Journal of Logical and Algebraic Methods in Programming 97 (2018) 55–84

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

Multiparty session types, beyond duality

Alceste Scalas *, Nobuko Yoshida *

Imperial College London, UK

ARTICLE INFO

Article history:
Received 16 August 2017
Accepted 22 January 2018
Available online xxxx

Keywords:
Concurrency
Process calculi
Multiparty session types
Duality

ABSTRACT

Multiparty Session Types (MPST) are a well-established typing discipline for message-passing processes interacting on *sessions* involving two or more participants. Session typing can ensure desirable properties: absence of communication errors and deadlocks, and protocol conformance.

We propose a novel MPST theory based on a *rely/guarantee typing system*, that checks *(1)* the *guaranteed* behaviour of the process being typed, and *(2)* the *relied upon* behaviour of other processes. Crucially, our theory achieves type safety by enforcing a typing context *liveness* invariant throughout typing derivations.

Unlike "classic" MPST works, our typing system does not depend on global session types, and does not use syntactic *duality* checks. As a result, our new theory can prove type safety for processes that implement protocols with complex inter-role dependencies, thus sidestepping an intrinsic limitation of "classic" MPST.

1. Introduction

Session types are a typing discipline for *processes* that exchange messages via *channels*, and implement *sessions* with structured protocols featuring inputs/outputs, choices, and recursion. The goal of session typing is to statically ensure that a given process correctly implements a given protocol, without causing communication errors or deadlocks at run-time. Years of research have shown that the session-based framework is applicable to a wide range of calculi, programming languages, and computing environments [23,1,16].

Binary session types In their original formulation, session types describe *binary* sessions involving exactly two participants, or *roles* [20,19,35]. In binary session theories, types are similar to $S = \oplus\mathtt{m}_1 . \&\mathtt{m}_2$, meaning "send message $\mathtt{m}_1$, and then receive message $\mathtt{m}_2$." Typing judgements have the following form:

$$\vdash P \triangleright \Delta \tag{1}$$

meaning that process P uses its communication channels according to the typing context Δ — which in turn maps channels to session types. Hence, when Δ contains the mapping $c : S$, the judgement (1) means that P uses the channel c first to send $\mathtt{m}_1$, and then to receive $\mathtt{m}_2$ — as prescribed by S.

Notably, the typing judgement (1) does not only determine the behaviour of P: it also implicitly describes how *other* processes in P's environment can send/receive messages to/from P. This is because in a well-typed system, each input/output

* Corresponding author.
E-mail addresses: alceste.scalas@imperial.ac.uk (A. Scalas), n.yoshida@imperial.ac.uk (N. Yoshida).

https://doi.org/10.1016/j.jlamp.2018.01.001
2352-2208/

of P must match a corresponding output/input of some other process playing the unique "opposite" role in the corresponding binary session. This complementarity between session endpoints is formalised as *duality*: a fundamental element of the binary session types theory, inspired by linear logic [18] as originally stated in [19], and subsequently studied in various works (e.g., [9,36,37,10]). By duality, if P interacts with another process Q through some channels, then we can infer the following typing judgement:

$$\vdash Q \triangleright \overline{\Delta} \tag{2}$$

where $\overline{\Delta}$ is the *dual of* Δ: it contains the same channels, but mapped to dual types where each input is turned into a corresponding output, and each output into an input. For example, if Δ contains the mapping $c{:}S$ above, then we can infer that $\overline{\Delta}$ contains $c{:}\overline{S}$, where $\overline{S} = \&\mathtt{m}_1.\oplus\mathtt{m}_2$ — meaning "use channel c to receive message $\mathtt{m}_1$, and then send $\mathtt{m}_2$." Now, since (2) guarantees that Q behaves dually w.r.t. P, we know that the parallel composition of P and Q will execute correctly: e.g., P and Q will use channel c to exchange messages $\mathtt{m}_1$ and then $\mathtt{m}_2$, according to types S and $\overline{S}$.

Multiparty session types Binary session types do not allow to directly formalise and verify protocols with more than two interacting roles. This limitation has been addressed by *Multiparty Session Types (MPST)*, which were introduced in [21] and have been actively studied both in theory and applications (see related work in [22]). In a nutshell, the MPST framework foster a top-down approach where:

(1) a *choreography* involving two or more interacting *roles* is formalised as a *global type* G;
(2) G is *projected* onto a set of *(local) session types* $S_1, \ldots, S_n$ (one per role of G), each one describing the expected inputs/outputs of a specific role from/to other roles in G;
(3) the session types $S_1, \ldots, S_n$ are assigned to *channels*, used by *MPST π-calculus processes* that are type-checked against those types (i.e., $S_1, \ldots, S_n$).

In the multiparty setting, typing judgements still look like (1); however, types contain role annotations, and look like $S' = \mathtt{alice}\&\mathtt{m}_1.\mathtt{bob}\&\mathtt{m}_2.\mathtt{carol}\oplus\mathtt{m}_3$, meaning "receive $\mathtt{m}_1$ from $\mathtt{alice}$, then receive $\mathtt{m}_2$ from $\mathtt{bob}$, then send $\mathtt{m}_3$ to $\mathtt{carol}$." As a consequence, the duality-based reasoning outlined above (for binary session types) does not work, and duality cannot be used directly. This is because in a multiparty session, P might interact with *more than one* other processes playing different roles in a session (e.g., $\mathtt{alice}$, $\mathtt{bob}$ and $\mathtt{carol}$) — and correspondingly, the multiparty typing context Δ would contain types requiring inputs/outputs from/to multiple roles (like S'). Hence, instead of a single "other process" Q as in (2), we might have n other typed processes $Q_1, \ldots, Q_n$ playing distinct roles:

$$\vdash Q_1 \triangleright \Gamma_1 \qquad \vdash Q_2 \triangleright \Gamma_2 \qquad \ldots \qquad \vdash Q_n \triangleright \Gamma_n \tag{3}$$

Furthermore, $Q_1, \ldots, Q_n$ might exchange messages between each other — and these inputs/outputs, albeit occurring in a multiparty session involving P, would not be "seen" by P. This means that such message exchanges would be described by the types in $\Gamma_1, \ldots, \Gamma_n$, but could not be inferred by "dualising" Δ, unlike the binary case. This is problematic because such invisible communications might determine which future inputs/outputs $Q_1, \ldots, Q_n$ will be willing to receive/send from/to P.

In the MPST theory, this lack of contextual information causes difficulties when proving *subject reduction* — i.e., when proving that typed processes reduce type-safely. These difficulties are addressed by imposing a "generalised duality" restriction, called *consistency*, that must hold for *all* typing contexts $\Delta, \Gamma_1, \ldots, \Gamma_n$ — taken both in isolation, or in combination. Consistency ensures that well-typed processes will not engage in incompatible input/outputs at run-time; but unfortunately, consistency also severely restricts the set of MPST processes that can be typed and/or proven type-safe: in particular, complex protocols with recursion and message dependencies are often non-consistent, and thus, unsupported by the "classic" MPST theory.[1]

Contributions We present a new multiparty session typing system that sidesteps the difficulties and limitations of consistency in "classic" MPST. Its typing judgement has the following form:

$$\vdash P \triangleright \Delta \triangleleft \Gamma$$

and intuitively reads: P uses its channels as specified by the *guarantee context* Δ, assuming that P is composed with other processes that use their channels according to the *rely context* Γ. For example, to type Q_1 in (3), the guarantee context is Γ_1, while the rely context is $\Gamma_2, \ldots, \Gamma_n$, abstracting the behaviour of other processes composed with Q_1 (in this case, $Q_2, \ldots, Q_n$). With this rely/guarantee approach, our typing judgement conveys a global picture of the possible interactions between a process P and its environment, that is used to verify the absence of communication errors, and determine which processes can safely interact with P.

[1] The reason for the consistency constraint, and its limitations, are rather technical; we discuss them in detail in Section 3.

Channels	c, d	::=	$x \mid s[\mathtt{p}]$	(variable, channel with role $\mathtt{p}$)
Declarations	D	::=	$X(\widetilde{x}) = P$	(declaration of process variable X as P)
Processes	P, Q	::=	$c[\mathtt{q}]\oplus\mathtt{m}\langle d\rangle.P$	(selection towards role $\mathtt{q}$)
			$c[\mathtt{q}]\sum_{i\in I}\{\mathtt{m}_i(x_i).P_i\}$	(branching from role $\mathtt{q}$, with $I \neq \emptyset$)
			$P \mid Q$	(parallel composition)
			$(\nu s)\,P$	(session restriction/creation)
			def D **in** P	(process definition)
			$X\langle\widetilde{c}\rangle$	(process call)
			$\mathbf{0}$	(terminated process)

Fig. 1. Syntax of the standard MPST π-calculus. Here, $\widetilde{x}$ represents a sequence $x_1, \ldots, x_n$, and $\widetilde{c}$ represents a sequence $c_1, \ldots, c_n$ (for some $n \geq 0$).

Crucially, our typing system replaces consistency with a *liveness* constraint for typing contexts. Since liveness is a *behavioural* (rather than syntactic) property, our theory does *not* depend on the existence of global types: hence, it does not depend on the associated technicalities (necessary, e.g., to compute *global/local type projections and merges*), and supports multiparty protocols that cannot be projected from any global type.

Outline of the paper In Section 2, we present the standard definition of multiparty session processes, types, and typing contexts. In Section 3 we reprise the discussion above about consistency in "classic" MPST, formally explaining its limitations. In Section 4 we present our novel rely/guarantee multiparty session typing system (Definition 4.5), and its subject reduction property (Theorem 4.9), that does *not* depend on consistency. In Section 5, we highlight the differences between our new theory and "classic" MPST: we show that we support all protocols projected from any global type — and for this purpose, in Section 5.1 we relate liveness with a property of Communicating Finite-State Machines [8] called *Multiparty Compatibility*; in addition, we show that we also support protocols that *cannot* be projected from any global type. In Section 6, we conclude and discuss related works. Detailed proofs are available in the appendices.

2. Multiparty sessions: standard calculus, types and typing contexts

In this section, we summarise some key elements of the "classic" Multiparty Session Types (MPST) theory. We describe the multiparty session π-calculus and types, adopting a notation roughly based on Coppo et al. [11] and Scalas et al. [33] (i.e., the most common formulation in MPST literature). We focus on a streamlined presentation, that will be sufficient for highlighting the limitations of "classic" MPST (Section 3) and for developing our new theory (Section 4).

We present the MPST π-calculus in Section 2.1, and session types and typing contexts in Section 2.2.

2.1. Multiparty session π-calculus

The multiparty session π-calculus is a variant of the π-calculus [30,29,32] where channels allow two *or more* roles to exchange messages.

Definition 2.1 *(MPST π-calculus).* MPST processes have the syntax shown in Fig. 1. Session restriction, branching, and declarations act as binders, as expected. We write $\mathrm{fc}(P)$ for the *free channels with roles in* P, $\mathrm{dpv}(D)$ for the *process variables declared in* D, $\mathrm{fpv}(D)$ for the *free process variables in* D, and $\mathrm{fpv}(P)$ for the *free process variables in* P. We write $s \notin \mathrm{fc}(P)$ iff there is no $\mathtt{p}$ such that $s[\mathtt{p}] \in \mathrm{fc}(P)$.

A **channel** c or d can be either a **variable** x, or a **channel with role** $s[\mathtt{p}]$: the latter represents a *multiparty channel endpoint* used to play the *role* $\mathtt{p}$ in the *session* s, and send/receive messages to/from other roles in s; variables, instead, are placeholders for channels with role, and are instantiated at run-time (details below). A **message** consists of a *label* $\mathtt{m}$ and a *payload*: different labels allow to distinguish different kinds of messages, while the payload is the actual data transmitted between processes. Note that payloads are channels themselves (and with minor extensions, they can be values like strings or integers — see Remark 2.2 later on). A **process** P or Q can be either:

- a **selection** that uses c to send a message with label $\mathtt{m}$ and payload d to $\mathtt{q}$, and then continues as P;
- a **branching** that uses c to receive from $\mathtt{q}$ a message with label $\mathtt{m}_i$ (for any $i \in I$); then, depending on which $\mathtt{m}_i$ has been received, the process instantiates the variable x_i with the message payload, and continues as P_i;
- a **parallel composition** where P and Q run concurrently, and possibly communicate through their channels;
- a **session restriction** (or **session creation**) delimiting the scope of s to P;
- a **process definition** declaring X (with *parameters* $\widetilde{x}$) as P, with scope Q;
- a **process call** where X is invoked with *actual parameters* $\widetilde{c}$;
- a **terminated process** $\mathbf{0}$.

Remark 2.2 *(Abbreviations and simplifications).* For brevity, when writing MPST π-calculus processes we will often omit trailing $\mathbf{0}$s and immaterial message payloads, i.e.:

(a) $c[\mathtt{p}]\oplus\mathtt{m}.P$ stands for $(\nu s)\,(c[\mathtt{p}]\oplus\mathtt{m}\langle s[\mathtt{q}]\rangle.P)$ with some $\mathtt{q}$ and $s \notin \mathrm{fc}(P)$;

$$[\text{R-Comm}]\quad s[\mathtt{p}][\mathtt{q}]\textstyle\sum_{i\in I}\{\mathtt{m}_i(x_i).P_i\} \mid s[\mathtt{q}][\mathtt{p}]\oplus\mathtt{m}_k\langle s'[\mathtt{r}]\rangle.Q \;\to\; P_k\{^{s'[\mathtt{r}]}/_{x_k}\} \mid Q \quad (k\in I)$$

$$[\text{R-}X]\quad \mathbf{def}\ X(x_1,\ldots,x_n)=P\ \mathbf{in}\ (X\langle s_1[\mathtt{p}_1],\ldots,s_n[\mathtt{p}_n]\rangle \mid Q) \to \mathbf{def}\ X(x_1,\ldots,x_n)=P\ \mathbf{in}\ (P\{^{s_1[\mathtt{p}_1]}/_{x_1}\}\cdots\{^{s_n[\mathtt{p}_n]}/_{x_n}\} \mid Q)$$

$$[\text{R-}|]\quad P\to P' \text{ implies } P\mid Q\to P'\mid Q \qquad [\text{R-}\nu]\quad P\to P' \text{ implies } (\nu s)\,P\to(\nu s)\,P'$$

$$[\text{R-def}]\quad P\to P' \text{ implies } \mathbf{def}\ X(\widetilde{x})=Q\ \mathbf{in}\ P \to \mathbf{def}\ X(\widetilde{x})=Q\ \mathbf{in}\ P'$$

$$[\text{R-}\equiv]\quad P'\equiv P\to Q\equiv Q' \text{ implies } P'\to Q'$$

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

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

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

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

Fig. 2. Standard MPST π-calculus: semantics (top) and structural congruence $\equiv$ (bottom).

(b) $c[\mathtt{p}]\sum\{\mathtt{m1}.P_1\,,\,\mathtt{m2}(x).P_2\}$ stands for $c[\mathtt{p}]\sum\{\mathtt{m1}(z).P_1\,,\,\mathtt{m2}(x).P_2\}$ with $z\notin\mathrm{fv}(P_1)$.

Furthermore, note that the calculus in Definition 2.1 can be easily extended with, e.g., basic values (strings, integers, booleans…) or conditionals. Such extensions are routine and orthogonal to our treatment; we will sometimes use them in examples, to make them more readable.

The abbreviations described in Remark 2.2 omit either *(a)* a channel with role $s[\mathtt{p}]$ that is sent just after s is created and just before s is discarded, or *(b)* a message payload variable z that is never referenced. We will see that the two abbreviations are often combined: in case *(a)* the message payload $s[\mathtt{p}]$ does not allow to interact with any other role (hence, the only "meaningful" element of the message is the label $\mathtt{m}$, that can trigger a corresponding branch of the recipient process); correspondingly, the recipient of $\mathtt{m}$ can safely discard its payload, as in case *(b)*.

Example 2.3. Below is an example of MPST π-calculus process:

$$(\nu s)\,(P_{\mathtt{a}}\mid P_{\mathtt{b}}\mid P_{\mathtt{c}}) \qquad \text{where:}\quad \begin{aligned} P_{\mathtt{a}} &= s[\mathtt{alice}][\mathtt{bob}]\oplus\mathtt{stop}\\ P_{\mathtt{b}} &= s[\mathtt{bob}][\mathtt{alice}]\textstyle\sum\left\{\begin{array}{l}\mathtt{m1}(x).s[\mathtt{bob}][\mathtt{carol}]\oplus\mathtt{m2}\langle\,\texttt{"}Hi\texttt{"}\rangle\,,\\ \mathtt{stop}.s[\mathtt{bob}][\mathtt{carol}]\oplus\mathtt{quit}\end{array}\right\}\\ P_{\mathtt{c}} &= s[\mathtt{carol}][\mathtt{bob}]\textstyle\sum\left\{\begin{array}{l}\mathtt{m2}(y).s[\mathtt{carol}][\mathtt{alice}]\oplus\mathtt{m3}\langle true\rangle\,,\\ \mathtt{quit}\end{array}\right\}\end{aligned} \tag{4}$$

Here, $(\nu s)\,(P_{\mathtt{a}}\mid P_{\mathtt{b}}\mid P_{\mathtt{c}})$ is the parallel composition of processes $P_{\mathtt{a}}$, $P_{\mathtt{b}}$, and $P_{\mathtt{c}}$ in the scope of the *multiparty session* s. Inside such processes, $s[\mathtt{alice}]$ (resp. $s[\mathtt{bob}]$, $s[\mathtt{carol}]$) is a *channel with role*: it is used to send/receive messages on s, while playing the role of $\mathtt{alice}$ (resp. $\mathtt{bob}$, $\mathtt{carol}$). In the process $P_{\mathtt{a}}$, the selection "$s[\mathtt{alice}][\mathtt{bob}]\oplus\mathtt{stop}$" means: the channel with role $s[\mathtt{alice}]$ is used to play role $\mathtt{alice}$ in session s, and send $\mathtt{stop}$ to $\mathtt{bob}$. In process $P_{\mathtt{b}}$, the channel with role $s[\mathtt{bob}]$ is used to play role $\mathtt{bob}$ in s and receive ($\sum$) either $\mathtt{m1}$ or $\mathtt{stop}$ from $\mathtt{alice}$; then, in the first case, $s[\mathtt{bob}]$ is used to send $\mathtt{m2}$ to $\mathtt{carol}$; in the second case, $s[\mathtt{bob}]$ is used to send $\mathtt{quit}$ to $\mathtt{carol}$.

Note that in (4), the trailing **0**s and some message payloads are omitted, as explained in Remark 2.2.

We adopt the standard *reduction relation* $\to$ *for MPST* π*-calculus processes*, formalised in Definition 2.4.

Definition 2.4 *(MPST* π*-calculus congruence and semantics).* The *structural congruence* $\equiv$ between MPST π-calculus processes is inductively defined by the rules in Fig. 2 (bottom). The *reduction relation* $\to$ between MPST π-calculus processes is inductively defined as shown in Fig. 2 (top). We write $\twoheadrightarrow$ to denote the reflexive and transitive closure of $\to$.

Fig. 2 contains the following rules:

- [R-Comm] is the **communication rule**: it says that the parallel composition of a branching process and a selection process operating on the same session s, and playing respectively the roles $\mathtt{p}$ and $\mathtt{q}$, can reduce provided that *(1)* the two processes are targeting each other (i.e., the branching is from $\mathtt{q}$ and the selection is towards $\mathtt{p}$), *(2)* the branching process supports the selected label $\mathtt{m}_k$, and *(3)* the selection payload is a channel with role $s'[\mathtt{r}]$ (not a variable). If these conditions hold, the two processes reduce to the respective continuations P_k and Q — with $s'[\mathtt{r}]$ replacing x_k in the former;
- [R-X] is the **process call rule**: inside the scope of the process declaration $X(\widetilde{x})=P$, calling X amounts to expanding P and replacing the formal parameters $\widetilde{x}$ with the actual parameters of the call (that must be channels with roles, and not variables);

- rules [R-|], [R-ν], and [R-**def**] are the **parallel, restriction, and process definition rules**: they say that reductions can occur respectively inside parallel compositions, session restrictions and process definitions;
- [R-$\equiv$] is the **up-to congruence rule**: it says that if P reduces to Q, then any $P' \equiv P$ reduces to any $Q' \equiv Q$. The **process congruence**, in turn, is standard: it says that $\mid$ is a commutative monoid with $\mathbf{0}$ as neutral element, and garbage-collects unnecessary session restrictions and process declarations, or reorders them without name capturing.

Example 2.5. Take the process in Example 2.3. By Definition 2.4, it reduces as follows:

$$(\nu s)\,(P_{\mathtt{a}} \mid P_{\mathtt{b}} \mid P_{\mathtt{c}}) \;\to\; (\nu s)\,(\mathbf{0} \mid s[\mathtt{bob}][\mathtt{carol}]\oplus\mathtt{quit} \mid P_{\mathtt{c}}) \;\to\; (\nu s)\,(\mathbf{0} \mid \mathbf{0} \mid \mathbf{0}) \;\equiv\; (\nu s)\,\mathbf{0} \;\equiv\; \mathbf{0}$$

2.2. Multiparty session types and typing contexts

Similarly to "classic" MPST works, our new typing system (introduced later on, in Section 4) will assign *session types* (Definition 2.6 below) to channels, using *typing contexts* (Definition 2.9). For both, we adopt standard definitions from MPST literature.

Definition 2.6 *(Multiparty session types). Session types* (ranged over by S, T) have the syntax:

$$S, T \;::=\; \mathtt{p}\&_{i\in I}\mathtt{m}_i(S_i).S_i' \;\mid\; \mathtt{p}\oplus_{i\in I}\mathtt{m}_i(S_i).S_i' \;\mid\; \mathbf{end} \;\mid\; \mu\mathbf{t}.S \;\mid\; \mathbf{t} \qquad \text{with } I \neq \emptyset$$

where $\mathtt{m}_i$ range over pairwise distinct *message labels*. We require recursive types to be *contractive*: i.e., in $\mu\mathbf{t}.S$ we have $S \neq \mathbf{t}'$ ($\forall \mathbf{t}'$). We define $\equiv$ as the *coinductive equivalence between type trees*, such that $\mu\mathbf{t}.S \equiv S\{^{\mu\mathbf{t}.S}/_{\mathbf{t}}\}$ (i.e., a recursive type is equivalent to its unfolding, cf. Pierce [31, §21.7 and §21.8]).

In Definition 2.6, the **branching type** (or *external choice*) $\mathtt{p}\&_{i\in I}\mathtt{m}_i(S_i).S_i'$ says that a channel must be used to receive from $\mathtt{p}$ one input of the form $\mathtt{m}_i(S_i)$, for any $i \in I$ chosen by $\mathtt{p}$; then, the channel must be used following the *continuation type* S_i'. The **selection type** (or *internal choice*) $\mathtt{p}\oplus_{i\in I}\mathtt{m}_i(S_i).S_i'$, instead, says that a channel must be used to perform one output $\mathtt{m}_i(S_i)$ towards $\mathtt{p}$, for some $i \in I$, and then according to S_i'. The type **end** describes a **terminated** channel that cannot be further used for inputs/outputs. Finally, $\mu\mathbf{t}.S$ is a **recursive** session type: μ binds the *recursion variable* $\mathbf{t}$ in S. We distinguish a recursive type from its unfolding; when necessary, we explicitly relate them with the equivalence $\equiv$.

Remark 2.7. For brevity, we will often omit the trailing **end** in types, and **end**-typed message payloads: e.g., $\mathtt{p}\oplus\mathtt{m}$ stands for $\mathtt{p}\oplus\mathtt{m}(\mathbf{end}).\mathbf{end}$.

For readability, we will sometimes write examples using ground types like Int, Bool, *etc.*: their addition to Definition 2.6 is standard, and we omit it for simplicity.

Example 2.8. Consider the following session types:

$$S_{\mathtt{a}} = \mathtt{bob}\oplus\left\{\begin{array}{l}\mathtt{m1(Int)}\,.\,\mathtt{carol}\&\mathtt{m3(Bool)}\,.\,\mathbf{end}\,,\\ \mathtt{stop}\,.\,\mathbf{end}\end{array}\right\} \qquad S_{\mathtt{b}} = \mathtt{alice}\&\left\{\begin{array}{l}\mathtt{m1(Int)}\,.\,\mathtt{carol}\oplus\mathtt{m2(Str)}\,.\,\mathbf{end}\,,\\ \mathtt{stop}\,.\,\mathtt{carol}\oplus\mathtt{quit}\,.\,\mathbf{end}\end{array}\right\}$$
$$S_{\mathtt{c}} = \mathtt{bob}\&\left\{\begin{array}{l}\mathtt{m2(Str)}\,.\,\mathtt{alice}\oplus\mathtt{m3(Bool)}\,.\,\mathbf{end}\,,\\ \mathtt{quit}\,.\,\mathbf{end}\end{array}\right\} \tag{5}$$

Intuitively, $S_{\mathtt{a}}$ says that a process using an $S_{\mathtt{a}}$-typed channel must send ($\oplus$) to $\mathtt{bob}$ a message that can be either $\mathtt{m1(Int)}$, or $\mathtt{stop}$; in the first case, the process must then use the same channel to receive (&) message $\mathtt{m3(Bool)}$ from $\mathtt{carol}$, after which the session **end**s; otherwise, in the second case, the session **end**s without further interactions. The other types follow a similar intuition.

