamma_{Y'}}\boldsymbol{\sigma}(\Gamma_{Y'}) \mid P_X\right) =$$

$$(\boldsymbol{\nu}[\![X_1]\!])\dots(\boldsymbol{\nu}[\![X_n]\!])[\![(\boldsymbol{\nu}\widetilde{s_*})]\!]\left(Q'_1 \mid \dots \mid Q'_m \mid \left(\left[\!\!\left[P_{X_1}\left\{{}^{\widetilde{v_1}}/_{\widetilde{x_1}}\right\}\right]\!\!\right]_{\Theta,X_1:\widetilde{U_1}\cdot\Gamma_{X_1}}\right)\boldsymbol{\sigma}(\Gamma_{X_1}) \mid [\![Q_{Y'}]\!]_{\Theta_X\cdot\Gamma_{Y'}}\boldsymbol{\sigma}(\Gamma_{Y'}) \mid P_X\right) \quad \text{(by (103) + Lemma B.26)}$$

$$= P''$$

Now, from (73), and from the proof of Theorem 2.16 (case (R-Call)) we have:

$$\Theta\cdot\Gamma\vdash P \;\rightarrow\; \Theta\cdot\Gamma'\vdash P' = \textbf{def}\,\widetilde{D}\,\textbf{in}\,(\boldsymbol{\nu}\widetilde{s_*})\left(P_{X_1}\left\{{}^{\widetilde{x_1}}/_{\widetilde{v_1}}\right\} \mid \dots \mid Q_m \mid Q_{Y'}\right) \qquad \text{with } \Gamma'=\Gamma$$

$$(\text{T}\pi\text{-Reify})\;\frac{[\![\Theta\cdot\Gamma'\vdash P']\!]}{[\![\Theta]\!],\boldsymbol{\delta}(\Gamma')\vdash [\![P']\!]\boldsymbol{\sigma}(\Gamma')} \;\xrightarrow{\textbf{with}}{}^{*}\; P'' \tag{106}$$

Hence, we conclude by taking:

- $\widetilde{x}=\emptyset$;
- P'' as above;
- $\Gamma'=\Gamma$
- P' as in (106).

◀

▶ **Lemma E.24** (Operational soundness of encoding). *If* $\Theta\cdot\Gamma\vdash P$*:*

$$(\text{T}\pi\text{-Reify})\;\frac{[\![\Theta\cdot\Gamma\vdash P]\!]}{[\![\Theta]\!],\boldsymbol{\delta}(\Gamma)\vdash [\![P]\!]\boldsymbol{\sigma}(\Gamma)} \rightarrow^* P^* \quad \textit{implies that } \exists\widetilde{x},P'',\Gamma',P' \textit{ such that } P_* \rightarrow^* (\boldsymbol{\nu}\widetilde{x})P'' \textit{ and } \quad (\text{T}\pi\text{-Reify})\;\frac{[\![\Theta\cdot\Gamma'\vdash P']\!]}{[\![\Theta]\!],\boldsymbol{\delta}(\Gamma')\vdash [\![P']\!]\boldsymbol{\sigma}(\Gamma')} \xrightarrow{\textbf{with}}{}^{*} P''$$

and $\Theta\cdot\Gamma\vdash P \;\rightarrow^*\; P'$.

Proof. Let m be the length of the sequence of reductions $[\![\Theta]\!],\boldsymbol{\delta}(\Gamma)\vdash[\![P]\!]\boldsymbol{\sigma}(\Gamma) \rightarrow^* P_*$. We proceed by induction on m:

- base case $m=0$. We have $\Gamma_*=[\![\Theta]\!],\boldsymbol{\delta}(\Gamma)$ and $P_*=[\![P]\!]\boldsymbol{\sigma}(\Gamma)$. We conclude by letting $\tilde{x}=\emptyset$, $P''=P_*$, $\Gamma'=\Gamma$, and $P'=P$;
- inductive case $m=m'+1$. Take P'_* such that:

$$[\![\Theta]\!],\boldsymbol{\delta}(\Gamma)\vdash[\![P]\!]\boldsymbol{\sigma}(\Gamma)\;\overbrace{\rightarrow\cdots\rightarrow}^{m'\text{ times}}\;P'_*\;\rightarrow\;P_* \tag{107}$$

