Journal of Logical and Algebraic Methods in Programming 104 (2019) 127–173

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

Precise subtyping for synchronous multiparty sessions

Silvia Ghilezan [a], Svetlana Jakšić [a,c], Jovanka Pantović [a], Alceste Scalas [b], Nobuko Yoshida [b]

[a] Univerzitet u Novom Sadu, Serbia
[b] Imperial College London, UK
[c] Høgskulen på Vestlandet, Norway

ARTICLE INFO

Article history:
Received 12 December 2017
Received in revised form 17 December 2018
Accepted 18 December 2018
Available online 29 December 2018

Keywords:
Concurrency
Process calculi
Multiparty session types
Subtyping

ABSTRACT

This paper proves the soundness and completeness, together referred to as *preciseness*, of the subtyping relation for a synchronous multiparty session calculus.
We address preciseness from *operational* and *denotational* viewpoints. The operational preciseness has been recently developed with respect to *type safety*, i.e., the safe replacement of a process of a smaller type in a context where a process of a bigger type is expected. The denotational preciseness is based on the denotation of a type: a mathematical object describing the meaning of the type, in accordance with the denotations of other expressions from the language.
The main technical contribution of this paper is a novel proof strategy for the operational completeness of subtyping. We develop the notion of *characteristic global type* of a session type T, which describes a deadlock-free circular communication protocol involving all participants appearing in T. We prove operational completeness by showing that, if we place a process not conforming to a subtype of T in a context that matches the characteristic global type of T, then we obtain a *deadlock*. The denotational preciseness is proved as a corollary of the operational preciseness.

1. Introduction

Concurrency and The Quest for Canonicity. In concurrency theory, it is often the case that open problems are solved not just by formulating a suitable solution, but also by providing *general methods* to *justify* whether the given solution is *canonical* or not. With this approach, justification methods often lead to general and reusable formal reasoning and proof techniques, adaptable to different concurrent calculi.

The most notable instance of this approach can be found in the study of *bisimulations* in the π-calculus [31,47,55]. Variations of the π-calculus can be formalised with various labelled transition systems (LTSs), that in turn, can be used to define various notions of bisimulation. This leads to the question: given a calculus equipped with an LTS, and a notion of bisimulation, how can we ascertain whether the latter is, in some sense, canonical for the former? To answer this question, is now common to adopt a *general justification method* which consists in proving that, in the given calculus and LTS, the given bisimulation coincides with contextual congruence [29,33,48].

In this work, our aim is to extend this approach to *subtyping relations* for multiparty session calculi:

E-mail address: n.yoshida@imperial.ac.uk (N. Yoshida).

https://doi.org/10.1016/j.jlamp.2018.12.002
2352-2208/

(1) we adopt a general justification method for determining whether a given subtyping relation is canonical in a process calculus with *binary* sessions [14];
(2) from this basis, we develop a new justification method for *multiparty* sessions, and
(3) we show that the *subtyping relation for synchronous multiparty session types* [16,34,35,39,40] is canonical, as it satisfies such a method.

Denotational Preciseness of Subtyping in Functional Calculi. To introduce our approach, we first summarise a *denotational* notion of canonicity for subtyping relations in functional calculi.

A subtyping relation is a pre-order (reflexive and transitive relation) on types that conforms to the well-known Liskov's substitution principle [44,45]: if σ is a subtype of τ (notation $\sigma \leq \tau$), then a term of type σ may be provided whenever a term of type τ is needed: see also [53] (Chapter 15) and [28] (Chapter 23).

A subtyping relation can be deemed canonical when it is both *sound* and *complete*. Given a functional calculus, the usual approach to assess soundness and completeness of subtyping is to consider the *interpretation* of a type σ (notation $[\![\sigma]\!]$) as a set that describes the meaning of the type in accordance with the denotations of the terms of the calculus; in general, $[\![\sigma]\!]$ yields a subset of the domain of a model of the calculus. Then, we say a subtyping relation is *denotationally sound* when $\sigma \leq \tau$ implies $[\![\sigma]\!] \subseteq [\![\tau]\!]$, and *denotationally complete* when $[\![\sigma]\!] \subseteq [\![\tau]\!]$ implies $\sigma \leq \tau$. A subtyping relation is *denotationally precise* if it is both denotationally sound and denotationally complete. This well-established definition has been studied in the context of λ-calculi: e.g., in the pure λ-calculus with arrow and intersection types by Barendregt et al. [4], in a call-by-value λ-calculus with arrow, intersection and union types by Bakel et al. [1] and by Ishihara and Kurata [38], in a wide class of calculi with arrow, union and pair types by Vouillon [59], and in a concurrent λ-calculus by Dezani-Ciancaglini and Ghilezan [21]. When introducing a new subtyping relation for a functional calculus, it is therefore natural to adopt denotational preciseness as a notion of canonicity, and thus investigate whether the proposed relation satisfies the requirement.

Operational Preciseness. In principle, one would want to adopt such a rigorous notion of subtyping canonicity to concurrency. This, however, requires to adapt the notions of soundness and completeness to the *operational* (rather than denotational) semantics commonly found in process calculi; and this, in turn, leads to linking the canonicity of subtyping to *type safety*.

In this work, we adopt the following intuitions:

- *Operational soundness* of subtyping means that, if we have a context expecting a term of type τ, then $\sigma \leq \tau$ implies that a term of type σ can be placed into that context without causing errors; e.g., if the context requires a real number, it is always correct to fill it with a natural number.
- *Operational completeness* means that, if $\sigma \not\leq \tau$, then we can find a context expecting a term of type τ, such that if we fill it with a certain σ-typed term, it will behave "badly". For example, $\mathsf{nat} \not\leq \mathsf{bool}$, and by taking the negation $\neg(\cdot)$ (that requires a boolean value) we can show that if we fill it with a natural number, then we obtain the erroneous term $\neg(5)$.

Operational completeness is the opposite direction of operational soundness, which is formalised above in a contrapositive form; henceforth we adopt classical reasoning.

To formally define operational soundness and completeness, we need a boolean predicate bad on terms, standard typing judgements $\Gamma \vdash M : \sigma$ (where Γ is a mapping from variables to types and M is a term) and evaluation contexts C. Then, we say that:

- A subtyping relation is *operationally sound* when $\sigma \leq \tau$ implies that if (for some ρ) $x : \tau \vdash C[x] : \rho$ and $\vdash M : \sigma$, then $\mathsf{bad}(C[M])$ is false, for all C and M.
- A subtyping relation is *operationally complete* when $\sigma \not\leq \tau$ implies that, for some ρ, C and M, we have $x : \tau \vdash C[x] : \rho$ and $\vdash M : \sigma$ and $\mathsf{bad}(C[M])$.

and we say a subtyping relation is *operationally precise* if it is both operationally sound and operationally complete.

When the typing judgement $\Gamma \vdash M : \sigma$ is induced by a set of rules that include subsumption, then operational soundness immediately follows after we prove type safety (through a subject reduction theorem). To prove operational completeness, more sophistication is needed. We adopt the following general methodology, originally proposed for *binary* sessions by Chen et al. [15], Chen et al. [14]:

- **[Step 1]** characterise the negation of the subtyping relation;
- **[Step 2]** for each type σ, define a *characteristic term* M_σ;
- **[Step 3]** for each type σ, define a *characteristic context* C_σ, behaving well if filled with σ-typed terms;
- **[Step 4]** show that if $\sigma \not\leq \tau$, then $\mathsf{bad}(C_\tau[M_\sigma])$.

In this work, we adapt this proof method to *multiparty session types*: the adaptation requires a novel notion of *characteristic global types*, as explained below.

Multiparty Session Types. The Multiparty Session Type discipline is a framework to guarantee safety properties for the interactions among multiple concurrent peers. The structures of interactions are naturally distilled as *protocols*. Each protocol describes a bare skeleton of how interactions should proceed, by combining and sequencing inputs/outputs, choices, and recursions. In the theory of multiparty session types [16,34,35], protocols are captured as types for interaction; type checking statically ensures that all participants engaged in an instance of a protocol, or a *session*, follow the specified protocol structure: this guarantees type safety, and fidelity of programs to a stipulated protocol. This implies that

($\star$) *a group of processes typed within a single multiparty session is deadlock-free.*

We illustrate the key ideas of our formulation and proofs for preciseness, through an example. Let us consider a simple *ring protocol* where participant `Alice` sends a message with label ℓ_1, carrying a natural value, to participant `Bob`, who forwards the value to `Carol` (in a message with label ℓ_2), who finally forwards it to `Alice` (in a message with label ℓ_3). To develop the code for this protocol, we start by specifying the global type, which can concisely and clearly describe the high-level interactions of multiple participants [16,17,34,35], as follows (`end` denotes protocol termination):

$$\mathsf{G} = \texttt{Alice} \to \texttt{Bob}\colon \ell_1(\mathsf{nat})\,.\,\texttt{Bob} \to \texttt{Carol}\colon \ell_2(\mathsf{nat})\,.\,\texttt{Carol} \to \texttt{Alice}\colon \ell_3(\mathsf{nat})\,.\,\texttt{end}$$

The flow of communication is indicated with the symbol $\to$; upon agreement on G as a specification for `Alice`, `Bob` and `Carol`, each program can be implemented separately: e.g., a possible implementation for `Alice` is $\texttt{Bob}!\ell_1(50).\texttt{Carol}?\ell_3(z).\mathbf{0}$ (output 50 to `Bob` and input z from `Carol`). For type-checking, G is *projected* into end-point session types: e.g., a type for `Alice`'s point of view is $\texttt{Bob}!\ell_1(\mathsf{nat}).\texttt{Carol}?\ell_3(\mathsf{nat})$ (send a natural value to `Bob`, then input a natural from `Carol`). Importantly, type checking ensures that `Alice`'s process matches her session type; similarly, we can define the implementation of `Bob` and `Carol`, and type-check them against their projected end-point types.

Operational Preciseness of Multiparty Session Types. As explained above, the operational soundness follows straightforwardly after proving type safety. For completeness, we can expect the negation of subtyping as well as characteristic processes to be defined as the one by Chen et al. [14] (in the synchronous case), since the endpoint types are similar to binary session types, and the synchronous subtyping is similar to the one in the literature [18,25]. The remaining task is to find out the appropriate "bad" behaviour and characteristic contexts. Observing ($\star$) above, the "bad" behaviour should be defined as a set of *deadlocked* processes engaged in a single multiparty session. However, to capture the role of subtyping, we need some sophistication: we must ensure that the characteristic context behaves well (i.e., does *not* deadlock) when it includes a process of type τ (say, for participant p), but behaves badly (i.e., deadlocks) when such a process is replaced with another process of type σ such that $\sigma \not\leqslant \tau$. This requires to develop a notion of *characteristic global types*, in addition to characteristic processes and contexts. The participants that follow the characteristic global type will interact smoothly with p when its implementation P_τ follows the corresponding projected endpoint type τ — i.e., we have $\neg\mathsf{bad}(C_\tau[P_\tau])$; however, following **[Step 4]**, we show that if a process P_σ follows a type $\sigma \not\leqslant \tau$, then we have $\mathsf{bad}(C_\tau[P_\sigma])$. The characteristic global type of p is given as a collection of cyclic communications so that incorrect behaviours can be caught out as bad behaviour (i.e., causing a deadlock). Summing up, we prove the operational completeness by the following novel methodology:

- **[Step 1]** We characterise the negation of the subtyping relation (notation $\not\leqslant$).
- **[Step 2]** For each type $\mathbb{T}$, we define a *characteristic process* $\mathcal{P}(\mathbb{T})$, which offers the series of interactions described by $\mathbb{T}$.
- **[Step 3]** For each type $\mathbb{T}'$ and participant r not appearing in $\mathbb{T}'$, we define a *characteristic global type* $\mathcal{G}(\mathbb{T}', \mathsf{r})$ and a *characteristic context* $C_{\mathsf{r},\mathbb{T}'}$. The characteristic context is a parallel composition of processes following the behaviour of participants of $\mathcal{G}(\mathsf{r}, \mathbb{T}')$ — except for the participant r, whose behaviour is left unspecified.
- **[Step 4]** We show that if $\mathbb{T} \not\leqslant \mathbb{T}'$ then $\texttt{stuck}\big(C_{\mathsf{r},\mathbb{T}'}[\mathsf{r} \triangleleft \mathcal{P}(\mathbb{T})]\big)$ — where $C_{\mathsf{r},\mathbb{T}'}[\mathsf{r} \triangleleft \mathcal{P}(\mathbb{T})]$ is the characteristic context $C_{\mathsf{r},\mathbb{T}'}$ with participant r performing actions $\mathcal{P}(\mathbb{T})$.

The methodology is developed and explained in Section 4.

Contributions. In this work, we study the multiparty session subtyping relation $\leqslant$ (Definition 3.15). We show that $\leqslant$ can be considered a canonical subtyping for the synchronous multiparty session calculus presented in Section 2: we develop the 4-steps approach outlined above, and show that $\leqslant$ is operationally sound (Theorem 3.21) and complete (Theorem 4.18), hence *precise*. From these results, we also prove that $\leqslant$ is *denotationally precise* (Theorem 6.2), when characteristic processes (**[Step 2]** above) are used as denotations. As an additional result, we present an algorithm for deciding $\leqslant$, proving that it is terminating (Lemma 3.25), and sound and complete (Theorem 3.26).

This work is an extended version of a workshop paper presented at PLACES 2015 [22], with:

- a revised theoretical development, including:

  - a novel coinductive treatment of global type projections (Definition 3.6), based on global and local type trees (Remarks 3.10 and 3.14);
  - a revised session subtyping algorithm (Table 6);
  - a revised definition of the negated subtyping relation (Definition 4.3);
- more discussion and examples (especially in Section 3);
- revised and detailed proofs.

Outline. In Section 2, we define the syntax and the semantics of a synchronous multiparty session calculus. At the end of this section we define when a multiparty session gets stuck. Section 3 introduces the type system and shows its soundness (well-typed multiparty sessions do not get stuck); it also introduces the multiparty session subtyping $\leqslant$, with a decision algorithm: the latter (described in Section 3.4) gives us some technical tools that we use to prove our main results. In Section 4, we prove that the subtyping relation introduced in Section 3 is operationally precise. Section 5 provides an illustrative example. In Section 6 we show that the subtyping relation from Section 3 is denotationally precise. Section 7 concludes with a brief summary of the work and discussion of the related work. The appendices contain detailed proofs and further technical discussion.

2. Synchronous multiparty session calculus

This section introduces the syntax and semantics of a synchronous multiparty session calculus. Since our focus is on subtyping, we simplify the calculus in [39] eliminating both shared channels for session initiations and session channels for communications inside sessions — i.e., our calculus is akin to value-passing CCS [46, Chapter 2.8]. In Section 4 and Section 6, we prove the preciseness of the subtyping in this simplified calculus.

Notation 2.1 *(Base sets).* We use the following base sets: values, ranged over by $\mathsf{v}, \mathsf{v}', \ldots$; expressions, ranged over by $\mathsf{e}, \mathsf{e}', \ldots$; expression variables, ranged over by $x, y, z \ldots$; labels, ranged over by $\ell, \ell', \ldots$; session participants, ranged over by $\mathsf{p}, \mathsf{q}, \ldots$; process variables, ranged over by $X, Y, \ldots$; processes, ranged over by $P, Q, \ldots$; and *multiparty sessions*, ranged over by $\mathcal{M}, \mathcal{M}', \ldots$.

Syntax. A value v can be a natural number n, an integer i, or a boolean true / false. An expression e can be a variable, a value, or a term built from expressions by applying the operators $\mathtt{succ}, \mathtt{neg}, \neg, \oplus$, or the relation $>$. An *evaluation context* $\mathcal{E}$ is an expression with exactly one hole. The only non-standard operator is $\oplus$, that models non-determinism: $\mathsf{e}_1 \oplus \mathsf{e}_2$ is an expression that might yield either e_1 or e_2. This will be useful later on, to obtain compact definitions and statements (especially Definition 4.8).

The processes of the synchronous multiparty session calculus are defined by:

$$P \ ::= \ \mathsf{p}!\ell(\mathsf{e}).P \ \mid \ \sum_{i\in I} \mathsf{p}?\ell_i(x_i).P_i \ \mid \ \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ P\ \mathsf{else}\ P \ \mid \ \mu X.P \ \mid \ X \ \mid \ \mathbf{0}$$

The output process $\mathsf{p}!\ell(\mathsf{e}).Q$ sends the value of expression e with label ℓ to participant p. The sum of input processes (external choice) $\sum_{i\in I} \mathsf{p}?\ell_i(x_i).P_i$ is a process that can accept a value with label ℓ_i from participant p, for any $i \in I$. According to the label ℓ_i of the received value, the variable x_i is instantiated with the value in the continuation process P_i. We assume that the set I is always finite and non-empty. The conditional process $\mathsf{if}\ \mathsf{e}\ \mathsf{then}\ P\ \mathsf{else}\ Q$ represents the internal choice between processes P and Q. Which branch of the conditional process will be taken depends on the evaluation of the expression e. The process $\mu X.P$ is a recursive process. We assume that the recursive processes are *guarded*. For example, $\mu X.\mathsf{p}?\ell(x).X$ is a valid process, while $\mu X.X$ is not.

We define a *multiparty session* as a parallel composition of pairs (denoted by $\mathsf{p} \triangleleft P$) of participants and processes:

$$\mathcal{M} \ ::= \ \mathsf{p} \triangleleft P \ \mid \ \mathcal{M} \mid \mathcal{M}$$

with the intuition that process P plays the role of participant p, and can interact with other processes playing other roles in $\mathcal{M}$. A multiparty session is *well formed* if all its participants are different. We consider only well-formed multiparty sessions.

Operational semantics. *The value* v *of expression* e (notation $\mathsf{e} \downarrow \mathsf{v}$) is computed as expected, see Table 1. The successor operation $\mathtt{succ}$ is defined only on natural numbers, the negation $\mathtt{neg}$ is defined on integers, and $\neg$ is defined only on boolean values. The internal choice $\mathsf{e}_1 \oplus \mathsf{e}_2$ evaluates either to the value of e_1 or to the value of e_2.

The *computational rules of multiparty sessions* are given in Table 3. They are closed with respect to the structural congruence defined in Table 2. In rule [R-COMM], the participant q sends the value v choosing the label ℓ_j to participant p, who offers inputs on all labels ℓ_i with $i \in I$. In rules [T-CONDITIONAL] and [F-CONDITIONAL], the participant p chooses to continue as P if the condition e evaluates to true and as Q if e evaluates to false. Rule [R-STRUCT] states that the reduction relation is closed with respect to structural congruence. We use $\longrightarrow^*$ with the standard meaning.

We adopt some standard conventions regarding the syntax of processes and sessions. Namely, we will use $\prod_{i\in I} \mathsf{p}_i \triangleleft P_i$ as short for $\mathsf{p}_1 \triangleleft P_1 \ \mid \ \ldots \ \mid \ \mathsf{p}_n \triangleleft P_n$, where $I = \{1, \ldots, n\}$. We will sometimes use infix notation for external choice process. For example, instead of $\sum_{i\in\{1,2\}} \mathsf{p}?\ell_i(x).P_i$, we will write $\mathsf{p}?\ell_1(x).P_1 + \mathsf{p}?\ell_2(x).P_2$.

Table 1
Expression evaluation.

$$\mathtt{succ}(n)\downarrow(n+1) \qquad \mathtt{neg}(i)\downarrow(-i) \qquad \neg\mathsf{true}\downarrow\mathsf{false} \qquad \neg\mathsf{false}\downarrow\mathsf{true} \qquad v\downarrow v$$

$$(i_1 > i_2)\downarrow \begin{cases} \mathsf{true} & \text{if } i_1 > i_2, \\ \mathsf{false} & \text{otherwise} \end{cases} \qquad \frac{e_1\downarrow v}{e_1 \oplus e_2 \downarrow v} \qquad \frac{e_2\downarrow v}{e_1 \oplus e_2 \downarrow v} \qquad \frac{e\downarrow v \quad \mathcal{E}(v)\downarrow v'}{\mathcal{E}(e)\downarrow v'}$$

Table 2
Structural congruence.

$$[\text{S-REC}]\ \mu X.P \equiv P\{\mu X.P/X\} \qquad [\text{S-MULTI}]\ P \equiv Q \Rightarrow p \triangleleft P \mid \mathcal{M} \equiv p \triangleleft Q \mid \mathcal{M}$$

$$[\text{S-PAR 1}]\ p \triangleleft \mathbf{0} \mid \mathcal{M} \equiv \mathcal{M} \qquad [\text{S-PAR 2}]\ \mathcal{M} \mid \mathcal{M}' \equiv \mathcal{M}' \mid \mathcal{M} \qquad [\text{S-PAR 3}]\ (\mathcal{M} \mid \mathcal{M}') \mid \mathcal{M}'' \equiv \mathcal{M} \mid (\mathcal{M}' \mid \mathcal{M}'')$$

Table 3
Reduction rules.

$$[\text{R-COMM}]\ \frac{j \in I \qquad e\downarrow v}{p \triangleleft \sum_{i\in I} q?\ell_i(x).P_i \mid q \triangleleft p!\ell_j(e).Q \mid \mathcal{M} \longrightarrow p \triangleleft P_j\{v/x\} \mid q \triangleleft Q \mid \mathcal{M}}$$

$$[\text{T-CONDITIONAL}]\ \frac{e\downarrow \mathsf{true}}{p \triangleleft \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \mid \mathcal{M} \longrightarrow p \triangleleft P \mid \mathcal{M}} \qquad [\text{F-CONDITIONAL}]\ \frac{e\downarrow \mathsf{false}}{p \triangleleft \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \mid \mathcal{M} \longrightarrow p \triangleleft Q \mid \mathcal{M}}$$

$$[\text{R-STRUCT}]\ \frac{\mathcal{M}'_1 \equiv \mathcal{M}_1 \quad \mathcal{M}_1 \longrightarrow \mathcal{M}_2 \quad \mathcal{M}_2 \equiv \mathcal{M}'_2}{\mathcal{M}'_1 \longrightarrow \mathcal{M}'_2}$$

Example 2.2. We now show the operational semantics in action. Consider the following multiparty session with three participants, `Alice`, `Bob` and `Carol`:

$$\mathcal{M} = \mathtt{Alice} \triangleleft P_{\mathtt{Alice}} \mid \mathtt{Bob} \triangleleft P_{\mathtt{Bob}} \mid \mathtt{Carol} \triangleleft P_{\mathtt{Carol}}$$

where

$$\begin{aligned} P_{\mathtt{Alice}} &= \mathtt{Bob}!\ell_1(50).\mathtt{Carol}?\ell_3(x).\mathbf{0} \\ P_{\mathtt{Bob}} &= \mathtt{Alice}?\ell_1(x).\mathtt{Carol}!\ell_2(100).\mathbf{0} + \mathtt{Alice}?\ell_4(x).\mathtt{Carol}!\ell_2(2).\mathbf{0} \\ P_{\mathtt{Carol}} &= \mathtt{Bob}?\ell_2(x).\mathtt{Alice}!\ell_3(\mathtt{succ}(x)).\mathbf{0} \end{aligned}$$

This multiparty session reduces to

$$\mathtt{Alice} \triangleleft \mathbf{0} \mid \mathtt{Bob} \triangleleft \mathbf{0} \mid \mathtt{Carol} \triangleleft \mathbf{0}$$

after three communications occur. First, `Alice` sends to `Bob` natural number 50 with the label ℓ_1. `Bob` is able to receive values with labels ℓ_1 and ℓ_4. Next, the only possible communication is between `Bob` and `Carol`. So, `Carol` receives natural number 100 from `Bob`. The value 100 is substituted in the continuation process. Finally, since $\mathtt{succ}(100)\downarrow 101$, `Carol` sends 101 to `Alice`. We can then reduce the session to, for example, $\mathtt{Alice} \triangleleft \mathbf{0}$, but not further.