Note that in (5), **end**-typed message payloads are omitted, and we use ground types as explained in Remark 2.7.

We also adopt the standard definitions of *MPST typing context* (Definition 2.9) and typing context reduction (Definition 2.10).

Definition 2.9 *(Typing context).* A *multiparty session typing context*, denoted by Δ or Γ, is a partial mapping defined as:

$$\Delta, \Gamma \;::=\; \emptyset \;\mid\; \Delta, x{:}S \;\mid\; \Delta, s[\mathtt{p}]{:}S$$

- The *typing context composition* Δ_1, Δ_2 is defined iff $\mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2) = \emptyset$.
- We write $\Delta \equiv \Delta'$ iff $\mathrm{dom}(\Delta) = \mathrm{dom}(\Delta')$ and $\forall c \in \mathrm{dom}(\Delta) : \Delta(c) \equiv \Delta'(c)$.
- We define $\Delta \backslash c$ so that $(\Delta \backslash c)(d) = \Delta(d)$ iff $d \neq c$ (and is undefined otherwise).

As anticipated in Section 1, a multiparty session typing context maps channels (either variables or channels with role) to session types, thus specifying how a channel is expected to be used. The typing context composition Δ_1, Δ_2 is defined iff the domains of Δ_1 and Δ_2 do not overlap.[2] The notation $\Delta \setminus c$ stands for the typing context defined as Δ, minus its entry for c (if present).

Multiparty session typing contexts are equipped with reduction semantics: this feature can be found in many typing systems for process calculi, when types abstract the behaviour of processes (for a survey, see [23]).

Definition 2.10 *(Typing context reduction).* The *typing contexts reduction relation* $\rightarrow$ is inductively defined by the following rules:

$$\frac{k \in I \quad S_k \equiv T_k}{s[\mathtt{p}]:\mathtt{q}\oplus_{i\in I}\mathtt{m}_i(S_i).S'_i,\ s[\mathtt{q}]:\mathtt{p}\&_{i\in I\cup J}\mathtt{m}_i(T_i).T'_i \ \rightarrow\ s[\mathtt{p}]:S'_k, s[\mathtt{q}]:T'_k}\ [\text{T-Comm}]$$

$$\frac{k \in I}{x:\mathtt{p}\oplus_{i\in I}\mathtt{m}_i(S_i).S'_i \rightarrow x:S'_k}\ [\text{T-}x\oplus] \qquad \frac{k \in I}{x:\mathtt{p}\&_{i\in I}\mathtt{m}_i(S_i).S'_i \rightarrow x:S'_k}\ [\text{T-}x\&]$$

$$\frac{\Delta, c:S\{\mu\mathbf{t}.S/\mathbf{t}\} \rightarrow \Delta'}{\Delta, c:\mu\mathbf{t}.S \rightarrow \Delta'}\ [\text{T-}\mu] \qquad \frac{\Delta \rightarrow \Delta'}{\Delta, c:S \rightarrow \Delta', c:S}\ [\text{T-Comp}]$$

We write $\twoheadrightarrow$ to denote the reflexive and transitive closure of $\rightarrow$.

The rules in Definition 2.10 have the following meaning:

- [T-Comm] says that two entries $s[\mathtt{p}]:S_\mathtt{p}, s[\mathtt{q}]:S_\mathtt{q}$ can "interact", and reduce to their continuations, provided that *(1)* $S_\mathtt{p}$ is a selection towards $\mathtt{q}$, *(2)* $S_\mathtt{q}$ is a branching from $\mathtt{p}$, and *(3)* the message labels of $S_\mathtt{q}$ are a superset of those in $S_\mathtt{p}$, with equivalent carried types. The last clause implies that if $S_\mathtt{p}$ has some output message that is not supported by $S_\mathtt{q}$, then the pair $s[\mathtt{p}]:S_\mathtt{p}, s[\mathtt{q}]:S_\mathtt{q}$ is stuck;
- [T-$x\oplus$] and [T-$x\&$] says that an entry $x:S$ (i.e., mapping a variable to a type) can reduce autonomously. These rules are a minor extension w.r.t. "classic" MPST papers; they will be useful later on, to define our typing system (Section 4) and formalise some results (Section 5.3);
- [T-μ] says that a recursive type can take part in a reduction, provided that its unfolding can induce the same reduction;
- [T-Comp] says that a reduction can occur within a larger typing context, with each additional entry $c:S$ left unchanged.

Example 2.11. Consider the typing context:

$$\Delta_\mu = s[\mathtt{p}]:\mu\mathbf{t}.\mathtt{q}\oplus\mathtt{m}.\mathbf{t}, s[\mathtt{q}]:\mu\mathbf{t}.\mathtt{p}\&\mathtt{m}.\mathbf{t}$$

By Definition 2.10, it reduces as follows (for clarity, we restore some **end** payloads omitted above):

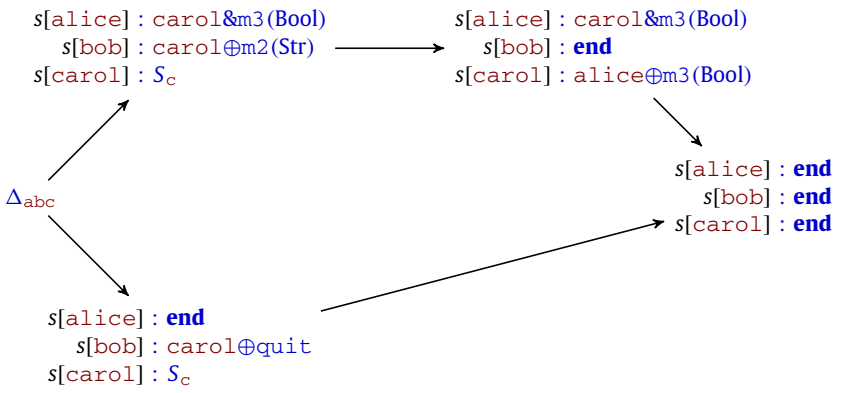
$$\cfrac{\cfrac{\cfrac{\mathbf{end} \equiv \mathbf{end}}{s[\mathtt{p}]:\mathtt{q}\oplus\mathtt{m}(\mathbf{end}).\mu\mathbf{t}.\mathtt{q}\oplus\mathtt{m}.\mathbf{t}, s[\mathtt{q}]:\mathtt{p}\&\mathtt{m}(\mathbf{end}).\mu\mathbf{t}.\mathtt{p}\&\mathtt{m}.\mathbf{t} \ \rightarrow\ \Delta_\mu}\ [\text{T-Comm}]}{s[\mathtt{p}]:\mathtt{q}\oplus\mathtt{m}.\mu\mathbf{t}.\mathtt{q}\oplus\mathtt{m}.\mathbf{t}, s[\mathtt{q}]:\mu\mathbf{t}.\mathtt{p}\&\mathtt{m}.\mathbf{t} \ \rightarrow\ \Delta_\mu}\ [\text{T-}\mu]}{\Delta_\mu \rightarrow \Delta_\mu}\ [\text{T-}\mu]$$

Example 2.12. Consider the types $S_\mathtt{a}$, $S_\mathtt{b}$, and $S_\mathtt{c}$ in Example 2.8, and the following typing context:

$$\Delta_{\mathtt{abc}} = s[\mathtt{alice}]:S_\mathtt{a}, s[\mathtt{bob}]:S_\mathtt{b}, s[\mathtt{carol}]:S_\mathtt{c}$$

By Definition 2.10, $\Delta_{\mathtt{abc}}$ can reduce as follows, by rules [T-Comm] and [T-Comp]:

[2] As in many MPST works, we could slightly relax the condition by allowing Δ_1 and Δ_2 to overlap on **end**-typed entries (i.e., treat such entries as *unrestricted* rather than *linear*). For simplicity, we treat all typing context entries in the same way. Note that this does not impact the expressiveness of the typed fragment of our MPST calculus: in fact, **end**-typed entries can be created "for free" when necessary, using session restriction.



In the upper part of the transition diagram, $s[\texttt{alice}]$ sends $\texttt{m1}$ to $s[\texttt{bob}]$, who sends $\texttt{m2}$ to $s[\texttt{carol}]$, who sends $\texttt{m3}$ to $s[\texttt{alice}]$. In the lower part of the diagram, $s[\texttt{alice}]$ sends $\texttt{stop}$ to $s[\texttt{bob}]$, who sends $\texttt{quit}$ to $s[\texttt{carol}]$. In both cases, the reductions reach a final typing context configuration where all entries are **end**-typed.

3. "Classic" multiparty session typing system and subject reduction

In this section, we outline the "classic" multiparty session typing system, and its limitations. We reprise our claims in Section 1 more formally: we discuss "classic" multiparty session typing (Section 3.1), subject reduction (Section 3.2), and the limitations introduced by consistency (Section 3.3). Moreover, in Section 3.4, we discuss further cases of protocols that are "safe" but non-consistent: the fact that "classic" MPST do *not* cover such simple examples suggests that a more flexible approach is desirable.

We will provide pointers to some definitions in Appendix A, that are technically necessary in "classic" MPST theory. We relegate such technicalities in the appendix to make this section more readable — and also because they will *not* be needed for our new typing system in Section 4.

3.1. "Classic" multiparty session typing, global types, and projections

The "classic" multiparty session typing judgements have the form shown in Section 1, eq. (1). We repeat it here for clarity:

$$\vdash P \triangleright \Delta \qquad \text{with } \Delta \text{ from Definition 2.9} \tag{6}$$

which reads: "P uses its channels as described in Δ".

Under the typical "top-down" approach outlined in Section 1, the typing context Δ usually contains multiparty session types that are *projected from some given global type*, whose syntax is formalised in Definition 3.1 below.

Definition 3.1 *(Global types).* The syntax of a *global type* G is:

$$G \quad ::= \quad \mathtt{p}\to\mathtt{q}\colon \{\mathtt{m}_i(T_i)\,.\,G_i\}_{i\in I} \;\mid\; \mu\mathbf{t}.G \;\mid\; \mathbf{t} \;\mid\; \mathbf{end} \qquad \text{where } \mathtt{p}\neq\mathtt{q},\ I\neq\emptyset,\ \text{and } \forall i\in I : \mathrm{fv}(T_i)=\emptyset$$

We require recursive global types to be *contractive*, i.e., in $\mu\mathbf{t}.G$ we have $G \neq \mathbf{t}'$ $(\forall \mathbf{t}')$. We will sometimes omit **end**-typed payloads, and trailing **end**s: i.e., $\mathtt{p}\to\mathtt{q}\colon\mathtt{m}$ stands for $\mathtt{p}\to\mathtt{q}\colon\mathtt{m}(\mathbf{end})\,.\,\mathbf{end}$.

We explain Definition 3.1 with an example. Consider the global type G below:

$$G = \mathtt{alice}\to\mathtt{bob}\colon \begin{Bmatrix} \mathtt{m1}(\mathrm{Int})\,.\,\mathtt{bob}\to\mathtt{carol}\colon\mathtt{m2}(\mathrm{Str})\,.\,\mathtt{carol}\to\mathtt{alice}\colon\mathtt{m3}(\mathrm{Bool})\,.\,\mathbf{end}, \\ \mathtt{stop}\,.\,\mathtt{bob}\to\mathtt{carol}\colon\mathtt{quit}\,.\,\mathbf{end} \end{Bmatrix} \tag{7}$$

The global type G involves three roles: $\texttt{alice}$, $\texttt{bob}$, and $\texttt{carol}$. It says that $\texttt{alice}$ sends to $\texttt{bob}$ *either* a message $\texttt{m1}$ (carrying an Integer) *or* $\texttt{stop}$; in the first case, $\texttt{bob}$ sends $\texttt{m2}$ to $\texttt{carol}$ (carrying a String), then $\texttt{carol}$ sends $\texttt{m3}$ to $\texttt{alice}$ (carrying a Boolean), and the session **end**s; otherwise, in the second case, $\texttt{bob}$ sends $\texttt{quit}$ to $\texttt{carol}$, and the session **end**s.

The *projections of* G (Definition A.1) describe the I/O actions expected from processes that play the roles in G. When G is projected onto $\texttt{alice}$, $\texttt{bob}$ and $\texttt{carol}$, it yields respectively the types $S_{\mathtt{a}}$, $S_{\mathtt{b}}$ and $S_{\mathtt{c}}$ shown in Example 2.8. Such types can be used in the typing judgement (6), in combination with the processes in Example 2.3, to yield a typing derivation similar to the following:

$$\dfrac{\dfrac{\dfrac{\vdots}{\vdash P_{\mathtt{a}} \triangleright s[\mathtt{alice}]:S_{\mathtt{a}}} \quad \dfrac{\vdots}{\vdash P_{\mathtt{b}} \triangleright s[\mathtt{bob}]:S_{\mathtt{b}}}}{\vdash P_{\mathtt{a}} \mid P_{\mathtt{b}} \triangleright s[\mathtt{alice}]:S_{\mathtt{a}}, s[\mathtt{bob}]:S_{\mathtt{b}}}\ [\text{STD-}|] \quad \dfrac{\vdots}{\vdash P_{\mathtt{c}} \triangleright s[\mathtt{carol}]:S_{\mathtt{c}}}}{\vdash P_{\mathtt{a}} \mid P_{\mathtt{b}} \mid P_{\mathtt{c}} \triangleright s[\mathtt{alice}]:S_{\mathtt{a}}, s[\mathtt{bob}]:S_{\mathtt{b}}, s[\mathtt{carol}]:S_{\mathtt{c}}}\ [\text{STD-}|] \tag{8}$$

This derivation says that the parallel composition of processes $P_{\mathtt{a}} \mid P_{\mathtt{b}} \mid P_{\mathtt{c}}$ uses the channels with role $s[\mathtt{alice}]$, $s[\mathtt{bob}]$, and $s[\mathtt{carol}]$ respectively according to types $S_{\mathtt{a}}$, $S_{\mathtt{b}}$, and $S_{\mathtt{c}}$. Notice that $P_{\mathtt{a}} \mid P_{\mathtt{b}} \mid P_{\mathtt{c}}$ is typed by rule [STD-|], that splits the typing context in the conclusion so that a channel with role is not used by two parallel sub-processes in its premises. This kind of context splitting implies that channels are used *linearly*, i.e., by one process a time, and cannot be shared.

In the omitted part of the derivation (8), the sub-processes $P_{\mathtt{a}}$, $P_{\mathtt{b}}$, and $P_{\mathtt{c}}$ are typed: this means, in particular, that they use respectively the channel with role $s[\mathtt{alice}]$, $s[\mathtt{bob}]$, and $s[\mathtt{carol}]$ abiding by types $S_{\mathtt{a}}$, $S_{\mathtt{b}}$, and $S_{\mathtt{c}}$. Hence, we know that the processes in Example 2.3 respect the local session types in Example 2.8, and thus, play the roles of the global type G in (7).

Inter-role dependencies Observing the processes in Example 2.3, we can notice that $P_{\mathtt{a}}$ sends $\mathtt{stop}$ to $\mathtt{bob}$, and then terminates without receiving messages from $\mathtt{carol}$. Indeed, this is what we expect from the type $S_{\mathtt{a}}$ in (5): $P_{\mathtt{a}}$ must input a message from $\mathtt{carol}$ if and only if it decides to send $\mathtt{m1}$ (instead of $\mathtt{stop}$) to $\mathtt{bob}$. However, if we observe the process $P_{\mathtt{c}}$, we see that it *could* potentially try to send $\mathtt{m3}$ to $\mathtt{alice}$ — and this should be regarded as an error, because $P_{\mathtt{a}}$ would not be expecting such a message. Fortunately, this does *not* occur: by observing the whole ensemble $P_{\mathtt{a}} \mid P_{\mathtt{b}} \mid P_{\mathtt{c}}$, we can see that when $P_{\mathtt{b}}$ receives $\mathtt{stop}$ from $P_{\mathtt{b}}$, it "forwards" $\mathtt{quit}$ to $\mathtt{carol}$ (as per $S_{\mathtt{b}}$); and when $P_{\mathtt{c}}$ receives $\mathtt{quit}$ from $\mathtt{bob}$, it does not try to communicate again with $\mathtt{alice}$ (as per $S_{\mathtt{c}}$). Intuitively, we can say that the process $P_{\mathtt{a}}$ "assumes" that by following the type $S_{\mathtt{a}}$, and thus sending a certain message to the process playing role $\mathtt{bob}$, it will also influence the behaviour of the process playing $\mathtt{carol}$. This is a case of implicit *inter-role communication dependency*, that can be more explicitly observed in the global type G.

3.2. Subject reduction and consistency

Multiparty session typing ensures that processes enjoy properties such as protocol conformance (i.e., all sent/received messages comply with the session types, and ultimately with their originating global type G), and type safety (i.e., typed processes "never go wrong").

As in most typing systems, type safety for MPST π-calculus processes is based on the *subject reduction* property: it guarantees that typed processes can only reduce to typed processes, and therefore, no untypable configurations can be reached. Among such untypable configurations there are, e.g., the undesired cases where a process is willing to send a message that is not supported by the intended recipient — as in the hypothetical error involving $P_{\mathtt{a}}$ and $P_{\mathtt{b}}$ discussed above. Hence, one might expect that the subject reduction theorem for MPST has a statement similar to:

$$\vdash P \triangleright \Delta \text{ and } P \to^* P' \quad \text{implies} \quad \exists \Delta' \text{ such that } \Delta \to^* \Delta' \text{ and } \vdash P' \triangleright \Delta' \tag{9}$$

Then, one might also expect that the subject reduction statement in (9) can ensure type safety for our typed example in (8), by letting $P = P_{\mathtt{a}} \mid P_{\mathtt{b}} \mid P_{\mathtt{c}}$ and $\Delta = s[\mathtt{alice}]:S_{\mathtt{a}}, s[\mathtt{bob}]:S_{\mathtt{b}}, s[\mathtt{carol}]:S_{\mathtt{c}}$. But quite surprisingly, this is *not* the case! The reason is that (9) is not quite accurate: it does *not* hold for all processes typed by any arbitrary Δ, as illustrated in Example 3.2.

Example 3.2. In "classic" MPST, an arbitrary typing context can type arguably incorrect processes. For example, we have:

$$\vdash s[\mathtt{p}][\mathtt{q}] \oplus \mathtt{m}_\Delta \langle s[\mathtt{r}] \rangle \mid s[\mathtt{q}][\mathtt{p}] \textstyle\sum \mathtt{m}_\Theta(x) \triangleright s[\mathtt{p}]:\mathtt{q} \oplus \mathtt{m}_1(\mathbf{end}),\ s[\mathtt{q}]:\mathtt{p} \,\&\, \mathtt{m}_2(\mathbf{end}),\ s[\mathtt{r}]:\mathbf{end} \qquad (\mathtt{m}_\Delta \neq \mathtt{m}_\Theta) \tag{10}$$

Note that the typed processes in (10) are willing to interact, but are stuck (by Definition 2.4). Eq. (10) can be considered an error configuration, where a process expecting $\mathtt{m}_1$ receives an unsupported message $\mathtt{m}_2$.

Another problematic example can be shown using ground types and expressions, e.g., as in [11, p. 163]:

$$\vdash s[\mathtt{p}][\mathtt{q}] \oplus \mathtt{m}\langle \mathtt{"Hi"} \rangle . s[\mathtt{p}][\mathtt{q}] \textstyle\sum \mathtt{m}(y) \mid s[\mathtt{q}][\mathtt{p}] \sum \mathtt{m}(x) . s[\mathtt{q}][\mathtt{p}] \oplus \mathtt{m}\langle x + 42 \rangle \triangleright \begin{array}{l} s[\mathtt{p}]: \mathtt{q} \oplus \mathtt{m}(\mathrm{Str}) . \mathtt{q} \,\&\, \mathtt{m}(\mathrm{Int}), \\ s[\mathtt{q}]: \mathtt{p} \,\&\, \mathtt{m}(\mathrm{Int}) . \mathtt{p} \oplus \mathtt{m}(\mathrm{Int}) \end{array} \tag{11}$$

By Definition 2.4, the process in (11) reduces to $s[\mathtt{p}][\mathtt{q}] \sum \mathtt{m}(y) \mid s[\mathtt{q}][\mathtt{p}] \oplus \mathtt{m}\langle \mathtt{"Hi"} + 42 \rangle$, which is clearly untypable. This disproves the tentative subject reduction statement in (9).

To avoid the scenarios described in Example 3.2, in "classic" MPST works (e.g., by Coppo et al. [11]), the subject reduction statement is more restrictive, and reads:

$$\vdash P \triangleright \Delta \ \underline{\textit{with } \Delta \textit{ consistent}} \text{ and } ; P \to^* P' \quad \text{implies} \quad \exists \Delta' \ \underline{\textit{consistent}} \text{ such that } \Delta \to^* \Delta' \text{ and } \vdash P' \triangleright \Delta' \tag{12}$$

i.e., subject reduction is only proved for *consistent* (sometimes called *coherent*) typing contexts, as per Definition 3.3 below.

Definition 3.3 *(Consistency).* We say that Δ *is consistent*, written consistent(Δ), iff $\forall \Delta', s, \mathtt{p}, \mathtt{q}, S, T : \Delta = \Delta', s[\mathtt{p}]:S, s[\mathtt{q}]:T$ implies $\overline{S \upharpoonright \mathtt{q}} \leqslant T \upharpoonright \mathtt{p}$.

Checking whether Definition 3.3 holds for some typing context Δ is quite cumbersome, as it requires to:

(1) take each pair of channels with role in Δ belonging to the same session s — say, having roles $\mathtt{p}$ and $\mathtt{q}$, and mapping respectively to session types S and T;
(2) further *project* S and T onto the "opposite" role of the pair (Definition A.4) — i.e., project S onto $\mathtt{q}$ and T onto $\mathtt{p}$. The outcome of such projections are *partial session types* (Definition A.2), similar to *binary* session types, and only describing the interactions between $\mathtt{p}$ and $\mathtt{q}$;
(3) check that such projected partial types are *dual* (Definition A.3) — modulo *partial session subtyping* $\leqslant$ (Definition A.5).

These complications have a payoff: consistency ensures that all possible interactions between pairs of roles in a multiparty session are "correct," and avoids situations like Example 3.2 above; in fact, the typing contexts in (10) and (11) are *not* consistent, because their partial type projections yield non-dual input/output message labels or payloads, violating Definition 3.3.

3.3. Limitations of consistency

Unfortunately, the consistency-based approach outlined above has a drawback: it is very restrictive, and does not hold for many "intuitively correct" processes — especially when they implement protocols with inter-role dependencies. For instance, the processes in Example 2.3 are "intuitively correct", since they interact smoothly until they terminate, as shown in Example 2.5. However, as shown in (8), they are only typable under a typing context $\Delta = s[\mathtt{alice}]:S_\mathtt{a}, s[\mathtt{bob}]:S_\mathtt{b}, s[\mathtt{carol}]:S_\mathtt{c}$ that is *not* consistent, as explained in Example 3.4 below.

Example 3.4. Take $S_\mathtt{a}$, $S_\mathtt{b}$, $S_\mathtt{c}$ from Example 2.8, and $\Delta = s[\mathtt{alice}]:S_\mathtt{a}, s[\mathtt{bob}]:S_\mathtt{b}, s[\mathtt{carol}]:S_\mathtt{c}$. By Definition 3.3, in order to check whether Δ is consistent we need to compute the partial projections of all types (using Definition A.4), and check their duality. For example, we have the following partial projections:

$$S_\mathtt{a} \upharpoonright \mathtt{bob} = \oplus\{\mathtt{m1}(\mathsf{Int})\,,\ \mathtt{stop}\} \qquad S_\mathtt{b} \upharpoonright \mathtt{alice} = \&\{\mathtt{m1}(\mathsf{Int})\,,\ \mathtt{stop}\}$$

and we also get $S_\mathtt{a} \upharpoonright \mathtt{bob} = \overline{S_\mathtt{b} \upharpoonright \mathtt{alice}}$ — i.e., the projections are dual (Definition A.3), because the inputs/outputs of one match the outputs/inputs of the other. This means that, by Definition 3.3, the pair $s[\mathtt{alice}]:S_\mathtt{a}, s[\mathtt{bob}]:S_\mathtt{b}$ is consistent.