By the induction hypothesis, $\exists\widetilde{x_\diamond},P''_\diamond,\Gamma_\diamond,P_\diamond,n_\diamond$ such that:

$$\Theta\cdot\Gamma\vdash P\;\rightarrow^*\;\Theta\cdot\Gamma_\diamond\vdash P_\diamond \quad\text{and}\quad P'_*\rightarrow^*(\boldsymbol{\nu}\widetilde{x_\diamond})P''_\diamond \quad\text{and}\quad (\text{T}\pi\text{-Reify})\;\frac{[\![\Theta\cdot\Gamma_\diamond\vdash P_\diamond]\!]}{[\![\Theta]\!],\boldsymbol{\delta}(\Gamma_\diamond)\vdash[\![P_\diamond]\!]\boldsymbol{\sigma}(\Gamma_\diamond)}\xrightarrow{\textbf{with}}{}^{*}P''_\diamond \tag{108}$$

By Proposition C.6, P'_* is quasi-linear. Therefore, from (107) and (108), by Cor. C.12, we have either:

- $P_*\rightarrow^* P''_\diamond$. In this case, we conclude by letting $\tilde{x}=\widetilde{x_\diamond}$, $P''=P''_\diamond$, $\Gamma'=\Gamma_\diamond$, and $P'=P_\diamond$;
- $\exists P''_*$ such that $P_*\rightarrow^* P''_*$, and $(\boldsymbol{\nu}\widetilde{x_\diamond})P''_\diamond\rightarrow P''_*$. In this case, the latter transition implies $P''_*\equiv(\boldsymbol{\nu}\widetilde{x_\diamond})P''_{**}$ (for some P''_{**}), and (by inverting rule (Rπ-Res) once per element of $\widetilde{x_\diamond}$) $P''_\diamond\rightarrow P''_{**}$. Therefore, by (108) and Lemma E.23, we know that $\exists\widetilde{x_{**}},P'',\Gamma',P'$ such that $\Theta\cdot\Gamma_\diamond\vdash P_\diamond\;\rightarrow^*\;\Theta\cdot\Gamma'\vdash P'$, and:

$$P''_{**}\rightarrow^*(\boldsymbol{\nu}\widetilde{x_{**}})P'' \quad\text{and}\quad (\text{T}\pi\text{-Reify})\;\frac{[\![\Theta\cdot\Gamma'\vdash P']\!]}{[\![\Theta]\!],\boldsymbol{\delta}(\Gamma')\vdash[\![P']\!]\boldsymbol{\sigma}(\Gamma')}\xrightarrow{\textbf{with}}{}^{*}P''$$

Therefore, by letting $\tilde{x} = \widetilde{x_\diamond}\,\widetilde{x_{**}}$, we conclude $\Theta \cdot \Gamma \vdash P \to^* \Theta \cdot \Gamma' \vdash P'$, and:

$$P_* \to^* (\boldsymbol{\nu}\tilde{x})P'' \qquad \text{and} \qquad (\text{T}\pi\text{-Reify})\,\frac{[\![\Theta \cdot \Gamma' \vdash P']\!]}{[\![\Theta]\!], \boldsymbol{\delta}(\Gamma') \vdash [\![P']\!]\boldsymbol{\sigma}(\Gamma')} \xrightarrow{\text{with}}{}^* P''$$