From the end of Example 2.2, we can see that a session $\mathcal{M}$ always has at least one participant, since we do not have neutral element for the parallel composition. In Section 3, we will introduce a type system ensuring that if a well-typed multiparty session has only one participant, then the corresponding process is $\mathbf{0}$ – hence, the participant's process has no inputs/outputs to perform.

In order to define the operational preciseness of subtyping (in Section 4), it is crucial to formalise when a multiparty session contains communications that will never be executed.

Definition 2.3. A multiparty session $\mathcal{M}$ is *stuck* if $\mathcal{M} \not\equiv p \triangleleft \mathbf{0}$ and there is no multiparty session $\mathcal{M}'$ such that $\mathcal{M} \longrightarrow \mathcal{M}'$. A multiparty session $\mathcal{M}$ gets *stuck*, notation $\mathtt{stuck}(\mathcal{M})$, if it reduces to a stuck multiparty session.

E.g., the multiparty session $\mathcal{M}$ in Example 2.2 is not stuck, and it does not get stuck. A similar multiparty session, where instead of $P_{\mathtt{Alice}}$ we take $P'_{\mathtt{Alice}} = \mathtt{Bob}!\ell_1(50).\mathtt{Carol}?\ell_5(x).\mathbf{0}$, gets stuck because of label mismatch.

3. Type system

This section introduces a type system for the calculus presented in Section 2. The formulation is based on Kouzapas and Yoshida [39,40], with adaptations to account for our simplified calculus. We formalise types and projections (Section 3.1), the subtyping relation (Section 3.2), and the typing rules and their properties (Section 3.3).

We also prove that the subtyping relation is decidable, by developing an algorithmic characterisation (Section 3.4): in doing this, we introduce the tools for proving our main results in Section 4.

3.1. Types and projections

Global types provide global conversation scenarios of multiparty sessions, with a bird's eye view describing the message exchanges between pairs of participants.

Definition 3.1 *(Sorts and global types). Sorts*, ranged over by S, are defined as:

$$S \quad ::= \quad \texttt{nat} \mid \texttt{int} \mid \texttt{bool}$$

Global types, ranged over by $\mathbb{G}$, are terms generated by the following grammar:

$$\mathbb{G} \quad ::= \quad \texttt{end} \mid \mu\mathbf{t}.\mathbb{G} \mid \mathbf{t} \mid \mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathbb{G}_i\}_{i\in I}$$

We require that $\mathsf{p} \neq \mathsf{q}$, $I \neq \emptyset$, and $\ell_i \neq \ell_j$ whenever $i \neq j$, for all $i, j \in I$. We postulate that recursion is guarded. Unless otherwise noted, global types are *closed*: a recursion variable $\mathbf{t}$ only occurs bounded by $\mu\mathbf{t}.\ldots$

In Definition 3.1, the type $\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathbb{G}_i\}_{i\in I}$ formalises a protocol where participant p must send to q one message with label ℓ_i and a value of type S_i as payload, for some $i \in I$; then, depending on which ℓ_i was sent by p, the protocol continues as $\mathbb{G}_i$. Value types are restricted to sorts, that can be natural (`nat`), integer (`int`) and boolean (`bool`). The type `end` represents a terminated protocol. Recursive protocol is modelled as $\mu\mathbf{t}.\mathbb{G}$, where recursion variable $\mathbf{t}$ is bound and guarded in $\mathbb{G}$ – e.g., $\mu\mathbf{t}.\mathsf{p} \to \mathsf{q} : \ell(\texttt{nat}).\mathbf{t}$ is a valid global type, whereas $\mu\mathbf{t}.\mathbf{t}$ is not.

We define the *set of participants of a global type* $\mathbb{G}$, by structural induction on $\mathbb{G}$, as follows:

$$\mathtt{pt}\{\mu\mathbf{t}.\mathbb{G}\} = \mathtt{pt}\{\mathbb{G}\} \qquad \mathtt{pt}\{\mathtt{end}\} = \mathtt{pt}\{\mathbf{t}\} = \emptyset \qquad \mathtt{pt}\{\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathbb{G}_i\}_{i\in I}\} = \{\mathsf{p}, \mathsf{q}\} \cup \mathtt{pt}\{\mathbb{G}_i\}\ (i \in I)$$

We will often write $\mathsf{p} \in \mathbb{G}$ instead of $\mathsf{p} \in \mathtt{pt}\{\mathbb{G}\}$.

Global type trees, equality, and balancing. In the rest of this paper, we will often work on *global type trees* $\mathcal{T}(\mathbb{G})$, i.e., possibly infinite regular trees whose structure derives from the syntax of $\mathbb{G}$ in Definition 3.1. The approach is standard, and based on [53, Part IV]: we provide a visual representation of type trees in Fig. 1, and summarise the main elements in Notation 3.2 below; for the full technical details, see Appendix A.1.

Notation 3.2. We write $\mathbb{G} = \mathbb{G}'$ whenever $\mathcal{T}(\mathbb{G}) = \mathcal{T}(\mathbb{G}')$ (details: Definition A.1, Remark A.12), thus equating global types up-to the unfolding of their recursive subterms [53, Chapter 20]. We will write $\mathbb{G} \equiv \mathbb{G}'$ for the *syntactic* type equality. Hence, a recursive global type $\mu\mathbf{t}.\mathbb{G}$ and its unfolding $\mathbb{G}\{\mu\mathbf{t}.\mathbb{G}/\mathbf{t}\}$ are related by $=$, but *not* by $\equiv$. To save some notation, we will often write G to refer to a type tree $\mathcal{T}(\mathbb{G})$ (for some $\mathbb{G}$); correspondingly, we will write $\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}$ for a type tree $\mathcal{T}(\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathbb{G}_i\}_{i\in I})$ (for some $\mathbb{G}_i$). The set of participants of tree $\mathcal{T}(\mathbb{G})$ is the set of participants of the corresponding global type $\mathbb{G}$, i.e., $\mathtt{pt}\{\mathcal{T}(\mathbb{G})\} = \mathtt{pt}\{\mathbb{G}\}$. (For more details, see Notation A.6.)

From now on, we will only consider *balanced global types*, as per Definition 3.3 below. Roughly, a global type is balanced if *all* its active participants are involved in some communication within a finite number of message exchanges. When inspecting the participants of a global type $\mathbb{G}$, the balancing requirement assures that, if $\mathbb{G} = \mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathbb{G}_i\}_{i\in I}$, then $\mathtt{pt}\{\mathbb{G}_i\} = \mathtt{pt}\{\mathbb{G}_j\}$ for all $i, j \in I$; and consequently, in a recursive type $\mu\mathbf{t}.\mathbb{G}$, all participants are involved in some communication in every path between $\mu\mathbf{t}.\ldots$ and an occurrence of $\mathbf{t}$ in $\mathbb{G}$. Similar constraints are enforced in most works on multiparty session types, albeit with different means; we will exemplify and further discuss balancing later, in Example 3.12 (case $\mathbb{G}''$) and Remark 3.14.

Definition 3.3 *(Balanced global types).* We say that a node n of a global type tree *involves participant* p iff n is marked with $\mathsf{p} \to \mathsf{q}$ or $\mathsf{q} \to \mathsf{p}$ (for some q). We say that a global type $\mathbb{G}$ *is balanced* iff, for each node n of $\mathcal{T}(\mathbb{G})$, whenever a node involving some p is reachable from n, there exists a finite k such that within k steps *all* paths starting from n reach some node involving p. (For a more formal account, see Definition A.28.)

A *(multiparty) session type* describes the behaviour of a single participant in a multiparty session.

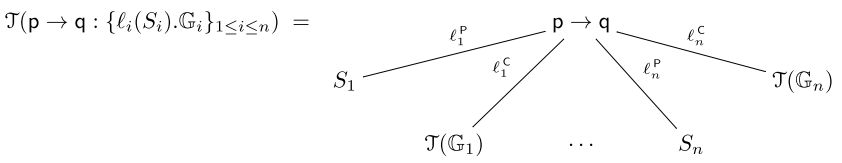

Fig. 1. Graphical representation of global type trees yielded by the function $\mathcal{T}(\mathbb{G})$ (for a more formal definition, see Appendix A.1). $\mathcal{T}(\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathbb{G}_i\}_{1\le i\le n})$ yields a type tree where the edges annotated with ℓ_i^{C} are *continuation edges*, pointing to the type sub-tree of $\mathbb{G}_i$; those annotated with ℓ_i^{P}, instead, are *payload edges*, pointing to a leaf node with the sort S_i. $\mathcal{T}(\mathtt{end})$ yields a type tree leaf node, marked as $\mathtt{end}$. The tree of a recursive type $\mu\mathbf{t}.\mathbb{G}$ is a possibly infinite tree, obtained via unfolding: for an example, see Fig. 2.

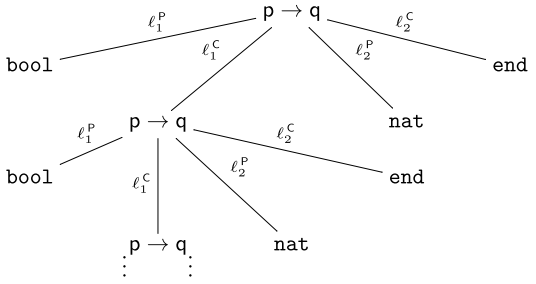

Fig. 2. The infinite global type tree of: $\mu\mathbf{t}.\mathsf{p} \to \mathsf{q} : \{\ell_1(\mathtt{bool}).\mathbf{t},\ \ell_2(\mathtt{nat}).\mathtt{end}\}$.

Definition 3.4 *(Multiparty Session Types).* The grammar of session types, ranged over by $\mathbb{T}$, is:

$$\mathbb{T} ::= \mathtt{end} \;\mid\; \textstyle\bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathbb{T}_i \;\mid\; \bigvee_{i\in I} \mathsf{q}!\ell_i(S_i).\mathbb{T}_i \;\mid\; \mu\mathbf{t}.\mathbb{T} \;\mid\; \mathbf{t}$$

We require that $\ell_i \neq \ell_j$ whenever $i \neq j$, for all $i, j \in I$. We postulate that recursion is always guarded. Unless otherwise noted, session types are closed.

Note that, according to the previous definition, labels in a type need to be pairwise different. For example, $\mathsf{p}?\ell(\mathtt{int}).\mathtt{end} \wedge \mathsf{p}?\ell(\mathtt{nat}).\mathtt{end}$ is not a type.

The session type $\mathtt{end}$ says that no further communication is possible and the protocol is completed. The *external choice* or *branching type* $\bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathbb{T}_i$ requires to wait to receive a value of sort S_i (for some $i \in I$) from the participant p, via a message with label ℓ_i; if the received message has label ℓ_i, the protocol will continue as prescribed by $\mathbb{T}_i$. The *internal choice* or *selection type* $\bigvee_{i\in I} \mathsf{q}!\ell_i(S_i).\mathbb{T}_i$ says that the participant implementing the type must choose a labelled message to send to q; if the participant chooses the message ℓ_i, for some $i \in I$, it must include in the message to q a payload value of sort S_i, and continue as prescribed by $\mathbb{T}_i$. Recursion is modelled by the session type $\mu\mathbf{t}.\mathbb{T}$. We adopt the following conventions: we do not write branch/selection symbols in case of a singleton choice, we do not write unnecessary parentheses, and we often omit trailing $\mathtt{end}$s.

The set $\mathtt{pt}\{\mathbb{T}\}$ of participants of a session type $\mathbb{T}$ is defined inductively as follows

$$\mathtt{pt}\{\bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathbb{T}_i\} = \mathtt{pt}\{\bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathbb{T}_i\} = \{\mathsf{p}\} \cup \bigcup_{i\in I} \mathtt{pt}\{\mathbb{T}_i\} \quad \mathtt{pt}\{\mu\mathbf{t}.\mathbb{T}\} = \mathtt{pt}\{\mathbb{T}\} \quad \mathtt{pt}\{\mathbf{t}\} = \mathtt{pt}\{\mathtt{end}\} = \emptyset.$$

In a similar way to global types, we will often work on *session type trees* $\mathcal{T}(\mathbb{T})$, shown in Fig. 3.

Notation 3.5. We write $\mathbb{T} = \mathbb{T}'$ iff $\mathcal{T}(\mathbb{T}) = \mathcal{T}(\mathbb{T}')$ (i.e., for equality up-to unfolding — details: Definition A.1 and Remark A.18), and $\mathbb{T} \equiv \mathbb{T}'$ for the syntactic session type equality. Hence, a recursive session type $\mu\mathbf{t}.\mathbb{T}$ and its unfolding $\mathbb{T}\{^{\mu\mathbf{t}.\mathbb{T}}/_{\mathbf{t}}\}$ are related by $=$, but *not* by $\equiv$. To save some notation, we will often write T to refer to a type tree $\mathcal{T}(\mathbb{T})$ (for some $\mathbb{T}$); correspondingly, we will write, e.g., $\mathsf{p}?\ell(\mathtt{int}).\mathsf{T}'$ for a type tree $\mathcal{T}(\mathsf{p}?\ell(\mathtt{int}).\mathbb{T}')$ (for some $\mathbb{T}'$). (For more details, see Notation A.15.)

In Definition 3.6 below, we define the *global type projection* as a coinductive relation $\mathsf{G} \upharpoonright_{\mathsf{r}} \mathsf{T}$ between global and session type trees. Our definition extends the one originally proposed by Honda et al. [34], Honda et al. [35], along the lines of Yoshida et al. [60] and Deniélou et al. [19]: i.e., it uses a *merging operator* $\sqcap$ to combine multiple session type trees into a single tree.

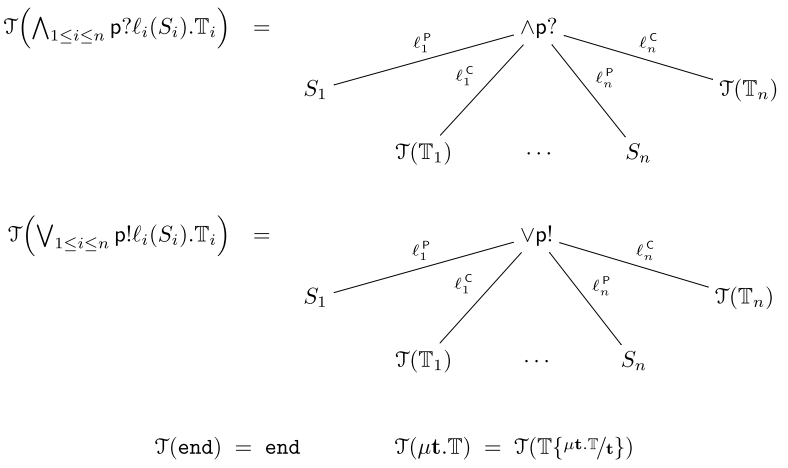

Fig. 3. Graphical representation of session type trees. The intuition is similar to Fig. 1, but here we have distinct nodes for internal and external choices. For a more formal definition, see Appendix A.2.

Definition 3.6. The *projection of a global type onto a participant* r is the largest relation $\upharpoonright_{\mathsf{r}}$ between global type trees and session type trees such that, whenever $\mathsf{G} \upharpoonright_{\mathsf{r}} \mathsf{T}$:

- $\mathsf{r} \notin \mathtt{pt}\{\mathsf{G}\}$ implies $\mathsf{T} = \mathtt{end}$; [PROJ-END]
- $\mathsf{G} = \mathsf{p} \to \mathsf{r} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}$ implies $\mathsf{T} = \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i$, and $\mathsf{G}_i \upharpoonright_{\mathsf{r}} \mathsf{T}_i$, $\forall i \in I$; [PROJ-IN]
- $\mathsf{G} = \mathsf{r} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}$ implies $\mathsf{T} = \bigvee_{i\in I} \mathsf{q}!\ell_i(S_i).\mathsf{T}_i$, and $\mathsf{G}_i \upharpoonright_{\mathsf{r}} \mathsf{T}_i$, $\forall i \in I$; [PROJ-OUT]
- $\mathsf{G} = \mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}$ and $\mathsf{r} \notin \{\mathsf{p}, \mathsf{q}\}$ implies that there are T_i, $i \in I$ such that $\mathsf{T} = \bigsqcap_{i\in I} \mathsf{T}_i$, and $\mathsf{G}_i \upharpoonright_{\mathsf{r}} \mathsf{T}_i$, for every $i \in I$. [PROJ-CONT]

Above, $\bigsqcap$ is the *merging operator*, that is a partial operation over session type trees defined as:

$$\mathsf{T}_1 \bigsqcap \mathsf{T}_2 = \begin{cases} \mathsf{T}_1 & \text{if } \mathsf{T}_1 = \mathsf{T}_2 \qquad \text{[MRG-ID]} \\ \mathsf{T}_3 & \text{if } \exists I, J : \begin{cases} \mathsf{T}_1 = \bigwedge_{i\in I} \mathsf{p}'?\ell_i(S_i).\mathsf{T}_i & \text{and} \\ \mathsf{T}_2 = \bigwedge_{j\in J} \mathsf{p}'?\ell_j(S_j).\mathsf{T}_j & \text{and} \\ \mathsf{T}_3 = \bigwedge_{k\in I\cup J} \mathsf{p}'?\ell_k(S_k).\mathsf{T}_k \end{cases} \qquad \text{[MRG-BRA]} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

We naturally lift $\upharpoonright_{\mathsf{r}}$ to syntactic types: i.e., $\mathbb{G} \upharpoonright_{\mathsf{r}} \mathbb{T}$ holds iff $\mathcal{T}(\mathbb{G}) \upharpoonright_{\mathsf{r}} \mathcal{T}(\mathbb{T})$.

The attentive reader might have noticed that in Definition 3.6, we use a multi-way merging operator $\bigsqcap_{i\in I} \mathsf{T}_i$: it is naturally defined from the binary merging in Definition 3.6, which is associative, by Proposition 3.7 below.

Proposition 3.7. *The merging operation is associative, i.e.:* $\mathsf{T} \sqcap (\mathsf{T}' \sqcap \mathsf{T}'') = (\mathsf{T} \sqcap \mathsf{T}') \sqcap \mathsf{T}''$.

Proof. See Appendix A.4. □

Note that our definition of $\bigsqcap$ is slightly simplified w.r.t. the one of Yoshida et al. [60] and Deniélou et al. [19]: rule [MRG-BRA] does not use $\bigsqcap$ recursively to merge the continuations T_i and T_j, and this restricts the set of mergeable session types. This simplification is harmless for our purposes.[1] By Definition 3.6, merging a type tree with itself results in the tree itself (rule [MRG-ID]). Moreover, Definition 3.6 allows to combine different external choices (rule [MRG-BRA]) *if and only if* common labels have identical sorts and identical continuations, as formalised in Proposition 3.8 below and illustrated in Examples 3.9, 3.12 and 3.13.

Proposition 3.8. *For two type trees* $\mathsf{T}' = \bigwedge_{i\in I} \mathsf{p}'?\ell_i(S_i).\mathsf{T}_i$ *and* $\mathsf{T}'' = \bigwedge_{j\in J} \mathsf{p}''?\ell_j(S_j).\mathsf{T}_j$, *we have that* $\mathsf{T}' \sqcap \mathsf{T}''$ *is defined* if and only if $\mathsf{p}' = \mathsf{p}''$ *and, whenever* $\ell_i = \ell_j$ *(for some* $i \in I$ *and* $j \in J$*),* $S_i = S_j$ *and* $\mathsf{T}_i = \mathsf{T}_j$.