However, the following projections (also required by Definition 3.3) are *not* defined:

$$\begin{array}{rcl} S_\mathtt{a} \upharpoonright \mathtt{carol} & = & \mathtt{carol}\&\mathtt{m3}(\mathsf{Bool}) \upharpoonright \mathtt{carol} \sqcap \mathbf{end} \upharpoonright \mathtt{carol} \;=\; \&\mathtt{m3}(\mathsf{Bool}) \sqcap \mathbf{end} \quad \text{(undefined)} \\ S_\mathtt{c} \upharpoonright \mathtt{alice} & = & \mathtt{alice} \oplus \mathtt{m3}(\mathsf{Bool}) \upharpoonright \mathtt{alice} \sqcap \mathbf{end} \upharpoonright \mathtt{alice} \;=\; \oplus\mathtt{m3}(\mathsf{Bool}) \sqcap \mathbf{end} \quad \text{(undefined)} \end{array} \tag{13}$$

because (by Definition A.4) they use the *partial types merging operator* $\sqcap$ in a case where it is not defined.

Therefore, the pair $s[\mathtt{alice}]:S_\mathtt{a}, s[\mathtt{carol}]:S_\mathtt{c}$ is *not* consistent; hence, by Definition 3.3, we conclude that the typing context Δ is not consistent.

In Example 3.4, the partial projection $S_\mathtt{a} \upharpoonright \mathtt{carol}$ (resp. $S_\mathtt{c} \upharpoonright \mathtt{alice}$) is undefined because the session type $S_\mathtt{a}$ (resp. $S_\mathtt{c}$) starts with an interaction with $\mathtt{bob}$; as a consequence, in order to project the type onto $\mathtt{carol}$ (resp. $\mathtt{alice}$), it is necessary to:

(1) "skip" the interaction with $\mathtt{bob}$,
(2) take all possible continuations and project each one onto $\mathtt{carol}$ (resp. $\mathtt{alice}$), and
(3) *merge* such projected continuations, with the operator $\sqcap$ (Definition A.4) that tries to combine them into a single partial type.

The merge operator between partial types is quite restrictive: it does *not* allow to combine a branching (resp. selection) type with **end**, as shown in (13). This means that the interaction between $\mathtt{alice}$ (resp. $\mathtt{carol}$) and $\mathtt{bob}$ can only have a limited influence on the interactions between $\mathtt{alice}$ (resp. $\mathtt{carol}$) and other roles. This curtails the variety of inter-role dependencies supported by the "classic" MPST theory.

The consequence of this limitation is that the subject reduction statement in (12) is vacuously true for the typed process $P = P_\mathtt{a} \mid P_\mathtt{b} \mid P_\mathtt{c}$ from Example 2.3. Therefore, we cannot use (12) to formally prove that $P_\mathtt{a} \mid P_\mathtt{b} \mid P_\mathtt{c}$ reduce in a type-safe way. Further limitations arise for recursive protocols: we illustrate them in Section 3.4 below.

3.4. More "correct" but non-consistent examples

We now describe more cases of "correct" choreographies that, when projected, yield typing contexts that are *not consistent* according to Definition 3.3. As a consequence, the "classic" MPST theory does *not* provide subject reduction guarantees for processes that implement such choreographies.

Note that all the following examples will be supported by our theory, presented in Section 4: in fact, they all yield *live* typing contexts (Definition 4.1) that can type processes enjoying subject reduction (Theorem 4.9).

3.4.1. Consistency vs. history-dependent branching

In Example 3.4, consistency does *not* hold because the partial projections $S_{\mathtt{a}}\upharpoonright\mathtt{carol}$ and $S_{\mathtt{c}}\upharpoonright\mathtt{alice}$ are undefined, due to the non-mergeability of internal/external choice and **end** (as per Definition A.4). We now try to "adjust" G in (7) and make it consistent, by adding another communication between carol and alice:

$$G' = \mathtt{alice}\to\mathtt{bob}:\begin{Bmatrix}\mathtt{m1}(\mathsf{Int})\,.\,\mathtt{bob}\to\mathtt{carol}:\mathtt{m2}(\mathsf{Str})\,.\,\mathtt{carol}\to\mathtt{alice}:\mathtt{m3}(\mathsf{Bool})\,.\,\mathbf{end}\,,\\ \mathtt{stop}\,.\,\mathtt{bob}\to\mathtt{carol}:\mathtt{quit}\,.\,\mathtt{carol}\to\mathtt{alice}:\mathtt{m4}(\mathsf{Int})\,.\,\mathbf{end}\end{Bmatrix}$$

The projections of G' onto alice, bob and carol are respectively:

$$S'_{\mathtt{a}} = \mathtt{bob}\oplus\begin{Bmatrix}\mathtt{m1}(\mathsf{Int})\,.\,\mathtt{carol}\,\&\,\mathtt{m3}(\mathsf{Bool})\,,\\ \mathtt{stop}\,.\,\mathtt{carol}\,\&\,\mathtt{m4}(\mathsf{Int})\end{Bmatrix}\qquad S'_{\mathtt{b}} = \mathtt{alice}\,\&\begin{Bmatrix}\mathtt{m1}(\mathsf{Int})\,.\,\mathtt{carol}\oplus\mathtt{m2}(\mathsf{Str})\,,\\ \mathtt{stop}\,.\,\mathtt{carol}\oplus\mathtt{quit}\end{Bmatrix}$$

$$S'_{\mathtt{c}} = \mathtt{bob}\,\&\begin{Bmatrix}\mathtt{m2}(\mathsf{Str})\,.\,\mathtt{alice}\oplus\mathtt{m3}(\mathsf{Bool})\,,\\ \mathtt{quit}\,.\,\mathtt{alice}\oplus\mathtt{m4}(\mathsf{Int})\end{Bmatrix}$$

Let us now examine the partial projections $S'_{\mathtt{a}}\upharpoonright\mathtt{carol}$ and $S'_{\mathtt{c}}\upharpoonright\mathtt{alice}$:

$$\begin{array}{lcll} S'_{\mathtt{a}}\upharpoonright\mathtt{carol} & = & \mathtt{carol}\,\&\,\mathtt{m3}(\mathsf{Bool})\upharpoonright\mathtt{carol}\sqcap\mathtt{carol}\,\&\,\mathtt{m4}(\mathsf{Int})\upharpoonright\mathtt{carol} & = \quad \&\,\mathtt{m3}(\mathsf{Bool})\sqcap\&\,\mathtt{m4}(\mathsf{Int}) \qquad \text{(undefined)}\\ S'_{\mathtt{c}}\upharpoonright\mathtt{alice} & = & \mathtt{alice}\oplus\mathtt{m3}(\mathsf{Bool})\upharpoonright\mathtt{alice}\sqcap\mathtt{alice}\oplus\mathtt{m4}(\mathsf{Int})\upharpoonright\mathtt{alice} & = \quad \oplus\mathtt{m3}(\mathsf{Bool})\sqcap\oplus\mathtt{m4}(\mathsf{Int}) \;=\; \oplus\{\mathtt{m3}(\mathsf{Bool})\,,\,\mathtt{m4}(\mathsf{Int})\}\end{array}$$

Unfortunately, partial type merging does not allow to combine different external choices, thus making $S'_{\mathtt{a}}\upharpoonright\mathtt{carol}$ undefined. Hence, external choices cannot depend on the outcome of previous interactions with other roles: i.e., alice cannot wait for different messages from carol, depending on the interaction between alice and bob. Still, different *internal* choices can be merged: e.g., carol can send a different message to alice, depending on the previous interaction between carol and bob: for this reason, $S'_{\mathtt{c}}\upharpoonright\mathtt{alice}$ is defined.

3.4.2. Consistency vs. recursion

Consider the following global type:

$$G'' = \mu\mathbf{t}.\mathtt{alice}\to\mathtt{bob}:\begin{Bmatrix}\mathtt{m1}(\mathsf{Int})\,.\,\mathtt{bob}\to\mathtt{carol}:\mathtt{m2}(\mathsf{Str})\,.\,\mathtt{carol}\to\mathtt{alice}:\mathtt{m3}(\mathsf{Bool})\,.\,\mathbf{t}\,,\\ \mathtt{stop}\,.\,\mathtt{bob}\to\mathtt{carol}:\mathtt{quit}\,.\,\mathtt{carol}\to\mathtt{alice}:\mathtt{m3}(\mathsf{Bool})\,.\,\mathbf{end}\end{Bmatrix}$$

The global type G'' above is again similar to G in (7) — but now, if alice sends m1(Int) to bob, the choreography loops: in fact, bob then sends m2(Str) to carol, which sends m3(Bool) to alice, that continues recursively. Instead, if alice chooses to send stop to bob, the latter notifies quit to carol, which sends m3(Bool) to alice, and the choreography **end**s. The projections of G'' onto alice, bob and carol are respectively:

$$S''_{\mathtt{a}} = \mu\mathbf{t}.\mathtt{bob}\oplus\begin{Bmatrix}\mathtt{m1}(\mathsf{Int}).\mathtt{carol}\,\&\,\mathtt{m3}(\mathsf{Bool}).\mathbf{t}\,,\\ \mathtt{stop}.\mathtt{carol}\,\&\,\mathtt{m3}(\mathsf{Bool}).\mathbf{end}\end{Bmatrix}\qquad S''_{\mathtt{b}} = \mu\mathbf{t}.\mathtt{alice}\,\&\begin{Bmatrix}\mathtt{m1}(\mathsf{Int}).\mathtt{carol}\oplus\mathtt{m2}(\mathsf{Str}).\mathbf{t}\,,\\ \mathtt{stop}.\mathtt{carol}\oplus\mathtt{quit}.\mathbf{end}\end{Bmatrix}$$

$$S''_{\mathtt{c}} = \mu\mathbf{t}.\mathtt{bob}\,\&\begin{Bmatrix}\mathtt{m2}(\mathsf{Str}).\mathtt{alice}\oplus\mathtt{m3}(\mathsf{Bool}).\mathbf{t}\,,\\ \mathtt{quit}.\mathtt{alice}\oplus\mathtt{m3}(\mathsf{Bool}).\mathbf{end}\end{Bmatrix}$$

Note that in all cases, carol sends to alice the message m3(Bool): this circumvents the merging problems described in Example 3.4 and Section 3.4.1. However, we stumble into another roadblock:

$$\begin{array}{lcll} S''_{\mathtt{a}}\upharpoonright\mathtt{carol} & = & \mu\mathbf{t}.((\mathtt{carol}\,\&\,\mathtt{m3}(\mathsf{Bool})\,.\,\mathbf{t})\upharpoonright\mathtt{carol}\sqcap(\mathtt{carol}\,\&\,\mathtt{m3}(\mathsf{Bool})\,.\,\mathbf{end})\upharpoonright\mathtt{carol}) & \\ & = & \mu\mathbf{t}.(\&\,\mathtt{m3}(\mathsf{Bool})\,.\,(\mathbf{t}\sqcap\mathbf{end})) & \text{(undefined)}\end{array}$$

$$\begin{array}{lcll} S''_{\mathtt{c}}\upharpoonright\mathtt{alice} & = & \mu\mathbf{t}.((\mathtt{alice}\oplus\mathtt{m3}(\mathsf{Bool})\,.\,\mathbf{t})\upharpoonright\mathtt{alice}\sqcap(\mathtt{alice}\oplus\mathtt{m3}(\mathsf{Bool})\,.\,\mathbf{end})\upharpoonright\mathtt{alice}) & \\ & = & \mu\mathbf{t}.(\oplus\mathtt{m3}(\mathsf{Bool})\,.\,(\mathbf{t}\sqcap\mathbf{end})) & \text{(undefined)}\end{array}$$

This is another case of (unsupported) inter-role dependency: in $S''_{\mathtt{a}}$ (resp. $S''_{\mathtt{c}}$), the interaction with bob determines whether, after receiving (resp. sending) m3(Bool) from carol (resp. to alice), the type will *(a)* loop on **t**, and thus, keep interacting with carol (resp. alice), or *(b)* **end** the session. Hence, the partial projection $S''_{\mathtt{a}}\upharpoonright\mathtt{carol}$ (resp. $S''_{\mathtt{c}}\upharpoonright\mathtt{alice}$) tries to merge **t** (from case *(a)*) with **end** (from case *(b)*) — but the operation is undefined.

4. Multiparty session typing, beyond duality and consistency

We now introduce our new multiparty session typing system. Following the discussion in Section 3, our goal is to replace consistency with a more flexible typing context property that avoids the issues in Example 3.2, but supports protocols with complex inter-role dependencies. Our key idea is to avoid the "loss of contextual information" discussed in Section 1, by introducing a new *rely/guarantee* typing judgement.

In Section 4.1, we define a behavioural *liveness* invariant used by our novel typing system, that is presented in Section 4.2; then, in Section 4.3, we formalise and discuss its type safety guarantees.

4.1. Liveness

In order to replace consistency, we need an alternative typing context property that still allows to prove subject reduction. Technically, this means that the new property must satisfy the following requirements:

(R1) it must ensure that the typing context is "safe" (e.g., if the type of $s'[\mathtt{r}]$ sends a message to $\mathtt{r}'$, then the type of $s'[\mathtt{r}']$ can input such a message from $\mathtt{r}$), thus avoiding cases like Example 3.2;
(R2) it must be preserved when typed processes communicate and reduce; and
(R3) it must be preserved when typing contexts are (de)composed by parallel composition (see [STD-|] in (8)).

To address requirements **(R1)** and **(R2)**, we introduce a *co*inductive notion of *typing context liveness* (Definition 4.1). It is based on the reductions of typing contexts (Definition 2.10), and ensures that each selection is matched by a compatible branching, and each branching can be triggered by a compatible selection. We will address requirement **(R3)** later on, with our liveness-based typing system (Section 4.2).

Definition 4.1 *(Liveness).* The predicate $\mathrm{live}(\Delta)$ (read "Δ *is live*") is the largest predicate such that:

- [L-&] $\mathrm{live}(\Delta, s[\mathtt{p}]{:}S)$ with $S = \mathtt{q}\&_{i\in I}\mathtt{m}_i(S_i).S_i'$ implies $\exists i \in I : \exists \Delta' : \Delta, s[\mathtt{p}]{:}S \twoheadrightarrow^* \Delta', s[\mathtt{p}]{:}S_i'$;
- [L-⊕] $\mathrm{live}(\Delta, s[\mathtt{p}]{:}S)$ with $S = \mathtt{q}\oplus_{i\in I}\mathtt{m}_i(S_i).S_i'$ implies $\forall i \in I : \exists \Delta' : \Delta, s[\mathtt{p}]{:}S \twoheadrightarrow^* \Delta', s[\mathtt{p}]{:}S_i'$;
- [L-μ] $\mathrm{live}(\Delta, s[\mathtt{p}]{:}\mu\mathbf{t}.S)$ implies $\mathrm{live}(\Delta, s[\mathtt{p}]{:}S\{{}^{\mu\mathbf{t}.S}/_{\mathbf{t}}\})$;
- [L-→] $\mathrm{live}(\Delta)$ and $\Delta \to \Delta'$ implies $\mathrm{live}(\Delta')$.

The crucial difference between liveness and consistency (Definition 3.3) is that the former is a *behavioural* property, and does not depend on syntactic projections and duality checks. If a typing context $\Delta, s[\mathtt{p}]{:}S$ is live, then:

- [L-&] says that if S is an external choice, then Δ must be able to reduce until *some* branch of S is triggered;
- [L-⊕] says that if S is an internal choice, then Δ must be able to reduce allowing to send *each* message of S;
- [L-μ] says that if S is recursive, then its unfolding must be live, too. This means that liveness inspects all step-by-step unfoldings of S, until the first non-recursive subterm is exposed,[3] and thus checked by clause [L-⊕]/[L-&];
- [L-→] says that if the typing context reduces, the reduct must be live, too.

Example 4.2. Consider the typing context Δ_μ from Example 2.11. We have $\mathrm{live}(\Delta_\mu)$. In fact, consider the following predicate, containing Δ_μ and its unfoldings:

$$\mathbb{P} = \left\{ \begin{array}{l} \Delta_\mu\,, \\ (s[\mathtt{p}]{:}\mathtt{q}\oplus\mathtt{m}.\mu\mathbf{t}.\mathtt{q}\oplus\mathtt{m}.\mathbf{t}\,,\, s[\mathtt{q}]{:}\mu\mathbf{t}.\mathtt{p}\&\mathtt{m}.\mathbf{t})\,, \\ (s[\mathtt{p}]{:}\mu\mathbf{t}.\mathtt{q}\oplus\mathtt{m}.\mathbf{t}\,,\, s[\mathtt{q}]{:}\mathtt{p}\&\mathtt{m}.\mu\mathbf{t}.\mathtt{p}\&\mathtt{m}.\mathbf{t})\,, \\ (s[\mathtt{p}]{:}\mathtt{q}\oplus\mu\mathbf{t}.\mathtt{q}\oplus\mathtt{m}.\mathbf{t}\,,\, s[\mathtt{q}]{:}\mathtt{p}\&\mathtt{m}.\mu\mathbf{t}.\mathtt{p}\&\mathtt{m}.\mathbf{t}) \end{array} \right\}$$

By inspecting each $\Delta \in \mathbb{P}$, we can verify that it satisfies the coinductive clauses of Definition 4.1:

- each $\Delta \in \mathbb{P}$ reduces to Δ_μ, that belongs to $\mathbb{P}$ — thus satisfying clause [L-→];
- if $\Delta = \Delta', s[\mathtt{r}]{:}\mu\mathbf{t}.S$ (for some $\mathtt{r}$), then $\Delta', s[\mathtt{r}]{:}S\{{}^{\mu\mathbf{t}.S}/_{\mathbf{t}}\}$ belongs to $\mathbb{P}$ — thus satisfying clause [L-μ];
- if some type in Δ exposes an internal/external choice, then it can be eventually triggered by the rest of the context (see Example 2.11) — thus satisfying clause [L-⊕]/[L-&].

Now, by Definition 4.1, live is the largest predicate satisfying such clauses, and therefore we have that $\Delta \in \mathbb{P}$ implies $\mathrm{live}(\Delta)$. Then, since $\Delta_\mu \in \mathbb{P}$, we conclude $\mathrm{live}(\Delta_\mu)$.

[3] Recall that session types are contractive, by Definition 2.6: hence, the unfolding of a sequence of nested recursions $\mu\mathbf{t}_1.\mu\mathbf{t}_2\ldots$ must eventually yield a type with a top-level internal/external choice, or **end**.

$$\frac{\forall c \in \mathrm{dom}(\Delta) : \Delta(c) \equiv \mathbf{end}}{\mathrm{end}(\Delta)}\ [\text{T-End}] \qquad \frac{\Delta \equiv c{:}S}{\Delta \vdash c{:}S}\ [\text{T-Chan}] \qquad \frac{\Theta(X) \equiv S_1, \ldots, S_n}{\Theta \vdash X : S_1, \ldots, S_n}\ [\text{T-Proc}] \qquad \frac{\mathrm{end}(\Delta)}{\Theta \vdash \mathbf{0} \triangleright \Delta \triangleleft \Gamma}\ [\text{T-}\mathbf{0}]$$

$$\frac{\Theta, X{:}S_1, \ldots, S_n \vdash P \triangleright x_1{:}S_1, \ldots, x_n{:}S_n \triangleleft \emptyset \quad \Theta, X{:}S_1, \ldots, S_n \vdash Q \triangleright \Delta \triangleleft \Gamma}{\Theta \vdash \mathbf{def}\ X(x_1, \ldots, x_n) = P\ \mathbf{in}\ Q \triangleright \Delta \triangleleft \Gamma}\ [\text{T-}\mathbf{def}] \qquad \frac{\Theta \vdash X : S_1, \ldots, S_n \quad \mathrm{end}(\Delta_0) \quad \forall i \in 1..n \quad \Delta_i \vdash c_i{:}S_i}{\Theta \vdash X\langle c_1, \ldots, c_n\rangle \triangleright \Delta_0, \Delta_1, \ldots, \Delta_n \triangleleft \Gamma}\ [\text{T-}X]$$

$$\frac{\Delta_s = \{s[\mathtt{p}]{:}S_\mathtt{p}\}_{\mathtt{p}\in I} \qquad \Theta \vdash P \triangleright \Delta, \Delta_s \triangleleft \Gamma}{\Theta \vdash (\nu s{:}\Delta_s)\, P \triangleright \Delta \triangleleft \Gamma}\ [\text{T-}\nu] \qquad \frac{\Theta \vdash P_1 \triangleright \Delta_1 \triangleleft \Gamma, \Delta_2 \qquad \Theta \vdash P_2 \triangleright \Delta_2 \triangleleft \Gamma, \Delta_1}{\Theta \vdash P_1 \mid P_2 \triangleright \Delta_1, \Delta_2 \triangleleft \Gamma}\ [\text{T-}|]$$

$$\frac{S \equiv \mathtt{q}\&_{i\in I}\mathtt{m}_i(S_i).S'_i \qquad \forall i \in I \qquad \forall \Gamma', \Gamma'' : \Gamma \twoheadrightarrow^* \Gamma' \text{ and } \Gamma', c{:}S \to \Gamma'', c{:}S'_i \qquad \Theta \vdash P_i \triangleright \Delta, y_i{:}S_i, c{:}S'_i \triangleleft \Gamma''}{\Theta \vdash c[\mathtt{q}] \textstyle\sum_{i\in I\cup J} \{\mathtt{m}_i(y_i).P_i\} \triangleright \Delta, c{:}S \triangleleft \Gamma}\ [\text{T-}\&]$$

$$\frac{S \equiv \mathtt{q}\oplus_{i\in I}\mathtt{m}_i(S_i).S'_i \qquad \exists k \in I \qquad \Delta_d \vdash d{:}S_k \qquad \forall \Gamma', \Gamma'' : \Gamma \twoheadrightarrow^* \Gamma' \text{ and } \Gamma', c{:}S \to \Gamma'', c{:}S'_k \qquad \Theta \vdash P \triangleright \Delta, c{:}S'_k \triangleleft \Gamma'', \Delta_d}{\Theta \vdash c[\mathtt{q}]\oplus\mathtt{m}_k\langle d\rangle.P \triangleright \Delta_d, \Delta, c{:}S \triangleleft \Gamma}\ [\text{T-}\oplus]$$

Fig. 3. Rely/guarantee multiparty session typing system.

Example 4.3. Consider $\Delta_{\mathtt{abc}}$ from Example 2.12, and the following predicate, containing $\Delta_{\mathtt{abc}}$ and all its reducts:

$$\mathbb{P} = \left\{\Delta' \mid \Delta_{\mathtt{abc}} \twoheadrightarrow^* \Delta'\right\}$$

We can verify that each element of $\mathbb{P}$ satisfies clauses [L-$\to$], [L-&], [L-$\oplus$] and [L-μ] of Definition 4.1 (possibly vacuously). Hence, similarly to Example 4.2, we conclude live($\Delta_{\mathtt{abc}}$).

Notably, a live typing context never deadlocks: i.e., if it cannot further reduce, then all its entries must be **end**-typed. The *vice versa* is not true: a deadlock-free typing context might not be live. Moreover, liveness and consistency are incomparable, i.e., neither implies the other. This is shown in Example 4.4 below.

Example 4.4. The typing context $s[\mathtt{p}]{:}S_\mathtt{a}, s[\mathtt{q}]{:}S_\mathtt{b}, s[\mathtt{r}]{:}S_\mathtt{c}$ in (8) is live, but not consistent.

The context $s[\mathtt{p}]{:}\mathtt{q}\&\mathtt{m}_1.\mathtt{r}\oplus\mathtt{m}_3$, $s[\mathtt{q}]{:}\mathtt{r}\&\mathtt{m}_2.\mathtt{p}\oplus\mathtt{m}_1$, $s[\mathtt{r}]{:}\mathtt{p}\&\mathtt{m}_3.\mathtt{q}\oplus\mathtt{m}_2$ is consistent but not live (nor deadlock-free): its inputs/outputs occur in the wrong order, thus preventing reductions.

Finally, the typing context $s[\mathtt{p}]{:}\mu\mathbf{t}.\mathtt{q}\oplus\mathtt{m}_1.\mathbf{t}$, $s[\mathtt{q}]{:}\mu\mathbf{t}.\mathtt{p}\&\mathtt{m}_1.\mathbf{t}$, $s[\mathtt{r}]{:}\mathtt{p}\&\mathtt{m}_2$ is deadlock-free and consistent, but not live: $s[\mathtt{p}]$ and $s[\mathtt{q}]$ generate infinite reductions, but $s[\mathtt{r}]$ cannot possibly perform its input (thus violating clause [L-&] of Definition 4.1).

4.2. A rely/guarantee typing system for multiparty sessions

We can now introduce our new typing system.

Definition 4.5 *(Rely/guarantee typing system).* Let Θ be a partial mapping defined as:

$$\Theta \ ::= \ \emptyset \mid \Theta, X{:}\widetilde{S}$$

A *rely/guarantee multiparty session typing judgement* has the form:

$$\Theta \vdash P \triangleright \Delta \triangleleft \Gamma \qquad \text{with live}(\Delta, \Gamma)$$

and is inductively defined by the rules in Fig. 3. Above, Δ and Γ are called respectively *guarantee* and *rely contexts*.

