Less Is More: Multiparty Session Types Revisited[*]

Technical Report (first version: 09/01/2019; last update: 21/07/2021)

ALCESTE SCALAS, Imperial College London, UK
NOBUKO YOSHIDA, Imperial College London, UK

Multiparty Session Types (MPST) are a typing discipline ensuring that a message-passing process implements a given *multiparty session protocol*, without errors. In this paper, we propose a new, generalised MPST theory.

Our contribution is fourfold. *(1)* We demonstrate that a revision of the theoretical foundations of MPST is *necessary*: classic MPST have a limited *subject reduction* property, with inherent restrictions that are easily overlooked, and in previous work have led to flawed type safety proofs; our new theory removes such restrictions and fixes such flaws. *(2)* We contribute a new MPST theory that is ***less*** complicated, and yet ***more*** general, than the classic one: it does *not* require *global multiparty session types* nor *binary session type duality* — instead, it is grounded on general behavioural type-level properties, and proves type safety of many more protocols and processes. *(3)* We produce a detailed analysis of type-level properties, showing how, in our new theory, they allow to ensure decidability of type checking, and statically guarantee that processes enjoy, e.g., deadlock-freedom and liveness at run-time. *(4)* We show how our new theory can integrate type and model checking: type-level properties can be expressed in modal μ-calculus, and verified with well-established tools.

CCS Concepts: • **Theory of computation** → **Process calculi**; **Type structures**; *Verification by model checking*;

Additional Key Words and Phrases: session types, duality, deadlock-freedom, liveness

1 INTRODUCTION

Session types are a type-based framework for formalising structured communication protocols, and verifying them in concurrent message-passing programs. The original *binary session types* theory [Honda et al. 1998] addresses protocols with two participants (e.g., client and server), and is built on a notion of *duality* in interactions, inspired by linear logic [Girard 1987]; this has led to several studies on the logical foundations for session types, e.g. [Caires et al. 2016; Wadler 2014]. This approach was later generalised to *multiparty* sessions [Bettini et al. 2008; Honda et al. 2008], supporting more sophisticated protocols with *any* number of participants (two or more); correspondingly, binary duality was generalised as multiparty *consistency*, leading to studies on its logical foundations [Caires and Pérez 2016; Carbone et al. 2016, 2015].

Unfortunately, this duality-based framework has intrinsic ***limitations***: the consistency requirement is not satisfied by many multiparty protocols — even surprisingly simple ones. Such limitations are subtle: in this paper, we show that they have been overlooked or wrongly bypassed in several previous works, leading to MPST extensions that are ***no longer correct***, and have flawed *subject reduction* proofs. Then, we provide a solution: a new, generalised MPST theory that subsumes classic MPST under a new theoretical foundation, removes its limitations, fixes the aforementioned flaws, and supports a richer set of multiparty protocols and processes.

The Multiparty Session Types (MPST) framework. Bettini et al. [2008]; Honda et al. [2008] introduce the seminal notion of *global types*, which describe multiparty conversations from a global perspective. MPST verification follows a top-down approach based on *endpoint projections*:

[*]This is the long version of the POPL'19 paper with the same title, and by the same authors: https://doi.org/10.1145/3290343

Authors' addresses: Alceste Scalas, Imperial College London, UK, alceste.scalas@imperial.ac.uk; Nobuko Yoshida, Imperial College London, UK, n.yoshida@imperial.ac.uk.

(1) a *multiparty protocol* is formalised as a *global type* G, providing a bird's eye view on the interactions between two or more *roles*;
(2) G is *projected* onto a set of *endpoint (local) session types* (one per role); and
(3) session types are assigned to *communication channels*, used by *MPST processes* that can be written and type-checked separately.

E.g., the global type G below models a protocol (based on OAuth 2.0 [OAuth Working Group 2012]) between service **s**, client **c**, and authorisation server **a**:

$$G = \mathbf{s}{\to}\mathbf{c}{:}\left\{\begin{array}{l}\mathtt{login}\,.\,\mathbf{c}{\to}\mathbf{a}{:}\mathtt{passwd}(\mathrm{Str})\,.\,\mathbf{a}{\to}\mathbf{s}{:}\mathtt{auth}(\mathrm{Bool})\,.\,\mathbf{end}\,,\\ \mathtt{cancel}\,.\,\mathbf{c}{\to}\mathbf{a}{:}\mathtt{quit}\,.\,\mathbf{end}\end{array}\right\} \tag{1}$$

The protocol of G says that the **s**ervice sends to the **c**lient *either* a request to `login`, *or* `cancel`; in the first case, **c** continues by sending `passwd` (carrying a String) to the **a**uthorisation server, who in turn sends `auth` to **s** (with a Boolean, telling whether the client is authorised), and the session **end**s; in the second case, **c** sends `quit` to **a**, and the session **end**s. The *projections of* G describe the local I/O actions (i.e., the interfaces) that programs must implement to play the roles in G:

$$S_{\mathbf{s}} = \mathbf{c}{\oplus}\left\{\begin{array}{l}\mathtt{login}.\mathbf{a}\&\mathtt{auth}(\mathrm{Bool})\,,\\ \mathtt{cancel}\end{array}\right\} \quad S_{\mathbf{c}} = \mathbf{s}\&\left\{\begin{array}{l}\mathtt{login}.\mathbf{a}{\oplus}\mathtt{passwd}(\mathrm{Str})\,,\\ \mathtt{cancel}.\mathbf{a}{\oplus}\mathtt{quit}\end{array}\right\} \quad S_{\mathbf{a}} = \mathbf{c}\&\left\{\begin{array}{l}\mathtt{passwd}(\mathrm{Str}).\mathbf{s}{\oplus}\mathtt{auth}(\mathrm{Bool})\,,\\ \mathtt{quit}\end{array}\right\} \tag{2}$$

Here, $S_{\mathbf{s}}$, $S_{\mathbf{c}}$, $S_{\mathbf{a}}$ are *session types*, obtained by projecting G resp. onto **s**, **c**, **a** (for brevity, we omit final **end**s). $S_{\mathbf{s}}$ represents the interface of **s** in G: it must send ($\oplus$) to **c** either `login` or `cancel`; in the first case, **s** must then receive ($\&$) message `auth(Bool)` from **a**, and the session ends; otherwise, in the second case, the session just ends. Types $S_{\mathbf{c}}$ and $S_{\mathbf{a}}$ follow the same intuition. The multiparty session type system assigns the types in (2) to *channels*, and checks that *endpoint programs* use them correctly: e.g., the program implementing the **s**ervice is checked against $S_{\mathbf{s}}$, and the programs implementing **c**/**a** against $S_{\mathbf{c}}$/$S_{\mathbf{a}}$. Endpoint programs, in turn, are formalised as *processes* in a π-calculus extended with multiparty communication primitives. Variations of this framework have been implemented in numerous programming languages (surveyed in Ancona et al. [2017]; Gay and Ravara [2017]), allowing to develop distributed applications with guaranteed protocol conformance.

Limitations and Theoretical Issues of MPST. Theories and implementations based on MPST crucially require "correct by construction" protocols that do not cause deadlocks nor communication errors when endpoint programs interact. This is achieved by imposing *well-formedness* conditions to global types, and *consistency* restrictions when processes are type-checked.

However, such restrictions introduce rather serious problems when proving *subject reduction* — i.e., when proving that typed processes only reduce to typed processes, and thus, no (untypable) error state can be reached (*"typed processes never go wrong"*). Usually, one expects a statement like:

$$\Gamma \vdash P \;\text{ and }\; P \to P' \quad \text{implies} \quad \exists\Gamma' : \;\Gamma' \vdash P' \tag{3}$$

where $\Gamma \vdash P$ is a typing judgement stating that process P abides by the typing context Γ, which can map, e.g., the communication channels $c_{\mathbf{s}}, c_{\mathbf{c}}, c_{\mathbf{a}}$ to the types $S_{\mathbf{s}}, S_{\mathbf{c}}, S_{\mathbf{a}}$ in (2).

Unfortunately, (3) *is wrong*. If we take Γ without any constraint as in (3), it might contain types like $\mathbf{c}{\oplus}\mathtt{m}(\mathrm{Str}).\mathbf{end}$ and $\mathbf{s}\&\mathtt{m}(\mathrm{Int}).\mathbf{end}$, and they could type a parallel process $P = P_1 \mid P_2$, where P_1 and P_2 interact according to the types, with P_1 sending a message `m("Hello")` (carrying a String), and P_2 receiving `m` but using its payload as an Integer. In this case, P would reduce to a "wrong" and untypable P' (see also [Coppo et al. 2015a, p. 163], and §3 later on): this means that (3) does *not* hold. For this reason, the MPST theory requires the aforementioned *consistency* restriction, and its actual subject reduction statement reads:

$$\Gamma \vdash P \;\underline{\textit{with } \Gamma \textit{ consistent}}\; \text{ and } P \to P' \quad \text{implies} \quad \exists\Gamma' \;\underline{\textit{consistent}}\text{:}\; \Gamma \to^* \Gamma' \text{ and } \Gamma' \vdash P' \tag{4}$$

(where $\Gamma \to^* \Gamma'$ denotes typing context reductions). Consistency is a syntactic constraint ensuring that the potential output messages of each role match the input capabilities of their recipient; as noted above, this requirement was developed by generalising the notion of *binary session duality* [Honda et al. 1998]. However, due to this binary session heritage, multiparty consistency is:

(1) **overly restrictive.** Consistency does *not* hold for many protocols: even the simple authorisation protocol in (1)/(2) above is *not* consistent. Hence, for such protocols, the MPST framework cannot prove type safety of *any* process, because (4) holds vacuously;

(2) **inflexible and error-prone.** Some MPST works, e.g. [Deniélou et al. 2012; Deniélou and Yoshida 2012; Yoshida et al. 2010], propose richer global types with flexible well-formedness conditions — but either overlook the consistency requirement, or fail to realise that their extensions do *not* satisfy it. Hence, their subject reduction theorems do not hold (like (3)), or hold vacuously (as above); and worryingly, such results are reused in later works and implementations (more details in § 8).

These two claims are based on technical arguments, that we develop in § 3. They clearly undermine the expressiveness and applicability of MPST: when the theory cannot ensure type safety for a given protocol, MPST-based implementations should either reject it (thus being overly restrictive), or forfeit the guaranteed absence of run-time errors. To solve these problems, we pose the questions:

Can we remove the duality/consistency requirements of MPST?
Can we use, instead, more flexible properties of session types, thus enlarging the subject reduction property, and the set of provably type-safe processes?

To answer positively, we need a new MPST theory that is *not* rooted in binary session duality — but has more general foundations, that still support duality as a special case.

Contributions. We present a *new theory of multiparty session types.* Its novel theoretical foundations leverage a weak *behavioural safety* invariant that, for the first time, eschews the limitations of duality/consistency, and allows to obtain much more general results than classic MPST.

We summarise MPST definitions and typing rules in § 2, highlighting where our new theory diverges from the classic (§ 2.3): i.e., when establishing the prerequisites for proving type safety.

(1) We explain how classic MPST establish such prerequisites: i.e., by imposing consistency/duality. We uncover that the resulting severe limitations lead to subtle theoretical issues (§ 3).
(2) We present our new MPST theory (§ 4), with a much weaker prerequisite: a *safety* invariant, *not* depending on global types, *nor* needing projection/duality/consistency from classic MPST.
(3) By removing consistency, we rebuild the theoretical foundations of MPST on a more general basis. Our rebuilding subsumes classic MPST works, and fixes their theoretical issues, by producing more general typing rules, with just small visible differences (Remark 5.12).
(4) We design our new type system to be parametric: its safety invariant is abstracted as a parameter φ. We show that φ can be fine-tuned to ensure decidability of type-checking, and statically enforce various run-time properties on processes — e.g., liveness (§ 5.3, § 5.4, § 5.5).
(5) The parameter φ can be a *behavioural* property: this allows for a novel integration of type/model checking techniques for MPST. We show how to express φ as a modal μ-calculus formula, and verify type-level properties via model checking, using the paper's companion artifact (§ 6). Via point 4 above, the model-checked properties transfer to processes.
(6) Our theory extends to *asynchronous* communication, to handle richer protocols and programs. Asynchrony makes φ (and type checking) undecidable; still, we present various ways to achieve decidable type checking, with methods based e.g. on communicating automata (§ 7).

NOTE: *The main difference between this technical report the conference paper [Scalas and Yoshida 2019] are the appendices, that contain technical details, proofs, and discussion on related work: we provide some pointers in the main text. In particular, in § H, we discuss more related topics (e.g., asynchronous subtyping), and other theories that have different goals, or cannot handle our examples (conversation types, choreographic programming).*

2 MULTIPARTY SESSION TYPES

This section describes the multiparty session π-calculus (§ 2.1), its types, and typing rules (§ 2.2). Our streamlined formulation is based on Coppo et al. [2015a] and Scalas et al. [2017a], i.e., the most common in literature; we include subtyping [Dezani-Ciancaglini et al. 2015], to later study its crucial influence on the behavioural properties of types and processes (§ 5).

Crucially, in this section we leave one typing rule under-specified: the rule for session restriction. The reason is explained in § 2.3: the exact form of this rule strictly depends on the theoretical foundations that allow to prove type safety — and the choice of such foundations is the crossroads where our new theory (§ 4) departs from classic MPST (§ 3).

2.1 The Multiparty Session π-Calculus

The multiparty session π-calculus models processes that interact via *multiparty channels*. We give a streamlined definition, sufficient for our developments. Extensions with, e.g., ground values (booleans, strings,...), or conditionals, are standard and orthogonal; we use them in examples.

Definition 2.1. The **multiparty session π-calculus** syntax is defined as follows:

$$\begin{array}{rcll}
c, d & ::= & x \;\big|\; s[\mathbf{p}] & \text{(variable, channel with role } \mathbf{p}\text{)} \\
P, Q & ::= & \mathbf{0} \;\big|\; P \mid Q \;\big|\; (\nu s)\, P & \text{(inaction, composition, restriction)} \\
 & & c[\mathbf{q}] \oplus \mathtt{m}\langle d \rangle . P & \text{(selection towards role } \mathbf{q}\text{)} \\
 & & c[\mathbf{q}] \sum_{i \in I} \mathtt{m}_i(x_i) . P_i & \text{(branching from role } \mathbf{q} \text{ with } I \neq \emptyset\text{)} \\
 & & \mathbf{def}\ D\ \mathbf{in}\ P \;\big|\; X\langle \widetilde{c} \rangle \;\big|\; \mathtt{err} & \text{(process definition, process call, error)} \\
D & ::= & X(\widetilde{x}) = P & \text{(declaration of process variable } X\text{)}
\end{array}$$

Restriction, branching and declarations act as binders, as expected; $\mathrm{fc}(P)$ is the set of *free channels with roles* in P, and $\mathrm{fv}(P)$ is the set of *free variables* in P. We adopt a form of Barendregt convention: bound sessions and process variables are assumed pairwise distinct, and different from free ones.

A **channel** c can be either a variable or a **channel with role** $s[\mathbf{p}]$, i.e., a multiparty communication endpoint whose user plays role $\mathbf{p}$ in the session s. The **inaction 0** represents a terminated process (and is often omitted). The **parallel composition** $P \mid Q$ represents two processes that can execute concurrently, and potentially communicate. The **session restriction** $(\nu s)\, P$ declares a new session s with scope limited to process P. Process $c[\mathbf{q}] \oplus \mathtt{m}\langle d \rangle . P$ performs a **selection (internal choice)** towards role $\mathbf{q}$, using the channel c: the *message label* $\mathtt{m}$ is sent with the *payload* channel d, and the execution continues as P. Dually, the **branching (external choice)** $c[\mathbf{q}] \sum_{i \in I} \mathtt{m}_i(x_i) . P_i$ uses channels c to wait for a message from role $\mathbf{q}$: if a message label $\mathtt{m}_k$ with payload d is received (for some $k \in I$), then the execution continues as P_k, with x_k replaced by d. Note that variable x_i is bound with scope P_i. **Process definition def** $X(\widetilde{x}) = P$ **in** Q and **process call** $X\langle \widetilde{c} \rangle$ model recursion: the call invokes X by expanding it into P, and replacing its formal parameters with the actual ones. $\mathtt{err}$ denotes the **error process**. Note that our simplified syntax does not have "pure" input/output prefixes: they can be easily encoded as singleton branch/selection.

Definition 2.2 (Semantics). A **reduction context** $\mathbb{C}$ is: $\mathbb{C} ::= \mathbb{C} \mid P \;\big|\; (\nu s)\, \mathbb{C} \;\big|\; \mathbf{def}\ D\ \mathbf{in}\ \mathbb{C} \;\big|\; [\,]$ **Reduction** $\to$ is inductively defined in Fig. 1, up-to a standard **structural congruence** $\equiv$ (§ A) including α-conversion. We say that P **has an error** iff, for some $\mathbb{C}$, $P = \mathbb{C}[\mathtt{err}]$.

$$
\begin{array}{rl}
\text{[R-Comm]} & s[\mathtt{p}][\mathtt{q}]\sum_{i\in I}\mathtt{m}_i(x_i).P_i \mid s[\mathtt{q}][\mathtt{p}]\oplus\mathtt{m}_k\langle s'[\mathtt{r}]\rangle.Q \;\to\; P_k\{s'[\mathtt{r}]/x_k\} \mid Q \quad \text{if } k\in I \\
\text{[R-}X\text{]} & \mathbf{def}\ X(x_1,\dots,x_n)=P\ \mathbf{in}\ (X\langle s_1[\mathtt{p}_1],\dots,s_n[\mathtt{p}_n]\rangle \mid Q) \\
& \quad\to\ \mathbf{def}\ X(x_1,\dots,x_n)=P\ \mathbf{in}\ (P\{s_1[\mathtt{p}_1]/x_1\}\cdots\{s_n[\mathtt{p}_n]/x_n\} \mid Q) \\
\text{[R-Ctx]} & P\to P' \text{ implies } \mathbb{C}[P]\to\mathbb{C}[P'] \\
\text{[R-Err]} & s[\mathtt{p}][\mathtt{q}]\sum_{i\in I}\mathtt{m}_i(x_i).P_i \mid s[\mathtt{q}][\mathtt{p}]\oplus\mathtt{m}\langle s'[\mathtt{r}]\rangle.Q \;\to\; \mathtt{err} \quad \text{if } \forall i\in I:\mathtt{m}_i\neq\mathtt{m}
\end{array}
$$

Fig. 1. MPST π-calculus semantics, defined up-to standard structural congruence *(details: §A)*.

In Def. 2.2, the **reduction context** $\mathbb{C}$ defines a process with a single hole $[\,]$, occurring in place of some subterm P. The **communication rule** [R-Comm] says that the parallel composition of a branching and a selection process, both operating on the same session s respectively as roles $\mathtt{p}$ and $\mathtt{q}$, reduces to the corresponding continuations, with the sent channel being substituted on the receiver side. The **process call rule** [R-X] allows to invoke the process P in the definition of X by creating a copy of P, and replacing the formal parameters x_i with actual parameters, i.e., channels with role $s_i[\mathtt{p}_i]$. The standard **context rule** [R-Ctx] says that reduction can happen under parallel composition, restriction and process definition (cf. definition of $\mathbb{C}$). Finally, the **error rule** [R-Err] says that a parallel composition of mismatching selection and branching processes reduces to $\mathtt{err}$: intuitively, it models a scenario where a process implementing role $\mathtt{q}$ is trying to send $\mathtt{m}$ to another process implementing $\mathtt{p}$ — who is indeed waiting for an input, but does not expect to receive $\mathtt{m}$.

Example 2.3. The following process interacts on session s using channels with role $s[\mathtt{s}]$, $s[\mathtt{c}]$, $s[\mathtt{a}]$, to play resp. roles $\mathtt{s}$, $\mathtt{c}$, $\mathtt{a}$. For brevity, we omit irrelevant message payloads.

$$
(\nu s)\,(P_{\mathtt{s}} \mid P_{\mathtt{c}} \mid P_{\mathtt{a}}) \qquad \text{where: } \begin{cases} P_{\mathtt{s}} = s[\mathtt{s}][\mathtt{c}]\oplus\mathtt{cancel} \\ P_{\mathtt{c}} = s[\mathtt{c}][\mathtt{s}]\sum\{\mathtt{login}.s[\mathtt{c}][\mathtt{a}]\oplus\mathtt{passwd}\langle\texttt{"XYZ"}\rangle\ ,\ \mathtt{cancel}.s[\mathtt{c}][\mathtt{a}]\oplus\mathtt{quit}\} \\ P_{\mathtt{a}} = s[\mathtt{a}][\mathtt{c}]\sum\{\mathtt{passwd}(y).s[\mathtt{a}][\mathtt{s}]\oplus\mathtt{auth}\langle y=\texttt{"secret"}\rangle\ ,\ \mathtt{quit}\} \end{cases}
$$

Here, $(\nu s)\,(P_{\mathtt{s}} \mid P_{\mathtt{c}} \mid P_{\mathtt{a}})$ is the parallel composition of processes $P_{\mathtt{s}}$, $P_{\mathtt{c}}$, $P_{\mathtt{a}}$ in the scope of session s. In $P_{\mathtt{s}}$, "$s[\mathtt{s}][\mathtt{c}]\oplus\mathtt{cancel}$" means: use $s[\mathtt{s}]$ to send $\mathtt{cancel}$ to $\mathtt{c}$. Process $P_{\mathtt{c}}$ uses $s[\mathtt{c}]$ to receive $\mathtt{login}$ or $\mathtt{cancel}$ from $\mathtt{s}$; then, in the first case it uses $s[\mathtt{c}]$ to send $\mathtt{passwd}$ to $\mathtt{a}$; in the second case, it uses $s[\mathtt{c}]$ to send $\mathtt{quit}$ to $\mathtt{a}$. By Def. 2.2, we have the reductions:

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

2.2 Types, Subtypes, and Typing

Session types (Def. 2.4) describe the intended use of communication channels in the MPST π-calculus (Def. 2.1); channels are mapped to their respective type by session typing contexts (Def. 2.6).

Definition 2.4. The syntax of **multiparty session types** is:

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

We require types to be closed, and recursion variables to be guarded.

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 $\mathtt{p}$, where $\mathtt{m}_i$ are *message labels* and S_i are *message payload types*; 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, requires to use a channel to perform one output $\mathtt{m}_i(S_i)$ towards $\mathtt{p}$, for some $i\in I$, and continue using the channel according to S'_i. Type $\mathbf{end}$ describes a **terminated** channel allowing no further inputs/outputs. Type $\mu\mathbf{t}.S$ models **recursion**: μ binds the **recursion variable** $\mathbf{t}$ in S. The guardedness requirement ensures

$$\frac{\Theta(X) = S_1,\ldots,S_n}{\Theta \vdash X{:}S_1,\ldots,S_n}\;[\text{T-}X] \qquad \frac{S \leqslant S'}{c{:}S \vdash c{:}S'}\;[\text{T-Sub}] \qquad \frac{\forall i \in 1..n \quad c_i{:}S_i \vdash c_i{:}\mathbf{end}}{\mathrm{end}(c_1{:}S_1,\ldots,c_n{:}S_n)}\;[\text{T-end}]$$

$$\frac{\mathrm{end}(\Gamma)}{\Theta \cdot \Gamma \vdash \mathbf{0}}\;[\text{T-}\mathbf{0}] \qquad \frac{\Theta, X{:}S_1,\ldots,S_n \cdot x_1{:}S_1,\ldots,x_n{:}S_n \vdash P \qquad \Theta, X{:}S_1,\ldots,S_n \cdot \Gamma \vdash Q}{\Theta \cdot \Gamma \vdash \mathbf{def}\ X(x_1{:}S_1,\ldots,x_n{:}S_n) = P\ \mathbf{in}\ Q}\;[\text{T-}\mathbf{def}]$$

$$\frac{\Theta \vdash X{:}S_1,\ldots,S_n \quad \mathrm{end}(\Gamma_0) \quad \forall i \in 1..n \quad \Gamma_i \vdash c_i{:}S_i}{\Theta \cdot \Gamma_0,\Gamma_1,\ldots,\Gamma_n \vdash X\langle c_1,\ldots,c_n\rangle}\;[\text{T-}X]$$

$$\frac{\Gamma_1 \vdash c{:}\mathbf{q}\&_{i\in I}\mathtt{m}_i(S_i).S'_i \quad \forall i \in I \quad \Theta \cdot \Gamma, y_i{:}S_i, c{:}S'_i \vdash P_i}{\Theta \cdot \Gamma, \Gamma_1 \vdash c[\mathbf{q}]\sum_{i\in I}\mathtt{m}_i(y_i).P_i}\;[\text{T-}\&]$$

$$\frac{\Gamma_1 \vdash c{:}\mathbf{q}\oplus\mathtt{m}(S).S' \quad \Gamma_2 \vdash d{:}S \quad \Theta \cdot \Gamma, c{:}S' \vdash P}{\Theta \cdot \Gamma, \Gamma_1, \Gamma_2 \vdash c[\mathbf{q}]\oplus\mathtt{m}\langle d\rangle.P}\;[\text{T-}\oplus] \qquad \frac{\Theta \cdot \Gamma_1 \vdash P_1 \qquad \Theta \cdot \Gamma_2 \vdash P_2}{\Theta \cdot \Gamma_1, \Gamma_2 \vdash P_1 \mid P_2}\;[\text{T-}|]$$

$$\frac{\Gamma' = \left\{s[\mathbf{p}]{:}S_{\mathbf{p}}\right\}_{\mathbf{p}\in I} \quad \varphi(\Gamma') \quad s \notin \Gamma \quad \Theta \cdot \Gamma, \Gamma' \vdash P}{\Theta \cdot \Gamma \vdash (\nu s{:}\Gamma')\,P}\;[\text{T-}\nu] \qquad \text{where } \varphi \text{ is a typing context property}$$

Fig. 2. Multiparty session typing rules. Rule [T-ν] for session restriction is discussed in §2.3.

that recursive types are *contractive*: i.e., in $\mu\mathbf{t}.S$ we have $S \neq \mathbf{t}'$ for all $\mathbf{t}'$. For brevity, we often omit the trailing **end** in types, and **end**-typed message payloads: e.g., $\mathbf{p}\oplus\mathtt{m}$ stands for $\mathbf{p}\oplus\mathtt{m}(\mathbf{end}).\mathbf{end}$.

In Def. 2.5 below, we define the *multiparty session subtyping* relation [Dezani-Ciancaglini et al. 2015].[1] Intuitively, Def. 2.5 says that a type S is smaller than S' when S is "less demanding" than S' — i.e., when S imposes to support less external choices and allows to perform more internal choices. Session subtyping is used in the type system to augment its flexibility.

Definition 2.5. The **session subtyping** $\leqslant$ is *co*inductively defined:

$$\frac{\forall i \in I \quad S_i \leqslant T_i \quad S'_i \leqslant T'_i}{\mathbf{p}\&_{i\in I}\mathtt{m}_i(S_i).S'_i \leqslant \mathbf{p}\&_{i\in I\cup J}\mathtt{m}_i(T_i).T'_i}\;[\text{Sub-}\&] \qquad \frac{\forall i \in I \quad T_i \leqslant S_i \quad S'_i \leqslant T'_i}{\mathbf{p}\oplus_{i\in I\cup J}\mathtt{m}_i(S_i).S'_i \leqslant \mathbf{p}\oplus_{i\in I}\mathtt{m}_i(T_i).T'_i}\;[\text{Sub-}\oplus]$$

$$\frac{}{\mathbf{end} \leqslant \mathbf{end}}\;[\text{Sub-}\mathbf{end}] \qquad \frac{S\{\mu\mathbf{t}.S/\mathbf{t}\} \leqslant T}{\mu\mathbf{t}.S \leqslant T}\;[\text{Sub-}\mu\text{L}] \qquad \frac{S \leqslant T\{\mu\mathbf{t}.T/\mathbf{t}\}}{S \leqslant \mu\mathbf{t}.T}\;[\text{Sub-}\mu\text{R}]$$

In Def. 2.5, rules [Sub-&]/[Sub-$\oplus$] define **subtyping on branch/select types**: [Sub-&] is covariant in both the carried types and in the number of branches, whereas [Sub-$\oplus$] is contravariant in both: this formalises the intuition of a smaller type having less external choices, and more internal choices. By rule [Sub-**end**], **end** is only subtype of itself. The **recursion rules** [Sub-μL]/[Sub-μR] relate types up-to their unfoldings, as usual for coinductive subtyping [Pierce 2002, Ch. 21].

Definition 2.6 (Typing Contexts). Θ denotes a partial mapping from process variables to n-tuples of types, and Γ denotes a partial mapping from channels to types, defined as:

$$\Theta \;::=\; \emptyset \;\big|\; \Theta,\, X{:}S_1,\ldots,S_n \qquad \Gamma \;::=\; \emptyset \;\big|\; \Gamma, x{:}S \;\big|\; \Gamma, s[\mathbf{p}]{:}S$$

The *composition* Γ_1, Γ_2 is defined iff $\mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2) = \emptyset$.
We write $s \notin \Gamma$ iff $\forall \mathbf{p} : s[\mathbf{p}] \notin \mathrm{dom}(\Gamma)$ (i.e., session s does not occur in Γ).
We write $\mathrm{dom}(\Gamma) = \{s\}$ iff $\forall c \in \mathrm{dom}(\Gamma)$ there is $\mathbf{p}$ such that $c = s[\mathbf{p}]$ (i.e., Γ only contains session s).
We write $\Gamma \leqslant \Gamma'$ iff $\mathrm{dom}(\Gamma) = \mathrm{dom}(\Gamma')$ and $\forall c \in \mathrm{dom}(\Gamma) : \Gamma(c) \leqslant \Gamma'(c)$.

[1]Our $\leqslant$ is inverted w.r.t. the "process-oriented" subtyping of Dezani-Ciancaglini et al. [2015] because, for convenience, we use the "channel-oriented" order of Gay and Hole [2005]; Scalas et al. [2017a]. For a thorough comparison, see [Gay 2016].

The type system uses two kinds of typing contexts: Θ to assign an n-tuple of types to each process variable X (one type per argument), and Γ to map variables and channels with roles to session types. Together, they are used in judgements of the following form:

$$\Theta \cdot \Gamma \vdash P \qquad \text{(with } \Theta \text{ omitted when empty)} \tag{5}$$

meaning: "given the process types in Θ, P uses its variables and channels *linearly* according to Γ."

The **typing judgement** (5) is inductively defined by the rules in Fig. 2. For convenience, we type-annotate channels bound by process definitions and restrictions.

The first three rules in Fig. 2 define auxiliary judgements. By [T-X], $\Theta \vdash X{:}S_1, ..., S_n$ holds if Θ maps X to an n-tuple of types $S_1, ..., S_n$. By [T-Sub], $\Gamma \vdash c{:}S'$ holds if Γ only contains *one* entry $c{:}S$ with $S \leqslant S'$: i.e., when typing processes, [T-Sub] allows to use a channel of type S whenever a channel with a larger type S' is needed, as per Liskov and Wing [1994]'s substitution principle; note that Def. 2.5 relates types up-to unfolding, hence [T-Sub] makes the type system *equi-recursive* [Pierce 2002, Ch. 21]. Finally, end(Γ) holds if Γ's entries are **end**-typed (under [T-Sub]).

The other rules in Fig. 2 define the process typing judgement in (5). The **termination rule** [T-**0**] says that **0** is typed if all channels in Γ are **end**-typed. By the **process definition rule** [T-**def**], **def** $X(\widetilde{x}) = P$ **in** Q is typed if P uses the arguments $x_1, ..., x_n$ according to $S_1, ..., S_n$, and the latter is the type of X when typing both P and Q: this means that P can refer to X, and this allows to type recursive processes. By the **process call rule** [T-X], $X\langle\widetilde{c}\rangle$ is typed if the types of $\widetilde{c}$ match those of the formal parameters of X, and any unused channel (in Γ_0) is **end**-typed: this preserves linearity by ensuring that channels requiring more inputs/outputs cannot be forgotten. By the **branching rule** [T-&], $c[\mathbf{q}]\sum_{i\in I}\mathtt{m}_i(y_i).P_i$ is typed if c has type S, where S is an external choice from $\mathbf{q}$, with the same branching labels $\mathtt{m}_i$. The **selection rule** [T-$\oplus$] says that $c[\mathbf{q}]\oplus\mathtt{m}\langle d\rangle.P$ is typed if c has type S, where S is an internal choice towards $\mathbf{q}$ with message label $\mathtt{m}$. By the **parallel rule** [T-|], two parallel processes are typed by splitting the context in the premises. The **session restriction rule** [T-ν] deserves special attention: we discuss it in §2.3.

Example 2.7. Take the processes from Ex. 2.3, and the types $S_{\mathbf{s}}$, $S_{\mathbf{c}}$, $S_{\mathbf{a}}$ from § 1, eq. (2). With the rules in Fig. 2, we have the following typing derivation:

$$\dfrac{\dfrac{\dfrac{\vdots}{s[\mathbf{s}]{:}S_{\mathbf{s}} \vdash P_{\mathbf{s}}} \quad \dfrac{\vdots}{s[\mathbf{c}]{:}S_{\mathbf{c}} \vdash P_{\mathbf{c}}}}{s[\mathbf{s}]{:}S_{\mathbf{s}}, s[\mathbf{c}]{:}S_{\mathbf{c}} \vdash P_{\mathbf{s}} \mid P_{\mathbf{c}}}\,[\text{T-}|] \quad \dfrac{\vdots}{s[\mathbf{a}]{:}S_{\mathbf{a}} \vdash P_{\mathbf{a}}}}{\Gamma \vdash P_{\mathbf{s}} \mid P_{\mathbf{c}} \mid P_{\mathbf{a}}}\,[\text{T-}|] \qquad \text{where } \Gamma = s[\mathbf{s}]{:}S_{\mathbf{s}}, s[\mathbf{c}]{:}S_{\mathbf{c}}, s[\mathbf{a}]{:}S_{\mathbf{a}}$$

The process $P_{\mathbf{s}} \mid P_{\mathbf{c}} \mid P_{\mathbf{a}}$ is typed by rule [T-|], that splits the typing context linearly ensuring that a channel is not used by two parallel sub-processes. In the omitted part of the derivation, processes $P_{\mathbf{s}}$, $P_{\mathbf{c}}$, $P_{\mathbf{a}}$ are typed separately, using rules [T-$\oplus$]/[T-&]: each process uses one of the channels with role $s[\mathbf{s}]$, $s[\mathbf{c}]$, $s[\mathbf{a}]$, according to the type $S_{\mathbf{s}}$, $S_{\mathbf{c}}$, $S_{\mathbf{a}}$, respectively.

We conclude with the transitions/reductions of typing contexts (Def. 2.8): intuitively, they abstract the message exchanges that might occur over typed channels. We adopt a standard formulation, with two adaptations: we compare payloads using $\leqslant$ (to cater for subtyping), and we specify transition labels for inputs, outputs, and communication.

Definition 2.8. Let α have the form $s{:}\mathbf{p}\&\mathbf{q}{:}\mathtt{m}(S)$, or $s{:}\mathbf{p}\oplus\mathbf{q}{:}\mathtt{m}(S)$, or $s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}$ (for any roles $\mathbf{p}$, $\mathbf{q}$, message label $\mathtt{m}$, and type S). The *typing context transition* $\xrightarrow{\alpha}$ is inductively defined by the rules:

$$\frac{k \in I}{s[\mathtt{p}]{:}\mathtt{q}{\oplus_{i\in I}}\mathtt{m}_i(S_i).S'_i \xrightarrow{s:\mathtt{p}\oplus\mathtt{q}:\mathtt{m}_k(S_k)} S'_k}\ [\Gamma\text{-}\oplus] \qquad \frac{k \in I}{s[\mathtt{p}]{:}\mathtt{q}\&_{i\in I}\mathtt{m}_i(S_i).S'_i \xrightarrow{s:\mathtt{p}\&\mathtt{q}:\mathtt{m}_k(S_k)} S'_k}\ [\Gamma\text{-}\&]$$

$$\frac{\Gamma_1 \xrightarrow{s:\mathtt{p}\oplus\mathtt{q}:\mathtt{m}(S)} \Gamma'_1 \quad \Gamma_2 \xrightarrow{s:\mathtt{q}\&\mathtt{p}:\mathtt{m}(T)} \Gamma'_2 \quad S \leqslant T}{\Gamma_1, \Gamma_2 \xrightarrow{s:\mathtt{p},\mathtt{q}:\mathtt{m}} \Gamma'_1, \Gamma'_2}\ [\Gamma\text{-}\textsc{Comm}] \quad \frac{\Gamma, c{:}S\{\mu t.S/t\} \xrightarrow{\alpha} \Gamma'}{\Gamma, c{:}\mu t.S \xrightarrow{\alpha} \Gamma'}\ [\Gamma\text{-}\mu] \quad \frac{\Gamma \xrightarrow{\alpha} \Gamma'}{\Gamma, c{:}S \xrightarrow{\alpha} \Gamma', c{:}S}\ [\Gamma\text{-}\textsc{Cong}]$$

We write $\Gamma\xrightarrow{\alpha}$ iff $\Gamma\xrightarrow{\alpha}\Gamma'$ for some Γ'. The *reduction* $\Gamma\rightarrow\Gamma'$ is defined iff $\Gamma \xrightarrow{s:\mathtt{p},\mathtt{q}:\mathtt{m}} \Gamma'$ for some $s, \mathtt{p}, \mathtt{q}, \mathtt{m}$. We write $\Gamma\rightarrow$ iff $\Gamma\rightarrow\Gamma'$ for some Γ', and $\Gamma\not\rightarrow$ for its negation (i.e., when there is no Γ' such that $\Gamma\rightarrow\Gamma'$). We define $\rightarrow^*$ as the reflexive and transitive closure of $\rightarrow$.

By [Γ-⊕]/[Γ-&] in Def. 2.8, a typing context entry can transition to one of its continuations by firing an output label of the form $s{:}\mathtt{p}\oplus\mathtt{q}{:}\mathtt{m}(S)$ (in case of selection types), or an input label of the form $s{:}\mathtt{p}\&\mathtt{q}{:}\mathtt{m}(S)$ (in case of branching types). Rule [Γ-Comm] models type-level communication: e.g., it allows two entries $s[\mathtt{p}]{:}S_{\mathtt{p}}, s[\mathtt{q}]{:}S_{\mathtt{q}}$ to interact, provided that: *(1)* $S_{\mathtt{p}}$ is a selection towards $\mathtt{q}$ (with a corresponding output transition); *(2)* $S_{\mathtt{q}}$ is a branching from $\mathtt{p}$ (with a corresponding input transition); and *(3)* they are firing a common message label $\mathtt{m}$, and the carried type S sent by $S_{\mathtt{p}}$ is subtype of the type T expected by $S_{\mathtt{q}}$. When all such conditions hold, $s[\mathtt{p}]{:}S_{\mathtt{p}}, s[\mathtt{q}]{:}S_{\mathtt{q}}$ transition to the respective continuations, by firing a communication label $s{:}\mathtt{p},\mathtt{q}{:}\mathtt{m}$ that records the session s, and the message sender $\mathtt{p}$, recipient $\mathtt{q}$, and label $\mathtt{m}$ (the payload types are discarded).

In the rest of the paper, we will mostly use the unlabelled reduction $\Gamma\rightarrow\Gamma'$, which means that Γ transitions to Γ' through some communication. The labelled transitions will be reprised in §5.

2.3 Towards Subject Reduction and Type Safety

In § 1, we mentioned that a process naively typed with an arbitrary Γ can "go wrong." Indeed, by themselves, the typing rules in Fig. 2 do *not* guarantee type safety, as shown by the following (counter-)example:

$$s[\mathtt{p}]{:}\mathtt{q}\oplus\mathtt{foo}(\mathbf{end}),\ s[\mathtt{q}]{:}\mathtt{p}\&\mathtt{bar}(\mathbf{end}),\ s'[\mathtt{r}]{:}\mathbf{end}\ \vdash\ s[\mathtt{p}][\mathtt{q}]\oplus\mathtt{foo}\langle s'[\mathtt{r}]\rangle \mid s[\mathtt{q}][\mathtt{p}]\textstyle\sum\mathtt{bar}(x) \quad \rightarrow \quad \mathtt{err} \tag{6}$$

Intuitively, the problem of this typing judgement can be seen in its typing context: the type of $s[\mathtt{p}]$ outputs $\mathtt{foo}$ to $\mathtt{q}$, but the type of $s[\mathtt{q}]$ expects $\mathtt{bar}$. This means that we need a criterion to reject (6).

Importantly, the same criterion must be applied for **typing session restriction**. Consider rule [T-ν] in Fig. 2: it types a restricted session s with Γ', provided that *(1)* Γ' only contains channels with roles belonging to s; *(2)* the restricted s does not occur in the remaining context Γ (to avoid clashes); and *(3)* Γ' satisfies a (yet unspecified) property φ. How should we define φ? It cannot be always true, because we would have this counterexample to type-safety, where Γ is the context in (6):

$$\emptyset \vdash (\nu s{:}\Gamma)\left(s[\mathtt{p}][\mathtt{q}]\oplus\mathtt{foo}\langle s'[\mathtt{r}]\rangle \mid s[\mathtt{q}][\mathtt{p}]\textstyle\sum\mathtt{bar}(x)\right) \ \rightarrow\ (\nu s)\,\mathtt{err} \quad \text{(by (6) and rule [R-Ctx] in Fig. 1)} \tag{7}$$

To achieve type safety, we want the process in (7) to be untypable — which means that, when type-checking $(\nu s{:}\Gamma)$..., we must ensure that φ in rule [T-ν] does *not* hold for Γ, in cases like (6).

Moreover, φ must be technically usable to prove subject reduction; this leads to three *desiderata*:

- **(D1)** φ must make the typing context "safe:" if the type of $s[\mathtt{p}]$ sends a message to $\mathtt{q}$, then the type of $s[\mathtt{q}]$ must be able to input such a message;
- **(D2)** φ must be preserved when the typing rule [T-|] splits typing contexts (see derivation in Ex. 2.7);
- **(D3)** φ must be preserved when processes, and typing contexts, interact and reduce (Def. 2.2/2.8).

Therefore, the choice of the criterion for handling cases like (6) has a deep impact on the theoretical foundations of the type system: it determines how subject reduction and type safety properties are stated and proved, and how general/restrictive they are; it also determines how to define φ in rule [T-ν], to correctly type session restriction $(\nu s)\,P$, and handle cases like (7).

In §4, we show how our new MPST theory establishes its foundations, and φ in rule [T-ν]. But first, in §3, we show how such choices are made in classic MPST, and what are the consequences.

3 LIMITATIONS AND THEORETICAL ISSUES OF CLASSIC MPST

This section gives a formal basis to our claims in §1: in §3.1 we use our opening example to show the technical issues of classic MPST, caused by *consistency* (also called *coherency*, e.g., by Deniélou et al. [2012]); and in §3.2, we provide further examples that are rejected by classic MPST. Our new MPST system (§4) eschews these problems, by adopting a more general theoretical basis.

Remark 3.1. *The issues described in this section do* not *apply to two recent MPST works, by Dezani-Ciancaglini et al. [2015] and Scalas and Yoshida [2018]: they have different,* non-classic *MPST theories. However, such works have other limitations, surmounted by this paper: they are detailed in §8.2.*

3.1 Consistency and Subject Reduction

To reject cases like (6) (§ 2.3), classic MPST require typing contexts to be *consistent*: for each pair of entries $\{s[\mathbf{p}]{:}S_{\mathbf{p}}\,,\ s[\mathbf{q}]{:}S_{\mathbf{q}}\} \subseteq \Gamma$, the inputs/outputs of $S_{\mathbf{p}}$ from/to $\mathbf{q}$ must be *dual* w.r.t. the outputs/inputs of $S_{\mathbf{q}}$ to/from $\mathbf{p}$. This guarantees that two roles $\mathbf{p}, \mathbf{q}$ can only send/receive compatible messages in a session s. More precisely, consistency requires to check the duality of the *partial projections* $S_{\mathbf{p}}\restriction\mathbf{q}$ and $S_{\mathbf{q}}\restriction\mathbf{p}$, using Def. 3.5, 3.6, 3.7, and 3.8 (collected in Fig. 3): this clearly shows that MPST were developed by adopting a proof framework based on *binary* session types.

Correspondingly, to reject cases like (7), classic MPST define rule [T-ν] in Fig. 2 by setting $\varphi =$ consistent. This yields the **classic session restriction typing rule**:

$$\dfrac{\Gamma' = \{s[\mathbf{p}]{:}S_{\mathbf{p}}\}_{\mathbf{p}\in I} \qquad s \notin \Gamma \qquad \text{consistent}(\Gamma') \qquad \Theta\cdot\Gamma,\Gamma' \vdash P}{\Theta\cdot\Gamma \vdash (\nu s{:}\Gamma')\,P}\ \text{[T-}\nu\text{Classic]}$$

and this is sound (indeed, consistency satisfies the *desiderata* **(D1)**–**(D3)** described in §2.3).

E.g., the typing context in (6) is not consistent; correspondingly, no consistent Γ can be assigned to $(\nu s)\ldots$ in (7): hence, with rule [T-νClassic], the process in (7) is untypable in classic MPST.

Limitations of Consistency. Take the processes from Ex. 2.3, and the typing derivation from Ex. 2.7. Using the rules in Fig. 2 with [T-νClassic] above, we might try to type our opening example as:

$$\dfrac{\Gamma\ \text{consistent} \qquad \dfrac{\vdots\ \text{(from Ex. 2.7)}}{\Gamma \vdash P_{\mathbf{s}} \mid P_{\mathbf{c}} \mid P_{\mathbf{a}}}\ \text{[T-}|\text{]}}{\emptyset \vdash (\nu s{:}\Gamma)\,(P_{\mathbf{s}} \mid P_{\mathbf{c}} \mid P_{\mathbf{a}})}\ \text{[T-}\nu\text{Classic]} \qquad \text{where}\ \ \Gamma = s[\mathbf{s}]{:}S_{\mathbf{s}}, s[\mathbf{c}]{:}S_{\mathbf{c}}, s[\mathbf{a}]{:}S_{\mathbf{a}} \tag{8}$$

As shown in §1(2), the types $S_{\mathbf{s}}$, $S_{\mathbf{c}}$, $S_{\mathbf{a}}$ assigned to $s[\mathbf{s}]$, $s[\mathbf{c}]$, $s[\mathbf{a}]$ are respectively $G\restriction\mathbf{s}$, $G\restriction\mathbf{c}$, $G\restriction\mathbf{a}$, i.e., the projections of G (Def. 3.3). However, *the derivation in* (8) *is wrong*, because the consistency premise of [T-νClassic] does *not* hold. To see why, we need to check all pairs of types for session s:

- $S_{\mathbf{s}}$,$S_{\mathbf{c}}$ are consistent: the outputs of $S_{\mathbf{s}}$ to $\mathbf{c}$ are *dual* w.r.t. the inputs of $S_{\mathbf{c}}$ from $\mathbf{s}$;
- $S_{\mathbf{s}}$,$S_{\mathbf{a}}$ are *not* consistent, because the partial projections $S_{\mathbf{s}}\restriction\mathbf{a}$ and $S_{\mathbf{a}}\restriction\mathbf{s}$ are *undefined* (Def. 3.6).

Intuitively, $S_{\mathbf{s}}\restriction\mathbf{a}$ and $S_{\mathbf{a}}\restriction\mathbf{s}$ are undefined because the inputs/outputs of $S_{\mathbf{s}}/S_{\mathbf{a}}$ from/to $\mathbf{a}/\mathbf{s}$ depend on previous I/O with $\mathbf{c}$: i.e., if the service $\mathbf{s}$ sends `login` (resp. `cancel`) to the client $\mathbf{c}$, then $\mathbf{s}$ will (resp. will *not*) later interact with the authorisation server $\mathbf{a}$. This is not captured by the syntactic nature of projection/duality checks: i.e., protocols with inter-role dependencies are often *not* consistent — even simple ones, like G in (1). Consequently, the process in Ex. 2.3 is untypable, albeit correct (does not reduce to **err**).

Definition 3.2. The syntax of a *global type* G is:

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

We write $\mathsf{p}\in\mathrm{roles}(G)$ (or simply $\mathsf{p}\in G$) iff, for some q, either $\mathsf{p}{\to}\mathsf{q}$ or $\mathsf{q}{\to}\mathsf{p}$ occurs in G.

Definition 3.3 (Global Type Projection). The *projection of* G *onto* p, written $G{\restriction}\mathsf{p}$, is:

$$(\mathsf{q}{\to}\mathsf{r}\colon\{\mathtt{m}_i(S_i)\,.\,G_i\}_{i\in I}){\restriction}\mathsf{p} \;=\; \begin{cases} \mathsf{r}{\oplus}_{i\in I}\mathtt{m}_i(S_i).(G_i{\restriction}\mathsf{p}) & \text{if } \mathsf{p}=\mathsf{q}\\ \mathsf{q}{\&}_{i\in I}\mathtt{m}_i(S_i).(G_i{\restriction}\mathsf{p}) & \text{if } \mathsf{p}=\mathsf{r}\\ \bigsqcap_{i\in I} G_i{\restriction}\mathsf{p} & \text{if } \mathsf{q}\neq\mathsf{p}\neq\mathsf{r}\end{cases}$$

$$(\mu\mathbf{t}.G){\restriction}\mathsf{p} \;=\; \begin{cases}\mu\mathbf{t}.(G{\restriction}\mathsf{p}) & \text{if } \mathsf{p}\in G \text{ or } \mathrm{fv}(\mu\mathbf{t}.G)\neq\emptyset\\ \mathbf{end} & \text{otherwise}\end{cases} \qquad \begin{array}{r}\mathbf{t}{\restriction}\mathsf{p} = \mathbf{t}\\ \mathbf{end}{\restriction}\mathsf{p} = \mathbf{end}\end{array}$$

where $\bigsqcap$ is the *merge operator for session types*, that could be either the *plain merging* defined as $S\sqcap S = S$, or the *full merging*:

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

$$\mathsf{p}{\oplus}_{i\in I}\mathtt{m}_i(S_i).S'_i \;\sqcap\; \mathsf{p}{\oplus}_{i\in I}\mathtt{m}_i(S_i).S'_i \;=\; \mathsf{p}{\oplus}_{i\in I}\mathtt{m}_i(S_i).S'_i$$

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

Definition 3.4 (Partial Session Types). Partial session types, ranged over by H, are:

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

Definition 3.5 (Duality of Partial Session Types). The *dual of* H, written $\overline{H}$, is:

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

Definition 3.6 (Partial Projection). The *projection of* S *onto* p, written $S{\restriction}\mathsf{p}$, is:

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

$$(\mu\mathbf{t}.S){\restriction}\mathsf{p} = \begin{cases}\mu\mathbf{t}.(S{\restriction}\mathsf{p}) & \text{if } \mathsf{p}\in S \text{ or } \mathrm{fv}(\mu\mathbf{t}.S)\neq\emptyset\\ \mathbf{end} & \text{otherwise}\end{cases} \qquad \begin{array}{r}\mathbf{t}{\restriction}\mathsf{p} = \mathbf{t}\\ \mathbf{end}{\restriction}\mathsf{p} = \mathbf{end}\end{array}$$

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

$$\&_{i\in I}\mathtt{m}_i(S_i).H_i \;\sqcap\; \&_{i\in I}\mathtt{m}_i(S_i).H'_i \;=\; \&_{i\in I}\mathtt{m}_i(S_i).(H_i\sqcap H'_i)$$

$$\oplus_{i\in I}\mathtt{m}_i(S_i).H_i \;\sqcap\; \oplus_{j\in J}\mathtt{m}_j(S_j).H'_j \;=\; \oplus_{k\in I\cap J}\mathtt{m}_k(S_k).(H_k\sqcap H'_k) \;\oplus\; \oplus_{i\in I\setminus J}\mathtt{m}_i(S_i).H_i \;\oplus\; \oplus_{j\in J\setminus I}\mathtt{m}_j(S_j).H'_j$$

$$\mu\mathbf{t}.H \sqcap \mu\mathbf{t}.H' \;=\; \mu\mathbf{t}.(H\sqcap H') \qquad \mathbf{t}\sqcap\mathbf{t} \;=\; \mathbf{t} \qquad \mathbf{end}\sqcap\mathbf{end} \;=\; \mathbf{end}$$

Definition 3.7. Subtyping for partial types is coinductively defined (we omit unfolding rules, cf. Def. 2.5):

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

Definition 3.8. Γ *is consistent* iff, $\forall s$, $\mathsf{p}\neq\mathsf{q}$, S, T, $\{s[\mathsf{p}]{:}S, s[\mathsf{q}]{:}T\}\subseteq\Gamma$ implies $\overline{S{\restriction}\mathsf{q}} \leqslant T{\restriction}\mathsf{p}$.

Fig. 3. Classic MPST: global types, projections, consistency, and duality. Note that all these definitions are **not** necessary in our new theory of multiparty session types (§4).

Subject Reduction and Type Safety (or Lack Thereof). As noted in § 1, the classic MPST subject reduction statement is (4). Now, consider (8) again: the conclusion is wrong, but the intermediate judgement $\Gamma\vdash P_{\mathsf{s}} \mid P_{\mathsf{c}} \mid P_{\mathsf{a}}$ holds. For this judgement, the subject reduction statement (4) is vacuously true (since Γ is not consistent): hence, we cannot prove that $P_{\mathsf{s}} \mid P_{\mathsf{c}} \mid P_{\mathsf{a}}$ "never goes wrong."

