Less Is More: Multiparty Session Types Revisited

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

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

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

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

This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2019 Copyright held by the owner/author(s).
2475-1421/2019/1-ART30
https://doi.org/10.1145/3290343

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*:

(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**:

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

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:

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

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 $\mathbf{c}\oplus\mathtt{m}(\mathsf{Str})\,.\,\mathbf{end}$ and $\mathbf{s}\&\mathtt{m}(\mathsf{Int})\,.\,\mathbf{end}$, 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 technical report [Scalas and Yoshida 2018a] contains more technical details, proofs, and discussion on related work.*

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:

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

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** $\mathbf{0}$ 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** $\mathbf{def}\ X(\widetilde{x}) = P\ \mathbf{in}\ Q$ 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: $\mathbb{C} ::= \mathbb{C} \mid P \mid (\nu s)\, \mathbb{C} \mid \mathbf{def}\ D\ \mathbf{in}\ \mathbb{C} \mid [\,]$

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

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

$$\text{[R-Ctx]}\quad P\to P' \ \text{ implies } \ \mathbb{C}[P]\to\mathbb{C}[P']$$

$$\text{[R-Err]}\quad s[\mathbf{p}][\mathbf{q}]\textstyle\sum_{i\in I}\mathtt{m}_i(x_i).P_i \mid s[\mathbf{q}][\mathbf{p}]\oplus\mathtt{m}\langle s'[\mathbf{r}]\rangle.Q \;\to\; \mathbf{err} \quad \text{if } \forall i\in I:\mathtt{m}_i\neq\mathtt{m}$$

Fig. 1. MPST π-calculus semantics, defined up-to standard structural congruence [Scalas and Yoshida 2018a].

Reduction $\to$ is inductively defined in Fig. 1, up-to a standard **structural congruence** $\equiv$ [Scalas and Yoshida 2018a] including α-conversion. We say that P **has an error** iff, for some $\mathbb{C}$, $P=\mathbb{C}[\mathbf{err}]$.

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 $\mathbf{p}$ and $\mathbf{q}$, 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 $s_i[\mathbf{p}_i]$. 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 $\mathbf{err}$: intuitively, it models a scenario where a process implementing role $\mathbf{q}$ is trying to send $\mathtt{m}$ to another process implementing $\mathbf{p}$ — 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 $s[\mathbf{s}]$, $s[\mathbf{c}]$, $s[\mathbf{a}]$, to play resp. roles $\mathbf{s}$, $\mathbf{c}$, $\mathbf{a}$. For brevity, we omit irrelevant message payloads.

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

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

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

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:

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

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

The **branching type** (or **external choice**) $\mathbf{p}\&_{i\in I}\mathtt{m}_i(S_i).S_i'$ says that a channel must be used to receive from $\mathbf{p}$ one input of the form $\mathtt{m}_i(S_i)$, for any $i\in I$ chosen by $\mathbf{p}$, 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**) $\mathbf{p}\oplus_{i\in I}\mathtt{m}_i(S_i).S_i'$, instead, requires to use a channel to perform one output $\mathtt{m}_i(S_i)$ towards $\mathbf{p}$, for some $i\in I$, and continue using the channel according

to S'_i. Type **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 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:

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

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

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:

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

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 $\forall c \in \mathrm{dom}(\Gamma) : \Gamma(c) \leqslant \Gamma'(c)$.

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], $\Theta \vdash X{:}S_1, \ldots, S_n$ holds if Θ maps X to an n-tuple of types $S_1, \ldots, S_n$. 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 $x_1, \ldots, x_n$ according to $S_1, \ldots, S_n$, and the latter

[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].

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

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

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

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

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

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

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

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:

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

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:

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

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

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

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

In the rest of the paper, we will mostly use the unlabelled reduction $\Gamma \to \Gamma'$, 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:

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

Intuitively, the problem of this typing judgement can be seen in its typing context: the type of $s[\mathbf{p}]$ outputs $\mathtt{foo}$ to $\mathbf{q}$, but the type of $s[\mathbf{q}]$ 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):

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

To achieve type safety, we want the process in (7) to be untypable — which means that, when type-checking $(vs{:}\Gamma)\ldots$, 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 $s[\mathbf{p}]$ sends a message to $\mathbf{q}$, then the type of $s[\mathbf{q}]$ 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 $(vs)\,P$, 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 [2018b]: 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* $S_{\mathbf{p}}\upharpoonright\mathbf{q}$ and $S_{\mathbf{q}}\upharpoonright\mathbf{p}$, 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**:

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

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:

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

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 $G\upharpoonright\mathbf{s}$, $G\upharpoonright\mathbf{c}$, $G\upharpoonright\mathbf{a}$, 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 $S_{\mathbf{s}}\upharpoonright\mathbf{a}$ and $S_{\mathbf{a}}\upharpoonright\mathbf{s}$ are *undefined* (Def. 3.6).