In Definition 4.5, similarly to most MPST works, Θ assigns an n-uple of types to each process variable X (one type per argument). Intuitively, the judgement $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma$ says: given the process definitions typed in Θ, P "guarantees" to use its channels *linearly* according to Δ, by "relying" on the assumption that other processes use their channels according to Γ. The additional Γ is the most visible departure from standard MPST typing rules (although we subsume them under some conditions, as we will show in Section 5.3). Note that the requirement live(Δ, Γ) implies that Δ, Γ must be defined: therefore, by Definition 2.9, we must have $\mathrm{dom}(\Delta) \cap \mathrm{dom}(\Gamma) = \emptyset$.

In Fig. 3, the first three rules define auxiliary judgements:

- [T-End] says that end(Δ) holds if all entries of Δ are **end**-typed;
- [T-Chan] says that the judgement $\Delta \vdash c{:}S$ (read *"c has type S in Δ"*) holds if Δ contains exactly one entry $c{:}S'$, with $S' \equiv S$. Note that this judgement allows to consider S' up-to unfolding: this makes the typing system *equi-recursive* [31, Chapter 21];
- [T-Proc] says that the judgement $\Theta \vdash X : S_1, \ldots, S_n$ (read *"process X has parameter types $S_1, \ldots, S_n$ in Θ"*) holds if Θ maps X to an n-uple of types equivalent to $S_1, \ldots, S_n$ (again, up-to unfolding).

The remaining typing rules are the core of the typing system:

- [T-**0**] says that **0** is typed if all channels in Δ are **end**-typed;
- [T-**def**] says that **def** $X\langle\widetilde{x}\rangle = P$ **in** Q is typed if P uses the arguments $x_1, \ldots, x_n$ according to $S_1, \ldots, S_n$ (with empty rely context), and the latter n-uple types the parameters of X when typing Q;
- [T-X] says that $X\langle\widetilde{c}\rangle$ is typed if the types of $\widetilde{c}$ match those of the formal parameters of X, and any unused channel (in Δ_0) is **end**-typed;
- [T-ν] says that $(\nu s)\, P$ is typed if P can be typed by adding Δ_s (that annotates s, and only has s-based entries) to Δ. Note that by Definition 4.5, we have live(Δ, Γ) in the conclusion of the rule, and $\Theta \vdash P \triangleright \Delta, \Delta_s \triangleleft \Gamma$ in its premise: as a consequence, the rule also implies live(Δ_s) (Lemma C.4);
- [T-|] says that $P_1 \mid P_2$ is typed by splitting the guarantee context in the premises (similarly to [Std-|] in (8)) — and crucially, by recording the "lost" information in the rely context. Hence, P_1 is typed by relying on the guarantees of P_2, and *vice versa*: this allows the typing rules to preserve liveness across a typing derivation;
- [T-&] says that $c[\mathtt{q}] \sum_{i \in I \cup J} \{\mathtt{m}_i(y_i).P_i\}$ is typed if c has type S, where S is a (possibly recursive) external choice from $\mathtt{q}$, with message labels $\mathtt{m}_i$ (where $i \in I$). As usual in MPST works, the process can have more branches than type S: i.e., it can support additional message labels $\mathtt{m}_j$ with $j \in J$, and they are immaterial for typing. Furthermore, the rule checks that if Γ reduces to Γ'' while moving $c{:}S$ to $c{:}S'_i$ (for any $i \in I$), then $c{:}S'_i$ and Γ'' must type the continuation P_i, with the received channel $y_i{:}S_i$ added to the guarantee context — i.e., P_i must use it correctly;
- [T-$\oplus$] says that $c[\mathtt{q}] \oplus \mathtt{m}_k\langle d\rangle.P$ is typed if c has type S, where S is a (possibly recursive) internal choice towards $\mathtt{q}$, whose message labels include $\mathtt{m}_k$. Furthermore, if Γ reduces to Γ'' while moving $c{:}S$ to $c{:}S'_k$, then $c{:}S'_k$ and Γ'' must type the continuation P — but with the sent channel $d{:}S_k$ now transferred into the rely context: i.e., after sending d, P cannot use it, but relies on its correct usage by the recipient.

Remark 4.6. The number of premises of the typing rules [T-&] and [T-$\oplus$] is always finite: this is because, by Definition 2.10, the relation $\to$ induces finite-state transition systems, and thus, quantifications like "$\forall \Gamma' : \Gamma \twoheadrightarrow^* \Gamma' \ldots$" can yield only a finite set of reducts — and correspondingly, a finite number of premises to check. We will revisit this topic in Section 5.3.

Example 4.7. We now revise the typing derivation (8) in Section 3 using our new rely/guarantee typing rules from Definition 4.5 (here, we abbreviate `alice`, `bob`, and `carol` respectively as `a`, `b`, and `c`):

$$\dfrac{\dfrac{\dfrac{\mathbb{D}\left\{\dfrac{\vdots}{\emptyset \vdash P_{\mathtt{a}} \triangleright s[\mathtt{a}]{:}S_{\mathtt{a}} \triangleleft s[\mathtt{b}]{:}S_{\mathtt{b}}, s[\mathtt{c}]{:}S_{\mathtt{c}}} \quad \dfrac{\vdots}{\emptyset \vdash P_{\mathtt{b}} \triangleright s[\mathtt{b}]{:}S_{\mathtt{b}} \triangleleft s[\mathtt{a}]{:}S_{\mathtt{a}}, s[\mathtt{c}]{:}S_{\mathtt{c}}}\right.}{\emptyset \vdash P_{\mathtt{a}} \mid P_{\mathtt{b}} \triangleright s[\mathtt{a}]{:}S_{\mathtt{a}}, s[\mathtt{b}]{:}S_{\mathtt{b}} \triangleleft s[\mathtt{c}]{:}S_{\mathtt{c}}}\,[\text{T-|}] \quad \dfrac{\vdots}{\emptyset \vdash P_{\mathtt{c}} \triangleright s[\mathtt{c}]{:}S_{\mathtt{c}} \triangleleft s[\mathtt{a}]{:}S_{\mathtt{a}}, s[\mathtt{b}]{:}S_{\mathtt{b}}}\,[\text{T-|}]}{\Delta = s[\mathtt{a}]{:}S_{\mathtt{a}}, s[\mathtt{b}]{:}S_{\mathtt{b}}, s[\mathtt{c}]{:}S_{\mathtt{c}} \qquad \emptyset \vdash P_{\mathtt{a}} \mid P_{\mathtt{b}} \mid P_{\mathtt{c}} \triangleright s[\mathtt{a}]{:}S_{\mathtt{a}}, s[\mathtt{b}]{:}S_{\mathtt{b}}, s[\mathtt{c}]{:}S_{\mathtt{c}} \triangleleft \emptyset}\,[\text{T-}\nu]}{\emptyset \vdash (\nu s{:}\Delta)(P_{\mathtt{a}} \mid P_{\mathtt{b}} \mid P_{\mathtt{c}}) \triangleright \emptyset \triangleleft \emptyset}$$

We have live(Δ) (see Example 4.3), and each application of [T-|] propagates the entries of Δ in its premises' guarantee/rely contexts, throughout the derivation. Now, consider the sub-derivation $\mathbb{D}$ (top left):

$$\dfrac{\mathbb{D}\left\{\dfrac{S_{\mathtt{a}} \equiv \mathtt{b} \oplus \left\{\begin{array}{l}\mathtt{m1(Int).\,c\&m3(Bool)},\\ \mathtt{stop}.\,\mathbf{end}\end{array}\right\} \qquad \begin{array}{l}s[\mathtt{b}]{:}S_{\mathtt{b}}, s[\mathtt{c}]{:}S_{\mathtt{c}},\\ s[\mathtt{a}]{:}S_{\mathtt{a}}\end{array} \to \begin{array}{l}s[\mathtt{b}]{:}\mathtt{c}\oplus\mathtt{quit}, s[\mathtt{c}]{:}S_{\mathtt{c}},\\ s[\mathtt{a}]{:}\mathbf{end}\end{array} \qquad \dfrac{\text{end}(s[\mathtt{a}]{:}\mathbf{end})}{\emptyset \vdash \mathbf{0} \triangleright s[\mathtt{a}]{:}\mathbf{end} \triangleleft s[\mathtt{b}]{:}\mathtt{c}\oplus\mathtt{quit}, s[\mathtt{c}]{:}S_{\mathtt{c}}}\,[\text{T-}\mathbf{0}]}{\emptyset \vdash s[\mathtt{a}][\mathtt{b}]\oplus\mathtt{stop}.\mathbf{0} \triangleright s[\mathtt{a}]{:}S_{\mathtt{a}} \triangleleft s[\mathtt{b}]{:}S_{\mathtt{b}}, s[\mathtt{c}]{:}S_{\mathtt{c}}}\,[\text{T-}\oplus]\right.}$$

Here, the rely context $s[\mathtt{b}]{:}S_{\mathtt{b}}, s[\mathtt{c}]{:}S_{\mathtt{c}}$ cannot reduce (i.e., in [T-$\oplus$] in Fig. 3, we have $\Gamma' = \Gamma$). By composing it with $s[\mathtt{a}]{:}S_{\mathtt{a}}$, and following the `stop` reduction as in Example 2.12 (thus matching the output `stop` in the process), we get the new rely context $s[\mathtt{b}]{:}\mathtt{c}\oplus\mathtt{quit}, s[\mathtt{c}]{:}S_{\mathtt{c}}$: by [T-$\oplus$], it must type the continuation **0**. Observe that in the instance of [T-**0**], the rely context *never interacts with* $s[\mathtt{a}]$, although `a` occurs in $S_{\mathtt{c}}$: this captures the inter-role dependency discussed in Sections 1 and 3.

4.3. Subject reduction

Our typing rules satisfy the *Substitution Lemma 4.8* below: if P is typed with an S-typed variable x, and its rely context Γ contains an S-typed channel with role $s[\mathtt{p}]$, then $P\{^{s[\mathtt{p}]}/_x\}$ can be typed by *(1)* removing the entry for $s[\mathtt{p}]$ from the rely context, and *(2)* using it in place of x in the guarantee context. Crucially, by moving $s[\mathtt{p}]$ between the two contexts, we preserve the liveness before and after the substitution (as required by Definition 4.5).

Lemma 4.8 *(Substitution). If* $\Theta \vdash P \triangleright \Delta, x{:}S \triangleleft \Gamma, \Gamma'$ *and* $\Gamma \vdash s[\mathtt{p}]{:}S$, *then* $\Theta \vdash P\{^{s[\mathtt{p}]}/_x\} \triangleright \Delta, s[\mathtt{p}]{:}S \triangleleft \Gamma'$.

Proof. See Appendix E. □

Using Lemma 4.8, we prove the *Subject Reduction Theorem* 4.9 below. Note that its statement is similar to the (tentative) statement (9) in Section 3, but without the counterexamples shown in Example 3.2: this is because in Definition 4.5, we "embed" the liveness requirement in the typing judgement, so that Theorem 4.9 holds for *all* typed processes (as one might usually expect).

Theorem 4.9 *(Subject reduction). If* $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma$ *and* $P \twoheadrightarrow^* P'$, *then* $\exists \Delta'$ *such that* $\Delta \twoheadrightarrow^* \Delta'$ *and* $\Theta \vdash P' \triangleright \Delta' \triangleleft \Gamma$.

Proof. See Appendix F. The proof crucially exploits the fact that, in our rely/guarantee typing system, liveness satisfies the requirements **(R1)**–**(R3)** outlined in Section 4.1. □

5. Comparison: rely/guarantee vs. "classic" multiparty session typing

In this section, we compare our new rely/guarantee multiparty session typing system with the "classic" MPST theory. In Example 4.4, we have shown that liveness and consistency are incomparable; therefore:

1. on the one hand, using our theory we can prove type safety in cases that are not covered in "classic" MPST, as shown in Example 4.7;
2. but on the other hand, the "classic" MPST subject reduction statement in (12) allows to prove type safety for processes that would be rejected by our theory, because their typing context is consistent but not live (see Example 4.4).

Regarding item 2, it is important to notice that if a consistent typing context Δ is not live, then (by Definition 4.1) it contains some internal/external choice that can be reached but never fired; therefore, Δ types processes that might wait forever on some output/input that cannot be possibly triggered by another process (see Example 4.4). For this reason, we argue that non-live typing contexts should be considered ill-formed, and can be reasonably rejected. Indeed, in "classic" MPST, well-formed global types and projections are used to rule out this kind of ill-formed protocols.

We now further investigate the differences between or new theory and the "classic" one. In Section 5.1, we show that our typing system supports all multiparty protocols projected from some global type, *plus* other "live" protocols that cannot be projected from any global type; in Section 5.2, we show that our typing system tolerates some (safe) mismatches between processes and types; finally, in Section 5.3, we show how our typing rules can be simplified (in some cases) to match the "classic" ones.

5.1. Global types, and beyond

Our theory covers all typing contexts obtained by projecting some global type G. This is implied by Lemma 5.1 below — where $\mathrm{roles}(G)$ is the set of roles occurring in G.

Lemma 5.1. *Take any global type* G, *and let* $\Delta_G = \{s[\mathtt{r}] : S_\mathtt{r}\}_{\mathtt{r} \in \mathrm{roles}(G)}$ *such that* $\forall \mathtt{r} \in \mathrm{roles}(G) : S_\mathtt{r} = G \upharpoonright \mathtt{r}$. *Then, we have* $\mathrm{live}(\Delta_G)$.

Proof. *(Sketch)* This is a consequence of the correspondence between multiparty session types and *Communicating Finite-State Machines (CFSMs)* [8], originally established by Deniélou and Yoshida [14], and later reprised by Bocchi et al. [6].

Deniélou and Yoshida [14] define the projection of a global type G onto a *system of CFSMs* (one per role of G), that we denote here as $\mathcal{S}_G$. The projection corresponds to the usual global-to-local type projection (Definition A.1), and generates restricted CFSMs whose states are *non-mixed* and *directed* — i.e., each state q is either an input state (with receive transitions, only) or an output state (with send transitions, only), and each transition from q is an input/output from/to the same participant. With these restrictions, each CFSM in $\mathcal{S}_G$ corresponds to a local multiparty session type (Definition 2.6), and the system $\mathcal{S}_G$ is homomorphic to the projected typing context Δ_G (since it contains the same projected session types).

Then, by Theorem 4.3 in Deniélou and Yoshida [14], we know that $\mathcal{S}_G$ is *Multiparty Compatible (MC)* ([14, Def. 4.2]; [6, Def. 4]). In our setting, MC (with the formulation of Bocchi et al. [6]) can be translated as follows:

(MC1) $\Delta_G \twoheadrightarrow^* \Delta', s[\mathtt{p}] : S$ with $S \equiv \mathtt{q} \& _{i \in I} \mathtt{m}_i(S_i).S'_i$ implies $\exists k \in I$: $\exists \Delta'', \Delta'''$: $\Delta' \twoheadrightarrow^* \Delta''$ and $\Delta'', s[\mathtt{p}] : S \to \Delta''', s[\mathtt{p}] : S'_k$;
(MC2) $\Delta_G \twoheadrightarrow^* \Delta', s[\mathtt{p}] : S$ with $S \equiv \mathtt{q} \oplus _{i \in I} \mathtt{m}_i(S_i).S'_i$ implies $\forall k \in I$: $\exists \Delta'', \Delta'''$: $\Delta' \twoheadrightarrow^* \Delta''$ and $\Delta'', s[\mathtt{p}] : S \to \Delta''', s[\mathtt{p}] : S'_k$.

Clause **(MC1)** says that whenever Δ_G reduces to a configuration with a type/CFSM for $\mathtt{p}$ that inputs from $\mathtt{q}$, then it can eventually receive a corresponding message from $\mathtt{q}$. Clause **(MC2)** says that whenever Δ_G reduces to a configuration with a type/CFSM for $\mathtt{p}$ that outputs some message towards $\mathtt{q}$, then the message can be eventually received by $\mathtt{q}$. Besides the use of session types instead of non-mixed and directed CFSMs, the above translation of MC has two differences w.r.t. Deniélou and Yoshida [14] and Bocchi et al. [6]:

1. our typing context reductions (Definition 2.10) are synchronous. Instead, Deniélou and Yoshida [14] and Bocchi et al. [6] have *asynchronous* communication via message queues. Still, MC is based on a notion of synchronous reduction: it only considers restricted pairs of "alternating" transitions where each message is first queued, and then received, without other transitions in between;

2. our typing context reductions (Definition 2.10) compare *both* message labels *and* message payloads — and the latter are compared with the up-to unfolding equivalence $\equiv$ (Definition 2.6). Instead, Deniélou and Yoshida [14] and Bocchi et al. [6] do not have payloads, and use CFSM transitions that only compare message labels. Still, extending CFSMs to support and compare message payloads with the same treatment of Definition 2.10 is not difficult, and has no consequence on MC and its properties.

Now, summing up: if we have Δ_G defined as in the hypotheses, by Deniélou and Yoshida [14] and Bocchi et al. [6] we know that Δ_G satisfies clauses **(MC1)** and **(MC2)** above. Then, we can show that Δ_G is live: it suffices to define a predicate $\mathbb{P}$ containing Δ_G and its reducts and unfoldings (similarly to Examples 4.2 and 4.3), and show that each element of $\mathbb{P}$ satisfies the clauses of Definition 4.1; in particular, clause [L-&] holds by **(MC1)**, and [L-$\oplus$] holds by **(MC2)**. Hence the result. □

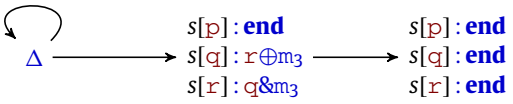
Lemma 5.1 means that our typing judgement (Definition 4.5) supports all multiparty protocols typically considered in "classic" MPST theory. Moreover, for all typed processes, it provides a subject reduction result that is not restricted by consistency (Theorem 4.9): this is *not* the case in "classic" MPST, as discussed in Section 3.3.

Additionally, our rely/guarantee typing system goes beyond global types: in Examples 5.2 and 5.3 below, we show two typing contexts that are live (and thus supported by our theory), but cannot be obtained by projecting any global type.

Example 5.2. Consider the following typing context (that is not consistent):

$$\Delta = s[\mathtt{p}]{:}\mu\mathbf{t}.\mathtt{q}\oplus\{\mathtt{m}_1.\mathbf{t},\ \mathtt{m}_2\},\ s[\mathtt{q}]{:}\mu\mathbf{t}.\mathtt{p}\&\{\mathtt{m}_1.\mathbf{t},\ \mathtt{m}_2.\mathtt{r}\oplus\mathtt{m}_3\},\ s[\mathtt{r}]{:}\mathtt{q}\&\mathtt{m}_3$$

It reduces as follows:



In this diagram, Δ loops when $s[\mathtt{p}]$ sends $\mathtt{m}_1$ to $s[\mathtt{q}]$; otherwise, if $s[\mathtt{p}]$ sends $\mathtt{m}_2$ to $s[\mathtt{q}]$, then $s[\mathtt{q}]$ sends $\mathtt{m}_3$ to $s[\mathtt{r}]$, and the final configuration is reached. By inspecting these reductions, we can verify that Δ is live.

However, Δ is not projectable from any global type. In fact, observe that the type of $s[\mathtt{q}]$ recursively waits to receive either $\mathtt{m}_1$ or $\mathtt{m}_2$ from $\mathtt{p}$, and only in the second case, $s[\mathtt{q}]$ sends $\mathtt{m}_3$ to $\mathtt{r}$. This means that an hypothetical G projecting into Δ should look like $G = \mu\mathbf{t}.\mathtt{p}\to\mathtt{q}\colon\{\mathtt{m1}.\mathbf{t},\ \mathtt{m2}.\mathtt{q}\to\mathtt{r}\colon\mathtt{m3}\}$, i.e., the global type syntactically involves role $\mathtt{r}$ in one branch, but *not* in the other — and this makes the projection $G\upharpoonright\mathtt{r}$ undefined (see Definition A.1).

Example 5.3. Consider the following typing context:

$$\Delta'_\mu = s[\mathtt{p}]{:}S,\ s[\mathtt{q}]{:}\mu\mathbf{t}'.\mathtt{p}\&\mathtt{m}(S).\mathbf{t}' \qquad \text{where } S = \mu\mathbf{t}.\mathtt{q}\oplus\mathtt{m}(\mathbf{t}).\mathbf{t}$$

Here, S is the type of a channel that must be used to recursively send a message $\mathtt{m}$ to a process playing role $\mathtt{q}$; moreover, $\mathtt{m}$ carries an S-typed payload channel that the recipient must use accordingly.

Note that Δ'_μ is not projectable from any global type, because projection (Definition A.1) cannot yield session types with recursion variables occurring as payloads. Moreover, Δ'_μ is *not* consistent: the projection $S\upharpoonright\mathtt{q} = \mu\mathbf{t}'.\oplus\mathtt{m}(\mathbf{t}).\mathbf{t}'$ yields an invalid partial type, with an open session type as message payload (Definition A.2).

By Definition 2.10, Δ'_μ reduces as follows:

$$\dfrac{\dfrac{\dfrac{S \equiv S}{s[\mathtt{p}]{:}\mathtt{q}\oplus\mathtt{m}(S).S,\ s[\mathtt{q}]{:}\mathtt{p}\&\mathtt{m}(S).\mu\mathbf{t}'.\mathtt{p}\&\mathtt{m}(S).\mathbf{t}' \ \to\ \Delta'_\mu}\ [\text{T-Comm}]}{s[\mathtt{p}]{:}\mathtt{q}\oplus\mathtt{m}(S).S,\ s[\mathtt{q}]{:}\mu\mathbf{t}'.\mathtt{p}\&\mathtt{m}(S).\mathbf{t}' \ \to\ \Delta'_\mu}\ [\text{T-}\mu]}{\Delta'_\mu \to \Delta'_\mu}\ [\text{T-}\mu]$$

and we can prove live(Δ'_μ) similarly to Example 4.2, by *(1)* defining a predicate $\mathbb{P}$ containing Δ'_μ and its unfoldings, and *(2)* showing that the elements of $\mathbb{P}$ satisfy the clauses of Definition 4.1.

Note that in our rely/guarantee session typing theory, the following judgement holds:

$$\emptyset \vdash Q_\mathtt{p} \mid Q_\mathtt{q} \triangleright \Delta'_\mu \triangleleft \emptyset \qquad \text{where :}$$

$$Q_\mathtt{p} \ = \ \mathbf{def}\ X(x) = \left(\begin{array}{l}\mathbf{def}\ Z(z) = z[\mathtt{p}]\sum\{\mathtt{m}(x').(Z\langle z\rangle \mid X\langle x'\rangle)\}\\ \mathbf{in}\ \left((\nu s')\left(x[\mathtt{q}]\oplus\mathtt{m}\langle s'[\mathtt{p}]\rangle.X\langle x\rangle \mid Z\langle s'[\mathtt{q}]\rangle\right)\right)\end{array}\right)\ \mathbf{in}\ X\langle s[\mathtt{p}]\rangle$$

$$Q_\mathtt{q} \ = \ \mathbf{def}\ Z(z) = \left(\begin{array}{l}\mathbf{def}\ X(x) = \left((\nu s')\left(x[\mathtt{q}]\oplus\mathtt{m}\langle s'[\mathtt{p}]\rangle.X\langle x\rangle \mid Z\langle s'[\mathtt{q}]\rangle\right)\right)\\ \mathbf{in}\ z[\mathtt{p}]\sum\{\mathtt{m}(x).(Z\langle z\rangle \mid X\langle x\rangle)\}\end{array}\right)\ \mathbf{in}\ Z\langle s[\mathtt{p}]\rangle$$

and therefore, by Theorem 4.9, we know that $Q_\mathtt{p} \mid Q_\mathtt{q}$ interact type-safely.

Remark 5.4. Example 5.3 shows that by using liveness, our rely/guarantee typing system does not incur in the limitations of duality applied to recursive types, that have been variously addressed in several works. We will reprise this topic in Section 6.

5.2. Leveraging typing context reductions

Unlike "classic" MPST typing systems, our rules [T-&] and [T-⊕] (Fig. 3) inspect typing context reductions. This adds a certain flexibility: the contextual information can be exploited to type processes in ways that would be forbidden in "classic" MPST, as shown in Example 5.5 below.