◀

▶ **Lemma E.25.** $\Theta \cdot \Gamma \vdash P \to P'$ *implies that* $\exists \Gamma', \widetilde{x}, P''$ *such that:*

$$(\text{T}\pi\text{-Reify})\,\frac{[\![\Theta \cdot \Gamma \vdash P]\!]}{[\![\Theta]\!], \boldsymbol{\delta}(\Gamma) \vdash [\![P]\!]\boldsymbol{\sigma}(\Gamma)} \to^* (\boldsymbol{\nu}\widetilde{x})P'' \qquad \textit{and} \qquad (\text{T}\pi\text{-Reify})\,\frac{[\![\Theta \cdot \Gamma' \vdash P']\!]}{[\![\Theta]\!], \boldsymbol{\delta}(\Gamma') \vdash [\![P']\!]\boldsymbol{\sigma}(\Gamma')} = P''$$

Proof. By induction on the derivation of $P \to P'$. In the base cases (R-Comm) and (R-Call), the shape of Γ' can be determined from the proof of Theorem 2.16 (page 44), and correspondingly, $\widetilde{x}$ and P'' can be determined from a simplification of the proof of Lemma E.23, by simplifying (73) so that it only contains the processes under study (without other $\xrightarrow{\text{with}}{}^*$-reducing processes). The inductive cases follow by the induction hypothesis. ◀

▶ **Lemma E.26** (Operational completeness of encoding)**.** $\Theta \cdot \Gamma \vdash P \to^* P'$ *implies that* $\exists \Gamma', \widetilde{x}, P''$ *such that:*

$$(\text{T}\pi\text{-Reify})\,\frac{[\![\Theta \cdot \Gamma \vdash P]\!]}{[\![\Theta]\!], \boldsymbol{\delta}(\Gamma) \vdash [\![P]\!]\boldsymbol{\sigma}(\Gamma)} \to^* (\boldsymbol{\nu}\widetilde{x})P'' \qquad \textit{and} \qquad (\text{T}\pi\text{-Reify})\,\frac{[\![\Theta \cdot \Gamma' \vdash P']\!]}{[\![\Theta]\!], \boldsymbol{\delta}(\Gamma') \vdash [\![P']\!]\boldsymbol{\sigma}(\Gamma')} = P''$$

Proof. Let m be the length of the sequence of reductions in $\Theta \cdot \Gamma \vdash P \to^* P'$. We prove a slightly stronger statement with the additional clause: "$\exists n$ such that $(\boldsymbol{\nu}\tilde{x}) = (\boldsymbol{\nu}x_1)\dots(\boldsymbol{\nu}x_n)$". We proceed by induction on m:

- base case $m = 0$. We trivially conclude by letting $\Gamma' = \Gamma$, $P' = P$, and $n = 0$;
- inductive case $m = m' + 1$. Take Γ_*, Θ_*, P_* such that:

$$\Theta \cdot \Gamma \vdash P \overbrace{\to \cdots \to}^{m' \text{ times}} \Theta \cdot \Gamma_* \vdash P_* \to \Theta \cdot \Gamma' \vdash P'$$

By the induction hypothesis, for some n_* we have:

$$(\text{T}\pi\text{-Reify})\,\frac{[\![\Theta \cdot \Gamma \vdash P]\!]}{[\![\Theta]\!], \boldsymbol{\delta}(\Gamma) \vdash [\![P]\!]\boldsymbol{\sigma}(\Gamma)} \to^* \frac{\dfrac{[\![\Theta \cdot \Gamma_* \vdash P_*]\!]}{[\![\Theta]\!], \boldsymbol{\delta}(\Gamma_*) \vdash [\![P_*]\!]\boldsymbol{\sigma}(\Gamma_*)}\,(\text{T}\pi\text{-Reify})}{\Gamma''_* \vdash (\boldsymbol{\nu}x'_1)\dots(\boldsymbol{\nu}x'_{n_*})[\![P_*]\!]\boldsymbol{\sigma}(\Gamma_*)}\,(\text{T}\pi\text{-Res}) \times n_* \tag{109}$$