Interplay Between Consistency and Global Type Projection. The consistency requirement constrains the MPST theory in non-obvious ways, causing subtle issues with *global type projections*. Several

MPST papers claim that if Γ is obtained by projecting a global type G, then Γ is consistent (see e.g.: [Deniélou et al. 2012, p.28], [Coppo et al. 2015a, Prop. 1], [Chen 2015, Prop. 2]). This claim corresponds to introducing the typing rule [T-νClassicG] below, that seemingly fixes derivation (8):

$$\frac{\Gamma' = \{s[\mathbf{p}]{:}G{\upharpoonright}\mathbf{p}\}_{\mathbf{p}\in \text{roles}(G)} \qquad s \notin \Gamma \qquad \Theta \cdot \Gamma, \Gamma' \vdash P}{\Theta \cdot \Gamma \vdash (\nu s{:}\Gamma')\, P}\ \text{[T-}\nu\text{ClassicG]}$$

Unfortunately, our example in §1 shows a global type whose projections are *not* consistent. This is because we use the *"full merging"* projection (Def. 3.3), introduced in Deniélou et al. [2012]; Yoshida et al. [2010] to type more processes. The intuition is the following. Take the initial choice of the global type G in §1(1) (reported below), that does *not* involve role $\mathbf{a}$:

$$G \;=\; \mathbf{s}{\to}\mathbf{c}{:}\{\mathtt{login}.G_1,\ \mathtt{cancel}.G_2\} \qquad \text{where} \begin{cases} G_1 = \mathbf{c}{\to}\mathbf{a}{:}\mathtt{passwd}(\mathtt{Str})\ .\ \mathbf{a}{\to}\mathbf{s}{:}\mathtt{auth}(\mathtt{Bool}) \\ G_2 = \mathbf{c}{\to}\mathbf{a}{:}\mathtt{quit} \end{cases}$$

To project G onto $\mathbf{a}$, we must "skip" the first interaction between $\mathbf{s}$ and $\mathbf{c}$, and *merge* the projections of G_1 and G_2 onto $\mathbf{a}$, rejecting potentially unsafe local types combinations (thus avoiding cases like (6) above). Consequently, projection works as follows:

$$G{\upharpoonright}\mathbf{a} \;=\; S_1 \sqcap S_2 \qquad \text{where} \begin{cases} S_1 \;=\; G_1{\upharpoonright}\mathbf{a} \;=\; \mathbf{c}\&\mathtt{passwd}(\mathtt{Str}).\mathbf{s}\oplus\mathtt{auth}(\mathtt{Bool}) \\ S_2 \;=\; G_2{\upharpoonright}\mathbf{a} \;=\; \mathbf{c}\&\mathtt{quit} \end{cases}$$

We now have two possibilities, depending on how we choose the *merging operator* $\sqcap$ (Def. 3.3):

- *plain merging:* $S_1 \sqcap S_2 = S_1$ iff $S_1 = S_2$ (undefined otherwise);
- *full merging:* $S_1 \sqcap S_2 = S_{\mathbf{a}}$ (see (2) in §1).

i.e., the restrictive plain merging is undefined for our example G, while full merging yields all desired projections — but they are *not consistent*, as shown above. Consequently, the tentative rule [T-νClassicG] with "full merging" projections *breaks subject reduction proofs*. E.g., take P typed by [T-νClassicG], and reducing to P', as follows:

$$\emptyset \cdot \emptyset \vdash P \ \text{ with } P \;=\; (\nu s{:}\Gamma)\, P_0 \;\to\; (\nu s{:}\Gamma')\, P_1 \;=\; P' \quad \text{(induced by } P_0 \to P_1 \text{ and rule [R-Ctx] in Fig. 1)} \quad (9)$$

To prove subject reduction as stated in (4), we need to invert P's typing and apply the induction hypothesis on $\Theta \cdot \Gamma \vdash P_0$ and $P_0 \to P_1$ (from (9)), to obtain that there is some Γ' such that $\Gamma \to^* \Gamma'$ and $\Theta \cdot \Gamma' \vdash P_1$; however, to apply (4) in the induction hypothesis we need Γ consistent, and we have shown that this hypothesis might not hold.

We can now revisit our claims in §1, making them precise, and highlighting the resulting impasse:

(C1) **overly restrictive:** requiring Γ consistent drastically constrains typability: it rejects our simple example in §1, and many other correct protocols (see §3.2 later on). Correspondingly, the restrictive "plain merging" projection of [Honda et al. 2008, Def. 4.1] and [Coppo et al. 2015a, Def. 1], guarantees consistency by rejecting many correct protocols;

(C2) **inflexible and error-prone:** if we use a "full merging" projection as in, e.g., Yoshida et al. [2010] or Deniélou et al. [2012], then Γ might *not* be consistent. This means that the proofs of subject reduction depending on "full merging" (e.g. [Yoshida et al. 2010, Thm 3.5], [Deniélou et al. 2012, Thm 4.6], and successive papers discussed in §8) do not work; we might fix such proofs by adding a consistency requirement — but then, we would fall back into **(C1)** above.

In §4, we completely eschew these issues by developing new theoretical foundations for MPST: we cut the ties with binary session types, adopting a more general, *behavioural* safety invariant, that subsumes consistency and binary session duality.

3.2 More Examples of Correct, yet Non-Consistent Protocols

We conclude this section with Fig. 4, that describes various multiparty protocols, formalised as typing contexts. None of such protocols is consistent, because some of their partial projections are

(1) OAuth2 fragment.

(See global type (1) in §1) | *(See types (2) in §1, and Γ in Ex. 2.7)*

(2) Recursive two-buyers protocol. This is a mild variation of a typical example in MPST literature. Alice ($\mathtt{a}$) queries the store ($\mathtt{s}$) for an item, and the store replies with a `price`; then, she asks Bob ($\mathtt{b}$) to `split` the price: if he says `yes`, then she `buy`s the item from the store; if he says `no`, then Alice recursively retries, proposing another split to Bob; at any point, Alice can `cancel` her bargaining with Bob, and say `no` to the store.

N/A

$$s[\mathtt{a}] : \mathtt{s}{\oplus}\mathtt{query}(\mathrm{Str}).\mathtt{s}\&\mathtt{price}(\mathrm{Int}).\mu\mathbf{t}.\mathtt{b}{\oplus}\left\{\begin{array}{l}\mathtt{split}(\mathrm{Int}).\mathtt{b}\&\left\{\begin{array}{l}\mathtt{yes}.\mathtt{s}{\oplus}\mathtt{buy}.\mathbf{end},\\ \mathtt{no}.\mathbf{t}\end{array}\right\},\\ \mathtt{cancel}.\mathtt{s}{\oplus}\mathtt{no}\end{array}\right\}$$
$$s[\mathtt{s}] : \mathtt{a}\&\mathtt{query}(\mathrm{Str}).\mathtt{a}{\oplus}\mathtt{price}(\mathrm{Int}).\mathtt{a}\&\left\{\mathtt{buy}.\mathbf{end}\,,\ \mathtt{no}.\mathbf{end}\right\}$$
$$s[\mathtt{b}] : \mu\mathbf{t}.\mathtt{a}\&\left\{\mathtt{split}(\mathrm{Int}).\mathtt{a}{\oplus}\left\{\mathtt{yes}.\mathbf{end}\,,\ \mathtt{no}.\mathbf{t}\right\}\,,\ \mathtt{cancel}.\mathbf{end}\right\}$$

(3) Recursive map/reduce. The mapper ($\mathtt{m}$) sends a `datum` to n workers ($\mathtt{w}_1, \ldots, \mathtt{w}_n$, for some given n), and each one sends a `result` to the reducer ($\mathtt{r}$); then, the reducer tells the mapper whether to `continue` with another iteration, or `stop`: in the first case, the mapper loops, while in the second case, it `stop`s the workers.

$$\begin{array}{l}\mu\mathbf{t}.\mathtt{m}{\to}\mathtt{w}_1{:}\mathtt{datum}(\mathrm{Int})\,.\,\ldots\,.\\ \quad\mathtt{m}{\to}\mathtt{w}_n{:}\mathtt{datum}(\mathrm{Int})\\ \qquad\mathtt{w}_1{\to}\mathtt{r}{:}\mathtt{result}(\mathrm{Int})\,.\,\ldots\,.\\ \qquad\mathtt{w}_n{\to}\mathtt{r}{:}\mathtt{result}(\mathrm{Int})\,.\\ \qquad\mathtt{r}{\to}\mathtt{m}{:}\left\{\begin{array}{l}\mathtt{continue}(\mathrm{Int})\,.\,\mathbf{t}\,,\\ \mathtt{stop}\,.\,\mathtt{m}{\to}\mathtt{w}_1{:}\mathtt{stop}\,.\,\ldots\,.\\ \qquad\qquad\mathtt{m}{\to}\mathtt{w}_n{:}\mathtt{stop}\end{array}\right\}\end{array}$$

$$s[\mathtt{m}] : \mu\mathbf{t}.\mathtt{w}_1{\oplus}\mathtt{datum}(\mathrm{Int}).\ldots.\mathtt{w}_n{\oplus}\mathtt{datum}(\mathrm{Int}).\mathtt{r}\&\left\{\begin{array}{l}\mathtt{continue}(\mathrm{Int}).\mathbf{t}\\ \mathtt{stop}.\mathtt{w}_1{\oplus}\mathtt{stop}.\ldots.\mathtt{w}_n{\oplus}\mathtt{stop}\end{array}\right\}$$
$$s[\mathtt{w}_i] : \mathtt{m}\&\mathtt{datum}(\mathrm{Int}).\mu\mathbf{t}.\mathtt{r}{\oplus}\mathtt{result}(\mathrm{Int}).\mathtt{m}\&\left\{\begin{array}{l}\mathtt{datum}(\mathrm{Int}).\mathbf{t}\,,\\ \mathtt{stop}.\mathbf{end}\end{array}\right\}\qquad(\forall i\in 1..n)$$
$$s[\mathtt{r}] : \mu\mathbf{t}.\mathtt{w}_1\&\mathtt{datum}(\mathrm{Int}).\ldots.\mathtt{w}_n\&\mathtt{datum}(\mathrm{Int}).\mathtt{m}{\oplus}\left\{\begin{array}{l}\mathtt{continue}(\mathrm{Int}).\mathbf{t}\\ \mathtt{stop}.\mathbf{end}\end{array}\right\}$$

(4) Independent multiparty workers. The starter process ($\mathtt{s}$) sends a `datum` to n worker processes ($\mathtt{wa}_1, \ldots, \mathtt{wa}_n$, for some given n), and each one starts exchanging `datum`/`result` messages with two other workers ($\mathtt{wb}_i$ and $\mathtt{wc}_i$, for $i \in 1..n$). Each triplet of workers $\mathtt{wa}_i$, $\mathtt{wb}_i$, $\mathtt{wc}_i$ ($i \in i..n$) keeps interacting until $\mathtt{wa}_i$ sends `stop` to $\mathtt{wb}_i$, who forwards `stop` to $\mathtt{wc}_i$.

$$\begin{array}{l}\mathtt{s}{\to}\mathtt{wa}_1{:}\mathtt{datum}(\mathrm{Int})\,.\,\ldots\,.\,\mathtt{s}{\to}\mathtt{wa}_n{:}\mathtt{datum}(\mathrm{Int})\,.\\ \mu\mathbf{t}.\mathtt{wa}_i{\to}\mathtt{wb}_i{:}\left\{\begin{array}{l}\mathtt{datum}(\mathrm{Int}).\mathtt{wb}_i{\to}\mathtt{wc}_i{:}\mathtt{datum}(\mathrm{Int})\,.\\ \qquad\qquad\mathtt{wc}_i{\to}\mathtt{wa}_i{:}\mathtt{result}(\mathrm{Int})\,.\,\mathbf{t}\,,\\ \mathtt{stop}\,.\,\mathtt{wb}_i{\to}\mathtt{wc}_i{:}\mathtt{stop}\end{array}\right\}\end{array}$$

$$s[\mathtt{s}] : \mathtt{wa}_1{\oplus}\mathtt{datum}(\mathrm{Int}).\ldots.\mathtt{wa}_n{\oplus}\mathtt{datum}(\mathrm{Int}).\mathbf{end}$$
$$s[\mathtt{wa}_i] : \mathtt{s}\&\mathtt{datum}(\mathrm{Int}).\mu\mathbf{t}.\mathtt{wb}_i{\oplus}\left\{\begin{array}{l}\mathtt{datum}(\mathrm{Int}).\mathtt{wc}_i\&\mathtt{result}(\mathrm{Int}).\mathbf{t},\\ \mathtt{stop}.\mathbf{end}\end{array}\right\}$$
$$s[\mathtt{wb}_i] : \mu\mathbf{t}.\mathtt{wa}_i\&\left\{\begin{array}{l}\mathtt{datum}(\mathrm{Int}).\mathtt{wc}_i{\oplus}\mathtt{datum}(\mathrm{Int}).\mathbf{t}\,,\\ \mathtt{stop}.\mathtt{wc}_i{\oplus}\mathtt{stop}.\mathbf{end}\end{array}\right\}$$
$$s[\mathtt{wc}_i] : \mu\mathbf{t}.\mathtt{wb}_i\&\left\{\begin{array}{l}\mathtt{datum}(\mathrm{Int}).\mathtt{wa}_i{\oplus}\mathtt{result}(\mathrm{Int}).\mathbf{t}\,,\\ \mathtt{stop}.\mathbf{end}\end{array}\right\}$$

Fig. 4. A selection of multiparty protocols: each one is expressed as a (non-consistent) typing context (on the right); for the sake of clarity, we also outline the shape of a global type with corresponding projections (on the left). The exception is protocol (2), that cannot be projected from *any* global type: see §3.2. Being non-consistent, all these protocols are not supported by classic MPST — but they are all supported by our new general type system (§4); moreover, they have different behavioural properties, analysed in §5.3 (Table 1).

undefined — as a consequence of the issues illustrated in §3.1; moreover, the protocols (2), (3) and (4) trigger further subtle restrictions in the partial projection/merging of recursive types (Def. 3.6).

Notably, Fig. 4 includes an example of multiparty protocol that cannot be projected from *any* global type: the recursive two-buyers protocol (2). The key issue is in the type of $s[\mathtt{a}]$, when $\mathtt{a}$lice interacts with $\mathtt{b}$ob: $\mathtt{a}$lice sends a message to the $\mathtt{s}$tore in one of the branches under recursion $\mu\mathbf{t}.\ldots$ (where $\mathtt{b}$ob answers `yes`), but not in the other branch (where $\mathtt{b}$ob says `no`). This is *not* supported by projection and merging (Def. 3.3): they can only generate session types where all branches under recursion syntactically contain a same set of roles. Consequently, no global type can be projected and yield the type of $s[\mathtt{a}]$ in Fig. 4(2). This restriction does not impact our new MPST theory (§4).

4 A NEW, GENERAL MULTIPARTY SESSION TYPE SYSTEM

We now present our new general MPST theory. Its generality comes from the fact that it is based on a weak *typing context safety* invariant, that rejects cases like (6)/(7) (§2.3) without the restrictions and drawbacks of classic MPST consistency. Moreover, we design the new type system to be *parametric* on the safety invariant itself: by fine-tuning the parameter, the type system can accept or reject MPST processes depending on the properties of the protocols they implement (we will take advantage of this feature in §5). Hence, different instantiations of the parameter yield different type system instances — but we just need to prove type safety *once*, under the *weakest* safety invariant. This design is inspired by Igarashi and Kobayashi [2004]'s Generic Type System for the π-calculus.

We first formalise what a "safety invariant" is, in Def. 4.1 below: it is a *behavioural* property of typing contexts, that depends on how they reduce (cf. Def. 2.8). The fundamental difference with classic MPST (§3) is that our safety is *not* based on binary session types, *nor* duality.

Definition 4.1. φ is a *safety property* of typing contexts iff:

$$\text{[S-}\oplus\&\text{]}\quad \varphi\Big(\Gamma,\ s[\mathbf{p}]{:}\mathbf{q}\oplus_{i\in I}\mathtt{m}_i(S_i).S'_i,\ s[\mathbf{q}]{:}\mathbf{p}\&_{j\in J}\mathtt{m}_j(T_j).T'_j\Big)\ \text{ implies }\ I\subseteq J,\ \text{and}\ \forall i\in I: S_i\leqslant T_i;$$

$$\text{[S-}\mu\text{]}\quad \varphi(\Gamma,\ s[\mathbf{p}]{:}\mu\mathbf{t}.S)\ \text{ implies }\ \varphi(\Gamma,\ s[\mathbf{p}]{:}S\{\mu\mathbf{t}.S/\mathbf{t}\});$$

$$\text{[S-}\!\rightarrow\!\text{]}\quad \varphi(\Gamma)\ \text{ and }\ \Gamma\rightarrow\Gamma'\ \text{ implies }\ \varphi(\Gamma').$$

We say Γ *is safe*, written $\mathrm{safe}(\Gamma)$, if $\varphi(\Gamma)$ for some safety property φ.

The rules of Def. 4.1 directly satisfy the *desiderata* **(D1)** and **(D3)** discussed in §2.3 (whereas **(D2)** is satisfied by Lemma 4.3, as we will see shortly). Rule [S-$\oplus\&$] says that the roles in a safe typing context can only exchange compatible messages (this is *desideratum* **(D1)**): more precisely, if the typing context contains entries for $s[\mathbf{p}]$ and $s[\mathbf{q}]$, with $\mathbf{p}$ sending to $\mathbf{q}$ and $\mathbf{q}$ receiving from $\mathbf{p}$, then $\mathbf{p}$ support all $\mathbf{q}$'s messages — and thus, they can reduce, by Def. 2.8. Rule [S-μ] says that φ contains all recursive type unfoldings: this allows rule [S-$\oplus\&$] to check unfolded types, where $\oplus/\&$ occur at the the top-level. By rule [S-$\rightarrow$], safety is preserved whenever Γ reduces (this is *desideratum* **(D3)**).

Example 4.2. The typing context Γ of (8) in §3 is safe. This can be easily verified by: (1) defining φ as $\varphi=\{\Gamma' \mid \Gamma\rightarrow^{*}\Gamma'\}$, i.e., containing Γ and all its reductions; (2) checking that φ is a safety property, because all its elements satisfy the clauses of Def. 4.1; and (3) concluding that, since $\varphi(\Gamma)$ holds, Γ is safe. Instead, the typing context in (6) is *not* safe: any property φ containing such typing context is *not* a safety property, as it violates clause [S-$\oplus\&$] of Def. 4.1.

Def. 4.1 also has the properties below, useful for proving subject reduction: typing context splits preserve safety (Lemma 4.3, which satisfies the remaining *desideratum* **(D2)** in §2.3); if Γ is safe, then supertyping/reductions commute (Lemma 4.4); supertyping preserves safety (Lemma 4.5).

Lemma 4.3. *If* Γ,Γ' *is safe, then* Γ *is safe.*

Lemma 4.4. *If* Γ *safe and* $\Gamma\leqslant\Gamma'\rightarrow\Gamma''$, *then there is* Γ''' *such that* $\Gamma\rightarrow\Gamma'''\leqslant\Gamma''$.

Lemma 4.5. *If* Γ *is safe and* $\Gamma\leqslant\Gamma'$, *then* Γ' *is safe.*

We can now define our new multiparty session type system. As explained in §2.3, since we are adopting safety (Def. 4.1) as the criterion for accepting/rejecting typing contexts, we use the same criterion to define a typing rule for session restriction.

Definition 4.6 (General Multiparty Session Type System). The *general MPST typing judgement* is inductively defined by the rules in Fig. 2 — with rule [T-ν] restricted as follows:

$$\frac{\Gamma'=\left\{s[\mathbf{p}]{:}S_{\mathbf{p}}\right\}_{\mathbf{p}\in I}\qquad \varphi(\Gamma')\qquad s\notin\Gamma\qquad \Theta\cdot\Gamma,\Gamma'\vdash P}{\Theta\cdot\Gamma\vdash(\nu s{:}\Gamma')\,P}\ \text{[TGen-}\nu\text{]}\qquad \text{where } \varphi \text{ is a safety property}$$

Given a safety property φ, we write "$\Theta\cdot\Gamma\vdash P$ *with* φ" to instantiate φ in [TGen-ν] above; when "*with* φ" is omitted, then the instantiation is $\varphi=\mathrm{safe}$ (i.e., the largest safety property, cf. Def. 4.1).

Example 4.7. Take the (wrong) typing derivation (8) in §3.1, and replace the (wrong) application of rule [T-νCLASSIC] with [TGEN-ν] from Def. 4.6, instantiating φ with the safety property of Ex. 4.2 (that contains Γ). The resulting typing derivation is correct.

Ex.4.7 above shows that our new type system is not limited by consistency requirements, and types our opening example. Notably, the only visible difference between our new type system (Def. 4.6) and the classic one (§3.1) is that [TGEN-ν] uses a (parametric) safety property φ, instead of consistency.[2] As explained in §2.3, this small visible difference between typing rules is a manifestation of a deeper underlying change: by removing the crucial consistency/duality assumption of classic MPST, we are replacing its theoretical underpinnings, and this requires a revision of all MPST soundness proofs. The payoff is that our new MPST theory enjoys a much more general subject reduction property (Thm. 4.8, based on Lemmas 4.3 to 4.5); from this, we get that typed processes "never go wrong" (Cor. 4.9). And again, unlike classic MPST, these results are *not* limited by consistency.

THEOREM 4.8 (SUBJECT REDUCTION). *Assume $\Theta \cdot \Gamma \vdash P$ and Γ safe. Then, $P \rightarrow P'$ implies $\exists \Gamma'$ safe such that $\Gamma \rightarrow^* \Gamma'$ and $\Theta \cdot \Gamma' \vdash P'$.*

COROLLARY 4.9 (TYPE SAFETY). *If $\emptyset \cdot \emptyset \vdash P$ and $P \rightarrow^* P'$, then P' has no error.*

PROOF. We first prove a more general result. Assume $\Theta \cdot \Gamma \vdash P$ with Γ safe, and $P = P_1 \rightarrow \cdots \rightarrow P_n = P'$. By induction on n, using Thm. 4.8, we prove $\Theta \cdot \Gamma' \vdash P'$, for some safe Γ' such that $\Gamma \rightarrow^* \Gamma'$. Now, by contradiction, assume that P' has an error (Def. 2.2); then, P' is untypable, since its **err** subterm is untypable: contradiction. Hence, P' has no errors. We obtain Cor. 4.9 as a special case of the result above, with $\Theta = \emptyset$ and $\Gamma = \Gamma' = \emptyset$ (that is vacuously safe). □

Example 4.10. Take our opening example in §1, and its typed process from Ex. 2.7 and 4.7. Using our new Thm. 4.8 instead of the classic MPST subject reduction (4) in §1, we infer that all process reductions are well-typed. And by Cor. 4.9, we are guaranteed that they do not contain errors.

Finally, note that type checking is decidable, whenever Def. 4.6 is instantiated with a decidable safety property: this mainly follows because typing rules are syntax-directed, and for any P, at most one can be applied. Also note that, since we proved Thm. 4.8 and Cor. 4.9 using the largest (i.e., the weakest) safety property, we do not need to repeat the proof depending on how φ is instantiated in Def. 4.6: subject reduction and type safety hold for any safety property φ.

THEOREM 4.11. *If φ is decidable, then "$\Theta \cdot \Gamma \vdash P$ with φ" is decidable.*

5 VERIFYING RUN-TIME PROPERTIES OF PROCESSES, USING TYPES

In this section, we show that by suitably instantiating φ in our type system (Def. 4.6), we can statically enforce desired run-time properties on processes — e.g., deadlock freedom and liveness.

In order to achieve this result, we study several typing context properties, and compare them with safety (Def. 4.1). The main reason for this study is that safety, albeit guaranteeing error-freedom (Thm. 4.8, Cor. 4.9), is otherwise rather weak. E.g., the following typing context is safe but deadlocked (it cannot reduce, because **p** is waiting an input from **q**, who is waiting for **r**, who is waiting for **p**):

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

and the context above types deadlocked processes that cannot reduce, either. This is undesirable: "real-world" programs should be deadlock-free, or even *live* (i.e., each pending input/output should be fired, eventually). Therefore, stronger typing context properties are needed — and in our new MPST theory, we can use the parameter φ of Def. 4.6 to enforce them, without consistency limitations.

[2] In §5.4, we show that all typing derivations of classic MPST are valid under Def. 4.6: consistency implies safety, hence in [TGEN-ν] we can let φ = consistent; and in §5.5, we show how φ statically determines the run-time properties on processes.

We first discuss several desirable, although undecidable, run-time properties of processes, such as deadlock-freedom and liveness (§5.1); next, we prove *session fidelity*, a crucial result that connects typing context reductions to processes reductions (§5.2). Then, we present various typing context properties (§5.3), and compare them (§5.4); finally, we show that they are decidable, and, with our new type system, they can be used to ensure that processes are, e.g., deadlock-free and live (§5.5).

5.1 Run-Time Properties of Processes

In Def. 5.1 below, we formalise various desirable process properties. All these properties are *undecidable*, because the MPST π-calculus is Turing-powerful [Busi et al. 2009]. To surmount this obstacle, from §5.3 we will reason on analogous properties for types (that are not Turing-powerful).

Definition 5.1 (Process properties). P is **deadlock-free** iff $P \to^* P' \not\to$ implies $P' \equiv \mathbf{0}$. P is **terminating** iff it is deadlock-free, and $\exists j$ finite such that, $\forall n \geq j$, $P = P_0 \to P_1 \to \cdots \to P_n$ implies $P_n \equiv \mathbf{0}$. P is **never-terminating** iff $P \to^* P'$ implies $P' \to$. P is **live** iff $P \to^* P' \equiv \mathbb{C}[Q]$ implies:

(1) if $Q = c[\mathtt{q}] \oplus \mathtt{m}\langle s'[\mathtt{r}]\rangle.Q'$ (for some $\mathtt{m}, s', \mathtt{r}, Q'$), then $\exists \mathbb{C}'\colon P' \to^* \mathbb{C}'[Q']$; and
(2) if $Q = c[\mathtt{q}] \sum_{i \in I} \mathtt{m}_i(x_i).Q'_i$ (for some $\mathtt{m}_i, x_i, Q'_i$), then $\exists \mathbb{C}', k \in I, s', \mathtt{r}$: $P' \to^* \mathbb{C}'\left[Q'_k\{{}^{s'[\mathtt{r}]}/_{x_k}\}\right]$.

P is **strongly live** iff $P \to^* P' \equiv \mathbb{C}[Q]$ implies:

(3) item 1 above, and moreover, there is n finite such that, whenever $P' = P'_0 \to P'_1 \to \cdots \to P'_n$, then for some $j \leq n$ we have $P'_j \to \mathbb{C}''[Q']$ (for some $\mathbb{C}''$);
(4) item 2 above, and moreover, there is n finite such that, whenever $P' = P'_0 \to P'_1 \to \cdots \to P'_n$, then for some $j \leq n$ we have $P'_j \to \mathbb{C}''\left[Q'_k\{{}^{s'[\mathtt{r}]}/_{x_k}\}\right]$ (for some $\mathbb{C}'', k \in I, s', \mathtt{r}$).

In Def. 5.1, a process P is deadlock-free when it only stops reducing by becoming $\mathbf{0}$; P is terminating when it always reaches $\mathbf{0}$ after a finite number of reductions; P is never-terminating when it reduces forever; P is live (a.k.a. “lock-free” [Kobayashi and Sangiorgi 2010; Padovani 2014]) when all its pending inputs/outputs *can* always eventually communicate with a corresponding output/input; P is strongly live when all its pending inputs/outputs *will* always find a corresponding output/input, enabling communication after a finite number of reductions.

Example 5.2. We now illustrate the differences among the properties in Def. 5.1. Let:

$$P \;=\; P_1 \mid P_2 \qquad \text{where} \begin{cases} P_1 \;=\; s[\mathtt{p}][\mathtt{q}]\sum \mathtt{resp}.P \\ P_2 \;=\; \mathbf{def}\;\; X(x) = x[\mathtt{r}]\sum\{\mathtt{m}_1.X\langle x\rangle\,,\, \mathtt{m}_2.x[\mathtt{p}]\oplus\mathtt{resp}.\mathbf{0}\}\;\; \mathbf{in}\;\; X\langle s[\mathtt{q}]\rangle \mid Q \end{cases}$$

i.e., P_1 implements $\mathtt{p}$, and waits a $\mathtt{response}$ from $\mathtt{q}$; P_2 implements $\mathtt{q}$, and loops every time role $\mathtt{r}$ (whose omitted implementation is in Q) sends $\mathtt{m}_1$; if/when $\mathtt{r}$ chooses to send $\mathtt{m}_2$, then P_2 sends the $\mathtt{response}$ to $\mathtt{p}$, triggering the input in P_1. Now, consider the following implementation of Q:

$$Q \;=\; \mathbf{def}\;\; Y(y) = y[\mathtt{q}]\oplus\mathtt{m}_1.Y\langle y\rangle \;\; \mathbf{in} \;\; Y\langle s[\mathtt{r}]\rangle$$

i.e., $\mathtt{r}$ sends $\mathtt{m}_1$ to $\mathtt{q}$ forever — hence, P reduces forever, which means that P is never-terminating and deadlock-free. But note that the sub-process P_1 never has a chance to receive the desired $\mathtt{response}$ from $\mathtt{q}$: hence, P is *not* live. To address this, we can instead define Q above as:

$$Q \;=\; s[\mathtt{r}][\mathtt{q}]\oplus\mathtt{m}_1.s[\mathtt{r}][\mathtt{q}]\oplus\mathtt{m}_2.\mathbf{0} \mid Q' \quad \text{where} \;\; Q' = \begin{cases} \mathbf{def}\; Z(z) = z[\mathtt{r}'']\oplus\mathtt{m}_3.Z\langle z\rangle \;\mathbf{in} \\ \quad \mathbf{def}\; Z'(z') = z'[\mathtt{r}']\sum\mathtt{m}_3(x).Z'\langle z'\rangle \;\mathbf{in} \\ \quad\quad Z\langle s[\mathtt{r}']\rangle \mid Z'\langle s[\mathtt{r}'']\rangle \end{cases}$$

i.e., $\mathtt{r}$ sends $\mathtt{m}_1$ and then $\mathtt{m}_2$ to $\mathtt{q}$, and this causes $\mathtt{q}$ to send $\mathtt{resp}$ to $\mathtt{p}$ (cf. P_2 above); meanwhile, the sub-process Q' loops, with $\mathtt{r}'$ and $\mathtt{r}''$ exchanging message $\mathtt{m}_3$. With this definition of Q, we obtain that P is live, because P_1 *can* always eventually receive its input while P_2 reduces.

Still, P is *not* strongly live, because the input of P_1 could be arbitrarily delayed by letting Q' reduce forever, without firing the outputs of Q. We can make P strongly live, e.g., by redefining Q' as $Q' = \mathbf{0}$: this guarantees that P_1 *will* receive its input within 3 reductions.[3]

5.2 Session Fidelity

We now prove that if a typing context can reduce, then a typed process P simulates the reduction (Thm. 5.4). A related result can be proved for classic MPST — but in our new theory, it is stronger: we do *not* assume consistency of the typing context, *nor* the existence of a global type projecting it. Session fidelity requires P to be *(1)* not deadlocked, and *(2) productive*, i.e., not trapped in a loop like $\mathbf{def}\ X(x) = X\langle x\rangle\ \mathbf{in}\ X\langle s[\mathbf{p}]\rangle$, if $s[\mathbf{p}]$ needs to be used for input/output: this is formalised in Def. 5.3.

Definition 5.3. Assume $\emptyset \cdot \Gamma \vdash P$. We say that P:

(1) **has guarded definitions** iff in each subterm of the form $\mathbf{def}\ X(x_1{:}S_1, ..., x_n{:}S_n) = Q\ \mathbf{in}\ P'$, for all $i \in 1..n$, $S_i \not\leqslant \mathbf{end}$ implies that a call $Y\langle ..., x_i, ...\rangle$ can only occur in Q as subterm of $x_i[\mathbf{q}]\sum_{j\in J}\mathtt{m}_j(y_j).P_j$ or $x_i[\mathbf{q}]\oplus\mathtt{m}\langle c\rangle.P''$ (i.e., after using x_i for input/output);

(2) **only plays role $\mathbf{p}$ in s, by Γ,** iff: *(i)* P has guarded definitions; *(ii)* $\mathrm{fv}(P) = \emptyset$; *(iii)* $\Gamma = \Gamma_0, s[\mathbf{p}]{:}S$ with $S \not\leqslant \mathbf{end}$ and $\mathrm{end}(\Gamma_0)$; *(iv)* in all subterms $(\nu s'{:}\Gamma')\,P'$ of P, we have $\Gamma' = s'[\mathbf{p}']{:}\mathbf{end}$ (for some $\mathbf{p}'$).

We say "P **only plays role $\mathbf{p}$ in** s" iff $\exists\Gamma : \emptyset \cdot \Gamma \vdash P$, and item 2 holds.

We will explain item 1 of Def. 5.3 shortly (after Thm. 5.4). Item 2 identifies a process that plays exactly *one* role on *one* session: clearly, an ensemble of such processes cannot deadlock by waiting for each other on multiple sessions. All our examples (except a few, duly noted) satisfy Def. 5.3(2).

Now, in Thm. 5.4 we prove that a set of processes involved in a single session simulates the typing context, following its types/protocols. This addresses the typical application scenario of MPST: an ensemble of programs $P_{\mathbf{p}}$ interact on a multiparty session s, each one playing a distinct role $\mathbf{p}$.

Theorem 5.4 (Session Fidelity). *Assume $\emptyset \cdot \Gamma \vdash P$, where Γ is safe, $P \equiv \big|_{\mathbf{p}\in I} P_{\mathbf{p}}$, and $\Gamma = \bigcup_{\mathbf{p}\in I}\Gamma_{\mathbf{p}}$ such that, for each $P_{\mathbf{p}}$, we have $\emptyset \cdot \Gamma_{\mathbf{p}} \vdash P_{\mathbf{p}}$; further, assume that each $P_{\mathbf{p}}$ is either $\mathbf{0}$ (up-to $\equiv$), or only plays $\mathbf{p}$ in s, by $\Gamma_{\mathbf{p}}$. Then, $\Gamma\to$ implies $\exists\Gamma', P'$ such that $\Gamma \to \Gamma'$, $P \to^* P'$ and $\emptyset \cdot \Gamma' \vdash P'$, with Γ' safe, $P' \equiv \big|_{\mathbf{p}\in I} P'_{\mathbf{p}}$, and $\Gamma' = \bigcup_{\mathbf{p}\in I}\Gamma'_{\mathbf{p}}$ such that, for each $P'_{\mathbf{p}}$, we have $\emptyset \cdot \Gamma'_{\mathbf{p}} \vdash P'_{\mathbf{p}}$, and each $P'_{\mathbf{p}}$ is either $\mathbf{0}$ (up-to $\equiv$), or only plays $\mathbf{p}$ in s, by $\Gamma'_{\mathbf{p}}$.*

Note that in Thm. 5.4, P chooses which reduction of Γ to follow: in fact, a selection type in Γ might allow to choose $\mathtt{m}_1, ..., \mathtt{m}_n$ (with different continuations), but P might select only one $\mathtt{m}_k$ (by [T-⊕] in Fig. 2, and subtyping). This observation will be a crucial when reasoning about process liveness (§5.5). Also note that Thm. 5.4 relies on item 1 of Def. 5.3. In fact, by rule [T-**def**] (Fig. 2), an unguarded definition $X(x{:}S) = X\langle x\rangle$ can be typed with *any* S; therefore, we have e.g.:

$$\emptyset \cdot s[\mathbf{p}]{:}\mathbf{q}\oplus\mathtt{m},\ s[\mathbf{q}]{:}\mathbf{p}\&\mathtt{m}\ \vdash\ \mathbf{def}\ X(x{:}\mathbf{q}\oplus\mathtt{m}) = X\langle x\rangle\ \mathbf{in}\ X\langle s[\mathbf{p}]\rangle \mid s[\mathbf{q}][\mathbf{p}]\textstyle\sum\mathtt{m}$$

and the unguarded process above reduces vacuously by calling X infinitely, without matching any typing context reduction; this explains the need of guarded definitions in Thm. 5.4.

5.3 Typing Context Properties

Fig. 5 lists several behavioural properties of typing contexts. In §5.5, we will show how they can statically enforce the run-time process properties discussed in §5.1.

[3]As a more laborious alternative, we could formalise and assume a notion of *fair scheduling*, that eventually fires any action that is persistently enabled; we adopt a similar intuition for type reductions, in Def. 5.5.

(1) Γ is **safe**, written safe(Γ), iff: *(see Def. 4.1)*	$\Gamma \models \nu Z.\left(\begin{array}{l}\forall s, p, q, m, m', S, S'.\\ \quad (\langle s{:}p\oplus q{:}m(S)\rangle\top \land \langle s{:}q\&p{:}m'(S')\rangle\top \Rightarrow \langle s{:}p,q{:}m\rangle\top)\\ \quad \land\ [s{:}p,q{:}m]Z\end{array}\right)$
(2) Γ is **deadlock-free**, written df(Γ), iff: $\Gamma \to^* \Gamma' \not\to$ implies end(Γ')	$\Gamma \models \nu Z.\left(\begin{array}{l}\left(\begin{array}{l}(\forall s, p, q, m.\,[s{:}p,q{:}m]\bot) \Rightarrow\\ \quad \forall p, q, m, S.\,[s{:}p\&q{:}m(S)]\bot \land [s{:}p\oplus q{:}m(S)]\bot\end{array}\right)\\ \land\ \forall p, q, m.\,[s{:}p,q{:}m]Z\end{array}\right)$
(3) Γ is **terminating**, written term(Γ), iff: Γ is deadlock-free, and there is $j \in \mathbb{N}^0$ such that for all $n \geq j$, $\Gamma = \Gamma_0 \to \Gamma_1 \to \cdots \to \Gamma_n$ implies end(Γ_n)	$\Gamma \models \mu Z.\left(\begin{array}{l}\left(\begin{array}{l}(\forall s, p, q, m.\,[s{:}p,q{:}m]\bot) \Rightarrow\\ \quad \forall s, p, q, m, S.\,[s{:}p\&q{:}m(S)]\bot \land [s{:}p\oplus q{:}m(S)]\bot\end{array}\right)\\ \land\ \forall s, p, q, m.\,[s{:}p,q{:}m]Z\end{array}\right)$
(4) Γ is **never-terminating**, written nterm(Γ), iff: $\Gamma \to^* \Gamma'$ implies $\Gamma' \to$	$\Gamma \models \nu Z.(\exists s, p, q, m.\,\langle s{:}p,q{:}m\rangle\top \ \land\ \forall s, p, q, m.\,[s{:}p,q{:}m]Z)$
(5) Γ is **live**, written live(Γ), iff: $\varphi(\Gamma)$, for some φ such that [L-&] whenever $\varphi(\Gamma', s[p]{:}S)$ with $S = q\&_{i\in I} m_i(S_i).S'_i$, $\exists i \in I$: $\exists \Gamma''$: $\Gamma', s[p]{:}S \to^* \Gamma'', s[p]{:}S'_i$ [L-⊕] whenever $\varphi(\Gamma', s[p]{:}S)$ with $S = q\oplus_{i\in I} m_i(S_i).S'_i$, $\forall i \in I$: $\exists \Gamma''$: $\Gamma', s[p]{:}S \to^* \Gamma'', s[p]{:}S'_i$ plus clauses [S-μ], [S-$\to$] (Def. 4.1).	$\Gamma \models \nu Z.\left(\begin{array}{l}\forall s, p, q.\\ \quad \left(\begin{array}{l}(\exists m, S.\,\langle s{:}q\&p{:}m(S)\rangle\top) \Rightarrow\\ \quad \mu Z'.\exists m.\,\langle s{:}p,q{:}m\rangle\top \ \lor\ \exists p', q', m'.\,\langle s{:}p',q'{:}m'\rangle Z'\end{array}\right)\\ \quad \land\\ \quad \forall m.\left(\begin{array}{l}(\exists S.\,\langle s{:}p\oplus q{:}m(S)\rangle\top) \Rightarrow\\ \quad \mu Z'.\langle s{:}p,q{:}m\rangle\top \ \lor\ \exists p', q', m'.\,\langle s{:}p',q'{:}m'\rangle Z'\end{array}\right)\\ \quad \land\\ \quad \forall m.\,[s{:}p,q{:}m]Z\end{array}\right)$
(6) Γ is **live$^+$**, written live$^+(\Gamma)$, iff: $\varphi(\Gamma)$, for φ such that [L-&$^+$] clause [L-&] above; *moreover*, $\Gamma', s[p]{:}S$ belongs to some fair traversal set $\mathbb{X}$ with targets $\mathbb{Y}$ (Def. 5.5) such that, $\forall \Gamma_t \in \mathbb{Y}$, we have $\Gamma_t = \Gamma'', s[p]{:}S'_i$ (for some $\Gamma'', i \in I$) [L-⊕$^+$] clause [L-⊕] above, *plus* the "*moreover...*" part of [L-&$^+$] plus clauses [S-μ], [S-$\to$] (Def. 4.1).	$\Gamma \models \nu Z.\left(\begin{array}{l}\forall s, p, q.\\ \quad \left(\begin{array}{l}(\exists m, S.\,\langle s{:}q\&p{:}m(S)\rangle\top) \Rightarrow\\ \mu Z'.\exists m.\,\langle s{:}p,q{:}m\rangle\top \ \lor\ \exists p', q'.\left(\begin{array}{l}\exists m'.\,\langle s{:}p',q'{:}m'\rangle\top\\ \land\ \forall m'.\,[s{:}p',q'{:}m']Z'\end{array}\right)\end{array}\right)\\ \quad \land\\ \quad \forall m.\left(\begin{array}{l}(\exists S.\,\langle s{:}p\oplus q{:}m(S)\rangle\top) \Rightarrow\\ \mu Z'.\langle s{:}p,q{:}m\rangle\top \ \lor\ \exists p', q'.\left(\begin{array}{l}\exists m'.\,\langle s{:}p',q'{:}m'\rangle\top\\ \land\ \forall m'.\,[s{:}p',q'{:}m']Z'\end{array}\right)\end{array}\right)\\ \quad \land\\ \quad \forall m.\,[s{:}p,q{:}m]Z\end{array}\right)$
(7) Γ is **live^{++}**, written live$^{++}(\Gamma)$, iff: $\varphi(\Gamma)$, for φ such that [L-&$^{++}$] clause [L-&] above; *moreover*, $\exists n \in \mathbb{N}^0$ such that, whenever $\Gamma', s[p]{:}S = \Gamma_0 \to \Gamma_1 \to \cdots \to \Gamma_n$, then $\exists j \leq n, \Gamma''$ such that $\Gamma_j \to \Gamma'', s[p]{:}S'_i$ (for some $i \in I$) [L-⊕$^{++}$] clause [L-⊕] above, *plus* the "*moreover...*" part of [L-&$^{++}$] plus clauses [S-μ], [S-$\to$] (Def. 4.1).	$\Gamma \models \nu Z.\left(\begin{array}{l}\forall s, p, q.\\ \quad \left(\begin{array}{l}(\exists m, S.\,\langle s{:}q\&p{:}m(S)\rangle\top) \Rightarrow\\ \mu Z'.\exists m.\,\langle s{:}p,q{:}m\rangle\top \ \lor\ \left(\begin{array}{l}\exists s', p', q', m'.\,\langle s'{:}p',q'{:}m'\rangle\top\\ \land\ \forall s', p', q', m'.\,[s'{:}p',q'{:}m']Z'\end{array}\right)\end{array}\right)\\ \quad \land\\ \quad \forall m.\left(\begin{array}{l}(\exists S.\,\langle s{:}p\oplus q{:}m(S)\rangle\top) \Rightarrow\\ \mu Z'.\langle s{:}p,q{:}m\rangle\top \ \lor\ \left(\begin{array}{l}\exists s', p', q', m'.\,\langle s'{:}p',q'{:}m'\rangle\top\\ \land\ \forall s', p', q', m'.\,[s'{:}p',q'{:}m']Z'\end{array}\right)\end{array}\right)\\ \quad \land\\ \quad \forall m.\,[s{:}p,q{:}m]Z\end{array}\right)$

Fig. 5. Properties of typing contexts. Each property is presented in two equivalent formalisations: the left-side ones are based on the notation and definitions introduced up to § 5.4 (excluded); the right-side ones are μ-calculus formulas (explained in § 6), and allow to verify typing contexts via model checking (e.g., with tools like mCRL2 [Groote and Mousavi 2014]).

- Γ is *deadlock-free* iff it stops reducing only when it only contains **end**s;
- Γ is *terminating* iff it always reaches a final configuration, in a finite number of steps;
- Γ is *never-terminating* iff it never stops reducing;
- Γ is *live, live*$^{+}$ or *live*$^{++}$ iff each branching/selection can be eventually fired.

The intuition behind live/live^{+}/live^{++} is the following. Take a typing context $\Gamma, s[\mathtt{p}]{:}S$. If such a context is live, then, by clause [L-&] of Fig. 5(5), if S is an external choice, then Γ can reduce until *some* branch of S is triggered; and by clause [L-⊕], if S is an internal choice, then Γ can reduce allowing to send *each* message of S. The clauses of liveness^{+} are stricter: they ensure that, under "fair scheduling" (details below) the interaction with S will be enabled in a finite number of steps. The clauses of liveness^{++} are even stricter, and ensure that the interaction with S will be enabled within a finite number of steps, no matter how other roles are scheduled. We will give examples and more explanations shortly (Ex. 5.10, Ex. 5.11, Ex. 5.14, Thm. 5.15). But first, we explain what "under fair scheduling" means: roughly, we ensure that there is a set of roles whose interactions *always* cause a desired input/output to meet a corresponding output/input. This requires some sophistication, and the formalisation of the "fair traversal set" mentioned in the definition of liveness^{+} (Fig. 5(6)).

Definition 5.5 (Fair traversal set). Let $\mathbb{X}, \mathbb{Y}$ be sets of typing contexts. We say that $\mathbb{X}$ *is a fair traversal set with targets* $\mathbb{Y}$ iff $\mathbb{X}$ is closed under the rules:

$$\frac{\Gamma \in \mathbb{Y}}{\Gamma \in \mathbb{X}}\ \text{[TS-Target]} \qquad \frac{\exists s, \mathtt{p}, \mathtt{q}: \quad \exists \mathtt{m}: \Gamma \xrightarrow{s:\mathtt{p},\mathtt{q}:\mathtt{m}} \qquad \forall \mathtt{m}: \Gamma \xrightarrow{s:\mathtt{p},\mathtt{q}:\mathtt{m}} \Gamma' \text{ implies } \Gamma' \in \mathbb{X}}{\Gamma \in \mathbb{X}}\ \text{[TS-Comm]}$$

Def. 5.5 says that if a fair traversal set $\mathbb{X}$ contains a typing context Γ, then $\mathbb{X}$ also contains (part of) Γ's reductions (inductive rule [TS-Comm]), reaching one of the target contexts in $\mathbb{Y}$ (base rule [TS-Target]). Notably, by rule [TS-Comm], for each reduction of Γ, it is enough to choose just *two* roles $\mathtt{p}, \mathtt{q}$ who can interact (clause "$\exists \mathtt{m}: \Gamma \xrightarrow{s:\mathtt{p},\mathtt{q}:\mathtt{m}}$"), as long as, for *all* interactions they can engage in, the corresponding reductum belongs to $\mathbb{X}$ (clause "$\ldots\Gamma' \in \mathbb{X}$"). Consequently, if we prove that $\mathbb{X}$ is a fair traversal set with targets $\mathbb{Y}$, then any $\Gamma \in \mathbb{X}$ is supported by an inductive derivation $\mathcal{D}$ — that, in turn, shows how we can reach some $\Gamma' \in \mathbb{Y}$ in a finite number of steps, by choosing a set of participants and following *any* of their possible interactions (one per instance of [TS-Comm] in $\mathcal{D}$).

Example 5.6. By Def. 5.5, fair traversal sets are inductively defined: this excludes cases where target elements are reachable, but can be "infinitely delayed" by choices and recursion. E.g., let:

$$\begin{array}{l} \Gamma = s[\mathtt{p}]{:}\mu\mathbf{t}.\mathtt{q}\oplus\{\mathtt{m}_1.\mathbf{t},\ \mathtt{m}_2\},\ s[\mathtt{q}]{:}\mu\mathbf{t}.\mathtt{p}\&\{\mathtt{m}_1.\mathbf{t},\ \mathtt{m}_2.\mathtt{r}\oplus\mathtt{m}_3\},\ s[\mathtt{r}]{:}\mathtt{q}\&\mathtt{m}_3 \\ \Gamma' = s[\mathtt{p}]{:}\mathbf{end}, s[\mathtt{q}]{:}\mathbf{end}, s[\mathtt{r}]{:}\mathbf{end} \end{array} \quad \text{and thus,} \quad \Gamma \xrightarrow{s:\mathtt{p},\mathtt{q}:\mathtt{m}_2}\xrightarrow{s:\mathtt{q},\mathtt{r}:\mathtt{m}_3} \Gamma'$$

Note that Γ is live, and Γ' is reachable — and yet, we *cannot* define a fair traversal set $\mathbb{X}$ containing Γ, with a target set $\mathbb{Y} = \{\Gamma'\}$. This is because $\mathtt{p}, \mathtt{q}$ can interact infinitely by exchanging $\mathtt{m}_1$, yielding the infinite run $\Gamma \xrightarrow{s:\mathtt{p},\mathtt{q}:\mathtt{m}_1} \Gamma \xrightarrow{s:\mathtt{p},\mathtt{q}:\mathtt{m}_1} \cdots$; consequently, to support $\Gamma \in \mathbb{X}$ we would need an inductive derivation with an *infinite* series of instances of rule [TS-Comm] — i.e., the derivation would be invalid.

Example 5.7. Fair traversal sets can be defined when elements of the target set are reachable, but can be infinitely delayed by "unfair scheduling." E.g., consider:

$$\begin{array}{rcl} \Gamma & = & s[\mathtt{p}]{:}\mathtt{q}\oplus\mathtt{m}_1.\mathtt{q}'\oplus\mathtt{m}_2,\ s[\mathtt{q}]{:}\mathtt{p}\&\mathtt{m}_1,\ s[\mathtt{q}']{:}\mathtt{p}\&\mathtt{m}_2,\ s[\mathtt{r}]{:}\mu\mathbf{t}.\mathtt{r}'\oplus\mathtt{m}_2.\mathbf{t},\ s[\mathtt{r}']{:}\mu\mathbf{t}.\mathtt{r}\&\mathtt{m}_2.\mathbf{t} \\ \Gamma' & = & s[\mathtt{p}]{:}\mathbf{end},\qquad\qquad s[\mathtt{q}]{:}\mathbf{end},\ s[\mathtt{q}']{:}\mathbf{end},\ s[\mathtt{r}]{:}\mu\mathbf{t}.\mathtt{r}'\oplus\mathtt{m}_2.\mathbf{t},\ s[\mathtt{r}']{:}\mu\mathbf{t}.\mathtt{r}\&\mathtt{m}_2.\mathbf{t} \end{array}$$

Note that Γ is live, and Γ' is reachable from Γ, via the reductions $\Gamma \xrightarrow{s:\mathtt{p},\mathtt{q}:\mathtt{m}_1}\xrightarrow{s:\mathtt{p},\mathtt{q}':\mathtt{m}_2} \Gamma'$; however, Γ' can be infinitely delayed in the unfair run $\Gamma \xrightarrow{s:\mathtt{r},\mathtt{r}':\mathtt{m}_2} \Gamma \xrightarrow{s:\mathtt{r},\mathtt{r}':\mathtt{m}_2} \cdots$ that never fires the communication between $\mathtt{p}$ and $\mathtt{q}$, and thus, never enables the interaction between $\mathtt{p}$ and $\mathtt{q}'$. Yet, unlike Ex. 5.6, we *can* define a fair traversal set $\mathbb{X} = \{\Gamma, \Gamma'\}$, with target $\mathbb{Y} = \{\Gamma'\}$: in fact, we can

Table 1. Verification of the multiparty protocols in Fig. 4 against the properties in Fig. 5. The results for protocol (3) hold for $n \geq 1$, while the results for protocol (4) hold for $n \geq 2$.

	consistent	safe	deadlock-free	live	live$^+$	live^{++}	never-terminat.	terminat.
(1) OAuth2 fragment	false	**true**	**true**	**true**	**true**	**true**	false	**true**
(2) Rec. two-buyers	false	**true**	**true**	**true**	false	false	false	false
(3) Rec. map/reduce	false	**true**	**true**	**true**	**true**	**true**	false	false
(4) MP workers	false	**true**	**true**	**true**	**true**	false	false	false

build a finite derivation that supports $\Gamma \in \mathbb{X}$ by instantiating rule [TS-Comm] twice — choosing p, q for the fist reduction, and then p, q' to reach the axiom [TS-Target], ignoring the interactions between r, r'.

Ex. 5.6 and Ex. 5.7 clarify why live$^+$ in Fig. 5(6) requires the existence of a certain traversal set: this ensures that, when Γ has some pending input/output, then under "fair scheduling," Γ can reach a target Γ_t where such input/output has been fired, by interacting with a matching output/input.

5.4 Relationships Between Typing Context Properties

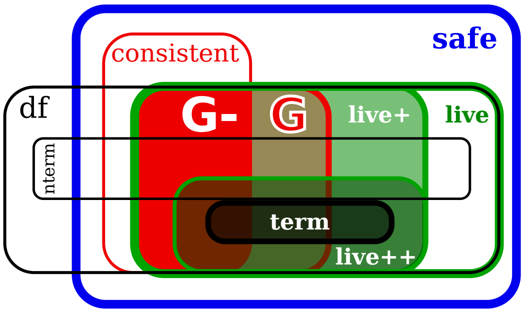
We now study how typing context properties are related: this is formalised in Lemma 5.9 below, that also conveys the expressiveness of our new type system (Remark 5.12).

To cover classic MPST theory, we first define *projected typing contexts*, in Def. 5.8; note that the projections with plain and full merging correspond to claims **(C1)** and **(C2)** in §3.1, respectively.

Definition 5.8. We say that Γ **is the full (resp. plain) projection of** G **for session** s, written $\mathrm{fproj}_{G,s}(\Gamma)$ (resp. $\mathrm{pproj}_{G,s}(\Gamma)$), iff $\Gamma = \{s[\mathsf{p}]{:}G{\upharpoonright}\mathsf{p}\}_{\mathsf{p}\in\mathrm{roles}(G)}$, where $G{\upharpoonright}\mathsf{p}$ is the *projection with full merging (resp. plain merging)* in Def. 3.3.