Example 5.5. Consider the following variation of the session types in Example 2.8:

$$S'_{\mathtt{a}} = \mathtt{bob}\oplus\mathtt{stop}\,.\,\mathbf{end} \qquad S'_{\mathtt{b}} = \mathtt{alice}\&\left\{\begin{array}{l}\mathtt{m1}(\mathsf{Int})\,.\,\mathtt{alice}\oplus\mathtt{m2}(\mathsf{Str})\,.\,\mathbf{end}\,,\\ \mathtt{stop}\,.\,\mathtt{carol}\oplus\mathtt{quit}\,.\,\mathbf{end}\end{array}\right\} \qquad S'_{\mathtt{c}} = \mathtt{bob}\&\left\{\begin{array}{l}\mathtt{m2}(\mathsf{Str})\,.\,\mathbf{end}\,,\\ \mathtt{quit}\,.\,\mathbf{end}\end{array}\right\}$$

The differences with Example 2.8 are:

1. $S'_{\mathtt{a}}$ must send $\mathtt{stop}$ to $\mathtt{bob}$, and has no other choice;
2. if $S'_{\mathtt{b}}$ receives $\mathtt{m}_1$ from $\mathtt{alice}$, then it sends $\mathtt{m}_2$ to $\mathtt{alice}$ (instead of $\mathtt{carol}$);
3. if $S'_{\mathtt{c}}$ receives $\mathtt{m}_2$ from $\mathtt{bob}$, then it terminates without sending $\mathtt{m}_3$ to $\mathtt{alice}$.

Note that the typing context $s[\mathtt{alice}]:S'_{\mathtt{a}}, s[\mathtt{bob}]:S'_{\mathtt{b}}, s[\mathtt{carol}]:S'_{\mathtt{c}}$ is live, because the mismatching outputs in the altered branches of $S'_{\mathtt{b}}$ and $S'_{\mathtt{c}}$ are never reached, and therefore, do not violate clauses [L-&] and [L-⊕] of Definition 4.1.

Now, consider the processes in Example 2.3: with the variations 1–3 above, the process $P_{\mathtt{a}}$ still matches $S'_{\mathtt{a}}$, but $P_{\mathtt{b}}$ and $P_{\mathtt{c}}$ do *not* match $S'_{\mathtt{b}}$ and $S'_{\mathtt{c}}$. Nonetheless, the following typing derivation holds (here, we abbreviate $\mathtt{alice}$, $\mathtt{bob}$, $\mathtt{carol}$ resp. as $\mathtt{a}$, $\mathtt{b}$, $\mathtt{c}$):

$$\dfrac{\dfrac{\dfrac{\vdots}{\emptyset \vdash P_{\mathtt{a}} \triangleright s[\mathtt{a}]:S'_{\mathtt{a}} \triangleleft s[\mathtt{b}]:S'_{\mathtt{b}}, s[\mathtt{c}]:S'_{\mathtt{c}}} \quad \dfrac{\vdots}{\emptyset \vdash P_{\mathtt{b}} \triangleright s[\mathtt{b}]:S'_{\mathtt{b}} \triangleleft s[\mathtt{a}]:S'_{\mathtt{a}}, s[\mathtt{c}]:S'_{\mathtt{c}}}}{\emptyset \vdash P_{\mathtt{a}} \mid P_{\mathtt{b}} \triangleright s[\mathtt{a}]:S'_{\mathtt{a}}, s[\mathtt{b}]:S'_{\mathtt{b}} \triangleleft s[\mathtt{c}]:S'_{\mathtt{c}}}\ [\text{T-}|] \quad \dfrac{\vdots}{\emptyset \vdash P_{\mathtt{c}} \triangleright s[\mathtt{c}]:S'_{\mathtt{c}} \triangleleft s[\mathtt{a}]:S'_{\mathtt{a}}, s[\mathtt{b}]:S'_{\mathtt{b}}}\ [\text{T-}|]}{\Delta' = s[\mathtt{a}]:S'_{\mathtt{a}}, s[\mathtt{b}]:S'_{\mathtt{b}}, s[\mathtt{c}]:S'_{\mathtt{c}} \qquad \emptyset \vdash P_{\mathtt{a}} \mid P_{\mathtt{b}} \mid P_{\mathtt{c}} \triangleright s[\mathtt{a}]:S'_{\mathtt{a}}, s[\mathtt{b}]:S'_{\mathtt{b}}, s[\mathtt{c}]:S'_{\mathtt{c}} \triangleleft \emptyset}\ [\text{T-}\nu]}{\emptyset \vdash (\nu s{:}\Delta')(P_{\mathtt{a}} \mid P_{\mathtt{b}} \mid P_{\mathtt{c}}) \triangleright \emptyset \triangleleft \emptyset}$$

The reason is that, when typing $P_{\mathtt{b}}$ and $P_{\mathtt{c}}$, the rely context does *not* trigger the altered branches of $S'_{\mathtt{b}}$ and $S'_{\mathtt{c}}$, and correspondingly, the typing rule [T-&] in Fig. 3 does *not* check its inductive clauses for the corresponding process continuations. As a consequence, the typing system (safely) accepts this process/type discrepancy.

5.3. From "new" to simpler "old" rules

The typing rules [T-&]/[T-⊕] (Fig. 3) can be demanding: the quantification "$\forall \Gamma' : \Gamma \twoheadrightarrow^* \Gamma' \ldots$" can yield many premises (but their number is always finite, cf. Remark 4.6). Fortunately, Lemma 5.6 below allows to reduce the number of premises that must be checked explicitly.

Lemma 5.6 *(Rely context strengthening). If* $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma$ *and* $\Gamma \twoheadrightarrow^* \Gamma'$*, then* $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma'$.

Intuitively, Lemma 5.6 holds because if a typing context reduces, it becomes "more deterministic"; hence, if P is typed relying on Γ, it can also rely on its reductions.

Additionally, type checks can be simplified using Proposition 5.7 below (where x is a variable, and *not* a channel with role, by Definition 2.1).

Proposition 5.7. *Assume* $x \notin \mathrm{dom}(\Delta) \cup \mathrm{dom}(\Gamma)$. *Then,* $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma, x{:}S$ *iff* $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma$.

By Proposition 5.7, variables in the rely context Γ are immaterial — intuitively, because they do *not* influence liveness, nor the possible reductions of Δ and Γ in the statement.

Lemma 5.6 and Proposition 5.7 become particularly useful when typing a process that branches/selects from/to a variable — and (in the latter case) sends a variable. In this case, rules [T-&] and [T-⊕] become the standard MPST typing rules for branch/input and select/output (see [11]), plus an *unused* rely context:

$$\frac{S \equiv \mathtt{q}\&_{i\in I}\mathtt{m}_i(S_i).S_i' \quad \forall i\in I \quad \Theta \vdash P_i \triangleright \Delta, y_i{:}S_i, x{:}S_i' \triangleleft \Gamma}{\Theta \vdash x[\mathtt{q}]\textstyle\sum_{i\in I\cup J}\{\mathtt{m}_i(y_i).P_i\} \triangleright \Delta, x{:}S \triangleleft \Gamma} \ [\mathrm{T}\&x]$$

$$\frac{S \equiv \mathtt{q}\oplus_{i\in I}\mathtt{m}_i(S_i).S_i' \quad k\in I \quad \Delta_y \vdash y{:}S_k \quad \Theta \vdash P \triangleright \Delta, x{:}S_k' \triangleleft \Gamma}{\Theta \vdash x[\mathtt{q}]\oplus\mathtt{m}_k\langle y\rangle.P \triangleright \Delta_y, \Delta, x{:}S \triangleleft \Gamma} \ [\mathrm{T}\oplus x]$$

This is because in [T-⊕], by Proposition 5.7 we can omit Δ_y from the rely context of the rightmost premise. Further, since $x{:}S$ reduces autonomously (cf. Definition 2.10), in both [T-&] and [T-⊕] the quantification "$\forall\Gamma', \Gamma'' : \Gamma \not\to^* \Gamma'$ and $\Gamma', x{:}S \to \Gamma'', x{:}S_i'$" amounts to "$\forall\Gamma' : \Gamma \not\to^* \Gamma'$ and $\Gamma', x{:}S \to \Gamma', x{:}S_i'$"; hence, by Lemma 5.6, checking the rightmost premise *once* with Γ is sufficient to know that it holds $\forall\Gamma'$ such that $\Gamma \not\to^* \Gamma'$.

This shows that in most cases, our rely/guarantee type checking can be simplified, becoming similar to the "classic" MPST rules.

6. Conclusions and related work

We presented a new multiparty session typing system (Definition 4.5), based on a *rely/guarantee* typing strategy, and a *liveness* invariant on typing contexts (Definition 4.1). Our typing system provides a subject reduction result (Theorem 4.9) that overcomes the consistency restrictions of "classic" MPST works. By directly exploiting the typing context semantics, our typing rules provide flexible type checks supporting all protocols projectable from global types, *plus* various kinds of live protocols that cannot be projected from any global type (Section 5). As a bonus, the independence from global types means that our theory is simpler than the "classic" one, in the sense that it does *not* depend on its syntactic type manipulations (i.e., the type projection/merging/duality machinery of "classic" MPST, summarised in Appendix A).

Inter-role dependencies Our rely/guarantee typing system allows to prove type safety of processes implementing protocols with complex inter-role dependencies: this is generally not supported by "classic" MPST, as discussed in Sections 1, 3.4 and 5, and Examples 3.4 and 4.7. In this respect, to the best of our knowledge, the only work with comparable features is the one by Dezani-Ciancaglini et al. [15]: it provides a "non-classic" MPST theory where processes can only interact on *one* session — and this limitation is crucially exploited to type parallel compositions without splitting their typing context (cf. Table 8, rule [T-SESS]). Their approach directly tracks a global type along process reductions, without introducing consistency restrictions. However, unlike the present work, Dezani-Ciancaglini et al. [15] do not support multiple sessions and delegation — and extending their work in that direction appears challenging. Moreover, our approach does not depend on the existence of global types: therefore, it is arguably simpler (since it only uses local types), and supports protocols for which no corresponding global type exists, as shown in Section 5.1.

Global types and liveness Our type system does *not* assume that a session is typed by projecting some global type: it only (implicitly) requires that a session is typed by a live composition of *local* session types, by rule [T-ν] (Fig. 3). If needed, a set of local types can be related to a choreography either via "top-down" projection (Section 3.1), or via "bottom-up" synthesis (studied, e.g., by Lange and Tuosto [26] and Lange et al. [27]): these concerns are orthogonal to our typing system. Our Definition 4.1 (liveness) is inspired by *(1)* the notion of *safety* for Communicating Finite-State Machines (CFSMs) Brand and Zafiropulo [8] (no deadlocks, orphan messages, unspecified receptions) and *(2)* the idea of "multiparty session types as CFSMs" outlined by Deniélou and Yoshida [14]. In particular, liveness (Definition 4.1) is closely related to the notion of *Multiparty Compatibility* for CFSMs, as illustrated in Section 5.1 (proof of Lemma 5.1). Notably, Deniélou and Yoshida [14] focus on projecting CFSMs and studying their interactions, and do *not* technically develop the intuition of "CFSMs as types" into an actual typing system. For this reason, they do not realise that the "classic" MPST theory cannot fully support their intuition, due to the consistency requirement and the resulting restrictions. Our work provides an actual multiparty session typing system that sidesteps these limitations, and comes closer to using "CFSMs as types."

Deadlock freedom Similarly to most previous MPST works, our approach ensures that a typed ensemble of processes interacting on a single session (i.e., a typed $(\nu s)(\mid_{\mathtt{p}\in I} P_{\mathtt{p}})$, with each $P_{\mathtt{p}}$ only interacting on $s[\mathtt{p}]$) is deadlock-free; however, it does not guarantee deadlock freedom in presence multiple interleaved sessions. This topic is studied by Bettini et al. [5] and Coppo et al. [12]: as future work, we plan to investigate how to reuse their results in our theory.

Recursion and duality Various works on binary session types have investigated the subtle interplay between recursion and duality. Bono and Padovani [7] and Bernardi and Hennessy [3] independently noticed that the "standard" duality proposed by Honda et al. [20] (corresponding to Definition A.3) does *not* commute with the unfolding of recursion for non-tail-recursive types (i.e., with type variables as payloads). An example of such types is $\mu\mathbf{t}.\oplus\mathtt{m}(\mathbf{t}).\mathbf{t}$, similar to our Example 5.3. This implies that some "safe" process compositions cannot be typed in binary session theories — and this limitation can also be found in "classic" MPST works, due to their consistency requirement (that is rooted in binary duality, cf. Definition 3.3). To solve this issue, Bono and Padovani [7] and Bernardi and Hennessy [3] defined a new notion of duality, called *complement* by the latter, and further studied by Bernardi et al. [2]. However, [4, Example 5.21] later noticed that even complement does *not* commute, e.g., when unfolding $\mu\mathbf{t}.\mu\mathbf{t}'.\&\mathtt{m}(\mathbf{t}').\mathbf{t}$. This issue was later solved by Lindley and Morris [28, Section 5.2]:

they add *dualised type variables* to session types, with specialised duality and unfolding definitions that commute as desired. Our approach sidesteps these issues by *not* requiring syntactic duality, but using a liveness property that unfolds types as necessary, and (coinductively) compares the trees of payload types (see Example 5.3 and its discussion).

Encodability and implementations Scalas et al. [33] have proven that the consistency requirement of "classic" MPST creates a strong link between MPST and linear π-calculus with *binary* channels, allowing to encode the former into the latter. They also exploit this result to provide an implementation of "classic" MPST in Scala [34]. Our new rely/guarantee typing system does not depend on consistency: therefore, the encoding strategy of Scalas et al. [33] can only be applied to the consistent fragment of our theory, and implementing the rest is now an open problem.

Future work We plan to investigate typed behavioural congruences and bisimulations for our new theory, as done by Kouzapas and Yoshida [24,25] for "classic" MPST.

We also plan to extend our approach to *asynchronous* multiparty session types [21,22] where process interaction is mediated by message queues (similarly to the TCP/IP protocol). The main differences and challenges we foresee stem from the fact that, in the present work, the *synchronous* typing context reduction semantics (Definition 2.10) induce *finite* state transition systems — whereas the addition of types for message queues can yield infinite-state systems, impacting decidability of liveness and type checking.

Acknowledgements

This work was partially supported by EPSRC grants EP/K011715/1, EP/K034413/1, EP/L00058X/1, EP/N028201/1, and EP/N027833/1. We thank the anonymous reviewers for their valuable comments and suggestions, and Raymond Hu, Julien Lange, and Bernardo Toninho for the fruitful discussion.

Appendix A. "Classic" multiparty session types: definitions

This appendix contains standard definitions from MPST literature. They are mainly referenced from Section 3, but are *not* necessary for our rely/guarantee typing system (Section 4): it uses a *behavioural* liveness property (Definition 4.1), and does not depend on the existence of global types and their projections.

A.1. Partial session types and projections

The *global and local type projections* in Definition A.1 and Definition A.4 below, with their respective *merging operators*, are based on [38] and [13]. They are more flexible than the simpler projection/merging used, e.g., by Coppo et al. [11]: hence, they allow for more global types to be projectable.

Intuitively, the projection and merging in Definition A.1 below allow to reconstruct the expected behaviour of a role in a global type, by collecting and combining its internal/external choices along different execution paths.

Definition A.1 *(Projection of a global type).* The *projection of G onto role* $\mathtt{p}$, written $G\upharpoonright\mathtt{p}$, is:

$$(\mathtt{q}\to\mathtt{r}\colon\{\mathtt{m}_i(T_i)\,.\,G_i\}_{i\in I})\upharpoonright\mathtt{p}=\begin{cases}\mathtt{r}\oplus_{i\in I}\mathtt{m}_i(T_i).(G_i\upharpoonright\mathtt{p}) & \text{if } \mathtt{p}=\mathtt{q}\\ \mathtt{q}\&_{i\in I}\mathtt{m}_i(T_i).(G_i\upharpoonright\mathtt{p}) & \text{if } \mathtt{p}=\mathtt{r}\\ \bigsqcap_{i\in I} G_i\upharpoonright\mathtt{p} & \text{if } \mathtt{q}\neq\mathtt{p}\neq\mathtt{r}\end{cases}\qquad(\mu\mathbf{t}.G)\upharpoonright\mathtt{p}=\begin{cases}\mu\mathbf{t}.(G\upharpoonright\mathtt{p}) & \text{if } G\upharpoonright\mathtt{p}\neq\mathbf{t}'\ (\forall\mathbf{t}')\\ \mathbf{end} & \text{otherwise}\end{cases}$$

$$\mathbf{t}\upharpoonright\mathtt{p}=\mathbf{t}\qquad\mathbf{end}\upharpoonright\mathtt{p}=\mathbf{end}$$

where $\bigsqcap$ is the *merge operator for (local) session types*, defined as:

$$\mathtt{p}\&_{i\in I}\mathtt{m}_i(S_i).S'_i\sqcap\mathtt{p}\&_{j\in J}\mathtt{m}_j(S_j).T'_j=\mathtt{p}\&_{k\in I\cap J}\mathtt{m}_k(S_k).(S'_k\sqcap T'_k)\ \&\ \mathtt{p}\&_{i\in I\setminus J}\mathtt{m}_i(S_i).S'_i\ \&\ \mathtt{p}\&_{j\in J\setminus I}\mathtt{m}_j(S_j).T'_j$$

$$\mathtt{p}\oplus_{i\in I}\mathtt{m}_i(S_i).S'_i\sqcap\mathtt{p}\oplus_{i\in I}\mathtt{m}_i(S_i).S'_i=\mathtt{p}\oplus_{i\in I}\mathtt{m}_i(S_i).S'_i$$

$$\mu\mathbf{t}.S\sqcap\mu\mathbf{t}.T=\mu\mathbf{t}.(S\sqcap T)\qquad\mathbf{t}\sqcap\mathbf{t}=\mathbf{t}\qquad\mathbf{end}\sqcap\mathbf{end}=\mathbf{end}$$

For example, if we have a global type $G=\mathtt{p}\to\mathtt{q}\colon\{\mathtt{m}_i(T_i)\,.\,\mathbf{end}\}_{i\in I}$, then the projection $G\upharpoonright\mathtt{p}$ describes $\mathtt{p}$'s behaviour as an internal choice towards $\mathtt{q}$, with labels $\mathtt{m}_i$ and payload T_i ($i\in I$); dually, the projection $G\upharpoonright\mathtt{q}$ is an external choice from $\mathtt{p}$, with labels $\mathtt{m}_i$ and payloads T_i. Instead, the projection of G onto a role $\mathtt{r}$ that is neither $\mathtt{p}$ nor $\mathtt{q}$ requires to skip the first interaction (since it does not involve $\mathtt{r}$), compute the projections onto $\mathtt{r}$ along each branch $\mathtt{m}_i$ (for each $i\in I$), and then combine the resulting projections using the *merge operator* $\bigsqcap$: in this case, $G\upharpoonright\mathtt{r}=\bigsqcap_{i\in I}(\mathbf{end}\upharpoonright\mathtt{r})=\bigsqcap_{i\in I}\mathbf{end}=\mathbf{end}$, i.e., $\mathtt{r}$ does not perform any input/output in G.

Note that projection and marging are sometimes undefined: for example, it is possible to merge different external choices (by merging the continuation types that follow common message label), but *not* different internal choices. This rules out meaningless global types like $G'=\mathtt{p}\to\mathtt{q}\colon\{\mathtt{m}_1\,.\,\mathtt{r}\to\mathtt{p}\colon\mathtt{m}_1\,,\,\mathtt{m}_2\,.\,\mathtt{r}\to\mathtt{p}\colon\mathtt{m}_2\}$: here, $\mathtt{r}$ is supposed to send either $\mathtt{m}_1$ or $\mathtt{m}_2$ to $\mathtt{p}$, depending on what message $\mathtt{p}$ sent to $\mathtt{q}$ earlier — but $\mathtt{r}$ is not privy to the interactions between other roles, and correspondingly, the projection $G'\upharpoonright\mathtt{r}=\mathtt{p}\oplus\mathtt{m}_1\sqcap\mathtt{p}\oplus\mathtt{m}_2$ is undefined.

A.2. Partial session types and duality

A session type S can be further projected onto a role $\mathtt{p}$, to "isolate" the interactions of S involving $\mathtt{p}$ from those involving other roles (Definition A.4 below). The projection mechanics remind of those in Definition A.1 — but in this case, the result is a *partial session type* (Definition A.2 below); moreover, the merging operator is different (details below). Partial session types feature branches, selections and recursion — but no role annotations: therefore, they are similar to binary session types (except that their payload types are multiparty), and endowed with a notion of duality (Definition A.3 below). As shown in Section 3.2, these technicalities are necessary to define consistency (Definition 3.3).

Definition A.2 *(Partial session types). Partial session types*, ranged over by H, have the syntax:

$$H \;::=\; \&_{i\in I}\mathtt{m}_i(S_i).H_i \;\mid\; \oplus_{i\in I}\mathtt{m}_i(S_i).H_i \;\mid\; \mathbf{end} \;\mid\; \mu\mathbf{t}.H \;\mid\; \mathbf{t} \qquad \text{with } I\neq\emptyset \text{ and } \forall i\in I : \mathrm{fv}(S_i)=\emptyset$$

where $\mathtt{m}_i$ range over pairwise distinct *message labels*.

Definition A.3 *(Duality of partial session types).* The *dual of a partial type* H, written $\overline{H}$, is:

$$\overline{\&_{i\in I}\mathtt{m}_i(S_i).H_i} = \oplus_{i\in I}\mathtt{m}_i(S_i).\overline{H_i} \qquad \overline{\oplus_{i\in I}\mathtt{m}_i(S_i).H_i} = \&_{i\in I}\mathtt{m}_i(S_i).\overline{H_i} \qquad \overline{\mu\mathbf{t}.H} = \mu\mathbf{t}.\overline{H} \qquad \overline{\mathbf{t}} = \mathbf{t} \qquad \overline{\mathbf{end}} = \mathbf{end}$$

We say that H *and* H' *are dual* iff $\overline{H} = H'$.

Definition A.4 *(Partial projection).* The *projection of* S *onto* $\mathtt{p}$, written $S\upharpoonright\mathtt{p}$, is a partial type defined as:

$$(\mathtt{q}\&_{i\in I}\mathtt{m}_i(S_i).S'_i)\upharpoonright\mathtt{p} = \begin{cases}\&_{i\in I}\mathtt{m}_i(S_i).(S'_i\upharpoonright\mathtt{p}) & \text{if } \mathtt{p}=\mathtt{q}\\ \bigsqcap_{i\in I} S'_i\upharpoonright\mathtt{p} & \text{if } \mathtt{p}\neq\mathtt{q}\end{cases} \qquad (\mathtt{q}\oplus_{i\in I}\mathtt{m}_i(S_i).S'_i)\upharpoonright\mathtt{p} = \begin{cases}\oplus_{i\in I}\mathtt{m}_i(S_i).(S'_i\upharpoonright\mathtt{p}) & \text{if } \mathtt{p}=\mathtt{q}\\ \bigsqcap_{i\in I} S'_i\upharpoonright\mathtt{p} & \text{if } \mathtt{p}\neq\mathtt{q}\end{cases}$$

$$(\mu\mathbf{t}.S)\upharpoonright\mathtt{p} = \begin{cases}\mu\mathbf{t}.(S\upharpoonright\mathtt{p}) & \text{if } S\upharpoonright\mathtt{p}\neq\mathbf{t}' \;(\forall\mathbf{t}')\\ \mathbf{end} & \text{otherwise}\end{cases} \qquad \mathbf{t}\upharpoonright\mathtt{p} = \mathbf{t} \qquad \mathbf{end}\upharpoonright\mathtt{p} = \mathbf{end}$$

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

$$\&_{i\in I}\mathtt{m}_i(S_i).H_i \sqcap \&_{i\in I}\mathtt{m}_i(S_i).H'_i = \&_{i\in I}\mathtt{m}_i(S_i).(H_i\sqcap H'_i)$$
$$\oplus_{i\in I}\mathtt{m}_i(S_i).H_i \sqcap \oplus_{j\in J}\mathtt{m}_j(S_j).H'_j = \oplus_{k\in I\cap J}\mathtt{m}_k(S_k).(H_k\sqcap H'_k) \oplus \oplus_{i\in I\setminus J}\mathtt{m}_i(S_i).H_i \oplus \oplus_{j\in J\setminus I}\mathtt{m}_j(S_j).H'_j$$
$$\mu\mathbf{t}.H \sqcap \mu\mathbf{t}.H' = \mu\mathbf{t}.(H\sqcap H') \qquad \mathbf{t}\sqcap\mathbf{t} = \mathbf{t} \qquad \mathbf{end}\sqcap\mathbf{end} = \mathbf{end}$$

For example, if we have a session type $S = \mathtt{p}\oplus_{i\in I}\mathtt{m}_i(S_i).\mathbf{end}$, its projection $S\upharpoonright\mathtt{p}$ is the partial session type $\oplus_{i\in I}\mathtt{m}_i(S_i).\mathbf{end}$, reflecting the fact that S requires to perform an internal choice towards $\mathtt{p}$. Instead, if S is projected onto a role $\mathtt{q}\neq\mathtt{p}$, we need to skip the first output towards $\mathtt{p}$, compute the projections onto $\mathtt{q}$ along each branch $\mathtt{m}_i$ $(i\in I)$, and merge them, using the partial session type merging operator $\bigsqcap$. Therefore, we have $S\upharpoonright\mathtt{p} = \bigsqcap_{i\in I}(\mathbf{end}\upharpoonright\mathtt{q}) = \bigsqcap_{i\in I}\mathbf{end} = \mathbf{end}$ — i.e., $\mathtt{q}$ is not involved in any interaction in S.