Moreorer, from $\Theta \cdot \Gamma_* \vdash P_* \to \Theta \cdot \Gamma' \vdash P'$, by Lemma E.25 we get (for some n_{**}):

$$(\text{T}\pi\text{-Reify})\,\frac{[\![\Theta \cdot \Gamma_* \vdash P_*]\!]}{[\![\Theta]\!], \boldsymbol{\delta}(\Gamma_*) \vdash [\![P_*]\!]\boldsymbol{\sigma}(\Gamma_*)} \to^* \frac{\dfrac{[\![\Theta \cdot \Gamma' \vdash P']\!]}{[\![\Theta]\!], \boldsymbol{\delta}(\Gamma') \vdash [\![P']\!]\boldsymbol{\sigma}(\Gamma')}\,(\text{T}\pi\text{-Reify})}{\Gamma''' \vdash (\boldsymbol{\nu}x''_1)\dots(\boldsymbol{\nu}x''_{n_{**}})[\![P']\!]\boldsymbol{\sigma}(\Gamma')}\,(\text{T}\pi\text{-Res}) \times n_{**} \tag{110}$$

Note that each transition in (110) is preserved when fired inside the n_* delimitations taken from (109), via n_* applications of rule (Rπ-Res). Therefore, letting $n = n_* + n_{**}$, $x_i = x'_i$ $(\forall i \in 1..n_*)$ and $x_{j+n_*} = x''_j$ $(\forall j \in 1..n_{**})$, we conclude:

$$(\text{T}\pi\text{-Reify})\,\frac{[\![\Theta \cdot \Gamma \vdash P]\!]}{[\![\Theta]\!], \boldsymbol{\delta}(\Gamma) \vdash [\![P]\!]\boldsymbol{\sigma}(\Gamma)} \to^* \frac{\dfrac{[\![\Theta \cdot \Gamma' \vdash P']\!]}{[\![\Theta]\!], \boldsymbol{\delta}(\Gamma') \vdash [\![P']\!]\boldsymbol{\sigma}(\Gamma')}\,(\text{T}\pi\text{-Reify})}{\Gamma'' \vdash (\boldsymbol{\nu}x_1)\dots(\boldsymbol{\nu}x_n)[\![P']\!]\boldsymbol{\sigma}(\Gamma')}\,(\text{T}\pi\text{-Res}) \times n$$

◀

Items (1) and (2) of Theorem E.27 below use $\boldsymbol{\sigma}(\Gamma)$ to allow reductions of encoded open channels with roles (cf. Ex. 5.8). Note that when we write $[\![P']\!]_{\Theta\cdot\Gamma'}$, we imply $\Theta \cdot \Gamma' \vdash P'$ (cf. Def. 5.7).

▶ **Theorem E.27** (Open operational correspondence). *If* $\Theta \cdot \Gamma \vdash P$ *and* $\mathrm{fv}(P) = \emptyset$*:*

1. (Completeness) $P \to^* P'$ *implies* $\exists \Gamma', \widetilde{x}, P''$ *such that* $[\![P]\!]_{\Theta\cdot\Gamma}\boldsymbol{\sigma}(\Gamma) \to^* (\boldsymbol{\nu}\widetilde{x})P''$ *and* $P'' = [\![P']\!]_{\Theta\cdot\Gamma'}\boldsymbol{\sigma}(\Gamma')$*;*
2. (Soundness) $[\![P]\!]_{\Theta\cdot\Gamma}\boldsymbol{\sigma}(\Gamma) \to^* P_*$ *implies* $\exists \widetilde{x}, P'', \Gamma', P' : P_* \to^* (\boldsymbol{\nu}\widetilde{x})P''$, $P \to^* P'$ *and* $[\![P']\!]_{\Theta\cdot\Gamma'}\boldsymbol{\sigma}(\Gamma') \xrightarrow{\text{with}}{}^* P''$.