[1] In particular, our restricted merging operator is sufficient to project the characteristic global types introduced in Section 4.3. This would not change if we used a more powerful operator supporting all the mergeable types of Yoshida et al. [60] and Deniélou et al. [19].

Proof. See Appendix A.4. □

Example 3.9. We now give some small examples that illustrate the definition of the merging operator (here, $i \neq j$ implies $\ell_i \neq \ell_j$):

$$\begin{aligned}
\mathsf{q}!\ell(\mathtt{nat}) \sqcap \mathsf{q}!\ell(\mathtt{nat}) &= \mathsf{q}!\ell(\mathtt{nat}) \\
\mathsf{p}!\ell(\mathtt{nat}) \sqcap \mathsf{q}!\ell(\mathtt{nat}) &\text{ undefined: outputs to different participants} \\
\mathsf{q}!\ell_3(\mathtt{nat}) \sqcap \mathsf{q}!\ell_4(\mathtt{nat}) &\text{ undefined: outputs with different labels} \\
\big(\mathsf{q}?\ell_3(\mathtt{int}) \wedge \mathsf{q}?\ell_5(\mathtt{nat})\big) \sqcap \big(\mathsf{q}?\ell_4(\mathtt{int}) \wedge \mathsf{q}?\ell_5(\mathtt{nat})\big) &= \mathsf{q}?\ell_3(\mathtt{int}) \wedge \mathsf{q}?\ell_4(\mathtt{int}) \wedge \mathsf{q}?\ell_5(\mathtt{nat}) \\
\mathsf{q}?\ell_3(\mathtt{nat}) \sqcap \mathsf{q}?\ell_3(\mathtt{nat}).\mathsf{q}?\ell_3(\mathtt{nat}) &\text{ undefined: same prefixes, but different continuations} \\
\mathsf{q}?\ell(\mathtt{nat}) \sqcap \mathsf{q}?\ell(\mathtt{int}) &\text{ undefined: the payload sorts do not match}
\end{aligned}$$

Remark 3.10. Before describing the clauses of Definition 3.6, we emphasise that, unlike most papers on session types, our projection is coinductive and, by reasoning on type trees instead of type syntax, it relates syntactic types up-to unfolding. When session types are considered equal up-to unfolding (Notation 3.5), our projection is a partial function: i.e., for all syntactic $\mathbb{G}, \mathbb{T}, \mathbb{T}'$, $\mathbb{G} \upharpoonright_{\mathsf{r}} \mathbb{T}$ and $\mathbb{G} \upharpoonright_{\mathsf{r}} \mathbb{T}'$ implies $\mathbb{T} = \mathbb{T}'$, since $\mathcal{T}(\mathbb{T}) = \mathcal{T}(\mathbb{T}')$ (Proposition 3.11). Therefore, we will often write $\mathsf{G}\upharpoonright\mathsf{r}$ whenever $\mathsf{G} \in \mathrm{dom}(\upharpoonright_{\mathsf{r}})$ (i.e., when projection is defined) to refer to the *unique* type tree T such that $\mathsf{G} \upharpoonright_{\mathsf{r}} \mathsf{T}$.

Proposition 3.11. *The projection relation $\upharpoonright_{\mathsf{r}}$ is a partial function.*

Proof. See Appendix A.3 □

We now describe the clauses of Definition 3.6:

- clause [PROJ-END] states that when a global type G is projected onto a participant r who does *not* appear in G (e.g., because $\mathsf{G} = \mathtt{end}$), then the result is the inactive session type $\mathtt{end}$;
- clause [PROJ-IN] (resp. [PROJ-OUT]) states that a global type G starting with a communication from p to r (resp. from r to q) projects onto an external (resp. internal) choice T, provided that the continuations of T are also projections of the corresponding global type continuations.
- clause [PROJ-CONT] states that if G starts with a communication between p and q, and we are projecting G onto a third participant r, then we need to *(1)* skip the initial communication, *(2)* project all the continuations onto r, and *(3) merge* the resulting session type trees, using the *merging operator* $\sqcap$.

As a result, clause [PROJ-CONT] of Definition 3.6 allows participant r to receive different messages (from a same participant p') in different branches of a global type, as shown in Example 3.12 below.

Example 3.12. We demonstrate interesting points of Definition 3.6 and Definition 3.3. First, we show some projections of global types. Consider the global type:

$$\mathbb{G} = \mathsf{p} \to \mathsf{q} : \{\ell_1(\mathtt{nat}).\mathbb{G}_1, \ell_2(\mathtt{bool}).\mathbb{G}_2\} \qquad \text{where } \begin{cases} \mathbb{G}_1 = \mathsf{q} \to \mathsf{r} : \{\ell_3(\mathtt{int}), \ell_5(\mathtt{nat})\} \\ \mathbb{G}_2 = \mathsf{q} \to \mathsf{r} : \{\ell_4(\mathtt{int}), \ell_5(\mathtt{nat})\} \\ \mathsf{r} \neq \mathsf{p} \end{cases}$$

We have:

$$\mathbb{G}\upharpoonright\mathsf{p} = \mathsf{q}!\ell_1(\mathtt{nat}).(\mathbb{G}_1\upharpoonright\mathsf{p}) \vee \mathsf{q}!\ell_2(\mathtt{bool}).(\mathbb{G}_2\upharpoonright\mathsf{p}) = \mathsf{q}!\ell_1(\mathtt{nat}).\mathtt{end} \vee \mathsf{q}!\ell_2(\mathtt{bool}).\mathtt{end}$$

$$\begin{aligned}
\mathbb{G}\upharpoonright\mathsf{q} &= \mathsf{p}?\ell_1(\mathtt{nat}).(\mathbb{G}_1\upharpoonright\mathsf{q}) \wedge \mathsf{p}?\ell_2(\mathtt{bool}).(\mathbb{G}_2\upharpoonright\mathsf{q}) \\
&= \mathsf{p}?\ell_1(\mathtt{nat}).\big(\mathsf{r}!\ell_3(\mathtt{int}) \vee \mathsf{r}!\ell_5(\mathtt{nat})\big) \wedge \mathsf{p}?\ell_2(\mathtt{bool}).\big(\mathsf{r}!\ell_4(\mathtt{int}) \vee \mathsf{r}!\ell_5(\mathtt{nat})\big)
\end{aligned}$$

$$\begin{aligned}
\mathbb{G}\upharpoonright\mathsf{r} &= \mathbb{G}_1\upharpoonright\mathsf{r} \sqcap \mathbb{G}_2\upharpoonright\mathsf{r} = \big(\mathsf{q}?\ell_3(\mathtt{int}) \wedge \mathsf{q}?\ell_5(\mathtt{nat})\big) \sqcap \big(\mathsf{q}?\ell_4(\mathtt{int}) \wedge \mathsf{q}?\ell_5(\mathtt{nat})\big) \\
&= \mathsf{q}?\ell_3(\mathtt{int}) \wedge \mathsf{q}?\ell_4(\mathtt{int}) \wedge \mathsf{q}?\ell_5(\mathtt{nat})
\end{aligned}$$

Note that in $\mathbb{G}$, q could output different messages towards r, depending on whether p sends ℓ_1 or ℓ_2 to q; therefore, in $\mathbb{G}\upharpoonright\mathsf{r}$, the possible inputs of r in $\mathbb{G}_1$ and $\mathbb{G}_2$ are merged into a larger external choice that supports all possible outputs of q.

Let us now consider the global type $\mathbb{G}' = \mu\mathbf{t}.\mathsf{p} \to \mathsf{q} : \ell(\mathtt{nat}).\mathbf{t}$. When projecting $\mathbb{G}'$ onto p, by Definition 3.6 and Fig. 1 we have the following equation:

$$\mathcal{T}\big(\mathbb{G}'\big)\upharpoonright\mathsf{p} = (\mathsf{p} \to \mathsf{q} : \ell(\mathtt{nat}).\mathcal{T}\big(\mathbb{G}'\big))\upharpoonright\mathsf{p} = \mathsf{q}!\ell(\mathtt{nat}).(\mathcal{T}\big(\mathbb{G}'\big)\upharpoonright\mathsf{p})$$

whose solution is the infinite type tree of the session type $\mu\mathbf{t}.\mathsf{q}!\ell(\mathtt{nat}).\mathbf{t}$ (see Fig. 3). Similarly, when projecting $\mathbb{G}'$ onto q, we obtain $\mathcal{T}\big(\mathbb{G}'\big)\upharpoonright\mathsf{q} = \mathcal{T}(\mu\mathbf{t}.\mathsf{p}?\ell(\mathtt{nat}).\mathbf{t})$.

Finally, we motivate constraining the global types to balanced global types (Definition 3.3). Let us consider the following *non-balanced* global type:

$$\mathbb{G}'' = \mu \mathbf{t}.\mathsf{p} \to \mathsf{q} : \left\{ \ell(\mathtt{nat}).\mathbf{t} \ , \ \ell'(\mathtt{int}).\mathsf{q} \to \mathsf{r} : \ell''(\mathtt{int}).\mathtt{end} \right\}$$

Note that p and q can exchange an arbitrary number of messages ℓ; and whenever p decides to send ℓ' to q, then q sends ℓ'' to r, and the session terminates. This makes $\mathbb{G}''$ non-balanced: in fact, in its type tree, a node involving r is reachable from the root, but it is possible to follow the continuation edges marked by ℓ for an arbitrary depth, only visiting nodes marked $\mathsf{p} \to \mathsf{q}$, without encountering a node involving r.

When projecting the type tree of $\mathbb{G}''$, we obtain the following equations (with solutions shown on the right):

$$\begin{aligned} \mathcal{T}(\mathbb{G}'')\upharpoonright \mathsf{p} &= \mathsf{q}!\ell(\mathtt{nat}).\left(\mathcal{T}(\mathbb{G}'')\upharpoonright \mathsf{p}\right) \vee \mathsf{q}!\ell'(\mathtt{int}).\left(\mathcal{T}(\mathsf{q} \to \mathsf{r} : \ell''(\mathtt{int}).\mathtt{end})\upharpoonright \mathsf{p}\right) \\ &= \mathsf{q}!\ell(\mathtt{nat}).\left(\mathcal{T}(\mathbb{G}'')\upharpoonright \mathsf{p}\right) \vee \mathsf{q}!\ell'(\mathtt{int}).\mathcal{T}(\mathtt{end}) && = \mathcal{T}\left(\mu \mathbf{t}.\mathsf{q}!\ell(\mathtt{nat}).\mathbf{t} \vee \mathsf{q}!\ell'(\mathtt{int}).\mathtt{end}\right) \end{aligned}$$

$$\begin{aligned} \mathcal{T}(\mathbb{G}'')\upharpoonright \mathsf{q} &= \mathsf{p}?\ell(\mathtt{nat}).\left(\mathcal{T}(\mathbb{G}'')\upharpoonright \mathsf{q}\right) \vee \mathsf{p}?\ell'(\mathtt{int}).\left(\mathcal{T}(\mathsf{q} \to \mathsf{r} : \ell''(\mathtt{int}).\mathtt{end})\upharpoonright \mathsf{q}\right) \\ &= \mathsf{p}?\ell(\mathtt{nat}).\left(\mathcal{T}(\mathbb{G}'')\upharpoonright \mathsf{q}\right) \vee \mathsf{p}?\ell'(\mathtt{int}).\mathsf{r}!\ell''(\mathtt{int}) && = \mathcal{T}\left(\mu \mathbf{t}.\mathsf{p}?\ell(\mathtt{nat}).\mathbf{t} \wedge \mathsf{p}?\ell'(\mathtt{int}).\mathsf{r}!\ell''(\mathtt{int})\right) \end{aligned}$$

$$\mathcal{T}(\mathbb{G}'')\upharpoonright \mathsf{r} = \left(\mathcal{T}(\mathbb{G}'')\upharpoonright \mathsf{r}\right) \sqcap \left(\mathsf{q}?\ell''(\mathtt{int}).\mathcal{T}(\mathtt{end})\right) \qquad = \mathcal{T}\left(\mathsf{q}?\ell''(\mathtt{int}).\mathtt{end}\right)$$

and the projections also highlight that p and q can interact forever (exchanging ℓ-labelled messages) while r might wait forever for ℓ'' from q. This is often undesirable: in a well-formed multiparty interaction, under mild fair scheduling assumptions, each non-terminated participant should be always eventually involved in a message exchange. Our balancing requirement guarantees this, by rejecting problematic global types like $\mathbb{G}''$.

Importantly, by Definition 3.6, there exist global types that *cannot* be projected onto all their participants. This is because G might describe meaningless protocols, that cause the merging operation $\sqcap$ in clause [PROJ-CONT] to be undefined, as shown in Example 3.13 below.

Example 3.13. We show two global types that cannot be projected according to the Definition 3.6. Consider the global type $\mathbb{G} = \mathsf{p} \to \mathsf{q} : \{\ell_1(\mathtt{nat}).\mathbb{G}_1 \ , \ \ell_2(\mathtt{bool}).\mathbb{G}_2\}$, with $\mathbb{G}_1 = \mathsf{r} \to \mathsf{q} : \ell_3(\mathtt{nat})$ and $\mathbb{G}_2 = \mathsf{r} \to \mathsf{q} : \ell_4(\mathtt{nat})$. Then,

$$\begin{aligned} \mathbb{G}\upharpoonright \mathsf{p} &= \mathsf{q}!\ell_1(\mathtt{nat}) \vee \mathsf{q}!\ell_2(\mathtt{bool}) \\ \mathbb{G}\upharpoonright \mathsf{q} &= \mathsf{p}?\ell_1(\mathtt{nat}).\mathsf{r}?\ell_3(\mathtt{nat}) \wedge \mathsf{p}?\ell_2(\mathtt{bool}).\mathsf{r}?\ell_4(\mathtt{nat}) \\ \mathbb{G}\upharpoonright \mathsf{r} &= \mathsf{q}!\ell_3(\mathtt{nat}) \sqcap \mathsf{q}!\ell_4(\mathtt{nat}) \ \textbf{(undefined if } \ell_3 \neq \ell_4\textbf{)} \end{aligned}$$

Intuitively, when $\ell_3 \neq \ell_4$, $\mathbb{G}\upharpoonright \mathsf{r}$ is undefined because in $\mathbb{G}$, depending on whether p and q exchange ℓ_1 or ℓ_2, r is supposed to send either ℓ_3 or ℓ_4 to q; however, r is not privy to the interactions between p and q, and thus, $\mathbb{G}$ provides an invalid specification for r. Instead, if $\ell_3 = \ell_4$, then by Definition 3.6 we have $\mathbb{G}\upharpoonright \mathsf{r} = \mathsf{q}!\ell_3(\mathtt{nat}) \sqcap \mathsf{q}!\ell_3(\mathtt{nat}) = \mathsf{q}!\ell_3(\mathtt{nat})$.

Now, consider the global type $\mathbb{G}' = \mathsf{p} \to \mathsf{q} : \{\ell_1(\mathtt{nat}).\mathbb{G}'_1 \ , \ \ell_2(\mathtt{bool}).\mathbb{G}'_2\}$, with $\mathbb{G}'_1 = \mathsf{q} \to \mathsf{r} : \ell_3(\mathtt{nat})$ and $\mathbb{G}'_2 = \mathsf{q} \to \mathsf{r} : \ell_3(\mathtt{nat}).\mathsf{q} \to \mathsf{r} : \ell_3(\mathtt{nat})$. Then,

$$\begin{aligned} \mathbb{G}'\upharpoonright \mathsf{p} &= \mathsf{q}!\ell_1(\mathtt{nat}) \vee \mathsf{q}!\ell_2(\mathtt{bool}) \\ \mathbb{G}'\upharpoonright \mathsf{q} &= \mathsf{p}?\ell_1(\mathtt{nat}).\mathsf{r}!\ell_3(\mathtt{nat}) \wedge \mathsf{p}?\ell_2(\mathtt{bool}).\mathsf{r}!\ell_3(\mathtt{nat}).\mathsf{r}!\ell_3(\mathtt{nat}) \\ \mathbb{G}'\upharpoonright \mathsf{r} &= \mathsf{q}?\ell_3(\mathtt{nat}) \sqcap \mathsf{q}?\ell_3(\mathtt{nat}).\mathsf{q}?\ell_3(\mathtt{nat}) \ \textbf{(undefined)} \end{aligned}$$

Here, $\mathbb{G}'\upharpoonright \mathsf{r}$ is undefined because in $\mathbb{G}'$, depending on whether p and q exchange ℓ_1 or ℓ_2, r is supposed to receive either one or two messages ℓ_3 from q; however, as in the previous example, r is not aware of the interactions between p and q, and thus, $\mathbb{G}$ provides an invalid specification for r. This example could be fixed, e.g., by replacing ℓ_3 with $\ell' \neq \ell_3$ in $\mathbb{G}'_2$, or by letting $\mathbb{G}'_1 = \mathbb{G}'_2$: both fixes would make $\mathbb{G}'\upharpoonright \mathsf{r}$ defined, similarly to Example 3.12.

Remark 3.14. Our coinductive global type projection (Definition 3.6) is, in some respects, more flexible than the standard *inductive* projections commonly found in session types literature (e.g., in [19]). The key difference is that such projections, albeit similar to Definition 3.6, traverse *syntactic* types (rather than type trees), hence cannot relate types up-to unfolding. In fact, the inductive projection $\upharpoonright^i$, and its inductive merging $\sqcap^i$, are defined as follows for recursive terms:

$$\mu \mathbf{t}.\mathbb{G}\upharpoonright^i \mathsf{p} = \mu \mathbf{t}.(\mathbb{G}\upharpoonright^i \mathsf{p}) \qquad \mathbf{t}\upharpoonright^i \mathsf{p} = \mathbf{t} \qquad \mu \mathbf{t}.\mathbb{T} \sqcap^i \mu \mathbf{t}.\mathbb{T}' = \mu \mathbf{t}.(\mathbb{T} \sqcap^i \mathbb{T}') \qquad \mathbf{t} \sqcap^i \mathbf{t} = \mathbf{t} \tag{1}$$

(plus other rules akin to [PROJ-END], [PROJ-IN], [PROJ-OUT], [PROJ-CONT], [MRG-ID], and [MRG-BRA] in Definition 3.6 — here omitted).

The inductive projection/merging in (1) fails when it tries to combine session types with "mismatching" μ-subterms. This limitation is sometimes undesirable: e.g., the global type

$$\mathbb{G}_\mu = \mathsf{p} \to \mathsf{q} : \left\{ \begin{array}{l} \ell_1(\mathtt{nat}).\mu\mathbf{t}.\mathsf{r} \to \mathsf{p} : \ell'(\mathtt{bool}).\mathbf{t}\,, \\ \ell_2(\mathtt{bool}).\mathsf{r} \to \mathsf{p} : \ell'(\mathtt{bool}).\mu\mathbf{t}.\mathsf{r} \to \mathsf{p} : \ell'(\mathtt{bool}).\mathbf{t} \end{array} \right\}$$

is *not* projectable with (1) onto participant r. This is because we have to skip the first interaction between p and q, project all continuations onto r, and merge the resulting session types[2]; therefore, when using (1), we have:

$$\mathbb{G}_\mu \upharpoonright^i \mathsf{r} = \mu\mathbf{t}.\mathsf{p}!\ell'(\mathtt{bool}).\mathbf{t} \sqcap^i \mathsf{p}!\ell'(\mathtt{bool}).\mu\mathbf{t}.\mathsf{p}!\ell'(\mathtt{bool}).\mathbf{t}$$

which is undefined: by (1), syntactic types with a top-level recursion can only be merged with types having a top-level recursion.

Our projection/merging of type trees, instead, does not consider the position of the syntactic occurrences of $\mu\mathbf{t}.\ldots$ — and in this example, we have $\mathcal{T}(\mathbb{G}_\mu)\upharpoonright\mathsf{r} = \mathcal{T}(\mu\mathbf{t}.\mathsf{p}!\ell'(\mathtt{bool}).\mathbf{t})$.

In some other cases, the limitation imposed by ((1)) is desirable, as it syntactically enforces a balancing requirement similar to ours (Definition 3.3). E.g., if we use inductive projection/merging on the examples in Example 3.12, we obtain the same results — *except* for the non-balanced type $\mathbb{G}''$, whose inductive projection onto r is:

$$\mathbb{G}'' \upharpoonright^i \mathsf{r} = \mu\mathbf{t}.\big((\mathbf{t} \upharpoonright^i \mathsf{r}) \sqcap^i (\mathsf{q} \to \mathsf{r} : \ell''(\mathtt{int}) \upharpoonright^i \mathsf{r})\big) = \mu\mathbf{t}.\left(\mathbf{t} \sqcap^i \mathsf{q}?\ell''(\mathtt{int})\right) \textbf{ (undefined)}$$

This shows that, albeit other session types works do not have explicit balancing requirements on global types (Definition 3.3), they actually enforce similar constraints via inductive projection/merging.

3.2. Subtyping

The subtyping relation $\leqslant$ is used to augment the flexibility of the type system (introduced in Section 3.3): by determining when a type $\mathbb{T}$ is "smaller" than $\mathbb{T}'$, it allows to use a process typed by the former whenever a process typed by the latter is required.

Definition 3.15 *(Subsorting and subtyping). Subsorting* $\leq:$ is the least reflexive binary relation such that $\mathtt{nat} \leq: \mathtt{int}$.