Lemma 5.9. *For all* Γ*, the following (non-)implications hold:*

(1) $\mathrm{consistent}(\Gamma) \;\not\Longleftarrow\!\!\Longrightarrow\; \mathrm{safe}(\Gamma)$;
(2) $\mathrm{live}(\Gamma) \;\not\Longleftarrow\!\!\Longrightarrow\; \mathrm{safe}(\Gamma)$;
(3) $\mathrm{live}(\Gamma) \;\not\Longleftarrow\!\!\Longrightarrow\; \mathrm{df}(\Gamma)$;
(4) $\mathrm{nterm}(\Gamma) \;\not\Longleftarrow\!\!\Longrightarrow\; \mathrm{df}(\Gamma)$;
(5) $\mathrm{consistent}(\Gamma) \;\not\Longleftarrow\!\!\not\Longrightarrow\; \mathrm{df}(\Gamma)$;
(6) $\mathrm{consistent}(\Gamma) \wedge \mathrm{df}(\Gamma) \;\not\Longleftarrow\!\!\not\Longrightarrow\; \mathrm{live}(\Gamma)$;
(7) $\mathrm{live}^{++}(\Gamma) \;\not\Longleftarrow\!\!\Longrightarrow\; \mathrm{live}^{+}(\Gamma) \;\not\Longleftarrow\!\!\Longrightarrow\; \mathrm{live}(\Gamma)$;
(8) $\mathrm{term}(\Gamma) \;\not\Longleftarrow\!\!\Longrightarrow\; \mathrm{live}^{++}(\Gamma)$;
(9) assume $\mathrm{dom}(\Gamma) = \{s\}$ *(Def. 2.6). Then:*
$\exists G : \mathrm{fproj}_{G,s}(\Gamma) \;\not\Longleftarrow\!\!\Longrightarrow\; \mathrm{live}^{+}(\Gamma)$.



In the diagram, the "safe" set contains all typing contexts supported by our general type system. The red subsets are the classic MPST theory: $\mathbb{G}$ contains all contexts projected by some global type; its subset $\mathbb{G}-$ only has *consistent* typing contexts, i.e. the only class of global types for which classic MPST proves type safety: this class excludes our example in §1, and also all protocols in Fig. 4, and more (see Ex. 5.10 and Ex. 5.11 below). Notably, in item (9), we prove that all projected contexts are live$^+$: this is discussed in Remark 5.16 later.

Example 5.10. The protocols described in Fig. 4 are verified in Table 1. We observe:

- all protocols are safe and live, but *none of them* is consistent: hence, they are *not* supported by the classic MPST theory;
- all protocols are live^{+}, except recursive two-buyers (2): this is because it allows **a**lice and **b**ob to bargain forever by exchanging split/no messages, without ever involving the **s**tore (that will keep waiting for **a**lice to send either buy or no). This violates clause [L-&$^{+}$] of Fig. 5(6), because we cannot find any traversal set whose targets trigger the **s**tore's pending input (the issue is similar to Ex. 5.6);
- two protocols are not live^{++}: recursive two-buyers (as expected, by the point above and the contrapositive of Lemma 5.9(7)), and MP workers (4). The latter is not live^{++} because each triplet of workers $\mathbf{wa}_i, \mathbf{wb}_i, \mathbf{wc}_i$ ($i \in 1..n \geq 2$) can loop independently from the others; therefore, the interaction between, e.g., two workers in triplet 1 might be delayed for an unbounded number of transitions, while triplet 2 keeps progressing. Note that this scenario arises if the roles are scheduled unfairly; otherwise, each enabled interaction *will* be eventually fired, and this is reflected by the fact that the MP workers protocol is live^{+};
- only the OAuth2 fragment (1) is terminating — while the other protocols are *neither* terminating, *nor* never-terminating: i.e., they might loop forever, but depending on the choices of one or more roles, they can reach a terminated state (where all roles have type **end**).

Example 5.11. We now provide some more small examples of multiparty protocols and their properties, complementing those discussed Ex. 5.10.

$\Gamma_A = s[\mathbf{p}]{:}\mathbf{q}\&\mathtt{m}_1.\mathbf{r}\oplus\mathtt{m}_3,\ s[\mathbf{q}]{:}\mathbf{r}\&\mathtt{m}_2.\mathbf{p}\oplus\mathtt{m}_1,\ s[\mathbf{r}]{:}\mathbf{p}\&\mathtt{m}_3.\mathbf{q}\oplus\mathtt{m}_2$ is consistent (hence safe), but *not* live *nor* deadlock-free: this is because its inputs/outputs, albeit dual, occur in the wrong order.

$\Gamma_B = s[\mathbf{p}]{:}\mu\mathbf{t}.\mathbf{q}\oplus\mathtt{m}_1.\mathbf{t},\ s[\mathbf{q}]{:}\mu\mathbf{t}.\mathbf{p}\&\mathtt{m}_1.\mathbf{t},\ s[\mathbf{r}]{:}\mathbf{p}\&\mathtt{m}_2$ is consistent, deadlock-free and safe, but *not* live: in fact, $s[\mathbf{p}]$,$s[\mathbf{q}]$ reduce infinitely, but $s[\mathbf{r}]$ cannot fire its input (violating [L-&] in Fig. 5).

$\Gamma_C = s[\mathbf{p}]{:}S, s[\mathbf{q}]{:}\mathbf{p}\&\mathtt{m}(S).\mathbf{end}$ with $S = \mu\mathbf{t}.\mathbf{q}\oplus\mathtt{m}(\mathbf{t}).\mathbf{end}$ (from [Bernardi and Hennessy 2016, Ex. 1.2]) is terminating (hence live^{++}, and safe), but *not* projectable from any global type, *nor* consistent: this is because a recursion variable $\mathbf{t}$ occurs as payload in S, which is disallowed by Def. 3.3 and Def. 3.8. Notably, Γ_C types the process below (from [Bernardi and Hennessy 2016, Ex. 1.2]): it creates infinitely many sessions s' where $\mathbf{p}$ and $\mathbf{q}$ exchange one message $\mathtt{m}$ (note that this process, although deadlock-free, does not satisfy Def. 5.3(2)).

$$\emptyset \cdot \Gamma_C \ \vdash\ \mathbf{def}\ \ X(x{:}S, y{:}\mathbf{p}\&\mathtt{m}(S)) = P\ \ \mathbf{in}\ \ X\langle s[\mathbf{p}], s[\mathbf{q}]\rangle$$

$$\text{where}\ \ P = \left(\nu s'{:}\Gamma'_C\right)\left(x[\mathbf{q}]\oplus\mathtt{m}\langle s'[\mathbf{p}]\rangle.\mathbf{0} \mid y[\mathbf{p}]\textstyle\sum \mathtt{m}(z).X\langle z, s'[\mathbf{q}]\rangle\right)\ \ \text{with}\ \ \Gamma'_C = s'[\mathbf{p}]{:}S, s'[\mathbf{q}]{:}\mathbf{p}\&\mathtt{m}(S).\mathbf{end}$$

Remark 5.12. *By Lemma 5.9(1,9), our general session type system instantiated with $\varphi = \mathrm{fproj}_{\mathrm{G},s}$ subsumes the classic MPST theory, and also proves subject reduction and type safety in presence of "full-merging" global type projections: this is because consistency/projectability are limited syntactic approximations of safety/liveness. Hence, the typing rule [T-νClassicG] in §3 is valid in our theory, and we can type our opening example (Ex. 4.7), and support complex protocols rejected by classic MPST, such as all those listed in Fig. 4. This retroactively fixes some flawed results in literature, described in §3.1 (claim (C2)), and impacting the works listed in §8. Further, we support protocols for which no global type exists: see Ex. 5.10 (case "recursive two-buyers") and Ex. 5.11 (case Γ_C).*

5.5 Static Verification of Run-Time Process Properties

We now show that, by using the type-level properties in Fig. 5, we can predict and constrain the run-time behaviour of processes. Roughly, the intuition is: if we have $\Gamma \vdash P$, and some property in Fig. 5 holds for Γ, then a similar corresponding property from Def. 5.1 holds for P. From this it follows that, to ensure that a closed process $(\nu s)\,P$ has a desired property from Def. 5.1, we can correspondingly instantiate φ in Def. 4.6, and check if the judgement "$\emptyset \vdash (\nu s{:}\Gamma)\,P$ with φ" holds.

First, we highlight that all typing context properties mentioned thus far are decidable (Thm. 5.13 below) — unlike the run-time process properties in Def. 5.1. This is clear for consistency and projectability, that are syntactic and inductive; others (safety, liveness,...) are decidable because, by Def. 2.8, typing contexts have finite-state transition systems. Consequently, by Thm. 4.11, type checking is decidable, if φ is instantiated with any property listed in Thm. 5.13.

Theorem 5.13 (Decidability of φ). *$\varphi(\Gamma)$ is decidable, for all Γ, and for all φ such that*

$$\varphi \in \{\text{consistent}, \text{fproj}_{G,s}, \text{pproj}_{G,s}, \text{safe}, \text{term}, \text{nterm}, \text{df}, \text{live}, \text{live}^{+}, \text{live}^{++}\} \qquad \textit{(for any } G\textit{)}$$

Now, assume $\Gamma \vdash P$. To predict the run-time behaviour of P from Γ, we need to overcome a complication: it might seem that if Γ is live (Fig. 5(5)), then P should be live, too. But this is *not* the case, due to a subtle interaction between the typing rule [T-Sub] in Fig. 2, and the fact that *supertyping does not preserve liveness*: this issue (that is related to the problem of *fair subtyping*, studied by Padovani [2016]), is illustrated in Ex. 5.14 below. For this reason, in Thm. 5.15 we guarantee process liveness via the stronger type-level property live^{+}: this is the payoff of fair traversal sets (Def. 5.5).

Example 5.14. Take Γ with the rec. two-buyer protocol (Fig. 4(2)): it is live (Table 1). Now, let:

$$\Gamma' = \begin{cases} s[\mathbf{a}] : \mathbf{s}\oplus\mathtt{query}(\mathrm{Str}).\mathbf{s}\&\mathtt{price}(\mathrm{Int}).\mu\mathbf{t}.\mathbf{b}\oplus\mathtt{split}(\mathrm{Int}).\mathbf{b}\&\{\mathtt{yes}(\mathrm{Int}).\mathbf{s}\oplus\mathtt{buy}\,,\ \mathtt{no}.\mathbf{t}\} \\ s[\mathbf{s}] : \mathbf{a}\&\mathtt{query}(\mathrm{Str}).\mathbf{a}\oplus\mathtt{price}(\mathrm{Int}).\mathbf{a}\&\{\mathtt{buy}.\mathbf{end}\,,\ \mathtt{no}.\mathbf{end}\} \qquad \textit{(as in Fig. 4(2))} \\ s[\mathbf{b}] : \mu\mathbf{t}.\mathbf{a}\&\{\mathtt{split}(\mathrm{Int}).\mathbf{a}\oplus\mathtt{no}.\mathbf{t}\,,\ \mathtt{cancel}.\mathbf{end}\} \end{cases}$$

i.e., the types of **a**lice and **b**ob in Γ' are *supertypes* (Def. 2.5) of those in Γ: **a**lice never chooses to send `cancel` to **b**ob, who in turn always answers `no` to all `split` proposals. We have $\Gamma \leqslant \Gamma'$ (Def. 2.5) and Γ' is safe (Lemma 4.5), but *not live*: after sending the `price`, the **s**tore will wait for either `buy` or `no` from **a**lice, but neither message will ever be sent, while **a**lice and **b**ob loop by exchanging `split`/`no`. Consequently, a process P typed by Γ' can have two sub-processes implementing **a**lice and **b**ob that interact forever, while a sub-process implementing the **s**tore waits for a buy/no message, but will never receive it: hence, P is not live, as it does not satisfy Def. 5.1(2). Now, note that such P is also typed by Γ (via rule [T-Sub] in Fig. 2): i.e., a live typing context can type a *non*-live process.

Theorem 5.15. *Assume $\emptyset \cdot \Gamma \vdash P$, with Γ safe, $P \equiv \prod_{\mathbf{p}\in I} P_{\mathbf{p}}$, each $P_{\mathbf{p}}$ having guarded definitions and either being $\mathbf{0}$ (up-to $\equiv$), or only playing role $\mathbf{p}$ in s. Then, (1) $\text{df}(\Gamma)$ implies that P is deadlock-free; (2) $\text{term}(\Gamma)$ implies that P is terminating; (3) $\text{nterm}(\Gamma)$ implies that P is never-terminating; (4) $\text{live}^{+}(\Gamma)$ implies that P is live; and (5) $\text{live}^{++}(\Gamma)$ implies that P is strongly live.*

Proof. The results follow by Thm. 5.4 (session fidelity). For (4) we also use the fact that, if $\text{live}^{+}(\Gamma)$ and $\Gamma \leqslant \Gamma'$, then $\text{live}^{+}(\Gamma')$. □

Remark 5.16. *With Lemma 5.9(9) and Thm. 5.15(4), we uncover that global types / projections (Fig. 3) are ways to produce live^{+} typing contexts, and ensure that processes are live. Since Thm. 5.15 does not need the technicalities of Fig. 3, our theory and results are more general than classic MPST. And importantly, the premises of all cases of Thm. 5.15 are decidable (by Thm. 5.13 and Thm. 4.11).*

6 VERIFYING TYPE-LEVEL PROPERTIES VIA MODEL CHECKING

Our new MPST theory (§ 4) is parametric on a general property φ, that is not constrained by syntactic duality/consistency. In this section, we leverage this distinguishing feature to integrate type checking and model checking, in two steps: *(1)* we show how to express φ as a *modal μ-calculus formula*, and *(2)* we use a model checker (through the paper's companion artifact) to verify whether the transitions of Γ satisfy the μ-calculus version of φ. This provides a practical method to verify whether $\varphi(\Gamma)$ holds — e.g., in rule [TGen-ν] (Def. 4.6), and in Thm. 5.15.

We focus on a fragment of the μ-calculus with data, adopting a formulation based on [Groote and Mousavi 2014, §6.5]. Let α range over the labels in Def. 2.8 — i.e., α can have the form $s{:}\mathbf{p}\&\mathbf{q}{:}\mathtt{m}(S)$

for input, or $s{:}\mathsf{p}{\oplus}\mathsf{q}{:}\mathtt{m}(S)$ for output, or $s{:}\mathsf{p}{,}\mathsf{q}{:}\mathtt{m}$ for communication. Then, *μ-calculus formulas* are defined as follows, where d ("data") ranges over sessions, roles, message labels, and session types:

$$\phi \;::=\; \top \mid \bot \mid [\alpha]\phi \mid \langle\alpha\rangle\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \Rightarrow \phi_2 \mid \mu Z.\phi \mid \nu Z.\phi \mid Z \mid \forall \mathsf{d}.\phi \mid \exists \mathsf{d}.\phi$$

A formula ϕ accepts or rejects a typing context Γ depending on the sequences of actions that Γ can fire along its transitions. A formula can be either: true/false ($\top/\bot$), i.e., accept any/no typing context; box modality $[\alpha]\phi$ ("for all transitions with label α, the reached typing context must satisfy ϕ"); diamond modality $\langle\alpha\rangle\phi$ ("for some transition with label α, the reached typing context satisfies ϕ"); implication $\Rightarrow$; least/greatest fixed point $\mu Z.\phi/\nu Z.\phi$, allowing to iterate ϕ for a finite/infinite number of times; a variable Z, for iteration; and universal/existential quantification $\forall \mathsf{d}.\phi/\exists \mathsf{d}.\phi$. When a typing context Γ satisfies a formula ϕ, we write $\Gamma \models \phi$.

Example 6.1. The μ-calculus formula $\phi = \exists s.\exists \mathsf{p}.\exists \mathsf{q}.\exists \mathtt{m}.\exists S.\langle s{:}\mathsf{p}{\oplus}\mathsf{q}{:}\mathtt{m}(S)\rangle\top$ says: "accept a typing context if, for some session s, roles p and q, message label $\mathtt{m}$, and type S, it can perform an output action $s{:}\mathsf{p}{\oplus}\mathsf{q}{:}\mathtt{m}(S)$" — and after such a transition, the reached typing context is always accepted, by $\top$. Therefore, if we take the typing context $\Gamma = s[\mathsf{r}]{:}\mathsf{r}'{\oplus}\mathtt{msg}(\mathsf{Str}).\mathbf{end}$, then we have $\Gamma \xrightarrow{s:\mathsf{r}\oplus\mathsf{r}':\mathtt{msg}(\mathsf{Str})}$ (by Def. 2.8), which means that Γ satisfies ϕ — in symbols, $\Gamma \models \phi$. Moreover, Γ satisfies the formula $\forall s.\forall \mathsf{p}.\forall \mathsf{q}.\forall \mathtt{m}.[s{:}\mathsf{p}{,}\mathsf{q}{:}\mathtt{m}]\bot$, that holds when *no* communication is possible, for any role: in fact, the formula says that any communication would reach a context rejected by $\bot$.

Instead, if we take the formula $\phi' = \exists s.\exists \mathsf{p}.\exists \mathsf{q}.\exists \mathtt{m}.\langle s{:}\mathsf{p}{,}\mathsf{q}{:}\mathtt{m}\rangle\top$, then Γ above does *not* satisfy ϕ', because it requires a communication transition to be enabled. However, if we extend Γ as $\Gamma' = \Gamma, s[\mathsf{r}']{:}\mathsf{r}\&\mathtt{msg}(\mathsf{Str}).\mathbf{end}$, then we have both $\Gamma' \models \phi$ and $\Gamma' \models \phi'$ — and thus, $\Gamma' \models \phi \wedge \phi'$.

Example 6.2 (Formulas in Fig. 5). We now describe the μ-calculus formulas in Fig. 5:

- **safety (1)** checks that, if an output $\mathtt{m}$ and an input $\mathtt{m}'$ are enabled between two roles p and q, then they can communicate via $\mathtt{m}$ (i.e., by Def. 2.8, the output message $\mathtt{m}$ must be supported by the recipient). This must hold for any context reachable via communication transitions: this is enforced by the greatest fixed point $\nu Z. \ldots$ and the clause $\ldots \wedge [s{:}\mathsf{p}{,}\mathsf{q}{:}\mathtt{m}]Z$;
- **deadlock-freedom (2)** checks whether communication is possible; if not ("$\forall\ldots.[s{:}\mathsf{p}{,}\mathsf{q}{:}\mathtt{m}]\bot$", that holds only when no roles can interact, cf. Ex. 6.1), then ($\Rightarrow$) there must be no input nor output transitions enabled — i.e., all typing context entries must be $\mathbf{end}$. This must hold for any context reachable via communications: it is enforced by $\nu Z. \ldots$ and $\ldots \wedge \forall\ldots.[s{:}\mathsf{p}{,}\mathsf{q}{:}\mathtt{m}]Z$;
- **termination (3)** is similar to deadlock-freedom, but uses a *least* fixed point $\mu Z. \ldots$: hence, the clause $\ldots \wedge \forall\ldots.[s{:}\mathsf{p}{,}\mathsf{q}{:}\mathtt{m}]Z$ can only iterate for a *finite* number of times, and then no communications, nor inputs, nor outputs must be enabled — i.e., all context entries are $\mathbf{end}$;
- **never-termination (4)** checks that in any context reachable via communication transitions ($\nu Z. \ldots$ and $\ldots \wedge \forall\ldots.[s{:}\mathsf{p}{,}\mathsf{q}{:}\mathtt{m}]Z$), some further communication is possible ($\exists\ldots.\langle s{:}\mathsf{p}{,}\mathsf{q}{:}\mathtt{m}\rangle\top$);
- **liveness (5)** checks that, if an input or output between two roles p and q is enabled, then ($\Rightarrow$) a corresponding communication can be fired, after a finite sequence of communications among any role. The sequence is built with a least fixed point $\mu Z'. \ldots$, that can iterate on the clause $\ldots \vee \exists\ldots.\langle s{:}\mathsf{p}'{,}\mathsf{q}'{:}\mathtt{m}'\rangle Z'$ for a finite number of times. The top-level greatest fixed point $\nu Z. \ldots$ repeats the check for all contexts reachable via communication (clause $\ldots \wedge \forall\ldots.[s{:}\mathsf{p}{,}\mathsf{q}{:}\mathtt{m}]Z$);
- **liveness^{+} (6)** is similar to liveness, but the nested fixed points $\mu Z'. \ldots$ build finite sequences of communications by picking a pair of roles p', q' at each step, and following *all* their interactions, until a communication between p, q is enabled. This corresponds to building the fair traversal set (Def. 5.5) required by the left-side definition of live^{+} in Fig. 5;
- **liveness^{++} (7)** is also similar to liveness, but the nested fixed points $\mu Z'. \ldots$ build finite sequences by following *any* communication between *any* pair of roles, until a communication

Table 2. Average time (in seconds ± std. dev.) for the verification of the protocols in Fig. 4. Protocols (3) and (4) are instantiated with $n=3$. The outcome of the verification is shown in Table 1. *(Benchmarking specs: Intel Core i7-4790 CPU, 3.60GHz, 16 GB RAM, mCRL2 201808.0 invoked 30 times (by* `mpstk`*) with:* `pbes2bool -strategy=2`*)*

	states	safe	deadlock-free	live	live^{+}	live^{++}	never-terminat.	terminat.
(1) OAuth2 fragment	37	1.00 ± 0%	1.00 ± 0%	1.00 ± 0%	1.00 ± 0%	1.00 ± 0%	1.00 ± 0%	0.98 ± 9%
(2) Rec. two-buyers	85	1.00 ± 0%	1.00 ± 0%	1.00 ± 0%	1.00 ± 0%	1.00 ± 0%	1.00 ± 0%	0.99 ± 3%
(3) Rec. map/reduce	2561	1.00 ± 0%	1.00 ± 0%	1.00 ± 0%	1.00 ± 0%	1.00 ± 0%	1.00 ± 0%	0.99 ± 3%
(4) MP workers	442369	1.01 ± 4%	0.98 ± 8%	0.98 ± 9%	1.03 ± 14%	1.02 ± 7%	0.99 ± 6%	1.00 ± 1%

between $\mathbf{p}, \mathbf{q}$ is enabled. This ensures that, along any execution path, after a finite number of steps, $\mathbf{p}$ and $\mathbf{q}$ *will* be able to interact, as in the left-side definition of live^{++} in Fig. 5.

Implementation. This paper has a companion artifact: a toolkit, called `mpstk` (*"MultiParty Session Types toolKit"*), that verifies the properties listed Fig. 5 (and described in Ex. 6.2). It is available at:

https://alcestes.github.io/mpstk

Internally, `mpstk` uses the mCRL2 model checker [Groote and Mousavi 2014], in combination with the theory in §2.2 and §4 (e.g., `mpstk` checks subtyping, as per Def. 2.5). We used `mpstk` to verify the protocols in Fig. 4: the results are in Table 1. We also measured the time needed to verify each case: the results are in Table 2. In all instances, the verification takes around one second. Notably, this also holds for the multiparty workers protocol (4), although it has 12000× more states than the OAuth2 fragment (1). This state space explosion is due to the interleaving of multiple parallel components — but still, its impact on verification time is minimal: in fact, the properties in Fig. 5 only follow the communication transitions of a typing context Γ, whereas the input and output transitions of Γ are checked for their presence/absence, but *not* followed to their destination state. Hence, mCRL2 can verify our formulas in Fig. 5 without exploring the whole state space of Γ.

7 ASYNCHRONOUS MULTIPARTY SESSION π-CALCULUS

In its original formulation [Bettini et al. 2008; Honda et al. 2008], the MPST π-calculus has *asynchronous* buffered semantics, to model typical "real-world" distributed message-passing programs. Our new theory extends to asynchrony, overcoming challenges due to queue handling and decidability.

***NOTE:** this section is a summary of the results that are discussed, in full detail, in §C–§G.*

Asynchronous MPST. We give an intuition of the asynchronous calculus with an example:

$$\begin{array}{l} s[\mathbf{p}][\mathbf{q}]\oplus \mathtt{m}\langle s'[\mathbf{r}]\rangle.P \mid s[\mathbf{q}][\mathbf{p}]\sum \mathtt{m}(x).Q \mid s \blacktriangleright \epsilon \\ \quad \to \quad P \mid s[\mathbf{q}][\mathbf{p}]\sum \mathtt{m}(x).Q \mid s \blacktriangleright (\mathbf{p},\mathbf{q},\mathtt{m}\langle s'[\mathbf{r}]\rangle)\cdot\epsilon \quad \to \quad P \mid Q\{s'[\mathbf{r}]/x\} \mid s \blacktriangleright \epsilon \end{array} \tag{10}$$

In the topmost process, $s \blacktriangleright \epsilon$ is the (empty) *message queue of session* s (not present in the calculus of §2.1). The first reduction enqueues the *pending message* $(\mathbf{p},\mathbf{q},\mathtt{m}\langle s'[\mathbf{r}]\rangle)$, meaning that $\mathbf{p}$ has sent to $\mathbf{q}$ a message with label $\mathtt{m}$ and payload $s'[\mathbf{r}]$. With the second reduction, the message is received.

The classic async MPST typing judgement has the following form:

$$\Theta \cdot \Gamma \vdash_{\mathcal{S}} P \tag{11}$$

where $\mathcal{S}$ is the set of sessions whose queue occurs in P (e.g., to type (10) above, we let $\mathcal{S} = \{s\}$). Types are extended to model pending messages; e.g., the processes in (10) are typed by, respectively:

$$\begin{array}{lcl} \Gamma & = & s[\mathbf{p}]{:}\mathbf{q}\oplus\mathtt{m}(S').S,\ s[\mathbf{q}]{:}\mathbf{p}\&\mathtt{m}(S').T,\ s'[\mathbf{r}]{:}S' \\ \Gamma' & = & s[\mathbf{p}]{:}(\mathbf{q}!\mathtt{m}(S')\cdot\epsilon; S),\ s[\mathbf{q}]{:}\mathbf{p}\&\mathtt{m}(S').T,\ s'[\mathbf{r}]{:}S' \\ \Gamma'' & = & s[\mathbf{p}]{:}S,\ s[\mathbf{q}]{:}T,\ s'[\mathbf{r}]{:}S' \end{array} \tag{12}$$

Note that Γ above is a typing context similar to Def. 2.6. Instead, in Γ' the type of $s[\mathtt{p}]$ is a pair $(M;S)$, where $M = \mathtt{q}!\mathtt{m}(S')\cdot\epsilon$ is a *message queue type* (abstracting the pending messages sent through $s[\mathtt{p}]$), followed by the continuation type S. In Γ', the topmost queued message type matches the branching type of $s[\mathtt{q}]$: their interaction leads to Γ'', with a reduction similar to Def. 2.8.

The classic async MPST theory has all the issues described in §3 — but the presence of message queues makes its subject reduction statement more complicated [Coppo et al. 2015a, Lemma 1]:

$$\text{If } \Theta\cdot\Gamma \vdash_{\mathbb{S}} P \ \underline{\text{and } \exists\Gamma_0 \text{ such that } \Gamma,\Gamma_0 \text{ consistent}} \text{ and } P \to P', \\ \text{then } \underline{\exists\Gamma',\Gamma_0' \text{ consistent:}}\ \Gamma,\Gamma_0 \to^* \Gamma',\Gamma_0' \text{ and } \Theta\cdot\Gamma' \vdash_{\mathbb{S}} P'$$

General Asynchronous MPST. We extend our new theory in §4 to asynchronous MPST, and prove a simpler and more general subject reduction statement: Thm. 7.1. To achieve it, we develop async typing rules based on an *async safety property* φ, with a more sophisticated *async typing context reduction* $\to_{\mathbb{S}}$, where $\mathbb{S}$ is a set of sessions, as in (11); e.g., in (12) we have $\Gamma \to_{\{s\}} \Gamma' \to_{\{s\}} \Gamma''$.

Theorem 7.1 (Async Subject Reduction). *Assume $\Theta\cdot\Gamma \vdash_{\mathbb{S}} P$ with Γ asynchronously safe. Then, $P \to P'$ implies $\exists\Gamma'$ asynchronously safe such that $\Gamma \to_{\mathbb{S}}^{*} \Gamma'$ and $\Theta\cdot\Gamma' \vdash_{\mathbb{S}} P'$.*

We define asynchronous variants of φ, similar to those in Fig. 5; and by suitably instantiating φ, we ensure that typed async processes are deadlock-free/live, similarly to Thm. 5.15.

(Un-)Decidability of Type Checking. A result akin to Thm. 4.11 holds for async MPST.

Theorem 7.2. *If φ is decidable, then "$\Theta\cdot\Gamma \vdash_{\mathbb{S}} P$ with φ" is decidable.*

However, under asynchrony we do *not* have a decidability result for φ as general as Thm. 5.13. On the contrary, async safety and most other properties are *undecidable*: the pairing of a session type with a message queue (cf. Γ' in (12)) corresponds to a Communicating Finite-State Machine (CFSM) [Brand and Zafiropulo 1983], and makes typing contexts Turing-powerful [Bartoletti et al. 2016, Thm. 2.5]. Still, we obtain decidable instances of φ through various sound approximations:

(M1) consistency is decidable, and implies asynchronous safety;

(M2) via the session type / CFSM correspondence established in [Deniélou and Yoshida 2013], we show that if Γ is *synchronously* live (Fig.5(5), decidable by Thm.5.13), then Γ is also *asynchronously* live; we extend the result to live$^+$ (Fig.5(6)); and by Lemma 5.9(9), this means that any Γ projected from a global type is asynchronously live$^+$;

(M3) given $n \geq 1$, we can decide if Γ enqueues at most n messages; if so, Γ is finite-state, hence async safety/liveness are decidable. For example, take $\Gamma = s[\mathtt{p}]{:}\mathtt{q}{\oplus}\mathtt{m}_1.\mathtt{q}\&\mathtt{m}_2,\ s[\mathtt{q}]{:}\mathtt{p}{\oplus}\mathtt{m}_2.\mathtt{p}\&\mathtt{m}_1$: it is deadlocked under synchronous semantics, and not projectable from any global type — but under asynchrony, the top-level outputs of $\mathtt{p}$ and $\mathtt{q}$ can be both enqueued, and then received; hence, we can decide that Γ enqueues at most 2 messages, and is asynchronously live.

Remark 7.3. *By instantiating φ in Thm. 7.2 with one of the methods above, we obtain an expressive decidable fragment of our new asynchronous MPST theory:* ***(M1)*** *subsumes classic async MPST;* ***(M2)*** *covers* all *live typing contexts, albeit non-consistent: e.g., it covers all cases in Fig. 4, and all global types (by Lemma 5.9(9));* ***(M3)*** *covers more typing contexts that are not projectable from global types.*

8 CONCLUSION, RELATED AND FUTURE WORK

We have presented a new theory of multiparty sessions types, with novel foundations that do *not* depend on duality/consistency, *nor* global types, *nor* projections. Our new theory subsumes classic MPST, also fixing subject reduction flaws in previous work (Remark 5.16). Moreover, our new type system is modular and reusable: by fine-tuning its parameter φ, we ensure that type-checking is decidable, and that processes are safe, deadlock-free, and live. A summary of the main results:

- **(R1)** our type safety results (Thm. 4.8, Cor. 4.9) are much more general than classic MPST;
- **(R2)** if we instantiate φ with projection/consistency, or any property in Fig. 5, then the type checking judgement "$\Theta \cdot \Gamma \vdash P$ with φ" is decidable. This follows from Thm. 4.11 and Thm. 5.13;
- **(R3)** by suitably choosing φ in **(R2)** above, we can statically guarantee that P "inherits" φ, and has certain desired run-time properties. This is formalised in Thm. 5.15;
- **(R4)** we can implement φ in **(R2)**/**(R3)** above as a syntactic check (Remark 5.12), or as a μ-calculus formula (Fig. 5). In the latter case, we can verify whether Γ satisfies φ via model checking — e.g., using mCRL2, through the paper's companion artifact (`mpstk`). This is shown in § 6;
- **(R5)** our new theory extends to asynchronous communication, as illustrated in § 7.

8.1 Classic Multiparty Session Types (MPST)

The classic MPST framework, and its notions of *global types* and *projections*, were introduced by Honda et al. [2008], with *linearity conditions* to check the well-formedness of global types, and ensure *projectability* of local types. Later, Bettini et al. [2008] proposed a simplified MPST system adopted by most works, including ours.

We now classify some related works w.r.t. their use of projection/consistency:

	papers	projection	consistency	subj. red.	claim
(a)	Bettini et al. [2008]; Carbone et al. [2016, 2015]; Coppo et al. [2015a]; Honda et al. [2008, 2016]	$\leq$ plain	yes	correct	**(C1)**
(b)	Chen [2015]; Deniélou et al. [2012]; Deniélou and Yoshida [2012]; Toninho and Yoshida [2016]	$\geq$ full	no	flawed	**(C2)**
(c)	Scalas et al. [2017a]; Toninho and Yoshida [2017]	full	yes (required)	correct	**(C1)**

Row **(a)** lists works using *plain* (or stricter) global type projection (Def. 3.3), guaranteeing consistency. As shown in § 5.4, our theory captures plain projection / consistency by setting its parameter φ as $\varphi = \text{pproj}_{G,s}$ / $\varphi = \text{consistent}$; however, this excludes many valid protocols, as per claim **(C1)** — e.g., all our examples in Fig. 4.

Row **(b)** lists works using *full* (or more flexible) global type projection, originally introduced in Yoshida et al. [2010] to support more protocols. Such works overlook the consistency requirement; and in § 3, we reveal that classic MPST subject reduction proofs relying on full projection (without consistency) are flawed, as per claim **(C2)**. To "fix" these works within the classic MPST theory, we must require consistency, as done by works in row **(c)** — but this restricts typability, thus falling back into claim **(C1)**. Instead, by Remark 5.12, our new MPST theory supports full projections with $\varphi = \text{fproj}_{G,s}$, thus subsuming classic MPST and fixing flaws, without losing expressiveness.

8.2 Non-Classic Multiparty Session Types

To the best of our knowledge, there are three MPST works (mentioned in Remark 3.1) that are *not* based on classic projection+consistency (Fig. 3) — but have other limitations, that we surmount.

The first work is by Dezani-Ciancaglini et al. [2015] (with a more recent journal version by Ghilezan et al. [2018]): it presents a *single-session* type system, with first-order session types (i.e., without channel-passing); it is rooted on global types and their projections, but does *not* require consistency. The resulting subject reduction proof strategy is quite complex, as it requires to reason on global types and their semantics (see the proof of subject reduction in Ghilezan et al. [2018]). Such a type system is subsumed by letting $\varphi = \text{fproj}_{G,s}$ in our Def. 4.6; in addition, our work also supports higher-order types, multiple interleaved sessions, and protocols for which no global type exists (see Table 1(2), and Ex. 5.11, case Γ_C).

The second non-classic MPST work is by Scalas and Yoshida [2018]: it was our first attempt (and, to the best of our knowledge, the first work in general) to directly address the limitations of consistency (claim **(C1)**), and propose a behavioural theory of MPST, *not* based on global types and projections. Unfortunately, we could not build upon that work, due to its intrinsic limitations:

(1) a major goal of this paper is subsuming and fixing classic MPST (cf. claim **(C2)** in §1, and §3). However, the theory of Scalas and Yoshida [2018] *cannot* achieve this goal: it has different (and more complicated) typing rules that require typing context liveness, and do not support consistency. Our new theory, instead, supports both consistency and liveness, as instances of φ (Lemma 5.9, Remark 5.12);

(2) from Scalas and Yoshida [2018], we reuse the definition of typing context liveness (Fig. 5(5)) — but we show that it is insufficient to guarantee *process* liveness (Def. 5.1, Ex. 5.14). Hence, we develop the stronger properties live^{+}/live^{++} (Fig. 5(6,7)), to obtain the results on run-time process behaviour in Thm. 5.15. Such results are absent in Scalas and Yoshida [2018];

(3) the branching/selection typing rules of Scalas and Yoshida [2018] (Fig. 3) directly inspect typing context reductions. This peculiarity is not problematic under synchronous semantics (where typing contexts have finite-state transition systems), and in some cases, it enables flexible typing judgements that cannot be obtained in classic MPST [Scalas and Yoshida 2018, Ex. 5.5]. The drawback is that, when extended to *a*synchronous semantics, typing contexts become Turing-powerful (§7), and typing rules that inspect their reductions become inherently undecidable; consequently, the theory of Scalas and Yoshida [2018] does not subsume classic works on asynchronous MPST, and cannot achieve this goal without a major overhaul. Instead, our typing rules do *not* inspect typing context reductions, but only check whether the parametric property φ holds: hence, type checking is decidable whenever φ is decidable (Thm. 7.2), and this allows us to subsume classic asynchronous MPST (Remark 7.3).

By instantiating φ = live in Def. 4.6, this paper largely subsumes Scalas and Yoshida [2018]'s work — minus some corner cases based on the inspection of typing context reductions (cf. item *(3)* above).

A third MPST work that can be considered non-classic is Caires and Pérez [2016]: it proposes a theory of multiparty session types encoded in binary sessions, with a type system based on linear logic [Caires and Pfenning 2010; Wadler 2012]. A related multiparty-to-binary session decomposition was later studied by Scalas et al. [2017a] — with a remarkable difference: in Scalas et al. [2017a], consistency is a *necessary* requirement (formalised in their Theorem 6.3), whereas in Caires and Pérez [2016] it is not, although the paper supports full projections and merging. This difference is due to the fact that the decomposition of Caires and Pérez [2016] introduces a centralised *medium process* that receives and forwards all messages between processes playing different roles — whereas the decomposition of Scalas et al. [2017a] maintains the peer-to-peer nature of MPST interactions. This suggests that, when decomposing multiparty choreographies into linear binary interactions, consistency is necessary *if and only if* there is no centralised medium process.[4] The present work supports general multiparty sessions (and binary sessions as a special case) without requiring consistency, nor global types, nor medium processes.

8.3 Binary Sessions Without Duality

Our work yields a generalised theory of *binary* sessions *not* based on classic duality (Def. 3.5), subsuming classic papers based on [Honda et al. 1998]. If we take a binary session typing context $\Gamma = s[\mathbf{p}]{:}S, s[\mathbf{q}]{:}T$, our Lemma 5.9 becomes:

$$\exists G\colon \mathrm{fproj}_{G,s}(\Gamma) \;\not\Longleftarrow\!\!\Longrightarrow\; \mathrm{consistent}(\Gamma) \;\not\Longleftarrow\!\!\Longrightarrow\; \mathrm{live}^{++}(\Gamma) \;\Longleftrightarrow\; \big(\mathrm{safe}(\Gamma) \text{ and } \mathrm{df}(\Gamma)\big) \qquad (13)$$

[4]In an earlier version of this work, we wrongly claimed that Caires and Pérez [2016] has an implicit (but overlooked) consistency assumption, similarly to other works listed in row **(b)** of the table above. This wrong claim is still readable in the conference version of this work [Scalas and Yoshida 2019].

Here, the leftmost "$\Longleftarrow\!\!\!\!\not\;$" is due to supertyping: e.g., if we take the global type $G = \mathbf{p}\!\to\!\mathbf{q}\!:\{\mathtt{m}, \mathtt{m}'\}$, it projects the typing context $\Gamma = s[\mathbf{p}]\!:\!\mathbf{q}\oplus\{\mathtt{m}, \mathtt{m}'\}, s[\mathbf{q}]\!:\!\mathbf{p}\&\{\mathtt{m}, \mathtt{m}'\}$, that is consistent and live^{++} (hence safe); however, if we replace $\mathbf{p}$'s entry with the supertype $\mathbf{p}\oplus\mathtt{m}$, the resulting context is still live^{++} and consistent, but *not* projectable from any global type. The other "$\Longleftarrow\!\!\!\!\not\;$" in (13) is due to *non-tail-recursive types* like $\mu\mathbf{t}.\mathbf{q}\oplus\mathtt{m}(\mathbf{t}).\mathbf{end}$: they have no dual in classic binary session types (since $\mathbf{t}$ is a forbidden payload): thus, they yield non-consistent typing contexts, and processes like P in Ex. 5.11 (case Γ_C) cannot be typed. This limitation has been addressed by several authors, extending duality with various pitfalls (see e.g. [Bernardi and Hennessy 2016, §5.3]): for a survey, and a logic-based solution, see [Lindley and Morris 2016, §3.2]. By not using duality, our theory eschews these issues.

8.4 Type Systems for the π-Calculus

Many type systems have been proposed for the π-calculus, also influencing MPST: see survey in [Hüttel et al. 2016]. Our new MPST theory is a case of *behavioural* type system: it treats *types as simple processes* that reduce and evolve along a typed computation; and since types are simpler than programs, they can be analysed with simpler methods (e.g., finite model checking via our parameter φ, cf. § 6). As stated in § 4, the design of our new MPST theory is inspired by Igarashi and Kobayashi [2004]'s Generic Type System (GTS) for the π-calculus: i.e., we define a type system that is parametric on a property φ, and we prove type safety under the weakest φ; then, we fine-tune φ to statically verify stronger properties of processes, like deadlock-freedom and liveness (§ 5). Besides this general analogy, our treatment is wholly different: we carefully reuse fundamental MPST definitions (§ 2.1) and develop new and more general results (§ 4, § 5) to ensure our new theory fully subsumes the classic one; moreover, for async MPST we devise a new treatment of queue types, obtaining a new, more general subject reduction result (Thm. 7.1).

As an alternative, we might have tried to encode MPST in the GTS, and develop our new results from there. However, this appears unfeasible. Gay et al. [2014] tried the approach for *binary* sessions, obtaining drawbacks in terms of complication and loss of abstraction (see "Assessment" in Gay et al. [2014]): such drawbacks would be greatly amplified for multiparty sessions. Moreover, [Igarashi and Kobayashi 2004, §4.2, §5] study process/type correspondence using a temporal logic without fixed points, with limited support for recursion: their logic would not allow, e.g., to model our variants of liveness (Fig. 5) and address the interplay between liveness, subtyping, and recursion (Ex. 5.14, Thm. 5.15). Further, the encoding approach would not work for async MPST: the types of Igarashi and Kobayashi [2004] lack message queues, and are akin to CCS without restriction, with decidable reachability [He 2011, p. 374]; hence, they cannot encode the Turing-powerful typing contexts of async MPST (§ 7), whose reachability is undecidable.

8.5 Choreographies and Communicating Finite-State Machines (CFSMs)

Various works model and verify multiparty protocols, a.k.a. *choreographies*, via automata-theoretic methods, by representing each party as a CFSM [Brand and Zafiropulo 1983]. The safety their interactions (that is generally undecidable) is verified with two main approaches: *(a)* assume the decidability of a *synchronisability* property [Basu and Bultan 2011, 2016; Basu et al. 2012], and then check temporal properties of CFSMs via model checking; *(b)* check decidable *synchronous* execution conditions on CFSMs, and prove that they ensure safe *asynchronous* executions [Bocchi et al. 2015; Deniélou and Yoshida 2013; Lange et al. 2015]. Both methods can help extending our new MPST theory: since we essentially treat async typing contexts as systems of CFSMs (§ 7), new decidable results on CFSM safety can yield new decidable instances of our type system (Thm. 7.2). Unfortunately, synchronisability has been recently proven *un*decidable by Finkel and Lozes [2017]: i.e., method *(a)* above might be unusable — hence, we adopt method *(b)* (cf. **(M2)** in § 7). Unlike this paper, the above CFSM works do *not* study type systems, nor properties of typed processes.

8.6 Future Work

We kept our typing rules close to classic MPST, to easily combine our results with existing works. E.g., we plan to integrate our work with Coppo et al. [2015b], that studies MPST deadlock-freedom in presence of *multiple* interleaved sessions: our generalised typing rules can be a drop-in replacement for the classic rules used by Coppo et al. [2015b], and this integration would combine their global deadlock-freedom checks, with our improved type safety results for individual sessions. We also plan to extend the calculus (e.g., with polymorphism [Caires and Pérez 2016; Goto et al. 2016]), and expand the properties/formulas studied in Fig. 5 and Thm. 5.15. We will investigate the logical foundations of our new MPST theory, aiming at results that generalise those by Carbone et al. [2016, 2015], which are focused on limited global types, projections, and consistency.

Another interesting research topic is the *completeness* of safety (Def. 4.1), i.e., studying whether the inverse implication w.r.t. Thm. 4.8/Cor. 4.9 holds. This corresponds to the following conjecture:

$$\textit{Take any } \Gamma.\ \ \textit{If } \forall P, P'\colon \Gamma \vdash P \textit{ and } P \longrightarrow^* P' \textit{ implies that } P' \textit{ has no error, then } \mathrm{safe}(\Gamma).$$

We will investigate whether this conjecture holds — and if not, what other completeness results are achievable. Since session subtyping is central for defining safety (via clause [S-$\rightarrow$] in Def. 4.1, and [Γ-Comm] in Def. 2.8), we will leverage Chen et al. [2017]'s work on the completeness of subtyping.

We will also study how to implement our new MPST theory. A basis is the work by Scalas et al. [2017a,b], that embeds classic MPST in Scala, through a linear π-calculus encoding based on consistency; however, since we do *not* require consistency, the work by Scalas et al. [2017a,b] only covers a fragment of our new theory. Using the μ-calculus formulas illustrated in § 6, a new implementation can verify typing context properties by offloading them to a model checker.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful remarks. Thanks to Francisco Ferreira, Julia Gabet, Sung-Shik Jongmans, and Julien Lange for their comments, and to Simon Castellan for testing the companion artifact. Thanks to Mariangiola Dezani and Paola Giannini for a discussion that helped us realise a misunderstanding in the related work (see Footnote 4). This work was partially supported by EPSRC (projects EP/K034413/1, EP/K011715/1, EP/L00058X/1, EP/N027833/1, EP/N028201/1), and by the EU COST Action CA15123 ("EUTypes").

Note: here I am assuming that the calculus is extended with *void* values of type *Void* (a more standard approach would be to have unit values, often represented as $\langle\rangle$, and having type *Unit*)

$$\dfrac{\dfrac{\mathbf{c}\oplus\mathtt{cancel}(\mathit{Void}).\mathbf{end} \leqslant \mathbf{c}\oplus\mathtt{cancel}(\mathit{Void}).\mathbf{end}}{\Gamma_1 \vdash s[\mathbf{s}]{:}\mathbf{c}\oplus\mathtt{cancel}(\mathit{Void}).\mathbf{end}}\;[\text{T-Sub}] \qquad \dfrac{\mathit{Void} \leqslant \mathit{Void}}{\Gamma_2 \vdash \mathit{void}{:}\mathit{Void}}\;[\text{T-Sub}] \qquad \dfrac{\mathrm{end}(\Gamma, s[\mathbf{s}]{:}\mathbf{end})}{\Theta \cdot \Gamma, s[\mathbf{s}]{:}\mathbf{end} \vdash \mathbf{0}}\;[\text{T-}\mathbf{0}]}{\Theta \cdot \Gamma, \Gamma_1, \Gamma_2 \vdash s[\mathbf{s}][\mathbf{c}]\oplus\mathtt{cancel}\langle \mathit{void}\rangle.\mathbf{0}}\;[\text{T-}\oplus]$$

Where:

- $\Gamma = \emptyset$. Note that Γ is the part of the typing context that is used to type the continuation of the internal choice (in this example, $\mathbf{0}$).
- $\Gamma_1 = s[\mathbf{s}]{:}\mathbf{c}\oplus\mathtt{cancel}(\mathit{Void}).\mathbf{end}$. Note that Γ_1 is the part of the typing context that types the channel with role used in the internal choice (in this example, $s[\mathbf{s}]$).
- $\Gamma_2 = \mathit{void}{:}\mathit{Void}$. Note that Γ_2 is the part of the typing context that types the payload of the message being sent (in this example, *void*).

REFERENCES

Davide Ancona, Viviana Bono, Mario Bravetti, Joana Campos, Giuseppe Castagna, Pierre-Malo Deniélou, Simon J. Gay, Nils Gesbert, Elena Giachino, Raymond Hu, Einar Broch Johnsen, Francisco Martins, Viviana Mascardi, Fabrizio Montesi, Rumyana Neykova, Nicholas Ng, Luca Padovani, Vasco T. Vasconcelos, and Nobuko Yoshida. 2017. Behavioral Types in Programming Languages. *Foundations and Trends in Programming Languages* 3(2-3) (2017). https://doi.org/10.1561/2500000031

Massimo Bartoletti, Alceste Scalas, Emilio Tuosto, and Roberto Zunino. 2016. Honesty by Typing. *LMCS* 12(4) (2016). https://doi.org/10.2168/LMCS-12(4:7)2016

Samik Basu and Tevfik Bultan. 2011. Choreography conformance via synchronizability. In *WWW*.

Samik Basu and Tevfik Bultan. 2016. On deciding synchronizability for asynchronously communicating systems. *Theor. Comput. Sci.* 656 (2016).

Samik Basu, Tevfik Bultan, and Meriem Ouederni. 2012. Synchronizability for Verification of Asynchronously Communicating Systems. In *VMCAI*.

Giovanni Bernardi and Matthew Hennessy. 2016. Using higher-order contracts to model session types. *LMCS* 12(2) (2016). https://doi.org/10.2168/LMCS-12(2:10)2016

Lorenzo Bettini, Mario Coppo, Loris D'Antoni, Marco De Luca, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. 2008. Global Progress in Dynamically Interleaved Multiparty Sessions. In *CONCUR*. https://doi.org/10.1007/978-3-540-85361-9_33

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

Daniel Brand and Pitro Zafiropulo. 1983. On Communicating Finite-State Machines. *JACM* 30, 2 (1983). https://doi.org/10.1145/322374.322380

Nadia Busi, Maurizio Gabbrielli, and Gianluigi Zavattaro. 2009. On the expressive power of recursion, replication and iteration in process calculi. *Mathematical Structures in Computer Science* 19, 6 (2009). https://doi.org/10.1017/S096012950999017X

Luís Caires and Jorge A. Pérez. 2016. Multiparty Session Types Within a Canonical Binary Theory, and Beyond. In *FORTE*. https://doi.org/10.1007/978-3-319-39570-8_6

Luís Caires and Frank Pfenning. 2010. Session Types as Intuitionistic Linear Propositions. In *CONCUR*. https://doi.org/10.1007/978-3-642-15375-4_16

Luís Caires, Frank Pfenning, and Bernardo Toninho. 2016. Linear logic propositions as session types. *MSCS* 26, 3 (2016).

Marco Carbone, Sam Lindley, Fabrizio Montesi, Carsten Schürmann, and Philip Wadler. 2016. Coherence Generalises Duality: A Logical Explanation of Multiparty Session Types. In *CONCUR*. https://doi.org/10.4230/LIPIcs.CONCUR.2016.33

Marco Carbone, Fabrizio Montesi, Carsten Schürmann, and Nobuko Yoshida. 2015. Multiparty Session Types as Coherence Proofs. In *CONCUR*. https://doi.org/10.4230/LIPIcs.CONCUR.2015.412

Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, Alceste Scalas, and Nobuko Yoshida. 2017. On the Preciseness of Subtyping in Session Types. *Logical Methods in Computer Science* 13, 2 (2017). https://doi.org/10.23638/LMCS-13(2:12)2017

Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. 2014. On the Preciseness of Subtyping in Session Types. In *PPDP*. https://doi.org/10.1145/2643135.2643138

Tzu-Chun Chen. 2015. Lightening global types. *JLAMP* 84, 5 (2015). https://doi.org/10.1016/j.jlamp.2015.06.003

Mario Coppo, Mariangiola Dezani-Ciancaglini, Luca Padovani, and Nobuko Yoshida. 2015a. A Gentle Introduction to Multiparty Asynchronous Session Types. In *Formal Methods for Multicore Programming*. https://doi.org/10.1007/978-3-319-18941-3_4

Mario Coppo, Mariangiola Dezani-Ciancaglini, Nobuko Yoshida, and Luca Padovani. 2015b. Global Progress for Dynamically Interleaved Multiparty Sessions. *MSCS* 760 (2015). https://doi.org/10.1017/S0960129514000188

Pierre-Malo Deniélou, Nobuko Yoshida, Andi Bejleri, and Raymond Hu. 2012. Parameterised Multiparty Session Types. *LMCS* 8, 4 (2012). https://doi.org/10.2168/LMCS-8(4:6)2012

Pierre-Malo Deniélou and Nobuko Yoshida. 2012. Multiparty Session Types Meet Communicating Automata. In *ESOP*. https://doi.org/10.1007/978-3-642-28869-2_10

Pierre-Malo Deniélou and Nobuko Yoshida. 2013. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. In *ICALP*. https://doi.org/10.1007/978-3-642-39212-2_18

Mariangiola Dezani-Ciancaglini, Silvia Ghilezan, Svetlana Jaksic, Jovanka Pantovic, and Nobuko Yoshida. 2015. Precise subtyping for synchronous multiparty sessions. In *PLACES*. https://doi.org/10.4204/EPTCS.203.3 Journal version: [Ghilezan et al. 2018].

Alain Finkel and Etienne Lozes. 2017. Synchronizability of Communicating Finite State Machines is not Decidable. In *ICALP*. https://doi.org/10.4230/LIPIcs.ICALP.2017.122

Simon Gay and António Ravara. 2017. *Behavioural Types: From Theory to Tools*. River Publishers, Series in Automation, Control and Robotics. https://doi.org/10.13052/rp-9788793519817

Simon J. Gay. 2016. Subtyping Supports Safe Session Substitution. In *A List of Successes That Can Change the World: Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday (LNCS)*, Vol. 9600. https://doi.org/10.1007/978-3-319-30936-1_5

Simon J. Gay, Nils Gesbert, and António Ravara. 2014. Session Types as Generic Process Types. In *EXPRESS/SOS*. https://doi.org/10.4204/EPTCS.160.9

Simon J. Gay and Malcolm Hole. 2005. Subtyping for session types in the π-calculus. *Acta Inf.* 42, 2-3 (2005). https://doi.org/10.1007/s00236-005-0177-z

Silvia Ghilezan, Svetlana Jakšić, Jovanka Pantović, Alceste Scalas, and Nobuko Yoshida. 2018. Precise subtyping for synchronous multiparty sessions. *Journal of Logical and Algebraic Methods in Programming* (2018). https://doi.org/10.1016/j.jlamp.2018.12.002

Jean-Yves Girard. 1987. Linear Logic. *TCS* 50 (1987), 1–102.