Intuitively, $S_{\mathbf{s}}\upharpoonright\mathbf{a}$ and $S_{\mathbf{a}}\upharpoonright\mathbf{s}$ 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 $\mathbf{err}$).

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

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

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

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

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

$$(\mathbf{q}\to\mathbf{r}:\{\mathtt{m}_i(S_i)\,.\,G_i\}_{i\in I})\upharpoonright\mathbf{p} = \begin{cases} \mathbf{r}\oplus_{i\in I}\mathtt{m}_i(S_i).(G_i\upharpoonright\mathbf{p}) & \text{if } \mathbf{p}=\mathbf{q} \\ \mathbf{q}\&_{i\in I}\mathtt{m}_i(S_i).(G_i\upharpoonright\mathbf{p}) & \text{if } \mathbf{p}=\mathbf{r} \\ \bigsqcap_{i\in I} G_i\upharpoonright\mathbf{p} & \text{if } \mathbf{q}\neq\mathbf{p}\neq\mathbf{r} \end{cases}$$

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

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*:

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

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

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

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

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

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* $\mathbf{p}$, written $S\upharpoonright\mathbf{p}$, is:

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

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

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

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

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

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

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, $\forall s, \mathbf{p}\neq\mathbf{q}, S, T$, $\{s[\mathbf{p}]:S, s[\mathbf{q}]:T\}\subseteq\Gamma$ implies $\overline{S\upharpoonright\mathbf{q}}\leqslant T\upharpoonright\mathbf{p}$.

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).

judgement $\Gamma \vdash P_\mathbf{s} \mid P_\mathbf{c} \mid P_\mathbf{a}$ holds. For this judgement, the subject reduction statement (4) is vacuously true (since Γ is not consistent): hence, we cannot prove that $P_\mathbf{s} \mid P_\mathbf{c} \mid P_\mathbf{a}$ "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):

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

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 **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 **a**, we must "skip" the first interaction between **s** and **c**, and *merge* the projections of G_1 and G_2 onto **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)

N/A

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

$$s[\mathbf{s}] : \mathbf{a}\&\texttt{query(Str)}.\mathbf{a}\oplus\texttt{price(Int)}.\mathbf{a}\&\left\{\texttt{buy}.\mathbf{end}\ ,\ \texttt{no}.\mathbf{end}\right\}$$

$$s[\mathbf{b}] : \mu\mathbf{t}.\mathbf{a}\&\left\{\texttt{split(Int)}.\mathbf{a}\oplus\left\{\texttt{yes}.\mathbf{end}\ ,\ \texttt{no}.\mathbf{t}\right\}\ ,\ \texttt{cancel}.\mathbf{end}\right\}$$

(2) Recursive two-buyers protocol. This is a mild variation of a typical example in MPST literature. Alice (**a**) queries the store (**s**) for an item, and the store replies with a `price`; then, she asks Bob (**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.

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

$$s[\mathbf{m}] : \mu\mathbf{t}.\mathbf{w}_1\oplus\texttt{datum(Int)}.\ldots.\mathbf{w}_n\oplus\texttt{datum(Int)}.\mathbf{r}\&\left\{\begin{array}{l}\texttt{continue(Int)}.\mathbf{t}\\ \texttt{stop}.\mathbf{w}_1\oplus\texttt{stop}.\ldots.\mathbf{w}_n\oplus\texttt{stop}\end{array}\right\}$$

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

$$s[\mathbf{r}] : \mu\mathbf{t}.\mathbf{w}_1\&\texttt{datum(Int)}.\ldots.\mathbf{w}_n\&\texttt{datum(Int)}.\mathbf{m}\oplus\left\{\begin{array}{l}\texttt{continue(Int)}.\mathbf{t}\\ \texttt{stop}.\mathbf{end}\end{array}\right\}$$

(3) Recursive map/reduce. The mapper (**m**) sends a `datum` to n workers ($\mathbf{w}_1, \ldots, \mathbf{w}_n$, for some given n), and each one sends a `result` to the reducer (**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.

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

$$s[\mathbf{s}] : \mathbf{wa}_1\oplus\texttt{datum(Int)}.\ldots.\mathbf{wa}_n\oplus\texttt{datum(Int)}.\mathbf{end}$$

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

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

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

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

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 $s[\mathbf{a}]$, when **a**lice interacts with **b**ob: **a**lice sends a message to the **s**tore in one of the branches under recursion $\mu\mathbf{t}.\ldots$ (where **b**ob answers `yes`), but not in the other branch (where **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 $s[\mathbf{a}]$ 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:

[S-⊕&] $\varphi\big(\Gamma,\ s[\mathbf{p}]{:}\mathbf{q}\oplus_{i\in I}\mathtt{m}_i(S_i).S_i',\ s[\mathbf{q}]{:}\mathbf{p}\&_{j\in J}\mathtt{m}_j(T_j).T_j'\big)$ implies $I\subseteq J$, and $\forall i\in I: S_i\leqslant T_i$;

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

[S-→] $\varphi(\Gamma)$ and $\Gamma\to\Gamma'$ implies $\varphi(\Gamma')$.