Note that, unlike the session type merging operator in Definition A.1, the partial type merging in Definition A.4 allows to combine different *internal* choices, but *not* different *external* choices. Therefore, we have the following examples:

$$S' = \mathtt{p}\&\left\{\begin{array}{l}\mathtt{m}_1(S_1).\mathtt{q}\oplus\mathtt{m}_1(S_1)\,,\\ \mathtt{m}_2(S_2).\mathtt{q}\oplus\mathtt{m}_2(S_2)\end{array}\right\} \qquad S'\upharpoonright\mathtt{q} = (\mathtt{q}\oplus\mathtt{m}_1(S_1))\upharpoonright\mathtt{q}\sqcap(\mathtt{q}\oplus\mathtt{m}_2(S_2))\upharpoonright\mathtt{q} = \oplus\mathtt{m}_1(S_1)\sqcap\oplus\mathtt{m}_2(S_2) = \oplus\left\{\begin{array}{l}\mathtt{m}_1(S_1)\,,\\ \mathtt{m}_2(S_2)\end{array}\right\}$$

$$S'' = \mathtt{p}\&\left\{\begin{array}{l}\mathtt{m}_1(S_1).\mathtt{q}\&\mathtt{m}_1(S_1)\,,\\ \mathtt{m}_2(S_2).\mathtt{q}\&\mathtt{m}_2(S_2)\end{array}\right\} \qquad S''\upharpoonright\mathtt{q} = (\mathtt{q}\&\mathtt{m}_1(S_1))\upharpoonright\mathtt{q}\sqcap(\mathtt{q}\&\mathtt{m}_2(S_2))\upharpoonright\mathtt{q} = \&\mathtt{m}_1(S_1)\sqcap\&\mathtt{m}_2(S_2) \quad \textit{(undefined)}$$

The partial projection $S'\upharpoonright\mathtt{q}$ is defined, and says that S' might send either $\mathtt{m}_1$ or $\mathtt{m}_2$ to $\mathtt{q}$; by inspecting S', we can further notice that $\mathtt{m}_1$ (resp. $\mathtt{m}_2$) should be sent to $\mathtt{q}$ only if $\mathtt{m}_1$ (resp. $\mathtt{m}_2$) is first received from $\mathtt{p}$ — but this dependency from $\mathtt{p}$'s messages is safely approximated as an internal choice.

Instead, the partial projection $S''\upharpoonright\mathtt{q}$ is undefined: this is because S'' might be willing to receive either $\mathtt{m}_1$ or $\mathtt{m}_2$ from $\mathtt{q}$, depending on which message is first received from $\mathtt{p}$ — but this might lead to incorrect interactions, as shown below:

$$s[\mathtt{r}]{:}S'',\ s[\mathtt{p}]{:}\mathtt{r}\oplus\mathtt{m}_1(S_1),\ s[\mathtt{q}]{:}\mathtt{r}\oplus\mathtt{m}_2(S_2) \;\to\; s[\mathtt{r}]{:}\mathtt{q}\&\mathtt{m}_1(S_1),\ s[\mathtt{p}]{:}\mathbf{end},\ s[\mathtt{q}]{:}\mathtt{r}\oplus\mathtt{m}_2(S_2) \;\not\to \tag{A.1}$$

Since the projection $S''\upharpoonright\mathtt{q}$ is undefined, the typing context in (A.1) is not consistent (Definition 3.3). By imposing such restrictions, the "classic" MPST theory excludes typing contexts where, e.g., an output from $\mathtt{q}$ to $\mathtt{r}$ might not be matched by a corresponding input. [4] For more examples and discussion, see Sections 3.3 and 3.4.

[4] Notably, the typing context in (A.1) is also *not* live: we can easily verify that it does not satisfy Definition 4.1.

The final ingredient for Definition 3.3 is partial session subtyping, formalised in Definition A.5 below.

Definition A.5 *(Partial session subtyping).* The *subtyping relation* $\leqslant$ *for partial types* is coinductively defined by the rules:

$$\frac{\forall i \in I \quad S_i \leqslant T_i \quad H_i' \leqslant H_i''}{\&_{i\in I}\mathtt{m}_i(S_i).H_i' \leqslant \&_{i\in I\cup J}\mathtt{m}_i(T_i).H_i''} \qquad \frac{\forall i \in I \quad T_i \leqslant S_i \quad H_i' \leqslant H_i''}{\oplus_{i\in I\cup J}\mathtt{m}_i(S_i).H_i' \leqslant \oplus_{i\in I}\mathtt{m}_i(T_i).H_i''}$$

$$\frac{}{\mathbf{end} \leqslant \mathbf{end}} \qquad \frac{H\{\mu\mathbf{t}.H/\mathbf{t}\} \leqslant H'}{\mu\mathbf{t}.H \leqslant H'} \qquad \frac{H \leqslant H'\{\mu\mathbf{t}.H'/\mathbf{t}\}}{H \leqslant \mu\mathbf{t}.H'}$$

Definition A.5 is similar to binary session subtyping, studied by Gay and Hole [17]: it says that a smaller type is "less demanding", as it allows to choose between more internal choices, and requires to support less external choices.

Appendix B. Typing context reductions

Proposition B.1. *For all* $\Delta, \Delta', \Delta_0$ *such that* $\mathrm{dom}(\Delta) \cap \mathrm{dom}(\Delta_0) = \emptyset$*:* $\Delta \to \Delta'$ *if and only if* $\Delta, \Delta_0 \to \Delta', \Delta_0$*, with* Δ_0 not *reducing.*

Proof. ($\Longrightarrow$) By induction on the size of Δ_0, and by rule [T-Comp] of Definition 2.10.
($\Longleftarrow$) By induction on the size of Δ_0, and by inversion of rule [T-Comp] of Definition 2.10. □

Proposition B.2. *For all* $\Delta, \Delta', \Delta_0$ *such that* $\mathrm{dom}(\Delta) \cap \mathrm{dom}(\Delta_0) = \emptyset$*:* $\Delta \overbrace{\to \cdots \to}^{n \text{ steps}} \Delta'$ *if and only if* $\Delta, \Delta_0 \overbrace{\to \cdots \to}^{n \text{ steps}} \Delta', \Delta_0$*, with* Δ_0 not *reducing.*

Proof. By induction on n, using (B.1) in the inductive case. □

Appendix C. Liveness

Proposition C.1. *If* $\mathrm{live}(\Delta)$ *and* $\Delta \twoheadrightarrow^* \Delta'$*, then* $\mathrm{live}(\Delta')$.

Proof. By induction on the number of reductions in $\Delta \twoheadrightarrow^* \Delta'$. The base case (0 reductions) is trivial, as it implies $\Delta' = \Delta$. The inductive case ($n+1$ reductions) is proved by the induction hypothesis, and clause [L-$\to$] of Definition 4.1 (liveness). □

Proposition C.2. *Assume* $\mathrm{live}(\Delta_0)$*. Then,* $\Delta, \Delta_0 \to \Delta'$ *implies* $\Delta' = \Delta'', \Delta_0''$ *such that either:*

(a) $\Delta'' = \Delta$ *and* $\Delta_0 \to \Delta_0''$*, or*
(b) $\Delta \to \Delta''$ *and* $\Delta_0'' = \Delta_0$.

Proof. Notice that we must have $\mathrm{dom}(\Delta) \cap \mathrm{dom}(\Delta_0) = \emptyset$ (otherwise, Δ, Δ_0 would be undefined, by Definition 2.9). By contradiction, assume that neither *(a)* nor *(b)* holds. Then, one entry of Δ and one of Δ_0 interact, via rule [T-Comm] of Definition 2.10. Hence, Δ_0 contains some (possibly recursive) internal/external choice that cannot be triggered by the other elements of Δ_0: this violates clause [L-$\oplus$]/[L-&] of Definition 4.1, and therefore, Δ_0 is not live — contradiction. We conclude that either *(a)* or *(b)* holds. □

Proposition C.3. *Assume* $\mathrm{live}(\Delta_0)$*. Then,* $\Delta, \Delta_0 \overbrace{\to \cdots \to}^{n \text{ steps}} \Delta'$ *implies* $\Delta' = \Delta'', \Delta_0''$ *such that, for some* $m \leq n$*,* $\Delta \overbrace{\to \cdots \to}^{m \text{ steps}} \Delta''$ *and* $\Delta_0 \overbrace{\to \cdots \to}^{n-m \text{ steps}} \Delta_0''$*; moreover,* $\Delta, \Delta_0 \overbrace{\to \cdots \to}^{m \text{ steps}} \Delta'', \Delta_0$.

Proof. By induction on n using Proposition C.2. The *"moreover…"* part of the statement is an immediate consequence of the first part, and Proposition B.2. □

Lemma C.4. *Assume* $\mathrm{live}(\Delta_0)$ *and* $\mathrm{dom}(\Delta) \cap \mathrm{dom}(\Delta_0) = \emptyset$*. Then,* $\mathrm{live}(\Delta, \Delta_0)$ *if and only if* $\mathrm{live}(\Delta)$.

Proof. Assuming the hypotheses, we have:

$$\mathrm{live}(\Delta_0) \qquad \text{(by hypothesis)} \qquad \text{(C.1)}$$
$$\mathrm{dom}(\Delta) \cap \mathrm{dom}(\Delta_0) = \emptyset \qquad \text{(by hypothesis)} \qquad \text{(C.2)}$$

($\Longrightarrow$) We show that the following predicate:

$$\mathbb{P} = \{\Delta \mid \text{live}(\Delta, \Delta_0)\} \tag{C.3}$$

satisfies the clauses of Definition 4.1 (liveness). We proceed by examining each $\Delta \in \mathbb{P}$, and we have the following (non-mutually exclusive) cases:

- $\Delta = \Delta', s[\mathtt{p}]:S$ with $S = \mathtt{q} \oplus_{i \in I} \mathtt{m}_i(S_i).S'_i$. We need to show that Δ satisfies clause [L-$\oplus$]. We have:

$$\text{live}(\Delta, \Delta_0) \quad \text{and thus} \quad \text{live}(\Delta', s[\mathtt{p}]:S, \Delta_0) \qquad \text{(by (C.3))} \tag{C.4}$$
$$\forall i \in I : \exists \Delta'' : \Delta', s[\mathtt{p}]:S, \Delta_0 \twoheadrightarrow^* \Delta'', s[\mathtt{p}]:S'_i, \Delta_0 \qquad \text{(by (C.4), [L-$\oplus$], (C.1) and Proposition C.3)} \tag{C.5}$$
$$\forall i \in I : \exists \Delta'' : \Delta', s[\mathtt{p}]:S \twoheadrightarrow^* \Delta'', s[\mathtt{p}]:S'_i \qquad \text{(by (C.5) and Proposition B.2)} \tag{C.6}$$

  and by the shape of Δ and (C.6), we conclude that Δ satisfies clause [L-$\oplus$];
- $\Delta = \Delta', s[\mathtt{p}]:S$ with $S = \mathtt{q} \&_{i \in I} \mathtt{m}_i(S_i).S'_i$. We need to show that Δ satisfies clause [L-&]. We have:

$$\text{live}(\Delta, \Delta_0) \quad \text{and thus} \quad \text{live}(\Delta', s[\mathtt{p}]:S, \Delta_0) \qquad \text{(by (C.3))} \tag{C.7}$$
$$\exists i \in I : \exists \Delta'' : \Delta', s[\mathtt{p}]:S, \Delta_0 \twoheadrightarrow^* \Delta'', s[\mathtt{p}]:S'_i, \Delta_0 \qquad \text{(by (C.7), [L-&], (C.1) and Proposition C.3)} \tag{C.8}$$
$$\exists i \in I : \exists \Delta'' : \Delta', s[\mathtt{p}]:S \twoheadrightarrow^* \Delta'', s[\mathtt{p}]:S'_i \qquad \text{(by (C.8) and Proposition B.2)} \tag{C.9}$$

  and by the shape of Δ and (C.9), we conclude that Δ satisfies clause [L-&];
- $\Delta = \Delta', s[\mathtt{p}]:\mu \mathbf{t}.S$. We need to show that Δ satisfies clause [L-μ]. We have:

$$\text{live}(\Delta, \Delta_0) \quad \text{and thus} \quad \text{live}(\Delta', s[\mathtt{p}]:\mu \mathbf{t}.S, \Delta_0) \qquad \text{(by (C.3))} \tag{C.10}$$
$$\text{live}(\Delta', s[\mathtt{p}]:S\{\mu \mathbf{t}.S/\mathbf{t}\}, \Delta_0) \qquad \text{(by (C.10) and [L-μ])} \tag{C.11}$$
$$\Delta', s[\mathtt{p}]:S\{\mu \mathbf{t}.S/\mathbf{t}\} \in \mathbb{P} \qquad \text{(by (C.11) and (C.3))} \tag{C.12}$$

  and by the shape of Δ and (C.12), we conclude that Δ satisfies clause [L-μ];
- $\Delta \to \Delta'$. We need to show that Δ satisfies clause [L-$\to$]. We have:

$$\Delta, \Delta_0 \to \Delta', \Delta_0 \qquad \text{(by (C.2) and Proposition B.1)} \tag{C.13}$$
$$\text{live}(\Delta, \Delta_0) \qquad \text{(by (C.3))} \tag{C.14}$$
$$\text{live}(\Delta', \Delta_0) \qquad \text{(by (C.14), (C.13) and [L-$\to$])} \tag{C.15}$$
$$\Delta' \in \mathbb{P} \qquad \text{(by (C.15) and (C.3))} \tag{C.16}$$

  and by the hypothesis $\Delta \to \Delta'$ and (C.16), we conclude that Δ satisfies clause [L-$\to$].

We have proved that $\mathbb{P}$ satisfies all clauses of Definition 4.1 (liveness). Since live is the largest predicate satisfying such clauses, we have that $\forall \Delta : \Delta \in \mathbb{P}$ implies live(Δ); and since $\forall \Delta : \text{live}(\Delta, \Delta_0)$ implies $\Delta \in \mathbb{P}$, we conclude that $\forall \Delta : \text{live}(\Delta, \Delta_0)$ implies live(Δ).

($\Longleftarrow$) We show that the following predicate:

$$\mathbb{P} = \{\Delta, \Delta_0 \mid \text{live}(\Delta)\} \tag{C.17}$$

satisfies the clauses of Definition 4.1 (liveness). We proceed similarly to the case above ($\Longrightarrow$): by examining each $\Delta, \Delta_0 \in \mathbb{P}$. The difference is that we now use Proposition B.2 to add (rather than remove) Δ_0 when examining the corresponding element of $\mathbb{P}$ yielded by the set comprehension (C.17). Then, we know that live is the largest predicate satisfying all clauses of Definition 4.1 (liveness); therefore, we have that $\forall \Gamma : \Gamma \in \mathbb{P}$ implies live(Γ); and since $\forall \Delta : \text{live}(\Delta)$ implies $(\Delta, \Delta_0) \in \mathbb{P}$, we conclude that $\forall \Delta : \text{live}(\Delta)$ implies $\text{live}(\Delta, \Delta_0)$. □

Proposition C.5. $\text{live}(\Delta, x:S)$ *if and only if* $\text{live}(\Delta)$.

Proof. Straightforward by Definition 4.1 (liveness), noticing that $x:S$ is not relevant for the definition (which only considers channels with roles). □

Proposition C.6. *If* $\text{live}(\Delta, c:S)$ *and* $S \equiv S'$, *then* $\text{live}(\Delta, c:S')$.

Proof. By showing that the following predicate:

$$\mathbb{P} = \left\{ \Delta, c{:}S' \;\middle|\; \text{live}(\Delta, c{:}S) \text{ and } S \equiv S' \right\}$$

satisfies the clauses of Definition 4.1 (liveness). Then, since live is the largest predicate satisfying such clauses, we have that $\forall \Gamma : \Gamma \in \mathbb{P}$ implies live(Γ); and since $\forall \Delta, S' : \text{live}(\Delta, c{:}S)$ and $S \equiv S'$ implies $(\Delta, c{:}S') \in \mathbb{P}$, we conclude that $\forall \Delta, S' :$ live$(\Delta, c{:}S)$ and $S \equiv S'$ implies live$(\Delta, c{:}S')$. □

Proposition C.7. *If* live(Δ, Δ') *and* $\Delta' \equiv \Delta''$, *then* live(Δ, Δ'').

Proof. By induction on the size of Δ': the base case $\Delta' = \emptyset$ is trivial, and the inductive case is proved using Proposition C.6. □

Proposition C.8. *If* end(Δ), *then* live(Δ).

Proof. Take Δ, Δ' such that $\text{dom}(\Delta) = \text{dom}(\Delta')$ and $\forall c \in \text{dom}(\Delta) : \Delta'(c) = \mathbf{end}$. By Definition 4.5, the singleton predicate $\{\Delta'\}$ (vacuously) satisfies clauses [L-&], [L-⊕] [L-μ], and [L-→] of Definition 4.1 (liveness): hence, live(Δ'). Now, assume end(Δ): by [T-End] in Fig. 3, we have $\Delta \equiv \Delta'$; therefore, by Proposition C.7, we conclude live(Δ). □

Appendix D. Basic properties of typing rules

Lemma D.1 *(Typing inversion). Assume* $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma$. *Then, exactly one of the following holds:*

- [T-**0**] $P = \mathbf{0}$ *and* end(Δ)
- [T-**def**] $P = \mathbf{def}\ X(x_1, \ldots, x_n) = P'\ \mathbf{in}\ Q$ *and* $\exists S_1, \ldots, S_n$ *such that:*
  - $\Theta, X{:}S_1, \ldots, S_n \vdash P' \triangleright x_1{:}S_1, \ldots, x_n{:}S_n \triangleleft \emptyset$
  - $\Theta, X{:}S_1, \ldots, S_n \vdash Q \triangleright \Delta \triangleleft \Gamma$
- [T-X] $P = X\langle c_1, \ldots, c_n \rangle$ *and* $\exists S_1, \ldots, S_n$ *such that:*
  - $\Delta = \Delta_0, \Delta_1, \ldots, \Delta_n$
  - $\Theta \vdash X : S_1, \ldots, S_n$
  - end(Δ_0)
  - $\forall i \in 1..n : \ \Delta_i \vdash c_i{:}S_i$
- [T-ν] $P = (\nu s{:}\Delta_s)\ Q$ *and:*
  - $\Delta_s = \left\{ s[\mathtt{p}]{:}S_{\mathtt{p}} \right\}_{\mathtt{p} \in I}$
  - $\Theta \vdash Q \triangleright \Delta, \Delta_s \triangleleft \Gamma$
- [T-|] $P = P_1 \mid P_2$ *and* $\exists \Delta_1, \Delta_2$ *such that:*
  - $\Delta = \Delta_1, \Delta_2$
  - $\Theta \vdash P_1 \triangleright \Delta_1 \triangleleft \Gamma, \Delta_2$
  - $\Theta \vdash P_2 \triangleright \Delta_2 \triangleleft \Gamma, \Delta_1$
- [T-&] $P = c[\mathtt{q}] \sum_{i \in I \cup J} \{\mathtt{m}_i(y_i).P_i\}$ *and* $\exists \Delta_0, S$ *such that:*
  - $\Delta = \Delta_0, c{:}S$
  - $S \equiv \mathtt{q} \& _{i \in I} \mathtt{m}_i(S_i).S'_i$
  - $\forall i \in I$ *and* $\forall \Gamma', \Gamma''$ *such that* $\Gamma \twoheadrightarrow^* \Gamma'$ *and* $\Gamma', c{:}S \to \Gamma'', c{:}S'_i$ *:*
    - $\Theta \vdash P_i \triangleright \Delta, y_i{:}S_i, c{:}S'_i \triangleleft \Gamma''$
- [T-⊕] $P = c[\mathtt{q}] \oplus \mathtt{m}_k\langle d \rangle.P'$ *and* $\exists \Delta_d, \Delta_0, S$ *such that:*
  - $\Delta = \Delta_d, \Delta_0, c{:}S$
  - $S \equiv \mathtt{q} \oplus _{i \in I} \mathtt{m}_i(S_i).S'_i$
  - $\exists k \in I$ *such that:*
    - $\Delta_d \vdash d{:}S_k$
    - $\forall \Gamma', \Gamma''$ *such that* $\Gamma \twoheadrightarrow^* \Gamma'$ *and* $\Gamma', c{:}S \to \Gamma'', c{:}S'_k$ *:*
      - $\Theta \vdash P' \triangleright \Delta, c{:}S'_k \triangleleft \Gamma'', \Delta_d$

Proof. Straightforward by observing that the typing rules in Fig. 3 can be deterministically inverted: i.e., each process production in Fig. 1 can be typed by at most one rule. □

Lemma 5.6 *(Rely context strengthening). If* $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma$ *and* $\Gamma \twoheadrightarrow^* \Gamma'$, *then* $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma'$.

Proof. By induction on the derivation of $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma$. The crucial cases are [T-&] and [T-⊕]: they are both proved by inverting the typing rule (using Lemma D.1) and observing that its premises require the continuations of P to be typed using all reducts of Γ — which include all reducts of Γ'. This allows to apply the induction hypothesis, and conclude by the same rule. □

Proposition D.2. *Assume* $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma$. *Take some* Γ' *such that* $\mathrm{live}(\Gamma')$ *and* $(\mathrm{dom}(\Delta) \cup \mathrm{dom}(\Gamma)) \cap \mathrm{dom}(\Gamma') = \emptyset$. *Then,* $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma, \Gamma'$.

Proof. By induction on the derivation of $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma$. □

Proposition D.3. *Assume* $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma$. *Take some* Δ' *such that* $\mathrm{end}(\Delta')$ *and* $(\mathrm{dom}(\Delta) \cup \mathrm{dom}(\Gamma)) \cap \mathrm{dom}(\Delta') = \emptyset$. *Then,* $\Theta \vdash P \triangleright \Delta, \Delta' \triangleleft \Gamma$.

Proof. By induction on the derivation of $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma$. □

Proposition D.4. *Assume* $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma$. *If* $X \notin \mathrm{dom}(\Theta)$, *then* $\Theta, X{:}\widetilde{S} \vdash P \triangleright \Delta \triangleleft \Gamma$.

Proof. By induction on the derivation of $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma$. □

Lemma D.5 *(Subject congruence). If* $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma$ *and* $P \equiv P'$, *then* $\Theta \vdash P' \triangleright \Delta \triangleleft \Gamma$.

Proof. Assuming, $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma$, the proof proceeds by cases on P, examining the possible shapes of P' allowed by $\equiv$ in Definition 2.4, and using Lemma D.1 to reconstruct their typing. □

Appendix E. Substitution lemma

Proposition 5.7. *Assume* $x \notin \mathrm{dom}(\Delta) \cup \mathrm{dom}(\Gamma)$. *Then,* $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma, x{:}S$ *iff* $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma$.

Proof. ($\Longrightarrow$) By induction on the derivation of $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma, x{:}S$.
($\Longleftarrow$) By induction on the derivation of $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma$. □

Corollary E.1. $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma$ *iff* $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma \setminus x$.

Proof. If $x \notin \mathrm{dom}(\Gamma)$, then the statement holds vacuously. Otherwise, if $x \in \mathrm{dom}(\Gamma)$, it is a direct consequence of Proposition 5.7. □

Proposition E.2. *Assume* $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma$ *and* $x \in \mathrm{fv}(P)$. *Then,* $x \in \mathrm{dom}(\Delta)$.

Proof. By induction on the derivation of $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma$. □

Lemma 4.8 *(Substitution). If* $\Theta \vdash P \triangleright \Delta, x{:}S \triangleleft \Gamma, \Gamma'$ *and* $\Gamma \vdash s[\mathtt{p}]{:}S$, *then* $\Theta \vdash P\{^{s[\mathtt{p}]}/_{x}\} \triangleright \Delta, s[\mathtt{p}]{:}S \triangleleft \Gamma'$.