Matthew Goto, Radha Jagadeesan, Alan Jeffrey, Corin Pitcher, and James Riely. 2016. An extensible approach to session polymorphism. *Mathematical Structures in Computer Science* 26, 3 (2016). https://doi.org/10.1017/S0960129514000231

Jan Friso Groote and Mohammad Reza Mousavi. 2014. *Modeling and Analysis of Communicating Systems*. The MIT Press.

Chaodong He. 2011. The Decidability of the Reachability Problem for CCS!. In *CONCUR*. https://doi.org/10.1007/978-3-642-23217-6_25

Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. 1998. Language Primitives and Type Discipline for Structured Communication-Based Programming. In *ESOP*. https://doi.org/10.1007/BFb0053567

Kohei Honda, Nobuko Yoshida, and Marco Carbone. 2008. Multiparty asynchronous session types. In *POPL*. https://doi.org/10.1145/1328438.1328472 Full version in [Honda et al. 2016].

Kohei Honda, Nobuko Yoshida, and Marco Carbone. 2016. Multiparty Asynchronous Session Types. *J. ACM* 63, 1, Article 9 (2016). https://doi.org/10.1145/2827695

Hans Hüttel, Ivan Lanese, Vasco T. Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniélou, Dimitris Mostrous, Luca Padovani, António Ravara, Emilio Tuosto, Hugo Torres Vieira, and Gianluigi Zavattaro. 2016. Foundations of Session Types and Behavioural Contracts. *ACM Comput. Surv.* 49, 1, Article 3 (2016). https://doi.org/10.1145/2873052

Atsushi Igarashi and Naoki Kobayashi. 2004. A generic type system for the π-calculus. *TCS* 311, 1 (2004). https://doi.org/10.1016/S0304-3975(03)00325-6

Naoki Kobayashi and Davide Sangiorgi. 2010. A hybrid type system for lock-freedom of mobile processes. *TOPLAS* 32, 5 (2010). https://doi.org/10.1145/1745312.1745313

Julien Lange, Emilio Tuosto, and Nobuko Yoshida. 2015. From Communicating Machines to Graphical Choreographies. In *POPL*. https://doi.org/10.1145/2676726.2676964

Sam Lindley and J. Garrett Morris. 2016. Talking Bananas: Structural Recursion for Session Types. In *ICFP*. https://doi.org/10.1145/2951913.2951921

Barbara H. Liskov and Jeannette M. Wing. 1994. A Behavioral Notion of Subtyping. *TOPLAS* 16, 6 (1994). https://doi.org/10.1145/197320.197383

OAuth Working Group. 2012. RFC 6749: OAuth 2.0 Framework. http://tools.ietf.org/html/rfc6749.

Luca Padovani. 2014. Deadlock and lock freedom in the linear π-calculus. In *CSL-LICS*. https://doi.org/10.1145/2603088.2603116

Luca Padovani. 2016. Fair Subtyping for Multi-Party Session Types. *Mathematical Structures in Computer Science* 26, 3 (2016). https://doi.org/10.1017/S096012951400022X

Benjamin C. Pierce. 2002. *Types and programming languages*. MIT Press.

Alceste Scalas, Ornela Dardha, Raymond Hu, and Nobuko Yoshida. 2017a. A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming. In *ECOOP*. https://doi.org/10.4230/LIPIcs.ECOOP.2017.24

Alceste Scalas, Ornela Dardha, Raymond Hu, and Nobuko Yoshida. 2017b. A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming (Artifact). *Dagstuhl Artifacts Series* 3, 1 (2017). https://doi.org/10.4230/DARTS.3.2.3

Alceste Scalas and Nobuko Yoshida. 2018. Multiparty session types, beyond duality. *Journal of Logical and Algebraic Methods in Programming* 97. https://doi.org/10.1016/j.jlamp.2018.01.001

Alceste Scalas and Nobuko Yoshida. 2019. Less is More: Multiparty Session Types Revisited. *Proc. ACM Program. Lang.* 3, POPL, Article 30 (Jan. 2019), 29 pages. https://doi.org/10.1145/3290343

Bernardo Toninho and Nobuko Yoshida. 2016. Certifying Data in Multiparty Session Types. In *A List of Successes That Can Change the World: Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday (LNCS)*, Vol. 9600. https://doi.org/10.1007/978-3-319-30936-1_23

Bernardo Toninho and Nobuko Yoshida. 2017. Certifying data in multiparty session types. *JLAMP* 90 (2017). https://doi.org/10.1016/j.jlamp.2016.11.005

Philip Wadler. 2012. Propositions as sessions. In *ICFP*. https://doi.org/10.1145/2364527.2364568

Philip Wadler. 2014. Propositions as sessions. *J. Funct. Program.* 24, 2-3 (2014).

Nobuko Yoshida, Pierre-Malo Deniélou, Andi Bejleri, and Raymond Hu. 2010. Parameterised Multiparty Session Types. In *FOSSACS*. https://doi.org/10.1007/978-3-642-12032-9_10

Appendices — Part 1

Additional definitions, and asynchronous MPST

	Synchronous	**Asynchronous (§7)**
Multiparty session π-calculus	**§2.1**	**§C**
Multiparty session types	**§2.2**	**§D**
Issues of classic MPST	**§3**	**§E**
A new, general MPST theory	**§4**	**§F**
• Typing context safety invariant	Def. 4.1	Def. F.2
• Subject reduction	Thm. 4.8	Thm. 7.1 / Thm. F.6
• Decidability of type checking	Thm. 4.11	Thm. 7.2
Verifying process behaviours using types	**§5**	**§G**
• Session fidelity	Thm. 5.4	Thm. F.8
• Typing context properties	Fig. 5, Def. 5.8	Def. G.2
• Static verification of process properties	Thm. 5.15	Thm. G.9
• (Un-)Decidability of typing ctx. properties	Thm. 5.13	Thm. G.5, Thm. G.6
Model checking of type-level properties	**§6**	**Remark G.10**
Related and Future Work	**§8 and §H**	

Table 3. Contents and contributions of this paper. The contents and main contributions in the "Asynchronous" column are summarised in §7. Proofs are available in the second part of the appendices (§I–§N).

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

$$(\nu s)\,(\nu s')\, P \equiv (\nu s')\,(\nu s)\, P \qquad (\nu s)\,(P \mid Q) \equiv P \mid (\nu s)\, Q \quad \text{if } s \notin \mathrm{fc}(P)$$

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

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

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

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

Fig. 6. MPST π-calculus: standard structural congruence (top), and up-to-congruence reduction rule (bottom), which completes Fig. 1. In the rules above, $\mathrm{fpv}(D)$ is the set of *free process variables* in D, and $\mathrm{dpv}(D)$ is the set of *declared process variables* in D.

A MULTIPARTY SESSION π-CALCULUS

The standard congruence delation of the MPST π-calculus, mentioned in Fig. 1, is formalised in Fig. 6. The **up-to congruence reduction rule** [R-≡], which was omitted in Fig. 1, says that reduction is closed under $\equiv$.

B SESSION FIDELITY

This section discusses some intermediate results leading to Thm. 5.4 (session fidelity). For the full proof details, see §I.

The MPST typing system enjoys the fundamental Lemmas B.1 to B.3 below: they hold in most typing systems, and will be necessary for session fidelity (§5.2).

Lemma B.1 (Substitution). *Assume* $\Theta \cdot \Gamma, x{:}S \vdash P$ *and* $\Gamma' \vdash s[\mathbf{p}]{:}S$*, with* Γ, Γ' *defined. Then,* $\Theta \cdot \Gamma, \Gamma' \vdash P\{s[\mathbf{p}]/x\}$.

Proof. Minor adaptation of [Coppo et al. 2015a, Lemma 5]. □

Lemma B.2 (Subject Congruence). *Assume* $\Theta \cdot \Gamma \vdash P$ *and* $P \equiv P'$*. Then,* $\Theta \cdot \Gamma \vdash P'$.

Proof. By examining the cases where $P \equiv P'$ holds, and by inversion of the typing judgements $\Theta \cdot \Gamma \vdash P$ and $\Theta \cdot \Gamma \vdash P'$ (Lemma I.4). □

Lemma B.3 (Narrowing). *If* $\Theta \cdot \Gamma \vdash P$ *and* $\Gamma' \leqslant \Gamma$*, then* $\Theta \cdot \Gamma' \vdash P$.

With Def. 5.3 and Lemmas B.1 and B.2, we can show how typing contexts determine process shapes. Thm. B.4 below addresses the typical application scenario of MPST, i.e., an ensemble of programs $P_{\mathbf{p}}$ that interact on a multiparty session s, each one playing a distinct role $\mathbf{p}$. The crucial parts are items (1) and (2): if the type of the channel used by $P_{\mathbf{p}}$ requires to select (resp. branch), then $P_{\mathbf{p}}$ will be ready to perform the corresponding operation, possibly after calling some X. Note that this crucially relies on item 1 of Def. 5.3. In fact, by rule [T-def] (Fig. 2), an unguarded definition like $X(x{:}S) = X\langle x\rangle$ can be typed with *any* S, even when S is an internal/external choice requiring to use x for selection/branching. This possibility would refute items (1)(b) and (2)(b) of Thm. B.4 — but item 1 of Def. 5.3 solves the issue: it forces unused process parameters to be **end**-typed.

Theorem B.4 (Session Inversion). *Assume* $\emptyset \cdot \Gamma \vdash \big|_{\mathbf{p} \in I} P_{\mathbf{p}}$ *with each* $P_{\mathbf{p}}$ *either being* $\mathbf{0}$ *(up-to* $\equiv$*), or only playing role* $\mathbf{p}$ *in* s*. Then,* $\Gamma = \Gamma_0, \left\{s[\mathbf{p}]{:}S_{\mathbf{p}}\right\}_{\mathbf{p} \in I'}$ *(for some* I'*) with* $\mathrm{end}(\Gamma_0)$*. Moreover,* $\forall \mathbf{p} \in I'$:

(1) *if* $\mathtt{q}\oplus_{j\in J}\mathtt{m}_j(S'_j).S'_j \leqslant S_\mathtt{p}$ *then* $\mathtt{p}\in I$ *and for some* $\mathbb{C}$, $\mathbb{C}'$, *and* $k\in J$, *either:*

 (a) $P_\mathtt{p} \equiv \mathbb{C}\left[s[\mathtt{p}][\mathtt{q}]\oplus\mathtt{m}_k\langle s'[\mathtt{r}]\rangle.P'_\mathtt{p}\right]$ *or*

 (b) $P_\mathtt{p} \equiv \mathbb{C}\left[\begin{array}{l}\mathbf{def}\ X(x_1{:}T_1,\ldots,x_n{:}T_n) = \mathbb{C}'\left[x_l[\mathtt{q}]\oplus\mathtt{m}_k\langle d\rangle.P'_\mathtt{p}\right]\ \mathbf{in}\\ \quad X\left\langle s'_1[\mathtt{r}_1],\ldots,s'_{l-1}[\mathtt{r}_{l-1}],\ s[\mathtt{p}],\ s'_{l+1}[\mathtt{r}_{l+1}],\ldots,s'_n[\mathtt{r}_n]\right\rangle\end{array}\right]$ *with* $1\leq l\leq n$;

(2) *if* $\mathtt{q}\&_{j\in J}\mathtt{m}_j(S'_j).S'_j \leqslant S_\mathtt{p}$ *then* $\mathtt{p}\in I$ *and for some* $\mathbb{C}$, $\mathbb{C}'$, *and* $K\supseteq J$, *either:*

 (a) $P_\mathtt{p} \equiv \mathbb{C}\left[s[\mathtt{p}][\mathtt{q}]\sum_{k\in K}\mathtt{m}_k(x_k).P'_{\mathtt{p}k}\right]$ *or*

 (b) $P_\mathtt{p} \equiv \mathbb{C}\left[\begin{array}{l}\mathbf{def}\ X(x_1{:}T_1,\ldots,x_n{:}T_n) = \mathbb{C}'\left[x_l[\mathtt{q}]\sum_{k\in K}\mathtt{m}_k(x_k).P'_{\mathtt{p}k}\right]\ \mathbf{in}\\ \quad X\left\langle s'_1[\mathtt{r}_1],\ldots,s'_{l-1}[\mathtt{r}_{l-1}],\ s[\mathtt{p}],\ s'_{l+1}[\mathtt{r}_{l+1}],\ldots,s'_n[\mathtt{r}_n]\right\rangle\end{array}\right]$ *with* $1\leq l\leq n$;

(3) *if* $\mathbf{end} \leqslant S_\mathtt{p}$ *then* $\mathtt{p}\in I$ *implies* $P_\mathtt{p}\equiv\mathbf{0}$.

Further, (4) $\forall\mathtt{p}\in I\setminus I' : P_\mathtt{p}\equiv\mathbf{0}$.

C ASYNCHRONOUS MULTIPARTY SESSION π-CALCULUS

We now address *asynchronous MPST*, as in the original MPST papers [Bettini et al. 2008; Honda et al. 2008] and in most successive works. Async MPST provide a more faithful model of real-world distributed applications, that usually employ *buffered* message-passing (e.g., via the TCP protocol); moreover, we will see that our new async MPST theory (unlike the classic one) can handle protocols whose correctness actually depends on message buffering (Ex. G.4(4),(5)). However, asynchrony will require us to address several challenges:

(1) the classic async MPST theory has additional complications (§E);
(2) to eschew such complications, and successfully extend our approach to asynchrony, we will develop a new proof strategy for subject reduction (§F);
(3) async typing context properties are generally *undecidable*; still, we will achieve decidable type checking by leveraging results from communicating automata (§G).

In this section, we start developing the asynchronous theory by adding *non-blocking send operations* and *message queues* to the π-calculus of §2.1. From now on, we overload the notation of §2.1, and focus on the differences between synchronous/asynchronous definitions and results.

Definition C.1. **Async MPST processes** have the syntax in Def. 2.1, plus *session queues*:

$$P,Q \ ::=\ \ldots\ \big|\ s\blacktriangleright\sigma \quad \text{where } \sigma \text{ is a } \textit{message queue}\text{:} \quad \sigma\ ::=\ (\mathtt{p},\mathtt{q},\mathtt{m}\langle s[\mathtt{r}]\rangle)\cdot\sigma\ \big|\ \epsilon$$

We require that in well-formed processes, *each session has a queue*: if $P = (\nu s)\,Q$, then $Q \equiv \left(\nu\widetilde{s'}\right)(Q' \mid s\blacktriangleright\sigma)$.

The **semantics of async processes** is induced by the rules in Fig. 1, *but (1)* we replace [R-Comm] and [R-Err] by [R-AOut], [R-AIn] and [R-AErr] in Fig. 7, and *(2)* the congruence $\equiv$ is extended with the rules in Fig. 7. The set of **message senders in** σ, or senders(σ), is:

$$\text{senders}((\mathtt{p},\mathtt{q},\mathtt{m}\langle s'[\mathtt{r}]\rangle)\cdot\sigma') \ = \ \{\mathtt{p}\}\cup\text{senders}(\sigma') \qquad\qquad \text{senders}(\epsilon) \ = \ \emptyset$$

In Fig. 7, the **output rule** [R-AOut] enqueues a *pending message* — i.e., a triple with the message sender role, the intended recipient, and the message itself. The **input rule** [R-AIn] dequeues a pending message if its sender, recipient, and label match the receiving process. The **error rule** [R-AErr] fires if a process with role $\mathtt{p}$ is waiting for a message $\mathtt{m}_i$ $(i\in I)$ from $\mathtt{q}$, but the queue head is an unsupported message. The semantics is defined up-to the *congruence* $\equiv$ in Fig. 6, *plus* the **rules for queues** in Fig. 7: the first is for garbage collection; the second reorders messages with different sender/recipient. E.g.:

$$s[\mathtt{p}][\mathtt{q}]\sum\mathtt{m}_2(x).P \mid s\blacktriangleright(\mathtt{r},\mathtt{p},\mathtt{m}_1\langle s_1[\mathtt{r}_1]\rangle)\cdot(\mathtt{q},\mathtt{p},\mathtt{m}_2\langle s_2[\mathtt{r}_2]\rangle)\cdot\epsilon \ \rightarrow\ P\{s_2[\mathtt{r}_2]/x\} \mid (\mathtt{r},\mathtt{p},\mathtt{m}_1\langle s_1[\mathtt{r}_1]\rangle)\cdot\epsilon$$

$$\begin{array}{lll}
\text{[R-AOut]} & s[\mathsf{q}][\mathsf{p}]\oplus \mathtt{m}\langle s'[\mathsf{r}]\rangle.Q \mid s \blacktriangleright \sigma \;\to\; Q \mid s \blacktriangleright \sigma \cdot (\mathsf{q},\mathsf{p},\mathtt{m}\langle s'[\mathsf{r}]\rangle)\cdot\epsilon & \\
\text{[R-AIn]} & s[\mathsf{p}][\mathsf{q}]\sum_{i\in I}\mathtt{m}_i(x_i).P_i \mid s \blacktriangleright (\mathsf{q},\mathsf{p},\mathtt{m}_k\langle s'[\mathsf{r}]\rangle)\cdot\sigma \;\to\; P_k\{s'[\mathsf{r}]/x_k\} \mid s \blacktriangleright \sigma & \text{if } k \in I \\
\text{[R-AErr]} & s[\mathsf{p}][\mathsf{q}]\sum_{i\in I}\mathtt{m}_i(x_i).P_i \mid s \blacktriangleright (\mathsf{q},\mathsf{p},\mathtt{m}\langle s'[\mathsf{r}]\rangle)\cdot\sigma \;\to\; \mathbf{err} & \text{if } \forall i \in I : \mathtt{m}_i \neq \mathtt{m}
\end{array}$$

$$(\nu s)\, s \blacktriangleright \epsilon \equiv \mathbf{0} \qquad \begin{array}{l} s \blacktriangleright \sigma \cdot (\mathsf{p}_1,\mathsf{q}_1,\mathtt{m}_1\langle s_1[\mathsf{r}_1]\rangle)\cdot(\mathsf{p}_2,\mathsf{q}_2,\mathtt{m}_2\langle s_2[\mathsf{r}_2]\rangle)\cdot\sigma' \\ \equiv\; s \blacktriangleright \sigma \cdot (\mathsf{p}_2,\mathsf{q}_2,\mathtt{m}_2\langle s_2[\mathsf{r}_2]\rangle)\cdot(\mathsf{p}_1,\mathsf{q}_1,\mathtt{m}_1\langle s_1[\mathsf{r}_1]\rangle)\cdot\sigma' \end{array} \quad \text{if } \mathsf{p}_1 \neq \mathsf{p}_2 \text{ or } \mathsf{q}_1 \neq \mathsf{q}_2$$

Fig. 7. Async MPST π-calculus: semantics (top) and congruence for queues (bottom).

i.e., the "swapping congruence" $\equiv$ moves the message $\mathtt{m}_2$ to the head of the queue, allowing to fire [R-AIn]. Hence, the session queue behaves as a *set* of unidirectional FIFO buffers, delivering messages between each pair of roles, akin to the TCP protocol.

D ASYNCHRONOUS MULTIPARTY SESSION TYPES

We now extend the type system of § 2 to the asynchronous calculus of § C. We reuse and overload definitions and notation from § 2.2. We will prove subject reduction and session fidelity results with our new async type system (§ F).

To type queues in the asynchronous calculus, we need *queue types* (Def. D.1).

Definition D.1. The **queue and session/queue types** are: (with S from Def. 2.4)

$$\text{(Queue types)}\;\; M \;::=\; \mathsf{p}!\mathtt{m}(S)\cdot M \;\big|\; \epsilon \qquad \text{(Session/queue types)}\;\; \tau \;::=\; S \;\big|\; M \;\big|\; (M;S)$$

The **congruence relation $\equiv$ for session/queue types τ** is inductively defined as:

$$\frac{}{S \equiv S} \qquad \frac{\mathsf{p} \neq \mathsf{q}}{\mathsf{p}!\mathtt{m}_1(S_1)\cdot\mathsf{q}!\mathtt{m}_2(S_2)\cdot M \;\equiv\; \mathsf{q}!\mathtt{m}_2(S_2)\cdot\mathsf{p}!\mathtt{m}_1(S_1)\cdot M} \qquad \frac{M \equiv M' \quad S \equiv S'}{(M;S) \equiv (M';S')}$$

Subtyping for session/queue types extends Def. 2.5 as: $\dfrac{}{M \leqslant M} \quad \dfrac{M \leqslant M' \quad S \leqslant S'}{(M;S) \leqslant (M';S')}$

Queue types are sequences of **message types** $\mathsf{p}!\mathtt{m}(S)$ having recipient p, label $\mathtt{m}$, and payload type S (omitted when $S = \mathbf{end}$). A **session/queue type** can be a session type, a queue type, or a session/queue types pair: the latter describes queued messages (sent "in the past," not yet received) and channel usage (that a process will fulfil "in the future"). The congruence $\equiv$ reorders queued messages with different recipients, like $\equiv$ in Fig. 7.

Definition D.2. An **async MPST typing context** is a partial mapping defined as:

$$\Gamma \;::=\; \Gamma, s[\mathsf{p}]{:}\tau \;\big|\; \Gamma, x{:}S \;\big|\; \emptyset$$

The *composition* Γ_1, Γ_2 is defined iff $\forall c \in \text{dom}(\Gamma_1) \cap \text{dom}(\Gamma_2) : \Gamma_i(c) = M$ and $\Gamma_j(c) = S$ and for all such c, we postulate $(\Gamma_1, \Gamma_2)(c) = (M;S)$. We extend $\equiv$ (from Def. D.1) to typing contexts as: $\Gamma \equiv \Gamma'$ iff $\text{dom}(\Gamma) = \text{dom}(\Gamma')$ and $\forall c \in \text{dom}(\Gamma) : \Gamma(c) \equiv \Gamma'(c)$.
The relation $\Gamma \leqslant \Gamma'$ follows Def. 2.6, but using session/queue subtyping from Def. D.1.

Unlike Def. 2.6, in Def. D.2 above we have that: *(1)* channels with role map to session/queue types (but variables x still map to session types, only); and *(2)* the **context composition** Γ_1, Γ_2 allows the domains of Γ_1 and Γ_2 to overlap on some c — but only if c maps to a session type in one context, and a queue type in the other. This can only occur if $c = s[\mathsf{p}]$ (for some s, p), i.e., c is *not* a variable x; then, $(\Gamma_1, \Gamma_2)(c)$ yields the combined session/queue type. The async typing rules in Fig. 8 induce the judgement:

$$\Theta \cdot \Gamma \vdash_{\mathbb{S}} P \qquad \text{where } \mathbb{S} \text{ is a set of sessions } \{s_1, \ldots, s_n\}, \text{ omitted when empty} \tag{14}$$

$$\frac{\Theta \cdot \Gamma \vdash P \;\; \text{(from Fig. 2, replacing rules [T-|] / [T-}\nu\text{] with [TA-|] / [TA-}\nu\text{])}}{\Theta \cdot \Gamma \vdash_{\emptyset} P}\;\text{[TA-Lift]}$$

$$\frac{\Theta \cdot \Gamma_1 \vdash_{\mathcal{S}_1} P_1 \qquad \Theta \cdot \Gamma_2 \vdash_{\mathcal{S}_2} P_2 \qquad \mathcal{S}_1 \cap \mathcal{S}_2 = \emptyset}{\Theta \cdot \Gamma_1, \Gamma_2 \vdash_{\mathcal{S}_1 \cup \mathcal{S}_2} P_1 \mid P_2}\;\text{[TA-|]}$$

$$\frac{\Gamma' = \left\{s[\mathtt{p}]{:}S_{\mathtt{p}}\right\}_{\mathtt{p}\in I} \quad s \notin \Gamma \quad \varphi(\Gamma') \quad \Theta \cdot \Gamma, \Gamma' \vdash_{\mathcal{S}} P}{\Theta \cdot \Gamma \vdash_{\mathcal{S}\setminus s} (\nu s{:}\Gamma')\, P}\;\text{[TA-}\nu\text{]} \qquad \text{where } \varphi \text{ is a typing context property}$$

- -

$$\frac{}{\Theta \cdot \emptyset \vdash_{\{s\}} s \blacktriangleright \epsilon}\;\text{[TA-}\epsilon\text{]} \qquad \frac{\Theta \cdot \Gamma \vdash_{\{s\}} s \blacktriangleright \sigma \qquad \Gamma' \vdash s'[\mathtt{r}]{:}S}{\Theta \cdot (\Gamma \leftsquigarrow s[\mathtt{p}]{:}\mathtt{q}!\mathtt{m}(S)\cdot\epsilon), \Gamma' \vdash_{\{s\}} s \blacktriangleright (\mathtt{p}, \mathtt{q}, \mathtt{m}\langle s'[\mathtt{r}]\rangle)\cdot\sigma}\;\text{[TA-}\sigma\text{]}$$

$$\text{where } \; \Gamma \leftsquigarrow s[\mathtt{p}]{:}M = \begin{cases} \Gamma\left\{{}^{M\cdot\Gamma(s[\mathtt{p}])}/_{s[\mathtt{p}]}\right\} & \text{if } s[\mathtt{p}] \in \operatorname{dom}(\Gamma) \\ \Gamma, s[\mathtt{p}]{:}M & \text{otherwise} \end{cases}$$

Fig. 8. Asynchronous MPST typing rules: processes (top) and queues (bottom).

Unlike the sync MPST judgement (5), the context Γ of (14) is asynchronous (Def. D.2); further, (14) includes a **set of sessions** $\mathcal{S}$ to track P's queues; e.g., the **parallel rule** [TA-|] types parallel processes by combining their contexts, and requiring their session queues not to overlap ($\mathcal{S}_1 \cap \mathcal{S}_2 = \emptyset$): this rejects processes with multiple queues per session, like $Q \mid s \blacktriangleright \sigma \mid s \blacktriangleright \sigma'$. We will use $\mathcal{S}$ in a more sophisticated way later, in §F. The **lifting rule** [TA-Lift] types queueless processes, by lifting the synchronous typing judgement in Fig. 2. The **session restriction rule** [TA-ν] is akin to [T-ν] (Fig. 2), but also removes the restricted s from the set of sessions; the typing context property φ is defined depending on the underlying proof strategy for subject reduction, with considerations similar to those highlighted in §2.3: we discuss the classic async MPST approach (and its issues) in §E, and our novel approach in §F.

The remaining rules are for typing queues. We have **two queue rules**: [TA-ϵ] types an empty queue $s \blacktriangleright \epsilon$ with the empty context; [TA-σ] types a non-empty queue by inserting a message type in Γ using $\leftsquigarrow$, that might *(a)* prepend the message to a queue type in Γ, or *(b)* add a queue-typed entry to Γ, if not present.

Example D.3. The queue typing rules produce judgements like the following:

$$\Theta \cdot \Gamma, \begin{array}{l} s[\mathtt{p}]{:}\mathtt{q}\&\mathtt{m}_2(S_2).S', \\ s[\mathtt{r}]{:}\mathtt{p}!\mathtt{m}_1(S_1)\cdot\epsilon, \\ s[\mathtt{q}]{:}\mathtt{p}!\mathtt{m}_2(S_2)\cdot\epsilon \end{array} \quad \vdash_{\{s\}} \; s[\mathtt{p}][\mathtt{q}]\textstyle\sum \mathtt{m}_2(x).P \mid s \blacktriangleright (\mathtt{r}, \mathtt{p}, \mathtt{m}_1\langle s_1[\mathtt{r}_1]\rangle)\cdot(\mathtt{q}, \mathtt{p}, \mathtt{m}_2\langle s_2[\mathtt{r}_2]\rangle)\cdot\epsilon$$

Note that $s[\mathtt{p}]$ has a session type (matching the process), while queued messages are typed by assigning them to their *sender* role, thus giving queue types to $s[\mathtt{r}]$ and $s[\mathtt{q}]$.

Async contexts reduce by Def. D.4 below: the definition is standard, except for the addition of transition labels. Unlike Def. 2.8, types interact in two phases: first, messages are queued ([Γ-AMsg]); then, they are consumed ([Γ-AComm]).

Definition D.4. Let α have the form $s{:}\mathtt{p}!\mathtt{q}{:}\mathtt{m}$ or $s{:}\mathtt{p},\mathtt{q}{:}\mathtt{m}$. The **async typing context transition** $\xrightarrow{\alpha}$ is inductively defined by the following rules, up-to congruence $\equiv$ (Def. D.1), plus rules [Γ-μ] and [Γ-Cong] (Def. 2.8):

$$\text{[Γ-AMsg]} \quad s[\mathtt{p}]{:}(M; \mathtt{q}\oplus_{i\in I}\mathtt{m}_i(S_i).S'_i) \xrightarrow{s:\mathtt{p}!\mathtt{q}:\mathtt{m}_k} s[\mathtt{p}]{:}(M\cdot\mathtt{q}!\mathtt{m}_k(S_k)\cdot\epsilon; S'_k) \quad \text{if } k \in I$$

$$\text{[Γ-AComm]} \quad s[\mathtt{p}]{:}\mathtt{q}!\mathtt{m}_k(S_k)\cdot M,\; s[\mathtt{q}]{:}\mathtt{p}\&_{i\in I}\mathtt{m}_i(T_i).T'_i \xrightarrow{s:\mathtt{p},\mathtt{q}:\mathtt{m}_k} s[\mathtt{p}]{:}M, s[\mathtt{q}]{:}T'_k \quad \text{if } k \in I,\; S_k \leqslant T_k$$

Definition E.1 (Partial Asynchronous Projection). The *message queue for* $\mathbf{p}$ *in* M, written $M(\mathbf{p})$, is:

$$(\mathbf{p}!\mathtt{m}(S)\cdot M)(\mathbf{p}) \;=\; \mathbf{p}!\mathtt{m}(S)\cdot(M(\mathbf{p})) \qquad (\mathbf{q}!\mathtt{m}(S)\cdot M)(\mathbf{p}) \;=\; M(\mathbf{p}) \;\;(\text{if } \mathbf{p}\neq\mathbf{q}) \qquad \epsilon(\mathbf{p}) \;=\; \epsilon$$

The *queue prefixing of* M *to* H is the partial type:

$$\mathbf{p}!\mathtt{m}(S)\cdot M \bullet H = \oplus\mathtt{m}(S).(M\bullet H) \qquad \epsilon\bullet H \;=\; H$$

The *projection of* τ *onto* $\mathbf{p}$, written $\tau\!\restriction\!\mathbf{p}$, is a partial session type defined as Def. 3.6 if $\tau = S$, and:

$$M\!\restriction\!\mathbf{p} \;=\; M(\mathbf{p})\bullet\mathbf{end} \qquad (M;S)\!\restriction\!\mathbf{p} \;=\; M(\mathbf{p})\bullet(S\!\restriction\!\mathbf{p})$$

Definition E.2. Γ *is asynchronously consistent*, written a-consistent(Γ), iff $\forall s,\mathbf{p},\mathbf{q},\tau,\tau'$:

$$\Gamma = \Gamma', s[\mathbf{p}]{:}\tau, s[\mathbf{q}]{:}\tau' \;\text{ implies }\; \overline{\tau\!\restriction\!\mathbf{q}} \leqslant \tau'\!\restriction\!\mathbf{p}$$

Table 4. Classic async MPST consistency. These definitions build upon Fig. 3, and are **not** <u>necessary</u> in our new async MPST theory.

The *asynchronous reduction* $\Gamma\to\Gamma'$ is defined iff $\Gamma\xrightarrow{\alpha}\Gamma'$ for some α.

In Def. D.4, both transition rules and labels can be seen as different forms of synchronisation:

- $s{:}\mathbf{p}!\mathbf{q}{:}\mathtt{m}$ denotes an interaction between a session type and its queue, with the addition of a pending message $\mathtt{m}$ sent from $\mathbf{p}$ to $\mathbf{q}$, on session s;
- $s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}$ (that reuses notation from Def. 2.8) denotes the interaction between a queue type and a recipient session type, with the reception by $\mathbf{q}$ of a queued message $\mathtt{m}$ previously sent by $\mathbf{p}$ on session s.

When the transition label is immaterial, we use the reduction $\to$.

E PROBLEMS OF CLASSIC ASYNCHRONOUS MPST

We now outline the issues of classic asynchronous MPST, to overcome them in §F. We summarise the technical definitions in Table 4.

Similarly to § 3, classic async MPST require "asynchronously consistent" (or "a-consistent") typing contexts. Therefore, rule [TA-ν] in Fig. 8 is instantiated as follows:

$$\frac{\Gamma' = \left\{s[\mathbf{p}]{:}S_{\mathbf{p}}\right\}_{\mathbf{p}\in I} \qquad s\notin\Gamma \qquad \text{a-consistent}(\Gamma') \qquad \Theta\cdot\Gamma,\Gamma'\vdash_{\mathbb{S}} P}{\Theta\cdot\Gamma\vdash_{\mathbb{S}\setminus s}(\nu s{:}\Gamma')\,P}\;[\text{TA-}\nu\text{Classic}]$$

Async consistency (Def. E.2) caters for message queues by building upon (and further complicating) Fig. 3, thus inheriting its problems (cf. §3.1) and limitations (cf. §3.2). Moreover, the presence of queues and queue types, and their interaction with consistency, introduce two further difficulties, that complicate the classic subject reduction statement:

Empty Queues and "Missing" Reductions. The typing rules [TA-ϵ]/[TA-σ] never map channels with role to empty queue types: this is visible in Ex. D.3 and Thm. L.5(6). E.g., take a typed process with empty queues $P{=}s[\mathbf{p}][\mathbf{q}]\oplus\mathtt{m}\langle s'[\mathbf{q}']\rangle.\mathbf{0}\mid s\blacktriangleright\epsilon$. By Fig. 8:

$$\frac{\Theta\cdot\Gamma, s[\mathbf{p}]{:}\mathbf{q}\oplus\mathtt{m}(S).\mathbf{end}\vdash_{\emptyset} s[\mathbf{p}][\mathbf{q}]\oplus\mathtt{m}\langle s'[\mathbf{q}']\rangle.\mathbf{0} \qquad \dfrac{}{\Theta\cdot\emptyset\vdash_{\{s\}} s\blacktriangleright\epsilon}\,[\text{TA-}\epsilon]}{\Theta\cdot\Gamma\vdash_{\{s\}} P}\,[\text{TA-}|] \tag{15}$$

$$\text{where }\; \Gamma = s[\mathbf{p}]{:}\mathbf{q}\oplus\mathtt{m}(S).\mathbf{end}, s'[\mathbf{q}']{:}S$$

Note that Γ maps $s[\mathbf{p}]$ to a session type *without* queue. Now, P reduces as:

$$P \longrightarrow P' = \mathbf{0} \mid s \blacktriangleright (\mathsf{p}, \mathsf{q}, \mathtt{m}\langle s'[\mathsf{q}']\rangle) \cdot \epsilon$$
$$\text{with} \quad \Theta \cdot \Gamma' \vdash_{\{s\}} P' \quad \text{and} \quad \Gamma' = s[\mathsf{p}]:(\mathsf{q}!\mathtt{m}(S) \cdot \epsilon; \mathbf{end}), s'[\mathsf{q}']:S$$

Here, Γ' maps $s[\mathsf{p}]$ to a session type paired with a (non-empty) queue. Hence, Γ in (15) cannot match the process transition by reducing to Γ'. By Def. D.4, the "missing" type reduction would be allowed if Γ mapped $s[\mathsf{p}]$ to the pair $(\epsilon; \mathsf{q}\oplus\mathtt{m}(S).\mathbf{end})$.

Async Consistency vs. Context Splits. Async consistency has a limitation w.r.t. synchronous consistency: it does *not* satisfy *desideratum* (D2) in §2.3. In fact, when the parallel typing rule [TA-|] (Fig. 8) splits a typing context, we might have:

$$\text{a-consistent}(\Gamma_1, \Gamma_2) \quad \not\Longrightarrow \quad \text{a-consistent}(\Gamma_1) \tag{16}$$

because Γ_1 might lose queue types in a way that breaks consistency. E.g., if we take:

$$\Gamma_1 = \begin{Bmatrix} s[\mathsf{p}]:\mathsf{q}\oplus\mathtt{m}_2.S', \\ s[\mathsf{q}]:\mathsf{p}\&\mathtt{m}_1.\mathsf{p}\&\mathtt{m}_2.S'' \end{Bmatrix} \qquad \Gamma_2 = s[\mathsf{p}]:\mathsf{q}!\mathtt{m}_1 \cdot \epsilon \qquad (\mathtt{m}_1 \neq \mathtt{m}_2) \tag{17}$$

then Γ_1, Γ_2 is consistent, but Γ_1 is not, due to the mismatching output $\mathtt{m}_2$ and input $\mathtt{m}_1$. Consequently, async consistency is *not* preserved across typing derivations, and cannot be used in an induction hypothesis.

E.g., consider the process P below (slightly simplified by omitting irrelevant message payloads). Its sub-process P_{p} (who plays role p) has already sent a queued message $\mathtt{m}_1$ and is about to send $\mathtt{m}_2$ to q, while the sub-process P_{q} (who plays role q) can receive both messages:

$$\begin{aligned} P &= P_{\mathsf{p}} \mid P_{\mathsf{q}} \mid s \blacktriangleright (\mathsf{p}, \mathsf{q}, \mathtt{m}_1) \cdot \epsilon \\ P_{\mathsf{p}} &= s[\mathsf{p}][\mathsf{q}]\oplus\mathtt{m}_2.P' \\ P_{\mathsf{q}} &= \mathbf{def}\ X(x) = x[\mathsf{p}]\textstyle\sum \mathtt{m}_1.x[\mathsf{p}]\sum \mathtt{m}_2.Q \ \ \mathbf{in}\ \ X\langle s[\mathsf{q}]\rangle \end{aligned}$$

Note that we have $P_{\mathsf{q}} \longrightarrow P'_{\mathsf{q}}$, by expanding the call to X (rules [R-X] and [R-Ctx] in Fig. 1); therefore, by [R-Ctx], we also have $P \longrightarrow P' = P_{\mathsf{p}} \mid P'_{\mathsf{q}} \mid s \blacktriangleright (\mathsf{p}, \mathsf{q}, \mathtt{m}_1) \cdot \epsilon$. Now, if P is well-typed, we have:

$$\dfrac{\Theta \cdot \Gamma_1 \vdash_{\emptyset} P_{\mathsf{p}} \mid P_{\mathsf{q}} \qquad \Theta \cdot \Gamma_2 \vdash_{\{s\}} s \blacktriangleright (\mathsf{p}, \mathsf{q}, \mathtt{m}_1) \cdot \epsilon}{\Theta \cdot \Gamma_1, \Gamma_2 \vdash_{\{s\}} P} \ [\text{TA-}|]$$

$$\text{where} \quad \begin{aligned} \Gamma_1 &= \begin{Bmatrix} s[\mathsf{p}]:\mathsf{q}\oplus\mathtt{m}_2.S', \\ s[\mathsf{q}]:\mathsf{p}\&\mathtt{m}_1.\mathsf{p}\&\mathtt{m}_2.S'' \end{Bmatrix} \\ \Gamma_2 &= s[\mathsf{p}]:\mathsf{q}!\mathtt{m}_1 \cdot \epsilon \end{aligned}$$

Note that Γ_1, Γ_2 is consistent, but Γ_1 is not (due to the mismatching output of $\mathtt{m}_2$ and input of $\mathtt{m}_1$). Thus, if we try to prove subject reduction for $P \longrightarrow P'$ by requiring an a-consistent typing context, we cannot apply the induction hypothesis on the premise $P_{\mathsf{p}} \mid P_{\mathsf{q}} \longrightarrow P_{\mathsf{p}} \mid P'_{\mathsf{q}}$.

Due to the issues above, the classic async subject reduction statement reads [Coppo et al. 2015a, Lemma 1]:

$$\begin{aligned} &\text{If } \ \Theta \cdot \Gamma \vdash_S P \ \ \underline{\text{and } \exists \Gamma_0 \text{ such that } \Gamma, \Gamma_0 \text{ a-consistent}} \ \text{ and } \ P \longrightarrow P', \\ &\quad \text{then } \ \underline{\exists \Gamma', \Gamma'_0 \text{ a-consistent}} \ \text{ such that } \ \Gamma, \Gamma_0 \longrightarrow^* \Gamma', \Gamma'_0 \ \text{ and } \ \Theta \cdot \Gamma' \vdash_S P' \end{aligned} \tag{18}$$

Intuitively, the statement solves the above issues by adding the typing context Γ_0, containing queue types that restore "missing" reductions and consistency.

In §F, we avoid these complications, and obtain more general results, by developing a new async MPST theory, that extends our new theory in §4 with novel semantics for asynchronous typing context, and a novel subject reduction statement and proof technique.

F GENERAL ASYNCHRONOUS MULTIPARTY SESSION TYPE SYSTEM

We now present our async MPST type system. As in §4, it is modular, parametric w.r.t. an *async* safety property φ (Def. F.2); to eschew the classic MPST difficulties summarised in § E, in the following we contribute a novel proof technique for asynchronous subject reduction, based on a novel handling of queue types:

(1) we define a smarter typing context reduction $\rightarrow_{\mathcal{S}}$ (Def. F.1) using the session set $\mathcal{S}$;
(2) we define async typing context $\mathcal{S}$-safety (Def. F.2) by exploiting $\rightarrow_{\mathcal{S}}$ above;
(3) we develop a new subject reduction statement (Thm. F.6), simpler than (18), by exploiting the fact that our $\mathcal{S}$-safety handles queueless type reductions, and survives context splits (Lemma F.5).

Definition F.1. The **async typing context $\mathcal{S}$-transition** $\xrightarrow{\alpha}_{\mathcal{S}}$ is inductively defined by the rules below, up-to $\equiv$ (Def. D.2):

$$\begin{array}{lll}
[\Gamma\text{-Base}] & \Gamma \xrightarrow{\alpha} \Gamma' \text{ implies } \Gamma \xrightarrow{\alpha}_{\mathcal{S}} \Gamma' & \\
[\Gamma\text{-DMsg}] & s[\mathbf{p}]{:}\mathbf{q}\oplus_{i\in I}\mathtt{m}_i(S_i).S'_i \xrightarrow{s:\mathbf{p}!\mathbf{q}:\mathtt{m}_k}_{\mathcal{S}} s[\mathbf{p}]{:}(\mathbf{q}!\mathtt{m}_k(S_k)\cdot\epsilon; S'_k) & \text{if } s\in\mathcal{S},\ k\in I \\
[\Gamma\text{-DCom}] & s[\mathbf{p}]{:}\mathbf{q}!\mathtt{m}_k(S_k)\cdot\epsilon,\ s[\mathbf{q}]{:}\mathbf{p}\&_{i\in I}\mathtt{m}_i(T_i).T'_i \xrightarrow{s:\mathbf{p},\mathbf{q}:\mathtt{m}_k}_{\mathcal{S}} s[\mathbf{q}]{:}T'_k & \text{if } s\in\mathcal{S},\ k\in I,\ S_k\leqslant T_k
\end{array}$$

plus rules $[\Gamma\text{-}\mu]$ and $[\Gamma\text{-Cong}]$ (Def. 2.8), replacing $\xrightarrow{\alpha}$ with $\xrightarrow{\alpha}_{\mathcal{S}}$.
We write $\Gamma\xrightarrow{\alpha}_{\mathcal{S}}$ iff there is Γ' such that $\Gamma \xrightarrow{\alpha}_{\mathcal{S}} \Gamma'$. We $\Gamma \rightarrow_{\mathcal{S}} \Gamma'$ iff $\Gamma \xrightarrow{\alpha}_{\mathcal{S}} \Gamma'$ for some α.

In Def. F.1, rule [Γ-Base] says that any *async* reduction $\Gamma \xrightarrow{\alpha} \Gamma'$ (Def. D.4) is matched by $\xrightarrow{\alpha}_{\mathcal{S}}$. To support reductions of session types without queues, [Γ-DMsg] allows an internal choice to reduce by creating a queue type carrying its output, and [Γ-DCom] allows a queue type to disappear when its last message is consumed. Crucially, [Γ-DMsg] and [Γ-DCom] only apply for sessions in $\mathcal{S}$; otherwise, $\xrightarrow{\alpha}_{\mathcal{S}}$ matches $\xrightarrow{\alpha}$, i.e., queueless types do not reduce.

Definition F.2 (Asynchronous Safety). φ is an $\mathcal{S}$-*safety property* on typing contexts iff:

$$\begin{array}{ll}
[\text{SA-}\&!] & \varphi\big(\Gamma,\ s[\mathbf{p}]{:}\mathbf{q}\&_{i\in I}\mathtt{m}_i(S_i).S'_i,\ s[\mathbf{q}]{:}M\big) \text{ and } M\equiv\mathbf{p}!\mathtt{m}(T)\cdot M' \text{ implies } \exists k\in I{:}\ \mathtt{m}_k=\mathtt{m},\ T\leqslant S_k; \\
[\text{SA-}\mu] & \varphi(\Gamma,\ s[\mathbf{p}]{:}\mu\mathbf{t}.S) \text{ implies } \varphi(\Gamma,\ s[\mathbf{p}]{:}S\{\mu\mathbf{t}.S/\mathbf{t}\}); \\
[\text{SA-}\rightarrow] & \varphi(\Gamma) \text{ and } \Gamma\rightarrow_{\mathcal{S}}\Gamma' \text{ implies } \varphi(\Gamma').
\end{array}$$

We say Γ *is asynchronously* $\mathcal{S}$*-safe*, or a-safe$_{\mathcal{S}}(\Gamma)$, iff $\varphi(\Gamma)$ for some $\mathcal{S}$-safety property φ.

The notion of $\mathcal{S}$-safety in Def. F.2 is akin to Def. 4.1, but caters for asynchrony: if a queue has a top-level message from $\mathbf{p}$ to $\mathbf{q}$, and $\mathbf{q}$ is trying to receive from $\mathbf{p}$, then their messages must be compatible and allow them to reduce, by Def. D.4. Note that clause [SA-→] uses $\rightarrow_{\mathcal{S}}$: i.e., if a queueless type belongs to a session in $\mathcal{S}$ it can reduce, otherwise is stuck and ignored, as shown in Ex. F.3.

Example F.3. Take Γ_1 and Γ_2 from (16) above. We have a-safe$_{\{s\}}(\Gamma_1,\Gamma_2)$, but a-safe$_{\{s\}}(\Gamma_1)$ does *not* hold: Γ_1 reduces by $\rightarrow_{\{s\}}$, queuing message $\mathtt{m}_2$, and violating clause [SA-&!] (Def. F.2). Instead, a-safe$_{\emptyset}(\Gamma_1)$ holds, since $\Gamma_1\not\rightarrow_{\emptyset}$, and Γ_1 (vacuously) satisfies Def. F.2.

We now have all the ingredients for our general async type system.

Definition F.4. The **general asynchronous MPST typing judgement** is induced by the rules in Fig. 8 — with rule [TA-ν] restricted as follows:

$$\frac{\Gamma' = \{s[\mathbf{p}]{:}S_{\mathbf{p}}\}_{\mathbf{p}\in I} \quad \varphi(\Gamma') \quad s\notin\Gamma \quad \Theta\cdot\Gamma,\Gamma'\vdash_{\mathcal{S}} P}{\Theta\cdot\Gamma\vdash_{\mathcal{S}\setminus s}(\nu s{:}\Gamma')\,P}\;[\text{TAGen-}\nu] \qquad \begin{array}{c}\text{where } \varphi \text{ is an}\\ \{s\}\text{-safety property}\end{array}$$

We write "$\Theta\cdot\Gamma\vdash P$ *with* φ" to specify how to instantiate φ in rule [TAGen-ν] above. When "*with* φ" is omitted, then the instantiation is $\varphi=\text{a-safe}_{\mathcal{S}_U}$ (i.e., the largest $\mathcal{S}_U$-safety property, cf. Def. F.2) where $\mathcal{S}_U$ is the set of all sessions.

As in Def. 4.6, Def. F.4 provides a novel foundation for asynchronous MPST, but has just one visible change w.r.t. the classic multiparty session typing rules: the rule for session restriction, that uses a parametric, behavioural (rather than syntactic) property on Γ'. Note that, crucially, we exploit the judgement's session set to determine $\mathcal{S}$ for async $\mathcal{S}$-safety (Def. F.2); by using the same $\mathcal{S}$ for $\rightarrow_{\mathcal{S}}$ (Def. F.1), we are able to formalise and prove asynchronous subject reduction, type safety, and session fidelity, as follows. Crucially, Thm. F.6 (that provides the precise formal statement of Thm. 7.1) uses Lemma F.5 as a weak form of *desideratum* **(D2)** in § 2.3: it provides fine-grained splits of async typing contexts (again, depending on the session set $\mathcal{S}$) that preserve safety along the subject reduction proof.

Lemma F.5. *Let* $\text{a-safe}_{\mathcal{S}}(\Gamma)$*: then,* $\text{a-safe}_{\mathcal{S}\setminus s}(\Gamma)$*; and if* $\Gamma = \Gamma', s[\mathsf{p}]{:}S$*, then* $\text{a-safe}_{\mathcal{S}}(\Gamma')$.

Theorem F.6 (Asynchronous Subject Reduction). *Assume* $\Theta \cdot \Gamma \vdash_{\mathcal{S}} P$ *with* Γ $\mathcal{S}$*-safe. Then,* $P \rightarrow P'$ *implies* $\exists \Gamma'$ $\mathcal{S}$*-safe such that* $\Gamma \rightarrow^{*}_{\mathcal{S}} \Gamma'$ *and* $\Theta \cdot \Gamma' \vdash_{\mathcal{S}} P'$.

Corollary F.7 (Async Type Safety). *If* $\emptyset \cdot \emptyset \vdash_{\emptyset} P$ *and* $P \rightarrow^{*} P'$*, then* P' *has no errors.*

Theorem F.8 (Async Session Fidelity). *Let* $\Theta \cdot \Gamma \vdash_{\mathcal{S}} P$*, with* $P \equiv \left(\prod_{\mathsf{p}\in I} P_{\mathsf{p}}\right) \mid s \blacktriangleright \sigma$*, and each* P_{p} *either being* $\mathbf{0}$ *(up-to* $\equiv$*), or only playing role* p *in* s*. Then,* $\Gamma \rightarrow_{\mathcal{S}}$ *implies* $\exists \Gamma', P'$ *such that* $\Gamma \rightarrow_{\mathcal{S}} \Gamma'$*,* $P \rightarrow^{*} P'$ *and* $\Theta \cdot \Gamma' \vdash_{\mathcal{S}} P'$*, with* $P' \equiv \left(\prod_{\mathsf{p}\in I} P'_{\mathsf{p}}\right) \mid s \blacktriangleright \sigma'$ *and each* P'_{p} *either being* $\mathbf{0}$ *(up-to* $\equiv$*), or only playing role* p *in* s*.*

Finally, similarly to the synchronous case (Thm. 4.11), also asynchronous type checking is decidable, when instantiated with a decidable safety property.

Theorem 7.2. *If* φ *is decidable, then "*$\Theta \cdot \Gamma \vdash_{\mathcal{S}} P$ *with* φ*" is decidable.*

G FROM ASYNC TYPING CONTEXT PROPERTIES TO PROCESS PROPERTIES

As in § 5, we now present several properties reinforcing $\text{a-safe}_{\mathcal{S}}$, compare them, and use them to instantiate φ in our type system, to predict and constrain the run-time behaviour of processes. However, under asynchrony, we need to address an additional challenge: since queue types are unbounded, an asynchronous typing context Γ can induce an *infinite* state transition system, making its properties *undecidable* (unlike Thm. 5.13).

Def. G.2 below is the async version of the typing context properties discussed in § 5.3. The key difference is that Def. G.2 checks queued message types, instead of internal choices. But first, we need to formalise the asynchronous version of Def. 5.5 (fair traversal sets), in Def. G.1 below. Its purpose is similar: find a set of roles that can interact and always reach a target state, under "fair scheduling"; but in Def. G.1, we also choose the "right time" to fire queuing and reception transitions: this allows to ignore unfair executions where a recursive output is fired infinitely, and enqueues infinitely many messages, without giving a chance to the recipient to consume them.

Definition G.1 (Asynchronous fair traversal set). Let $\mathbb{X}, \mathbb{Y}$ be sets of asynchronous typing contexts. We say that $\mathbb{X}$ *is a fair traversal set for sessions* $\mathcal{S}$ *with targets* $\mathbb{Y}$ iff $\mathbb{X}$ is closed under the rules:

$$\frac{\Gamma \in \mathbb{Y}}{\Gamma \in \mathbb{X}}\;[\text{TSA-Target}] \qquad \frac{\exists s\in\mathcal{S}, \mathsf{p},\mathsf{q} : \quad \exists \mathsf{m} : \begin{cases} \Gamma \xrightarrow{s:\mathsf{p},\mathsf{q}:\mathsf{m}}_{\mathcal{S}} \text{ and } \left(\Gamma \xrightarrow{s:\mathsf{p},\mathsf{q}:\mathsf{m}}_{\mathcal{S}} \Gamma' \text{ implies } \Gamma' \in \mathbb{X}\right) \\ \text{or} \\ \Gamma \xrightarrow{s:\mathsf{p}!\mathsf{q}:\mathsf{m}}_{\mathcal{S}} \text{ and } \left(\forall \mathsf{m} : \Gamma \xrightarrow{s:\mathsf{p}!\mathsf{q}:\mathsf{m}}_{\mathcal{S}} \Gamma' \text{ implies } \Gamma' \in \mathbb{X}\right) \end{cases}}{\Gamma \in \mathbb{X}}\;[\text{TS-IO}]$$

We can now formalise Def. G.2. Most definitions match those in Fig. 5; the additional item (7) places a bound on the length of type queues: it will be useful later, for Thm. G.5.