We say Γ *is safe*, written safe(Γ), 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-⊕&] 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-⊕&] to check unfolded types, where ⊕/& occur at the the top-level. By rule [S-→], 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 $\varphi=\{\Gamma' \mid \Gamma\to^*\Gamma'\}$, 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-⊕&] of Def. 4.1.

Def. 4.1 also has the properties in Lemma 4.3 below, useful for proving subject reduction: typing context splits preserve safety (item 1, which satisfies the remaining *desideratum* **(D2)** in §2.3); if Γ is safe, then supertyping/reductions commute (item 2); supertyping preserves safety (item 3).

Lemma 4.3. *For all typing contexts Γ and Γ':*

(1) if Γ,Γ' is safe, then Γ is safe;
(2) if Γ safe and $\Gamma\leqslant\Gamma'\to\Gamma''$ (for some Γ''), then there is Γ''' such that $\Gamma\to\Gamma'''\leqslant\Gamma''$;
(3) 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 4.4 (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:

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

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 $\varphi=$ safe (i.e., the largest safety property, cf. Def. 4.1).

Example 4.5. Take the (wrong) typing derivation (8) in §3.1, and replace the (wrong) application of rule [T-νClassic] with [TGen-ν] from Def. 4.4, instantiating φ with the safety property of Ex. 4.2 (that contains Γ). The resulting typing derivation is correct.

Ex. 4.5 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. 4.4) 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. 4.6, based on Lemmas 4.3 to 4.3); from this, we get that typed processes "never go wrong" (Cor. 4.7). And again, unlike classic MPST, these results are *not* limited by consistency.

Theorem 4.6 (Subject Reduction). *Assume* $\Theta \cdot \Gamma \vdash P$ *and* Γ *safe. Then,* $P \to P'$ *implies* $\exists \Gamma'$ *safe such that* $\Gamma \to^* \Gamma'$ *and* $\Theta \cdot \Gamma' \vdash P'$.

Corollary 4.7 (Type safety). *If* $\emptyset \cdot \emptyset \vdash P$ *and* $P \to^* P'$, *then* P' *has no error.*

Proof. We first prove a more general result. Assume $\Theta \cdot \Gamma \vdash P$ with Γ safe, and $P = P_1 \to \cdots \to P_n = P'$. By induction on n, using Thm. 4.6, we prove $\Theta \cdot \Gamma' \vdash P'$, for some safe Γ' such that $\Gamma \to^* \Gamma'$. 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. 4.7 as a special case of the result above, with $\Theta = \emptyset$ and $\Gamma = \Gamma' = \emptyset$ (that is vacuously safe). □

Example 4.8. Take our opening example in §1, and its typed process from Ex. 2.7 and 4.5. Using our new Thm. 4.6 instead of the classic MPST subject reduction (4) in §1, we infer that all process reductions are well-typed. And by Cor. 4.7, we are guaranteed that they do not contain errors.

Finally, note that type checking is decidable, whenever Def. 4.4 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. 4.6 and Cor. 4.7 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. 4.4: subject reduction and type safety hold for any safety property φ.

Theorem 4.9. *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. 4.4), 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. 4.6, Cor. 4.7), 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. 4.4 to enforce them, without consistency limitations.

[2] In §5.4, we show that all typing derivations of classic MPST are valid under Def. 4.4: 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 $Q = c[\mathbf{q}] \oplus \mathtt{m}\langle s'[\mathbf{r}]\rangle . Q'$ (for some $\mathtt{m}, s', \mathbf{r}, Q'$), then $\exists \mathbb{C}' \colon P' \to^* \mathbb{C}'[Q']$; and
(2) if $Q = c[\mathbf{q}] \sum_{i \in I} \mathtt{m}_i(x_i) . Q'_i$ (for some $\mathtt{m}_i, x_i, Q'_i$), then $\exists \mathbb{C}', k \in I, s', \mathbf{r} \colon P' \to^* \mathbb{C}'\left[Q'_k\{{}^{s'[\mathbf{r}]}/_{x_k}\}\right]$.

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 $P'_j \to \mathbb{C}''\left[Q'_k\{{}^{s'[\mathbf{r}]}/_{x_k}\}\right]$ (for some $\mathbb{C}'', k \in I, s', \mathbf{r}$).