Subtyping $\leqslant$ is the largest relation between session type trees coinductively defined by the following rules:

$$\text{[SUB-END]}\quad \mathtt{end} \leqslant \mathtt{end} \qquad \text{[SUB-IN]}\quad \frac{\forall i \in I : \quad S_i' \leq: S_i \quad \mathsf{T}_i \leqslant \mathsf{T}_i'}{\bigwedge_{i\in I\cup J} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i \leqslant \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i').\mathsf{T}_i'} \qquad \text{[SUB-OUT]}\quad \frac{\forall i \in I : \quad S_i \leq: S_i' \quad \mathsf{T}_i \leqslant \mathsf{T}_i'}{\bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathsf{T}_i \leqslant \bigvee_{i\in I\cup J} \mathsf{p}!\ell_i(S_i').\mathsf{T}_i'}$$

We lift $\leqslant$ to syntactic types: i.e., $\mathbb{T} \leqslant \mathbb{T}'$ holds iff $\mathcal{T}(\mathbb{T}) \leqslant \mathcal{T}(\mathbb{T}')$.

Intuitively, the session subtyping $\leqslant$ in Definition 3.15 says that T is smaller than T' when T is "less liberal" than T' — i.e., when T allows for less internal choices, and demands to handle more external choices.[3] A peculiarity of the relation is that, apart from a pair of inactive session types, only inputs and outputs from/to a same participant can be related (with additional conditions to be satisfied). Note that the double line in the subtyping rules indicates that the rules are interpreted *coinductively* [53, Chapter 21].

- Rule [SUB-END] says that $\mathtt{end}$ is only subtype of itself.
- Rule [SUB-IN] relates external choices from the same participant p: the subtype must support all the choices of the supertype, and for each common message label, the continuations must be related, too; note that the carried sorts are contravariant: e.g., if the supertype requires to receive a message $\ell_i(\mathtt{nat})$ (for some $i \in I$), then the subtype can support $\ell_i(\mathtt{int})$ or $\ell_i(\mathtt{nat})$, since $\mathtt{nat} \leq: \mathtt{int}$ and $\mathtt{nat} \leq: \mathtt{nat}$.
- Rule [SUB-OUT] relates internal choices towards the same participant p: the subtype must offer a subset of the choices of the supertype, and for each common message label, the continuations must be related, too; note that the carried sorts are covariant: e.g., if the supertype allows to send a message $\ell_i(\mathtt{int})$ (for some $i \in I$), then the subtype can allow to send $\ell_i(\mathtt{int})$ or $\ell_i(\mathtt{nat})$, since $\mathtt{int} \leq: \mathtt{int}$ and $\mathtt{nat} \leq: \mathtt{int}$.

Lemma 3.16. *The subtyping relation* $\leqslant$ *is reflexive and transitive.*

Proof. See Appendix A.6. □

[2] This "skip, project, and merge" step corresponds to rule [PROJ-CONT] in Definition 3.6; similar approaches are found in most projection/merging operators in literature.

[3] Readers familiar with the theory of session types might notice that our subtyping relation is inverted w.r.t. the original binary session subtyping, introduced in the works of Gay and Hole [27], Gay and Hole [25]. In such works, smaller types have less internal choices, and more external choices: this is because they formalise a "channel-oriented" notion of subtyping, while we adopt a "process-oriented" view. For a thorough analysis and comparison of the two approaches, see [26].

Table 4
Typing rules for expressions.

$$\Gamma \vdash \mathsf{n} : \mathtt{nat} \qquad \Gamma \vdash \mathsf{i} : \mathtt{int} \qquad \Gamma \vdash \mathsf{true} : \mathtt{bool} \qquad \Gamma \vdash \mathsf{false} : \mathtt{bool} \qquad \Gamma, x : S \vdash x : S$$

$$\frac{\Gamma \vdash \mathsf{e} : \mathtt{nat}}{\Gamma \vdash \mathtt{succ}(\mathsf{e}) : \mathtt{nat}} \qquad \frac{\Gamma \vdash \mathsf{e} : \mathtt{int}}{\Gamma \vdash \mathtt{neg}(\mathsf{e}) : \mathtt{int}} \qquad \frac{\Gamma \vdash \mathsf{e} : \mathtt{bool}}{\Gamma \vdash \neg \mathsf{e} : \mathtt{bool}}$$

$$\frac{\Gamma \vdash \mathsf{e}_1 : S \quad \Gamma \vdash \mathsf{e}_2 : S}{\Gamma \vdash \mathsf{e}_1 \oplus \mathsf{e}_2 : S} \qquad \frac{\Gamma \vdash \mathsf{e}_1 : \mathtt{int} \quad \Gamma \vdash \mathsf{e}_2 : \mathtt{int}}{\Gamma \vdash \mathsf{e}_1 > \mathsf{e}_2 : \mathtt{bool}} \qquad \frac{\Gamma \vdash \mathsf{e} : S \quad S \leq : S'}{\Gamma \vdash \mathsf{e} : S'}$$

Table 5
Typing rules for processes and sessions.

$$[\text{T-SUB}]\ \frac{\Gamma \vdash P : \mathsf{T} \qquad \mathsf{T} \leqslant \mathsf{T}'}{\Gamma \vdash P : \mathsf{T}'} \qquad [\text{T-}\mathbf{0}]\ \Gamma \vdash \mathbf{0} : \mathtt{end} \qquad [\text{T-REC}]\ \frac{\Gamma, X : \mathsf{T} \vdash P : \mathsf{T}}{\Gamma \vdash \mu X.P : \mathsf{T}} \qquad [\text{T-VAR}]\ \Gamma, X : \mathsf{T} \vdash X : \mathsf{T}$$

$$[\text{T-INPUT-CHOICE}]\ \frac{\forall i \in I \quad \Gamma, x_i : S_i \vdash P_i : \mathsf{T}_i}{\Gamma \vdash \sum_{i \in I} \mathsf{q}?\ell_i(x_i).P_i : \bigwedge_{i \in I} \mathsf{q}?\ell_i(S_i).\mathsf{T}_i} \qquad [\text{T-OUT}]\ \frac{\Gamma \vdash \mathsf{e} : S \quad \Gamma \vdash P : \mathsf{T}}{\Gamma \vdash \mathsf{q}!\ell(\mathsf{e}).P : \mathsf{q}!\ell(S).\mathsf{T}}$$

$$[\text{T-CHOICE}]\ \frac{\Gamma \vdash \mathsf{e} : \mathtt{bool} \quad \Gamma \vdash P_1 : \mathsf{T} \quad \Gamma \vdash P_2 : \mathsf{T}}{\Gamma \vdash \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 : \mathsf{T}}$$

$$[\text{T-SESS}]\ \frac{\forall i \in I \quad \vdash P_i : \mathsf{G} \upharpoonright \mathsf{p}_i \quad \mathtt{pt}\{\mathsf{G}\} \subseteq \{\mathsf{p}_i \mid i \in I\}}{\vdash \prod_{i \in I} \mathsf{p}_i \triangleleft P_i : \mathsf{G}}$$

3.3. Type system

We now introduce a type system for the multiparty session calculus presented in Section 2. We distinguish three kinds of typing judgements:

$$\Gamma \vdash \mathsf{e} : S \qquad \Gamma \vdash P : \mathsf{T} \qquad \vdash \mathcal{M} : \mathsf{G}$$

where Γ is the *typing environment*:

$$\Gamma ::= \emptyset \mid \Gamma, x : S \mid \Gamma, X : \mathsf{T}$$

i.e., a mapping that associates expression variables with sorts, and process variables with session type trees.

We say that *a multiparty session* $\mathcal{M}$ *is well typed* if there is a global type $\mathbb{G}$ such that $\vdash \mathcal{M} : \mathcal{T}(\mathbb{G})$. If a multiparty session is well typed, we will sometimes write just $\vdash \mathcal{M}$.

The typing rules for expressions are given in Table 4, and are self-explanatory. The typing rules for processes and multiparty sessions are content of Table 5:

- [T-SUB] is the *subsumption rule*: a process with type T is also typed by the supertype T';
- [T-$\mathbf{0}$] says that a terminated process implements the terminated session type;
- [T-REC] types a recursive process $\mu X.P$ with T if P can be typed as T, too, by extending the typing environment with the assumption that X has type T;
- [T-VAR] uses the typing environment assumption that process X has type T;
- [T-INPUT-CHOICE] types a summation of input prefixes as a branching type. It requires that each input prefix targets the same participant q, and that, for all $i \in I$, each continuation process P_i is typed by the continuation type T_i, having the bound variable x_i in the typing environment with sort S_i. Note that the rule implicitly requires the process labels ℓ_i to be pairwise distinct (as per Definition 3.4);
- [T-OUT] types an output prefix with a singleton selection type, provided that the expression in the message payload has the correct sort S, and the process continuation matches the type continuation;
- [T-CHOICE] types a conditional process with T if its sub-processes can be typed by T and expression e is boolean.
- [T-SESS] types multiparty sessions, by associating typed processes to participants. It requires that the processes being composed in parallel can play as participants of a global communication protocol: hence, their types must be projections of a single global type G. The condition $\mathtt{pt}\{\mathsf{G}\} \subseteq \{\mathsf{p}_i \mid i \in I\}$ allows to also type sessions containing $\mathsf{p} \triangleleft \mathbf{0}$: this is needed to assure invariance of typing under structural congruence (Lemma A.22).

Example 3.17. We show that the multiparty session $\mathcal{M}$ from Example 2.2 is well typed. Consider the following global type tree:

$$\mathsf{G} = \mathtt{Alice} \to \mathtt{Bob} : \{\ell_1(\mathsf{nat}).\mathtt{Bob} \to \mathtt{Carol} : \ell_2(\mathsf{nat}).\mathtt{Carol} \to \mathtt{Alice} : \ell_3(\mathsf{nat}).\mathtt{end},\ \ell_4(\mathsf{nat}).\mathtt{Bob} \to \mathtt{Carol} : \ell_2(\mathsf{nat}).\mathtt{Carol} \to \mathtt{Alice} : \ell_3(\mathsf{nat}).\mathtt{end}\}.$$

We show that participants Alice, Bob and Carol respect the prescribed protocol G, by showing that they participate in a well-typed multiparty session. Applying rules from Table 5, we derive

$$\vdash P_{\mathtt{Alice}} : \mathsf{T}_{\mathtt{Alice}} \qquad \vdash P_{\mathtt{Bob}} : \mathsf{T}_{\mathtt{Bob}} \qquad \vdash P_{\mathtt{Carol}} : \mathsf{T}_{\mathtt{Carol}}$$

where:

$$\begin{array}{l} \mathsf{T}_{\mathtt{Alice}} = \mathtt{Bob}!\ell_1(\mathtt{nat}).\mathtt{Carol}?\ell_3(\mathtt{nat}).\mathtt{end} \\ \mathsf{T}_{\mathtt{Bob}} \quad = \mathtt{Alice}?\ell_1(\mathtt{nat}).\mathtt{Carol}!\ell_2(\mathtt{nat}).\mathtt{end} \wedge \mathtt{Alice}?\ell_4(\mathtt{nat}).\mathtt{Carol}!\ell_2(\mathtt{nat}).\mathtt{end} \\ \mathsf{T}_{\mathtt{Carol}} = \mathtt{Bob}?\ell_2(\mathtt{nat}).\mathtt{Alice}!\ell_3(\mathtt{nat}).\mathtt{end} \end{array}$$

Now, let:

$$\mathsf{T}'_{\mathtt{Alice}} = \mathtt{Bob}!\ell_1(\mathtt{nat}).\mathtt{Carol}?\ell_3(\mathtt{nat}).\mathtt{end} \vee \mathtt{Bob}!\ell_4(\mathtt{nat}).\mathtt{Carol}?\ell_3(\mathtt{nat}).\mathtt{end}$$

Since it holds that $\mathsf{T}_{\mathtt{Alice}} \leqslant \mathsf{T}'_{\mathtt{Alice}}$, and the projections of G to the participants are

$$\mathsf{G}\upharpoonright\mathtt{Alice} = \mathsf{T}'_{\mathtt{Alice}} \qquad \mathsf{G}\upharpoonright\mathtt{Bob} = \mathsf{T}_{\mathtt{Bob}} \qquad \mathsf{G}\upharpoonright\mathtt{Carol} = \mathsf{T}_{\mathtt{Carol}}$$

we conclude:

$$\vdash \mathtt{Alice} \triangleleft P_{\mathtt{Alice}} \mid \mathtt{Bob} \triangleleft P_{\mathtt{Bob}} \mid \mathtt{Carol} \triangleleft P_{\mathtt{Carol}} : \mathsf{G}.$$

Note that this session respects other protocols as well, including the one in Section 1, thanks to subtyping.

The proposed type system for multiparty sessions enjoys two fundamental properties: typed sessions only reduce to typed sessions (subject reduction), and typed sessions never get stuck. The remaining of this section is devoted to the proof of these properties.

In order to state subject reduction, we need to formalise how global types are modified when multiparty sessions reduce and evolve.

Definition 3.18 *(Global types consumption and reduction).* The *consumption* of the communication $\mathsf{p} \xrightarrow{\ell} \mathsf{q}$ for the global type tree G (notation $\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}$) is the global type tree coinductively defined as follows:

$$\begin{array}{lcll} \big(\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}\big) \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q} & = & \mathsf{G}_k & \text{if } \exists k \in I : \ell = \ell_k \\ \big(\mathsf{r} \to \mathsf{s} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}\big) \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q} & = & \mathsf{r} \to \mathsf{s} : \{\ell_i(S_i).\mathsf{G}_i \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}\}_{i\in I} & \text{if } \begin{cases} \{\mathsf{r},\mathsf{s}\} \cap \{\mathsf{p},\mathsf{q}\} = \emptyset \text{ and} \\ \forall i \in I : \{\mathsf{p},\mathsf{q}\} \subseteq \mathsf{G}_i \end{cases} \end{array}$$

The *reduction of global type trees* is the smallest pre-order relation closed under the rule: $\mathsf{G} \Longrightarrow \mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}$ We lift $\Longrightarrow$ to syntactic types: i.e., $\mathbb{G} \Longrightarrow \mathbb{G}'$ holds iff $\mathcal{T}(\mathbb{G}) \Longrightarrow \mathcal{T}(\mathbb{G}')$.

Notice that $\mathtt{end} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}$ is undefined. Moreover, $\mathcal{T}(\mu\mathbf{t}.\mathsf{r} \to \mathsf{s} : \ell'(S).\mathbf{t}) \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}$ is undefined when $\mathsf{r} \neq \mathsf{p}$, or $\mathsf{s} \neq \mathsf{q}$, or $\ell' \neq \ell$, because in such cases, both side conditions of the definition are violated; in particular, when $\mathsf{r} \neq \mathsf{p}$ and/or $\mathsf{s} \neq \mathsf{q}$, we cannot apply neither the first case of the definition, nor the second, because $\{\mathsf{p},\mathsf{q}\} \not\subseteq \mathsf{G}$. Clearly, the consumption of a balanced global type (Definition 3.3) yields a balanced global type. Moreover, if G is projectable and $\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}$ is defined, then the global type $\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}$ is projectable, too; this is formalised in Lemma 3.19 below, which provides properties of consumption that are essential in the proof of subject reduction.

Lemma 3.19. *If* $\mathsf{q}!\ell(S).\mathsf{T} \leq \mathsf{G}\upharpoonright\mathsf{p}$ *and* $\mathsf{p}?\ell(S').\mathsf{T}' \leq \mathsf{G}\upharpoonright\mathsf{q}$*, then:*

(1) $\mathsf{T} \leqslant (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{p}$
(2) $\mathsf{T}' \leqslant (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{q}$*; and*
(3) $\mathsf{G}\upharpoonright\mathsf{r} \leqslant (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{r}$ *for* $\mathsf{r} \neq \mathsf{p}, \mathsf{r} \neq \mathsf{q}$.

Proof. See Appendix A.7. □

Intuitively, Lemma 3.19 says that global type consumption makes projections "more precise:" e.g., by item (3), $\mathsf{G}\upharpoonright\mathsf{r}$ (before the consumption) might require to support more external choices than $(\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{r}$, as per Definition 3.15: this is illustrated in Example 3.20 below. The intuition behind items (1) and (2) is similar, except that we remove the consumed input/output prefix from the projected types.

Example 3.20. We show that a projection of a global type before the consumption might require to support more external choices than the projection after the consumption. Take $\mathbb{G}$, its subterm $\mathbb{G}_1$, from Example 3.12, and their type trees denoted as G and G_1, respectively. Also take the projection:

$$\mathsf{G}\upharpoonright \mathsf{r} = \mathsf{q}?\ell_3(\mathtt{int}) \wedge \mathsf{q}?\ell_4(\mathtt{int}) \wedge \mathsf{q}?\ell_5(\mathtt{nat})$$

and recall the explanation on how $\mathsf{G}\upharpoonright \mathsf{r}$ above merges all the possible inputs that r might receive from q, depending on whether p first sends ℓ_1 or ℓ_2 to q. We have:

$$\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell_1} \mathsf{q} = \quad \mathsf{G}_1 \quad = \mathsf{q} \to \mathsf{r} : \{\ell_3(\mathtt{int}), \ell_5(\mathtt{nat})\}$$

$$(\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell_1} \mathsf{q})\upharpoonright \mathsf{r} = \mathsf{G}_1\upharpoonright \mathsf{r} = \mathsf{q}?\ell_3(\mathtt{int}) \wedge \mathsf{q}?\ell_5(\mathtt{nat})$$

and we obtain $\mathsf{G}\upharpoonright \mathsf{r} \leqslant (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell_1} \mathsf{q})\upharpoonright \mathsf{r}$, as per Lemma 3.19(3). The reason is that, after the transition from G to G_1, there is no possibility for q to send ℓ_4 to r, hence r does not need to support such a message in its projection.

Note that a process that plays the role of r in G, and is therefore typed by $\mathsf{G}\upharpoonright \mathsf{r}$, has to support the input of ℓ_4 from q, by rule [T-INPUT-CHOICE] in Table 5. After the transition from G to G_1, the same process is also typed by $\mathsf{G}_1\upharpoonright \mathsf{r}$, by rule [T-SUB] — but will never receive a message ℓ_4 from q.

We can now prove subject reduction.

Theorem 3.21 *(Subject Reduction). Let* $\vdash \mathcal{M} : \mathsf{G}$. *For all* $\mathcal{M}'$, *if* $\mathcal{M} \longrightarrow \mathcal{M}'$, *then* $\vdash \mathcal{M}' : \mathsf{G}'$ *for some* G' *such that* $\mathsf{G} \Longrightarrow \mathsf{G}'$.

Proof. See Appendix A.7 □

Corollary 3.22. *Let* $\vdash \mathcal{M} : \mathsf{G}$. *If* $\mathcal{M} \longrightarrow^* \mathcal{M}'$, *then* $\vdash \mathcal{M}' : \mathsf{G}'$ *for some* G' *such that* $\mathsf{G} \Longrightarrow \mathsf{G}'$.

Proof. See Appendix A.7 □

Theorem 3.23 *(Progress). If* $\vdash \mathcal{M} : \mathsf{G}$, *then either* $\mathcal{M} \equiv \mathsf{p} \triangleleft \mathbf{0}$ *or there is* $\mathcal{M}'$ *such that* $\mathcal{M} \longrightarrow \mathcal{M}'$.

Proof. See Appendix A.7 □

As a consequence of subject reduction and progress, we get the safety property stating that a typed multiparty session will never get stuck.

Theorem 3.24 *(Type Safety). If* $\vdash \mathcal{M} : \mathsf{G}$, *then it does not hold* $\mathtt{stuck}(\mathcal{M})$.

Proof. Direct consequence of Corollary 3.22, Theorem 3.23, and Definition 2.3. □

3.4. Algorithmic multiparty session subtyping

We now introduce a characterisation of the subtyping relation by means of an algorithm: *(1)* we define a procedure to induce a relation $\leqslant_a$ between syntactic types; *(2)* we prove that $\leqslant_a$ is equivalent to $\leqslant$ in Definition 3.15 (Theorem 3.26), and *(3)* we prove that the procedure is always terminating (Lemma 3.25).

Although decidability of subtyping is *not* necessary to prove our main result (i.e., the preciseness $\leqslant$); the algorithmic definition of $\leqslant_a$ provides the basis to formalise *failing derivations* (Definition 3.29 below), characterising when $\leqslant_a$ (and thus, $\leqslant$) does *not* hold: we will use this technical device in Section 4.1, to prove that a relation $\not\trianglelefteq$ is the complement of $\leqslant$.

We adopt the approach of Gay and Hole [25] and Pierce and Sangiorgi [54]. In Table 6, we inductively define an *algorithmic subtyping judgement* for *syntactic* types, of the form:

$$\Theta \vdash \mathbb{T} \leqslant_a \mathbb{T}' \tag{2}$$

where $\mathbb{T} \leqslant_a \mathbb{T}'$ is the *goal* of the algorithm, and the set Θ is the set of *assumed instances*: it contains pairs of session types $(\mathbb{T}, \mathbb{T}')$, assumed to be related by the subtyping relation. We obtain a verification procedure by reading the rules upwards: i.e., to verify the goal in the conclusion of each rule, we must verify its premises, possibly introducing more sub-goals, and adding elements to Θ. Given an input $\Theta \vdash \mathbb{T} \leqslant_a \mathbb{T}'$, the procedure returns true when it successfully verifies the given goal $\mathbb{T} \leqslant_a \mathbb{T}'$ in Θ; otherwise, if the procedure reaches a sub-goal that does not match any rule in Table 6, it returns false. We postulate that the rule [ALG-ASSUMP] always has priority (for the sake of termination); we also postulate that [ALG-REC-L] has priority over [ALG-REC-R] (for the sake of determinism). The procedure is an algorithm (Lemma 3.25 below): it terminates on all inputs, since Θ cannot grow indefinitely, as there is a finite maximum number of assumed instances that need to be considered.