Definition G.2 (Properties of Asynchronous Typing Contexts). We write $\text{a-end}_{\mathcal{S}}(\Gamma)$ iff $\forall s[\mathbf{p}] \in \text{dom}(\Gamma)$ with $s \in \mathcal{S}$, $\Gamma(s[\mathbf{p}]) \in \{S, \epsilon, (\epsilon; S) \mid S \leqslant \mathbf{end}\}$.

(1) We say that Γ **is $\mathcal{S}$-deadlock-free**, or $\text{a-df}_{\mathcal{S}}(\Gamma)$, iff $\Gamma \rightarrow^{*}_{\mathcal{S}} \Gamma' \not\rightarrow_{\mathcal{S}}$ implies $\text{a-end}_{\mathcal{S}}(\Gamma')$.

(2) We say that Γ **is $\mathcal{S}$-terminating**, written $\text{a-term}_{\mathcal{S}}(\Gamma)$, iff Γ is $\mathcal{S}$-deadlock-free, and there is $k \in \mathbb{N}$ such that for all $n \geq k$, $\Gamma = \Gamma_0 \rightarrow_{\mathcal{S}} \Gamma_1 \rightarrow_{\mathcal{S}} \cdots \rightarrow_{\mathcal{S}} \Gamma_n$ implies $\text{a-end}_{\mathcal{S}}(\Gamma_n)$.

(3) We say that Γ **is $\mathcal{S}$-never-terminating**, written $\text{a-nterm}_{\mathcal{S}}(\Gamma)$, iff $\Gamma \rightarrow^{*}_{\mathcal{S}} \Gamma'$ implies $\Gamma' \rightarrow_{\mathcal{S}}$.

(4) φ is an $\mathcal{S}$-**liveness property** on asynchronous typing contexts iff:

[LA-&] $\varphi(\Gamma, s[\mathbf{p}]{:}S)$ with $S = \mathbf{q} \&_{i \in I} \mathsf{m}_i(S_i).S'_i$ implies $\exists i \in I : \exists \Gamma' : \Gamma, s[\mathbf{p}]{:}S \rightarrow^{*}_{\mathcal{S}} \Gamma', s[\mathbf{p}]{:}S'_i$

[LA-!] $\varphi(\Gamma, s[\mathbf{p}]{:}M)$ with $M \equiv \mathbf{q}!\mathsf{m}(S) \cdot M'$ implies $\exists \Gamma' : \Gamma, s[\mathbf{p}]{:}M \rightarrow^{*}_{\mathcal{S}} \Gamma', s[\mathbf{p}]{:}M'$

plus clauses [SA-μ] and [SA-$\rightarrow$] from Def. F.2. We say that Γ *is asynchronously $\mathcal{S}$-live*, written $\text{a-live}_{\mathcal{S}}(\Gamma)$, iff $\varphi(\Gamma)$ for some liveness property φ.

(5) φ is an $\mathcal{S}$-**liveness^{+} property** iff:

[LA-&$^{+}$] clause [LA-&] above; *moreover*, Γ belongs to some async fair traversal set $\mathbb{X}$ for sessions $\mathcal{S}$ with targets $\mathbb{Y}$ (Def. G.1) such that, $\forall \Gamma_t \in \mathbb{Y}$, we have $\Gamma_t = \Gamma'', s[\mathbf{p}]{:}S'_i$ (for some Γ'', $i \in I$)

[LA-!$^{+}$] clause [LA-!] above, *moreover*, Γ belongs to some async fair traversal set $\mathbb{X}$ for sessions $\mathcal{S}$ with targets $\mathbb{Y}$ (Def. G.1) such that, $\forall \Gamma_t \in \mathbb{Y}$, we have $\Gamma_t = \Gamma'', s[\mathbf{p}]{:}M'$ (for some Γ'')

plus clauses [SA-μ] and [SA-$\rightarrow$] from Def. F.2. We say Γ *is asynchronously $\mathcal{S}$-live^{+}*, or $\text{a-live}^{+}_{\mathcal{S}}(\Gamma)$, iff $\varphi(\Gamma)$ for some $\mathcal{S}$-liveness^{+} property φ.

(6) φ is an $\mathcal{S}$-**liveness^{++} property** iff:

[LA-&$^{++}$] clause [LA-&] above; *moreover*, there is $n \in \mathbb{N}$ such that if $\Gamma = \Gamma_0 \rightarrow_{\mathcal{S}} \Gamma_1 \rightarrow_{\mathcal{S}} \cdots \rightarrow_{\mathcal{S}} \Gamma_n$, $\exists i < n$ such that $\Gamma_i \rightarrow_{\mathcal{S}} \Gamma_{i+1} = \Gamma'', s[\mathbf{p}]{:}S'_i$ (for some Γ'', $i \in I$)

[LA-!$^{++}$] clause [LA-!] above, *moreover*, there is $n \in \mathbb{N}$ such that if $\Gamma = \Gamma_0 \rightarrow_{\mathcal{S}} \Gamma_1 \rightarrow_{\mathcal{S}} \cdots \rightarrow_{\mathcal{S}} \Gamma_n$, $\exists i < n$ such that $\Gamma_i \rightarrow_{\mathcal{S}} \Gamma_{i+1} = \Gamma'', s[\mathbf{p}]{:}M'$ (for some Γ'')

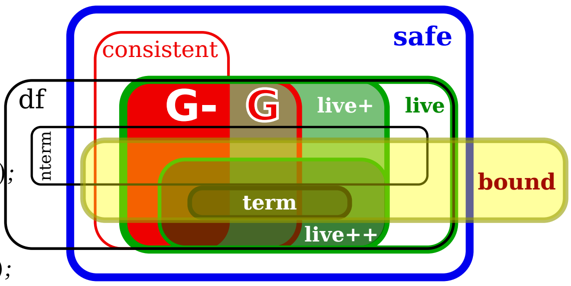
plus clauses [SA-μ] and [SA-$\rightarrow$] from Def. F.2. We say Γ *is asynchronously $\mathcal{S}$-live^{++}*, or $\text{a-live}^{++}_{\mathcal{S}}(\Gamma)$, iff $\varphi(\Gamma)$ for some $\mathcal{S}$-liveness^{++} property φ.

(7) Γ **is k-bounded (w.r.t. $\mathcal{S}$)**, or $\text{a-bound}_{\mathcal{S},k}(\Gamma)$, iff $k \in \mathbb{N}$ and $\Gamma \rightarrow^{*}_{\mathcal{S}} \Gamma, s[\mathbf{p}]{:}M$ implies $|M| \leq k$. Γ *is bounded (w.r.t. $\mathcal{S}$)*, or $\text{a-bound}_{\mathcal{S}}(\Gamma)$, iff $\exists k$ finite: $\text{a-bound}_{\mathcal{S},k}(\Gamma)$.

Lemma G.3 below is the async version of Lemma 5.9. Items 8 and 9 show that boundedness does not imply any other property, and is only implied by termination.

Lemma G.3. *For all Γ, letting $\mathcal{S} = \{s \mid \exists \mathbf{p} : s[\mathbf{p}] \in \text{dom}(\Gamma)\}$, we have:*

(1) $\text{a-consistent}(\Gamma) \;\;\not\Leftarrow\!\!\Longrightarrow\;\; \text{a-safe}_{\mathcal{S}}(\Gamma)$;
(2) $\text{a-live}_{\mathcal{S}}(\Gamma) \;\;\not\Leftarrow\!\!\Longrightarrow\;\; \text{a-safe}_{\mathcal{S}}(\Gamma)$;
(3) $\text{a-live}_{\mathcal{S}}(\Gamma) \;\;\not\Leftarrow\!\!\Longrightarrow\;\; \text{a-df}_{\mathcal{S}}(\Gamma)$;
(4) $\text{a-nterm}_{\mathcal{S}}(\Gamma) \;\;\not\Leftarrow\!\!\Longrightarrow\;\; \text{a-df}_{\mathcal{S}}(\Gamma)$;
(5) $\text{a-consistent}(\Gamma) \;\;\not\Leftarrow\!\!\not\Rightarrow\;\; \text{a-df}_{\mathcal{S}}(\Gamma)$;
(6) $\text{a-consistent}(\Gamma) \wedge \text{a-df}_{\mathcal{S}}(\Gamma) \;\;\not\Leftarrow\!\!\not\Rightarrow\;\; \text{a-live}_{\mathcal{S}}(\Gamma)$;
(7) $\text{a-term}_{\mathcal{S}}(\Gamma) \;\;\not\Leftarrow\!\!\Longrightarrow\;\; \text{a-live}^{++}_{\mathcal{S}}(\Gamma)$;
(8) $\text{a-term}_{\mathcal{S}}(\Gamma) \;\;\not\Leftarrow\!\!\Longrightarrow\;\; \text{a-bound}_{\mathcal{S}}(\Gamma)$;
(9) $\text{a-bound}_{\mathcal{S}}(\Gamma) \;\;\not\Leftarrow\!\!\not\Rightarrow\;\; \text{a-safe}_{\mathcal{S}}(\Gamma) \vee \text{a-df}_{\mathcal{S}}(\Gamma)$;
(10) $\text{a-live}^{++}_{\mathcal{S}}(\Gamma) \;\;\not\Leftarrow\!\!\Longrightarrow\;\; \text{a-live}^{+}_{\mathcal{S}}(\Gamma) \;\;\not\Leftarrow\!\!\Longrightarrow\;\; \text{a-live}_{\mathcal{S}}(\Gamma)$.



Example G.4. Ex. 5.10 and Ex. 5.11 also hold under the async properties of Def. G.2. Moreover:

(1) The typing context Γ from Ex. 2.7 (i.e., our example in § 1) is bounded;
(2) Γ_B from Ex. 5.11 is *not* bounded: $s[\mathbf{p}]$ can queue infinite outputs, before $s[\mathbf{q}]$ consumes them;
(3) the "arbitrary typing context" $s[\mathbf{p}]{:}\mathbf{q}{\oplus}\mathtt{foo}(\mathbf{end}), s[\mathbf{q}]{:}\mathbf{p}\&\mathtt{bar}(\mathbf{end}), s'[\mathbf{r}]{:}\mathbf{end}$ from § 2.3 is async bounded, but not async safe nor deadlock-free;
(4) $s[\mathbf{p}]{:}\mu\mathbf{t}.\mathbf{q}{\oplus}\mathtt{m}_1.\mathbf{q}\&\mathtt{m}_2.\mathbf{t},\ s[\mathbf{q}]{:}\mu\mathbf{t}.\mathbf{p}{\oplus}\mathtt{m}_2.\mathbf{p}\&\mathtt{m}_1.\mathbf{t}$ is *not* a-consistent, and is deadlocked under synchronous reductions (Def. 2.8); however, under $\rightarrow_{\{s\}}$ (Def. D.4) it is async live$^+$ (hence async safe) and bounded ($k{=}2$). In fact, both $s[\mathbf{p}]$ and $s[\mathbf{q}]$ can reduce by enqueuing their initial output messages ($\mathtt{m}_1$ and $\mathtt{m}_2$), and then receiving the message enqueued by the other party;
(5) $s[\mathbf{p}]{:}\mu\mathbf{t}.\mathbf{q}{\oplus}\mathtt{m}_1.\mathbf{q}{\oplus}\mathtt{m}_1.\mathbf{q}\&\mathtt{m}_2.\mathbf{t},\ s[\mathbf{q}]{:}\mu\mathbf{t}.\mathbf{p}{\oplus}\mathtt{m}_2.\mathbf{p}\&\mathtt{m}_1.\mathbf{t}$ (akin to (4) above) is live$^+$, but *not* a-consistent *nor* bounded: in each loop, two messages $\mathtt{m}_1$ are queued and only one is consumed.

Note that the diagram of Lemma G.3 includes $\mathbb{G}$ and $\mathbb{G}\!\!-$ from Lemma 5.9, i.e., the sets of contexts projected from a global type: they do not occur in the statement, but will be justified by Thm. G.6 and Remark G.7.

Decidability. Thm. G.5 below provides minimal decidability criteria for async properties.

Theorem G.5. *$\forall\Gamma$, a-consistent(Γ) is decidable. Furthermore, $\forall\mathcal{S}, k$, a-bound$_{\mathcal{S},k}(\Gamma)$ is decidable; and if a-bound$_{\mathcal{S},k}(\Gamma)$ holds, then Γ has a finite state transition system, hence a-safe$_{\mathcal{S}}(\Gamma)$, a-df$_{\mathcal{S}}(\Gamma)$, a-term$_{\mathcal{S}}(\Gamma)$, a-nterm$_{\mathcal{S}}(\Gamma)$, a-live$_{\mathcal{S}}(\Gamma)$, a-live$^+_{\mathcal{S}}(\Gamma)$ and a-live$^{++}_{\mathcal{S}}(\Gamma)$ are decidable.*

By Thm. G.5 and Thm. 7.2, we obtain various decidable instances of our general asynchronous MPST type system. Such instances require either asynchronous consistency (as in classic async MPST), or the enforcement of a limit on the size of queue types, to ensure that the properties in Def. G.2 are decidable.

Besides such results, decidability of φ (and type checking) is hampered due to the correspondence between session/queue types and Communicating Finite State Machine (CFSM) [Deniélou and Yoshida 2013]: *(1)* a session/queue type $(M; S)$ is a CFSM with finite control S and output queues M, and *(2)* an async Γ is a system of interacting CFSMs. Unfortunately, safety and other properties in Def. G.2 (including the *existence* of a queue bound k) are undecidable for CFSMs [Brand and Zafiropulo 1983] [Genest et al. 2007], and Γ can encode a Turing machine [Bartoletti et al. 2016, Thm. 2.5]. To address this problem, we cannot straightforwardly lift decidable synchronous properties to asynchrony, e.g.:

$$\begin{array}{rcl} \Gamma & = & s[\mathbf{p}]{:}\mathbf{r}{\oplus}\mathtt{m}_1.\mathbf{q}{\oplus}\mathtt{m}_2,\ s[\mathbf{q}]{:}\mathbf{p}\&\mathtt{m}_3 \qquad \text{with } \mathtt{m}_2 \neq \mathtt{m}_3 \\ \Gamma \rightarrow_{\{s\}} \rightarrow_{\{s\}} & \Gamma' \equiv & s[\mathbf{p}]{:}(\mathbf{q}!\mathtt{m}_2\cdot\mathbf{r}!\mathtt{m}_1\cdot\epsilon; \mathbf{end}),\ s[\mathbf{q}]{:}\mathbf{p}\&\mathtt{m}_3 \end{array}$$

Note that Γ is synchronously safe (Def. 4.1), but *not* asynchronously $\{s\}$-safe (Def. F.2): the outputs of $s[\mathbf{p}]$ are queued, and Γ' violates Def. F.2 ([SA-&!]). Luckily, Thm. G.6 says that the session types/CFSMs correspondence allows to lift *liveness* to asynchrony; and building upon this basis, we can also lift liveness$^+$ to asynchrony (notice that the latter is a completely new result).

Theorem G.6. *Let Γ be a* synchronous *typing context (Def. 2.6); moreover, let $\mathcal{S} = \{s \mid \exists \mathbf{p} : s[\mathbf{p}] \in \operatorname{dom}(\Gamma)\}$. Then,* live$(\Gamma)$ *is decidable, and implies* a-live$_{\mathcal{S}}(\Gamma)$*. Moreover,* live$^+(\Gamma)$ *is decidable, and implies* a-live$^+_{\mathcal{S}}(\Gamma)$.

Proof. [Deniélou and Yoshida 2013, Def. 4.2] and [Bocchi et al. 2015, Def. 4] define *Multiparty Compatibility (MC)* as a behavioural property based on Γ's *synchronous* reductions; its definition corresponds to our *synchronous* liveness (Fig. 5); further, MC is decidable (as our Thm. 5.13). Finally, [Deniélou and Yoshida 2013, Thm 4.1] and [Bocchi et al. 2015, Thm. 6] say that if Γ is live/MC, then queued outputs are eventually consumed, and external choices are eventually triggered, as in *async* liveness (Def. G.2). This leads to the result on liveness. For the result on liveness$^+$, we further leverage the live/MC correspondence above to rule out the existence of communication loops that would not allow to build the fair traversal sets required by clauses [LA-&$^+$]/[LA-!$^+$] of Def. G.2(5). For further details, see §N. □

Remark G.7. *With Theorems G.5, G.6, and 7.2, we can instantiate φ in Def. F.4 as described in §7 (methods* **(M1)**–**(M3)***), to obtain various decidable type-checking methods for async processes. In particular, by Lemma 5.9(9) and Thm. G.6, we can use global types to project* async live$^+$ *typing contexts (hence the sets $\mathbb{G}$ and $\mathbb{G}$— in the diagram of Lemma G.3). Thus, such decidable instances of our type system subsume classic async MPST (that only cover φ =* a-consistent*), and type the async version of our example in §1 — i.e., Remark 5.12 also applies to async MPST. Also note that we support protocols that are "inherently" asynchronous, and cannot be projected from any global type — cf. Ex. G.4(4),(5).*

To conclude, we show how the properties defined above influence process behaviours (Thm. G.9). These results are new, and intuitively similar to Thm. 5.15 (for synchronous MPST), modulo the presence of message queues.

Definition G.8. P **is asynchronously deadlock-free** iff $P \rightarrow^* P' \not\rightarrow$ implies $P' \equiv \mathbf{0}$. P is **asynchronously terminating** iff it is async deadlock-free, and $\exists k$ finite such that, $\forall n \geq k$, $P = P_0 \rightarrow P_1 \rightarrow \cdots \rightarrow P_n$ implies $P_n \equiv \mathbf{0}$. P is **asynchronously never-terminating** iff $P \rightarrow^* P'$ implies $P' \rightarrow$. We say that P **is asynchronously live** iff $P \rightarrow^* P' \equiv \mathbb{C}[Q]$ implies:

(1) if $Q = s \blacktriangleright \sigma$ with $\sigma \equiv (\mathsf{p}, \mathsf{q}, \mathsf{m}\langle s'[\mathsf{r}]\rangle) \cdot \sigma'$, then $P' \rightarrow^* \mathbb{C}'[s \blacktriangleright \sigma']$ for some $\mathbb{C}'$;
(2) if $Q = c[\mathsf{q}] \sum_{i \in I} \mathsf{m}_i(x_i).Q_i'$, then $P' \rightarrow^* \mathbb{C}'\left[Q_k'\{s'[\mathsf{r}]/x_k\}\right]$ for some $\mathbb{C}', k \in I, s', \mathsf{r}$;

P is **strongly asynchronously live** iff $P \rightarrow^* P' \equiv \mathbb{C}[Q]$ implies:

(3) item 1 above, and moveover, there is n finite such that, whenever $P' = P_0' \rightarrow P_1' \rightarrow \cdots \rightarrow P_n'$, then for some $j \leq n$ we have $P_j' \rightarrow \mathbb{C}''[s \blacktriangleright \sigma']$ (for some $\mathbb{C}''$);
(4) item 2 above, and moveover, there is n finite such that, whenever $P' = P_0' \rightarrow P_1' \rightarrow \cdots \rightarrow P_n'$, then for some $j \leq n$ we have $P_j' \rightarrow \mathbb{C}''\left[Q_k'\{s'[\mathsf{r}]/x_k\}\right]$ (for some $\mathbb{C}'', k \in I, s', \mathsf{r}$).

Theorem G.9. *Assume $\emptyset \cdot \Gamma \vdash_{\mathbb{S}} P \equiv \left(\big|_{\mathsf{p} \in I} P_{\mathsf{p}}\right) \mid s \blacktriangleright \sigma$, with all P_{p} having guarded definitions and being* $\mathbf{0}$ *(up-to $\equiv$), or only playing p in s. Then: (1)* a-df$_{\mathbb{S}}(\Gamma)$ *implies P is async deadlock-free; (2)* a-term$_{\mathbb{S}}(\Gamma)$ *implies P is async terminating; (3)* a-nterm$_{\mathbb{S}}(\Gamma)$ *implies P is async never-terminating; (4)* a-live$^+_{\mathbb{S}}(\Gamma)$ *implies P is async live. (5)* a-live$^{++}_{\mathbb{S}}(\Gamma)$ *implies P is strongly async live.*

Remark G.10. *The* a-df$_{\mathbb{S}}(\Gamma)$, a-live$_{\mathbb{S}}(\Gamma)$ a-live$^+_{\mathbb{S}}(\Gamma)$ *hypothesis used in Thm. G.9(1) are generally undecidable, but they are implied by (decidable) synchronous liveness/liveness+ (by Thm. G.6 and Lemma G.3) — which in turn, can be verified using the model checking techniques discussed in §6.*

Moreover, the premises of items (1)–(5) of Thm. G.9 can be decided by first checking whether Γ has k-bounded queues, given some k (by Thm. G.5). Furthermore, as highlighted in Remark G.7, global types are a (decidable) way to project typing contexts that are a-live$^+$, and thus, ensure that typed processes are live (by Thm. G.9(4)). These results are more general than those achievable under classic MPST, because they do not require the existence of global types, nor consistency of Γ.

H ADDITIONAL RELATED AND FUTURE WORK

The main related works are discussed in § 8. In this section, we discuss other approaches for formalising protocols and typing processes, and show why they cannot handle our examples; we also discuss other papers and extensions that are specifically related to our general asynchronous MPST theory.

H.1 Conversation Types

Conversation types where proposed by [Caires and Vieira 2009, 2010], as a typing discipline for ensuring that processes implement given protocols. They share various goals and fundamental ideas with session types (including a notion of *duality*), but their technical development is rather different; among their main features, the flexible assignment of protocol roles to processes, and the capability of letting processes dynamically join existing sessions; both features allow to model forms of multiparty interaction.

Conversation types do not have consistency requirements that can be directly compared with those of classic MPST; still, they have several type-level operators and constraints that remind syntactic duality, and do *not* support our examples in Fig. 4.

Consider, e.g., our opening example (§ 1). The conversation type corresponding to the global type G in (1) is (19) below, where $\tau\mathtt{m}(T)$ denotes a communication where two parties exchange message $\mathtt{m}$ carrying a value of type T, and $\oplus$ denotes an internal choice:

$$\oplus\begin{Bmatrix}\tau\mathtt{login}.\tau\mathtt{passwd}(\mathsf{Str}).\tau\mathtt{auth}(\mathsf{Bool})\,,\\ \tau\mathtt{cancel}.\tau\mathtt{quit}\end{Bmatrix} \tag{19}$$

Note the lack of explicit roles in (19). This is a distinguishing feature of conversation types: they can be decomposed in various ways, using a *merge relation* $\bowtie$ [Caires and Vieira 2010, Def. 3.11]. In order to match the scenario in § 1, we would need to decompose (19) into three types, combined by $\bowtie$: this is the only way to type three separate processes (for the service, client, and authorisation server), using rule [PAR] in [Caires and Vieira 2010, Fig. 9]. Therefore, we would like to decompose (19) as follows, where $!/?$ denote output/input messages, and $\&$ is an external choice:

$$\oplus\begin{Bmatrix}!\mathtt{login}.?\mathtt{auth}(\mathsf{Bool})\,,\\ !\mathtt{cancel}\end{Bmatrix} \;\bowtie\; \&\begin{Bmatrix}?\mathtt{login}.!\mathtt{passwd}(\mathsf{Str})\,,\\ ?\mathtt{cancel}.!\mathtt{quit}\end{Bmatrix} \;\bowtie\; \&\begin{Bmatrix}?\mathtt{passwd}(\mathsf{Str}).!\mathtt{auth}(\mathsf{Bool})\,,\\ ?\mathtt{quit}\end{Bmatrix} \tag{20}$$

However, the merging in (20) is undefined: by [Caires and Vieira 2010, Def. 3.11], the rightmost external choice (i.e., the type of the authorisation server) can only be merged with an internal choice $\oplus$ having dual output messages (rules [PLAIN-L]/[PLAIN-R] in [Caires and Vieira 2010, FIg. 8]) — but the other types in (20) do *not* satisfy this requirement: this is because the desired output messages are prefixed by (i.e., depend on) choices and other interactions.[5] Alternatively, one might try to adjust (19), and leverage *directions* and their *projection* [Caires and Vieira 2010, Def. 3.7], aiming at an implementation that starts a binary conversation between the service and client, and involves the authorisation server at a later stage. This would be a significant change; besides, it would still require to decompose the overall protocol using $\bowtie$, with the limitations described above.

As a future research direction, it would be interesting to investigate whether our approach based on safety (Def. 4.1) can provide a new foundation for conversation types, replacing its merge relation with a more flexible alternative.

H.2 Global Types Semantics and Choreographic Programming

Various works investigate the semantic properties of global types (also called *choreographies*) and their projections: for example, [Castagna et al. 2012; Lanese et al. 2008]. Unlike the present work, such papers do not investigate type systems (i.e., do not use projections as types), nor type safety results; hence, they do not address the issues described in § 2.3, nor develop results like ours.

[5]This reminds the reason why, in classic MPST, some partial projections of the same protocol are undefined, cf. § 3.1. This suggests a common fundamental limitation: the authorisation protocol in § 1, and all other examples in Fig. 4, are inherently multiparty, and cannot be cleanly decomposed into sets of binary interactions; however, a binary decomposition is required both in classic MPST (via partial projection/consistency), and in conversation types (via merging). Our new MPST theory (§ 4) does not have this requirement.

On a related line of research, [Carbone and Montesi 2013] propose choreographic programming: an approach where concurrent and distributed application are directly developed as *choreographies*, using a language that reminds a global type specification (Def. 3.2) enriched with programming constructs (e.g., variables, data, conditionals). Choreographic programs are compiled by projecting them into endpoint processes (in a variant of the multiparty session π-calculus); once deployed and executed, such processes interact as specified in the original choreographic program, which is deadlock-free by construction.

The choreographic programming approach is somewhat "opposite" w.r.t. MPST, and does not address scenarios like our example in § 1. In fact, in [Carbone and Montesi 2013], distributed applications are developed as *single* programs, type-checked against a global type; unlike MPST, distributed components are not supposed to be developed independently, and separately verified against a desired interface (i.e., a session type). For example:

- with MPST, the **a**uthorisation server, the **c**lient and the **s**ervice in § 1 can be developed and verified separately and independently, by different programmers, working on distinct codebases. Hence, the developer of the **a**uthorisation server only needs to type-check the implementation P_{a} against the session type S_{a} in (2) (cf. Ex. 2.7); this is enough to guarantee correct interaction with other third-party processes, as long as their combined types are safe (cf. Def. 4.1);
- instead, in choreographic programming, the **a**uthorisation server, the **c**lient and the **s**ervice in § 1 need to be written as a single combined "global program," that would look like G in (1). In many cases, this is not desirable or feasible.

A payoff of this restriction is that type system of [Carbone and Montesi 2013] does not encounter the difficulties of MPST systems: since processes are *not* type-checked separately against a desired interface, typing derivations like Ex. 2.7 do not arise, and this avoids the issues illustrated in § 2.3.

The limitations above are directly addressed in [Montesi and Yoshida 2013], who propose *compositional choreographies*: they develop a unified language for both choreographies and endpoint programs — and the latter are called *partial choreographies*, with the possibility of being separately developed, and reused. To this purpose, unlike [Carbone and Montesi 2013], [Montesi and Yoshida 2013] introduce parallel composition of (partial) choreographies, and communication via synchronisation (cf. *(par)* in Fig. 2, and rule [C|Sync] in Fig. 3 of [Montesi and Yoshida 2013]). This leads to the introduction of session types, and typing contexts denoted with Δ [Montesi and Yoshida 2013, p. 432], and reductions similar to our Definitions 2.6 and 2.8, and a typing rule for parallel composition (rule [T|Par] in [Montesi and Yoshida 2013, p. 432]) similar to our rule [T-|] in Fig. 2. The resulting typing derivations are similar to our Ex. 2.7, and have issues like those discussed in § 2.3, that require to constrain Δ. The subject reduction statement [Montesi and Yoshida 2013, Thm. 1] does *not* show explicit constraints on Δ — but since the paper integrates classic MPST from Honda et al. [2008] and Deniélou et al. [2012], it should either *(a)* inherit the duality/consistency issues and limitations discussed in § 3 (i.e., not supporting our opening example in § 1, nor the other examples in Fig. 4), or *(b)* implement a rather complex subject reduction proof strategy similar to the "non-classic" approach of Dezani-Ciancaglini et al. [2015] and Ghilezan et al. [2018] (discussed in § 8.2). On the positive side, we believe that our new MPST theory can be used as a drop-in replacement for classic MPST in [Montesi and Yoshida 2013]: the resulting integration would support our examples, without significant changes to the rest of their paper.

H.3 Asynchronous Subtyping

Our new asynchronous type system (even in its decidable instances) supports asynchronous protocols whose correctness depends on the capability of buffering messages, and consuming them at a later time. This means that we can use typing contexts (and type-check processes) that interact correctly under asynchrony, but would deadlock under synchronous semantics: see Ex. G.4, cases (4) and (5). This feature is not supported by the classic async MPST theory, because its a-consistency requirement (Def. E.2) only accepts types (and processes) that interact dually under *synchronous* semantics, disregarding asynchronous message buffering.

To overcome this limitation of classic MPST, [Mostrous et al. 2009] introduced an asynchronous subtyping relation $\leqslant_a$ that allows to "anticipate" outputs w.r.t. inputs: for example, if $S = \mathtt{p}\&\mathtt{m}_1.\mathtt{p}\oplus\mathtt{m}_2$ and $S' = \mathtt{p}\oplus\mathtt{m}_2.\mathtt{p}\&\mathtt{m}_1$, we have $S \leqslant_a S'$; therefore, by using $\leqslant_a$ in Fig. 2 (rule [T-Sub]) and Fig. 8, a typed asynchronous process can use an S-typed channel according to S', to first send $\mathtt{m}_2$ (that is buffered at run-time) and then receive $\mathtt{m}_1$, without causing deadlocks nor communication errors.[6] Asynchronous subtyping has been further studied (for *binary* sessions) in [Chen et al. 2017, 2014], and later discovered to be undecidable in [Bravetti et al. 2017; Lange and Yoshida 2017]. We can seamlessly integrate $\leqslant_a$ in our new MPST theory by proving that if a-safe$_{\mathbb{S}}(\Gamma)$ and $\Gamma \leqslant_a \Gamma'$, then a-safe$_{\mathbb{S}}(\Gamma')$; and to preserve Thm. G.9, we also need to prove that if a-live$^{+}_{\mathbb{S}}(\Gamma)$ and $\Gamma \leqslant_a \Gamma'$, then a-live$^{+}_{\mathbb{S}}(\Gamma')$. However, the undecidability of $\leqslant_a$ would make type checking undecidable, thus falsifying Thm. 7.2; therefore, to preserve Thm. 7.2, we could only adopt *decidable* fragments of $\leqslant_a$ — and the known ones (studied in [Bravetti et al. 2017, 2018; Lange and Yoshida 2017]) are limited to binary sessions. Hence, integrating $\leqslant_a$ in our new theory would introduce a limited gain — especially because, as explained above, our new async MPST theory *already supports* asynchronous protocols. Therefore, we decided to leave the further study of multiparty asynchronous subtyping as future work.

ADDITIONAL REFERENCES FOR THE APPENDIX

Mario Bravetti, Marco Carbone, and Gianluigi Zavattaro. 2017. Undecidability of asynchronous session subtyping. *Inf. Comput.* 256 (2017). https://doi.org/10.1016/j.ic.2017.07.010

Mario Bravetti, Marco Carbone, and Gianluigi Zavattaro. 2018. On the boundary between decidability and undecidability of asynchronous session subtyping. *Theor. Comput. Sci.* 722 (2018). https://doi.org/10.1016/j.tcs.2018.02.010

Luís Caires and Hugo Torres Vieira. 2009. Conversation Types. In *ESOP*. https://doi.org/10.1007/978-3-642-00590-9_21

Luís Caires and Hugo Torres Vieira. 2010. Conversation types. *Theoretical Computer Science* 411, 51 (2010). https://doi.org/10.1016/j.tcs.2010.09.010

Marco Carbone and Fabrizio Montesi. 2013. Deadlock-freedom-by-design: multiparty asynchronous global programming. In *POPL*. https://doi.org/10.1145/2429069.2429101

Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, and Luca Padovani. 2012. On Global Types and Multi-Party Session. *Logical Methods in Computer Science* 8, 1 (2012). https://doi.org/10.2168/LMCS-8(1:24)2012

Blaise Genest, Dietrich Kuske, and Anca Muscholl. 2007. On Communicating Automata with Bounded Channels. *Fundam. Inform.* 80, 1-3 (2007).

Ivan Lanese, Claudio Guidi, Fabrizio Montesi, and Gianluigi Zavattaro. 2008. Bridging the Gap between Interaction- and Process-Oriented Choreographies. In *Sixth IEEE International Conference on Software Engineering and Formal Methods, SEFM 2008, Cape Town, South Africa, 10-14 November 2008*. 323–332. https://doi.org/10.1109/SEFM.2008.11

Julien Lange and Nobuko Yoshida. 2017. On the Undecidability of Asynchronous Session Subtyping. In *FOSSACS*. https://doi.org/10.1007/978-3-662-54458-7_26

Fabrizio Montesi and Nobuko Yoshida. 2013. Compositional Choreographies. In *CONCUR*. https://doi.org/10.1007/978-3-642-40184-8_30

Dimitris Mostrous, Nobuko Yoshida, and Kohei Honda. 2009. Global Principal Typing in Partially Commutative Asynchronous Sessions. In *ESOP*. https://doi.org/10.1007/978-3-642-00590-9_23

Alceste Scalas, Ornela Dardha, Raymond Hu, and Nobuko Yoshida. 2017. *A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming*. Technical Report 2. Imperial College London. https://www.doc.ic.ac.uk/research/technicalreports/2017/#2

Alceste Scalas and Nobuko Yoshida. 2019a. Less is More: Multiparty Session Types Revisited. *Proc. ACM Program. Lang.* 3, POPL, Article 30 (Jan. 2019), 29 pages. https://doi.org/10.1145/3290343

[6]The asynchronous subtyping relation $\leqslant_a$ outlined here is inverted w.r.t. the one in [Mostrous et al. 2009], for the reasons explained in footnote 1.

Alceste Scalas and Nobuko Yoshida. 2019b. Less is More: Multiparty Session Types Revisited (Artifact). https://doi.org/10.1145/3291638 Peer-reviewed artifact of [Scalas and Yoshida 2019] (to appear). Latest version available at: https://alcestes.github.io/mpstk.

Alfred Tarski. 1955. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.* 5, 2 (1955).

Appendices — Part 2

Proofs

I SESSION INVERSION AND FIDELITY

Proposition I.1. *If* $P \equiv P'$, *then* $\mathrm{fc}(P) = \mathrm{fc}(P')$ *and* $\mathrm{fv}(P) = \mathrm{fv}(P')$.

Proof. By examining the cases where $P \equiv P'$ holds, and by applying the definition of $\mathrm{fc}(\cdot)$ and $\mathrm{fv}(\cdot)$. □

Proposition I.2 (Normal Form). *For all* P, $P \equiv \mathbf{def}\ \widetilde{D}\ \mathbf{in}\ (\nu\widetilde{s})\, P_1 \mid \ldots \mid P_n$, *where* $\forall i \in 1..n$, P_i *is either a branching, a selection, or a process call.*

Proof. From [Coppo et al. 2015a, Proof of Thm. 1]. □

Proposition I.3. *For all* S, $S \leqslant \mathbf{end}$ *if and only if* $\mathbf{end} \leqslant S$.

Proof. Follows by Def. 2.5. □

Lemma I.4 (Typing Inversion). *Assume* $\Theta \cdot \Gamma \vdash P$. *Then:*

(1) $P = \mathbf{0}$ *implies* $\mathrm{end}(\Gamma)$;
(2) $P = \mathbf{def}\ X(x_1{:}S_1, \ldots, x_n{:}S_n) = Q\ \mathbf{in}\ P'$ *implies:*
  (i) $\Theta \cdot x_1{:}S_1, \ldots, x_n{:}S_n \vdash Q$ *and*
  (ii) $\Theta, X{:}S_1, \ldots, S_n \cdot \Gamma \vdash P'$;
(3) $P = X\langle c_1, \ldots, c_n\rangle$ *implies:*
  (i) $\Theta \vdash X{:}S_1, \ldots, S_n$;
  (ii) $\Gamma = \Gamma_0, \Gamma_1, \ldots, \Gamma_n$;
  (iii) $\mathrm{end}(\Gamma_0)$;
  (iv) $\forall i \in 1..n :\ \Gamma_i \vdash c_i{:}S_i$;
(4) $P = (\nu s{:}G)\, P$ *implies:*
  (i) $s \notin \Gamma$;
  (ii) $\Theta \cdot \Gamma, \Gamma' \vdash P$, *for some* Γ' *such that* $\Gamma' = \{s[\mathbf{p}]{:}G{\upharpoonright}\mathbf{p}\}_{\mathbf{p} \in G}$;
(5) $P = P_1 \mid P_2$ *implies:*
  (i) $\Gamma = \Gamma_1, \Gamma_2$, *such that*
  (ii) $\Theta \cdot \Gamma_1 \vdash P_1$ *and*
  (iii) $\Theta \cdot \Gamma_2 \vdash P_2$;
(6) $P = c[\mathbf{q}] \sum_{i \in I} \mathtt{m}_i(y_i).P_i$ *implies:*
  (i) $\Gamma = \Gamma_0, \Gamma_1$ *such that*
  (ii) $\Gamma_1 \vdash c{:}\mathbf{q}\&_{i \in I}\mathtt{m}_i(S_i).S'_i$ *and*
  (iii) $\forall i \in I :\ \Theta \cdot \Gamma_0, y_i{:}S_i, c{:}S'_i \vdash P_i$;
(7) $P = c[\mathbf{q}] \oplus \mathtt{m}\langle d\rangle.P'$ *implies:*
  (i) $\Gamma = \Gamma_0, \Gamma_1, \Gamma_2$ *such that*
  (ii) $\Gamma_1 \vdash c{:}\mathbf{q}\oplus\mathtt{m}(S).S'$ *and*
  (iii) $\Gamma_2 \vdash d{:}S$ *and*
  (iv) $\Theta \cdot \Gamma_0, c{:}S' \vdash P'$.

Proof. Straightforward by the rules in Fig. 2, noticing that they are syntax-driven: i.e., for each shape of P in cases 1–7 of the statement, the typing judgement in the hypothesis can be obtained by exactly one rule. More in detail, we have:

- case 1 by rule [T-**0**];
- case 2 by rule [T-**def**];
- case 3 by rule [T-X];
- case 4 by rule [T-νClassic];
- case 5 by rule [T-$|$];
- case 6 by rule [T-&];
- case 7 by rule [T-$\oplus$].

□

Proposition I.5. *Assume* $\Theta \cdot \Gamma \vdash P$. *Then,* $\mathrm{fc}(P) \subseteq \mathrm{dom}(\Gamma)$ *and* $\forall c \in (\mathrm{dom}(\Gamma) \setminus \mathrm{fc}(P)) :\ \Gamma(c) \leqslant \mathbf{end}$.

Proof. By induction on the typing derivation of $\Theta \cdot \Gamma \vdash P$. □

Proposition I.6. *Assume* $\Theta \cdot \Gamma \vdash P$. *Then,* $\mathrm{fpv}(P) \subseteq \mathrm{dom}(\Theta)$.

Proof. By induction on the typing derivation of $\Theta \cdot \Gamma \vdash P$. □

Proposition I.7. *Assume* $\Theta \cdot \Gamma \vdash P$ *and* $\Theta \cdot \Gamma' \vdash P$. *Then:*

(1) $\forall c \in \mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Gamma') : \Gamma(c) \leqslant \Gamma'(c)$ *or* $\Gamma'(c) \leqslant \Gamma(c)$;
(2) $\forall c \in \mathrm{dom}(\Gamma) \setminus \mathrm{dom}(\Gamma') : \Gamma(c) \leqslant \mathbf{end}$;
(3) $\forall c \in \mathrm{dom}(\Gamma') \setminus \mathrm{dom}(\Gamma) : \Gamma'(c) \leqslant \mathbf{end}$;

Proof. By induction on the typing derivation of $\Theta \cdot \Gamma \vdash P$, observing the shape of P and the consequent constraints on the entries of Γ' imposed by Lemma I.4. □

Proposition I.8. *Assume* $\Theta \cdot \Gamma \vdash P$ *and* $\Theta \cdot \Gamma' \vdash P$. *Further, assume that P has guarded definitions, by Γ. Then, P has guarded definitions, by Γ'.*

Proof. By induction on the typing derivation of $\Theta \cdot \Gamma \vdash P$, applying Prop. I.7 to determine the shape of any alternative typing context Γ'. The key observation is that Γ and Γ' do not influence the types of the bound variables $x_1, \ldots, x_n$ in Def. 5.3(1). □

Proposition I.9. *Assume* $\Theta \cdot \Gamma \vdash P$ *and* $P \equiv P'$. *Further, assume that P has guarded definitions, by Γ. Then, P' has guarded definitions, by Γ.*

Proof. By cases on the definition of $\equiv$, and by applying Def. 5.3(1) — noticing that by Lemma B.2, we have $\Theta \cdot \Gamma \vdash P'$. □

Lemma I.10. *Assume* $\Theta \cdot \Gamma \vdash P$ *and* $\Theta \cdot \Gamma' \vdash P$. *Further, assume that P only plays role p in session s, by Γ. Then, P only plays role p in session s, by Γ'.*

Proof. Follows by Prop. I.8, Prop. I.7 and Def. 5.3. □

Proposition I.11. *If* $P \equiv \mathbf{0}$, *then* $\Theta \cdot \Gamma \vdash P$ *implies* $\mathrm{end}(\Gamma)$.

Proof. Assume $\Theta \cdot \Gamma \vdash P$ with $P \equiv \mathbf{0}$. By Lemma B.2, $\Theta \cdot \Gamma \vdash \mathbf{0}$; hence, by Lemma I.4(1), we conclude $\mathrm{end}(\Gamma)$. □

Lemma I.12. *Assume* $\Theta \cdot \Gamma \vdash P$ *and* $\Gamma = \Gamma_0, c{:}S$ *with* $\mathrm{end}(\Gamma_0)$ *and* $\mathsf{q}{\oplus}_{j \in J}\mathtt{m}_j(S'_j) \leqslant S$. *Further, assume that for each subterm* $(\nu s'{:}\Gamma')\,P'$ *of P, we have* $\Gamma' = s'[\mathsf{p}']{:}\mathbf{end}$ *(for some p'). Then,* $P \equiv \mathbf{def}\ \widetilde{D}\ \mathbf{in}\ (\nu\widetilde{s})\,P_1 \mid \ldots \mid P_n$, *where:*

(1) *there is* exactly one $j \in 1..n$, *such that either:*
  (a) $P_j = c[\mathsf{q}]{\oplus}\mathtt{m}_k\langle d\rangle.P'_{\mathsf{p}}$ *for some* $k \in J$, *or*
  (b) $P_j = X\langle d_1, \ldots, d_{l-1}, c, d_{l+1}, \ldots, d_m\rangle$ *for some l, m such that* $1 \le l \le m$;
(2) *for all $i \in 1..n$ and $i \neq j$ (with j from item 1 above),* $P_i = X\langle d_1, \ldots, d_m\rangle$ *for some m.*

Proof. *(Sketch)* The congruence for P holds by Prop. I.2, from which we also get:

$$\forall i \in 1..n : \ P_i \text{ is either a branching, a selection, or a process call} \qquad \text{(by Prop. I.2)} \tag{21}$$

We now prove the two claims.

Item 1. By Lemma I.4(5), $c{:}S$ can only appear in the typing context Γ_j of *exactly one* P_j, for some $j \in 1..n$; moreover, the rest of the entries of Γ_j must be (subtypes of) $\mathbf{end}$, by the hypothesis on the typing of restricted sessions, and [T-end]. Therefore, P_j *cannot* be a branching, by the contrapositive of Lemma I.4(6). Thus, by (21) and by the rules in Fig. 2, we conclude that the only possible typable shapes for P_j are either (a) (by rule [T-$\oplus$]) or (b) (by rule [T-X]). This proves the thesis.

Item 2. Observe that for all $i \in 1..n$ and $i \neq j$ (with j from item 1 above), we have $\Theta_i \cdot \Gamma_i \vdash P_i$ with $\mathrm{end}(\Gamma_i)$. By the contrapositive of Lemma I.4(6), P_i *cannot* be a branching; moreover, by the contrapositive of Lemma I.4(7), P_i *cannot* be a selection. Thus, by (21) and by the rules in Fig. 2, we conclude that the only possible typable shape for P_i is $P_i = X\langle d_1, \ldots, d_m\rangle$ (by rule [T-X]). This proves the thesis.

□

Lemma I.13. *Assume* $\Theta \cdot \Gamma \vdash P$ *and* $\Gamma = \Gamma_0, c{:}S$ *with* $\mathrm{end}(\Gamma_0)$ *and* $\mathsf{q}{\&}_{j \in J}\mathtt{m}_j(S'_j) \leqslant S$. *Further, assume that for each subterm* $(\nu s'{:}\Gamma')\,P'$ *of P, we have* $\Gamma' = s'[\mathsf{p}']{:}\mathbf{end}$ *(for some p'). Then,* $P \equiv \mathbf{def}\ \widetilde{D}\ \mathbf{in}\ (\nu\widetilde{s})\,P_1 \mid \ldots \mid P_n$, *where:*

(1) *there is* exactly one $j \in 1..n$, *such that either:*
  (a) $P_j = c[\mathsf{q}]\sum_{i \in K}\mathtt{m}_i(d).P'_{\mathsf{p}i}$ *for some* $K \supseteq J$, *or*
  (b) $P_j = X\langle d_1, \ldots, d_{l-1}, c, d_{l+1}, \ldots, d_m\rangle$ *for some l, m such that* $1 \le l \le m$;
(2) *for all $i \in 1..n$ and $i \neq j$ (with j from item 1 above),* $P_i = X\langle d_1, \ldots, d_m\rangle$ *for some m.*

Proof. *(Sketch)* The congruence for P holds by Prop. I.2, from which we also get:

$$\forall i \in 1..n: \; P_i \text{ is either a branching, a selection, or a process call} \qquad \text{(by Prop. I.2)} \tag{22}$$

We now prove the two claims.

Item 1. By Lemma I.4(5), $c{:}S$ can only appear in the typing context Γ_j of *exactly one* P_j, for some $j \in 1..n$; moreover, the rest of the entries of Γ_j must be (subtypes of) **end**, by the hypothesis on the typing of restricted sessions, and [T-end]. Therefore, P_j *cannot* be a selection, by the contrapositive of Lemma I.4(7). Thus, by (22) and by the rules in Fig. 2, we conclude that the only possible typable shapes for P_j are either (a) (by rule [T-&]) or (b) (by rule [T-X]). This proves the thesis.

Item 2. Observe that for all $i \in 1..n$ and $i \neq j$ (with j from item 1 above), we have $\Theta_i \cdot \Gamma_i \vdash P_i$ with $\text{end}(\Gamma_i)$. By the contrapositive of Lemma I.4(7), P_i *cannot* be a selection; moreover, by the contrapositive of Lemma I.4(6), P_i *cannot* be a branching. Thus, by (22) and by the rules in Fig. 2, we conclude that the only possible typable shape for P_i is $P_i = X\langle d_1, \ldots, d_m\rangle$ (by rule [T-X]). This proves the thesis.

□

Theorem B.4 (Session Inversion). *Assume* $\emptyset \cdot \Gamma \vdash |_{\mathsf{p}\in I} P_{\mathsf{p}}$ *with each* P_{p} *either being* $\mathbf{0}$ *(up-to* $\equiv$*), or only playing role* p *in* s*. Then,* $\Gamma = \Gamma_0, \{s[\mathsf{p}]{:}S_{\mathsf{p}}\}_{\mathsf{p}\in I'}$ *(for some I') with* $\text{end}(\Gamma_0)$*. Moreover,* $\forall \mathsf{p} \in I'$*:*

(1) if $\mathsf{q}\oplus_{j\in J}\mathsf{m}_j(S'_j).S'_j \leqslant S_{\mathsf{p}}$ *then* $\mathsf{p} \in I$ *and for some* $\mathbb{C}, \mathbb{C}'$*, and* $k \in J$*, either:*

(a) $P_{\mathsf{p}} \equiv \mathbb{C}\big[s[\mathsf{p}][\mathsf{q}]\oplus\mathsf{m}_k\langle s'[\mathsf{r}]\rangle.P'_{\mathsf{p}}\big]$ *or*

(b) $P_{\mathsf{p}} \equiv \mathbb{C}\left[\begin{array}{l}\textbf{def } X(x_1{:}T_1,\ldots,x_n{:}T_n) = \mathbb{C}'\big[x_l[\mathsf{q}]\oplus\mathsf{m}_k\langle d\rangle.P'_{\mathsf{p}}\big] \textbf{ in} \\ \quad X\Big\langle s'_1[\mathsf{r}_1],\ldots,s'_{l-1}[\mathsf{r}_{l-1}],\, s[\mathsf{p}],\, s'_{l+1}[\mathsf{r}_{l+1}],\ldots,s'_n[\mathsf{r}_n]\Big\rangle\end{array}\right]$ *with* $1 \le l \le n$*;*

(2) if $\mathsf{q}\&_{j\in J}\mathsf{m}_j(S'_j).S'_j \leqslant S_{\mathsf{p}}$ *then* $\mathsf{p} \in I$ *and for some* $\mathbb{C}, \mathbb{C}'$*, and* $K \supseteq J$*, either:*

(a) $P_{\mathsf{p}} \equiv \mathbb{C}\Big[s[\mathsf{p}][\mathsf{q}]\sum_{k\in K}\mathsf{m}_k(x_k).P'_{\mathsf{p}k}\Big]$ *or*

(b) $P_{\mathsf{p}} \equiv \mathbb{C}\left[\begin{array}{l}\textbf{def } X(x_1{:}T_1,\ldots,x_n{:}T_n) = \mathbb{C}'\Big[x_l[\mathsf{q}]\sum_{k\in K}\mathsf{m}_k(x_k).P'_{\mathsf{p}k}\Big] \textbf{ in} \\ \quad X\Big\langle s'_1[\mathsf{r}_1],\ldots,s'_{l-1}[\mathsf{r}_{l-1}],\, s[\mathsf{p}],\, s'_{l+1}[\mathsf{r}_{l+1}],\ldots,s'_n[\mathsf{r}_n]\Big\rangle\end{array}\right]$ *with* $1 \le l \le n$*;*

(3) if $\textbf{end} \leqslant S_{\mathsf{p}}$ *then* $\mathsf{p} \in I$ *implies* $P_{\mathsf{p}} \equiv \mathbf{0}$*.*

Further, (4) $\forall \mathsf{p} \in I \setminus I' : P_{\mathsf{p}} \equiv \mathbf{0}$*.*

Proof. Assume the hypotheses:

$$\exists I: \; \emptyset \cdot \Gamma \vdash |_{\mathsf{p}\in I} P_{\mathsf{p}} \tag{23}$$

$$\forall \mathsf{p} \in I, \text{ either:} \tag{24}$$

$$P_{\mathsf{p}} \equiv \mathbf{0} \qquad \text{or} \tag{25}$$

$$P_{\mathsf{p}} \text{ only plays role } \mathsf{p} \text{ in } s \text{ (Def. 5.3)} \tag{26}$$

We observe:

$$\begin{array}{l}\Gamma = \Gamma_{\mathsf{p}_1},\ldots,\Gamma_{\mathsf{p}_n} \text{ such that} \\ \qquad I = \{\mathsf{p}_1,\ldots,\mathsf{p}_n\} \text{ and } \forall \mathsf{p} \in I: \; \emptyset \cdot \Gamma_{\mathsf{p}} \vdash P_{\mathsf{p}}\end{array} \qquad \text{(by (23) and Lemma I.4(5))} \tag{27}$$

$$\begin{array}{l}\text{if } P_{\mathsf{p}} \text{ only plays } \mathsf{p} \text{ in } s, \text{ then } s[\mathsf{p}] \in \text{fc}(P_{\mathsf{p}}) \text{ and } \text{fv}(P_{\mathsf{p}}) = \emptyset \\ \qquad \Gamma_{\mathsf{p}} = \Gamma'_{\mathsf{p}}, s[\mathsf{p}]{:}S_{\mathsf{p}} \text{ with } \text{end}(\Gamma'_{\mathsf{p}}) \text{ and } S_{\mathsf{p}} \not\leqslant \textbf{end}\end{array} \qquad \left(\begin{array}{l}\text{by Lemma I.10, Def. 5.3,} \\ \text{Prop. I.5, and (23)}\end{array}\right) \tag{28}$$

Notice that:

$$\exists I', \Gamma_0: \quad \Gamma = \Gamma_0, \{s[\mathsf{p}]{:}S_{\mathsf{p}}\}_{\mathsf{p}\in I'} \quad \text{and} \quad \text{end}(\Gamma_0) \qquad \text{by (27) and (28)} \tag{29}$$

and (29) satisfies the first claim in the statement. We now prove the remaining claims.