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:

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

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

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

i.e., $\mathbf{r}$ sends $\mathtt{m}_1$ to $\mathbf{q}$ 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 `response` from $\mathbf{q}$: hence, P is *not* live. To address this, we can instead define Q above as:

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

i.e., $\mathbf{r}$ sends $\mathtt{m}_1$ and then $\mathtt{m}_2$ to $\mathbf{q}$, and this causes $\mathbf{q}$ to send `resp` to $\mathbf{p}$ (cf. P_2 above); meanwhile, the sub-process Q' loops, with $\mathbf{r'}$ and $\mathbf{r''}$ 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 $\mathbf{def}\ X(x_1{:}S_1, \ldots, x_n{:}S_n) = Q\ \mathbf{in}\ P'$, for all $i \in 1..n$, $S_i \not\leqslant \mathbf{end}$ implies that a call $Y\langle \ldots, x_i, \ldots\rangle$ 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 $\mathrm{end}(\Gamma')$.

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, $P \equiv \left|_{\mathbf{p}\in I} P_{\mathbf{p}}\right.$, and each $P_{\mathbf{p}}$ either is $\mathbf{0}$ (up-to $\equiv$), or only plays $\mathbf{p}$ in s. Then, $\Gamma \not\to$ implies $\exists \Gamma', P'$ such that $\Gamma \to \Gamma'$, $P \to^* P'$ and $\emptyset \cdot \Gamma' \vdash P'$, where $P' \equiv \left|_{\mathbf{p}\in I} P'_{\mathbf{p}}\right.$ and each $P'_{\mathbf{p}}$ either is $\mathbf{0}$ (up-to $\equiv$), or only plays $\mathbf{p}$ in s.*

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 $\mathtt{m}_1, \ldots, \mathtt{m}_n$ (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.

- 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.

[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)

$$\Gamma \models \nu Z.\left(\begin{array}{l}\forall s, \mathbf{p}, \mathbf{q}, \mathtt{m}, \mathtt{m}', S, S'.\\ \quad \left(\langle s{:}\mathbf{p}{\oplus}\mathbf{q}{:}\mathtt{m}(S)\rangle\top \wedge \langle s{:}\mathbf{p}\&\mathbf{q}{:}\mathtt{m}'(S')\rangle\top \Rightarrow \langle s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}\rangle\top\right)\\ \quad \wedge\ [s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}]Z\end{array}\right)$$

(2) Γ is **deadlock-free**, written df(Γ), iff:

$\Gamma \to^* \Gamma' \not\to$ implies end(Γ')

$$\Gamma \models \nu Z.\left(\begin{array}{l}\left(\begin{array}{l}(\forall s, \mathbf{p}, \mathbf{q}, \mathtt{m}.[s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}]\bot) \Rightarrow\\ \quad \forall \mathbf{p}, \mathbf{q}, \mathtt{m}, S.[s{:}\mathbf{p}\&\mathbf{q}{:}\mathtt{m}(S)]\bot \wedge [s{:}\mathbf{p}{\oplus}\mathbf{q}{:}\mathtt{m}(S)]\bot\end{array}\right)\\ \wedge\ \forall \mathbf{p}, \mathbf{q}, \mathtt{m}.[s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}]Z\end{array}\right)$$

(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)

$$\Gamma \models \mu Z.\left(\begin{array}{l}\left(\begin{array}{l}(\forall s, \mathbf{p}, \mathbf{q}, \mathtt{m}.[s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}]\bot) \Rightarrow\\ \quad \forall s, \mathbf{p}, \mathbf{q}, \mathtt{m}, S.[s{:}\mathbf{p}\&\mathbf{q}{:}\mathtt{m}(S)]\bot \wedge [s{:}\mathbf{p}{\oplus}\mathbf{q}{:}\mathtt{m}(S)]\bot\end{array}\right)\\ \wedge\ \forall s, \mathbf{p}, \mathbf{q}, \mathtt{m}.[s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}]Z\end{array}\right)$$

(4) Γ is **never-terminating**, written nterm(Γ), iff:

$\Gamma \to^* \Gamma'$ implies $\Gamma' \to$

$$\Gamma \models \nu Z.\left(\exists s, \mathbf{p}, \mathbf{q}, \mathtt{m}.\langle s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}\rangle\top \wedge \forall s, \mathbf{p}, \mathbf{q}, \mathtt{m}.[s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}]Z\right)$$

(5) Γ is **live**, written live(Γ), iff:

$\varphi(\Gamma)$, for some φ such that

[L-&] whenever $\varphi(\Gamma', s[\mathbf{p}]{:}S)$ with $S = \mathbf{q}\&_{i\in I}\mathtt{m}_i(S_i).S_i'$, $\exists i \in I$: $\exists \Gamma''$: $\Gamma', s[\mathbf{p}]{:}S \to^* \Gamma'', s[\mathbf{p}]{:}S_i'$

[L-⊕] whenever $\varphi(\Gamma', s[\mathbf{p}]{:}S)$ with $S = \mathbf{q}\oplus_{i\in I}\mathtt{m}_i(S_i).S_i'$, $\forall i \in I$: $\exists \Gamma''$: $\Gamma', s[\mathbf{p}]{:}S \to^* \Gamma'', s[\mathbf{p}]{:}S_i'$

plus clauses [S-μ], [S-$\to$] (Def. 4.1).