Table 6
Algorithmic subtyping rules. Note that in the premise of rule [ALG-ASSUMP], the types $\mathbb{T}, \mathbb{T}'$ are compared with those in Θ using *syntactic* type equality $\equiv$ (Notation 3.5), thus distinguishing a recursive type from its unfoldings.

$$\frac{(\mathbb{T}, \mathbb{T}') \in \Theta}{\Theta \vdash \mathbb{T} \leqslant_a \mathbb{T}'} \ [\text{ALG-ASSUMP}] \qquad \Theta \vdash \mathtt{end} \leqslant_a \mathtt{end} \ [\text{ALG-END}]$$

$$\frac{\mathbb{T} \equiv \bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathbb{T}_i \quad \mathbb{T}' \equiv \bigvee_{i\in I\cup J} \mathsf{p}!\ell_i(S'_i).\mathbb{T}'_i \quad \forall i \in I : \Theta \vdash \mathbb{T}_i \leqslant_a \mathbb{T}'_i \quad \forall i \in I : S_i \leq: S'_i}{\Theta \vdash \mathbb{T} \leqslant_a \mathbb{T}'} \ [\text{ALG-SUB-OUT}]$$

$$\frac{\mathbb{T} \equiv \bigwedge_{i\in I\cup J} \mathsf{p}?\ell_i(S_i).\mathbb{T}_i \quad \mathbb{T}' \equiv \bigwedge_{i\in I} \mathsf{p}?\ell_i(S'_i).\mathbb{T}'_i \quad \forall i \in I : \Theta \vdash \mathbb{T}_i \leqslant_a \mathbb{T}'_i \quad \forall i \in I : S'_i \leq: S_i}{\Theta \vdash \mathbb{T} \leqslant_a \mathbb{T}'} \ [\text{ALG-SUB-IN}]$$

$$\frac{\Theta \cup \{(\mu\mathbf{t}.\mathbb{T}_1, \mathbb{T}_2)\} \vdash \mathbb{T}_1\{\mu\mathbf{t}.\mathbb{T}_1/\mathbf{t}\} \leqslant_a \mathbb{T}_2}{\Theta \vdash \mu\mathbf{t}.\mathbb{T}_1 \leqslant_a \mathbb{T}_2} \ [\text{ALG-REC-L}] \qquad \frac{\Theta \cup \{(\mathbb{T}_1, \mu\mathbf{t}.\mathbb{T}_2)\} \vdash \mathbb{T}_1 \leqslant_a \mathbb{T}_2\{\mu\mathbf{t}.\mathbb{T}_2/\mathbf{t}\}}{\Theta \vdash \mathbb{T}_1 \leqslant_a \mu\mathbf{t}.\mathbb{T}_2} \ [\text{ALG-REC-R}]$$

Lemma 3.25. *The subtyping procedure in Table 6 is terminating.*

Proof. See Appendix A.9. □

The following theorem shows that the definition of algorithmic subtyping and the definition of subtyping in Definition 3.15 are equivalent.

Theorem 3.26 *(Subtyping for multiparty session types is decidable). Let $\mathbb{T}$ and $\mathbb{T}'$ be session types. Then, $\emptyset \vdash \mathbb{T} \leqslant_a \mathbb{T}'$ if and only if $\mathcal{T}(\mathbb{T}) \leqslant \mathcal{T}(\mathbb{T}')$.*

Proof. The proof is similar to those from Pierce and Sangiorgi [54] and [25]. □

Rule function and failing derivations. We now introduce the notions of *rule function* and *failing derivation* for algorithmic subtyping, following an approach inspired by Ligatti et al. [42,43]. These technical tools will be useful later on, to prove the results in Section 4.1.

When the algorithmic subtyping procedure above returns `true` for an input $\Theta \vdash \mathbb{T} \leqslant_a \mathbb{T}'$, it generates a *derivation* for $\Theta \vdash \mathbb{T} \leqslant_a \mathbb{T}'$ under the rules in Table 6, such that:

- each node of the derivation is a judgement $\Theta_i \vdash \mathbb{T}_i \leqslant_a \mathbb{T}'_i$;
- the root of the derivation is the input $\Theta \vdash \mathbb{T} \leqslant_a \mathbb{T}'$;
- the leaves of the derivation have only two possible forms:
  a $\Theta_j \vdash \mathbb{T}_j \leqslant_a \mathbb{T}'_j$ with $(\mathbb{T}_j, \mathbb{T}'_j) \in \Theta_j$ (from the axiom [ALG-ASSUMP]), or
  b $\Theta_j \vdash \mathtt{end} \leqslant_a \mathtt{end}$ (from the axiom [ALG-END]).

With this insight, we can define a *rule function* (Definition 3.27 below), to associate a judgement $\Theta \vdash \mathbb{T} \leqslant_a \mathbb{T}'$ with the set of its premises — i.e., a set $\mathcal{J}$ such that $\Theta \vdash \mathbb{T} \leqslant_a \mathbb{T}'$ can be derived (under the rules in Table 6) if and only if *(1)* the set $\mathcal{J}$ is non-empty, and *(2)* each element of $\mathcal{J}$ is either true, or a derivable judgement of the form in (2).

Definition 3.27. The *rule function* Φ maps judgements to sets containing judgements or true, as follows:

$$\Phi(\Theta \vdash \mathbb{T} \leqslant_a \mathbb{T}') = \begin{cases} \bigcup_{i\in I} \{\Theta \vdash \mathbb{T}_i \leqslant_a \mathbb{T}'_i\} & \text{if } \begin{cases} \mathbb{T} \equiv \bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathbb{T}_i \\ \text{and } \mathbb{T}' \equiv \bigvee_{i\in I\cup J} \mathsf{p}!\ell_i(S'_i).\mathbb{T}'_i \\ \text{and } \forall i \in I : S_i \leq: S'_i \text{ and } (\mathbb{T}, \mathbb{T}') \notin \Theta \end{cases} \\ \bigcup_{i\in I} \{\Theta \vdash \mathbb{T}_i \leqslant_a \mathbb{T}'_i\} & \text{if } \begin{cases} \mathbb{T} \equiv \bigwedge_{i\in I\cup J} \mathsf{p}?\ell_i(S_i).\mathbb{T}_i \\ \text{and } \mathbb{T}' \equiv \bigwedge_{i\in I} \mathsf{p}?\ell_i(S'_i).\mathbb{T}'_i \\ \text{and } \forall i \in I : S'_i \leq: S_i \text{ and } (\mathbb{T}, \mathbb{T}') \notin \Theta \end{cases} \\ \{\Theta \cup \{(\mathbb{T}, \mathbb{T}')\} \vdash \mathbb{T}_1\{\mathbb{T}/\mathbf{t}\} \leqslant_a \mathbb{T}'\} & \text{if } \mathbb{T} \equiv \mu\mathbf{t}.\mathbb{T}_1 \text{ and } (\mathbb{T}, \mathbb{T}') \notin \Theta \\ \{\Theta \cup \{(\mathbb{T}, \mathbb{T}')\} \vdash \mathbb{T} \leqslant_a \mathbb{T}_2\{\mathbb{T}'/\mathbf{t}\}\} & \text{if } \mathbb{T}' \equiv \mu\mathbf{t}.\mathbb{T}_2 \text{ and } \forall \mathbb{T}'' : \mathbb{T} \not\equiv \mu\mathbf{t}.\mathbb{T}'' \text{ and } (\mathbb{T}, \mathbb{T}') \notin \Theta \\ \{\mathtt{true}\} & \text{if } (\mathbb{T}, \mathbb{T}') \equiv (\mathtt{end}, \mathtt{end}) \text{ or } (\mathbb{T}, \mathbb{T}') \in \Theta \\ \emptyset & \text{in all other cases} \end{cases}$$

Lemma 3.28. $\Theta \vdash \mathbb{T} \leqslant_a \mathbb{T}'$ *can be derived iff* $\Phi(\Theta \vdash \mathbb{T} \leqslant_a \mathbb{T}')$ *is non-empty, and only contains* true, *or derivable judgements.*

Proof. It follows directly from Table 6 and Definition 3.27. □

Dually, if the subtyping procedure applied to the input $\Theta \vdash \mathbb{T} \leqslant_a \mathbb{T}'$ returns false, then the algorithm constructs a sequence of judgements of the form in (2) that starts from $\Theta \vdash \mathbb{T} \leqslant_a \mathbb{T}'$, takes 0 or more steps following the rule function, and finally reaches some judgement J that does not match any rule in Table 6 (i.e., the rule function applied to J returns $\emptyset$). We call such a sequence a *failing derivation*, as formalised in Definition 3.29 below.

Definition 3.29. Consider a finite sequence $(J_1, \ldots, J_n)$, where $n \geq 1$, and each J_i $(1 \leq i \leq n)$ is a judgement of the form (2). We say that the sequence is a *failing derivation of* $\Theta \vdash \mathbb{T} \leqslant_a \mathbb{T}'$ iff it satisfies the following requirements:

1. J_1 is $\Theta \vdash \mathbb{T} \leqslant_a \mathbb{T}'$;
2. for $m \in \{2, \ldots, n\}$, J_m is $\Theta_m \vdash \mathbb{T}_m \leqslant_a \mathbb{T}'_m$, with $(\Theta_m \vdash \mathbb{T}_m \leqslant_a \mathbb{T}'_m) \in \Phi(\Theta_{m-1} \vdash \mathbb{T}_{m-1} \leqslant_a \mathbb{T}'_{m-1})$;
3. J_n is $\Theta_n \vdash \mathbb{T}_n \leqslant_a \mathbb{T}'_n$, with $\Phi(\Theta_n \vdash \mathbb{T}_n \leqslant_a \mathbb{T}'_n) = \emptyset$.

Note that by Definition 3.29, when $(J_1, \ldots, J_n)$ is a failing derivation of $\Theta \vdash \mathbb{T} \leqslant_a \mathbb{T}'$, then in each $J_i = \Theta_i \vdash \mathbb{T}_i \leqslant_a \mathbb{T}'_i$ $(1 \leq i \leq n)$, the set of assumed instances Θ_i never contains $(\mathbb{T}_i, \mathbb{T}'_i)$: otherwise, by Definition 3.27, we would have $\Phi(J_i) = \{\text{true}\}$, and thus, either clause 2 or 3 of Definition 3.29 would not hold. Moreover, if $n = 1$, then J_1 satisfies item 1 of Definition 3.29 and has one of the forms listed in Corollary 3.30 below — and this implies, e.g., $\Theta \not\vdash (\mathbb{T}, \mathbb{T}') \not\equiv (\mathtt{end}, \mathtt{end})$.

Corollary 3.30. *Assume that the sequence of judgements* $(J_1, \ldots, J_n)$ *is a failing derivation of* $\Theta \vdash \mathbb{T} \leqslant_a \mathbb{T}'$. *Then,* J_n *is a judgement* $\Theta_n \vdash \mathbb{T}_n \leqslant_a \mathbb{T}'_n$ *such that one of the following holds:*

(a) $(\mathbb{T}_n, \mathbb{T}'_n) \equiv (\mathtt{end}, \mathbb{T}')$ *and* $\mathbb{T}' \not\equiv \mathtt{end}$;
(b) $(\mathbb{T}_n, \mathbb{T}'_n) \equiv (\mathbb{T}, \mathtt{end})$ *and* $\mathbb{T} \not\equiv \mathtt{end}$;
(c) $(\mathbb{T}_n, \mathbb{T}'_n) \equiv (\bigwedge_{i\in I} \mathtt{p}?\ell_i(S_i).\mathbb{T}_i, \bigwedge_{j\in J} \mathtt{q}?\ell'_j(S'_j).\mathbb{T}'_j)$ *and* $\mathtt{p} \neq \mathtt{q}$;
(d) $(\mathbb{T}_n, \mathbb{T}'_n) \equiv (\bigvee_{i\in I} \mathtt{p}!\ell_i(S_i).\mathbb{T}_i, \bigvee_{j\in J} \mathtt{q}!\ell'_j(S'_j).\mathbb{T}'_j)$ *and* $\mathtt{p} \neq \mathtt{q}$;
(e) $(\mathbb{T}_n, \mathbb{T}'_n) \equiv (\bigwedge_{i\in I} \mathtt{p}?\ell_i(S_i).\mathbb{T}_i, \bigvee_{j\in J} \mathtt{q}!\ell'_j(S'_j).\mathbb{T}'_j)$;
(f) $(\mathbb{T}_n, \mathbb{T}'_n) \equiv (\bigvee_{i\in I} \mathtt{p}!\ell_i(S_i).\mathbb{T}_i, \bigwedge_{j\in J} \mathtt{q}?\ell'_j(S'_j).\mathbb{T}'_j)$;
(g) $(\mathbb{T}_n, \mathbb{T}'_n) \equiv (\bigwedge_{i\in I} \mathtt{p}?\ell_i(S_i).\mathbb{T}_i, \bigwedge_{j\in J} \mathtt{p}?\ell'_j(S'_j).\mathbb{T}'_j)$ *and* $\exists j \in J$ *such that* $\ell_i \neq \ell'_j$ *for every* $i \in I$;
(h) $(\mathbb{T}_n, \mathbb{T}'_n) \equiv (\bigwedge_{i\in I} \mathtt{p}!\ell_i(S_i).\mathbb{T}_i, \bigwedge_{j\in J} \mathtt{p}!\ell'_j(S'_j).\mathbb{T}'_j)$ *and* $\exists i \in I$ *such that* $\ell_i \neq \ell'_j$ *for every* $j \in J$;
(i) $(\mathbb{T}_n, \mathbb{T}'_n) \equiv (\bigwedge_{i\in I} \mathtt{p}?\ell_i(S_i).\mathbb{T}_i, \bigwedge_{j\in J} \mathtt{p}?\ell'_j(S'_j).\mathbb{T}'_j)$ *and* $\exists i \in I,\ j \in J$ *such that* $\ell_i = \ell'_j$ *and* $S'_j \not\leqslant: S_i$;
(j) $(\mathbb{T}_n, \mathbb{T}'_n) \equiv (\bigwedge_{i\in I} \mathtt{p}!\ell_i(S_i).\mathbb{T}_i, \bigwedge_{j\in J} \mathtt{p}!\ell'_j(S'_j).\mathbb{T}'_j)$ *and* $\exists i \in I,\ j \in J$ *such that* $\ell_i = \ell'_j$ *and* $S_i \not\leqslant: S'_j$.

We can now characterise the judgements that do *not* hold under the rules in Table 6.

Lemma 3.31. *The judgement* $\emptyset \vdash \mathbb{T} \leqslant_a \mathbb{T}'$ *is not derivable iff there is a failing derivation of* $\emptyset \vdash \mathbb{T} \leqslant_a \mathbb{T}'$.

We will use Lemma 3.31 in Section 4.1, to prove that a relation $\not\trianglelefteq$ is the negation of subtyping.

4. Operational preciseness

In this section, we provide an *operational* characterisation of preciseness of the synchronous multiparty subtyping relation $\leqslant$ (Definition 3.15), by adapting the approach of Chen et al. [14,15], that was developed for *binary* sessions.

From the operational perspective, a subtyping relation $\mathcal{R}$ is sound if it satisfies the substitutability principle: i.e., if $\mathsf{T}\ \mathcal{R}\ \mathsf{T}'$, then a process of type T' engaged in a well-typed session may be safely replaced with a process of type T. If $\mathcal{R}$ is also the largest relation with that property, then we say that the subtyping $\mathcal{R}$ is operationally precise; in this case, the implication in the operational soundness statement is also true when reversed — and this reversed direction is called operational completeness.

For the purpose of introducing a formal notion of operational preciseness (Definition 4.1 below), we define multiparty session contexts as: $C ::= \_ \mid C \mid \mathcal{M}$

Definition 4.1 *(Operational preciseness).* The subtyping relation $\leqslant$ is

(1) *operationally sound* if $\mathsf{T} \leqslant \mathsf{T}'$ **implies** that for all r, C, P the following implication holds:

$$\text{if } (\forall Q)\ (\vdash Q : \mathsf{T}' \Rightarrow \vdash C[\mathsf{r} \triangleleft Q] : \mathsf{G} \text{ for some } \mathsf{G}) \text{ then } (\vdash P : \mathsf{T} \Rightarrow C[\mathsf{r} \triangleleft P] \text{ does not get stuck});$$

Table 7
Negation of subtyping.

$$[\text{NSUB-ENDR}]\ \frac{\mathsf{T}' \neq \mathtt{end}}{\mathtt{end} \not\leqslant \mathsf{T}'} \qquad [\text{NSUB-ENDL}]\ \frac{\mathsf{T} \neq \mathtt{end}}{\mathsf{T} \not\leqslant \mathtt{end}}$$

$$[\text{NSUB-DIFF-IN}]\ \frac{\mathsf{p} \neq \mathsf{q}}{\bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i \not\leqslant \bigwedge_{j\in J} \mathsf{q}?\ell'_j(S'_j).\mathsf{T}'_j} \qquad [\text{NSUB-DIFF-OUT}]\ \frac{\mathsf{p} \neq \mathsf{q}}{\bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathsf{T}_i \not\leqslant \bigvee_{j\in J} \mathsf{q}!\ell'_j(S'_j).\mathsf{T}'_j}$$

$$[\text{NSUB-IN-OUT}]\ \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i \not\leqslant \bigvee_{j\in J} \mathsf{q}!\ell'_j(S'_j).\mathsf{T}'_j \qquad [\text{NSUB-OUT-IN}]\ \bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathsf{T}_i \not\leqslant \bigwedge_{j\in J} \mathsf{q}?\ell'_j(S'_j).\mathsf{T}'_j$$

$$[\text{NSUB-LABEL-IN}]\ \frac{\exists j \in J\ \forall i \in I : \ell_i \neq \ell'_j}{\bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i \not\leqslant \bigwedge_{j\in J} \mathsf{p}?\ell'_j(S'_j).\mathsf{T}'_j} \qquad [\text{NSUB-LABEL-OUT}]\ \frac{\exists i \in I\ \forall j \in J : \ell_i \neq \ell'_j}{\bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathsf{T}_i \not\leqslant \bigvee_{j\in J} \mathsf{p}!\ell'_j(S'_j).\mathsf{T}'_j}$$

$$[\text{NSUB-EXCH-IN}]\ \frac{\exists i \in I\ \exists j \in J : \ell_i = \ell'_j \quad S'_j \not\leqslant: S_i}{\bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i \not\leqslant \bigwedge_{j\in J} \mathsf{p}?\ell'_j(S'_j).\mathsf{T}'_j}$$

$$[\text{NSUB-CONT-IN}]\ \frac{\exists i \in I\ \exists j \in J : \ell_i = \ell'_j \quad \mathsf{T}_i \not\leqslant \mathsf{T}'_j}{\bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i \not\leqslant \bigwedge_{j\in J} \mathsf{p}?\ell'_j(S'_j).\mathsf{T}'_j} \qquad [\text{NSUB-CONT-OUT}]\ \frac{\exists i \in I\ \exists j \in J : \ell_i = \ell'_j \quad \mathsf{T}_i \not\leqslant \mathsf{T}'_j}{\bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathsf{T}_i \not\leqslant \bigvee_{j\in J} \mathsf{p}!\ell'_j(S'_j).\mathsf{T}'_j}$$

$$[\text{NSUB-EXCH-OUT}]\ \frac{\exists i \in I\ \exists j \in J : \ell_i = \ell'_j \quad S_i \not\leqslant: S'_j}{\bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathsf{T}_i \not\leqslant \bigvee_{j\in J} \mathsf{p}!\ell'_j(S'_j).\mathsf{T}'_j}$$

(2) *operationally complete* if $\mathsf{T} \not\leqslant \mathsf{T}'$ **implies** that there are r, C, P such that

$$(\forall Q)(\vdash Q : \mathsf{T}' \Rightarrow \vdash C[\mathsf{r} \triangleleft Q] : \mathsf{G} \text{ for some } \mathsf{G}) \text{ and } \vdash P : \mathsf{T} \text{ and } \mathtt{stuck}(C[\mathsf{r} \triangleleft P])\,;$$

(3) *operationally precise* if it is operationally sound and operationally complete.

Operational soundness follows from the subsumption rule and the type safety property, which is formalised in the following theorem.

Theorem 4.2 *(Soundness). The synchronous multiparty session subtyping $\leqslant$ is operationally sound.*

Proof. Take T, T' such that $\mathsf{T} \leqslant \mathsf{T}'$, and r, C satisfying the following condition:

$$\forall Q : \quad \vdash Q : \mathsf{T}' \Rightarrow \vdash C[\mathsf{r} \triangleleft Q] : \mathsf{G} \text{ for some } \mathsf{G}. \tag{3}$$

If $\vdash P : \mathsf{T}$, we derive by [T-SUB] that $\vdash P : \mathsf{T}'$ holds. By (3), $\vdash C[\mathsf{r} \triangleleft P] : \mathsf{G}$, for some G. Hence, by Theorem 3.24, $C[\mathsf{r} \triangleleft P]$ does not get stuck, which completes the proof. □

Operational completeness is the opposite direction of operational soundness, which is formalised above in a contrapositive form. Henceforth we prove the completeness using classical reasoning, and leave open the question of a constructive proof of completeness. We will derive the proof of operational completeness in four steps, as outlined in Section 1:

[Step 1] **(Section 4.1)** We define a relation $\not\leqslant$ over session type trees and prove that for all T, T', $\mathsf{T} \not\leqslant \mathsf{T}'$ is derivable iff $\mathsf{T} \leqslant \mathsf{T}'$ is not derivable.