Item (1). Take any $\mathsf{p} \in I'$ such that $\mathsf{q}\oplus_{j\in J}\mathsf{m}_j(S'_j).S'_j \leqslant S_{\mathsf{p}}$. We first prove that $\mathsf{p} \in I$. By contradiction, assume $\mathsf{p} \notin I$ (i.e., none of the processes plays only role p in s): then, by (25), (26) and (28), we obtain that $s[\mathsf{p}] \notin \text{fc}\big(|_{\mathsf{p}\in I} P_{\mathsf{p}}\big)$, which (by Prop. I.5) implies $S_{\mathsf{p}} \leqslant \textbf{end}$. But then, by Def. 2.5, we obtain $\mathsf{q}\oplus_{j\in J}\mathsf{m}_j(S'_j).S'_j \not\leqslant S_{\mathsf{p}}$ — contradiction. Hence, we have proved $\mathsf{p} \in I$. We now examine the two possible cases for P_{p}:

- $P_{\mathsf{p}} \equiv \mathbf{0}$ (by (25)). We show that this case is absurd. By contradiction, assume $P_{\mathsf{p}} \equiv \mathbf{0}$. Observe that $\forall \mathsf{r} \in I$ such that $\mathsf{r} \neq \mathsf{p}$, $s[\mathsf{p}] \notin \text{fc}(P_{\mathsf{r}})$ (by (25), (26) and (28)), which (by Def. 2.5) implies $S_{\mathsf{p}} \leqslant \textbf{end}$. But then, by Def. 2.5, we obtain $\mathsf{q}\oplus_{j\in J}\mathsf{m}_j(S'_j).S'_j \not\leqslant S_{\mathsf{p}}$ — contradiction. Hence, we have proved $P_{\mathsf{p}} \not\equiv \mathbf{0}$;

- P_{p} only plays role p in s (by (26)). Then, we apply Lemma I.12, and we have two sub-cases. If Lemma I.12(1)(a) holds, we observe that d therein cannot be a variable (otherwise we would have $\text{fv}(P_{\mathsf{p}}) \neq \emptyset$, i.e., by Def. 5.3, P_{p} does *not* only play p in s — contradiction), and therefore $d = s'[\mathsf{r}]$ (for some s', r), from which we conclude by obtaining case (a) of the statement with the following context:
$$\mathbb{C} \quad = \quad \textbf{def } \widetilde{D} \textbf{ in } (\nu\tilde{s})\ [\,]\mid Q_2 \mid \ldots \mid Q_n \tag{30}$$
Otherwise, if Lemma I.12(1)(b) holds, we observe:
  1. by (23) and Prop.I.6, $X \notin \text{fpv}(P_{\mathsf{p}})$, and therefore, by Prop.I.2, P_{p} has the form:
  $$P_{\mathsf{p}} \;\equiv\; \textbf{def } \widetilde{D_1} \textbf{ in def } X(x_1{:}T_1,\ldots,x_m{:}T_m) = Q \textbf{ in def } \widetilde{D_2} \textbf{ in } \left(\nu\widetilde{s'}\right) X\langle d_1,\ldots,d_n\rangle \tag{31}$$
  2. hence, we can use the congruence $\equiv$ to remove unused process declarations $\widetilde{D_2}$ in (31), and place the definition of X immediately before the call:
  $$P_{\mathsf{p}} \;\equiv\; \textbf{def } \widetilde{D_1} \textbf{ in } \left(\nu\widetilde{s'}\right) \textbf{def } X(x_1{:}T_1,\ldots,x_m{:}T_m) = Q \textbf{ in } X\langle d_1,\ldots,d_n\rangle \tag{32}$$
  3. from (32), by Lemma I.4 and Lemma B.2, and observing that $d_1,\ldots,d_n$ cannot be variables (since $\text{fv}(P_{\mathsf{p}}) = \emptyset$, by Def. 5.3 and Prop.I.1), we get $\Gamma_{\mathsf{p}} = \Gamma_{\mathsf{p}0},\ldots,\Gamma_{\mathsf{p}m}$ such that (note that the following derivation applies [T-**def**] once for each process declaration in $\widetilde{D_1}$, yielding the sequence of process typings $\widetilde{X'{:}\widetilde{T'}}$):
  $$\dfrac{\dfrac{\cdots \qquad \dfrac{\widetilde{X'{:}\widetilde{T'}} \cdot \Gamma_{\mathsf{p}}, x_1{:}T_1,\ldots,x_m{:}T_m \vdash Q \qquad \dfrac{\begin{array}{c}\Theta, \widetilde{X'{:}\widetilde{T'}}, X{:}T_1,\ldots,T_n \vdash X{:}T_1,\ldots,T_m \\ \text{end}(\Gamma_{\mathsf{p}0}) \quad \forall i \in 1..m \quad \Gamma_{\mathsf{p}i} \vdash s'_i[\mathsf{r}_i]{:}T_i\end{array}}{\widetilde{X'{:}\widetilde{T'}}, X{:}T_1,\ldots,T_n \cdot \Gamma_{\mathsf{p}}, \vdash X\left\langle s'_1[\mathsf{r}_1],\ldots,s'_n[\mathsf{r}_n]\right\rangle}\,[\text{T-}X]}{\widetilde{X'{:}\widetilde{T'}} \cdot \Gamma_{\mathsf{p}} \vdash \textbf{def } X(x_1{:}T_1,\ldots,x_m{:}T_m) = Q \textbf{ in } X\left\langle s'_1[\mathsf{r}_1],\ldots,s'_n[\mathsf{r}_n]\right\rangle}\,[\text{T-}\textbf{def}]}{\vdots}\,\left[\text{T-}\textbf{def} \times \left|\text{dpv}\left(\widetilde{D_1}\right)\right|\right]}{\emptyset \cdot \Gamma_{\mathsf{p}} \vdash P_{\mathsf{p}}} \tag{33}$$
  From (33), we get:
  $$m = n \tag{34}$$
  $$\exists l : \quad s'_l[\mathsf{r}_l] = s[\mathsf{p}] \tag{35}$$
  $$\forall i \in 1..n : \quad i \neq l \text{ implies } \textbf{end} \leqslant T_i \qquad \text{by Def. 5.3(2)}\textit{(iv)} \tag{36}$$
  $$\mathsf{q}{\oplus}_{j\in J}\mathtt{m}_j(S'_j).S'_j \leqslant S_{\mathsf{p}} \leqslant T_l \qquad (\text{since } \Gamma_{\mathsf{p}l} \vdash d_l{:}T_l\text{, and by transitivity of } \leqslant) \tag{37}$$
  4. by Prop.I.2 we have:
  $$Q \;\equiv\; \textbf{def } \widetilde{D''} \textbf{ in } \left(\nu\widetilde{s''}\right) Q_1 \mid \ldots \mid Q_{n''} \quad \text{where } \forall i \in 1..n'' : Q_i \text{ is either a branching, a selection, or a process call} \tag{38}$$
  $$\widetilde{X'{:}\widetilde{T'}} \cdot \Gamma_{\mathsf{p}}, x_1{:}T_1,\ldots,x_l{:}T_l,\ldots,x_m{:}T_m \vdash Q \qquad (\text{by } (33)) \tag{39}$$
  5. since P_{p} has guarded definitions (by Def. 5.3(2), then by (38), Prop.I.9 and (39) we know that in $Q_1 \mid \ldots \mid Q_{n''}$ (from (38)) some branching or selection uses x_l, before further process calls. Without loss of generality, assume that Q_1 satisfies the requirement;
  6. by (39) and the contrapositive of Lemma I.4(6), Q_1 cannot use x_l for branching;
  7. hence, by rule [T-$\oplus$], we conclude that Q_1 is a selection on x_l;
  8. therefore, by Lemma I.12(1)(a), we obtain:
  $$Q_1 \;=\; x_l[\mathsf{q}]{\oplus}\mathtt{m}_k\langle d\rangle.P'_{\mathsf{p}} \qquad \text{for some } k \in J \tag{40}$$

  Summing up, from (32), (34), (35), (38) and (40) we have the following reduction contexts, as required by case (b) of the statement:
  $$\begin{array}{rcl}\mathbb{C} & = & \textbf{def } \widetilde{D_1} \textbf{ in } \left(\nu\widetilde{s'}\right)\left(\begin{array}{l}\textbf{def } X(x_1{:}T_1,\ldots,x_n{:}T_n) = \mathbb{C}' \textbf{ in} \\ \quad X\left\langle s'_1[\mathsf{r}_1],\ldots,s'_{l-1}[\mathsf{r}_{l-1}],\ s[\mathsf{p}],\ s'_{l+1}[\mathsf{r}_{l+1}],\ldots,s'_n[\mathsf{r}_n]\right\rangle\end{array}\right) \\ \mathbb{C}' & = & \textbf{def } \widetilde{D''} \textbf{ in } \left(\nu\widetilde{s''}\right)\ [\,]\mid Q_2 \mid \ldots \mid Q_{n''}\end{array} \tag{41}$$

and this concludes the proof.

Item (2). The proof is similar to that for item (2) above, except that we use Lemma I.13 instead of Lemma I.12, obtaining the same reduction contexts: either (30) (thus obtaining case (a) of the steatement) or (41) (thus obtaining case (b) of the statement).

Item (3). If $I \cap I' = \emptyset$, the statement holds vacuously. Otherwise, take any $\mathsf{p} \in I \cap I'$, and assume $\mathbf{end} \leqslant S_{\mathsf{p}} = \Gamma(s[\mathsf{p}])$. We have two cases:

- $P_{\mathsf{p}} \equiv \mathbf{0}$ (by (25)). This is the thesis;
- P_{p} only plays role p in s (by (26)). This case is impossible: otherwise, by (28) we would get $S_{\mathsf{p}} \not\leqslant \mathbf{end}$, and thus $\mathbf{end} \not\leqslant S_{\mathsf{p}}$ (by Prop. I.3) — which would contradict the assumption $\mathbf{end} \leqslant S_{\mathsf{p}}$.

Item (4). If $I \setminus I' = \emptyset$, the statement holds vacuously. Otherwise, take any $\mathsf{p} \in I \setminus I'$, which implies $\mathsf{p} \notin I'$. We have two cases:

- $P_{\mathsf{p}} \equiv \mathbf{0}$ (by (25)). This is the thesis;
- P_{p} only plays role p in s (by (26)). This case is impossible: otherwise, by (28) we would get $\Gamma_{\mathsf{p}} = \Gamma'_{\mathsf{p}}, s[\mathsf{p}]{:}S_{\mathsf{p}}$ and therefore $s[\mathsf{p}] \in \mathrm{dom}(\Gamma)$ (by (27)), that means $\mathsf{p} \in I'$ (by (29)) — which would contradict the assumption $\mathsf{p} \in I \setminus I'$.

□

Definition I.14 (Type Unfolding). The *one-step unfolding of a type S*, written $\mathrm{unf}(S)$, is:

$$\mathrm{unf}(\mu \mathbf{t}.T) = T\{\mu\mathbf{t}.T/\mathbf{t}\} \qquad \mathrm{unf}(T) = T \quad \text{if } T \neq \mu\mathbf{t}.T'$$

The *n-steps unfolding of a type S*, written $\mathrm{unf}^{\mathrm{n}}(S)$, is:

$$\mathrm{unf}^{0}(T) = T \qquad \mathrm{unf}^{\mathrm{m}+1}(T) = \mathrm{unf}\big(\mathrm{unf}^{\mathrm{m}}(T)\big)$$

The *complete unfolding of a session type S*, written $\mathrm{unf}^{*}(S)$, is defined as:

$$\mathrm{unf}^{*}(S) \;=\; \mathrm{unf}^{\mathrm{n}}(S) \qquad \text{for the smallest } n \text{ such that } \mathrm{unf}^{\mathrm{n}}(S) = \mathrm{unf}^{\mathrm{n}+1}(S)$$

Proposition I.15. *For all S, T:*

(1) $S \leqslant T$ if and only if $\mathrm{unf}(S) \leqslant T$, and
(2) $S \leqslant T$ if and only if $S \leqslant \mathrm{unf}(T)$.

Proof. Item 1.

- ($\Longrightarrow$) Assume $S \leqslant T$. If $S \neq \mu\mathbf{t}.S'$, then $S = \mathrm{unf}(S) \leqslant T$, and we obtain the thesis. Otherwise, by Def. I.14, we have $\mathrm{unf}(S) = S'\{\mu\mathbf{t}.S'/\mathbf{t}\}$, and we conclude by the coinductive rule [Sub-μL] of Def. 2.5.
- ($\Longleftarrow$) Assume $\mathrm{unf}(S) \leqslant T$. If $S \neq \mu\mathbf{t}.S'$, then $\mathrm{unf}(S) = S \leqslant T$, and we obtain the thesis. Otherwise, by Def. I.14 we have $S'\{\mu\mathbf{t}.S'/\mathbf{t}\} \leqslant T$, and since $\leqslant$ is the *largest* relation closed backward under the coinductive rule [Sub-μL] of Def. 2.5, we conclude $\mu\mathbf{t}.S' = S \leqslant T$.

Item 2. The proofs for the two the implications in the statement are similar to the corresponding proofs for item 1, but using the coinductive rule [Sub-μR] of Def. 2.5 (instead of [Sub-μL]). □

Proposition I.16. *For all S, T:*

(1) $S \leqslant T$ if and only if $\mathrm{unf}^{*}(S) \leqslant T$, and
(2) $S \leqslant T$ if and only if $S \leqslant \mathrm{unf}^{*}(T)$.

Proof. Take the smallest n such that $\mathrm{unf}^{*}(S) = \mathrm{unf}^{\mathrm{n}}(S)$ (by Def. I.14).

- Item 1 By induction on n, applying Prop. I.15(1) in the inductive case.
- Item 2 By induction on n, applying Prop. I.15(2) in the inductive case.

□

Proposition I.17. $s[\mathsf{p}]{:}S, s[\mathsf{q}]{:}T \;\to\; \Gamma'$ *if and only if* $s[\mathsf{p}]{:}\mathrm{unf}(S), s[\mathsf{q}]{:}T \;\to\; \Gamma'$.

Proof. ($\Longrightarrow$). If $S \neq \mu\mathbf{t}.S'$, then $S = \mathrm{unf}(S)$, and the statement holds vacuously. Otherwise, we conclude by Def. I.14 and Def. 2.8 (rule [Γ-Comm]).

($\Longleftarrow$). If $S \neq \mu\mathbf{t}.S'$, then $S = \mathrm{unf}(S)$, and the statement holds vacuously. Otherwise, we conclude by Def. I.14 and inversion of rule [Γ-Comm] in Def. 2.8. □

Proposition I.18. $s[\mathsf{p}]{:}S, s[\mathsf{q}]{:}T \;\to\; \Gamma'$ *if and only if* $s[\mathsf{p}]{:}\mathrm{unf}^{*}(S), s[\mathsf{q}]{:}T \;\to\; \Gamma'$.

Proof. Take the smallest n such that $\mathrm{unf}^*(S) = \mathrm{unf}^n(S)$ (by Def. I.14). The statement is proved by induction on n, applying Prop. I.17 in the inductive case. □

Lemma I.19. *Assume* $s[\mathbf{p}]{:}S, s[\mathbf{q}]{:}T\!\rightarrow$. *Moreover, assume:*

$$\Gamma \;=\; \Gamma_0, s[\mathbf{p}]{:}S, s[\mathbf{q}]{:}T \;\leqslant\; \Gamma' \;=\; \Gamma'_0, s[\mathbf{p}]{:}S', s[\mathbf{q}]{:}T' \;\rightarrow\; \Gamma'' \;=\; \Gamma'_0,\, s[\mathbf{p}]{:}S'',\, s[\mathbf{q}]{:}T''$$

Then, there is Γ''' *such that* $\Gamma \rightarrow \Gamma''' \leqslant \Gamma''$.

Proof. Assume:

$$s[\mathbf{p}]{:}S, s[\mathbf{q}]{:}T\!\rightarrow \tag{42}$$
$$\Gamma \leqslant \Gamma' \tag{43}$$
$$\Gamma' \rightarrow \Gamma'' \tag{44}$$

We have:

$$\begin{array}{ll} \Gamma' \;=\; \Gamma'_0, s[\mathbf{p}]{:}S', s[\mathbf{q}]{:}T' & \\ \quad \text{with } \mathrm{unf}^*(S') = \mathbf{q}\oplus_{i\in I'}\mathtt{m}_i(S''_i).S'''_i & \\ \quad \text{and } \mathrm{unf}^*(T') = \mathbf{p}\&_{j\in J'}\mathtt{m}_j(T''_j).T'''_j & \text{(by (44), Def. 2.8, Prop. I.18)} \\ \quad \text{and } I' \subseteq J' \text{ and } \forall i\in I' : S''_i \leqslant T''_i & \end{array} \tag{45}$$

$$\begin{array}{ll} \Gamma'' \;=\; \Gamma'_0,\, s[\mathbf{p}]{:}S'''_k,\, s[\mathbf{q}]{:}T'''_k & \\ \quad \text{with } k \in I' \subseteq J' & \text{(by (44), (45), Prop. I.18)} \end{array} \tag{46}$$

$$\begin{array}{ll} \Gamma \;=\; \Gamma_0, s[\mathbf{p}]{:}S, s[\mathbf{q}]{:}T & \\ \quad \text{with } \Gamma_0 \leqslant \Gamma'_0 & \\ \quad \text{and } \mathrm{unf}^*(S) = \mathbf{q}\oplus_{i\in I}\mathtt{m}_i(S_i).S'_i & \\ \quad \text{and } \mathrm{unf}^*(T') = \mathbf{p}\&_{j\in J}\mathtt{m}_j(T_j).T'_j & \\ \quad \text{and } I' \subseteq I \text{ and } J \subseteq J' & \text{(by (43), (45), Prop. I.16, Def. 2.5)} \\ \quad \text{and } \forall i\in I' : \begin{cases} S''_i \leqslant S_i \\ S'_i \leqslant S'''_i \end{cases} & \\ \quad \text{and } \forall j\in J : \begin{cases} T''_j \leqslant T_j \\ T'_j \leqslant T'''_j \end{cases} & \end{array} \tag{47}$$

$$k \in I \subseteq J \text{ and } \forall i\in I : S_i \leqslant T_i \qquad \text{(by (46), (47), (42) and Def. 2.8)} \tag{48}$$

Now, let:

$$\Gamma''' \;=\; \Gamma_0, s[\mathbf{p}]{:}S'_k, s[\mathbf{q}]{:}T'_k \tag{49}$$

We conclude:

$$\Gamma' \rightarrow \Gamma''' \qquad \text{(by (47), (49), (48), Def. 2.8, Prop. I.18)} \tag{50}$$
$$\Gamma''' \leqslant \Gamma'' \qquad \text{(by (47), (49), (48), Def. 2.6)} \tag{51}$$

□

Lemma B.3 (Narrowing). *If* $\Theta \cdot \Gamma \vdash P$ *and* $\Gamma' \leqslant \Gamma$, *then* $\Theta \cdot \Gamma' \vdash P$.

Proof. By induction on the derivation of $\Theta \cdot \Gamma \vdash P$, and by Def. 2.5. □

Theorem 5.4 (Session Fidelity). *Assume* $\emptyset \cdot \Gamma \vdash P$, *where* Γ *is safe,* $P \equiv \left|_{\mathbf{p}\in I} P_{\mathbf{p}}\right.$, *and* $\Gamma = \bigcup_{\mathbf{p}\in I}\Gamma_{\mathbf{p}}$ *such that, for each* $P_{\mathbf{p}}$, *we have* $\emptyset \cdot \Gamma_{\mathbf{p}} \vdash P_{\mathbf{p}}$; *further, assume that each* $P_{\mathbf{p}}$ *is either* $\mathbf{0}$ *(up-to* $\equiv$*), or only plays* $\mathbf{p}$ *in* s, *by* $\Gamma_{\mathbf{p}}$. *Then,* $\Gamma\!\rightarrow$ *implies* $\exists \Gamma', P'$ *such that* $\Gamma \rightarrow \Gamma'$, $P \rightarrow^* P'$ *and* $\emptyset \cdot \Gamma' \vdash P'$, *with* Γ' *safe,* $P' \equiv \left|_{\mathbf{p}\in I} P'_{\mathbf{p}}\right.$, *and* $\Gamma' = \bigcup_{\mathbf{p}\in I}\Gamma'_{\mathbf{p}}$ *such that, for each* $P'_{\mathbf{p}}$, *we have* $\emptyset \cdot \Gamma'_{\mathbf{p}} \vdash P'_{\mathbf{p}}$, *and each* $P'_{\mathbf{p}}$ *is either* $\mathbf{0}$ *(up-to* $\equiv$*), or only plays* $\mathbf{p}$ *in* s, *by* $\Gamma'_{\mathbf{p}}$.

Proof. We have:

$$\mathrm{safe}(\Gamma) \qquad \text{(by hypothesis)} \tag{52}$$
$$\Gamma\!\rightarrow \qquad \text{(by hypothesis)} \tag{53}$$
$$\Gamma \;\; (\leqslant \cap \geqslant) \;\; \Gamma_0, s[\mathbf{p}]{:}S, s[\mathbf{q}]{:}T \quad \text{where} \begin{cases} S = \mathbf{q}\oplus_{i\in I}\mathtt{m}_i(S_i).S'_i \quad \text{and} \\ T = \mathbf{p}\&_{j\in J}\mathtt{m}_j(S''_j).S'''_j \quad \text{and} \\ I \subseteq J \\ \forall i\in I : S_i \leqslant S''_i \end{cases} \qquad \left(\begin{array}{l}\text{by (52), (53),} \\ \text{Def. 2.8,} \\ \text{Prop. I.18,} \\ \text{Prop. I.16}\end{array}\right) \tag{54}$$

By (54) and Thm. B.4, we have one of the following cases, where the two processes with reduction contexts $\mathbb{C}_{\mathbf{p}}$ and $\mathbb{C}_{\mathbf{q}}$ play respectively only roles $\mathbf{p}$ and $\mathbf{q}$ in s, and $k \in I \subseteq J \subseteq L$:

(a) $P \equiv \mathbb{C}_{\mathbf{p}}\left[s[\mathbf{p}][\mathbf{q}]\oplus\mathtt{m}_k\langle s'[\mathbf{r}]\rangle.P'_{\mathbf{p}}\right] \mid \mathbb{C}_{\mathbf{q}}\left[s[\mathbf{q}][\mathbf{p}]\sum_{l\in L}\mathtt{m}_l(x_l).P'_{\mathbf{q}l}\right] \mid Q$

(b) $P \equiv \mathbb{C}_{\mathbf{p}}\left[\begin{array}{l}\mathbf{def}\ Y(y_1{:}T_1,\ldots,y_m{:}T_m) = \mathbb{C}'_{\mathbf{p}}\left[y_{i'}[\mathbf{q}]\oplus\mathtt{m}_k\langle d\rangle.P'_{\mathbf{p}}\right]\ \mathbf{in}\\ Y\langle s'_1[\mathbf{r}_1],\ldots,s'_{i'-1}[\mathbf{r}_{i'-1}],\ s[\mathbf{p}],\ s'_{i'+1}[\mathbf{r}_{i'+1}],\ldots,s'_m[\mathbf{r}_m]\rangle\end{array}\right] \mid \mathbb{C}_{\mathbf{q}}\left[s[\mathbf{q}][\mathbf{p}]\sum_{l\in L}\mathtt{m}_l(x_l).P'_{\mathbf{q}l}\right] \mid Q$

(c) $P \equiv \mathbb{C}_{\mathbf{p}}\left[s[\mathbf{p}][\mathbf{q}]\oplus\mathtt{m}_k\langle s'[\mathbf{r}]\rangle.P'_{\mathbf{p}}\right] \mid \mathbb{C}_{\mathbf{q}}\left[\begin{array}{l}\mathbf{def}\ Z(z_1{:}T'_1,\ldots,z_n{:}T'_n) = \mathbb{C}'_{\mathbf{q}}\left[z_{j'}[\mathbf{p}]\sum_{l\in L}\mathtt{m}_l(x_l).P'_{\mathbf{q}l}\right]\ \mathbf{in}\\ Z\left\langle s''_1[\mathbf{r}'_1],\ldots,s''_{j'-1}[\mathbf{r}'_{j'-1}],\ s[\mathbf{q}],\ s''_{j'+1}[\mathbf{r}'_{j'+1}],\ldots,s''_n[\mathbf{r}'_n]\right\rangle\end{array}\right] \mid Q$

(d)
$$\begin{array}{rcl} P & \equiv & \mathbb{C}_{\mathbf{p}}\left[\begin{array}{l}\mathbf{def}\ Y(y_1{:}T_1,\ldots,y_m{:}T_m) = \mathbb{C}'_{\mathbf{p}}\left[y_{i'}[\mathbf{q}]\oplus\mathtt{m}_k\langle d\rangle.P'_{\mathbf{p}}\right]\ \mathbf{in}\\ \quad Y\langle s'_1[\mathbf{r}_1],\ldots,s'_{i'-1}[\mathbf{r}_{i'-1}],\ s[\mathbf{p}],\ s'_{i'+1}[\mathbf{r}_{i'+1}],\ldots,s'_m[\mathbf{r}_m]\rangle\end{array}\right]\\ & & \mid \mathbb{C}_{\mathbf{q}}\left[\begin{array}{l}\mathbf{def}\ Z(z_1{:}T'_1,\ldots,z_n{:}T'_n) = \mathbb{C}'_{\mathbf{q}}\left[z_{j'}[\mathbf{p}]\sum_{l\in L}\mathtt{m}_l(x_l).P'_{\mathbf{q}l}\right]\ \mathbf{in}\\ \quad Z\left\langle s''_1[\mathbf{r}'_1],\ldots,s''_{j'-1}[\mathbf{r}'_{j'-1}],\ s[\mathbf{q}],\ s''_{j'+1}[\mathbf{r}'_{j'+1}],\ldots,s''_n[\mathbf{r}'_n]\right\rangle\end{array}\right] \mid Q\end{array}$$

We develop the proof for case (b) (the other cases are similar).
By induction on $\mathbb{C}_{\mathbf{p}}$ and $\mathbb{C}'_{\mathbf{p}}$, using Lemma I.4, we can prove that the term inside $\mathbb{C}_{\mathbf{p}}$ is typed as:

$$\dfrac{\dfrac{\dfrac{\begin{array}{l} y_{i'}{:}T_{i'} \vdash y_{i'}{:}\mathbf{q}\oplus\mathtt{m}_k(T''_k).T'''_k \quad \Gamma'_{\mathbf{p}d} \vdash d{:}T''_k\\ \Theta_{\mathbf{p}},\Theta'_{\mathbf{p}},Y{:}T_1,\ldots,T_m \cdot \Gamma'_{\mathbf{p}0}, y_{i'}{:}T'''_k \vdash P'_{\mathbf{p}}\end{array}}{\begin{array}{l}\Theta_{\mathbf{p}},\Theta'_{\mathbf{p}},Y{:}T_1,\ldots,T_m\\ \quad\cdot\,\Gamma'_{\mathbf{p}0},\Gamma'_{\mathbf{p}d},y_{i'}{:}T_{i'}\end{array} \vdash y_{i'}[\mathbf{q}]\oplus\mathtt{m}_k\langle d\rangle.P'_{\mathbf{p}}}\,[\text{T-}\oplus]}{\vdots}}{\begin{array}{l}\Theta_{\mathbf{p}},Y{:}T_1,\ldots,T_m\\ \cdot\,y_1{:}T_1,\ldots y_m{:}T_m\end{array} \vdash \mathbb{C}'_{\mathbf{p}}\left[y_{i'}[\mathbf{q}]\oplus\mathtt{m}_k\langle d\rangle.P'_{\mathbf{p}}\right]} \quad \dfrac{\begin{array}{l}\mathrm{end}(\Gamma_0) \quad \Gamma_{\mathbf{p}i'} \vdash s[\mathbf{p}]{:}T_{i'}\\ \forall i \in 1..(i'-1),(i'+1)..m \quad \Gamma_{\mathbf{p}i} \vdash s'_i[\mathbf{r}_i]{:}T_i\end{array}}{\begin{array}{l}\Theta_{\mathbf{p}},Y{:}T_1,\ldots,T_m\\ \quad\cdot\,\Gamma_{\mathbf{p}0},\ldots,\Gamma_{\mathbf{p}m}\end{array} \vdash Y\left\langle\begin{array}{l}s'_1[\mathbf{r}_1],\ldots,s'_{i'-1}[\mathbf{r}_{i'-1}],\ s[\mathbf{p}],\\ s'_{i'+1}[\mathbf{r}_{i'+1}],\ldots,s'_m[\mathbf{r}_m]\end{array}\right\rangle}\,[\text{T-}X]$$
$$\overline{\Theta_{\mathbf{p}} \cdot \Gamma_{\mathbf{p}0},\Gamma_{\mathbf{p}1},\ldots,\Gamma_{\mathbf{p}i'},\ldots,\Gamma_{\mathbf{p}m} \vdash \begin{array}{l}\mathbf{def}\ Y(y_1{:}T_1,\ldots,y_m{:}T_m) = \mathbb{C}'_{\mathbf{p}}\left[y_{i'}[\mathbf{q}]\oplus\mathtt{m}_k\langle d\rangle.P'_{\mathbf{p}}\right]\ \mathbf{in}\\ \quad Y\langle s'_1[\mathbf{r}_1],\ldots,s'_{i'-1}[\mathbf{r}_{i'-1}],\ s[\mathbf{p}],\ s'_{i'+1}[\mathbf{r}_{i'+1}],\ldots,s'_m[\mathbf{r}_m]\rangle\end{array}}\,[\text{T-}\mathbf{def}] \tag{55}$$

The term above can reduce by rule [R-X] (Fig. 1), becoming the term (56) below, that we can type from (55) by applying Lemma B.1 m times (one per argument of Y). Notice, in particular, that d in (55) becomes a channel with role $s''[\mathbf{r}'']$ in (56): $s''[\mathbf{r}'']$ could come either from the call substitutions (i.e., it replaces some $y_{j'}$, $j' \in 1..m$), or from the session restrictions in $\mathbb{C}'_{\mathbf{p}}$; in both cases, $s''[\mathbf{r}'']$ is typed by some $\Gamma'_{\mathbf{p}s''}$ (taking the place of $\Gamma'_{\mathbf{p}d}$ in (55)):

$$\dfrac{\vdots \qquad \dfrac{\dfrac{\dfrac{\begin{array}{l}\Gamma_{\mathbf{p}i'} \vdash s[\mathbf{p}]{:}\mathbf{q}\oplus\mathtt{m}_k(T''_k).T'''_k \quad \Gamma'_{\mathbf{p}s''} \vdash s''[\mathbf{r}'']{:}T''_k\\ \Theta_{\mathbf{p}},\Theta'_{\mathbf{p}},Y{:}T_1,\ldots,T_m \cdot \Gamma''_{\mathbf{p}0}, s[\mathbf{p}]{:}T'''_k \vdash P'_{\mathbf{p}}\{s'_1[\mathbf{r}_1]/y_1\}\cdots\{s'_1[\mathbf{r}_m]/y_m\}\end{array}}{\begin{array}{l}\Theta_{\mathbf{p}},\Theta'_{\mathbf{p}},Y{:}T_1,\ldots,T_m\\ \quad\cdot\,\Gamma''_{\mathbf{p}0},\Gamma'_{\mathbf{p}s''},\Gamma_{\mathbf{p}i'}\end{array} \vdash s[\mathbf{p}][\mathbf{q}]\oplus\mathtt{m}_k\langle s''[\mathbf{r}'']\rangle.P'_{\mathbf{p}}\{s'_1[\mathbf{r}_1]/y_1\}\cdots\{s'_1[\mathbf{r}_m]/y_m\}}\,[\text{T-}\oplus]}{\vdots}}{\begin{array}{l}\Theta_{\mathbf{p}},Y{:}T_1,\ldots,T_m\\ \quad\cdot\,\Gamma_{\mathbf{p}1},\ldots\Gamma_{\mathbf{p}m}\end{array} \vdash \mathbb{C}'_{\mathbf{p}}\{s'_1[\mathbf{r}_1]/y_1\}\cdots\{s'_1[\mathbf{r}_m]/y_m\}\left[s[\mathbf{p}][\mathbf{q}]\oplus\mathtt{m}_k\langle s''[\mathbf{r}'']\rangle.P'_{\mathbf{p}}\{s'_1[\mathbf{r}_1]/y_1\}\cdots\{s'_1[\mathbf{r}_m]/y_m\}\right]}}{\begin{array}{l}\quad\Theta_{\mathbf{p}}\\ \cdot\,\Gamma_{\mathbf{p}0},\Gamma_{\mathbf{p}1},\ldots,\Gamma_{\mathbf{p}m}\end{array} \vdash \begin{array}{l}\mathbf{def}\ Y(y_1{:}T_1,\ldots,y_m{:}T_m) = \mathbb{C}'_{\mathbf{p}}\left[y_{i'}[\mathbf{q}]\oplus\mathtt{m}_k\langle d\rangle.P'_{\mathbf{p}}\right]\ \mathbf{in}\\ \quad\mathbb{C}'_{\mathbf{p}}\{s'_1[\mathbf{r}_1]/y_1\}\cdots\{s[\mathbf{p}]/y_{i'}\}\cdots\{s'_1[\mathbf{r}_m]/y_m\}\left[s[\mathbf{p}][\mathbf{q}]\oplus\mathtt{m}_k\langle s''[\mathbf{r}'']\rangle.P'_{\mathbf{p}}\{s'_1[\mathbf{r}_1]/y_1\}\cdots\{s'_1[\mathbf{r}_m]/y_m\}\right]\end{array}}\,[\text{T-}\mathbf{def}] \tag{56}$$

Applying the above reduction to the process in case (b) (via rule [R-Ctx]), and rearranging the reduction contexts via $\equiv$ into a single context $\mathbb{C}_{\mathbf{pq}}$ with one hole (using Prop. I.2), we get the following reduction, by (56) and [R-$\equiv$]:

$$P \;\to\; P'' \;\equiv\; \mathbb{C}_{\mathbf{pq}}\left[s[\mathbf{p}][\mathbf{q}]\oplus\mathtt{m}_k\langle s''[\mathbf{r}'']\rangle.P'_{\mathbf{p}}\{s'_1[\mathbf{r}_1]/y_1\}\cdots\{s'_1[\mathbf{r}_m]/y_m\} \;\mid\; s[\mathbf{q}][\mathbf{p}]\sum_{l\in L}\mathtt{m}_l(x_l).P'_{\mathbf{q}l}\right] \tag{57}$$

Notice that the typing context in the conclusion of (56) is the same of (55) (since the reduction involves a closed process variable Y and does not use any channel). Therefore:

$$\emptyset \cdot \Gamma \vdash \mathbb{C}_{\mathbf{pq}}\left[\begin{array}{l} s[\mathbf{p}][\mathbf{q}]\oplus \mathtt{m}_k\langle s''[\mathbf{r}'']\rangle.P'_{\mathbf{p}}\{s'_1[\mathbf{r}_1]/y_1\}\cdots\{s'_1[\mathbf{r}_m]/y_m\} \\ \mid s[\mathbf{q}][\mathbf{p}]\sum_{l\in L}\mathtt{m}_l(x_l).P'_{\mathbf{q}l} \end{array}\right] \quad \text{(by the hyp. } \emptyset \cdot \Gamma \vdash P,\ (57) \text{ and Lemma B.2)} \tag{58}$$

By (56) and Lemma I.4, the term inside $\mathbb{C}_{\mathbf{pq}}$ is typed as:

$$\dfrac{\dfrac{\begin{array}{c}\Gamma_{\mathbf{p}i'} \vdash s[\mathbf{p}]{:}\mathbf{q}\oplus\mathtt{m}_k(T''_k).T'''_k \quad \Gamma'_{\mathbf{p}s''} \vdash s''[\mathbf{r}'']{:}T''_k \\ \Theta''_{\mathbf{p}} \cdot \Gamma''_{\mathbf{p}0}, s[\mathbf{p}]{:}T'''_k \vdash P'_{\mathbf{p}}\{s'_1[\mathbf{r}_1]/y_1\}\cdots\{s'_1[\mathbf{r}_m]/y_m\}\end{array}}{\begin{array}{l}\Theta''_{\mathbf{p}} \\ \cdot\, \Gamma''_{\mathbf{p}0}, \Gamma'_{\mathbf{p}s''}, \Gamma_{\mathbf{p}i'}\end{array} \vdash s[\mathbf{p}][\mathbf{q}]\oplus\mathtt{m}_k\langle s''[\mathbf{r}'']\rangle.P'_{\mathbf{p}}\{s'_1[\mathbf{r}_1]/y_1\}\cdots\{s'_1[\mathbf{r}_m]/y_m\}}\,[\text{T-}\oplus] \quad \dfrac{\begin{array}{c}\Gamma_{\mathbf{q}} \vdash s[\mathbf{q}]{:}\mathbf{p}\&_{l\in L}\mathtt{m}_l(U''_l).U'''_l \\ \forall l \in L \quad \Theta''_{\mathbf{q}} \cdot \Gamma''_{\mathbf{q}0}, x_l{:}U''_l, s[\mathbf{q}]{:}U''' \vdash P'_{\mathbf{q}l}\end{array}}{\begin{array}{l}\Theta''_{\mathbf{q}} \\ \cdot\, \Gamma''_{\mathbf{q}0}, \Gamma_{\mathbf{q}}\end{array} \vdash s[\mathbf{q}][\mathbf{p}]\sum_{l\in L}\mathtt{m}_l(x_l).P'_{\mathbf{q}l}}\,[\text{T-}\&]}{\begin{array}{l}\Theta''_{\mathbf{p}}, \Theta''_{\mathbf{q}} \\ \cdot\, \Gamma''_{\mathbf{p}0}, \Gamma'_{\mathbf{p}s''}, \Gamma_{\mathbf{p}i'}, \Gamma''_{\mathbf{q}0}, \Gamma_{\mathbf{q}}\end{array} \vdash s[\mathbf{p}][\mathbf{q}]\oplus\mathtt{m}_k\langle s''[\mathbf{r}'']\rangle.P'_{\mathbf{p}}\{s'_1[\mathbf{r}_1]/y_1\}\cdots\{s'_1[\mathbf{r}_m]/y_m\} \mid s[\mathbf{q}][\mathbf{p}]\sum_{l\in L}\mathtt{m}_l(x_l).P'_{\mathbf{q}l}}\,[\text{T-}|] \tag{59}$$

Now, observe that from (58) and (54), we know that:

$$T''_k \leqslant S_k \leqslant S''_k \leqslant U''_k \quad \text{and thus} \quad \Gamma'_{\mathbf{p}s''} \vdash s''[\mathbf{r}'']{:}U''_k \qquad \text{(by } \Gamma'_{\mathbf{p}s''} \vdash s''[\mathbf{r}'']{:}T''_k,\ [\text{T-Sub}] \text{ and transit. of } \leqslant) \tag{60}$$

The process in (59) reduces to the following process (by [R-Comm]), which we can type by (60) and Lemma B.1:

$$\dfrac{\Theta''_{\mathbf{p}} \cdot \Gamma''_{\mathbf{p}0}, s[\mathbf{p}]{:}T'''_k \vdash P'_{\mathbf{p}}\{s'_1[\mathbf{r}_1]/y_1\}\cdots\{s'_1[\mathbf{r}_m]/y_m\} \qquad \Theta''_{\mathbf{q}} \cdot \Gamma''_{\mathbf{q}0}, \Gamma'_{\mathbf{p}s''}, s[\mathbf{q}]{:}U''' \vdash P'_{\mathbf{q}k}\{s''[\mathbf{r}'']/x_k\}}{\Theta''_{\mathbf{p}}, \Theta''_{\mathbf{q}} \cdot \Gamma''_{\mathbf{p}0}, \Gamma'_{\mathbf{p}s''}, s[\mathbf{p}]{:}T'''_k, \Gamma''_{\mathbf{q}0}, s[\mathbf{q}]{:}U''' \vdash P'_{\mathbf{p}}\{s'_1[\mathbf{r}_1]/y_1\}\cdots\{s'_1[\mathbf{r}_m]/y_m\} \mid P'_{\mathbf{q}k}\{s''[\mathbf{r}'']/x_k\}}\,[\text{T-}|] \tag{61}$$

Notice that the process typing context $\Theta''_{\mathbf{p}}, \Theta''_{\mathbf{q}}$ does not change in the reduction from (59) to (61). For the channel typing context, instead, we have:

$$\begin{array}{l}\Gamma''_{\mathbf{p}0}, \Gamma'_{\mathbf{p}s''}, \Gamma_{\mathbf{p}i'}, \Gamma''_{\mathbf{q}0}, \Gamma_{\mathbf{q}} \\ \quad \leqslant\ \Gamma''_{\mathbf{p}0}, \Gamma'_{\mathbf{p}s''}, s[\mathbf{p}]{:}\mathbf{q}\oplus\mathtt{m}_k(T''_k).T'''_k, \Gamma''_{\mathbf{q}0}, s[\mathbf{q}]{:}\mathbf{p}\&_{l\in L}\mathtt{m}_l(U''_l).U'''_l \\ \quad \to\ \Gamma''_{\mathbf{p}0}, \Gamma'_{\mathbf{p}s''}, s[\mathbf{p}]{:}T'''_k, \Gamma''_{\mathbf{q}0}, s[\mathbf{q}]{:}U'''\end{array} \qquad \left(\begin{array}{l}\text{by (59), [T-Sub], Def. 2.6,} \\ \text{(60), Def. 2.8}\end{array}\right) \tag{62}$$

$$\exists \Gamma'_{\mathbf{pq}} : \left\{\begin{array}{l}\Gamma''_{\mathbf{p}0}, \Gamma'_{\mathbf{p}s''}, \Gamma_{\mathbf{p}i'}, \Gamma''_{\mathbf{q}0}, \Gamma_{\mathbf{q}} \ \to\ \Gamma'_{\mathbf{pq}} \quad \text{and} \\ \Gamma'_{\mathbf{pq}} \ \leqslant\ \Gamma''_{\mathbf{p}0}, \Gamma'_{\mathbf{p}s''}, s[\mathbf{p}]{:}T'''_k, \Gamma''_{\mathbf{q}0}, s[\mathbf{q}]{:}U'''\end{array}\right. \qquad \text{(by (62) and Lemma I.19)} \tag{63}$$

$$\Theta''_{\mathbf{p}}, \Theta''_{\mathbf{q}} \cdot \Gamma'_{\mathbf{pq}} \vdash P'_{\mathbf{p}}\{s'_1[\mathbf{r}_1]/y_1\}\cdots\{s'_1[\mathbf{r}_m]/y_m\} \mid P'_{\mathbf{q}k}\{s''[\mathbf{r}'']/x_k\} \qquad \text{(by (61), (63) and Lemma B.3)} \tag{64}$$

Observe that in 62 and 63, the entries for $s[\mathbf{p}]$ and $s[\mathbf{q}]$ reduce, and (by (63) and Def. 2.8) such entries are the only difference between $\Gamma''_{\mathbf{p}0}, \Gamma'_{\mathbf{p}s''}, \Gamma_{\mathbf{p}i'}, \Gamma''_{\mathbf{q}0}, \Gamma_{\mathbf{q}}$ and $\Gamma'_{\mathbf{pq}}$. By applying the same update of $s[\mathbf{p}]$ and $s[\mathbf{q}]$ to Γ from the statement, we obtain a typing context Γ' such that:

$$\emptyset \cdot \Gamma' \vdash P' \ = \ \mathbb{C}_{\mathbf{pq}}\left[P'_{\mathbf{p}}\{s'_1[\mathbf{r}_1]/y_1\}\cdots\{s'_1[\mathbf{r}_m]/y_m\} \ \mid\ P'_{\mathbf{q}k}\{s''[\mathbf{r}'']/x_k\}\right] \qquad \text{(by (58), (59), (63), (64))} \tag{65}$$

$$P \to P'' \to P' \qquad \text{(by (57), (61) and (65))} \tag{66}$$

$$\Gamma \to \Gamma' \qquad \text{(by Def. 2.8)} \tag{67}$$

Summing up, we have shown that if Γ is safe and reduces ($\Gamma\!\to$), and the rest of the hypotheses in the statement hold, then there exist Γ', P' such that $\Gamma \to \Gamma'$ (by (67)), $P \to^* P'$ (by (66)) and $\emptyset \cdot \Gamma' \vdash P'$ (by (65)).

We are left to prove that:

(1) $P' \equiv \big|_{\mathbf{r}\in I} P'_{\mathbf{r}}$. This is proved by noticing that, from the hypothesis (b), (66) and [R-≡], the reductions from P to P' yield:

$$\begin{array}{l} P \to P'' \to P' \\ \quad \equiv \mathbb{C}_{\mathbf{p}}\left[\begin{array}{l}\mathbf{def}\ Y(y_1{:}T_1,\ldots,y_m{:}T_m) = \mathbb{C}'_{\mathbf{p}}\left[y_{i'}[\mathbf{q}]\oplus\mathtt{m}_k\langle d\rangle.P'_{\mathbf{p}}\right]\ \mathbf{in} \\ \quad \mathbb{C}'_{\mathbf{p}}\left[P'_{\mathbf{p}}\right]\{s'_1[\mathbf{r}_1]/y_1\}\cdots\{s'_1[\mathbf{r}_m]/y_m\}\end{array}\right] \mid \mathbb{C}_{\mathbf{q}}\left[P'_{\mathbf{q}k}\{s''[\mathbf{r}'']/x_k\}\right] \mid Q \end{array} \tag{68}$$

and this satisfies the requirement;

(2) each $P'_{\mathbf{r}}$ has guarded definitions and is either $\mathbf{0}$ (up-to ≡), or only plays role $\mathbf{r}$ in s. From (68), since Q and the reduction contexts $\mathbb{C}_{\mathbf{p}}$, $\mathbb{C}'_{\mathbf{p}}$, $\mathbb{C}_{\mathbf{q}}$ are unchanged w.r.t. the hypothesis (b), we only need to show that the process inside $\mathbb{C}_{\mathbf{p}}$ (resp. $\mathbb{C}_{\mathbf{q}}$) is either $\mathbf{0}$ (up-to ≡), or only plays role $\mathbf{p}$ (resp. $\mathbf{q}$) in s. We only examine the process inside

$\mathbb{C}_{\mathsf{q}}$ (the reasoning for the other is similar). If $P'_{\mathsf{q}k}\{s''[\mathsf{r}'']/x_k\} \equiv \mathbf{0}$, we have $\mathbb{C}_{\mathsf{q}}\left[P'_{\mathsf{q}k}\{s''[\mathsf{r}'']/x_k\}\right] \equiv \mathbf{0}$, and we conclude easily. Otherwise, observe that:

$$\emptyset \cdot \Gamma'_{\mathsf{q}} \vdash \mathbb{C}_{\mathsf{q}}\left[P'_{\mathsf{q}k}\{s''[\mathsf{r}'']/x_k\}\right]$$

where Γ'_{q} is a part of Γ' (by (68) and Lemma I.4(5)) where the only non-**end**-typed channel is $s[\mathsf{q}]$. Moreover, from the initial hypotheses, in all subterms $(\nu s''':\Gamma''')\ P'''$ of $\mathbb{C}_{\mathsf{q}}$ and $P'_{\mathsf{q}k}$, we have $\Gamma''' = s'[\mathsf{p}']:\mathbf{end}$ (for some p') and all the process definitions in $\mathbb{C}_{\mathsf{q}}$ and $P'_{\mathsf{q}k}$ are guarded. Hence, by Def. 5.3, we conclude that $\mathbb{C}_{\mathsf{q}}\left[P'_{\mathsf{q}k}\{s''[\mathsf{r}'']/x_k\}\right]$ only plays role q in s (by Γ'_{q}) □

J SUBJECT REDUCTION

Proposition J.1. *Assume:*

$$\Gamma \ = \ s[\mathsf{p}]:S, s[\mathsf{q}]:T \ \leqslant \ \Gamma'\!\rightarrow$$

with safe(Γ). *Then,* $\Gamma\!\rightarrow$.

Proof. We have:

$$\begin{array}{lll}
\Gamma' \ = \ s[\mathsf{p}]:S', s[\mathsf{q}]:T' & & \\
\quad \text{with } \ \mathrm{unf}^*(S') = \mathsf{q}\oplus_{i\in I'}\mathtt{m}_i(S''_i).S'''_i & & \\
\quad \text{and } \ \mathrm{unf}^*(T') = \mathsf{p}\&_{j\in J'}\mathtt{m}_j(T''_j).T'''_j & (\text{by } \Gamma'\!\rightarrow, \text{Def. } 2.8, \text{Prop.I.18}) & (69)\\
\quad \text{and } \ I' \subseteq J' \ \text{ and } \ \forall i\in I' : S''_i \leqslant T''_i & & \\
\Gamma \ = \ s[\mathsf{p}]:S, s[\mathsf{q}]:T & & \\
\quad \text{with } \ \mathrm{unf}^*(S) = \mathsf{q}\oplus_{i\in I}\mathtt{m}_i(S_i).S'_i & (\text{by } \Gamma \leqslant \Gamma', (69), \text{Prop.I.16, Def. } 2.5) & (70)\\
\quad \text{and } \ \mathrm{unf}^*(T') = \mathsf{p}\&_{j\in J}\mathtt{m}_j(T_j).T'_j & & \\
k \in I \subseteq J \ \text{ and } \ \forall i \in I : S_i \leqslant T_i & (\text{by } (70) \text{ and Def. } 4.1) & (71)
\end{array}$$

Therefore, by (70), (71), Def. 2.8 and Prop.I.18, we conclude $\Gamma\!\rightarrow$. □

Lemma 4.4. *If* Γ *safe and* $\Gamma \leqslant \Gamma' \rightarrow \Gamma''$, *then there is* Γ''' *such that* $\Gamma \rightarrow \Gamma''' \leqslant \Gamma''$.

Proof. Assume $\Gamma' \rightarrow \Gamma''$, induced by the interaction of two entries $s[\mathsf{p}]:S', s[\mathsf{q}]:T'$ in Γ'. Now, assume $\Gamma \leqslant \Gamma'$: by Def. 2.6, Γ contains the entries $s[\mathsf{p}]:S, s[\mathsf{q}]:T$ (for some S, T) and they reduce by the safety hypothesis and Prop.J.1. Then, we conclude by Lemma I.19. □

Lemma 4.3. *If* Γ, Γ' *is safe, then* Γ *is safe.*

Proof. By contradiction, assume that Γ is *not* safe. Then, by Def. 4.1 (clause [S-$\rightarrow$]), there is Γ'' such that $\Gamma \rightarrow^* \Gamma''$, and Γ'' violates clause [S-$\oplus\&$] (possibly after applying [S-μ] to unfold its entries). But then, by Def. 2.8 (rule [Γ-Cong]), $\Gamma, \Gamma' \rightarrow^* \Gamma'', \Gamma'$ and the latter is *not* safe. This means that Γ, Γ' violates clause [S-$\rightarrow$] of Def. 4.1, and therefore is *not* safe: contradiction. We conclude that Γ is safe. □

Theorem 4.8 (Subject Reduction). *Assume* $\Theta \cdot \Gamma \vdash P$ *and* Γ *safe. Then,* $P \rightarrow P'$ *implies* $\exists \Gamma'$ *safe such that* $\Gamma \rightarrow^* \Gamma'$ *and* $\Theta \cdot \Gamma' \vdash P'$.