Proof. Assume the hypotheses. We have:

$$\Theta \vdash P \triangleright \Delta, x{:}S \triangleleft \Gamma \qquad \text{(by hypothesis)} \tag{E.1}$$

$$\Gamma = s[\mathtt{p}]{:}S' \text{ with } S \equiv S' \qquad \text{(by hypothesis and [T-Chan] in Fig. 3)} \tag{E.2}$$

$$\mathrm{live}(\Delta, x{:}S, \Gamma, \Gamma') \qquad \text{(by Definition 4.5)} \tag{E.3}$$

$$\mathrm{live}(\Delta, \Gamma, \Gamma') \qquad \text{(by (E.3) and Proposition 5.7)} \tag{E.4}$$

Note that, in the judgement in the conclusion, the liveness requirement on the guarantee/rely contexts (Definition 4.5) is trivially satisfied:

$$\mathrm{live}(\Delta, s[\mathtt{p}]{:}S, \Gamma') \qquad \text{(by (E.4), (E.2) and Proposition C.7)} \tag{E.5}$$

We proceed by induction on the derivation of $\Theta \vdash P \triangleright \Delta, x{:}S \triangleleft \Gamma, \Gamma'$:

- Base case [T-**0**]. We have:

$$P = \mathbf{0} \quad \text{and} \quad \mathrm{end}(\Delta, x{:}S) \qquad \text{(by conclusion of [T-0] and Lemma D.1, case [T-0])} \tag{E.6}$$

$$S \equiv \mathbf{end} \qquad \text{(by (E.6) and [T-End] in Fig. 3)} \tag{E.7}$$

$$\mathrm{end}(\Delta) \qquad \text{(by (E.6) and [T-End] in Fig. 3)} \tag{E.8}$$

$$\mathrm{end}(\Delta, s[\mathtt{p}]:S) \qquad \text{(by (E.8), (E.7) and [T-End] in Fig. 3)} \tag{E.9}$$

$$P\{^{s[\mathtt{p}]}/_{x}\} = \mathbf{0}\{^{s[\mathtt{p}]}/_{x}\} = \mathbf{0} \qquad \text{(from (E.6))} \tag{E.10}$$

$$\frac{\mathrm{end}(\Delta, s[\mathtt{p}]:S)}{\Theta \vdash P\{^{s[\mathtt{p}]}/_{x}\} \triangleright \Delta, s[\mathtt{p}]:S \triangleleft \Gamma'}\ [\text{T-}\mathbf{0}] \qquad \text{(by (E.9), (E.10) and (E.5))}$$

- Base case [T-X]. We have:

$$\frac{\Theta \vdash X : S_1, \ldots, S_n \quad \mathrm{end}(\Delta_0) \quad \forall i \in 1..n \quad \Delta_i \vdash c_i : S_i}{\Theta \vdash P \triangleright \Delta, x:S \triangleleft \Gamma}\ [\text{T-}X] \qquad \text{(by Lemma D.1, case [T-X])} \tag{E.11}$$

$$P = X\langle c_1, \ldots, c_n\rangle \qquad \text{(from the conclusion of (E.11))} \tag{E.12}$$

$$\Delta, x:S = \Delta_0, \Delta_1, \ldots, \Delta_n \qquad \text{(by (E.11))} \tag{E.13}$$

Now, from (E.13), we have two sub-cases, depending on whether x is used as parameter for X (i.e., whether $c_i = x$, for some $i \in 1..n$):
– x is *not* used as parameter for calling X, i.e., $\forall i \in 1..n : c_i \neq x$. Then:

$$P\{^{s[\mathtt{p}]}/_{x}\} = P \qquad \text{(from the hypothesis and (E.12))} \tag{E.14}$$

$$\Delta_0 \vdash x:S \qquad \text{(from the hypothesis, (E.11) and (E.13))} \tag{E.15}$$

$$\Delta_0 = x:S \ \text{ and } \ S \equiv \mathbf{end} \qquad \text{(by (E.15), (E.11) and [T-End] in Fig. 3)} \tag{E.16}$$

$$\mathrm{end}(s[\mathtt{p}]:S) \qquad \text{(by (E.16) and [T-End] in Fig. 3)} \tag{E.17}$$

$$\frac{\Theta \vdash X : S_1, \ldots, S_n \quad \mathrm{end}(s[\mathtt{p}]:S) \quad \forall i \in 1..n \quad \Delta_i \vdash c_i : S_i}{\Theta \vdash P\{^{s[\mathtt{p}]}/_{x}\} \triangleright \Delta, s[\mathtt{p}]:S \triangleleft \Gamma'}\ [\text{T-}X] \qquad \text{(by (E.16), (E.17), (E.14))}$$

– x is used as parameter for calling X, i.e., $c_k = x$ for some $k \in 1..n$. Then:

$$P\{^{s[\mathtt{p}]}/_{x}\} = X\langle c_1, \ldots, c_{k-1}, s[\mathtt{p}], c_{k+1}, \ldots, c_n\rangle \qquad \text{(from the hypothesis and (E.12))} \tag{E.18}$$

$$\Delta_k \equiv x:S \qquad \text{(by (E.11) and [T-Chan] in Fig. 3)} \tag{E.19}$$

$$\frac{\begin{array}{c}\Theta \vdash X : S_1, \ldots, S_n \quad \mathrm{end}(\Delta_0) \quad s[\mathtt{p}]:S \vdash s[\mathtt{p}]:S \\ \forall i \in \{1..n\} \setminus \{k\} \quad \Delta_i \vdash c_i : S_i\end{array}}{\Theta \vdash P\{^{s[\mathtt{p}]}/_{x}\} \triangleright \Delta, s[\mathtt{p}]:S \triangleleft \Gamma'}\ [\text{T-}X] \qquad \text{(by (E.11), (E.19) and (E.18))}$$

- Inductive case [T-&]. In this case, P is a branching on some channel. We have two sub-cases:
  (a) P is a branching on some $c \neq x$. Then, the substitution in the statement does not involve c, and the statement holds by the induction hypothesis and rule [T-&]; or
  (b) P is a branching on x, which in turn is substituted by a channel with role $s[\mathtt{p}]$.
  We detail the proof for the latter (and most interesting) case. We have:

$$P = x[\mathtt{q}]\textstyle\sum_{i\in I\cup J}\{\mathtt{m}_i(y_i).P_i\} \qquad \text{(conclusion of rule [T-&])} \tag{E.20}$$

$$\frac{\begin{array}{c}S \equiv \mathtt{q}\&_{i\in I}\mathtt{m}_i(S_i).S_i' \quad \forall i \in I \\ \forall \Gamma'', \Gamma''' : \Gamma, \Gamma' \twoheadrightarrow^{*} \Gamma'' \text{ and } \Gamma'', x:S \to \Gamma''', x:S_i' \\ \Theta \vdash P_i \triangleright \Delta, y_i:S_i, x:S_i' \triangleleft \Gamma'''\end{array}}{\Theta \vdash x[\mathtt{q}]\textstyle\sum_{i\in I\cup J}\{\mathtt{m}_i(y_i).P_i\} \triangleright \Delta, x:S \triangleleft \Gamma, \Gamma'}\ [\text{T-}\&] \qquad \text{(by Lemma D.1, case [T-&])} \tag{E.21}$$

$$\Gamma = s[\mathtt{p}]:S' \text{ with } S' \equiv S \equiv \mathtt{q}\&_{i\in I}\mathtt{m}_i(S_i).S_i' \qquad \text{(by (E.2), and (E.21))} \tag{E.22}$$

We can now observe that in (E.21), by Definition 2.10, we have $\forall i \in I : \Gamma'', x:S \to \Gamma'', x:S_i'$ — i.e., the typing context entry for x reduces "in isolation", and does not involve the rest of the typing context Γ''. Therefore, from the reductions in (E.21) we have:

$$\forall i \in I : \forall \Gamma'', \Gamma''' : \Gamma, \Gamma' \twoheadrightarrow^* \Gamma'' \text{ and } \Gamma'', x{:}S \to \Gamma''', x{:}S'_i \tag{E.23}$$

$$\text{implies } \Gamma''' = \Gamma'' \text{ and } \Theta \vdash P_i \triangleright \Delta, y_i{:}S_i, x{:}S'_i \triangleleft \Gamma'' \qquad \text{(from (E.21), premises of [T-\&])} \tag{E.24}$$

In order to apply the induction hypothesis, we focus on a *subset* of the reductions in (E.23) that involves the entries for x and $s[\mathtt{p}]$ as specified in (E.25) below (recall that $\Gamma = s[\mathtt{p}]{:}S' \equiv s[\mathtt{p}]{:}S$, by (E.22)):

$$\forall i \in I : \forall \Gamma'', \Gamma''' \text{ such that } \begin{cases} \Gamma' \twoheadrightarrow^* \Gamma'' \text{ and} \\ \Gamma'', s[\mathtt{p}]{:}S \to \Gamma''', s[\mathtt{p}]{:}S'_i \text{ and} \\ \Gamma''', s[\mathtt{p}]{:}S'_i, x{:}S \to \Gamma''', s[\mathtt{p}]{:}S'_i, x{:}S'_i : \end{cases} \tag{E.25}$$

$$\Theta \vdash P_i \triangleright \Delta, y_i{:}S_i, x{:}S'_i \triangleleft \Gamma''', s[\mathtt{p}]{:}S'_i \qquad \text{(by (E.25) and (E.24))} \tag{E.26}$$

$$s[\mathtt{p}]{:}S'_i \vdash s[\mathtt{p}]{:}S'_i \qquad \text{(by Definition 4.5)} \tag{E.27}$$

$$\Theta \vdash P_i\{s[\mathtt{p}]/x\} \triangleright \Delta, y_i{:}S_i, s[\mathtt{p}]{:}S'_i \triangleleft \Gamma''' \qquad \text{(by (E.26), (E.27) and induction hyp.)} \tag{E.28}$$

By exploiting (E.28), we can now conclude:

$$P\{s[\mathtt{p}]/x\} = s[\mathtt{p}][\mathtt{q}] \textstyle\sum_{i \in I \cup J} \{\mathtt{m}_i(y_i).P_i\{s[\mathtt{p}]/x\}\} \qquad \text{(by (E.20))} \tag{E.29}$$

$$\frac{\begin{array}{c} S \equiv \mathtt{q}\&_{i\in I}\mathtt{m}_i(S_i).S'_i \qquad \forall i \in I \\ \forall \Gamma'', \Gamma''' : \Gamma' \twoheadrightarrow^* \Gamma'' \text{ and } \Gamma'', s[\mathtt{p}]{:}S \to \Gamma''', s[\mathtt{p}]{:}S'_i \\ \Theta \vdash P_i\{s[\mathtt{p}]/x\} \triangleright \Delta, y_i{:}S_i, s[\mathtt{p}]{:}S'_i \triangleleft \Gamma''' \end{array}}{\Theta \vdash P\{s[\mathtt{p}]/x\} \triangleright \Delta, s[\mathtt{p}]{:}S \triangleleft \Gamma'} \text{ [T-\&]} \qquad \text{(by (E.29), (E.28))}$$

- Inductive case [T-⊕]. In this case, P is a selection on some variable. We have two sub-cases:
  (a) P is a selection on some $c \neq x$. Then, the substitution in the statement does not involve c, and the statement holds by the induction hypothesis and rule [T-⊕]; or
  (b) P is a selection on x, which in turn is substituted by a channel with role $s[\mathtt{p}]$.

  We detail the proof for the latter (and most interesting) case. The strategy is similar to the one we adopted for proving the case [T-&]. We have:

$$\exists k \in I : \; P = x[\mathtt{q}]\oplus\mathtt{m}_k\langle d\rangle.P' \qquad \text{(conclusion of rule [T-⊕])} \tag{E.30}$$

$$\Delta = \Delta_0, \Delta_d \text{ and } \frac{\begin{array}{c} S \equiv \mathtt{q}\oplus_{i\in I}\mathtt{m}_i(S_i).S'_i \quad \exists k \in I \quad \Delta_d \vdash d{:}S_k \\ \forall \Gamma'', \Gamma''' : \Gamma, \Gamma' \twoheadrightarrow^* \Gamma'' \text{ and } \Gamma'', x{:}S \to \Gamma''', x{:}S'_k \\ \Theta \vdash P' \triangleright \Delta_0, x{:}S'_k \triangleleft \Gamma''', \Delta_d \end{array}}{\Theta \vdash x[\mathtt{q}]\oplus\mathtt{m}_k\langle d\rangle.P' \triangleright \Delta, x{:}S \triangleleft \Gamma, \Gamma'} \text{ [T-⊕]} \qquad \text{(by Lemma D.1, case [T-⊕])} \tag{E.31}$$

$$\Gamma = s[\mathtt{p}]{:}S' \quad \text{with} \quad S' \equiv S \equiv s[\mathtt{p}]{:}\mathtt{q}\oplus_{i\in I}\mathtt{m}_i(S_i).S'_i \qquad \text{(by (E.2), and (E.31))} \tag{E.32}$$

$$\forall \Gamma'', \Gamma''' : \Gamma \twoheadrightarrow^* \Gamma'' \text{ and } \Gamma'', x{:}S \to \Gamma''', x{:}S'_k$$

$$\text{implies } \Gamma''' = \Gamma'' \text{ and } \Theta \vdash P \triangleright \Delta_0, x{:}S'_k \triangleleft \Gamma'', \Delta_d \qquad \text{(from (E.31), premises of [T-⊕])} \tag{E.33}$$

$$\forall \Gamma'', \Gamma''' \text{ such that } \begin{cases} \Gamma' \twoheadrightarrow^* \Gamma'' \text{ and} \\ \Gamma'', s[\mathtt{p}]{:}S \to \Gamma''', s[\mathtt{p}]{:}S'_k \text{ and} \\ \Gamma''', s[\mathtt{p}]{:}S'_k, x{:}S \to \Gamma''', s[\mathtt{p}]{:}S'_k, x{:}S'_k : \end{cases} \tag{E.34}$$

$$\Theta \vdash P \triangleright \Delta_0, x{:}S'_k \triangleleft \Gamma''', s[\mathtt{p}]{:}S'_k, \Delta_d \qquad \text{(by (E.34) and (E.33))} \tag{E.35}$$

$$s[\mathtt{p}]{:}S'_k, \vdash s[\mathtt{p}]{:}S'_k \qquad \text{(by Definition 4.5)} \tag{E.36}$$

$$\Theta \vdash P'\{s[\mathtt{p}]/x\} \triangleright \Delta_0, s[\mathtt{p}]{:}S'_k \triangleleft \Gamma''', \Delta_d \qquad \text{(by (E.35), (E.36) and induction hyp.)} \tag{E.37}$$

$$P\{s[\mathtt{p}]/x\} = s[\mathtt{p}][\mathtt{q}]\oplus\mathtt{m}_k\langle d\rangle.P'\{s[\mathtt{p}]/x\} \qquad \text{(by (E.30))} \tag{E.38}$$

$$\frac{\begin{array}{c} S \equiv \mathtt{q}\oplus_{i\in I}\mathtt{m}_i(S_i).S'_i \qquad \exists k \in I \qquad \Delta_d \vdash d{:}S_k \\ \forall \Gamma'', \Gamma''' : \Gamma' \twoheadrightarrow^* \Gamma'' \text{ and } \Gamma'', s[\mathtt{p}]{:}S \to \Gamma''', s[\mathtt{p}]{:}S'_k \\ \Theta \vdash P'\{s[\mathtt{p}]/x\} \triangleright \Delta_0, s[\mathtt{p}]{:}S'_k \triangleleft \Gamma''', \Delta_d \end{array}}{\Theta \vdash P\{s[\mathtt{p}]/x\} \triangleright \Delta, s[\mathtt{p}]{:}S \triangleleft \Gamma'} \text{ [T-⊕]} \qquad \text{(by (E.38), (E.37), (E.31))}$$

- Inductive case [T-|]. We have:

$$P = P_1 \mid P_2 \qquad \text{(conclusion of rule [T-|])} \tag{E.39}$$

$$\Delta = \Delta_1, \Delta_2 \quad \text{and} \quad \Theta \vdash P_1 \mid P_2 \triangleright \Delta_1, \Delta_2, x{:}S \triangleleft \Gamma, \Gamma' \qquad \text{(by (E.39), Lemma D.1, case [T-|])} \tag{E.40}$$

Now, the typing judgement in (E.40) implies that, in the corresponding instance of [T-|], $x{:}S$ could be used to type either P_1 or P_2 in the rule premises — i.e., we can have either:

$$\begin{cases} \Theta \vdash P_1 \triangleright \Delta_1, x{:}S \triangleleft \Gamma, \Gamma', \Delta_2 \\ \Theta \vdash P_2 \triangleright \Delta_2 \triangleleft \Gamma, \Gamma', \Delta_1, x{:}S \\ x \notin \mathrm{fv}(P_2) \end{cases} \qquad \text{(by Proposition E.2)} \tag{E.41}$$

or

$$\begin{cases} \Theta \vdash P_1 \triangleright \Delta_1 \triangleleft \Gamma, \Gamma', \Delta_2, x{:}S \\ \Theta \vdash P_2 \triangleright \Delta_2, x{:}S \triangleleft \Gamma, \Gamma', \Delta_1 \\ x \notin \mathrm{fv}(P_1) \end{cases} \qquad \text{(by Proposition E.2)} \tag{E.42}$$

We proceed assuming (E.41) (the proof for (E.42) is symmetric). We have:

$$\Theta \vdash P_1\{s[\mathtt{p}]/x\} \triangleright \Delta_1, s[\mathtt{p}]{:}S \triangleleft \Gamma', \Delta_2 \qquad \text{(by (E.41), (E.2) and induction hyp.)} \tag{E.43}$$

$$\Theta \vdash P_2 \triangleright \Delta_2 \triangleleft \Gamma', \Delta_1, s[\mathtt{p}]{:}S \qquad \text{(by (E.41), (E.2) and Proposition 5.7)} \tag{E.44}$$

$$P\{s[\mathtt{p}]/x\} = (P_1 \mid P_2)\{s[\mathtt{p}]/x\} = P_1\{s[\mathtt{p}]/x\} \mid P_2\{s[\mathtt{p}]/x\} \qquad \text{(by (E.39))} \tag{E.45}$$

$$P_2 = P_2\{s[\mathtt{p}]/x\} \qquad \text{(by (E.41), since } x \notin \mathrm{fv}(P_2)\text{)} \tag{E.46}$$

$$\frac{\Theta \vdash P_1\{s[\mathtt{p}]/x\} \triangleright \Delta_1, s[\mathtt{p}]{:}S \triangleleft \Gamma', \Delta_2 \quad \Theta \vdash P_2\{s[\mathtt{p}]/x\} \triangleright \Delta_2 \triangleleft \Gamma', \Delta_1, s[\mathtt{p}]{:}S}{\Theta \vdash P\{s[\mathtt{p}]/x\} \triangleright \Delta, s[\mathtt{p}]{:}S \triangleleft \Gamma'}\ [\text{T-|}] \qquad \text{(by (E.45), (E.43), (E.44), (E.46), (E.40))} \tag{E.47}$$

- Inductive case [T-**def**]. We have:

$$P = \mathbf{def}\ X(x_1, \ldots, x_n) = Q\ \mathbf{in}\ R \qquad \text{(conclusion of rule [T-def])} \tag{E.48}$$

$$\frac{\Theta, X{:}S_1, \ldots, S_n \vdash Q \triangleright x_1{:}S_1, \ldots, x_n{:}S_n \triangleleft \emptyset \quad \Theta, X{:}S_1, \ldots, S_n \vdash R \triangleright \Delta, x{:}S \triangleleft \Gamma, \Gamma'}{\Theta \vdash P \triangleright \Delta, x{:}S \triangleleft \Gamma, \Gamma'}\ [\text{T-def}] \qquad \text{(by (E.48) and Lemma D.1, case [T-def])} \tag{E.49}$$

$$\mathrm{fv}(Q) \subseteq \{x_1, \ldots, x_n\} \qquad \text{(by (E.49) (typing of } Q \text{) and Proposition E.2 } \times n\text{)} \tag{E.50}$$

$$\mathrm{fv}(\, X(x_1, \ldots, x_n) = Q\,) = \emptyset \qquad \text{(by (E.50))} \tag{E.51}$$

$$P\{s[\mathtt{p}]/x\} = \mathbf{def}\ X(x_1, \ldots, x_n) = Q\ \mathbf{in}\ (R\{s[\mathtt{p}]/x\}) \qquad \text{(by (E.48) and (E.51))} \tag{E.52}$$

$$\Theta, X{:}S_1, \ldots, S_n \vdash R\{s[\mathtt{p}]/x\} \triangleright \Delta, s[\mathtt{p}]{:}S \triangleleft \Gamma' \qquad \text{(by (E.49) (typing of } R\text{), induction hyp.)} \tag{E.53}$$

$$\frac{\Theta, X{:}S_1, \ldots, S_n \vdash Q \triangleright x_1{:}S_1, \ldots, x_n{:}S_n \triangleleft \emptyset \quad \Theta, X{:}S_1, \ldots, S_n \vdash R\{s[\mathtt{p}]/x\} \triangleright \Delta, s[\mathtt{p}]{:}S \triangleleft \Gamma'}{\Theta \vdash P\{s[\mathtt{p}]/x\} \triangleright \Delta, s[\mathtt{p}]{:}S \triangleleft \Gamma'}\ [\text{T-def}] \qquad \text{(by (E.52), (E.49) and (E.53))}\ \square$$

Appendix F. Subject reduction

Theorem 4.9 *(Subject reduction). If* $\Theta \vdash P \triangleright \Delta \triangleleft \Gamma$ *and* $P \twoheadrightarrow P'$, *then* $\exists \Delta'$ *such that* $\Delta \twoheadrightarrow \Delta'$ *and* $\Theta \vdash P' \triangleright \Delta' \triangleleft \Gamma$.

Proof. Assume:

$$\Theta \vdash P \triangleright \Delta \triangleleft \Gamma \qquad \text{(by hypothesis)} \tag{F.1}$$

Note that by (F.1) and Definition 4.5, we have live(Δ, Γ), and to obtain a valid typing judgement for P', we must ensure live(Δ', Γ): by Proposition C.1, this is guaranteed once we prove $\Delta \twoheadrightarrow^* \Delta'$, since it implies $\Delta, \Gamma \twoheadrightarrow^* \Delta', \Gamma$.

First, we prove the following property, for *one* reduction step of P:

$$P \to P' \text{ implies } \exists \Delta' \text{ such that } \Delta \twoheadrightarrow^* \Delta' \text{ and } \Theta \vdash P' \triangleright \Delta' \triangleleft \Gamma \tag{F.2}$$

Assuming the transition $P \to P'$, we proceed by induction on its derivation.