▶ **Theorem E.27** (Open operational correspondence). *If* $\Theta \cdot \Gamma \vdash P$ *and* $\mathrm{fv}(P) = \emptyset$*:*

1. (Completeness) $P \to^* P'$ *implies* $\exists \Gamma', \widetilde{x}, P''$ *such that* $[\![P]\!]_{\Theta\cdot\Gamma}\boldsymbol{\sigma}(\Gamma) \to^* (\boldsymbol{\nu}\widetilde{x})P''$ *and* $P'' = [\![P']\!]_{\Theta\cdot\Gamma'}\boldsymbol{\sigma}(\Gamma')$*;*
2. (Soundness) $[\![P]\!]_{\Theta\cdot\Gamma}\boldsymbol{\sigma}(\Gamma) \to^* P_*$ *implies* $\exists \widetilde{x}, P'', \Gamma', P' : P_* \to^* (\boldsymbol{\nu}\widetilde{x})P''$, $P \to^* P'$ *and* $[\![P']\!]_{\Theta\cdot\Gamma'}\boldsymbol{\sigma}(\Gamma') \xrightarrow{\text{with}}{}^* P''$.

Proof. We prove the following equivalent formulation of the statement:

1. *(Completeness)* $\Theta \cdot \Gamma \vdash P \to^* P'$ implies that $\exists \Gamma', \widetilde{x}, P''$ such that:

$$(\text{T}\pi\text{-Reify})\ \frac{[\![\Theta \cdot \Gamma \vdash P]\!]}{[\![\Theta]\!], \boldsymbol{\delta}(\Gamma) \vdash [\![P]\!]\boldsymbol{\sigma}(\Gamma)} \to^* (\boldsymbol{\nu}\widetilde{x})P'' \quad \text{and} \quad (\text{T}\pi\text{-Reify})\ \frac{[\![\Theta \cdot \Gamma' \vdash P']\!]}{[\![\Theta]\!], \boldsymbol{\delta}(\Gamma') \vdash [\![P']\!]\boldsymbol{\sigma}(\Gamma')} = P''$$

2. *(Soundness)* If $\Theta \cdot \Gamma \vdash P$:

$$(\text{T}\pi\text{-Reify})\ \frac{[\![\Theta \cdot \Gamma \vdash P]\!]}{[\![\Theta]\!], \boldsymbol{\delta}(\Gamma) \vdash [\![P]\!]\boldsymbol{\sigma}(\Gamma)} \to^* P^* \quad \begin{array}{l}\text{implies that } \exists \widetilde{x}, P'', \Gamma', P' \\ \text{such that } P_* \to^* (\boldsymbol{\nu}\widetilde{x})P'' \text{ and}\end{array} \quad (\text{T}\pi\text{-Reify})\ \frac{[\![\Theta \cdot \Gamma' \vdash P']\!]}{[\![\Theta]\!], \boldsymbol{\delta}(\Gamma') \vdash [\![P']\!]\boldsymbol{\sigma}(\Gamma')} \xrightarrow{\text{with}}{}^* P''$$

and $\Theta \cdot \Gamma \vdash P \to^* P'$.

Item 1 holds by Lemma E.26. Item 2 holds by Lemma E.24. ◀

▶ **Theorem 6.6** (Operational correspondence). *If* $\varnothing \cdot \varnothing \vdash P$*, then:*

1. (Completeness) $P \to^* P'$ *implies* $\exists \widetilde{x}, P''$ *such that* $[\![P]\!] \to^* (\boldsymbol{\nu}\widetilde{x})P''$ *and* $P'' = [\![P']\!]$*;*
2. (Soundness) $[\![P]\!] \to^* P_*$ *implies* $\exists \widetilde{x}, P'', P'$ *s.t.* $P_* \to^* (\boldsymbol{\nu}\widetilde{x})P''$, $P \to^* P'$ *and* $[\![P']\!] \xrightarrow{\text{with}}{}^* P''$.

Proof. Direct consequence of Theorem E.27, noticing that $\boldsymbol{\sigma}(\varnothing)$ is vacuous. ◀