$$\Gamma \models \nu Z.\left(\begin{array}{l}\forall s, \mathbf{p}, \mathbf{q}.\\ \quad \left(\begin{array}{l}(\exists \mathtt{m}, S.\langle s{:}\mathbf{p}\&\mathbf{q}{:}\mathtt{m}(S)\rangle\top) \Rightarrow\\ \quad \mu Z'.\exists \mathtt{m}.\langle s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}\rangle\top \vee \exists \mathbf{p}', \mathbf{q}', \mathtt{m}'.\langle s{:}\mathbf{p}',\mathbf{q}'{:}\mathtt{m}'\rangle Z'\end{array}\right)\\ \quad \wedge\\ \quad \forall \mathtt{m}.\left(\begin{array}{l}(\exists S.\langle s{:}\mathbf{p}{\oplus}\mathbf{q}{:}\mathtt{m}(S)\rangle\top) \Rightarrow\\ \quad \mu Z'.\langle s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}\rangle\top \vee \exists \mathbf{p}', \mathbf{q}', \mathtt{m}'.\langle s{:}\mathbf{p}',\mathbf{q}'{:}\mathtt{m}'\rangle Z'\end{array}\right)\\ \quad \wedge\\ \quad \forall \mathtt{m}.[s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}]Z\end{array}\right)$$

(6) Γ is **live**$^+$, written live$^+$(Γ), iff:

$\varphi(\Gamma)$, for φ such that

[L-&$^+$] clause [L-&] above; *moreover*, $\Gamma', s[\mathbf{p}]{:}S$ 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 $\Gamma_t = \Gamma'', s[\mathbf{p}]{:}S_i'$ (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).

$$\Gamma \models \nu Z.\left(\begin{array}{l}\forall s, \mathbf{p}, \mathbf{q}.\\ \quad \left(\begin{array}{l}(\exists \mathtt{m}, S.\langle s{:}\mathbf{p}\&\mathbf{q}{:}\mathtt{m}(S)\rangle\top) \Rightarrow\\ \quad \mu Z'.\exists \mathtt{m}.\langle s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}\rangle\top \vee \exists \mathbf{p}', \mathbf{q}'.\left(\begin{array}{l}\exists \mathtt{m}'.\langle s{:}\mathbf{p}',\mathbf{q}'{:}\mathtt{m}'\rangle\top\\ \wedge\ \forall \mathtt{m}'.[s{:}\mathbf{p}',\mathbf{q}'{:}\mathtt{m}']Z'\end{array}\right)\end{array}\right)\\ \quad \wedge\\ \quad \forall \mathtt{m}.\left(\begin{array}{l}(\exists S.\langle s{:}\mathbf{p}{\oplus}\mathbf{q}{:}\mathtt{m}(S)\rangle\top) \Rightarrow\\ \quad \mu Z'.\langle s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}\rangle\top \vee \exists \mathbf{p}', \mathbf{q}'.\left(\begin{array}{l}\exists \mathtt{m}'.\langle s{:}\mathbf{p}',\mathbf{q}'{:}\mathtt{m}'\rangle\top\\ \wedge\ \forall \mathtt{m}'.[s{:}\mathbf{p}',\mathbf{q}'{:}\mathtt{m}']Z'\end{array}\right)\end{array}\right)\\ \quad \wedge\\ \quad \forall \mathtt{m}.[s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}]Z\end{array}\right)$$

(7) Γ is **live**$^{++}$, written live^{++}(Γ), iff:

$\varphi(\Gamma)$, for φ such that

[L-&$^{++}$] clause [L-&] above; *moreover*, $\exists n \in \mathbb{N}^0$ such that, whenever $\Gamma', s[\mathbf{p}]{:}S = \Gamma_0 \to \Gamma_1 \to \cdots \to \Gamma_n$, then $\exists j \leq n, \Gamma''$ such that $\Gamma_j \to \Gamma'', s[\mathbf{p}]{:}S_i'$ (for some $i \in I$)

[L-⊕$^{++}$] clause [L-⊕] above, *plus* the *"moreover…"* part of [L-&$^{++}$]

plus clauses [S-μ], [S-$\to$] (Def. 4.1).