- Base case [R-Comm]. We have:

$$P = s[\mathtt{p}][\mathtt{q}] \textstyle\sum_{i \in I} \{\mathtt{m}_i(y_i).P_i\} \mid s[\mathtt{q}][\mathtt{p}] \oplus \mathtt{m}_k \langle s'[\mathtt{r}] \rangle . Q \quad (k \in I) \quad \text{(by inversion of the rule)} \tag{F.3}$$

$$P' = P_k\{^{s'[\mathtt{r}]}/_{y_k}\} \mid Q \quad \text{(by inversion of the rule)} \tag{F.4}$$

$$\exists \Delta_1, \Delta_2 : \ \Delta = \Delta_1, \Delta_2 \ \text{ and } \ \frac{\Theta \vdash s[\mathtt{p}][\mathtt{q}] \textstyle\sum_{i \in I} \{\mathtt{m}_i(y_i).P_i\} \triangleright \Delta_1 \triangleleft \Gamma, \Delta_2 \quad \Theta \vdash s[\mathtt{q}][\mathtt{p}] \oplus \mathtt{m}_k \langle s'[\mathtt{r}] \rangle . Q \triangleright \Delta_2 \triangleleft \Gamma, \Delta_1}{\Theta \vdash P \triangleright \Delta \triangleleft \Gamma}\ [\text{T-}|] \quad \text{(by (F.3), Lemma D.1, case [T-}|\text{])} \tag{F.5}$$

$$\exists \Gamma_1 = \Gamma, \Delta_2 \quad \text{and} \quad \exists \Gamma_2 = \Gamma, \Delta_1 \quad \text{(by (F.5))} \tag{F.6}$$

$$\exists J : \emptyset \neq J \subseteq I \quad \exists \Delta'_1, S : \ \Delta_1 = \Delta'_1, s[\mathtt{p}]{:}S \quad \text{such that}$$
$$\frac{\begin{array}{c} S \equiv \mathtt{q} \&_{i \in J} \mathtt{m}_i(S_i).S'_i \qquad \forall i \in J \\ \forall \Gamma', \Gamma'' : \ \Gamma_1 \twoheadrightarrow^* \Gamma' \text{ and } \Gamma', s[\mathtt{p}]{:}S \to \Gamma'', s[\mathtt{p}]{:}S'_i \\ \Theta \vdash P_i \triangleright \Delta'_1, y_i{:}S_i, s[\mathtt{p}]{:}S'_i \triangleleft \Gamma'' \end{array}}{\Theta \vdash s[\mathtt{p}][\mathtt{q}] \textstyle\sum_{i \in I} \{\mathtt{m}_i(y_i).P_i\} \triangleright \Delta_1 \triangleleft \Gamma_1}\ [\text{T-}\&] \quad \text{(by (F.5), Lemma D.1, case [T-}\&\text{])} \tag{F.7}$$

$$\exists I' \neq \emptyset \quad \exists \Delta_{2s'}, \Delta'_2, T : \ \Delta_2 = \Delta_{2s'}, \Delta'_2, s[\mathtt{q}]{:}T \quad \text{such that}$$
$$\frac{\begin{array}{c} T \equiv \mathtt{p} \oplus_{i \in I'} \mathtt{m}_i(T_i).T'_i \\ \exists k \in I' \qquad \Delta_{2s'} \vdash s'[\mathtt{r}]{:}T_k \\ \forall \Gamma', \Gamma'' : \ \Gamma_2 \twoheadrightarrow^* \Gamma' \text{ and } \Gamma', s[\mathtt{q}]{:}T \to \Gamma'', s[\mathtt{q}]{:}T'_k \\ \Theta \vdash Q \triangleright \Delta'_2, s[\mathtt{q}]{:}T'_k \triangleleft \Gamma'', \ \Delta_{2s'} \end{array}}{\Theta \vdash s[\mathtt{q}][\mathtt{p}] \oplus \mathtt{m}_k \langle s'[\mathtt{r}] \rangle . Q \triangleright \Delta_2 \triangleleft \Gamma_2}\ [\text{T-}\oplus] \quad \text{(by (F.5), Lemma D.1, case [T-}\oplus\text{])} \tag{F.8}$$

$$s[\mathtt{q}]{:}\mathtt{p} \oplus_{i \in I'} \mathtt{m}_i(T_i).T'_i \equiv s[\mathtt{q}]{:}T \in \Delta_2 \subsetneq \Gamma_1 \quad \text{(by (F.8) and (F.6))} \tag{F.9}$$

$$s[\mathtt{p}]{:}\mathtt{q} \&_{i \in J} \mathtt{m}_i(S_i).S'_i \equiv s[\mathtt{p}]{:}S \in \Delta_1 \subsetneq \Gamma_2 \quad \text{(by (F.7) and (F.6))} \tag{F.10}$$

$$I' \subseteq J \quad \text{and therefore} \quad k \in J \quad \text{(by (F.9), (F.6), (F.7), live}(\Delta_1, \Gamma_1) \text{ and Definition 4.1)} \tag{F.11}$$

$$\forall i \in J : \ S_i \equiv T_i \quad \text{(by (F.11), live}(\Delta_1, \Gamma_1) \text{ and Definition 4.1)} \tag{F.12}$$

$$\Delta_{2s'} \vdash s'[\mathtt{r}]{:}S_k \quad \text{(by (F.8) and (F.12))} \tag{F.13}$$

$$s'[\mathtt{r}]{:}S_k \equiv \in \Gamma_1 \quad \text{(by (F.13), (F.8) and (F.6))} \tag{F.14}$$

$$\forall i \in J : \ s[\mathtt{p}]{:}S, s[\mathtt{q}]{:}T \ \to \equiv \ s[\mathtt{p}]{:}S'_i, s[\mathtt{q}]{:}T_i \quad \text{((F.11), (F.9), live}(\Delta_1, \Gamma_1), \text{Definition 4.1)} \tag{F.15}$$

$$\exists \Gamma'_1 : \ \Gamma_1, s[\mathtt{p}]{:}S \ \to \equiv \ \Gamma'_1, s[\mathtt{p}]{:}S'_k, , s[\mathtt{q}]{:}T'_k \quad \text{(by (F.9) and (F.15))} \tag{F.16}$$

$$\Theta \vdash P_k \triangleright \Delta'_1, y_k{:}S_k, s[\mathtt{p}]{:}S'_k \triangleleft \Gamma'_1 \quad \text{(by (F.16) and (F.7))} \tag{F.17}$$

$$s'[\mathtt{r}]{:}S_k \equiv \in \Gamma'_1 \quad \text{(by (F.14) and (F.16))} \tag{F.18}$$

$$\Theta \vdash P_k\{^{s'[\mathtt{r}]}/_{y_k}\} \triangleright \Delta'_1, s'[\mathtt{r}]{:}S_k, s[\mathtt{p}]{:}S'_k \triangleleft \Gamma'_1 \backslash s'[\mathtt{r}] \quad \text{((F.17), (F.18), Lemma 4.8)} \tag{F.19}$$

$$\Gamma'_1 \backslash s'[\mathtt{r}] \equiv \Gamma, \Delta'_2, s[\mathtt{q}]{:}T'_k \quad \text{(by (F.6), (F.8), (F.13), and (F.16))} \tag{F.20}$$

$$\exists \Gamma'_2 : \ \Gamma_2, s[\mathtt{q}]{:}T \ \to \equiv \ \Gamma'_2, s[\mathtt{p}]{:}S'_k, s[\mathtt{q}]{:}T'_k \quad \text{(by (F.10) and (F.15))} \tag{F.21}$$

$$\Theta \vdash Q \triangleright \Delta'_2, s[\mathtt{q}]{:}T'_k \triangleleft \Gamma'_2, \ \Delta_{2s'} \quad \text{(by (F.21) and (F.8))} \tag{F.22}$$

$$\Gamma'_2, \ \Delta_{2s'} \equiv \Gamma, \Delta'_1, \Delta_{2s'}, s[\mathtt{p}]{:}S'_k \quad \text{(by (F.6), (F.7), (F.21), (F.13) and Definition 4.5)} \tag{F.23}$$

$\exists \Delta' = \Delta''_1, \Delta''_2$ such that:

$$\Delta''_1 = \Delta'_1,\ \Delta_{2s'},\ s[\mathtt{p}]{:}S'_k \tag{F.24}$$

$$\Delta''_2 = \Delta'_2,\ s[\mathtt{q}]{:}T'_k \tag{F.25}$$

$$\Delta \twoheadrightarrow^* \Delta' \qquad \text{(by (F.5), (F.7), (F.8), (F.24), (F.25))} \tag{F.26}$$

$$\frac{\Theta \vdash P_k\{s'[\mathtt{r}]/y_k\} \triangleright \Delta''_1 \triangleleft \Gamma, \Delta''_2 \quad \Theta \vdash Q \triangleright \Delta''_2 \triangleleft \Gamma, \Delta''_1}{\Theta \vdash P_k\{s'[\mathtt{r}]/y_k\} \mid Q \triangleright \Delta' \triangleleft \Gamma}\ [\text{T-}|] \qquad \left(\begin{array}{l}\text{by (F.19), (F.24), (F.20), (F.25);}\\ \text{by (F.22), (F.25), (F.23), (F.24)}\end{array}\right) \tag{F.27}$$

- Base case [R-X]. We have:

$$P = \mathbf{def}\ X(x_1, \ldots, x_n) = Q\ \mathbf{in}\ (X\langle s_1[\mathtt{p}_1], \ldots, s_1[\mathtt{p}_1]\rangle \mid R) \qquad \text{(by inversion of the rule)} \tag{F.28}$$

$$P' = \mathbf{def}\ X(x_1, \ldots, x_n) = Q\ \mathbf{in}\ (Q\{s_1[\mathtt{p}_1]/x_1\} \cdots \{s_n[\mathtt{p}_n]/x_n\} \mid R) \qquad \text{(by inversion of the rule)} \tag{F.29}$$

$$\Delta = \Delta_*, \Delta_R \text{ and } \Delta_* = \Delta_0, \Delta_1, \ldots, \Delta_n \text{ and } \Theta' = \Theta, X{:}S_1, \ldots, S_n \quad \text{such that:} \tag{F.30}$$

$$\frac{\Theta' \vdash Q \triangleright x_1{:}S_1, \ldots, x_n{:}S_n \triangleleft \emptyset \qquad \dfrac{\dfrac{\Theta' \vdash X : S_1, \ldots, S_n \quad \text{end}(\Delta_0) \quad \forall i \in 1..n \quad \Delta_i \vdash s_i[\mathtt{p}_i]{:}S_i}{\Theta' \vdash X\langle s_1[\mathtt{p}_1], \ldots, s_1[\mathtt{p}_1]\rangle \triangleright \Delta_* \triangleleft \Gamma, \Delta_R}\ [\text{T-}X] \quad \Theta' \vdash R \triangleright \Delta_R \triangleleft \Gamma, \Delta_*}{\Theta' \vdash X\langle s_1[\mathtt{p}_1], \ldots, s_1[\mathtt{p}_1]\rangle \mid R \triangleright \Delta \triangleleft \Gamma}\ [\text{T-}|]}{\Theta \vdash P \triangleright \Delta \triangleleft \Gamma}\ [\text{T-}\mathbf{def}]$$

(by (F.28), (F.1), Lemma D.1, cases [T-**def**],[T-|],[T-X]) (F.31)

$$\text{live}(\Delta_0) \qquad \text{(by (F.31) and Proposition C.8)} \tag{F.32}$$

$$\text{live}(\Delta_1, \ldots, \Delta_n, \Gamma, \Delta_R) \qquad \text{(by (F.31), Definition 4.5, (F.32), Lemma C.4)} \tag{F.33}$$

$$\text{live}((\Delta_1, \ldots, \Delta_n, \Gamma, \Delta_R)\backslash x_1, \ldots, x_n) \qquad \text{(by (F.33) and Proposition C.5 } \times n) \tag{F.34}$$

$$\text{live}(\Delta_1, \ldots, \Delta_n, ((\Gamma, \Delta_R)\backslash x_1, \ldots, x_n)) \qquad \text{(by (F.34), (F.31) and Definition 4.5)} \tag{F.35}$$

$$\Theta' \vdash Q \triangleright x_1{:}S_1, \ldots, x_n{:}S_n \triangleleft \Delta_1, \ldots, \Delta_n, ((\Gamma, \Delta_R)\backslash x_1, \ldots, x_n) \qquad \text{((F.35), (F.31), Proposition D.2)} \tag{F.36}$$

$$\Theta' \vdash Q\{s_1[\mathtt{p}_1]/x_1\} \cdots \{s_n[\mathtt{p}_n]/x_n\} \triangleright \Delta_1, \ldots, \Delta_n \triangleleft (\Gamma, \Delta_R)\backslash x_1, \ldots, x_n \qquad \left(\begin{array}{l}\text{(F.36), (F.31),}\\ \text{Lemma 4.8 } \times n\end{array}\right) \tag{F.37}$$

$$\Theta' \vdash Q\{s_1[\mathtt{p}_1]/x_1\} \cdots \{s_n[\mathtt{p}_n]/x_n\} \triangleright \Delta_* \triangleleft \Gamma, \Delta_R \qquad \left(\begin{array}{l}\text{(F.37), (F.30), (F.31),}\\ \text{Proposition D.3, Corollary E.1 } \times n\end{array}\right) \tag{F.38}$$

$$\text{Let } \Delta' = \Delta = \Delta_*, \Delta_R = \Delta_0, \Delta_1, \ldots, \Delta_n, \Delta_R \quad \text{(hence, } \Delta \twoheadrightarrow^* \Delta'\text{)} \qquad \text{(by (F.30))} \tag{F.39}$$

$$\frac{\Theta' \vdash Q \triangleright x_1{:}S_1, \ldots, x_n{:}S_n \triangleleft \emptyset \qquad \dfrac{\Theta' \vdash Q\{s_1[\mathtt{p}_1]/x_1\} \cdots \{s_n[\mathtt{p}_n]/x_n\} \triangleright \Delta_* \triangleleft \Gamma, \Delta_R \quad \Theta' \vdash R \triangleright \Delta_R \triangleleft \Gamma, \Delta_*}{\Theta' \vdash Q\{s_1[\mathtt{p}_1]/x_1\} \cdots \{s_n[\mathtt{p}_n]/x_n\} \mid R \triangleright \Delta' \triangleleft \Gamma}\ [\text{T-}|]}{\Theta \vdash P' \triangleright \Delta' \triangleleft \Gamma}\ [\text{T-}\mathbf{def}]$$

(by (F.38), (F.39), (F.29), (F.31)) (F.39)

- Inductive case [R-|]. We have:

$$P = P_1 \mid P_2 \qquad \text{(by inversion of the rule)} \tag{F.40}$$

$$\exists \Delta_1, \Delta_2 : \quad \Delta = \Delta_1, \Delta_2 \qquad \text{(by (F.40), (F.1) and Lemma D.1, case [T-|])} \tag{F.41}$$

$$\Theta \vdash P_1 \triangleright \Delta_1 \triangleleft \Gamma, \Delta_2 \qquad \text{(by (F.40), (F.41) and Lemma D.1, case [T-|])} \tag{F.42}$$

$$\Theta \vdash P_2 \triangleright \Delta_2 \triangleleft \Gamma, \Delta_1 \qquad \text{(by (F.40), (F.41) and Lemma D.1, case [T-|])} \tag{F.43}$$

$$P_1 \to P'_1 \quad \text{and} \quad P' = P'_1 \mid P_2 \qquad \text{(by inversion of [R-|])} \tag{F.44}$$

$$\exists \Delta'_1 : \quad \Delta_1 \twoheadrightarrow^* \Delta'_1 \quad \text{and} \quad \Theta \vdash P'_1 \triangleright \Delta'_1 \triangleleft \Gamma, \Delta_2 \qquad \text{(by (F.42), (F.44) and i.h.)} \tag{F.45}$$

$\Gamma, \Delta_1 \twoheadrightarrow^* \Gamma, \Delta_1'$ (by (F.45) and Definition 2.10) (F.46)

$\Theta \vdash P_2 \triangleright \Delta_2 \triangleleft \Gamma, \Delta_1'$ (by (F.43), (F.46) and Lemma 5.6) (F.47)

$\exists \Delta' = \Delta_1', \Delta_2 \quad \text{such that} \quad \Delta \twoheadrightarrow^* \Delta'$ (by (F.41), (F.45) and Definition 2.10) (F.48)

$$\frac{\Theta \vdash P_1' \triangleright \Delta_1' \triangleleft \Gamma, \Delta_2 \qquad \Theta \vdash P_2 \triangleright \Delta_2 \triangleleft \Gamma, \Delta_1'}{\Theta \vdash P' \triangleright \Delta' \triangleleft \Gamma}\ [\text{T-}|]$$

(by (F.44), (F.45), (F.47), (F.48))

- Inductive case [R-ν]. We have:

$\exists Q : \ P = (\nu s)\, Q$ (by inversion of the rule) (F.49)

$P \to P' \quad \text{implies} \quad \exists Q' : Q \to Q' \ \text{and} \ P' = (\nu s)\, Q'$ (by (F.49), inv. of [R-ν]) (F.50)

$$\frac{\Delta_s = \left\{c[\mathtt{p}] : S_{\mathtt{p}}\right\}_{\mathtt{p} \in I} \qquad \Theta \vdash Q \triangleright \Delta, \Delta_s \triangleleft \Gamma}{\Theta \vdash (\nu s : \Delta_s)\, Q \triangleright \Delta \triangleleft \Gamma}\ [\text{T-}\nu]$$

(by (F.49), Lemma D.1, case [T-ν]) (F.51)

$\exists \Delta'' : \ \Delta, \Delta_s \twoheadrightarrow^* \Delta'' \ \text{and} \ \Theta \vdash Q' \triangleright \Delta'' \triangleleft \Gamma$ (by (F.51), (F.50), i.h.) (F.52)

$\text{live}(\Delta_s)$ (by (F.51) (premise and conclusion of [T-ν]) and Lemma C.4) (F.53)

$\exists \Delta', \Delta_s' : \ \Delta'' = \Delta', \Delta_s' \ \text{and} \ \Delta \twoheadrightarrow^* \Delta' \ \text{and} \ \Delta_s \twoheadrightarrow^* \Delta_s'$ ((F.51), (F.52), (F.53), Proposition C.3) (F.54)

$\Delta, \Gamma \twoheadrightarrow^* \Delta', \Gamma$ (by (F.54) and Definition 2.10) (F.55)

$\text{live}(\Delta', \Gamma)$ (by (F.51), (F.55) and Proposition C.1) (F.56)

$$\frac{\Delta_s' = \left\{s[\mathtt{p}] : S_{\mathtt{p}}'\right\}_{\mathtt{p} \in I} \qquad \Theta \vdash Q' \triangleright \Delta', \Delta_s' \triangleleft \Gamma}{\Theta \vdash P' \triangleright \Delta' \triangleleft \Gamma}\ [\text{T-}\nu]$$

(by (F.50), (F.52), (F.54), (F.56))

- Inductive case [R-**def**]. We have:

$P \ = \ \mathbf{def}\ X(\widetilde{x}) = Q\ \mathbf{in}\ R$ (by inversion of the rule) (F.57)

$P' \ = \ \mathbf{def}\ X(\widetilde{x}) = Q\ \mathbf{in}\ R' \quad \text{with} \quad R \to R'$ (from rule premises) (F.58)

$$\frac{\Theta, X{:}S_1, \ldots, S_n \vdash Q \triangleright x_1{:}S_1, \ldots, x_n{:}S_n \triangleleft \emptyset \qquad \Theta, X{:}S_1, \ldots, S_n \vdash R \triangleright \Delta \triangleleft \Gamma}{\Theta \vdash P \triangleright \Delta \triangleleft \Gamma}\ [\text{T-}\mathbf{def}]$$

(by (F.57), (F.1), Lemma D.1, case [T-**def**]) (F.59)

$\exists \Delta' : \ \Delta \twoheadrightarrow^* \Delta' \ \text{and} \ \Theta, X{:}S_1, \ldots, S_n \vdash R' \triangleright \Delta' \triangleleft \Gamma$ (by (F.59), (F.58), i.h.) (F.60)

$$\frac{\Theta, X{:}S_1, \ldots, S_n \vdash Q \triangleright x_1{:}S_1, \ldots, x_n{:}S_n \triangleleft \emptyset \qquad \Theta, X{:}S_1, \ldots, S_n \vdash R' \triangleright \Delta' \triangleleft \Gamma}{\Theta \vdash P' \triangleright \Delta' \triangleleft \Gamma}\ [\text{T-}\mathbf{def}]$$

(by (F.60) and (F.58))

- Inductive case [R-$\equiv$]. We have:

$P \equiv Q \to Q' \equiv P' \ \text{(for some } Q, Q')$ (by inversion of the rule) (F.61)

$\Theta \vdash Q \triangleright \Delta \triangleleft \Gamma$ (by (F.1), (F.61) and Lemma D.5) (F.62)

$\exists \Delta' : \ \Delta \twoheadrightarrow^* \Delta' \ \text{and} \ \Theta \vdash Q' \triangleright \Delta' \triangleleft \Gamma$ (by (F.62), (F.61) and induction hyp.) (F.63)

$\Theta \vdash P' \triangleright \Delta' \triangleleft \Gamma$ (by (F.63), (F.61) and Lemma D.5)

We have thus proved (F.2). Then, we prove the main statement by induction on the number of reductions in $P \twoheadrightarrow^* P'$: the base case is trivial (0 reductions, and thus $P = P'$); in the inductive case ($n + 1$ reductions), we apply the induction hypothesis and (F.2). □

References

[1] D. Ancona, V. Bono, M. Bravetti, J. Campos, G. Castagna, P.-M. Deniélou, S.J. Gay, N. Gesbert, E. Giachino, R. Hu, E.B. Johnsen, F. Martins, V. Mascardi, F. Montesi, R. Neykova, N. Ng, L. Padovani, V.T. Vasconcelos, N. Yoshida, Behavioral types in programming languages, Foundations and Trends in Programming Languages 3 (2–3) (2017) 95–230.
[2] G. Bernardi, O. Dardha, S.J. Gay, D. Kouzapas, On duality relations for session types, in: Trustworthy Global Computing (TGC), 2014, pp. 51–66.
[3] G. Bernardi, M. Hennessy, Using higher-order contracts to model session types (extended abstract), in: CONCUR, in: Lect. Notes Comput. Sci., vol. 8704, Springer, 2014, pp. 387–401.
[4] G. Bernardi, M. Hennessy, Using higher-order contracts to model session types, Log. Methods Comput. Sci. 12 (2) (2016).
[5] L. Bettini, M. Coppo, L. D'Antoni, M. De Luca, M. Dezani-Ciancaglini, N. Yoshida, Global progress in dynamically interleaved multiparty sessions, in: CONCUR, 2008, pp. 418–433.
[6] L. Bocchi, J. Lange, N. Yoshida, Meeting deadlines together, in: CONCUR, in: LIPIcs. Leibniz Int. Proc. Inform., vol. 42, 2015, pp. 283–296.
[7] V. Bono, L. Padovani, Typing copyless message passing, Log. Methods Comput. Sci. 8 (1) (2012).
[8] D. Brand, P. Zafiropulo, On communicating finite-state machines, J. ACM 30 (2) (Apr. 1983) 323–342.
[9] L. Caires, F. Pfenning, Session types as intuitionistic linear propositions, in: CONCUR, 2010, pp. 222–236.
[10] L. Caires, F. Pfenning, B. Toninho, Linear logic propositions as session types, Math. Struct. Comput. Sci. 26 (3) (2016) 367–423.
[11] M. Coppo, M. Dezani-Ciancaglini, L. Padovani, N. Yoshida, A gentle introduction to multiparty asynchronous session types, in: Formal Methods for Multicore Programming, 2015, pp. 146–178.
[12] M. Coppo, M. Dezani-Ciancaglini, N. Yoshida, L. Padovani, Global progress for dynamically interleaved multiparty sessions, Math. Struct. Comput. Sci. 26 (2) (2016) 238–302.
[13] P. Deniélou, N. Yoshida, A. Bejleri, R. Hu, Parameterised multiparty session types, Log. Methods Comput. Sci. 8 (4) (2012).
[14] P.-M. Deniélou, N. Yoshida, Multiparty compatibility in communicating automata: characterisation and synthesis of global session types, in: ICALP, 2013, pp. 174–186.
[15] M. Dezani-Ciancaglini, S. Ghilezan, S. Jakšić, J. Pantović, N. Yoshida, Precise subtyping for synchronous multiparty sessions, in: PLACES 2015, vol. 203, 2016, pp. 29–43.
[16] S. Gay, A. Ravara (Eds.), Behavioural Types: From Theory to Tools, River Publishers Series in Automation, Control and Robotics, 2017.
[17] S.J. Gay, M. Hole, Subtyping for session types in the pi calculus, Acta Inform. 42 (2–3) (2005) 191–225.
[18] J.-Y. Girard, Linear logic, Theor. Comput. Sci. 50 (1987) 1–102.
[19] K. Honda, Types for dyadic interaction, in: CONCUR, 1993, pp. 509–523.
[20] K. Honda, V.T. Vasconcelos, M. Kubo, Language primitives and type discipline for structured communication-based programming, in: ESOP, 1998, pp. 122–138.
[21] K. Honda, N. Yoshida, M. Carbone, Multiparty asynchronous session types, in: POPL, 2008, pp. 273–284, full version: [22].
[22] K. Honda, N. Yoshida, M. Carbone, Multiparty asynchronous session types, J. ACM 63 (1) (2016) 9:1–9:67.
[23] H. Hüttel, I. Lanese, V.T. Vasconcelos, L. Caires, M. Carbone, P.-M. Deniélou, D. Mostrous, L. Padovani, A. Ravara, E. Tuosto, H.T. Vieira, G. Zavattaro, Foundations of session types and behavioural contracts, ACM Comput. Surv. 49 (1) (2016) 3:1–3:36.
[24] D. Kouzapas, N. Yoshida, Globally governed session semantics, in: CONCUR, 2013, pp. 395–409.
[25] D. Kouzapas, N. Yoshida, Globally governed session semantics, Log. Methods Comput. Sci. 10 (4) (2014).
[26] J. Lange, E. Tuosto, Synthesising choreographies from local session types, in: CONCUR, 2012, pp. 225–239.
[27] J. Lange, E. Tuosto, N. Yoshida, From communicating machines to graphical choreographies, in: POPL, 2015, pp. 221–232.
[28] S. Lindley, J.G. Morris, Talking bananas: structural recursion for session types, in: ICFP, 2016, pp. 434–447.
[29] R. Milner, Communicating and Mobile Systems: The π-Calculus, Cambridge University Press, 1999.
[30] R. Milner, J. Parrow, D. Walker, A calculus of mobile processes, parts I and II, Inf. Comput. 100 (1) (1992) 1–77.
[31] B.C. Pierce, Types and Programming Languages, MIT Press, MA, USA, 2002.
[32] D. Sangiorgi, D. Walker, The π-Calculus: A Theory of Mobile Processes, Cambridge University Press, 2001.
[33] A. Scalas, O. Dardha, R. Hu, N. Yoshida, A linear decomposition of multiparty sessions for safe distributed programming, in: ECOOP, 2017, pp. 24:1–24:31.
[34] A. Scalas, O. Dardha, R. Hu, N. Yoshida, A linear decomposition of multiparty sessions for safe distributed programming (artifact), Dagstuhl Artifacts Ser. 3 (1) (2017) 3:1–3:2.
[35] K. Takeuchi, K. Honda, M. Kubo, An interaction-based language and its typing system, in: PARLE, 1994, pp. 398–413.
[36] P. Wadler, Propositions as sessions, in: ICFP, 2012, pp. 273–286.
[37] P. Wadler, Propositions as sessions, J. Funct. Program. 24 (2–3) (2014) 384–418.
[38] N. Yoshida, P. Deniélou, A. Bejleri, R. Hu, Parameterised multiparty session types, in: FoSSaCS, 2010, pp. 128–145.