Proof. By induction of the derivation of $P \rightarrow P'$, and when the reduction holds by rule [R-Ctx], with a further structural induction on the reduction context $\mathbb{C}$. Most cases hold by inversion of the typing $\Theta \cdot \Gamma \vdash P$, and by applying the induction hypothesis. The most interesting case is the base case where $P \to P'$ holds by [R-Comm]:

$$\begin{array}{rcl} P & = & s[\mathtt{p}][\mathtt{q}]\sum_{i\in I}\mathtt{m}_i(x_i).P_i \mid s[\mathtt{q}][\mathtt{p}]\oplus\mathtt{m}_k\langle s'[\mathtt{r}]\rangle.Q \\ P' & = & P_k\{s'[\mathtt{r}]/x_k\} \mid Q \quad (k \in I)\end{array} \qquad \text{(by inversion of [R-Comm])} \tag{72}$$

$$\Gamma = \Gamma_{\&}, \Gamma_{\oplus} \quad \text{such that} \quad \dfrac{\begin{array}{c}\Theta\cdot\Gamma_{\&} \vdash s[\mathtt{p}][\mathtt{q}]\sum_{i\in I}\mathtt{m}_i(x_i).P_i \\ \Theta\cdot\Gamma_{\oplus} \vdash s[\mathtt{q}][\mathtt{p}]\oplus\mathtt{m}_k\langle s'[\mathtt{r}]\rangle.Q\end{array}}{\Theta\cdot\Gamma \vdash P}\,[\text{T-}|] \qquad \text{(by (72) and inv. of [T-|])} \tag{73}$$

$$\Gamma_{\&} = \Gamma_0, \Gamma_1 \quad \text{such that} \quad \dfrac{\begin{array}{c}\Gamma_1 \vdash s[\mathtt{p}]{:}\mathtt{q}\&_{i\in I}\mathtt{m}_i(S_i).S'_i \\ \forall i \in I \quad \Theta\cdot\Gamma_0, x_i{:}S_i, s[\mathtt{p}]{:}S'_i \vdash P_i\end{array}}{\Theta\cdot\Gamma_{\&} \vdash s[\mathtt{p}][\mathtt{q}]\sum_{i\in I}\mathtt{m}_i(x_i).P_i}\,[\text{T-}\&] \qquad \text{(by (73) and inv. of [T-\&])} \tag{74}$$

$$\Gamma_{\oplus} = \Gamma_2, \Gamma_3, \Gamma_4 \quad \text{such that} \quad \dfrac{\begin{array}{c}\Gamma_4 \vdash s[\mathtt{q}]{:}\mathtt{q}\oplus\mathtt{m}_k(T_k).T'_k \\ \Gamma_3 \vdash s'[\mathtt{r}]{:}T_k \quad \Theta\cdot\Gamma_2, s[\mathtt{q}]{:}T'_k \vdash Q\end{array}}{\Theta\cdot\Gamma_{\oplus} \vdash s[\mathtt{q}][\mathtt{p}]\oplus\mathtt{m}_k\langle s'[\mathtt{r}]\rangle.Q}\,[\text{T-}\oplus] \qquad \text{(by (73) and inv. of [T-}\oplus])} \tag{75}$$

Now, notice that:

$$\Gamma = \Gamma_0, \Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4 \qquad \text{(by (73), (74), and (75))} \tag{76}$$

$$\Gamma_1 = s[\mathtt{p}]{:}S \quad \text{with} \quad S \leqslant \mathtt{q}\&_{i\in I}\mathtt{m}_i(S_i).S'_i \qquad \text{(by (74) and Fig. 2, rule [T-Sub])} \tag{77}$$

$$\Gamma_4 = s[\mathtt{q}]{:}T \quad \text{with} \quad T \leqslant \mathtt{q}\oplus\mathtt{m}_k(T_k).T'_k \qquad \text{(by (75) and Fig. 2, rule [T-Sub])} \tag{78}$$

$$\Gamma \leqslant \Gamma'' = \Gamma_0, s[\mathtt{p}]{:}\mathtt{q}\&_{i\in I}\mathtt{m}_i(S_i).S'_i, \Gamma_2, \Gamma_3, s[\mathtt{q}]{:}\mathtt{q}\oplus\mathtt{m}_k(T_k).T'_k \qquad \text{(by (76), (77), (78), and Def. 2.6)} \tag{79}$$

$$\text{safe}(\Gamma'') \qquad \text{(since safe}(\Gamma)\text{, and by (79) and Lemma 4.5)} \tag{80}$$

$$k \in I \quad \text{and} \quad T_k \leqslant S_k \qquad \text{(by (79), (80) and Def. 4.1, clause [S-}\oplus\&])} \tag{81}$$

$$\Gamma'' \to \Gamma''' = \Gamma_0, s[\mathtt{p}]{:}S'_k, \Gamma_2, \Gamma_3, s[\mathtt{q}]{:}T'_k \qquad \text{(by (79), (81) and Def. 2.8)} \tag{82}$$

$$\text{safe}(\Gamma''') \qquad \text{(by (80), (82) and Def. 4.1, clause [S-}\to])} \tag{83}$$

We can now use Γ''' to type P':

$$\Theta\cdot\Gamma_0, x_k{:}S_k, s[\mathtt{p}]{:}S'_k \vdash P_k \qquad \text{(by (81), (75) and (74))} \tag{84}$$

$$\Gamma_3 \vdash s'[\mathtt{r}]{:}S_k \qquad \text{(by (75) (for } \Gamma_3 \vdash s'[\mathtt{r}]{:}T_k\text{), (81), transitivity of } \leqslant\text{, and [T-Sub])} \tag{85}$$

$$\Gamma_0, \Gamma_3, s[\mathtt{p}]{:}S'_k \quad \text{defined} \qquad \text{(by (75), (74), and (73))} \tag{86}$$

$$\Theta\cdot\Gamma_0, \Gamma_3, s[\mathtt{p}]{:}S'_k \vdash P_k\{s'[\mathtt{r}]/x_k\} \qquad \text{(by (84), (85), (86), and Lemma B.1)} \tag{87}$$

$$\dfrac{\Theta\cdot\Gamma_0, \Gamma_3, s[\mathtt{p}]{:}S'_k \vdash P_k\{s'[\mathtt{r}]/x_k\} \qquad \Theta\cdot\Gamma_2, s[\mathtt{q}]{:}T'_k \vdash Q}{\Theta\cdot\Gamma''' \vdash P'}\,[\text{T-}|] \qquad \text{(by (87), (75), (82), (83) and (72))} \tag{88}$$

We conclude the proof by showing that there exists some Γ' that satisfies the statement:

$$\exists\Gamma' : \Gamma \to \Gamma' \leqslant \Gamma''' \qquad \text{(by (79), (82), and Lemma 4.4)} \tag{89}$$

$$\text{safe}(\Gamma') \qquad \text{(by (89) and Def. 4.1, clause [S-}\to])} \tag{90}$$

$$\Theta\cdot\Gamma' \vdash P' \qquad \text{(by (88), (89), and Lemma B.3)}$$

□

Proposition J.2. *The multiparty session subtyping relation $\leqslant$ is decidable.*

Proof. An algorithm for checking whether a pair of types S, T belongs to $\leqslant$ can be obtained as a variation of the binary session subtyping algorithm of [Gay and Hole 2005, Fig. 11] — which in turn is based on the subtyping algorithm for recursive types of [Pierce 2002, Fig. 21-4]. See [Ghilezan et al. 2018] for a detailed description of the algorithm, and the paper artifact [Scalas and Yoshida 2019b] for an implementation. □

Theorem 4.11. *If φ is decidable, then "$\Theta\cdot\Gamma \vdash P$ with φ" is decidable.*

Proof. An algorithm for deciding $\Theta\cdot\Gamma \vdash P$ can be straightforwardly obtained by inverting the typing rules in Fig. 2 and Def. 4.6, noticing that:

(1) all typing rules are deterministically invertible — i.e., for each shape of P, at most one rule can conclude $\Theta \cdot \Gamma \vdash P$;
(2) at each inversion, the typing contexts Θ, Γ and the type annotations in P determine how to populate the typing context in the premises the rule — with the exception of [T-|], that (in the worst case) might require to try all possible Γ_1, Γ_2 such that $\Gamma_1, \Gamma_2 = \Gamma$;
(3) each rule inversion yields premises with strictly smaller subterms of P (thus, recursive type checking eventually terminates).

Moreover, since $\leqslant$ is decidable (by Prop. J.2), it is always decidable whether the judgement [T-Sub] holds. Finally, notice that when inverting rule [TGen-ν] (Def. 4.6), the check $\varphi(\Gamma')$ is decidable by hypothesis. □

K TYPING CONTEXT PROPERTIES

Global Type Projections as Behavioural Properties. To develop the proofs below, we establish a link between *syntactic* projections and *behavioural* typing context properties: this is done in Def. K.4, using some tools from Def. K.1.

Definition K.1 (Typing Context Unfoldings and Behavioural Set). The *set of unfoldings of* Γ, written $\text{unf}(\Gamma)$, is defined as follows (where $\Gamma\{S/c\}$ is a mapping update):

$$\text{unf}(\Gamma) = \bigcup_{c:S\in\Gamma} \{\, \Gamma\{\text{unf}(S)/c\} \,\} \qquad \text{extended to sets as} \qquad \text{unf}(\{\Gamma_i\}_{i\in I}) = \bigcup_{i\in I} \text{unf}(\Gamma_i)$$

Given a set of typing contexts $\mathcal{E}$, the *closure of the unfoldings of its elements* is:

$$\text{unf}^*(\mathcal{E}) = \text{lfix}\left(\lambda\mathcal{E}'.\,\mathcal{E} \cup \mathcal{E}' \cup \text{unf}(\mathcal{E} \cup \mathcal{E}')\right) \qquad \text{where lfix is the } \textit{least fixed point} \text{ of its argument}$$

Given Γ, the *behavioural set of* Γ, written $\text{beh}(\Gamma)$, is the set: $\text{beh}(\Gamma) \;=\; \text{unf}^*(\,\{\Gamma' \mid \Gamma \to^* \Gamma'\}\,)$

In Def. K.1, $\text{beh}(\Gamma)$ is the set of all reductions of Γ, extended with all unfoldings of all their entries. This ensures that $\text{beh}(\Gamma)$ mechanically satisfies clauses [S-$\to$] and [S-μ] of Def. 4.1 and Fig. 5 (cf. K.2 below). Therefore, one can verify whether $\text{beh}(\Gamma)$ is a safety/liveness property by only checking whether the remaining clauses hold for all elements — and if they do, it means that Γ is safe/live. Note that the least fixed point in $\text{unf}^*(\mathcal{E})$ exists because the function is monotonic w.r.t. $\subseteq$ [Tarski 1955].

Proposition K.2. *Let $\varphi = \text{beh}(\Gamma)$, for some Γ. Then, each element of φ satisfies clauses [S-$\to$] and [S-μ] of Def. 4.1.*

For each safe Γ, Def. K.1 gives the smallest safety property, as formalised in Prop. K.3 below.

Proposition K.3. *Γ is safe (resp. live, live$^+$) if and only if $\text{beh}(\Gamma)$ is a safety (resp. liveness, liveness$^+$) property.*

Proof. ($\Longrightarrow$). Assume that Γ is safe (resp. live, live$^+$). We have to prove that each $\Gamma' \in \text{beh}(\Gamma)$ satisfies the clauses of Def. 4.1 (resp. 5(5), 5(6)). By Prop. K.2, we know that each Γ' satisfies clauses [S-μ] and [S-$\to$]; the remaining clauses are easily proved by contradiction: if we assume that Γ' does *not* satisfy some clause, by observing that Γ' is a (possibly unfolded) reduct of Γ, we obtain that Γ is *not* safe (resp. live, live$^+$) — contradiction. Therefore, each $\Gamma' \in \text{beh}(\varphi)$ satisfies all clauses of Def. 4.1 (resp. 5(5), 5(6)), and we conclude that $\text{beh}(\Gamma)$ is a safety (resp. liveness, liveness$^+$) property.
($\Longleftarrow$). Immediate by Def. 4.1 (resp. 5(5), 5(6)). □

We can now use global type projections to produce behavioural properties that are directly usable in our framework. This is formalised in Def. K.4.

Definition K.4. With an abuse of notation, we will use the following definitions instead of those in Def. 5.8:

$$\begin{aligned} \text{fproj}_{G,s} &= \text{beh}(\Gamma) \qquad &&\text{where } \Gamma \text{ is the projection of } G \text{ for } s \\ \text{pproj}_{G,s} &= \text{beh}(\Gamma) \qquad &&\text{where } \Gamma \text{ is the plain projection of } G \text{ for } s \end{aligned}$$

i.e., we extend the properties in Def. 5.8 to contain the unfoldings and reductions of a typing context Γ projected from G. The results below will hold for such extensions, and thus, also for the original Def. 5.8.

Proposition K.5. *If* $\text{consistent}(\Gamma)$ *and* $\Gamma \to \Gamma'$, *then* $\text{consistent}(\Gamma')$.

Proof. From [Scalas et al. 2017, Lemma D.22]. □

Definition K.6 (Type contexts). A *global type context* is defined as follows:

$$\mathbb{G} \;::=\; \mathbf{p}{\to}\mathbf{q}{:}\,\{\mathtt{m}_i(S_i)\,.\,\mathbb{G}_i\}_{i\in I} \;\;\Big|\;\; \mu\mathbf{t}.\mathbb{G} \;\;\Big|\;\; [\;]^i$$

We write $\mathbb{G}[G_i]^i_{i\in I}$ to denote the global type obtained by filling the hole $[\;]^i$ of $\mathbb{G}$ with the global type G_i, for all $i \in I$, with the understanding that I indexes all the holes in $\mathbb{G}$.

A *session type context* is defined as follows:

$$\mathbb{S} \ ::= \ \mathsf{p}\&_{i\in I}\mathsf{m}_i(T_i).\mathbb{S}_i \ \big| \ \mathsf{p}\oplus_{i\in I}\mathsf{m}_i(T_i).\mathbb{S}_i \ \big| \ \mu\mathbf{t}.\mathbb{S} \ \big| \ [\ ]^i$$

We write $\mathbb{S}[T_i]^i_{i\in I}$ to denote the session type obtained by filling the hole $[\ ]^i$ of $\mathbb{S}$ with the session type T_i, for all $i \in I$, with the understanding that I indexes all the holes in $\mathbb{S}$.

We now adapt to our framework the notion of *multiparty compatibility* defined in [Deniélou and Yoshida 2013, Def. 4.2] and [Bocchi et al. 2015, Def. 4], for Communicating Finite-State Machines (CFSMs).

Definition K.7 (Multiparty Compatibility [Bocchi et al. 2015; Deniélou and Yoshida 2013]). Assume $\mathrm{dom}(\Gamma) = \{s\}$. We say that Γ *is multiparty compatible* iff:

(1) $\Gamma \rightarrow^* \Gamma', s[\mathsf{p}]{:}S$ with $\mathrm{unf}^*(S) = \mathsf{q}\&_{i\in I}\mathsf{m}_i(S_i).S'_i$ implies $\exists k \in I\colon \exists \Gamma'', \Gamma'''\colon \Gamma' \rightarrow^* \Gamma''$ and $\Gamma'', s[\mathsf{p}]{:}S \rightarrow \Gamma''', s[\mathsf{p}]{:}S'_k$;
(2) $\Gamma \rightarrow^* \Gamma', s[\mathsf{p}]{:}S$ with $\mathrm{unf}^*(S) = \mathsf{q}\oplus_{i\in I}\mathsf{m}_i(S_i).S'_i$ implies $\forall k \in I\colon \exists \Gamma'', \Gamma'''\colon \Gamma' \rightarrow^* \Gamma''$ and $\Gamma'', s[\mathsf{p}]{:}S \rightarrow \Gamma''', s[\mathsf{p}]{:}S'_k$.

Note that in Def. K.7 is specifically based on [Bocchi et al. 2015, Def. 4]: clause 1 models a CFSM that is waiting to input from another CFSM in the system (i.e., a type in Γ'), and eventually succeeds; clause 2 models a CFSM that has queued a single output message, that is eventually received by another CFSM in the system. Also note the reductions in Def. K.7 mimic the "synchronous transition system" of the CFSMs in the original formulation: intuitively, it means that each message enqueued by a CFSM is immediately consumed by the recipient, and thus, at most one message can be queued at each reduction step, and must be received at the very next step. In our setting, such pairs of alternating queueing/dequeueing reductions are captured by a single synchronisation step.

Proposition K.8. *Assume* $\mathrm{dom}(\Gamma) = \{s\}$. *Then,* Γ *is multiparty compatible* if and only if Γ *is live.*

Proof. First, let $\varphi = \mathrm{beh}(\Gamma)$.
($\Longrightarrow$) We show that when Γ is multiparty compatible, φ satisfies all clauses of Fig. 5(5), hence is a liveness property; then, by Prop. K.3, we conclude that Γ is live.
($\Longleftarrow$) If Γ is live, φ is a liveness property (by Prop. K.3); therefore, by Def. K.1, $\Gamma \rightarrow^* \Gamma'$ implies that Γ' is contained (with its unfoldings) in φ, and satisfies the clauses of Fig. 5(5). By inspecting each Γ', we prove that it satisfies the clauses of Def. K.7; then, we conclude that Γ is multiparty compatible. □

Lemma K.9. *Assume* $\exists G : \mathrm{fproj}_{G,s}(\Gamma)$. *Then,* $\mathrm{dom}(\Gamma) = \{s\}$ *(Def. 2.6), and* $\mathrm{live}(\Gamma)$.

Proof. We obtain $\mathrm{dom}(\Gamma) = \{s\}$ straightforwardly from Def. 5.8. The rest of the statement is consequence of [Deniélou and Yoshida 2013, Thm. 4.3]: if all projections of G onto its roles are defined (which is our hypothesis), then the resulting set of local types is multiparty compatible (Def. K.7); then, we conclude by Prop. K.8. □

Proposition K.10. *Assume that* $G{\upharpoonright}\mathsf{q}$ *is defined. Then, either:*

(A) $\mathsf{q} \notin G$ *and* $G{\upharpoonright}\mathsf{q} \leqslant \mathbf{end}$, *or*
(B) $\mathsf{q} \notin G$ *and* $G{\upharpoonright}\mathsf{q} = \mathbf{t}$ *(for some* $\mathbf{t}$*), or*
(C) $\exists \mathbb{G}, I, G_i\ (i \in I), \mathsf{p}$ *such that* $G = \mathbb{G}[G_i]^i_{i\in I}$, $\mathsf{q} \notin \mathbb{G}$, *and either:*
  (a) $\forall i \in I : G_i = \mathsf{p}{\rightarrow}\mathsf{q}\colon \left\{\mathsf{m}_j(S_j)\,.\,G'_j\right\}_{j\in J_i}$; *or*
  (b) $\forall i \in I : G_i = \mathsf{q}{\rightarrow}\mathsf{p}\colon \left\{\mathsf{m}_j(S_j)\,.\,G'_j\right\}_{j\in J_i}$.

Proof. By structural induction on G:

- base case $G = \mathbf{end}$. Then, $\mathsf{q} \notin G$; moreover, we have $(\mu\mathbf{t}.G){\upharpoonright}\mathsf{q} = \mathbf{end}$ (by Def. 3.3). Hence, we conclude by obtaining (A);
- base case $G = \mathbf{t}$ (for some $\mathbf{t}$). Then, $\mathsf{q} \notin G$; moreover, we have $G{\upharpoonright}\mathsf{q} = \mathbf{t}$ (by Def. 3.3). Hence, we conclude by obtaining (B);
- inductive case $G = \mathsf{r}{\rightarrow}\mathsf{r}'\colon \left\{\mathsf{m}_k(S_k)\,.\,G'_k\right\}_{k\in K}$. Then, we have the following sub-cases:
  - $\mathsf{r}' = \mathsf{q}$. Then, by letting $\mathbb{G} = [\ ]^1$, $I = \{1\}$, $G_1 = G$, we conclude by obtaining (C)(a);
  - $\mathsf{r} = \mathsf{q}$. Similar to the previous case, and we conclude by obtaining (C)(b);

– $\mathbf{r} \neq \mathbf{q} \neq \mathbf{r}'$. Then, we have:

$$G\restriction\mathbf{q} = \bigsqcap_{k\in K} (G'_k\restriction\mathbf{q}) \qquad \text{(by Def. 3.3)} \qquad (91)$$

$$\forall k \in K: \quad G'_k\restriction\mathbf{q} \text{ is defined} \qquad \text{(by (91))} \qquad (92)$$

$$\forall k \in K: \quad G_k \text{ satisfies either (A), (B) or (C), with } G_k \text{ in place of } G \qquad \text{(by (92) and the induction hyp.)} \qquad (93)$$

Now, by inspecting the cases in which the merging in (91) is defined (by Def. 3.3), we can reduce (93) to the following possibilities:

* $\exists n \geq 0 : \forall k \in K : G_k\restriction\mathbf{q} = \mu\mathbf{t}_1.\cdots\mu\mathbf{t}_n.\mathbf{end}$. Then, $G\restriction\mathbf{q} = \mu\mathbf{t}_1.\cdots\mu\mathbf{t}_n.\mathbf{end}$ (by Def. 3.3), and thus, $G\restriction\mathbf{q} \leqslant \mathbf{end}$ (by Prop. I.15). Hence, we conclude by obtaining (A);
* $\forall k \in K : G_k\restriction\mathbf{q} = \mathbf{t}$ (for some $\mathbf{t}$). Then, $G\restriction\mathbf{q} = \mathbf{t}$ (by Def. 3.3), and thus, we conclude by obtaining (B);
* $\forall k \in K : \exists \mathbb{G}_k, I_k, G'_i \ (i \in I_k)$ such that $G_k = \mathbb{G}_k[G'_i]^i_{i\in I_k}$, $\mathbf{q}\notin\mathbb{G}_k$, and either:
  · $\forall k\in K, i\in I_k : G'_i = \mathbf{p}{\to}\mathbf{q}{:}\left\{\mathtt{m_j}(S_j)\,.\,G'_j\right\}_{j\in J_i}$. Then, by letting $\mathbb{G} = \mathbf{r}{\to}\mathbf{r}'{:}\{\mathtt{m_k}(S_k)\,.\,\mathbb{G}_k\}_{k\in K}$, $I = \bigcup_{k\in K} I_k$, and $\forall i \in I : G_i = G'_i$, we conclude by obtaining (C)(a);
  · $\forall k\in K, i\in I_k : G_i = \mathbf{q}{\to}\mathbf{p}{:}\left\{\mathtt{m_j}(S_j)\,.\,G'_j\right\}_{j\in J_i}$. Similar to the previous case, and we conclude by obtaining (C)(b);

• inductive case $G = \mu\mathbf{t}.G'$. Then, we have:

$$G\restriction\mathbf{q} = \begin{cases} \mu\mathbf{t}.(G'\restriction\mathbf{q}) & \text{if } G'\restriction\mathbf{q} \neq \mathbf{t}' \ (\forall\mathbf{t}') \\ \mathbf{end} & \text{otherwise} \end{cases} \qquad \text{(by Def. 3.3)} \qquad (94)$$

$$G'\restriction\mathbf{q} \text{ is defined} \qquad \text{(by (94))} \qquad (95)$$

$$G' \text{ satisfies either (A), (B) or (C), with } G' \text{ in place of } G \qquad \text{(by (95) and the induction hyp.)} \qquad (96)$$

Now, we have the following possibilities:

– $G'\restriction\mathbf{q} \leqslant \mathbf{end}$. Then, $\exists n\geq 0 : G'\restriction\mathbf{q} = \mu\mathbf{t}_1.\cdots\mu\mathbf{t}_n.\mathbf{end}$, which implies $G\restriction\mathbf{q} = \mu\mathbf{t}.\mu\mathbf{t}_1.\cdots\mu\mathbf{t}_n.\mathbf{end}$ (by Def. 3.3), and thus, $G\restriction\mathbf{q} \leqslant \mathbf{end}$ (by Prop. I.15). Hence, we conclude by obtaining (A);
– $G'\restriction\mathbf{q} = \mathbf{t}'$ (for some $\mathbf{t}'$). Then, by (94) we have $G\restriction\mathbf{q} = \mathbf{end}$, and we conclude by obtaining (A).
– $\exists \mathbb{G}', I', G'_i \ (i \in I')$ such that $G' = \mathbb{G}'[G'_i]^i_{i\in I'}$, $\mathbf{q}\notin\mathbb{G}'$, and either:
  * $\forall i\in I' : G'_i = \mathbf{p}{\to}\mathbf{q}{:}\left\{\mathtt{m_j}(S_j)\,.\,G'_j\right\}_{j\in J_i}$. Then, by letting $\mathbb{G} = \mu\mathbf{t}.\mathbb{G}'$, $I = I'$, and $\forall i \in I : G_i = G'_i$, we conclude by obtaining (C)(a);
  * $\forall i\in I' : G'_i = \mathbf{q}{\to}\mathbf{p}{:}\left\{\mathtt{m_j}(S_j)\,.\,G'_j\right\}_{j\in J_i}$. Similar to the previous case, and we conclude by obtaining (C)(b).

□

Corollary K.11. *Let $G = \mu\mathbf{t}.G'$, and assume that $G\restriction\mathbf{q}$ is defined. Then, either:*

(A) $\mathbf{q}\notin G$ *and* $(\mu\mathbf{t}.G)\restriction\mathbf{q} \leqslant \mathbf{end}$, *or*
(B) $\exists \mathbb{G}, I, G_i (i \in I), \mathbf{p}$ *such that* $G = \mu\mathbf{t}.\mathbb{G}[G_i]^i_{i\in I}$, $\mathbf{q}\notin\mathbb{G}$, *and either:*
  (a) $\forall i\in I : G_i = \mathbf{p}{\to}\mathbf{q}{:}\left\{\mathtt{m_j}(S_j)\,.\,G'_j\right\}_{j\in J_i}$; *or*
  (b) $\forall i\in I : G_i = \mathbf{q}{\to}\mathbf{p}{:}\left\{\mathtt{m_j}(S_j)\,.\,G'_j\right\}_{j\in J_i}$.

Proof. Assuming the hypothesis, we know that $G'\restriction\mathbf{q}$ is defined, and does *not* equal $\mathbf{t}'$ for any $\mathbf{t}'$ (by Def. 3.3). Therefore, we can apply Prop. K.10 on G', obtaining either K.10(A), or K.10(C); from these, with the same reasoning of the case "$G = \mu\mathbf{t}.G'$" in the proof of Prop. K.10, we obtain respectively items (A) or (B) of the thesis. □

Proposition K.12. *Assume that $T = G\restriction\mathbf{q}$ is defined. Then:*

(1) $T = \mu\mathbf{t}.T'$ *implies* $\exists \mathbb{G} :\ \mathbf{q}\notin\mathbb{G}$ *and* $G = \mathbb{G}[\mu\mathbf{t}.G_i]^i_{i\in I}$ *and* $\forall i\in I : G_i\restriction\mathbf{q} \neq \mathbf{t}' \ (\forall\mathbf{t}')$ *and* $T' = \bigsqcap_{i\in I}(G_i\restriction\mathbf{q})$;
(2) $T = \mu\mathbf{t}_1.\cdots\mu\mathbf{t}_n.\mathbf{p}\&_{j\in J}\mathtt{m}_j(S_j).S'_j$ *implies* $\exists \mathbb{G} :\ \mathbf{q}\notin\mathbb{G}$ *and* $G = \mathbb{G}[\mathbf{p}{\to}\mathbf{q}{:}\left\{\mathtt{m_j}(S_j)\,.\,G'_j\right\}_{j\in J}]^i_{i\in I}$;
(3) $T = \mu\mathbf{t}_1.\cdots\mu\mathbf{t}_n.\mathbf{p}\oplus_{j\in J}\mathtt{m}_j(S_j).S'_j$ *implies* $\exists \mathbb{G} :\ \mathbf{q}\notin\mathbb{G}$ *and* $G = \mathbb{G}[\mathbf{q}{\to}\mathbf{p}{:}\left\{\mathtt{m_j}(S_j)\,.\,G'_j\right\}_{j\in J_i}]^i_{i\in I}$;
(4) $T = \mu\mathbf{t}_1.\cdots\mu\mathbf{t}_n.\mathbf{end}$ *or* $T = \mathbf{t}$ *(for some* $\mathbf{t}$*)* *implies* $\mathbf{q} \notin G$.

Proof. Item (1) is proven by assuming

$$G{\restriction}\mathsf{q} = \mu\mathbf{t}.T' \qquad \text{(by hypothesis)} \qquad (97)$$

and proceeding by structural induction on G:

- base cases $G = \mathbf{end}$ and $G = \mathbf{t}'$ (for some $\mathbf{t}'$). Impossible, because by Def. 3.3, they would contradict (97);
- inductive case $G = \mathsf{p}\to\mathsf{r}: \{\mathtt{m_j}(S_j)\,.\,G_j\}_{j\in J}$. By Def. 3.3, we have $\mathsf{p}\neq\mathsf{q}\neq\mathsf{r}$ (otherwise, we would contradict (97)). Then:

$$G{\restriction}\mathsf{q} = \bigsqcap_{j\in J}(G_j{\restriction}\mathsf{q}) \;\text{ with }\; G_j{\restriction}\mathsf{q} = \mu\mathbf{t}.T_j \;\text{ and }\; T' = \bigsqcap_{j\in J} T_j \qquad \text{(by (97) and Def. 3.3)} \qquad (98)$$

$$\forall j\in J:\; G_j{\restriction}\mathsf{q} = \mu\mathbf{t}_j.T'_j \;\text{ implies }\; \begin{cases} \exists \mathbb{G}_j : \mathsf{q}\notin\mathbb{G}_j & \text{and} \\ G_j = \mathbb{G}_j[\mu\mathbf{t}_j.G'_i]^i_{i\in I_j} & \text{and} \\ \forall i\in I_j : G'_i{\restriction}\mathsf{q} \neq \mathbf{t}'\;(\forall \mathbf{t}') & \text{and} \\ T'_j = \bigsqcap_{i\in I_j}(G_i{\restriction}\mathsf{q}) \end{cases} \qquad \text{(by i.h.)} \qquad (99)$$

and letting $I = \bigcup_{j\in J} I_j$ and $\mathbb{G} = \mathsf{p}\to\mathsf{r}: \{\mathtt{m_j}(S_j)\,.\,\mathbb{G}_j\}_{j\in J}$, by (99) and (98) we get $G = \mathbb{G}[\mu\mathbf{t}.G'_i]^i_{i\in I}$ and the rest of the thesis, and we conclude;
- inductive case $G = \mu\mathbf{t}'.G'$. Then, we have:

$$\mathbf{t}' = \mathbf{t} \;\text{ and }\; T' = G'{\restriction}\mathsf{q} \neq \mathbf{t}''\;(\forall\mathbf{t}'') \qquad \text{(by (97) and Def. 3.3)} \qquad (100)$$

and letting $I = \{1\}$, $G_1 = G'$ and $\mathbb{G} = [\,]^1$, by (100) we get $G = \mathbb{G}[\mu\mathbf{t}.G_i]^i_{i\in I}$ and the rest of the thesis, and we conclude.

For the other items, we have:

$$\begin{array}{lll} G{\restriction}\mathsf{q} \text{ is defined} & \text{(by hypothesis)} & (101) \\ \text{either} \quad \mathsf{q}\notin G & \text{(by (101) and Prop. K.10)} & (102) \\ \text{or} \quad G = \mathbb{G}[G_i]^i_{i\in I'} \text{ with } \mathsf{q}\notin\mathbb{G} \text{ and for some } \mathsf{p}\text{, either:} & \text{(by (101) and Prop. K.10)} & (103) \\ \quad \forall i\in I' : G_i = \mathsf{p}\to\mathsf{q}: \{\mathtt{m_j}(S_j)\,.\,G'_j\}_{j\in J_i} & & (104) \\ \qquad \text{and} \quad G_i{\restriction}\mathsf{q} = \mathsf{p}\&_{j\in J_i}\mathtt{m}_j(S_j).S'_j \text{ where } S'_j = G'_j{\restriction}\mathsf{q} & \text{(by (104) and Def. 3.3)} & \\ \text{or} \quad \forall i\in I' : G_i = \mathsf{q}\to\mathsf{p}: \{\mathtt{m_j}(S_j)\,.\,G'_j\}_{j\in J_i} & & (105) \\ \qquad \text{and} \quad G_i{\restriction}\mathsf{q} = \mathsf{p}\oplus_{j\in J_i}\mathtt{m}_j(S_j).S'_j \text{ where } S'_j = G'_j{\restriction}\mathsf{q} & \text{(by (105) and Def. 3.3)} & \end{array}$$

Note that (101) rules out cases (A) and (B) of Prop. K.10, and thus leaves us with either (104) or (105).

Using (101) and (103), we rephrase the statement with $\mathbb{G}[G_i]^i_{i\in I'}$ in place of G, and proceed by structural induction on $\mathbb{G}$. The results follow by item (1) and Def. 3.3. □

Proposition K.13. *Assume* $\mathrm{dom}(\Gamma) = \{s\}$, *and* $\Gamma \to \Gamma_1 \to \Gamma_2 \to \cdots \to \Gamma_n = \Gamma$ *(with $n \geq 1$), and:*

$$\forall \Gamma',\Gamma'' : \left(\Gamma = \Gamma',\Gamma'' \;\textit{and}\; \left(\begin{array}{l}\exists\Gamma'_1,\ldots,\Gamma'_n : \Gamma' \to \Gamma'_1 \to \cdots \to \Gamma'_n = \Gamma' \\ \textit{and}\;\; \forall i\in 1..n : \Gamma_i = \Gamma'_i,\Gamma''\end{array}\right)\right) \;\textit{implies}\; \Gamma'' = \emptyset$$

Then, $\forall s[\mathsf{p}]{:}S \in \Gamma : S \not\leqslant \mathbf{end}$ *and* S *contains a recursive sub-term* $\mu\mathbf{t}.S'$ *(for some* $\mathbf{t}, S'$*).*

Proof. By contradiction, assume that:

$$\exists s[\mathsf{p}]{:}S \in \Gamma \;\text{ such that }\; S \leqslant \mathbf{end} \;\text{ or }\; S \text{ has no recursive sub-term} \qquad (106)$$

Then, we have two possibilities:

(a) if the entry $s[\mathsf{p}]{:}S$ interacts with other entries at least once along $\Gamma \to \Gamma_1 \to \cdots \to \Gamma$, then it cannot reduce into $s[\mathsf{p}]{:}S$ (by (106) and Def. 2.8); therefore, Γ cannot not reduce into itself, thus contradicting the hypothesis $\Gamma \to \Gamma_1 \to \cdots \to \Gamma$;

(b) otherwise, if the entry $s[\mathsf{p}]{:}S$ does *not* interact with other entries along $\Gamma \to \Gamma_1 \to \cdots \to \Gamma$, then we can find Γ'' (as defined in the statement) such that $s[\mathsf{p}]{:}S \in \Gamma''$, thus contradicting the hypothesis $\Gamma'' = \emptyset$.

Therefore, assuming (106) leads to a contradiction; hence, we conclude that $\forall s[\mathsf{p}]{:}S \in \Gamma : S \not\leqslant \mathbf{end}$ and $S = \mu\mathbf{t}.S'$ (for some $\mathbf{t}, S'$). □

Proposition K.14. *Assume that S has a recursive subterm $\mu\mathbf{t}.S'$, for some $\mathbf{t}$ and S'. Then:*

(1) if $\mathbf{t} \in \mathrm{fv}(S')$, *then* $\mu\mathbf{t}.S'$ *is a subterm of* $\mathrm{unf}^n(S)$ *(for all n);*
(2) $\forall \Gamma_0, S_0, \Gamma, s, \mathbf{p}$: *if* $\Gamma_0, s[\mathbf{p}]{:}S_0 \longrightarrow^* \Gamma, s[\mathbf{p}]{:}S$, *then* $\mu\mathbf{t}.S'$ *is a subterm of* S_0.

Moreover, for all S, *if* $\mu\mathbf{t}.S'$ *is a subterm of* $\mathrm{unf}^n(S)$ *(for some n), then* $\mu\mathbf{t}.S'$ *is also a subterm of* S.

Proof. Item (1): by structural induction on S, and then by induction on n.

Item (2): by induction on the number of reductions, showing that $\mu\mathbf{t}.S'$ is a subterm of the type of $s[\mathbf{p}]$, in each predecessor of $\Gamma, s[\mathbf{p}]{:}S$.

The *"moreover..."* part of the statement is proven by first showing that

$$\forall S_0 : \text{if } \mu\mathbf{t}.S' \text{ is a subterm of } \mathrm{unf}^1(S_0), \text{ then it is also a subterm of } S_0 \tag{107}$$

and then proceeding by induction on n, using (107) in the inductive case. □

Theorem K.15. *Assume* $\exists G, \Gamma : \mathrm{fproj}_{G,s}(\Gamma)$. *Then,* $\mathrm{dom}(\Gamma) = \{s\}$ *(Def. 2.6), and* $\mathrm{live}^+(\Gamma)$.

Proof. Assuming the hypotheses, we have (for some G):

$$\Gamma = \{s[\mathbf{p}]{:}G\!\upharpoonright\!\mathbf{p}\}_{\mathbf{p} \in \mathrm{roles}(G)} \qquad \text{(by Def. 5.8)} \tag{108}$$
$$\mathrm{dom}(\Gamma) = \{s\} \qquad \text{(by (108))} \tag{109}$$
$$\mathrm{live}(\Gamma) \qquad \text{(by Lemma K.9)} \tag{110}$$

Note that (109) proves the first part of the thesis. To prove the second part, let:

$$\varphi = \mathrm{beh}(\Gamma) \tag{111}$$

We now show that φ is a liveness$^+$ property. Therefore, we examine each element $(\Gamma', s[\mathbf{p}]{:}S) \in \varphi$, and we show that it satisfies all clauses of Fig. 5(6). We have the following (non-mutually exclusive) possibilities:

- $S = \mathbf{q}\&_{i \in I}\mathtt{m}_i(S_i).S'_i$. In this case, clauses [L-$\oplus^+$] and [L-μ^+] of Fig. 5(6) are vacuously satisfied, and we are left to prove clause [L-&$^+$]. The first part of such a clause (i.e., the part that matches clause [L-&] of Fig. 5(5)) holds by (110), (111) and Prop. K.3.
  We now prove the *"moreover..."* part of clause [L-&$^+$]. We need to prove that:

  $\Gamma', s[\mathbf{p}]{:}S$ belongs to some fair traversal set $\mathbb{X}$ (Def. 5.5) with targets $\mathbb{Y}$ such that, $\forall \Gamma_t \in \mathbb{Y}$, we have $\Gamma_t = \Gamma'', s[\mathbf{p}]{:}S'_k$ (for some $\Gamma'', k \in I$) (112)

  We proceed by contradiction, assuming that:

  there is no fair traversal set that contains $\Gamma', s[\mathbf{p}]{:}S$, and has targets $\mathbb{Y}$ such that, $\forall \Gamma_t \in \mathbb{Y}$, we have $\Gamma_t = \Gamma'', s[\mathbf{p}]{:}S'_k$ (for some $\Gamma'', k \in I$) (113)

  This means that there is Γ_0 such that:

  $$\Gamma_0 \subseteq \Gamma' \tag{114}$$
  $$\exists \Gamma'_0, \Gamma''_0, k \in I : \ \Gamma_0 \longrightarrow^* \Gamma'_0 \ \text{ and } \ \Gamma'_0, s[\mathbf{p}]{:}S \longrightarrow \Gamma''_0, s[\mathbf{p}]{:}S'_k \tag{115}$$
  $$\exists n \geq 1, \Gamma_{00}, \Gamma_{01}, \ldots, \Gamma_{0n} : \ \Gamma_0 \longrightarrow^* \Gamma_{00} \longrightarrow \Gamma_{01} \longrightarrow \cdots \longrightarrow \Gamma_{0n} \longrightarrow \Gamma_{00} \tag{116}$$

  i.e., Γ_0 is a subset of Γ' (114) that can interact with $s[\mathbf{p}]{:}S$ (115), but can also loop indefinitely without interacting with $s[\mathbf{p}]{:}S$ (116); we also pick Γ_0 to be minimal, in the sense that if we remove any entry, then (115) does not hold. The combination of minimality, (115) and (116) is due to liveness (110) and (113), that claims the impossibility to construct a fair traversal set with targets that always interact with $s[\mathbf{p}]{:}S$ (similarly to Ex. 5.6). Now, take Γ_{00} in (116), and partition it as $\Gamma_{00} = \Gamma_1, \Gamma_2$ such that:

$$\exists \Gamma_{10}, \Gamma_{11}, \ldots, \Gamma_{1n} : \ \Gamma_1 = \Gamma_{10} \longrightarrow \Gamma_{11} \longrightarrow \Gamma_{12} \longrightarrow \cdots \longrightarrow \Gamma_{1n} \longrightarrow \Gamma_{10} \ \text{ and } \forall i \in 0..n : \Gamma_{0i} = \Gamma_{1i}, \Gamma_2 \tag{117}$$

$$\forall \Gamma'_1, \Gamma''_1 : \ \left(\Gamma_1 = \Gamma'_1, \Gamma''_1 \text{ and } \left(\begin{array}{l}\exists \Gamma'_{10}, \ldots, \Gamma'_{1n} : \ \Gamma'_1 = \Gamma'_{10} \longrightarrow \Gamma'_{11} \longrightarrow \cdots \longrightarrow \Gamma'_{1n} \longrightarrow \Gamma'_{10} \\ \text{and } \forall i \in 0..n : \Gamma_{1i} = \Gamma'_{1i}, \Gamma''_1\end{array}\right)\right) \text{ implies } \Gamma''_1 = \emptyset \tag{118}$$

  i.e., we pick Γ_1, Γ_2, so that Γ_1 induces the reductions in (116), and reduces into itself, without interacting with Γ_2 (by (117)); moreover, we are picking Γ_1 so that it is minimal w.r.t. the reductions in (117), i.e., all its entries reduce at least once (by (118)). This implies that:

  $$\text{all entries of } \Gamma_1 \text{ contain a recursive subterm} \qquad \text{(by (117), (109), (118) and Prop. K.13)} \tag{119}$$

  Now, by (115) and (117), we have two mutually exclusive sub-cases:

(A) $\exists S'_{\mathsf{q}} : s[\mathsf{q}]{:}S'_{\mathsf{q}} \in \Gamma_1$. Then, we have:

$\exists \mathbb{S}$ such that $\mathsf{p} \notin \mathbb{S}$ and $\exists T_j (j \in J)$:

$$S'_{\mathsf{q}} \text{ has a subterm } S''_{\mathsf{q}} = \mu\mathbf{t}.\mathbb{S}[T_j]^{j}_{j\in J} \qquad \text{(by (119))} \qquad (120)$$

$$\exists k \in J : T_k = \mathsf{p}\oplus_{i\in I_k}\mathtt{m}_i(S_i).S'_i \qquad \text{(by (115), (117) and Def. 2.8)} \qquad (121)$$

$$\exists l \in J : \mathsf{p} \notin T_l \qquad \text{(by (117) and Def. 2.8)} \qquad (122)$$

i.e., S'_{q} has a recursive subterm (120) with some branch that interacts with $s[\mathsf{p}]{:}S$ (121), and some branch that doesn't interact with $s[\mathsf{p}]{:}S$ (122): the former exists by the liveness hypothesis (110), and the latter exists because otherwise Γ_1 could not loop without interacting with $s[\mathsf{p}]$ (hence, (117) would be contradicted). Therefore:

$$\nexists \mathbb{S}' : S''_{\mathsf{q}} = \mu\mathbf{t}.\mathbb{S}'[T_j]^{j}_{j\in J'} \text{ where } \begin{cases} \forall j \in J' : T_j = \mathsf{p}\oplus_{i\in I_{j'}}\mathtt{m}_i(S_i).S'_i \\ \text{or} \\ \forall j \in J' : T_j = \mathsf{p}\&_{i\in I_{j'}}\mathtt{m}_i(S_i).S'_i \end{cases} \qquad \text{(by (120), (121) and (122))} \qquad (123)$$

Now, observe:

$$\exists S_{\mathsf{p}}, S_{\mathsf{q}} \text{ such that } S_{\mathsf{p}} = \Gamma(s[\mathsf{p}]) = G{\upharpoonright}\mathsf{p} \text{ and } S_{\mathsf{q}} = \Gamma(s[\mathsf{q}]) = G{\upharpoonright}\mathsf{q} \qquad \text{(by (108))} \qquad (124)$$

$$S''_{\mathsf{q}} \text{ is a subterm of } S_{\mathsf{q}} \qquad \text{(by (120), Def. K.1, (124) and Prop. K.14)} \qquad (125)$$

$$\exists G' : \mu\mathbf{t}.G' \text{ is a subterm of } G \text{ and } S''_{\mathsf{q}} = (\mu\mathbf{t}.G'){\upharpoonright}\mathsf{q} \qquad \text{(by (125), Prop. K.12(1) and Def. 3.3)} \qquad (126)$$

$$\mathsf{p} \in S'_{\mathsf{q}} \text{ and therefore } \mathsf{p} \in \mu\mathbf{t}.G' \qquad \text{(by (120), (121), (126) and Def. 3.3)} \qquad (127)$$

and note that in G' there must be:

(A-1) a branch where the first interaction of p is either $\mathsf{p}{\to}\mathsf{q}$ or $\mathsf{q}{\to}\mathsf{p}$ (since, by hypothesis, $s[\mathsf{p}]{:}S$ is waiting for q, and by Prop. K.12); and

(A-2) a branch where p does *not* interact with q (otherwise, by Def. 3.3, we could not project S'_{q} according to (126) and (123)).

But then:

$$\nexists \mathbb{G}, \mathsf{q}', I, G_i\ (i \in I) : \begin{cases} \mu\mathbf{t}.G' = \mu\mathbf{t}.\mathbb{G}[G_i]^{i}_{i\in I} \\ \mathsf{p} \notin \mathbb{G} \\ \text{and either } \begin{cases} \forall i \in I : G_i = \mathsf{p}{\to}\mathsf{q}'{:}\left\{\mathtt{m}_j(S_j)\,.\,G'_j\right\}_{j\in J_i} \\ \text{or} \\ \forall i \in I : G_i = \mathsf{q}'{\to}\mathsf{p}{:}\left\{\mathtt{m}_j(S_j)\,.\,G'_j\right\}_{j\in J_i} \end{cases} \end{cases} \qquad \text{(by (A-1) and (A-2))} \qquad (128)$$

which implies:

$$(\mu\mathbf{t}.G'){\upharpoonright}\mathsf{p} \text{ is undefined} \qquad \text{(by (128), (127), and the contrapositive of Cor. K.11)} \qquad (129)$$

$$G{\upharpoonright}\mathsf{p} \text{ is undefined} \qquad \text{(by (129), (126) and Def. 3.3)} \qquad (130)$$

and (130) contradicts (124), and thus, the hypothesis (108);

(B) $s[\mathsf{q}] \in \mathrm{dom}(\Gamma_2)$. Then, we have two more sub-cases:

(a) Γ_2 can first interact with Γ_1, and then with $s[\mathsf{p}]{:}S$. More formally:

$$\exists k \in I, \Gamma'_1, \Gamma'_2 : \quad \Gamma_1, \Gamma_2 \longrightarrow\!\!\to^{*} \Gamma'_1, \Gamma'_2 \text{ and } \Gamma'_2, s[\mathsf{p}]{:}S \to \Gamma'_2, s[\mathsf{p}]{:}S'_k \qquad (131)$$

where in the first sequence of reductions, an entry of Γ_2 (say, $s[\mathsf{r}']{:}S'_{\mathsf{r}'}$) interacts with an entry of Γ_1 (say, $s[\mathsf{r}]{:}S'_{\mathsf{r}}$). But then, we can apply the same reasoning of case (A) above, with the following adaptations:

(i) use r' in place of q;

(ii) use r in place of p;

(iii) if S'_{r} is a (possibly recursive) external choice, let T_k in (121) be an external choice, too.

Then, the adaptation of (130) says that $G{\upharpoonright}\mathsf{r}$ is undefined, contradicting the hypothesis (108);

(b) Γ_2 can first interact with $s[\mathsf{p}]{:}S$, and then with Γ_1. More formally:

$$\exists k \in I, \Gamma'_1, \Gamma'_2, \Gamma''_2, \Gamma_3, \Gamma'_3, S'', \mathsf{r}, S_{\mathsf{r}}, S'_{\mathsf{r}}, S''_{\mathsf{r}} : \begin{cases} s[\mathsf{r}] \in \mathrm{dom}(\Gamma_2) \quad \text{and} \\ \Gamma_2, s[\mathsf{p}]{:}S \longrightarrow\!\!\to^{*} \Gamma_3, s[\mathsf{r}]{:}S_{\mathsf{r}}, s[\mathsf{p}]{:}S'_k \to^{*} \Gamma'_3, s[\mathsf{r}]{:}S'_{\mathsf{r}}, s[\mathsf{p}]{:}S'' \quad \text{and} \\ \Gamma_1, s[\mathsf{r}]{:}S'_{\mathsf{r}} \longrightarrow\!\!\to^{*} \Gamma'_1, s[\mathsf{r}]{:}S''_{\mathsf{r}} \end{cases} \qquad (132)$$

where in the last sequence of reductions, $s[\mathbf{r}]{:}S'_{\mathbf{r}}$ interacts with some element of Γ_1 — say, $s[\mathbf{r}']{:}S'_{\mathbf{r}'}$. But then, we can use the same strategy described in case (a) above: i.e., apply the reasoning of case **(A)** with the adaptations *(i)*, *(ii)* and *(iii)*, obtaining the same contradiction;

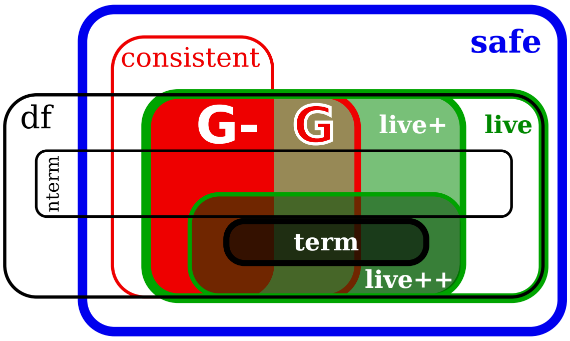
To recap: in all cases above, assuming (113) leads to a contradiction, hence we obtain that its negation (112) holds. Therefore, we conclude that $\Gamma', s[\mathbf{p}]{:}S$ satisfies clause [L-&$^+$] of Fig. 5(6);

- $S = \mathbf{q}\oplus_{i\in I}\mathsf{m}_i(S_i).S'_i$. In this case, clauses [L-&$^+$] and [L-μ^+] of Fig. 5(6) are vacuously satisfied, and we are left to prove clause [L-$\oplus^+$]. Its proof is similar to the previous case.
- $S = \mu\mathbf{t}.S'$. In this case, clauses [L-$\oplus^+$] and [L-&$^+$] of Fig. 5(6) are vacuously satisfied, and we are left to prove clause [L-μ^+] — which holds by Equation (111) and Prop. K.2;
- $S = \mathbf{end}$. In this case, clauses [L-$\oplus^+$], [L-&$^+$] and [L-μ^+] hold vacuously;
- $\Gamma', s[\mathbf{p}]{:}S \to \Gamma''$. In this case, we also need to satisfy clause [L-$\to$] of Fig. 5(6) — which holds by Equation (111) and Prop. K.2.

Summing up, we have proven that if we take any G whose projections are defined, we can also define φ as in (111), and prove that it is a liveness$^+$ property. Moreover, $\{s[\mathbf{p}]{:}G{\upharpoonright}\mathbf{p}\}_{\mathbf{p}\in\text{roles}(G)} \in \varphi$ (by (108), (111) and Def. K.1); and since live$^+$ is the largest liveness$^+$ property (by Fig. 5(6)), we have $\varphi \subseteq$ live$^+$, and thus, $\{s[\mathbf{p}]{:}G{\upharpoonright}\mathbf{p}\}_{\mathbf{p}\in\text{roles}(G)} \in$ live$^+$. Therefore, by Def. 5.8 we conclude that if $\exists G, \Gamma : \text{fproj}_{G,s}(\Gamma)$, then live$^+(\Gamma)$. □

Lemma 5.9. *For all* Γ*, the following (non-)implications hold:*

(1) $\text{consistent}(\Gamma) \;\not\Longleftarrow\!\!\Longrightarrow\; \text{safe}(\Gamma)$*;*
(2) $\text{live}(\Gamma) \;\not\Longleftarrow\!\!\Longrightarrow\; \text{safe}(\Gamma)$*;*
(3) $\text{live}(\Gamma) \;\not\Longleftarrow\!\!\Longrightarrow\; \text{df}(\Gamma)$*;*
(4) $\text{nterm}(\Gamma) \;\not\Longleftarrow\!\!\Longrightarrow\; \text{df}(\Gamma)$*;*
(5) $\text{consistent}(\Gamma) \;\not\Longleftarrow\!\!\not\Longrightarrow\; \text{df}(\Gamma)$*;*
(6) $\text{consistent}(\Gamma) \wedge \text{df}(\Gamma) \;\not\Longleftarrow\!\!\not\Longrightarrow\; \text{live}(\Gamma)$*;*
(7) $\text{live}^{++}(\Gamma) \;\not\Longleftarrow\!\!\Longrightarrow\; \text{live}^{+}(\Gamma) \;\not\Longleftarrow\!\!\Longrightarrow\; \text{live}(\Gamma)$*;*
(8) $\text{term}(\Gamma) \;\not\Longleftarrow\!\!\Longrightarrow\; \text{live}^{++}(\Gamma)$*;*
(9) assume $\text{dom}(\Gamma) = \{s\}$ *(Def. 2.6). Then:*
$\exists G : \text{fproj}_{G,s}(\Gamma) \;\not\Longleftarrow\!\!\Longrightarrow\; \text{live}^+(\Gamma)$.



Proof. The negated implications in the statement are proved in Table 1 and Ex. 5.11.
We now examine the remaining implications.

1. Assume Γ consistent, and take the property $\varphi = \text{beh}(\Gamma)$ (Def. K.1). By contradiction, assume that Γ is not safe. Then, by Prop. K.3, φ must contain some Γ' such that $\Gamma \to^* \Gamma'$ and Γ' violates clause [S-$\oplus$&] (possibly after applying [S-μ] to unfold its entries). By Def. 3.8, such Γ' is not consistent. But then, by the contrapositive of Prop. K.5, we obtain that Γ is not consistent — contradiction. We conclude that Γ is safe.
2. Straightforward by Fig. 5(5).
3. Straightforward by Fig. 5(5).
4. Straightforward by Fig. 5(4).
7. Straightforward by Fig. 5(6) and Fig. 5(7).
8. By contradiction, assume that Γ is *not* live^{++}. Then, there is Γ' such that $\Gamma \to^* \Gamma'$, and Γ' (once unfolded) violates clause [L-&$^{++}$]/[L-$\oplus^{++}$] of Fig. 5(7) — i.e., $\Gamma' = \Gamma'', s[\mathbf{p}]{:}S$ (for some Γ''), where S is a branching/selection type that is *not* triggered within a finite number of steps by a corresponding selection/branching along the reductions of Γ''. But then, there is no guarantee that, in a finite number of steps, Γ' will reduce to some Γ''' such that end(Γ'''); and since $\Gamma \to^* \Gamma'$, there is no such guarantee for Γ, either. This implies that Γ does *not* satisfy Fig. 5(3) — contradiction. Therefore, we conclude that Γ is live^{++}.
9. Direct consequence of Thm. K.15.

□

Theorem 5.13 (Decidability of φ). $\varphi(\Gamma)$ *is decidable, for all* Γ*, and for all* φ *such that*

$$\varphi \in \{\text{consistent}, \text{fproj}_{G,s}, \text{pproj}_{G,s}, \text{safe}, \text{term}, \text{nterm}, \text{df}, \text{live}, \text{live}^+, \text{live}^{++}\} \quad \textit{(for any } G\textit{)}$$

Proof. If φ = consistent, $\Gamma \in \varphi$ is decidable because, by Def. 3.8, it is sufficient to check (at most) all pairs of types contained in Γ (which are finite), using partial projection (that always terminates), duality and $\leqslant$ (that are both decidable).

For the other cases, observe that for any Γ, the transitive closure of the typing context reduction relation $\longrightarrow$ (Def. 2.8) induces a finite-state transition system.

When $\varphi = \text{fproj}_{G,s}$ or $\varphi = \text{pproj}_{G,s}$, observe the two possible definitions of the set $\mathcal{E}$ in Def. 5.8: in both cases, the projection of any G always terminates, either by being undefined (then $\mathcal{E}$ is empty, φ is empty, and $\varphi(\Gamma)$ is false) or by returning some Γ, from which Def. K.1 collects all reachable typing contexts, that are finite. Then, notice that $\text{unf}^*(\mathcal{E})$ in Def. K.1 is the least fixed point of a function that is monotonic (w.r.t. the partial order $\subseteq$), and therefore, can be computed with a chain of successive applications starting with $\emptyset$ (by the Knaster-Tarski theorem [Tarski 1955]) — and since $\mathcal{E}$ is finite, this procedure always terminates by yielding the (finite) set of typing contexts with all combinations of all unfoldings of all elements of all typing contexts contained in $\mathcal{E}$. Hence, φ is finite, and therefore, $\Gamma \in \varphi$ is trivially decidable.