$$\Gamma \models \nu Z.\left(\begin{array}{l}\forall s, \mathbf{p}, \mathbf{q}.\\ \quad \left(\begin{array}{l}(\exists \mathtt{m}, S.\langle s{:}\mathbf{p}\&\mathbf{q}{:}\mathtt{m}(S)\rangle\top) \Rightarrow\\ \quad \mu Z'.\exists \mathtt{m}.\langle s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}\rangle\top \vee \left(\begin{array}{l}\exists s', \mathbf{p}', \mathbf{q}', \mathtt{m}'.\langle s'{:}\mathbf{p}',\mathbf{q}'{:}\mathtt{m}'\rangle\top\\ \wedge\ \forall s', \mathbf{p}', \mathbf{q}', \mathtt{m}'.[s'{:}\mathbf{p}',\mathbf{q}'{:}\mathtt{m}']Z'\end{array}\right)\end{array}\right)\\ \quad \wedge\\ \quad \forall \mathtt{m}.\left(\begin{array}{l}(\exists S.\langle s{:}\mathbf{p}{\oplus}\mathbf{q}{:}\mathtt{m}(S)\rangle\top) \Rightarrow\\ \quad \mu Z'.\langle s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}\rangle\top \vee \left(\begin{array}{l}\exists s', \mathbf{p}', \mathbf{q}', \mathtt{m}'.\langle s'{:}\mathbf{p}',\mathbf{q}'{:}\mathtt{m}'\rangle\top\\ \wedge\ \forall s', \mathbf{p}', \mathbf{q}', \mathtt{m}'.[s'{:}\mathbf{p}',\mathbf{q}'{:}\mathtt{m}']Z'\end{array}\right)\end{array}\right)\\ \quad \wedge\\ \quad \forall \mathtt{m}.[s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}]Z\end{array}\right)$$

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]).

The intuition behind live/live$^+$/live^{++} is the following. Take a typing context $\Gamma, s[\mathbf{p}]{:}S$. 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:

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

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 $\mathbf{p}, \mathbf{q}$ who can interact (clause "$\exists \mathtt{m} : \Gamma \xrightarrow{s:\mathbf{p},\mathbf{q}:\mathtt{m}}$"), 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:

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

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 $\mathbf{p}, \mathbf{q}$ can interact infinitely by exchanging $\mathtt{m}_1$, yielding the infinite run $\Gamma \xrightarrow{s:\mathbf{p},\mathbf{q}:\mathtt{m}_1} \Gamma \xrightarrow{s:\mathbf{p},\mathbf{q}:\mathtt{m}_1} \cdots$; 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:

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

Note that Γ is live, and Γ' is reachable from Γ, via the reductions $\Gamma \xrightarrow{s:\mathbf{p},\mathbf{q}:\mathtt{m}_1} \xrightarrow{s:\mathbf{p},\mathbf{q}':\mathtt{m}_2} \Gamma'$; however, Γ' can be infinitely delayed in the unfair run $\Gamma \xrightarrow{s:\mathbf{r},\mathbf{r}':\mathtt{m}_2} \Gamma \xrightarrow{s:\mathbf{r},\mathbf{r}':\mathtt{m}_2} \cdots$ that never fires the communication between $\mathbf{p}$ and $\mathbf{q}$, and thus, never enables the interaction between $\mathbf{p}$ and $\mathbf{q}'$. 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 build a finite derivation that supports $\Gamma \in \mathbb{X}$ by instantiating rule [TS-Comm] twice — choosing $\mathbf{p}, \mathbf{q}$ for the fist reduction, and then $\mathbf{p}, \mathbf{q}'$ to reach the axiom [TS-Target], ignoring the interactions between $\mathbf{r}, \mathbf{r}'$.

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.

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

5.4 Relationships Between Typing Context Properties

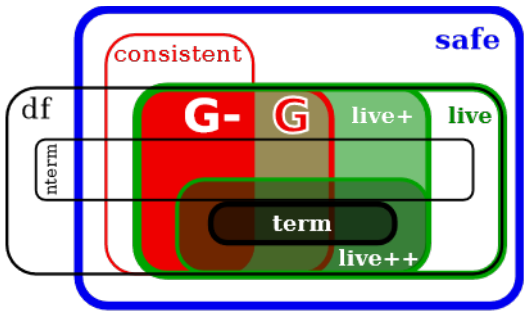
We now study how typing context properties are related: this is formalised in Lemma 5.9 below, that also conveys the expressiveness of our new type system (Remark 5.12).

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 $\Gamma = \{s[\mathbf{p}] : G\upharpoonright\mathbf{p}\}_{\mathbf{p}\in\mathrm{roles}(G)}$, where $G\upharpoonright\mathbf{p}$ 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) $\mathrm{consistent}(\Gamma) \not\Longleftrightarrow \mathrm{safe}(\Gamma)$;
(2) $\mathrm{live}(\Gamma) \not\Longleftrightarrow \mathrm{safe}(\Gamma)$;
(3) $\mathrm{live}(\Gamma) \not\Longleftrightarrow \mathrm{df}(\Gamma)$;
(4) $\mathrm{nterm}(\Gamma) \not\Longleftrightarrow \mathrm{df}(\Gamma)$;
(5) $\mathrm{consistent}(\Gamma) \not\Longleftrightarrow \mathrm{df}(\Gamma)$;
(6) $\mathrm{consistent}(\Gamma) \wedge \mathrm{df}(\Gamma) \not\Longleftrightarrow \mathrm{live}(\Gamma)$;
(7) $\mathrm{live}^{++}(\Gamma) \not\Longleftrightarrow \mathrm{live}^{+}(\Gamma) \not\Longleftrightarrow \mathrm{live}(\Gamma)$;
(8) $\mathrm{term}(\Gamma) \not\Longleftrightarrow \mathrm{live}^{++}(\Gamma)$;
(9) assume $\mathrm{dom}(\Gamma) = \{s\}$ *(Def. 2.6). Then:* $\exists G : \mathrm{fproj}_{G,s}(\Gamma) \not\Longleftrightarrow \mathrm{live}^{+}(\Gamma)$.