[Step 2] **(Section 4.2)** For each $\mathbb{T}$, we define a *characteristic process* $\mathcal{P}(\mathbb{T})$ such that $\vdash \mathcal{P}(\mathbb{T}) : \mathcal{T}(\mathbb{T})$.

[Step 3] **(Section 4.3)** For each type $\mathbb{T}'$ and participant $\mathsf{r} \notin \mathtt{pt}\{\mathbb{T}'\}$, we define a *characteristic global type* $\mathcal{G}(\mathbb{T}', \mathsf{r})$ and a *characteristic context* $C_{\mathsf{r},\mathbb{T}'}$ with the property that

$$(\forall Q) \vdash Q : \mathcal{T}\big(\mathbb{T}'\big) \Rightarrow \vdash C_{\mathsf{r},\mathbb{T}'}[\mathsf{r} \triangleleft Q] : \mathcal{T}\big(\mathcal{G}(\mathbb{T}', \mathsf{r})\big)\,.$$

[Step 4] **(Section 4.4)** We prove that for all $\mathbb{T}, \mathbb{T}'$, if $\mathcal{T}(\mathbb{T}) \not\leqslant \mathcal{T}\big(\mathbb{T}'\big)$ then $\mathtt{stuck}\big(C_{\mathsf{r},\mathbb{T}'}[\mathsf{r} \triangleleft \mathcal{P}(\mathbb{T})]\big)$. Thus, we achieve completeness by finding r, $Q = \mathcal{P}(\mathbb{T})$ and $C_{\mathsf{r},\mathbb{T}'}$ that satisfy item *(2)* of Definition 4.1.

4.1. Step 1: negation of subtyping

We introduce the relation $\not\leqslant$ that characterises when a type is not a subtype of another type. It relies on the negation of subsorting $\not\leqslant:$, which is the complementary relation of the subsorting $\leq:$, and is defined as:

$$\mathtt{int} \not\leqslant: \mathtt{nat} \qquad \mathtt{bool} \not\leqslant: \mathtt{nat} \qquad \mathtt{nat} \not\leqslant: \mathtt{bool} \qquad \mathtt{bool} \not\leqslant: \mathtt{int} \qquad \mathtt{int} \not\leqslant: \mathtt{bool}$$

Definition 4.3 *(Negation of subtyping).* The relation $\not\leqslant$ between session type trees is inductively defined by the rules in Table 7. We lift $\not\leqslant$ to syntactic types: i.e., $\mathbb{T} \not\leqslant \mathbb{T}'$ holds iff $\mathcal{T}(\mathbb{T}) \not\leqslant \mathcal{T}\big(\mathbb{T}'\big)$.

The rules in Table 7 relate two type trees when:

- one is $\mathtt{end}$, but the other is not $\mathtt{end}$ ([NSUB-ENDR], [NSUB-ENDL]);
- they target different participants ([NSUB-DIFF-IN], [NSUB-DIFF-OUT]);
- they perform different actions (branching vs. selection) ([NSUB-OUT-IN], [NSUB-IN-OUT]);
- they are both branching types (resp. selection types), targeting the same participant, and the labels of the LHS (resp. RHS) do not cover all the labels of the RHS (resp. LHS) ([NSUB-LABEL-IN], resp. [NSUB-LABEL-OUT]);
- they are both branching/selection types, targeting the same participant, with matching labels followed by mismatching sorts (resp. continuations) ([NSUB-EXCH-IN], [NSUB-EXCH-OUT], resp. [NSUB-CONT-IN], [NSUB-CONT-OUT]).

We now give some examples of types related by $\not\leqslant$.

Example 4.4. We show that $\mathbb{T}_1 \not\leqslant \mathbb{T}_2$ holds for the following pairs of types:

1. $\mathbb{T}_1 \equiv \mu\mathbf{t}.\mathsf{p}!\ell_1(\mathtt{nat}).\mathsf{p}!\ell_2(\mathtt{nat}).\mathbf{t}$, $\mathbb{T}_2 \equiv \mu\mathbf{t}.\mathsf{p}!\ell_2(\mathtt{nat}).\mathsf{p}!\ell_1(\mathtt{nat}).\mathbf{t}$: by [NSUB-LABEL-OUT], we deduce
$$\mathsf{p}!\ell_1(\mathtt{nat}).\mathsf{p}!\ell_2(\mathtt{nat}).\mathcal{T}(\mathbb{T}_1) \not\leqslant \mathsf{p}!\ell_2(\mathtt{nat}).\mathsf{p}!\ell_1(\mathtt{nat}).\mathcal{T}(\mathbb{T}_2),$$
which gives $\mathcal{T}(\mathbb{T}_1) \not\leqslant \mathcal{T}(\mathbb{T}_2)$ by definition of $\mathcal{T}$ and folding.
2. $\mathbb{T}_1 \equiv \mu\mathbf{t}.\mathsf{p}!\ell_1(\mathtt{nat}).\mathbf{t}$, $\mathbb{T}_2 \equiv \mathsf{p}!\ell_1(\mathtt{nat}).\mathtt{end}$: we have
$$\begin{array}{rl} \mathcal{T}(\mathbb{T}_1) \not\leqslant \mathtt{end} & \text{by [NSUB-ENDL]} \\ \mathsf{p}!\ell_1(\mathtt{nat}).\mathcal{T}(\mathbb{T}_1) \not\leqslant \mathsf{p}!\ell_1(\mathtt{nat}).\mathtt{end} & \text{by [NSUB-CONT-OUT]} \end{array}$$
which again implies $\mathcal{T}(\mathbb{T}_1) \not\leqslant \mathcal{T}(\mathbb{T}_2)$.

Intuitively, the rules in Table 7 describe all the cases where two types cannot be related by $\leqslant$ (Definition 3.15). This intuition is formalised in Proposition 4.7 below, which says that the relation $\not\leqslant$ introduced in Definition 4.3 is the complement of the subtyping introduced in Definition 3.15. Hence, for two arbitrary type trees T, T', *either* $\mathsf{T} \leqslant \mathsf{T}'$ *or* $\mathsf{T} \not\leqslant \mathsf{T}'$ must hold, but not both. To prove that relations $\leqslant$ and $\not\leqslant$ are complementary, we rely on algorithmic subtyping. We follow the approach of Ligatti et al. [42,43], with the method of induction on a failing derivation (Definition 3.29); this exploits, in particular, the correspondence between the axioms in Table 7 and the cases listed in Corollary 3.30.

Lemma 4.5. *If* $\mathcal{T}(\mathbb{T}) \not\leqslant \mathcal{T}(\mathbb{T}')$, *then there is a failing derivation of* $\emptyset \vdash \mathbb{T} \leqslant_a \mathbb{T}'$.

Proof. See Appendix B.1. □

Lemma 4.6. *Let* $(J_1, \ldots, J_n)$ *be a failing derivation of* $\emptyset \vdash \mathbb{T} \leqslant_a \mathbb{T}'$, *where* J_m *has the form* $\Theta_m \vdash \mathbb{T}_m \leqslant_a \mathbb{T}'_m$, $m \in \{1, \ldots, n\}$. *Then,* $\mathcal{T}(\mathbb{T}_m) \not\leqslant \mathcal{T}(\mathbb{T}'_m)$ *for every* $m \in \{1, \ldots, n\}$.

Proof. See Appendix B.1. □

Proposition 4.7. $\mathcal{T}(\mathbb{T}) \not\leqslant \mathcal{T}(\mathbb{T}')$ *is derivable* if and only if $\mathcal{T}(\mathbb{T}) \leqslant \mathcal{T}(\mathbb{T}')$ *is* not *derivable.*

Proof. The proof follows from Theorem 3.26, Lemma 3.31, Lemma 4.5 and Lemma 4.6. □

As a direct consequence of the Proposition 4.7, and Definition 4.3, we have that $\mathbb{T}_1 \not\leqslant \mathbb{T}_2$ holds iff $\mathcal{T}(\mathbb{T}_1) \leqslant \mathcal{T}(\mathbb{T}_2)$ is not derivable.

We now proceed with the second step on our way to achieving completeness, in which we introduce suitable processes that behave as specified by given session types.

4.2. Step 2: characteristic processes

The characteristic process $\mathcal{P}(\mathbb{T})$ of a session type $\mathbb{T}$ (Definition 4.8 below) is a process that fulfils two requirements:

1. it behaves as specified by $\mathbb{T}$, and
2. whenever it receives a value v, it checks that v is of the sort expected in $\mathbb{T}$.

To define the characteristic process of a selection type, we define internal choice of processes as a nesting of conditional processes. Namely, if $I = \{1, \ldots, n\}$ with $n \geq 2$, we will write $\bigoplus_{i \in I} P_i$ for:

Table 8
Characteristic processes. Note that we use *syntactic* session types, and the syntactic type equality $\equiv$ (Notation 3.5) — hence, we distinguish a recursive type from its unfoldings, and they yield different characteristic processes.

$$\mathcal{P}(\mathbb{T}) = \begin{cases} \mathbf{0} & \text{if } \mathbb{T} \equiv \mathtt{end} \\ \sum_{i\in I} \mathsf{p}?\ell_i(x_i).\mathtt{if}\ \mathtt{expression}(x_i, S_i)\ \mathtt{then}\ \mathcal{P}(\mathbb{T}'_i)\ \mathtt{else}\ \mathcal{P}(\mathbb{T}'_i) & \text{if } \mathbb{T} \equiv \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathbb{T}'_i \\ \bigoplus_{i\in I} \mathsf{p}!\ell_i(\mathtt{value}(S_i)).\mathcal{P}(\mathbb{T}'_i) & \text{if } \mathbb{T} \equiv \bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathbb{T}'_i \\ \mu X_{\mathbf{t}}.\mathcal{P}(\mathbb{T}') & \text{if } \mathbb{T} \equiv \mu\mathbf{t}.\mathbb{T}' \\ X_{\mathbf{t}} & \text{if } \mathbb{T} \equiv \mathbf{t} \end{cases}$$

$$\mathtt{expression}(x, S) = \begin{cases} \mathtt{succ}(x) > 0 & \text{if } S = \mathtt{nat} \\ \mathtt{neg}(x) > 0 & \text{if } S = \mathtt{int} \\ \neg x & \text{if } S = \mathtt{bool} \end{cases} \qquad \mathtt{value}(S) = \begin{cases} 5 & \text{if } S = \mathtt{nat} \\ -5 & \text{if } S = \mathtt{int} \\ \mathsf{true} & \text{if } S = \mathtt{bool} \end{cases}$$

$$\begin{array}{l} \text{if } (\mathsf{true} \oplus \mathsf{false}) \text{ then } P_1 \text{ else} \\ \qquad \text{if } (\mathsf{true} \oplus \mathsf{false}) \text{ then } P_2 \text{ else} \\ \qquad\quad \cdots \\ \qquad\qquad \text{if } (\mathsf{true} \oplus \mathsf{false}) \text{ then } P_{n-1} \text{ else } P_n \end{array} \tag{4}$$

and if $I = \{1\}$, then $\bigoplus_{i\in I} P_i$ stands for P_1. Note that we can derive $\bigoplus_{i\in I} P_i \longrightarrow^* P_j$ for every $j \in I$, because the non-deterministic operator $\oplus$ on expressions allows $\bigoplus_{i\in I} P_i$ to potentially reduce to any P_i: this capability will be exploited later, to state and prove subtyping completeness (Proposition 4.17).[4]

Definition 4.8. The characteristic process $\mathcal{P}(\mathbb{T})$ of a type $\mathbb{T}$ is defined by induction on $\mathbb{T}$ in Table 8.

By Definition 4.8, the characteristic process $\mathcal{P}(\mathbb{T})$ follows the structure of the originating type $\mathbb{T}$; e.g., for each recursive subterm $\mu\mathbf{t}.\ldots$ of $\mathbb{T}$, we have a corresponding recursive process $\mu X_{\mathbf{t}}.\ldots$ (i.e., we have a distinct process variable $X_{\mathbf{t}}$ for each recursion variable $\mathbf{t}$ in $\mathbb{T}$). To output values of the sorts required by $\mathbb{T}$, Definition 4.8 uses the function $\mathtt{value}(S)$: it associates sorts $\mathtt{nat}, \mathtt{int}$ and $\mathtt{bool}$ with their characteristic values $5, -5$ and true, respectively.[5] Moreover, Definition 4.8 inserts runtime checks to let $\mathcal{P}(\mathbb{T})$ verify that each received value is of the sort expected by $\mathbb{T}$: this is done with the function $\mathtt{expression}(x, S)$, which creates an expression e that can be evaluated only if x is substituted with a value of sort S; we will see later (in the proof of Proposition 4.17) that, when e cannot be evaluated (because a value of the wrong sort has been received), then the characteristic process $\mathcal{P}(\mathbb{T})$ will stop reducing, and this will cause its session to get stuck, too. Note that $\mathtt{expression}(x, S)$ is typable in an environment where x has type S (Lemma 4.9 below).

Lemma 4.9. $\Gamma, x : S \vdash \mathtt{expression}(x, S) : \mathtt{bool}$

Crucially, the characteristic process of $\mathbb{T}$ is typable by $\mathcal{T}(\mathbb{T})$, as per Proposition 4.10 below.

Proposition 4.10. *For all $\mathbb{T}$, it holds that $\vdash \mathcal{P}(\mathbb{T}) : \mathcal{T}(\mathbb{T})$.*

Proof. See Appendix B.2. □

4.3. Step 3: characteristic global types and characteristic contexts

The next step to prove completeness is to define for each session type $\mathbb{T}'$ and participant $\mathsf{r} \notin \mathtt{pt}\{\mathbb{T}'\}$ a (characteristic) context $C_{\mathsf{r},\mathbb{T}'}$, that is well typed when its hole is filled with $\mathsf{r} \triangleleft Q$, for any process Q of type $\mathcal{T}(\mathbb{T}')$. With this aim, we first define a (characteristic) global type that will provide typability of the session.

For a type $\mathbb{T}'$ and a participant r, characteristic global type $\mathcal{G}(\mathbb{T}', \mathsf{r})$ describes a global protocol where participant r behaves as specified by $\mathbb{T}'$, while all participants from $\mathtt{pt}\{\mathbb{T}'\}$ interact in a way that is compatible with $\mathbb{T}'$.

Definition 4.11. The *characteristic global type* $\mathcal{G}(\mathbb{T}', \mathsf{r})$ of the type $\mathbb{T}'$ for the participant $\mathsf{r} \notin \mathtt{pt}\{\mathbb{T}'\}$ is:

$$\mathcal{G}(\mathtt{end}, \mathsf{r}) = \mathtt{end} \quad \text{and} \quad \mathcal{G}(\mathbb{T}', \mathsf{r}) = \mathcal{G}_0(\mathbb{T}', \mathsf{r}, \{\mathsf{p}_j\}_{1\le j\le n}), \text{ if } \mathtt{pt}\{\mathbb{T}'\} = \{\mathsf{p}_j\}_{1\le j\le n},$$

where $\mathcal{G}_0(\mathbb{T}', \mathsf{r}, \{\mathsf{p}_j\}_{1\le j\le n})$ is given in Table 9.

[4] Our results can be formalised and proved without the $\oplus$ operator on expressions, but they would be more cumbersome and less readable: we would need to leave the boolean expressions in (4) unspecified; then, we would need to suitably instantiate such expressions (as true/false) in the statement and proof of subtyping completeness (Proposition 4.17), where we let $\bigoplus_{i\in I} P_i$ reduce to a desired P_k (for some $k \in I$).

[5] The choice of $5/-5$ is arbitrary: any other natural/negative integer value would suit the purpose. Similarly, we could use false instead of true.

Table 9
The function $\mathcal{G}_0(\mathbb{T}', \mathsf{r}, \{\mathsf{p}_j\}_{1\leq j\leq n})$, generating characteristic global types from local types. Note that it is inductively defined on syntactic types.

$$\begin{array}{rcl}
\mathcal{G}_0(\bigwedge_{i\in I} \mathsf{p}_{j_0}?\ell_i(S_i).\mathbb{T}_i, \mathsf{r}, \{\mathsf{p}_j\}_{1\leq j\leq n}) & = & \mathsf{p}_{j_0} \to \mathsf{r} : \{\ell_i(S_i).\mathbb{G}_i^{j_0}\}_{i\in I} \\
\mathcal{G}_0(\bigvee_{i\in I} \mathsf{p}_{j_0}!\ell_i(S_i).\mathbb{T}_i, \mathsf{r}, \{\mathsf{p}_j\}_{1\leq j\leq n}) & = & \mathsf{r} \to \mathsf{p}_{j_0} : \{\ell_i(S_i).\mathbb{G}_i^{j_0}\}_{i\in I} \\
\mathcal{G}_0(\mu\mathbf{t}.\mathbb{T}', \mathsf{r}, \{\mathsf{p}_j\}_{1\leq j\leq n}) & = & \mu\mathbf{t}.\mathcal{G}_0(\mathbb{T}', \mathsf{r}, \{\mathsf{p}_j\}_{1\leq j\leq n}) \\
\mathcal{G}_0(\mathbf{t}, \mathsf{r}, \{\mathsf{p}_j\}_{1\leq j\leq n}) = \mathbf{t} & & \mathcal{G}_0(\mathtt{end}, \mathsf{r}, \{\mathsf{p}_j\}_{1\leq j\leq n}) = \mathtt{end}
\end{array}$$

$$\begin{array}{rcl}
\mathbb{G}_i^{j_0} & = & \mathsf{p}_{j_0} \to \mathsf{p}_{j_0+1} : \ell_i(\mathtt{bool}). \ldots \mathsf{p}_{n-1} \to \mathsf{p}_n : \ell_i(\mathtt{bool}).\mathsf{p}_n \to \mathsf{p}_1 : \ell_i(\mathtt{bool}). \\
 & & \mathsf{p}_1 \to \mathsf{p}_2 : \ell_i(\mathtt{bool}). \ldots . \mathsf{p}_{j_0-1} \to \mathsf{p}_{j_0} : \ell_i(\mathtt{bool}).\mathcal{G}_0(\mathbb{T}_i, \mathsf{r}, \{\mathsf{p}_j\}_{1\leq j\leq n})
\end{array}$$

Definition 4.11 ensures that after each communication involving r and some $\mathsf{q} \in \mathtt{pt}\{\mathbb{T}'\}$, q starts a cyclic communication involving *all* participants from $\mathtt{pt}\{\mathbb{T}'\}$, both as receivers and senders. Such cyclic communications serve two purposes:

1. they ensure that $\mathcal{G}(\mathbb{T}', \mathsf{r})$ is projectable (see Example 4.12 and Lemma 4.14), and
2. they ensure that a session based on $\mathcal{G}(\mathbb{T}', \mathsf{r})$ gets stuck if r does not follow $\mathbb{T}'$ or its subtypes (see Example 4.13); this feature will be leveraged for the proof of operational completeness (Theorem 4.18).

Example 4.12. The cyclic communications produced by $\mathcal{G}(\mathbb{T}', \mathsf{r})$ (Definition 4.11) ensure that characteristic global types can be projected, thus producing a session type for all their participants: this is necessary to type sessions, by rule [T-SESS] in Table 5.

If we do not introduce cyclic communications, this requirement may not be guaranteed. Take for example:

$$\mathbb{T}' = \mathsf{q}!\ell_1(\mathtt{nat}).\mathsf{p}?\ell_2(\mathtt{int}).\mathtt{end} \vee \mathsf{q}!\ell_3(\mathtt{int}).\mathtt{end}.$$

In order to find a global type where a participant r behaves as specified by $\mathbb{T}'$, we might be tempted to consider, e.g., the global type:

$$\mathbb{G} = \mathsf{r} \to \mathsf{q} : \left\{ \begin{array}{l} \ell_1(\mathtt{nat}).\mathsf{p} \to \mathsf{r} : \ell_2(\mathtt{int}).\mathtt{end}, \\ \ell_3(\mathtt{int}).\mathtt{end} \end{array} \right\}$$

Indeed, if we check how r behaves in $\mathbb{G}$, we have $\mathcal{T}(\mathbb{G})\upharpoonright\mathsf{r} = \mathcal{T}(\mathbb{T}')$, as desired. However, by Definition 3.6, $\mathcal{T}(\mathbb{G})\upharpoonright\mathsf{p} = \mathcal{T}(\mathsf{r}!\ell_2(\mathtt{int}).\mathtt{end}) \sqcap \mathcal{T}(\mathtt{end})$ is undefined, and therefore, $\mathbb{G}$ cannot type any session.

To avoid this issue, $\mathcal{G}(\mathbb{T}', \mathsf{r})$ produces a global type where r behaves according to $\mathbb{T}'$ — and furthermore, the other participants appearing in $\mathbb{T}'$ are involved in cyclic communications that can be correctly projected:

$$\mathcal{G}(\mathbb{T}', \mathsf{r}) = \mathsf{r} \to \mathsf{q} : \left\{ \begin{array}{l} \ell_1(\mathtt{nat}).\mathsf{q} \to \mathsf{p} : \ell_1(\mathtt{bool}).\mathsf{p} \to \mathsf{q} : \ell_1(\mathtt{bool}). \\ \qquad \mathsf{p} \to \mathsf{r} : \ell_2(\mathtt{int}).\mathsf{p} \to \mathsf{q} : \ell_2(\mathtt{bool}).\mathsf{q} \to \mathsf{p} : \ell_2(\mathtt{bool}).\mathtt{end}, \\ \ell_3(\mathtt{int}).\mathsf{q} \to \mathsf{p} : \ell_3(\mathtt{bool}).\mathsf{p} \to \mathsf{q} : \ell_3(\mathtt{bool}).\mathtt{end} \end{array} \right\}$$

$$\mathcal{G}(\mathbb{T}', \mathsf{r})\upharpoonright\mathsf{p} = \mathsf{q}?\ell_1(\mathtt{bool}).\mathsf{q}!\ell_1(\mathtt{bool}).\mathsf{r}!\ell_2(\mathtt{int}).\mathsf{q}!\ell_2(\mathtt{bool}).\mathsf{q}?\ell_2(\mathtt{bool}).\mathtt{end} \wedge \mathsf{q}?\ell_3(\mathtt{bool}).\mathsf{q}!\ell_3(\mathtt{bool}).\mathtt{end}$$

Example 4.13. We now explain the necessity of cyclic communications in the definition of characteristic global types, for proving the completeness of $\leqslant$.