The rest of the cases are decidable because it is straightforward to produce an algorithm that inspects all (finite) elements of the behavioural set $\text{beh}(\Gamma)$ (Def. K.1), verifying whether they satisfy the clauses of Def. 4.1, Fig. 5(2), Fig. 5(5) or Fig. 5(6). □

L ASYNCHRONOUS SESSION FIDELITY

Proposition L.1 (Asynchronous Normal Form). *For all P, $P \equiv \textbf{def}\ \widetilde{D}\ \textbf{in}\ (\nu\widetilde{s})\,P_1 \mid \ldots \mid P_n$, where $\forall i \in 1..n$, P_i is either a branching, a selection, or a process call, or a session queue.*

Proof. Minor adaptation of Prop. I.2. □

Lemma B.1 (substitution) is unchanged in async MPST, it is only applied to processes occurring as premises of [TA-Lift]. Instead, Lemma B.2 (subject congruence) needs to be adapted as Lemma L.2 below. The difference w.r.t. the synchronous result is that, in the asynchronous setting, if $P \equiv P'$ then session queues might be reordered by $\equiv$ (Fig. 7), hence queue types might need reordering by $\equiv$ (Def. D.2).

Lemma L.2 (Async Subject Congruence). *Assume $\Theta \cdot \Gamma \vdash_{\mathcal{S}} P$ and $P \equiv P'$. Then, $\exists \Gamma'$ such that $\Gamma \equiv \Gamma'$ and $\Theta \cdot \Gamma' \vdash_{\mathcal{S}} P'$.*

Proof. The proof is similar to that of Lemma B.2, and in all corresponding cases we have $\Gamma = \Gamma'$. We have two additional cases, corresponding to the two queue congruence rules in Def. C.1:

- when $P = (\nu s{:}\Gamma'')\, s \blacktriangleright \sigma \equiv \mathbf{0} = P'$, we must have $\text{end}(\Gamma'')$ and $\text{end}(\Gamma)$, and we conclude with $\Gamma' = \Gamma$ by typing rule [T-$\mathbf{0}$];
- when $P \equiv P'$ holds by the order-swapping congruence on queues, we apply the same reordering on the queue types of Γ, getting a congruent typing context Γ' that satisfies the statement.

□

Proposition L.3. *If $\mathbf{p} \in \text{senders}(\sigma)$, then $\forall \Theta, \Gamma, s$: $\Theta \cdot \Gamma \vdash_{\{s\}} s \blacktriangleright \sigma$ implies $\Gamma(s[\mathbf{p}]) = M$, for some $M \neq \epsilon$.*

Proof. Assume $\mathbf{p} \in \text{senders}(\sigma)$. Then:

$$\exists \sigma', \sigma'' : \quad \sigma = \sigma' \cdot (\mathbf{p}, \mathbf{q}, \mathtt{m}\langle s'[\mathbf{r}]\rangle) \cdot \sigma'' \qquad \text{(by hypothesis and Def. C.1)} \tag{133}$$

$$\dfrac{\Theta \cdot \Gamma_{\sigma''} \vdash_{\{s\}} s \blacktriangleright \sigma'' \qquad \Gamma' \vdash s'[\mathbf{r}]{:}S}{\Theta \cdot (\Gamma_{\sigma''} \leftsquigarrow s[\mathbf{p}]{:}\mathbf{q}!\mathtt{m}(S)\cdot\epsilon), \Gamma' \vdash_{\{s\}} s \blacktriangleright (\mathbf{p}, \mathbf{q}, \mathtt{m}\langle s[\mathbf{r}]\rangle) \cdot \sigma''}\,[\text{TA-}\sigma] \qquad \text{(by (133) and induction on } \sigma''\text{)} \tag{134}$$

From (134), we proceed with a further induction on σ' (from (133)), to prove that $\exists \Gamma$ such that $\Theta \cdot \Gamma \vdash_{\{s\}} s \blacktriangleright \sigma$, with $\Gamma(s[\mathbf{p}]) = M$ for some $M \neq \epsilon$:

- base case $\sigma' = \epsilon$. Then, $\sigma = \sigma' \cdot (\mathbf{p}, \mathbf{q}, \mathtt{m}\langle s'[\mathbf{r}]\rangle) \cdot \sigma'' = (\mathbf{p}, \mathbf{q}, \mathtt{m}\langle s'[\mathbf{r}]\rangle) \cdot \sigma''$: we conclude by (134) and Fig. 8, letting $\Gamma = (\Gamma_{\sigma''} \leftsquigarrow s[\mathbf{p}]{:}\mathbf{q}!\mathtt{m}(S)\cdot\epsilon), \Gamma')$;
- inductive case $\sigma' = (\mathbf{p}', \mathbf{q}', \mathtt{m}'\langle s''[\mathbf{r}']\rangle) \cdot \sigma'''$. Then:

$$\exists \Gamma'' : \ \Theta \cdot \Gamma'' \vdash_{\{s\}} s \blacktriangleright \sigma''' \cdot (\mathbf{p}, \mathbf{q}, \mathtt{m}\langle s'[\mathbf{r}]\rangle) \cdot \sigma'' \ \text{ with } \Gamma''(s[\mathbf{p}]) = M' \text{ for some } M' \neq \epsilon \qquad \text{(by i.h.)} \tag{135}$$

$$\dfrac{\Theta \cdot \Gamma'' \vdash_{\{s\}} s \blacktriangleright \sigma''' \cdot (\mathbf{p}, \mathbf{q}, \mathtt{m}\langle s'[\mathbf{r}]\rangle) \cdot \sigma'' \qquad \Gamma''' \vdash s''[\mathbf{r}']{:}S'''}{\Theta \cdot (\Gamma'' \leftsquigarrow s'[\mathbf{p}']{:}\mathbf{q}'!\mathtt{m}'(S''')\cdot\epsilon), \Gamma''' \vdash_{\{s\}} s \blacktriangleright \sigma}\,[\text{TA-}|] \qquad \text{(by (135) and (133))} \tag{136}$$

and we conclude by (136) and Fig. 8, letting $\Gamma = (\Gamma'' \leftsquigarrow s'[\mathbf{p}']{:}\mathbf{q}'!\mathtt{m}'(S''')\cdot\epsilon), \Gamma'''$.

□

PROPOSITION L.4. *If* $\mathbf{p} \notin \mathrm{senders}(\sigma)$, *then* $\forall \Theta, \Gamma, s : \Theta \cdot \Gamma \vdash_{\{s\}} s \blacktriangleright \sigma$ *implies* $s[\mathbf{p}] \notin \mathrm{dom}(\Gamma)$.

PROOF. Assume $\mathbf{p} \notin \mathrm{senders}(\sigma)$. We proceed by induction on σ:

- base case $\sigma = \epsilon$. Then, we have $\Gamma = \emptyset$ (by inversion of [TA-ϵ]), and we conclude immediately;
- inductive case $\sigma = (\mathbf{p}', \mathbf{q}, \mathtt{m}\langle s[\mathbf{r}]\rangle) \cdot \sigma'$. Observe that $\mathbf{p}' \neq \mathbf{p}$ and $\mathbf{p} \notin \mathrm{senders}(\sigma')$ (otherwise, we would have the contradiction $\mathbf{p} \in \mathrm{senders}(\sigma)$). By the i.h., $\forall \Theta', \Gamma', s : \Theta' \cdot \Gamma' \vdash_{\{s\}} s \blacktriangleright \sigma'$ implies $s[\mathbf{p}] \notin \mathrm{dom}(\Gamma')$. By [TA-$\sigma$] and Fig. 8, we conclude that $\forall \Theta, \Gamma, s : \Theta \cdot \Gamma \vdash_{\{s\}} s \blacktriangleright \sigma$ implies $s[\mathbf{p}] \notin \mathrm{dom}(\Gamma)$.

□

Using Lemma B.1 (plus Lemma L.2) we extend Thm. B.4 (session inversion) to asynchrony, obtaining the new Thm. L.5 below: the difference w.r.t. Thm. B.4 are the new items (5)–(7), showing how queue types shape process queues.

THEOREM L.5 (ASYNC SESSION INVERSION). *Assume* $\emptyset \cdot \Gamma \vdash_{\mathcal{S}} |_{\mathbf{p} \in I} P_{\mathbf{p}}$ *with each* $P_{\mathbf{p}}$ *either being* $\mathbf{0}$ *(up-to* $\equiv$*), or only playing role* $\mathbf{p}$ *in* s*. Then,* $\Gamma = \{s[\mathbf{p}]{:}S_{\mathbf{p}}\}_{\mathbf{p} \in I'}, \{s[\mathbf{p}]{:}M_{\mathbf{p}}\}_{\mathbf{p} \in I''}$ *for some* I', I''*. Moreover,* $\forall \mathbf{p} \in I'$*, we have items (1) , (2) , (3) from Thm. B.4.*
Further, we have item (4) from Thm. B.4, and (5) $\forall \mathbf{p} \in I \setminus I'' : \mathbf{p} \notin \mathrm{senders}(\sigma)$.
Finally, $\forall \mathbf{p} \in I''$*: (6)* $\exists \mathbf{q}, \mathtt{m}, T, M' : M_{\mathbf{p}} = \mathbf{q}!\mathtt{m}(T) \cdot M'$*; (7)* $\exists s', \mathbf{r}, \sigma' : \sigma \equiv (\mathbf{p}, \mathbf{q}, \mathtt{m}\langle s'[\mathbf{r}]\rangle) \cdot \sigma'$.

PROOF. The proof is similar to that of Thm. B.4, using the asynchronous MPST normal form of Prop. L.1, and observing that, by inversion of [TA-|], we must have $\mathcal{S} = \{s\}$. We examine the additional items, involving the session queue.

Item (5). Holds by the contrapositive of Prop. L.3.
Item (6). By the contrapositive of Prop. L.4, we know that $\mathbf{p} \in \mathrm{senders}(\sigma)$; then, we conclude by Prop. L.3;
Item (7). By the contrapositive of Prop. L.4, we know that $\mathbf{p} \in \mathrm{senders}(\sigma)$; then, we conclude by Def. C.1.

□

THEOREM F.8 (ASYNC SESSION FIDELITY). *Let* $\Theta \cdot \Gamma \vdash_{\mathcal{S}} P$*, with* $P \equiv \left(|_{\mathbf{p} \in I} P_{\mathbf{p}}\right) \mid s \blacktriangleright \sigma$*, and each* $P_{\mathbf{p}}$ *either being* $\mathbf{0}$ *(up-to* $\equiv$*), or only playing role* $\mathbf{p}$ *in* s*. Then,* $\Gamma \rightarrow_{\mathcal{S}}$ *implies* $\exists \Gamma', P'$ *such that* $\Gamma \rightarrow_{\mathcal{S}} \Gamma'$*,* $P \rightarrow^* P'$ *and* $\Theta \cdot \Gamma' \vdash_{\mathcal{S}} P'$*, with* $P' \equiv \left(|_{\mathbf{p} \in I} P'_{\mathbf{p}}\right) \mid s \blacktriangleright \sigma'$ *and each* $P'_{\mathbf{p}}$ *either being* $\mathbf{0}$ *(up-to* $\equiv$*), or only playing role* $\mathbf{p}$ *in* s*.*

PROOF. Similar to the proof of Thm. 5.4, but using Thm. L.5 for asynchronous session inversion. □

M SUBJECT REDUCTION FOR ASYNCHRONOUS MPST

REMARK M.1. *If we want to explicitly instantiate the safety property* φ *for a typing derivation that restricts the multiparty sessions* $s_1{:}\Gamma_1, \ldots, s_n{:}\Gamma_n$*, then we can (1) take a set* $\{\varphi_i\}_{i \in 1..n}$ *where* φ_i *is an* $\{s_i\}$*-safety property such that* $\varphi_i(\Gamma_i)$*, and (2) instantiate Def. F.4 with* $\varphi = \bigcup_{i \in 1..n} \varphi_i$*. By construction,* φ *is an* $\{s_i\}$*-safety property such that* $\varphi(\Gamma_i)$ $(i \in 1..n)$*.*

LEMMA M.2. *If* $\text{a-safe}_{\mathcal{S}}(\Gamma)$ *and* $\Gamma \leqslant \Gamma'$*, then* $\text{a-safe}_{\mathcal{S}}(\Gamma')$*.*

PROOF. Similar to Lemma 4.5, noticing that by Def. D.1, subtyping of asynchronous typing contexts only involves session types (not queue types). □

LEMMA M.3. *If* Γ *safe and* $\Gamma \leqslant \Gamma' \rightarrow_{\mathcal{S}} \Gamma''$*, then there is* Γ''' *such that* $\Gamma \rightarrow_{\mathcal{S}} \Gamma''' \leqslant \Gamma''$*.*

PROOF. Similar to Lemma 4.4. □

LEMMA M.4 (NARROWING (ASYNCHRONOUS)). *If* $\Theta \cdot \Gamma \vdash_{\mathcal{S}} P$ *and* $\Gamma' \leqslant \Gamma$ *with* $\text{a-safe}_{\mathcal{S}}(\Gamma')$*, then* $\Theta \cdot \Gamma' \vdash_{\mathcal{S}} P$*.*

PROOF. Similar to Lemma B.3, noticing that by Def. D.1, subtyping of asynchronous typing contexts only involves session types (not queue types). □

PROPOSITION M.5. *Assume* $\Theta \cdot \Gamma \vdash_{\{s\}} s \blacktriangleright \sigma$ *with* $s'[\mathbf{r}] \notin \mathrm{dom}(\Gamma)$*,* $s[\mathbf{p}] \notin \mathrm{dom}(\Gamma)$ *and* $\Gamma' \vdash s'[\mathbf{r}]{:}S$*. Then, we have the judgement* $\Theta \cdot \Gamma, s[\mathbf{p}]{:}\mathbf{q}!\mathtt{m}(S) \cdot \epsilon, \Gamma' \vdash_{\{s\}} s \blacktriangleright \sigma \cdot (\mathbf{p}, \mathbf{q}, \mathtt{m}\langle s'[\mathbf{r}]\rangle)$.

Proof. Assume $\Theta\cdot\Gamma \vdash_{\{s\}} s\blacktriangleright\sigma$ with $s[\mathsf{p}] \notin \mathrm{dom}(\Gamma)$ and $s'[\mathsf{r}] \notin \mathrm{dom}(\Gamma)$, and $\Gamma' \vdash s'[\mathsf{r}]{:}S$. Notice that:

$$\dfrac{\dfrac{}{\Theta\cdot\emptyset \vdash_{\{s\}} s\blacktriangleright\epsilon}\,[\text{TA-}\epsilon] \qquad \Gamma' \vdash s[\mathsf{r}]{:}S}{\Theta\cdot s[\mathsf{p}]{:}\mathsf{q}!\mathtt{m}(S){\cdot}\epsilon, \Gamma' \vdash_{\{s\}} s\blacktriangleright(\mathsf{p},\mathsf{q},\mathtt{m}\langle s[\mathsf{r}]\rangle)\cdot\epsilon}\,[\text{TA-}\sigma] \tag{137}$$

Now, let n be the length of σ, and let:

- $\Gamma_0 = \emptyset$;
- $\sigma_0 = \epsilon$;
- $\Gamma_n = \Gamma$;
- $\sigma_n = \sigma$;
- $\forall i \in 1..n:\quad \sigma_i = (\mathsf{p}_i,\mathsf{q}_i,\mathtt{m}_i\langle s_i[\mathsf{r}_i]\rangle)\cdot\sigma_{i-1}$ and $\Gamma'_i \vdash s_i[\mathsf{r}_i]{:}S_i$ and $\Gamma_i = (\Gamma_{i-1}{\leftrightsquigarrow}s[\mathsf{p}_i]{:}\mathsf{q}_i!\mathtt{m}_i(s_i[\mathsf{r}_i]){\cdot}\epsilon), \Gamma'_i$.

Then, the derivation of $\Theta\cdot\Gamma \vdash_{\{s\}} s\blacktriangleright\sigma$ has the following shape:

$$\dfrac{\dfrac{\vdots \qquad \Gamma'_{n-1} \vdash s_{n-1}[\mathsf{r}_{n-1}]{:}S_{n-1}}{\Theta\cdot\Gamma_{n-1} \vdash_{\{s\}} s\blacktriangleright\sigma_{n-1}}\,[\text{TA-}\sigma] \qquad \Gamma'_n \vdash s_n[\mathsf{r}_n]{:}S_n}{\Theta\cdot\Gamma_n \vdash_{\{s\}} s\blacktriangleright\sigma_n}\,[\text{TA-}\sigma] \tag{138}$$

where the top of the derivation is

$$\dfrac{\dfrac{\dfrac{}{\Theta\cdot\Gamma_0 \vdash_{\{s\}} s\blacktriangleright\sigma_0}\,[\text{TA-}\epsilon] \qquad \Gamma'_1 \vdash s_1[\mathsf{r}_1]{:}S_1}{\Theta\cdot\Gamma_1 \vdash_{\{s\}} s\blacktriangleright\sigma_1}\,[\text{TA-}\sigma] \qquad \Gamma'_2 \vdash s_2[\mathsf{r}_2]{:}S_2}{\vdots}\,[\text{TA-}\sigma]$$

We can rewrite the derivation in (138) into a derivation for $\Theta\cdot\Gamma, s[\mathsf{p}]{:}\mathsf{q}!\mathtt{m}(S){\cdot}\epsilon, \Gamma' \vdash_{\{s\}} s\blacktriangleright\sigma\cdot(\mathsf{p},\mathsf{q},\mathtt{m}\langle s'[\mathsf{r}]\rangle)$, proceeding by induction on n:

(1) first, $\forall i \in 0..n$, rewrite $\Theta\cdot\Gamma_i \vdash_{\{s\}} s\blacktriangleright\sigma_i$ as $\Theta\cdot\Gamma_i, s[\mathsf{p}]{:}\mathsf{q}!\mathtt{m}(S){\cdot}\epsilon, \Gamma' \vdash_{\{s\}} s\blacktriangleright\sigma_i\cdot(\mathsf{p},\mathsf{q},\mathtt{m}\langle s[\mathsf{r}]\rangle)\cdot\epsilon$. Notice that the rewritten typing context is defined, since $\mathrm{dom}(\Gamma') = \{s'[\mathsf{r}]\} \not\subseteq \mathrm{dom}(\Gamma_i)$ and $s[\mathsf{p}] \notin \mathrm{dom}(\Gamma_i)$ (by hypothesis);
(2) then, graft (137) on top of the derivation, noticing that the conclusion of (137) matches the rewriting of $\Theta\cdot\Gamma_0 \vdash_{\{s\}} s\blacktriangleright\sigma_0$ after step 1 above.

We have thus obtained a typing derivation that proves the statement. □

Proposition M.6. *Assume* $\Theta\cdot\Gamma \vdash_{\{s\}} s\blacktriangleright\sigma$, *with* $s'[\mathsf{r}] \notin \mathrm{dom}(\Gamma)$, $\Gamma(s[\mathsf{p}]) = M$ *(for some* M*) and* $\Gamma' \vdash s'[\mathsf{r}]{:}S$. *Then, we have* $\Theta\cdot\Gamma\{M\cdot\mathsf{q}!\mathtt{m}(S)\cdot\epsilon/s[\mathsf{p}]\}, \Gamma' \vdash_{\{s\}} s\blacktriangleright\sigma\cdot(\mathsf{p},\mathsf{q},\mathtt{m}\langle s'[\mathsf{r}]\rangle)$.

Proof. The proof is similar to that of Prop. M.5. The only difference is that, in step 1 of the rewriting, the observation "$\forall i \in 0..n : s[\mathsf{p}] \notin \mathrm{dom}(\Gamma_i)$" does *not* hold. Hence, we use the following rewriting, for all $i \in 1..n$:

$$\Theta\cdot\Gamma_i \vdash_{\{s\}} s\blacktriangleright\sigma_i \qquad\mapsto\qquad \Theta\cdot(s[\mathsf{p}]{:}\mathsf{q}!\mathtt{m}(S){\cdot}\epsilon{\rightsquigarrow}\Gamma_i), \Gamma' \vdash_{\{s\}} s\blacktriangleright\sigma_i\cdot(\mathsf{p},\mathsf{q},\mathtt{m}\langle s[\mathsf{r}]\rangle)\cdot\epsilon$$

where:

$$s[\mathsf{p}]{:}M''{\rightsquigarrow}\Gamma'' = \begin{cases} \Gamma''\{\Gamma''(s[\mathsf{p}])\cdot M''/s[\mathsf{p}]\} & \text{if } s[\mathsf{p}] \in \mathrm{dom}(\Gamma'') \\ \Gamma''\{M''/s[\mathsf{p}]\} & \text{otherwise} \end{cases}$$

As a result, in the rewritten derivation, the type $\mathsf{q}!\mathtt{m}(S)$ for the additional queue message $(\mathsf{p},\mathsf{q},\mathtt{m}\langle s'[\mathsf{r}]\rangle)$ is added at the end of $\Gamma(s[\mathsf{p}])$ from the original typing context[7]. The rewritten derivation proves the statement. □

Lemma M.7 (Queueing/Dequeueing Typability). *Assume* $\Theta\cdot\Gamma \vdash_{\mathcal{S}} P$ *with* Γ $\mathcal{S}$*-safe. Then:*

(1) if $P = s[\mathsf{p}][\mathsf{q}]\oplus\mathtt{m}\langle s'[\mathsf{q}']\rangle.P' | s\blacktriangleright\sigma$, *then* $\exists\Gamma'$ $\mathcal{S}$*-safe such that* $\Gamma \to_{\mathcal{S}} \Gamma'$ *and* $\Theta\cdot\Gamma' \vdash_{\mathcal{S}} P' \mid s\blacktriangleright\sigma\cdot(\mathsf{p},\mathsf{q},\mathtt{m}\langle s'[\mathsf{q}']\rangle)$;
(2) if $P = s[\mathsf{q}][\mathsf{p}]\sum_{i\in I}\mathtt{m}_i(x_i).P'_i \mid s\blacktriangleright(\mathsf{p},\mathsf{q},\mathtt{m}\langle s'[\mathsf{q}']\rangle)\cdot\sigma$ *then there exist* $k \in I$ *and* Γ' $\mathcal{S}$*-safe such that* $\mathtt{m} = \mathtt{m}_k$, $\Gamma \to_{\mathcal{S}} \Gamma'$ *and* $\Theta\cdot\Gamma' \vdash_{\mathcal{S}} P'_k\{s'[\mathsf{q}']/x_k\} \mid s\blacktriangleright\sigma$.

[7]Note that the same rewriting also allows to prove Prop. M.5, when $s[\mathsf{p}] \notin \mathrm{dom}(\Gamma)$. For clarity, we chose to keep Propositions M.5 and M.6 separate, with the rewriting in the proof of Prop. M.5 as simple as possible.

Proof. **Item 1.** We have:

$$\begin{array}{l}\Gamma = \Gamma_{\oplus}, \Gamma_{\sigma}\,, \\ \mathcal{S} = \mathcal{S}_{\oplus} \cup \mathcal{S}_{\sigma}\end{array} \quad \text{and} \quad \dfrac{\Theta \cdot \Gamma_{\oplus} \vdash_{\mathcal{S}_{\oplus}} s[\mathbf{p}][\mathbf{q}]\oplus\mathtt{m}\langle s'[\mathbf{q}']\rangle.P' \quad \Theta \cdot \Gamma_{\sigma} \vdash_{\mathcal{S}_{\sigma}} s \blacktriangleright \sigma \quad \mathcal{S}_{\oplus} \cap \mathcal{S}_{\sigma} = \emptyset}{\Theta \cdot \Gamma \vdash_{\mathcal{S}} P}\,[\text{TA-}|] \qquad \text{(inv. of [TA-|])} \tag{139}$$

$$\mathcal{S}_{\oplus} = \emptyset \quad \text{and} \quad \dfrac{\dfrac{\Theta \cdot \Gamma_{\oplus} \vdash s[\mathbf{p}][\mathbf{q}]\oplus\mathtt{m}\langle s'[\mathbf{q}']\rangle.P'}{\Theta \cdot \Gamma_{\oplus} \vdash_{\mathcal{S}_{\oplus}} s[\mathbf{p}][\mathbf{q}]\oplus\mathtt{m}\langle s'[\mathbf{q}']\rangle.P'}\,[\text{TA-Lift}]}{} \qquad \text{(by (139) and inv. of [TA-Lift])} \tag{140}$$

$$\Gamma_{\oplus} = \Gamma_0, \Gamma_1, \Gamma_2 \quad \text{and} \quad \dfrac{\Gamma_1 \vdash s[\mathbf{p}]{:}\mathbf{q}{\oplus}\mathtt{m}(S).S' \quad \Gamma_2 \vdash s'[\mathbf{q}']{:}S \quad \Theta \cdot \Gamma_0, s[\mathbf{p}]{:}S' \vdash P'}{\Theta \cdot \Gamma_{\oplus} \vdash s[\mathbf{p}][\mathbf{q}]\oplus\mathtt{m}\langle s'[\mathbf{q}']\rangle.P'}\,[\text{T-}\oplus] \qquad \text{(by (140) and inv. [T-}\oplus\text{])} \tag{141}$$

$$\Gamma_2 \vdash s'[\mathbf{q}']{:}S \quad \text{and} \quad s'[\mathbf{q}'] \notin \operatorname{dom}(\Gamma_{\sigma}) \qquad \text{(by (139) and (141))} \tag{142}$$

$$\exists \Gamma'' = \Gamma_0, s[\mathbf{p}]{:}\mathbf{q}{\oplus}\mathtt{m}(S).S', \Gamma_2, \Gamma_{\sigma} \quad \text{and} \quad \Gamma \leqslant \Gamma'' \qquad \text{(by (139), (141), [T-Sub] and Def. 2.6)} \tag{143}$$

We now have two cases, that we study in order to prove the existence of a suitable Γ''' such that $\Gamma'' \to_{\mathcal{S}} \Gamma'''$:

- $\mathbf{p} \in \text{senders}(\sigma)$. Then:

$$\exists M \neq \epsilon : \; \Gamma_{\sigma}(s[\mathbf{p}]) = M \qquad \text{(by (139) and Prop. L.3)} \tag{144}$$

$$\Gamma'_{\sigma} = \Gamma_{\sigma}\{M{\cdot}\mathbf{q}!\mathtt{m}(S){\cdot}\epsilon / s[\mathbf{p}]\}, \Gamma_2 \;\text{ and }\; \Theta \cdot \Gamma'_{\sigma} \vdash_{\mathcal{S}_{\sigma}} s \blacktriangleright \sigma \cdot (\mathbf{p}, \mathbf{q}, \mathtt{m}\langle s'[\mathbf{q}']\rangle) \qquad \text{(by (139), (144), (142) and Prop. M.6)} \tag{145}$$

$$\exists \Gamma''' = \Gamma_0, s[\mathbf{p}]{:}S', \Gamma'_{\sigma} \;\text{ such that }\; \Gamma'' \to \Gamma''' \qquad \text{(by (143), (145) and Def. D.4)} \tag{146}$$

$$\Gamma'' \to_{\mathcal{S}} \Gamma''' \qquad \text{(by (143) and Def. F.1)} \tag{147}$$

- $\mathbf{p} \notin \text{senders}(\sigma)$. Then:

$$s[\mathbf{p}] \notin \operatorname{dom}(\Gamma_{\sigma}) \qquad \text{(by (139) and Prop. L.4)} \tag{148}$$

$$\Gamma'_{\sigma} = \Gamma_{\sigma}, s[\mathbf{p}]{:}\mathbf{q}!\mathtt{m}(S){\cdot}\epsilon, \Gamma_2 \;\text{ and }\; \Theta \cdot \Gamma'_{\sigma} \vdash_{\mathcal{S}_{\sigma}} s \blacktriangleright \sigma \cdot (\mathbf{p}, \mathbf{q}, \mathtt{m}\langle s'[\mathbf{q}']\rangle) \qquad \text{(by (139), (148), (142) and Prop. M.5)} \tag{149}$$

$$\exists \Gamma''' = \Gamma_0, s[\mathbf{p}]{:}S', \Gamma'_{\sigma} \;\text{ such that }\; \Gamma'' \to_{\mathcal{S}} \Gamma''' \qquad \text{(by (139), (141), (149) and Def. F.1)} \tag{150}$$

Therefore, for both cases above, using Γ'_{σ} from either (145) or (149), and Γ''' from either (146) or (150), we obtain:

$$\text{a-safe}_{\mathcal{S}}(\Gamma''') \qquad \text{(by a-safe}_{\mathcal{S}}(\Gamma)\text{, (143), Lemma M.2, (147)/(150) and Def. F.2, clause [SA-}\to\text{])} \tag{151}$$

$$\dfrac{\dfrac{\Theta \cdot \Gamma_0, s[\mathbf{p}]{:}S' \vdash P'}{\Theta \cdot \Gamma_0, s[\mathbf{p}]{:}S' \vdash_{\mathcal{S}_{\oplus}} P'}\,[\text{TA-Lift}] \quad \Theta \cdot \Gamma'_{\sigma} \vdash_{\mathcal{S}_{\sigma}} s \blacktriangleright \sigma \cdot (\mathbf{p}, \mathbf{q}, \mathtt{m}\langle s'[\mathbf{q}']\rangle) \quad \mathcal{S}_{\oplus} \cap \mathcal{S}_{\sigma} = \emptyset}{\Theta \cdot \Gamma''' \vdash_{\mathcal{S}} P' \mid s \blacktriangleright \sigma \cdot (\mathbf{p}, \mathbf{q}, \mathtt{m}\langle s'[\mathbf{q}']\rangle)}\,[\text{TA-}|] \qquad \text{(by (141), (145)/(149), (139) and (146)/(150), (151))} \tag{152}$$

$$\exists \Gamma' : \; \Gamma \to_{\mathcal{S}} \Gamma' \;\text{ and }\; \Gamma' \leqslant \Gamma''' \qquad \text{(by (143), (147)/(150) and Lemma M.3)} \tag{153}$$

$$\text{a-safe}_{\mathcal{S}}(\Gamma') \qquad \text{(by a-safe}_{\mathcal{S}}(\Gamma)\text{, (153) and Def. F.2, clause [SA-}\to\text{])} \tag{154}$$

$$\Theta \cdot \Gamma' \vdash_{\mathcal{S}} P' \mid s \blacktriangleright \sigma \cdot (\mathbf{p}, \mathbf{q}, \mathtt{m}\langle s'[\mathbf{q}']\rangle) \qquad \text{(by (152), (153), (154) and Lemma M.4)} \tag{155}$$

Hence, we conclude the proof by (153), (154) and (155).

Item 2. We have:

$$\Gamma = \Gamma_{\&}, \Gamma_{\sigma},\ \mathcal{S} = \mathcal{S}_{\&} \cup \mathcal{S}_{\sigma} \quad \text{and} \quad \dfrac{\Theta \cdot \Gamma_{\&} \vdash_{\mathcal{S}_{\&}} s[\mathtt{q}][\mathtt{p}]\sum_{i\in I}\mathtt{m}_i(x_i).P'_i \quad \Theta \cdot \Gamma_{\sigma} \vdash_{\mathcal{S}_{\sigma}} s \blacktriangleright (\mathtt{p},\mathtt{q},\mathtt{m}\langle s'[\mathtt{q}']\rangle)\cdot\sigma \quad \mathcal{S}_{\&} \cap \mathcal{S}_{\sigma} = \emptyset}{\Theta \cdot \Gamma \vdash_{\mathcal{S}} P}\,[\text{TA-}|] \quad \text{(inv. of [TA-|])} \tag{156}$$

$$\mathcal{S}_{\&} = \emptyset \quad \text{and} \quad \dfrac{\Theta \cdot \Gamma_{\&} \vdash s[\mathtt{q}][\mathtt{p}]\sum_{i\in I}\mathtt{m}_i(x_i).P'_i}{\Theta \cdot \Gamma_{\&} \vdash_{\mathcal{S}_{\&}} s[\mathtt{q}][\mathtt{p}]\sum_{i\in I}\mathtt{m}_i(x_i).P'_i}\,[\text{TA-Lift}] \quad \text{(by (156) and inv. of [TA-Lift])} \tag{157}$$

$$\Gamma_{\&} = \Gamma_0, \Gamma_1 \quad \text{and} \quad \dfrac{\Gamma_1 \vdash s[\mathtt{q}]:\mathtt{p}\&_{i\in I}\mathtt{m}_i(S_i).S'_i \quad \forall i \in I \quad \Theta \cdot \Gamma_0, x_i{:}S_i, s[\mathtt{q}]{:}S'_i \vdash P'_i}{\Theta \cdot \Gamma_{\&} \vdash s[\mathtt{q}][\mathtt{p}]\sum_{i\in I}\mathtt{m}_i(x_i).P'_i}\,[\text{T-}\&] \quad \text{(by (157) and inv. [T-\&])} \tag{158}$$

$$\mathcal{S}_{\sigma} = \{s\} \text{ and } \Gamma_{\sigma} = (\Gamma'_{\sigma} \leftsquigarrow s[\mathtt{p}]{:}\mathtt{q}!\mathtt{m}(S)\cdot\epsilon), \Gamma_2 \text{ and } \dfrac{\Theta \cdot \Gamma'_{\sigma} \vdash_{\mathcal{S}_{\sigma}} s \blacktriangleright \sigma \quad \Gamma_2 \vdash s'[\mathtt{q}']{:}S}{\Theta \cdot \Gamma_{\sigma} \vdash_{\mathcal{S}_{\sigma}} s \blacktriangleright (\mathtt{p},\mathtt{q},\mathtt{m}\langle s'[\mathtt{q}']\rangle)\cdot\sigma}\,[\text{TA-}\sigma] \quad \text{(by (156), inv. [TA-}\sigma\text{])} \tag{159}$$

$$\exists k \in I: \quad \mathtt{m} = \mathtt{m}_k \text{ and } S \leqslant S_k \qquad \text{(by (156), (158), (159), a-safe}_{\mathcal{S}}(\Gamma)\text{ and clauses [SA-\&!]/[SA-}\mu\text{] of Def. F.2)} \tag{160}$$

$$\Gamma_2 \vdash s'[\mathtt{q}']{:}S_k \qquad \text{(by (159), (160) and transitivity of } \leqslant) \tag{161}$$

$$\exists \Gamma_k = \Gamma_0, s[\mathtt{q}]{:}S'_k, \Gamma_2 \text{ defined} \qquad \text{(by (156), (158) and (159))} \tag{162}$$

$$\Theta \cdot \Gamma_k \vdash P'_k\{s'[\mathtt{q}']/x_k\} \qquad \text{(by (161), (158), (162) and Lemma B.1)} \tag{163}$$

$$\exists \Gamma'' = s[\mathtt{p}]{:}\mathtt{p}\&_{i\in I}\mathtt{m}_i(S_i).S'_i, \Gamma_1, \Gamma_{\sigma} \text{ and } \Gamma \leqslant \Gamma'' \qquad \text{(by (156), (158), (159), [T-Sub] and Def. 2.6)} \tag{164}$$

$$\exists \Gamma''' = \Gamma_k, \Gamma'_{\sigma} \qquad \text{(by (162), (159))} \tag{165}$$

We now have two cases, that we study in order to prove that $\Gamma'' \to_{\mathcal{S}} \Gamma'''$:

- $\mathtt{p} \in \text{senders}(\sigma)$. Then:

$$\Gamma'' \to \Gamma''' \qquad \text{(by (164), (165) and Def. D.4)} \tag{166}$$

$$\Gamma'' \to_{\mathcal{S}} \Gamma''' \qquad \text{(by (166) and Def. F.1)} \tag{167}$$

- $\mathtt{p} \notin \text{senders}(\sigma)$. Then:

$$s[\mathtt{p}] \notin \text{dom}(\Gamma'_{\sigma}) \qquad \text{(by (159) and Prop. L.4)} \tag{168}$$

$$\Gamma'''(s[\mathtt{p}]) = S'_k \qquad \text{(by (156), (158), (165) and (168))} \tag{169}$$

$$\Gamma_{\sigma}(s[\mathtt{p}]) = \mathtt{q}!\mathtt{m}(S)\cdot\epsilon \qquad \text{(by (159) and Fig. 8)} \tag{170}$$

$$\Gamma''(s[\mathtt{p}]) = (\mathtt{q}!\mathtt{m}(S)\cdot\epsilon; \mathtt{p}\&_{i\in I}\mathtt{m}_i(S_i).S'_i) \qquad \text{(by (164) and (170))} \tag{171}$$

$$\Gamma'' \to_{\mathcal{S}} \Gamma''' \qquad \text{(by (171), (169) and Def. F.1)} \tag{172}$$

Therefore, for both cases above, using either (167) or (172), we obtain:

$$\text{a-safe}_{\mathcal{S}}(\Gamma''') \qquad \text{(by a-safe}_{\mathcal{S}}(\Gamma)\text{, (164), Lemma M.2, (167)/(172) and Def. F.2, clause [SA-}\to\text{])} \tag{173}$$

$$\dfrac{\dfrac{\Theta \cdot \Gamma_k \vdash P'_k\{s'[\mathtt{q}']/x_k\}}{\Theta \cdot \Gamma_k \vdash_{\mathcal{S}_{\&}} P'_k\{s'[\mathtt{q}']/x_k\}}\,[\text{TA-Lift}] \quad \Theta \cdot \Gamma'_{\sigma} \vdash_{\mathcal{S}_{\sigma}} s \blacktriangleright \sigma}{\Theta \cdot \Gamma''' \vdash_{\mathcal{S}} P'_k\{s'[\mathtt{q}']/x_k\} \mid s \blacktriangleright \sigma}\,[\text{TA-}|] \quad \text{(by (163), (156), (157), (159), (165), (173))} \tag{174}$$

$$\exists \Gamma': \ \Gamma \to_{\mathcal{S}} \Gamma' \text{ and } \Gamma' \leqslant \Gamma''' \qquad \text{(by (164), (167)/(172) and Lemma M.3)} \tag{175}$$

$$\text{a-safe}_{\mathcal{S}}(\Gamma') \qquad \text{(by a-safe}_{\mathcal{S}}(\Gamma)\text{, (175) and Def. F.2, clause [SA-}\to\text{])} \tag{176}$$

$$\Theta \cdot \Gamma' \vdash_{\mathcal{S}} P'_k\{s'[\mathtt{q}']/x_k\} \mid s \blacktriangleright \sigma \qquad \text{(by (174), (175), (176) and Lemma M.4)} \tag{177}$$

Hence, we conclude the proof by (175), (176) and (177). □

Lemma F.5. *Let* a-safe$_{\mathcal{S}}(\Gamma)$*: then,* a-safe$_{\mathcal{S}\setminus s}(\Gamma)$*; and if* $\Gamma = \Gamma', s[\mathtt{p}]{:}S$*, then* a-safe$_{\mathcal{S}}(\Gamma')$.

Proof. Assume a-safe$_{\mathcal{S}}(\Gamma)$: it means that all $\to^*_{\mathcal{S}}$-reductions of Γ are safe (by Def. F.2, clause [SA-$\to$]). Note that, among such safe reductions of Γ, there is also the subset of all its $\to_{\mathcal{S}\setminus s}^*$-reductions (by Def. F.1): hence, by Def. F.2, we conclude a-safe$_{\mathcal{S}\setminus s}(\Gamma)$ — which proves the first part of the statement.

For the second part of the statement, assume a-safe$_{\mathcal{S}}(\Gamma', s[\mathtt{p}]{:}S)$, and by contradiction, also assume that Γ' is *not* $\mathcal{S}$-safe. Observe that by hypothesis and Def. D.2, Γ' can (possibly) map $s[\mathtt{p}]$ to a queue type, but *not* to a session type, nor to a session/queue type. Hence, by Def. F.1, Γ' cannot violate Def. F.2 due to new messages enqueued by $s[\mathtt{p}]$, nor due to messages sent to $s[\mathbf{p}]$. But then, the same violations of Def. F.2 can also be found by letting $\Gamma', s[\mathbf{p}]{:}S$ reduce, which therefore is *not* $\mathcal{S}$-safe — contradiction. Hence, we conclude $\text{a-safe}_{\mathcal{S}}(\Gamma)$. □

Proposition M.8. *Assume* $\Theta \cdot \Gamma, s[\mathbf{p}]{:}M \vdash_{\mathcal{S}} P$*. Then,* $s \in \mathcal{S}$ *and* $P \equiv P_0 \mid s \blacktriangleright \sigma$

Proof. By induction on the typing derivation, observing that the queue type can only be yielded by rule [TA-σ] (Fig. 8), which in turn requires $s \in \mathcal{S}$. □

Theorem F.6 (Asynchronous Subject Reduction). *Assume* $\Theta \cdot \Gamma \vdash_{\mathcal{S}} P$ *with* Γ $\mathcal{S}$*-safe. Then,* $P \to P'$ *implies* $\exists \Gamma'$ $\mathcal{S}$*-safe such that* $\Gamma \to_{\mathcal{S}}^{*} \Gamma'$ *and* $\Theta \cdot \Gamma' \vdash_{\mathcal{S}} P'$*.*

Proof. By induction of the derivation of $P \to P'$, and when the reduction holds by rule [R-Ctx], with a further structural induction on the reduction context $\mathbb{C}$. Most cases hold by inversion of the typing $\Theta \cdot \Gamma \vdash P$, and by applying the induction hypothesis,

The most complex cases are the base cases where $P \to P'$ is due to rules [R-AOut] or [R-AIn], and messages are added or removed from the session queue. Such cases are proved in Lemma M.7 above.

In the inductive case where $P = \mathbb{C}[P_0]$ with $\mathbb{C} = \mathbb{C}' \mid Q$, we have:

$$\exists P_1 : \ \mathbb{C}'[P_0] \to \mathbb{C}'[P_1] \quad \text{and} \quad P' = \mathbb{C}'[P_1] \mid Q \qquad \text{(by inversion of [R-Ctx])} \qquad (178)$$

$$\begin{array}{l} \mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \ \text{ and} \\ \Gamma = \Gamma_1, \Gamma_2 \ \text{ such that} \end{array} \ \dfrac{\Theta \cdot \Gamma_1 \vdash_{\mathcal{S}_1} \mathbb{C}'[P_0] \qquad \Theta \cdot \Gamma_2 \vdash_{\mathcal{S}_2} Q \qquad \mathcal{S}_1 \cap \mathcal{S}_2 = \emptyset}{\Theta \cdot \Gamma \vdash_{\mathcal{S}} P}\,[\text{TA-}|] \qquad \text{(by (178), inv. [TA-|])} \qquad (179)$$

$$\text{a-safe}_{\mathcal{S}_1}(\Gamma_1, \Gamma_2) \qquad \text{(by a-safe}_{\mathcal{S}}(\Gamma)\text{, (179) and Lemma F.5)} \qquad (180)$$

Now, our intermediate goal is to prove:

$$\text{a-safe}_{\mathcal{S}_1}(\Gamma_1, \Gamma_2) \ \text{ implies } \ \text{a-safe}_{\mathcal{S}_1}(\Gamma_1) \qquad (181)$$

For this purpose, we proceed by induction on Γ_2:

- base case $\Gamma_2 = \emptyset$. Then, $\Gamma_1 = \Gamma_1, \Gamma_2$, and we conclude trivially by (180);
- inductive case $\Gamma_2 = \Gamma_2', c{:}\tau$. Then, we have:

$$\text{a-safe}_{\mathcal{S}_1}(\Gamma_1, c{:}\tau) \qquad \text{(by the induction hypothesis on (181))} \qquad (182)$$

  and the following sub-cases:
  - $c = x$ (i.e., c is a variable). Then, observe that $\mathcal{S}_1$-safety (Def. F.2) only depends on typing context entries mapping channels with roles, and ignores any entry mapping variables: hence, from (182), we conclude $\text{a-safe}_{\mathcal{S}_1}(\Gamma_1)$;
  - $c = s[\mathbf{p}]$. Then, we have the following possibilities:
    * $\tau = S$. Then, by Lemma F.5, we conclude $\text{a-safe}_{\mathcal{S}_1}(\Gamma_1)$;
    * $\tau = M$. Then, by (179) and Prop. M.8, Q contains the queue for session s, and $s \in \mathcal{S}_2$. Hence, by (179), $s \notin \mathcal{S}_1$, i.e., $\mathcal{S}_1 \setminus \{s\} = \mathcal{S}_1$; from this, by Lemma F.5, we conclude $\text{a-safe}_{\mathcal{S}_1}(\Gamma_1)$;
    * $\tau = (M; S)$. Similar to the previous case.

We have thus proved (181), and therefore:

$$\text{a-safe}_{\mathcal{S}_1}(\Gamma_1) \qquad \text{(by (180) and (181))} \qquad (183)$$

$$\exists \Gamma_1' \ \mathcal{S}_1\text{-safe such that } \Gamma_1 \to_{\mathcal{S}_1}^{*} \Gamma_1' \ \text{ and } \ \Theta \cdot \Gamma_1' \vdash_{\mathcal{S}_1} \mathbb{C}'[P_1] \qquad \text{(by (179), (183), (178), i.h.)} \qquad (184)$$

$$\exists \Gamma' = \Gamma_1', \Gamma_2 \ \text{ such that } \ \Gamma \to_{\mathcal{S}}^{*} \Gamma' \qquad \text{(by (179) and (184))} \qquad (185)$$

$$\text{a-safe}_{\mathcal{S}}(\Gamma') \qquad \text{(by a-safe}_{\mathcal{S}}(\Gamma)\text{, (185) and Def. F.2, clause [SA-}\to\text{])} \qquad (186)$$

$$\dfrac{\Theta \cdot \Gamma_1' \vdash_{\mathcal{S}_1} \mathbb{C}'[P_1] \quad \Theta \cdot \Gamma_2 \vdash_{\mathcal{S}_2} Q \quad \mathcal{S}_1 \cap \mathcal{S}_2 = \emptyset}{\Theta \cdot \Gamma' \vdash_{\mathcal{S}} P'}\,[\text{TA-}|] \qquad \text{(by (179), (184), (185) and (178))} \qquad (187)$$

and we conclude by (186) and (187). □

Corollary F.7 (Async Type Safety). *If* $\emptyset \cdot \emptyset \vdash_{\emptyset} P$ *and* $P \to^{*} P'$*, then* P' *has no errors.*

Proof. Similar to Cor. 4.9. □

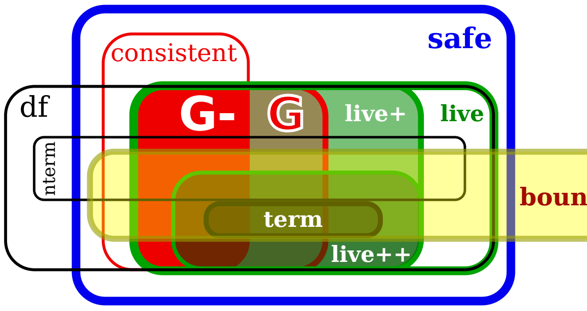
Theorem 7.2. *If* φ *is decidable, then "*$\Theta{\cdot}\Gamma \vdash_{\mathcal{S}} P$ *with* φ*" is decidable.*

Proof. Similar to Thm. 4.11. □

N ASYNCHRONOUS TYPING CONTEXT PROPERTIES

Lemma G.3. *For all* Γ, *letting* $\mathbb{S} = \{s \mid \exists \mathsf{p} : s[\mathsf{p}] \in \mathrm{dom}(\Gamma)\}$, *we have:*

(1) $\text{a-consistent}(\Gamma) \;\;\not\Longleftarrow\!\!\Longrightarrow\;\; \text{a-safe}_{\mathbb{S}}(\Gamma)$;
(2) $\text{a-live}_{\mathbb{S}}(\Gamma) \;\;\not\Longleftarrow\!\!\Longrightarrow\;\; \text{a-safe}_{\mathbb{S}}(\Gamma)$;
(3) $\text{a-live}_{\mathbb{S}}(\Gamma) \;\;\not\Longleftarrow\!\!\Longrightarrow\;\; \text{a-df}_{\mathbb{S}}(\Gamma)$;
(4) $\text{a-nterm}_{\mathbb{S}}(\Gamma) \;\;\not\Longleftarrow\!\!\Longrightarrow\;\; \text{a-df}_{\mathbb{S}}(\Gamma)$;
(5) $\text{a-consistent}(\Gamma) \;\;\not\Longleftarrow\!\!\not\Longrightarrow\;\; \text{a-df}_{\mathbb{S}}(\Gamma)$;
(6) $\text{a-consistent}(\Gamma) \wedge \text{a-df}_{\mathbb{S}}(\Gamma) \;\;\not\Longleftarrow\!\!\not\Longrightarrow\;\; \text{a-live}_{\mathbb{S}}(\Gamma)$;
(7) $\text{a-term}_{\mathbb{S}}(\Gamma) \;\;\not\Longleftarrow\!\!\Longrightarrow\;\; \text{a-live}^{++}_{\mathbb{S}}(\Gamma)$;
(8) $\text{a-term}_{\mathbb{S}}(\Gamma) \;\;\not\Longleftarrow\!\!\Longrightarrow\;\; \text{a-bound}_{\mathbb{S}}(\Gamma)$;
(9) $\text{a-bound}_{\mathbb{S}}(\Gamma) \;\;\not\Longleftarrow\!\!\not\Longrightarrow\;\; \text{a-safe}_{\mathbb{S}}(\Gamma) \vee \text{a-df}_{\mathbb{S}}(\Gamma)$;
(10) $\text{a-live}^{++}_{\mathbb{S}}(\Gamma) \;\;\not\Longleftarrow\!\!\Longrightarrow\;\; \text{a-live}^{+}_{\mathbb{S}}(\Gamma) \;\;\not\Longleftarrow\!\!\Longrightarrow\;\; \text{a-live}_{\mathbb{S}}(\Gamma)$.



Proof. The negated implications in the statement are proved in Ex. G.4. The remaining implications are similar to those in Lemma 5.9, using the corresponding asynchronous definitions. □

Lemma N.1. *Assume* $\mathrm{dom}(\Gamma) = \{s\}$ *and* $\mathrm{live}(\Gamma)$. *Then, for all* synchronous *(i.e., queue-less) typing contexts* Γ':

(1) $\Gamma \to^* \Gamma'$ *implies* $\Gamma \to_{\{s\}}^* \Gamma'$;
(2) $\Gamma \to_{\{s\}}^* \Gamma''$ *implies* $\exists \Gamma' : \Gamma'' \to_{\{s\}}^* \Gamma'$ *and* $\Gamma \to^* \Gamma'$.

Proof. *(Item 1)* We first prove the statement for one reduction step, i.e.:

$$\forall \Gamma, \Gamma' : \quad \Gamma \to \Gamma' \quad \text{implies} \quad \Gamma \to_{\{s\}} \to_{\{s\}} \Gamma' \tag{188}$$

where the two-step reduction represents a message being queued, and then immediately consumed — which always possible by the hypothesis $\mathrm{live}(\Gamma)$. Then, we prove the main statement by induction on the number of reductions in $\Gamma \to^* \Gamma'$, using (188) for the inductive step.

(Item 2) Consequence of [Deniélou and Yoshida 2013, Thm 4.1] and [Bocchi et al. 2015, Thm. 6], that say (roughly): if Γ is live, then queued outputs are eventually consumed, and external choices are eventually triggered. More in detail: if Γ'' contains queued messages, we can let Γ'' reduce by consuming each queued message, thus reaching a queue-less typing context that is the desired Γ'; then, we can reorder the transitions in $\Gamma \to_{\{s\}}^* \Gamma'' \to_{\{s\}}^* \Gamma'$ into a sequence of alternating queuing/dequeuing transitions (as in (188)) — which always possible by the hypothesis $\mathrm{live}(\Gamma)$; the resulting alternating queueing/dequeueing reductions give a corresponding synchronous reduction $\Gamma \to^* \Gamma'$. □

Lemma N.2. *Assume* $\mathrm{dom}(\Gamma) = \{s\}$ *and* $\mathrm{live}(\Gamma)$. *Then,* $\text{a-live}_{\{s\}}(\Gamma)$.

Proof. Let us define $\varphi = \text{a-beh}_{\{s\}}(\Gamma)$ similarly to Def. K.1, but using *asynchronous* reductions $\to_{\mathbb{S}}$. Then, we prove that φ is an asynchronous liveness property, by using Lemma N.1(2) to show that active external choices can be triggered (clause [LA-&] of Def. G.2(4)), and queued messages can be consumed (clause [LA-!] of Def. G.2(4)). Finally, by Def. G.2(4), we conclude $\text{a-live}_{\{s\}}(\Gamma)$. □

Lemma N.3. *Assume* $\mathrm{dom}(\Gamma) = \{s\}$ *and* $\mathrm{live}^+(\Gamma)$. *Then,* $\text{a-live}^+_{\{s\}}(\Gamma)$.

Proof. We use φ as in the proof of Lemma N.2, that we know is an $\{s\}$-liveness property by Lemma 5.9(7) and Lemma N.2. In order to prove the *"moreover..."* clauses of $\{s\}$-liveness$^+$, we also use the following result, that follows by Lemma N.1(2):

Proposition N.4. *If* $\mathrm{live}(\Gamma)$ *and* $\Gamma \to_{\{s\}}^* \Gamma_0, \Gamma_1$ *and* $\Gamma_0 \to_{\{s\}} \to_{\{s\}}^* \Gamma_0$, *there are* Γ'_0, Γ'_1 *queue-less, such that* $\Gamma_0 \to_{\{s\}}^* \Gamma'_0 \to\to^* \Gamma'_0$ *and* $\Gamma_1 \to_{\{s\}}^* \Gamma'_1$ *and* $\Gamma \to^* \Gamma'_0, \Gamma'_1$

i.e., if Γ is live and produces a loop under asynchronous semantics, then it also produces a corresponding loop under *synchronous* semantics.

Then, to prove the *"moreover..."* parts of clauses [LA-&$^+$]/[LA-!$^+$] of Def. G.2(5), we proceed by contradiction, similarly to the proof of Thm. K.15: we assume that there is no asynchronous traversal set that satisfies the requirements of Def. G.2(5) (similarly to step (113)); this means that Γ can perform asynchronous reduction loops that, by Prop. N.4, imply the existence of corresponding *synchronous* reduction loops, whose form matches the one described in steps (114)–(116). This leads to the contradiction that Γ is *not* live$^+$. Therefore, clauses [LA-&$^+$]/[LA-!$^+$] of Def. G.2(5) hold, and we conclude $\text{a-live}^+_{\{s\}}(\Gamma)$. □