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)).

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

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

Remark 5.12. *By Lemma 5.9(1,9), our general session type system instantiated with $\varphi = \mathrm{fproj}_{G,\mathtt{s}}$ 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. 4.5), 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 $(vs)\,P$ has a desired property from Def. 5.1, we can correspondingly instantiate φ in Def. 4.4, and check if the judgement "$\emptyset \vdash (vs{:}\Gamma)\,P$ 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. 4.9, 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*

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

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:

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

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 4.3), 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,* $P \equiv \left|_{\mathbf{p}\in I} P_{\mathbf{p}}\right.$, *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)* $\mathrm{df}(\Gamma)$ *implies that* P *is deadlock-free; (2)* $\mathrm{term}(\Gamma)$ *implies that* P *is terminating; (3)* $\mathrm{nterm}(\Gamma)$ *implies that* P *is never-terminating; (4)* $\mathrm{live}^+(\Gamma)$ *implies that* P *is live; and (5)* $\mathrm{live}^{++}(\Gamma)$ *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 $\mathrm{live}^+(\Gamma)$ and $\Gamma \leqslant \Gamma'$, then $\mathrm{live}^+(\Gamma')$. □

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. 4.9).*

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

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

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 $\mu\mathtt{Z}.\phi/\nu\mathtt{Z}.\phi$, allowing to iterate ϕ for a finite/infinite number of times; a variable $\mathtt{Z}$, for iteration; and universal/existential quantification $\forall\mathtt{d}.\phi/\exists\mathtt{d}.\phi$. When a typing context Γ satisfies a formula ϕ, we write $\Gamma \models \phi$.

Example 6.1. The μ-calculus formula $\phi = \exists s.\exists \mathbf{p}.\exists \mathbf{q}.\exists \mathtt{m}.\exists S.\langle s{:}\mathbf{p}\oplus\mathbf{q}{:}\mathtt{m}(S)\rangle\top$ says: "accept a typing context if, for some session s, roles $\mathbf{p}$ and $\mathbf{q}$, message label $\mathtt{m}$, and type S, it can perform an output action $s{:}\mathbf{p}\oplus\mathbf{q}{:}\mathtt{m}(S)$" — and after such a transition, the reached typing context is always accepted, by $\top$. Therefore, if we take the typing context $\Gamma = s[\mathbf{r}]{:}\mathbf{r}'\oplus\mathtt{msg}(\mathtt{Str}).\mathbf{end}$, then we have $\Gamma \xrightarrow{s:\mathbf{r}\oplus\mathbf{r}':\mathtt{msg}(\mathtt{Str})}$ (by Def. 2.8), which means that Γ satisfies ϕ — in symbols, $\Gamma \models \phi$. Moreover, Γ satisfies the formula $\forall s.\forall \mathbf{p}.\forall \mathbf{q}.\forall \mathtt{m}.[s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}]\bot$, 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 $\phi' = \exists s.\exists \mathbf{p}.\exists \mathbf{q}.\exists \mathtt{m}.\langle s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}\rangle\top$, then Γ above does *not* satisfy ϕ', because it requires a communication transition to be enabled. However, if we extend Γ as $\Gamma' = \Gamma, s[\mathbf{r}']{:}\mathbf{r}\,\&\,\mathtt{msg}(\mathtt{Str}).\mathbf{end}$, 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 $\mathbf{p}$ and $\mathbf{q}$, 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 $\nu\mathtt{Z}\ldots$ and the clause $\ldots \wedge [s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}]\mathtt{Z}$;
- **deadlock-freedom (2)** checks whether communication is possible; if not ("$\forall\ldots[s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}]\bot$", 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 **end**. This must hold for any context reachable via communications: it is enforced by $\nu\mathtt{Z}\ldots$ and $\ldots \wedge \forall\ldots[s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}]\mathtt{Z}$;
- **termination (3)** is similar to deadlock-freedom, but uses a *least* fixed point $\mu\mathtt{Z}\ldots$: hence, the clause $\ldots \wedge \forall\ldots[s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}]\mathtt{Z}$ 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 **end**;
- **never-termination (4)** checks that in any context reachable via communication transitions ($\nu\mathtt{Z}\ldots$ and $\ldots \wedge \forall\ldots[s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}]\mathtt{Z}$), some further communication is possible ($\exists\ldots\langle s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}\rangle\top$);
- **liveness (5)** checks that, if an input or output between two roles $\mathbf{p}$ and $\mathbf{q}$ 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 $\mu\mathtt{Z}'\ldots$, that can iterate on the clause $\ldots \vee \exists\ldots\langle s{:}\mathbf{p}',\mathbf{q}'{:}\mathtt{m}'\rangle\mathtt{Z}'$ for a finite number of times. The top-level greatest fixed point $\nu\mathtt{Z}\ldots$ repeats the check for all contexts reachable via communication (clause $\ldots \wedge \forall\ldots[s{:}\mathbf{p},\mathbf{q}{:}\mathtt{m}]\mathtt{Z}$);
- **liveness$^+$ (6)** is similar to liveness, but the nested fixed points $\mu\mathtt{Z}'\ldots$ build finite sequences of communications by picking a pair of roles $\mathbf{p}', \mathbf{q}'$ at each step, and following *all* their interactions, until a communication between $\mathbf{p}, \mathbf{q}$ 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 $\mu\mathtt{Z}'\ldots$ build finite sequences by following *any* communication between *any* pair of roles, until a communication 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