If we have a type $\mathbb{T}$, we can find many projectable global types where some participant p behaves as $\mathbb{T}$; however, not all such global types can be used to prove completeness of subtyping. Consider the two session types below, and note that $\mathbb{T} \leqslant \mathbb{T}'$ does *not* hold, whereas $\mathbb{T} \not\leqslant \mathbb{T}'$ does:

$$\begin{array}{rl}
\mathbb{T} & = \mathsf{p}_1!\ell_1(\mathtt{nat}).\mathsf{p}_2!\ell_2(\mathtt{nat}).\mathtt{end} \\
\mathbb{T}' & = \mathsf{p}_2!\ell_2(\mathtt{nat}).\mathsf{p}_1!\ell_1(\mathtt{nat}).\mathtt{end} \qquad \text{(note the swapping of actions w.r.t. } \mathbb{T}\text{)}
\end{array}$$

If we don't require the cyclic communications generated by Definition 4.11, we might be tempted to define a characteristic global type for $\mathbb{T}'$, where p behaves as $\mathbb{T}'$, as follows:

$$\mathbb{G} = \mathsf{p} \to \mathsf{p}_2 : \ell_2(\mathtt{nat}).\mathsf{p} \to \mathsf{p}_1 : \ell_1(\mathtt{nat}).\mathtt{end}$$

We can easily see that $\mathcal{P}(\mathbb{T}') = \mathsf{p}_2!\ell_2(5).\mathsf{p}_1!\ell_1(5).\mathbf{0}$ (by Definition 4.8), and then, $\mathsf{G} = \mathcal{T}(\mathbb{G})$ types the multiparty session $\mathsf{p} \triangleleft \mathcal{P}(\mathbb{T}') \mid \mathcal{M}$, where

$$\mathcal{M} = \mathsf{p}_1 \triangleleft \mathcal{P}(\mathsf{G}\upharpoonright\mathsf{p}_1) \mid \mathsf{p}_2 \triangleleft \mathcal{P}(\mathsf{G}\upharpoonright\mathsf{p}_2) = \mathsf{p}_1 \triangleleft \mathsf{p}?\ell_1(x).\mathbf{0} \mid \mathsf{p}_2 \triangleleft \mathsf{p}?\ell_2(x).\mathbf{0}.$$

Clearly, the session $\mathsf{p} \triangleleft \mathcal{P}(\mathbb{T}') \mid \mathcal{M}$ reduces to $\mathsf{p} \triangleleft \mathbf{0} \mid \mathsf{p}_1 \triangleleft \mathbf{0} \mid \mathsf{p}_2 \triangleleft \mathbf{0}$.

To prove the completeness of $\leqslant$, we need to show that, since $\mathbb{T} \leqslant \mathbb{T}'$ does *not* hold, $\mathcal{M}$ does *not* interact correctly with $\mathcal{P}(\mathbb{T})$. However, being $\mathcal{P}(\mathbb{T}) = \mathsf{p}_1!\ell_1(5).\mathsf{p}_2!\ell_2(5).\mathbf{0}$, also the session $\mathsf{p} \triangleleft \mathcal{P}(\mathbb{T}) \mid \mathcal{M}$ reduces to $\mathsf{p} \triangleleft \mathbf{0}$, without getting stuck. Therefore, $\mathbb{G}$ does not help in proving that subtyping is complete.

Instead, by Definition 4.11, we have the following characteristic global type:

$$\begin{aligned}\mathbb{G}' \;=\; \mathcal{G}(\mathbb{T}', \mathsf{p}) \;=\; & \mathsf{p} \to \mathsf{p}_2 : \ell_2(\mathtt{nat}).\mathsf{p}_2 \to \mathsf{p}_1 : \ell_2(\mathtt{bool}).\mathsf{p}_1 \to \mathsf{p}_2 : \ell_2(\mathtt{bool}).\\ & \mathsf{p} \to \mathsf{p}_1 : \ell_1(\mathtt{nat}).\mathsf{p}_1 \to \mathsf{p}_2 : \ell_1(\mathtt{bool}).\mathsf{p}_2 \to \mathsf{p}_1 : \ell_1(\mathtt{bool}).\mathtt{end}\end{aligned}$$

which implies:

$$\begin{aligned}\mathcal{P}\big(\mathbb{T}'_1\big) \;&=\; \mathsf{p}_2?\ell_2(x).\mathsf{p}_2!\ell(\mathsf{true})\ldots. \quad \text{where } \mathbb{T}'_1 = \mathsf{p}_2?\ell_2(\mathtt{bool}).\mathsf{p}_2!\ell_2(\mathtt{bool})\ldots. \text{ and } \mathcal{T}\big(\mathbb{T}'_1\big) = \mathcal{T}\big(\mathbb{G}'\big)\upharpoonright \mathsf{p}_1\\ \mathcal{P}\big(\mathbb{T}'_2\big) \;&=\; \mathsf{p}?\ell_2(x).\mathsf{p}!\ell(\mathsf{true})\ldots. \quad \text{where } \mathbb{T}'_2 = \mathsf{p}?\ell_2(\mathtt{nat}).\mathsf{p}_1!\ell_2(\mathtt{bool})\ldots. \text{ and } \mathcal{T}\big(\mathbb{T}'_2\big) = \mathcal{T}\big(\mathbb{G}'\big)\upharpoonright \mathsf{p}_2\end{aligned}$$

and it is then easy to verify that:

$$\begin{aligned}&\mathsf{p} \triangleleft \mathcal{P}\big(\mathbb{T}'\big) \mid \mathsf{p}_1 \triangleleft \mathcal{P}\big(\mathbb{T}'_1\big) \mid \mathsf{p}_2 \triangleleft \mathcal{P}\big(\mathbb{T}'_2\big) && \text{reduces to } \mathsf{p} \triangleleft \mathbf{0}\text{, hence is } \textit{not} \text{ stuck,}\\ \text{whereas}\quad &\mathsf{p} \triangleleft \mathcal{P}(\mathbb{T}) \mid \mathsf{p}_1 \triangleleft \mathcal{P}\big(\mathbb{T}'_1\big) \mid \mathsf{p}_2 \triangleleft \mathcal{P}\big(\mathbb{T}'_2\big) && \text{is stuck.}\end{aligned}$$

In order to type a session with the characteristic global type we need to verify that $\mathcal{T}(\mathcal{G}(\mathbb{T}, \mathsf{p}))\upharpoonright \mathsf{p} = \mathcal{T}(\mathbb{T})$, and that $\mathcal{T}(\mathcal{G}(\mathbb{T}, \mathsf{p}))\upharpoonright \mathsf{q}$ is defined for all $\mathsf{q} \in \mathtt{pt}\{\mathbb{T}\}$.

Lemma 4.14. *For all session types $\mathbb{T}$ and $\mathsf{p} \notin \mathtt{pt}\{\mathbb{T}\}$:*

1. $\mathcal{T}(\mathcal{G}(\mathbb{T}, \mathsf{p}))\upharpoonright \mathsf{p} = \mathcal{T}(\mathbb{T})$*;*
2. $\mathcal{T}(\mathcal{G}(\mathbb{T}, \mathsf{p}))\upharpoonright \mathsf{q}$ *is defined for all* $\mathsf{q} \in \mathtt{pt}\{\mathbb{T}\}$.

Proof. See Appendix B. □

We can now define characteristic contexts that satisfy desired properties for the proof of operational completeness.

Definition 4.15 *(Characteristic context).* Let $\mathbb{T}'$ be a session type and $\mathsf{r} \notin \mathtt{pt}\{\mathbb{T}'\}$. A *characteristic context* for $\mathbb{T}'$ and r, denoted by $C_{\mathsf{r},\mathbb{T}'}$, is a context such that:

$$C_{\mathsf{r},\mathbb{T}'} = \begin{cases} \_ & \text{if } \mathbb{T}' = \mathtt{end}\\ \_ \mid \prod\limits_{1 \le i \le n} \mathsf{p}_i \triangleleft \mathcal{P}(\mathbb{T}_i) & \text{if } \mathtt{pt}\{\mathbb{T}'\} = \{\mathsf{p}_1, \ldots, \mathsf{p}_n\}, \text{ and } \forall 1 \le i \le n : \mathcal{T}(\mathbb{T}_i) = \mathcal{T}\big(\mathcal{G}(\mathbb{T}', \mathsf{r})\big)\upharpoonright \mathsf{p}_i \end{cases}$$

When filled with the participant/process pair $\mathsf{r} \triangleleft Q$, where Q is typed as $\vdash Q : \mathcal{T}\big(\mathbb{T}'\big)$, a characteristic context produces a well-typed multiparty session. This is proved in Proposition 4.16 below.

Proposition 4.16. *Take any $\mathbb{T}'$ and $\mathsf{r} \notin \mathtt{pt}\{\mathbb{T}'\}$. If $\vdash Q : \mathcal{T}\big(\mathbb{T}'\big)$, then $\vdash C_{\mathsf{r},\mathbb{T}'_1}[\mathsf{r} \triangleleft Q] : \mathcal{T}\big(\mathcal{G}(\mathbb{T}', \mathsf{r})\big)$, for any characteristic context $C_{\mathsf{r},\mathbb{T}'_1}$ such that $\mathcal{T}\big(\mathbb{T}'\big) = \mathcal{T}\big(\mathbb{T}'_1\big)$.*

Proof. See Appendix B.2. □

4.4. Step 4: operational completeness

We have now all the necessary machinery to prove the last step and achieve the operational completeness of subtyping.

Proposition 4.17. *Take any $\mathbb{T}_1$, $\mathbb{T}'_1$ and $\mathsf{r} \notin \mathtt{pt}\{\mathbb{T}'_1\}$. If $\mathcal{T}(\mathbb{T}_1) \not\trianglelefteq \mathcal{T}\big(\mathbb{T}'_1\big)$, then there are $\mathbb{T}$, $\mathbb{T}'$ and a characteristic context $C_{\mathsf{r},\mathbb{T}'}$ such that $\mathcal{T}(\mathbb{T}) = \mathcal{T}(\mathbb{T}_1)$ and $\mathcal{T}\big(\mathbb{T}'\big) = \mathcal{T}\big(\mathbb{T}'_1\big)$ and $\mathtt{stuck}\big(C_{\mathsf{r},\mathbb{T}'}[\mathsf{r} \triangleleft \mathcal{P}(\mathbb{T})]\big)$.*

Proof. We denote $\mathcal{T}(\mathbb{T}_1)$ and $\mathcal{T}\big(\mathbb{T}'_1\big)$ by T and T', respectively. The proof is by induction on the derivation of $\mathsf{T} \not\trianglelefteq \mathsf{T}'$. More precisely, in each case of the proof, we examine the shape of the *session type trees* related by a rule of $\not\trianglelefteq$ and then pick two corresponding *syntactic* types $\mathbb{T}, \mathbb{T}'$ such that $\mathcal{T}(\mathbb{T}_1) = \mathcal{T}(\mathbb{T}) \not\trianglelefteq \mathcal{T}\big(\mathbb{T}'\big) = \mathcal{T}\big(\mathbb{T}'_1\big)$. We will always pick $\mathbb{T}$ and $\mathbb{T}'$ in unfolded form, so they never have $\mu\mathbf{t}\ldots.$ as top-level term. Then, we examine the characteristic processes of $\mathbb{T}$ and $\mathbb{T}'$. In case $\mathtt{pt}\{\mathbb{T}'\} = \emptyset$ we show that $\mathtt{stuck}(\mathsf{r} \triangleleft \mathcal{P}(\mathbb{T}))$. If $\mathtt{pt}\{\mathbb{T}'\} \neq \emptyset$, we show that

$$\mathtt{stuck}\left(\mathsf{r} \triangleleft \mathcal{P}(\mathbb{T}) \mid \prod_{\mathsf{p}_l \in \mathtt{pt}\{\mathbb{T}'\}} \mathsf{p}_l \triangleleft \mathcal{P}\big(\mathbb{T}_{\mathsf{p}_l}\big)\right)$$

- [NSUB-ENDR] $\mathtt{end} \not\leqslant \mathsf{T}'$ and $\mathsf{T}' \neq \mathtt{end}$:
  By definition of characteristic processes, $\mathcal{P}(\mathbb{T}) = \mathcal{P}(\mathtt{end}) = \mathbf{0}$. Since $\mathsf{T}' \neq \mathtt{end}$, by definition of characteristic global type, there is a participant $\mathsf{q} \in \mathtt{pt}\{\mathbb{T}'\}$ such that the projection of G to q is either selection of outputs to r or branching of inputs from r. Since participant r is inactive, it will never communicate with q, implying that the multiparty session is stuck (other participant are waiting to participate in the cyclic communication, which cannot start).
- [NSUB-ENDL] $\mathsf{T} \not\leqslant \mathtt{end}$ and $\mathsf{T} \neq \mathtt{end}$:
  Then, $\mathtt{pt}\{\mathbb{T}'\} = \mathtt{pt}\{\mathtt{end}\} = \emptyset$, hence $\mathbb{T}'$ has a characteristic context $C_{\mathsf{r},\mathbb{T}'} = \_$. From this, we obtain the multiparty session $C_{\mathsf{r},\mathbb{T}'}[\mathsf{r} \triangleleft \mathcal{P}(\mathbb{T})] = \mathsf{r} \triangleleft \mathcal{P}(\mathbb{T})$, where participant r is *not* inactive, since $\mathtt{pt}\{\mathbb{T}\} \neq \emptyset$: therefore, the session is stuck.
- [NSUB-DIFF-IN] $\bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i \not\leqslant \bigwedge_{j\in J} \mathsf{q}?\ell'_j(S'_j).\mathsf{T}'_j$ and $\mathsf{p} \neq \mathsf{q}$:
  Let us assume that $\mathtt{pt}\{\mathbb{T}'\} = \mathtt{pt}\{\mathsf{T}'\} = \{\mathsf{q}, \mathsf{p}_1, \ldots, \mathsf{p}_{n-1}\}$ (with auxiliary denotations $\mathsf{p}_0 := \mathsf{q}$ and $\mathsf{p}_n := \mathsf{q}$). By definition of characteristic process,

$$\mathcal{P}(\mathbb{T}) \;=\; \mathcal{P}\left(\bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathbb{T}_i\right) = \sum_{i\in I} \mathsf{p}?\ell_i(x).P_i^{\mathsf{r}},$$

  for some P_i^{r}. By definition of characteristic global type, $\mathbb{G} = \mathcal{G}(\mathbb{T}', \mathsf{r})$ where

$$\mathcal{G}(\mathbb{T}', \mathsf{r}) \;=\; \mathcal{G}_0(\mathbb{T}', \mathsf{r}, \mathtt{pt}\{\mathbb{T}'\}) = \mathcal{G}_0(\bigwedge_{j\in J} \mathsf{q}?\ell'_j(S'_j).\mathbb{T}'_j, \mathsf{r}, \mathtt{pt}\{\mathbb{T}'\}) = \mathsf{q} \to \mathsf{r} : \{\ell'_j(S'_j).\mathbb{G}_j^{\mathsf{q}}\}_{j\in J}.$$

  Now, the cyclic communication that includes all the participants from $\mathbb{T}'$ has the form:

$$\mathbb{G}_j^{\mathsf{q}} \;=\; \mathsf{q} \to \mathsf{p}_1 : \ell'_j(\mathtt{bool}).\mathsf{p}_1 \to \mathsf{p}_2 : \ell'_j(\mathtt{bool}).\ldots.\mathsf{p}_{n-1} \to \mathsf{q} : \ell'_j(\mathtt{bool}).\mathcal{G}_0(\mathsf{T}'_j, \mathsf{r}, \mathtt{pt}\{\mathbb{T}'\}).$$

  By definition of global type projection and characteristic process, we obtain

$$\begin{aligned}
\mathbb{T}_{\mathsf{q}} \quad &= \mathbb{G}\upharpoonright\mathsf{q} = \bigvee_{j\in J} \mathsf{r}!\ell'_j(S'_j).\mathbb{G}_j^{\mathsf{q}}\upharpoonright\mathsf{q} \\
\mathbb{T}_{\mathsf{p}_l} \quad &= \mathbb{G}\upharpoonright\mathsf{p}_l = \bigwedge_{j\in J} \mathsf{p}_{l-1}?\ell'_j(\mathtt{bool}).\mathsf{p}_{l+1}!\ell'_j(\mathtt{bool}).\mathbb{G}_{0j}^{\mathsf{q}}\upharpoonright\mathsf{p}_l,\ l \in \{1, \ldots, n-1\} \\
\mathcal{P}(\mathbb{T}_{\mathsf{q}}) \;&= \bigoplus_{j\in J} \mathsf{r}!\ell'_j(\mathtt{value}(S'_j)).P_j^{\mathsf{q}} \\
\mathcal{P}(\mathbb{T}_{\mathsf{p}_l}) \;&= \sum_{j\in J} \mathsf{p}_{l-1}?\ell'_j(x).P''_j
\end{aligned}$$

  for $\mathbb{G}_{0j}^{\mathsf{q}} = \mathcal{G}_0(\mathbb{T}'_j, \mathsf{r}, \mathtt{pt}\{\mathbb{T}'\})$ and some P_j^{q}, P''_j.
  By consecutive application of [T-CONDITIONAL], the multiparty session we consider reduces to

$$\mathsf{r} \triangleleft \sum_{i\in I} \mathsf{p}?\ell_i(x).P_i^{\mathsf{r}} \mid \mathsf{q} \triangleleft \mathsf{r}!\ell'_{j_0}(\mathtt{value}(S'_{j_0})).P_{j_0}^{\mathsf{q}} \mid \prod_{1\leq l\leq n-1} \mathsf{p}_l \triangleleft \sum_{j\in J} \mathsf{p}_{l-1}?\ell'_j(x).P''_j.$$

  The session is stuck since it can not reduce. Notice that all participants p_l, $l \in \{1, \ldots, n-1\}$, are waiting to communicate within the cyclic communication that starts by interaction between q and p_1. The participant q is emitting a message to r, but r is waiting for a message from some other participant (since $\mathsf{p} \neq \mathsf{q}$).
- [NSUB-DIFF-OUT] $\bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathsf{T}_i \not\leqslant \bigvee_{j\in J} \mathsf{q}!\ell'_j(S'_j).\mathsf{T}'_j$ and $\mathsf{p} \neq \mathsf{q}$:
  By definition of characteristic process,

$$\mathcal{P}(\mathbb{T}) \;=\; \mathcal{P}\left(\bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathbb{T}_i\right) = \bigoplus_{i\in I} \mathsf{p}!\ell_i(\mathtt{value}(S_i)).\mathcal{P}(\mathbb{T}_i)\,.$$

  By definition of characteristic global type,

$$\mathbb{G} \;=\; \mathcal{G}(\mathbb{T}', \mathsf{r}) = \mathcal{G}_0(\mathbb{T}', \mathsf{r}, \mathtt{pt}\{\mathbb{T}'\}) = \mathcal{G}_0(\bigvee_{j\in J} \mathsf{q}!\ell'_j(S'_j).\mathbb{T}'_j, \mathsf{r}, \mathtt{pt}\{\mathbb{T}'\}) = \mathsf{r} \to \mathsf{q} : \{\ell'_j(S'_j).\mathbb{G}_j^{\mathsf{q}}\}_{j\in J}.$$

  The cyclic communication and its projections are the same as in the previous case. By definition of global type projection and characteristic process,

$$\mathbb{T}_{\mathsf{q}} = \mathbb{G}\upharpoonright\mathsf{q} = \bigwedge_{j\in J} \mathsf{r}?\ell'_j(S'_j).\mathbb{G}_j^{\mathsf{q}}\upharpoonright\mathsf{q} \quad \text{and} \quad \mathcal{P}(\mathbb{T}_{\mathsf{q}}) = \sum_{j\in J} \mathsf{r}?\ell'_j(x).P_j^{\mathsf{q}},$$

  for some P_j^{q}, $j \in J$.
  By consecutive application of [T-CONDITIONAL], the considered multiparty session reduces to:

$$\mathsf{r} \triangleleft \mathsf{p}!\ell_{i_0}(\mathtt{value}(S_{i_0})).\mathcal{P}\big(\mathbb{T}_{i_0}\big) \mid \mathsf{q} \triangleleft \sum_{j\in J} \mathsf{r}?\ell'_j(x).P_j^{\mathsf{q}} \mid \prod_{1\le l\le n-1} \mathsf{p}_l \triangleleft \sum_{j\in J} \mathsf{p}_{l-1}?\ell'_j(x).P''_j.$$

Similarly as in the previous case, the session is stuck.

- [NSUB-IN-OUT]: $\bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i \not\leq \bigvee_{j\in J} \mathsf{q}!\ell'_j(S'_j).\mathsf{T}'_j$

  Type trees T and T' are the same as in [NSUB-DIFF-IN] and [NSUB-DIFF-OUT], respectively. Therefore, the multiparty session that we consider has the following form:

$$\mathsf{r} \triangleleft \sum_{i\in I} \mathsf{p}?\ell_i(x).P_i^{\mathsf{r}} \mid \mathsf{q} \triangleleft \sum_{j\in J} \mathsf{r}?\ell'_j(x).P_j^{\mathsf{q}} \mid \prod_{1\le l\le n-1} \mathsf{p}_l \triangleleft \sum_{j\in J} \mathsf{p}_{l-1}?\ell'_j(x).P''_j.$$

  The session is stuck, since all the participants are waiting to receive messages.