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%

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. Due to space limits, we summarise the main results from [Scalas and Yoshida 2018a].

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

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

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 $s[\mathbf{p}]$ is a pair $(M;S)$, where $M = \mathbf{q}!\mathtt{m}(S')\cdot\epsilon$ is a *message queue type* (abstracting the pending messages sent through $s[\mathbf{p}]$), followed by the continuation type S. In Γ', the topmost queued message type matches the branching type of $s[\mathbf{q}]$: 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]:

$$\begin{array}{l} \text{If } \Theta\cdot\Gamma \vdash_{\mathcal{S}} P \text{ and } \exists\Gamma_0 \text{ such that } \Gamma,\Gamma_0 \text{ consistent} \text{ and } P \to P', \\ \text{then } \exists\Gamma',\Gamma_0' \text{ consistent: } \Gamma,\Gamma_0 \to^* \Gamma',\Gamma_0' \text{ and } \Theta\cdot\Gamma' \vdash_{\mathcal{S}} P' \end{array}$$

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* $\rightarrow_{\mathcal{S}}$, where $\mathcal{S}$ is a set of sessions, as in (11); e.g., in (12) we have $\Gamma \rightarrow_{\{s\}} \Gamma' \rightarrow_{\{s\}} \Gamma''$.

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

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. 4.9 holds for async MPST.

Theorem 7.2. *If* φ *is decidable, then "*$\Theta \cdot \Gamma \vdash_{\mathcal{S}} P$ *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 $\Gamma = s[\mathbf{p}]{:}\mathbf{q}\oplus\mathtt{m}_1.\mathbf{q}\&\mathtt{m}_2,\ s[\mathbf{q}]{:}\mathbf{p}\oplus\mathtt{m}_2.\mathbf{p}\&\mathtt{m}_1$: it is deadlocked under synchronous semantics, and not projectable from any global type — but under asynchrony, the top-level outputs of $\mathbf{p}$ and $\mathbf{q}$ 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. 4.6, Cor. 4.7) 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. 4.9 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]	≤ plain	yes	correct	**(C1)**
(b)	Caires and Pérez [2016]; Chen [2015]; Deniélou et al. [2012]; Deniélou and Yoshida [2012]; Toninho and Yoshida [2016]	≥ 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 two 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]: 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. Such a type system is subsumed by letting $\varphi = \text{fproj}_{G,s}$ in our Def. 4.4; 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 [2018b]: 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 [2018b] *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 [2018b], 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 [2018b];

(3) the branching/selection typing rules of Scalas and Yoshida [2018b] (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 2018b, 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 [2018b] 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 $\varphi = \text{live}$ in Def. 4.4, this paper largely subsumes Scalas and Yoshida [2018b]'s work — minus some corner cases based on the inspection of typing context reductions (cf. item *(3)* above).

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:

$$\exists G : \text{fproj}_{G,s}(\Gamma) \iff\!\!\!\!\!\!\Longrightarrow \text{consistent}(\Gamma) \iff\!\!\!\!\!\!\Longrightarrow \text{live}^{++}(\Gamma) \iff \Big(\text{safe}(\Gamma) \text{ and } \text{df}(\Gamma)\Big) \tag{13}$$

Here, the leftmost "$\iff\!\!\!\!\!\!\Longrightarrow$" 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 "$\iff\!\!\!\!\!\!\Longrightarrow$" 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. 4.6/Cor. 4.7 holds. This corresponds to the following conjecture:

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

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-→] 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, Sung-Shik Jongmans, and Julien Lange for their comments, and to Simon Castellan for testing the companion artifact. 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").

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, 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. 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

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

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. 2018a. *Less is More: Multiparty Session Types Revisited*. Technical Report 6. Imperial College London. https://www.doc.ic.ac.uk/research/technicalreports/2018/6

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

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. 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