- [NSUB-OUT-IN]: $\bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathsf{T}_i \not\leq \bigwedge_{j\in J} \mathsf{q}?\ell'_j(S'_j)\mathsf{T}'_j$

  Type trees T and T' are the same as in [NSUB-DIFF-OUT] and [NSUB-DIFF-IN], respectively. Hence, we get the multiparty session:

$$\mathsf{r} \triangleleft \mathsf{p}!\ell_{i_0}(\mathtt{value}(S_{i_0})).\mathcal{P}\big(\mathbb{T}_{i_0}\big) \mid \mathsf{q} \triangleleft \mathsf{r}!\ell'_{j_0}(\mathtt{value}(S'_{j_0})).P_{j_0}^{\mathsf{q}} \mid \prod_{1\le l\le n-1} \mathsf{p}_l \triangleleft \sum_{j\in J} \mathsf{p}_{l-1}?\ell'_j(x).P''_j.$$

  It is stuck, since the cyclic communication should start by communication between q and p_1. The participant q is trying to send a message to r, whilst r is in the same time trying to output a message to p. Even in the case when $\mathsf{p} = \mathsf{p}_l$, for some $l \in L$, the message can not be received before the cyclic communication is completed (and r is not participating in it).

- $\mathsf{T} = \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i$, $\mathsf{T}' = \bigwedge_{j\in J} \mathsf{p}?\ell'_j(S'_j).\mathsf{T}'_j$:

  Let us assume that $\mathtt{pt}\{\mathbb{T}'\} = \mathtt{pt}\{\mathsf{T}'\} = \{\mathsf{p}, \mathsf{p}_1, \ldots, \mathsf{p}_{n-1}\}$ (with auxiliary denotations $\mathsf{p}_0 := \mathsf{p}$ and $\mathsf{p}_n := \mathsf{p}$).

$$\begin{aligned}
\mathbb{G} &= \mathcal{G}_0(\bigwedge_{j\in J} \mathsf{p}?\ell'_j(S'_j).\mathbb{T}'_j, \mathsf{r}, \mathtt{pt}\{\mathbb{T}'\}) = \mathsf{p} \to \mathsf{r} : \{\ell'_j(S'_j).\mathbb{G}_j^p\}_{j\in J}\\
\mathbb{G}_j^p &= \mathsf{p} \to \mathsf{p}_1 : \ell'_j(\mathtt{bool}).\mathsf{p}_1 \to \mathsf{p}_2 : \ell'_j(\mathtt{bool}).\ldots.\mathsf{p}_{n-1} \to \mathsf{p} : \ell'_j(\mathtt{bool}).\mathcal{G}_0(\mathbb{T}'_j, \mathsf{r}, \mathtt{pt}\{\mathbb{T}'\})\\
\mathbb{T}_{\mathsf{p}} &= \mathbb{G}\upharpoonright\mathsf{p} = \bigvee_{j\in J} \mathsf{r}!\ell'_j(S'_j).\mathbb{G}_j^p\upharpoonright\mathsf{p}\\
\mathbb{T}_{\mathsf{p}_l} &= \mathbb{G}\upharpoonright\mathsf{p}_l = \sum_{j\in J} \mathsf{p}_{l-1}?\ell'_j(\mathtt{bool}).\mathsf{p}_{l+1}!\ell'_j(\mathtt{bool}).\mathbb{G}_j\upharpoonright\mathsf{p}_l,\ \mathbb{G}_j = \mathcal{G}_0(\mathbb{T}'_j, \mathsf{r}, \mathtt{pt}\{\mathbb{T}'\})
\end{aligned}$$

  We consider the following cases:

  – [NSUB-LABEL-IN]: $\exists j_0 \in J\ \forall i \in I : \ell_i \neq \ell'_{j_0}$

$$\begin{aligned}
&\mathsf{r} \triangleleft \sum_{i\in I} \mathsf{p}?\ell_i(x).P_i^{\mathsf{r}} \mid \mathsf{p} \triangleleft \bigoplus_{j\in J} \mathsf{r}!\ell'_j(\mathtt{value}(S'_j)).Q_j^{\mathsf{p}} \mid \prod_{1\le l\le n-1} \mathsf{p}_l \triangleleft \mathcal{P}\big(\mathbb{T}_{\mathsf{p}_l}\big)\\
&\longrightarrow^* \mathsf{r} \triangleleft \sum_{i\in I} \mathsf{p}?\ell_i(x).P_i^{\mathsf{r}} \mid \mathsf{p} \triangleleft \mathsf{r}!\ell'_{j_0}(\mathtt{value}(S'_{j_0})).Q_{j_0}^{\mathsf{p}} \mid \prod_{1\le l\le n-1} \mathsf{p}_l \triangleleft \mathcal{P}\big(\mathbb{T}_{\mathsf{p}_l}\big) \not\longrightarrow
\end{aligned}$$

    The session is stuck due to the label mismatch.

  – [NSUB-EXCH-IN]: $\exists i_0 \in I\ \exists j_0 \in J :\ \ell_{i_0} = \ell'_{j_0}$ and $S'_{j_0} \not\leq: S_{i_0}$

    We prove the case $|I| > 1$ with $S_{i_0} = \mathtt{nat}$ and $S'_{j_0} = \mathtt{int}$. Other cases are similar.

$$\begin{aligned}
&\mathsf{r} \triangleleft \mathsf{p}?\ell'_{j_0}(x).\mathsf{if}\ \mathtt{succ}(x) > 0\ \mathsf{then}\ \mathcal{P}\big(\mathbb{T}_{i_0}\big)\ \mathsf{else}\ \mathcal{P}\big(\mathbb{T}_{i_0}\big) + P_2 \mid\\
&\mathsf{p} \triangleleft \bigoplus_{j\in J} \mathsf{r}!\ell'_j(\mathtt{value}(S'_j)).Q_j^{\mathsf{p}} \mid \prod_{1\le l\le n-1} \mathsf{p}_l \triangleleft \mathcal{P}\big(\mathbb{T}_{\mathsf{p}_l}\big) \longrightarrow^*\\
&\mathsf{r} \triangleleft \mathsf{p}?\ell'_{j_0}(x).\mathsf{if}\ \mathtt{succ}(x) > 0\ \mathsf{then}\ \mathcal{P}\big(\mathbb{T}_{i_0}\big)\ \mathsf{else}\ \mathcal{P}\big(\mathbb{T}_{i_0}\big) + P_2 \mid \mathsf{p} \triangleleft \mathsf{r}!\ell'_{j_0}(-5).Q_{j_0}^{\mathsf{p}} \mid\\
&\prod_{1\le l\le n-1} \mathsf{p}_l \triangleleft \mathcal{P}\big(\mathbb{T}_{\mathsf{p}_l}\big) \longrightarrow\\
&\mathsf{r} \triangleleft \mathsf{if}\ \mathtt{succ}(-5) > 0\ \mathsf{then}\ \mathcal{P}\big(\mathbb{T}_{i_0}\big)\ \mathsf{else}\ \mathcal{P}\big(\mathbb{T}_{i_0}\big) \mid \mathsf{p} \triangleleft Q_{j_0}^{\mathsf{p}} \mid \prod_{1\le l\le n-1} \mathsf{p}_l \triangleleft \mathcal{P}\big(\mathbb{T}_{\mathsf{p}_l}\big)
\end{aligned}$$

    This session gets stuck, since $\mathtt{succ}(-5)$ is not defined.

  – [NSUB-CONT-IN]: $\exists i_0 \in I\ \exists j_0 \in J :\ \ell_{i_0} = \ell'_{j_0}$ and $S'_{j_0} \leq: S_{i_0}$ and $\mathsf{T}_{i_0} \not\leq \mathsf{T}'_{j_0}$

    We prove the case $|I| > 1$ with $S_{i_0} = \mathtt{nat}$ and $S'_{j_0} = \mathtt{nat}$. Other cases are similar.

$$\mathsf{r} \triangleleft \mathsf{p}?\ell'_{j_0}(x).\mathsf{if}\ \mathtt{succ}(x) > 0\ \mathsf{then}\ \mathcal{P}(\mathbb{T}_{i_0})\ \mathsf{else}\ \mathcal{P}(\mathbb{T}_{i_0}) + P_2 \mid$$
$$\mathsf{p} \triangleleft \bigoplus_{j\in J} \mathsf{r}!\ell'_j(\mathtt{value}(S'_j)).Q_j^{\mathsf{p}} \mid \prod_{1\le l\le n-1} \mathsf{p}_l \triangleleft \mathcal{P}(\mathbb{T}_{\mathsf{p}_l}) \longrightarrow^*$$
$$\mathsf{r} \triangleleft \mathsf{p}?\ell'_{j_0}(x).\mathsf{if}\ \mathtt{succ}(x) > 0\ \mathsf{then}\ \mathcal{P}(\mathbb{T}_{i_0})\ \mathsf{else}\ \mathcal{P}(\mathbb{T}_{i_0}) + P_2 \mid \mathsf{p} \triangleleft \mathsf{r}!\ell'_{j_0}(5).Q_{j_0}^{\mathsf{p}} \mid$$
$$\prod_{1\le l\le n-1} \mathsf{p}_l \triangleleft \mathcal{P}(\mathbb{T}_l) \longrightarrow \mathsf{r} \triangleleft \mathsf{if}\ \mathtt{succ}(5) > 0\ \mathsf{then}\ \mathcal{P}(\mathbb{T}_{i_0})\ \mathsf{else}\ \mathcal{P}(\mathbb{T}_{i_0}) \mid$$
$$\mathsf{p} \triangleleft Q_{j_0}^{\mathsf{p}} \mid \prod_{1\le l\le n-1} \mathsf{p}_l \triangleleft \mathcal{P}(\mathbb{T}_{\mathsf{p}_l}) \longrightarrow^* \mathsf{r} \triangleleft \mathcal{P}(\mathbb{T}_{i_0}) \mid \mathsf{p} \triangleleft \mathcal{P}(\mathbb{G}_{j_0}\upharpoonright \mathsf{p}) \mid \prod_{1\le l\le n-1} \mathsf{p}_l \triangleleft \mathcal{P}(\mathbb{G}_{j_0}\upharpoonright \mathsf{p}_l)$$

This session is stuck by induction hypothesis.

- $\mathsf{T} = \bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathsf{T}_i$, $\mathsf{T}' = \bigvee_{j\in J} \mathsf{p}!\ell'_j(S'_j).\mathsf{T}'_j$:

Assume that $\mathtt{pt}\{\mathbb{T}'\} = \mathtt{pt}\{\mathsf{T}'\} = \{\mathsf{p}, \mathsf{p}_1, \ldots, \mathsf{p}_{n-1}\}$.

$$\begin{aligned}
\mathbb{G} &= \mathcal{G}_0(\textstyle\bigvee_{j\in J} \mathsf{p}_j!\ell'_j(S'_j).\mathbb{T}'_j, \mathsf{r}, \mathtt{pt}\{\mathbb{T}'\}) = \mathsf{r} \to \mathsf{p} : \{\ell'_j(S'_j).\mathbb{G}_j^p\}_{j\in J} \\
\mathbb{G}_j^p &= \mathsf{p} \to \mathsf{p}_1 : \ell'_j(\mathtt{bool}).\mathsf{p}_1 \to \mathsf{p}_2 : \ell'_j(\mathtt{bool}).\ldots.\mathsf{p}_{n-1} \to \mathsf{p} : \ell'_j(\mathtt{bool}).\mathcal{G}_0(\mathbb{T}'_j, \mathsf{r}, \mathtt{pt}\{\mathbb{T}'\}) \\
\mathbb{T}_{\mathsf{p}} &= \mathbb{G}\upharpoonright \mathsf{p} = \textstyle\bigwedge_{j\in J} \mathsf{r}?\ell'_j(S'_j).\mathbb{G}_j^p\upharpoonright \mathsf{p} \\
\mathbb{T}_{\mathsf{p}_l} &= \mathbb{G}\upharpoonright \mathsf{p}_l = \textstyle\sum_{j\in J} \mathsf{p}_{l-1}?\ell'_j(\mathtt{bool}).\mathsf{p}_{l+1}!\ell'_j(\mathtt{bool}).\mathbb{G}_j\upharpoonright \mathsf{p}_l,\ \mathbb{G}_j = \mathcal{G}_0(\mathbb{T}'_j, \mathsf{r}, \mathtt{pt}\{\mathbb{T}'\}) \\
\mathcal{P}(\mathbb{T}_{\mathsf{p}}) &= \textstyle\sum_{j\in J} \mathsf{r}?\ell_j(x).P_j^{\mathsf{p}} \\
\mathcal{P}(\mathbb{T}) &= \textstyle\bigoplus_{i\in I} \mathsf{p}!\ell_i(\mathtt{value}(S_i)).\mathcal{P}(\mathbb{T}_i)
\end{aligned}$$

We consider the following cases:

– [NSUB-LABEL-OUT]: $\exists i_0 \in I\ \forall j \in J : \ell_{i_0} \neq \ell'_j$
The session gets stuck due to a label mismatch:

$$\mathsf{r} \triangleleft \bigoplus_{i\in I} \mathsf{p}!\ell_i(\mathtt{value}(S_i)).\mathcal{P}(\mathbb{T}_i) \mid \mathsf{p} \triangleleft \sum_{j\in J} \mathsf{r}?\ell'_j(x).P_j^{\mathsf{p}} \mid \prod_{1\le l\le n-1} \mathsf{p}_l \triangleleft \mathcal{P}(\mathbb{T}_{\mathsf{p}_l})$$
$$\longrightarrow^* \mathsf{r} \triangleleft \mathsf{p}!\ell_{i_0}(\mathtt{value}(S_{i_0})).\mathcal{P}(\mathbb{T}_{i_0}) \mid \mathsf{p} \triangleleft \sum_{j\in J} \mathsf{r}?\ell'_j(x).P_j^{\mathsf{p}} \mid \prod_{1\le l\le n-1} \mathsf{p}_l \triangleleft \mathcal{P}(\mathbb{T}_{\mathsf{p}_l}) \not\longrightarrow$$

– [NSUB-EXCH-IN]: $\exists i_0 \in I\ \exists j_0 \in J :\ \ell_{i_0} = \ell'_{j_0}$ and $S_{i_0} \not\leq: S'_{j_0}$
We prove the case $|J| > 1$ with $S_{i_0} = \mathtt{int}$ and $S'_{j_0} = \mathtt{nat}$.
The session gets stuck, since $\mathtt{succ}(-5)$ is not defined:

$$\mathsf{r} \triangleleft \bigoplus_{i\in I} \mathsf{p}!\ell_i(\mathtt{value}(S_i)).\mathcal{P}(\mathbb{T}_i) \mid \mathsf{p} \triangleleft \mathsf{r}?\ell_{i_0}(x).\mathsf{if}\ \mathtt{succ}(x) > 0\ \mathsf{then}\ Q_{j_0}^{\mathsf{p}}\ \mathsf{else}\ Q_{j_0}^{\mathsf{p}} + P_2 \mid$$
$$\prod_{1\le l\le n-1} \mathsf{p}_l \triangleleft \mathcal{P}(\mathbb{T}_{\mathsf{p}_l}) \longrightarrow^* \mathsf{r} \triangleleft \mathsf{p}!\ell_{i_0}(-5).\mathcal{P}(\mathbb{T}_{i_0}) \mid$$
$$\mathsf{p} \triangleleft \mathsf{r}?\ell_{i_0}(x).\mathsf{if}\ \mathtt{succ}(x) > 0\ \mathsf{then}\ Q_{j_0}^{\mathsf{p}}\ \mathsf{else}\ Q_{j_0}^{\mathsf{p}} + P_2 \mid \prod_{1\le l\le n-1} \mathsf{p}_l \triangleleft \mathcal{P}(\mathbb{T}_{\mathsf{p}_l}) \longrightarrow$$
$$\mathsf{r} \triangleleft \mathcal{P}(\mathbb{T}_{i_0}) \mid \mathsf{p} \triangleleft \mathsf{if}\ \mathtt{succ}(-5) > 0\ \mathsf{then}\ Q_{j_0}^{\mathsf{p}}\ \mathsf{else}\ Q_{j_0}^{\mathsf{p}} \mid \prod_{1\le l\le n-1} \mathsf{p}_l \triangleleft \mathcal{P}(\mathbb{T}_{\mathsf{p}_l})$$

– [NSUB-CONT-IN]: $\exists i_0 \in I\ \exists j_0 \in J :\ \ell_{i_0} = \ell'_{j_0}$ and $S_{i_0} \leq: S'_{j_0}$ and $\mathbb{T}_{i_0} \not\leq \mathbb{T}'_{j_0}$
The session gets stuck by induction hypothesis:

$$\mathsf{r} \triangleleft \bigoplus_{i\in I} \mathsf{p}!\ell_i(\mathtt{value}(S_i)).\mathcal{P}(\mathbb{T}_i) \mid \mathsf{p} \triangleleft \mathsf{r}?\ell_{i_0}(x).\mathsf{if}\ \mathtt{succ}(x) > 0\ \mathsf{then}\ Q_{j_0}^{\mathsf{p}}\ \mathsf{else}\ Q_{j_0}^{\mathsf{p}} + P_2 \mid$$
$$\prod_{1\le l\le n-1} \mathsf{p}_l \triangleleft \mathcal{P}(\mathbb{T}_l) \longrightarrow^* \mathsf{r} \triangleleft \mathsf{p}!\ell_{i_0}(5).\mathcal{P}(\mathbb{T}_{i_0}) \mid$$
$$\mathsf{p} \triangleleft \mathsf{r}?\ell_{i_0}(x).\mathsf{if}\ \mathtt{succ}(x) > 0\ \mathsf{then}\ Q_{j_0}^{\mathsf{p}}\ \mathsf{else}\ Q_{j_0}^{\mathsf{p}} + P_2 \mid \prod_{1\le l\le n-1} \mathsf{p}_l \triangleleft \mathcal{P}(\mathbb{T}_l) \longrightarrow$$
$$\mathsf{r} \triangleleft \mathcal{P}(\mathbb{T}_{i_0}) \mid \mathsf{p} \triangleleft \mathsf{if}\ \mathtt{succ}(5) > 0\ \mathsf{then}\ Q_{j_0}^{\mathsf{p}}\ \mathsf{else}\ Q_{j_0}^{\mathsf{p}} \mid \prod_{1\le l\le n-1} \mathsf{p}_l \triangleleft \mathcal{P}(\mathbb{T}_l) \longrightarrow^*$$
$$\mathsf{r} \triangleleft \mathcal{P}(\mathbb{T}_{i_0}) \mid \mathsf{p} \triangleleft \mathcal{P}(\mathbb{G}_{j_0}\upharpoonright \mathsf{p}) \mid \prod_{1\le l\le n-1} \mathsf{p}_l \triangleleft \mathcal{P}(\mathbb{G}_{j_0}\upharpoonright \mathsf{p}_l) \qquad \square$$

Theorem 4.18 *(Completeness). The synchronous multiparty session subtyping is operationally complete.*

Proof. It follows from Proposition 4.10, Proposition 4.16 and Proposition 4.17. □

As a consequence of Theorem 4.2 and Theorem 4.18, we obtain operational preciseness.

Corollary 4.19 *(Preciseness). The synchronous multiparty session subtyping is operationally precise.*

Moreover, multiparty session subtyping $\leqslant$ is the unique precise subtyping for the given calculus.

Corollary 4.20 *(Uniqueness). The synchronous multiparty session subtyping is the unique operationally precise subtyping.*

Proof. Let $\sqsubseteq$ be a reflexive and transitive relation over tree types such that $\sqsubseteq \not\subseteq \leqslant$, i.e., there are types $\mathbb{T}$ and $\mathbb{T}'$ such that $\mathcal{T}(\mathbb{T}) \sqsubseteq \mathcal{T}(\mathbb{T}')$ but $\mathcal{T}(\mathbb{T}) \not\leqslant \mathcal{T}(\mathbb{T}')$. We prove that $\sqsubseteq$ is not an operationally sound subtyping. By the proof of Proposition 4.17, $\mathcal{T}(\mathbb{T}) \not\leqslant \mathcal{T}(\mathbb{T}')$ implies $\mathtt{stuck}(C_{\mathtt{r},\mathbb{T}'}[\mathtt{r} \triangleleft \mathcal{P}(\mathbb{T})])$. Therefore, by Definition 4.1, $\sqsubseteq$ is not an operationally sound subtyping. We conclude that if $\sqsubseteq$ is sound, then $\sqsubseteq \subseteq \leqslant$. Hence, $\leqslant$ is the largest operationally sound subtyping for the considered calculus, and therefore the unique operationally precise one. $\square$

5. Operational preciseness at work

Consider the following multiparty session, with four participants: client (cl), adder (add), increment (inc), and decrement (dec):

$$\mathtt{cl} \triangleleft P_{\mathtt{cl}} \mid \mathtt{add} \triangleleft P_{\mathtt{add}} \mid \mathtt{inc} \triangleleft P_{\mathtt{inc}} \mid \mathtt{dec} \triangleleft P_{\mathtt{dec}}.$$

Client sends two natural numbers to adder and expects the integer result of summation. Adder receives the two numbers and sums them by successively increasing the first one by 1 (done by inc) and decreasing the second one by 1 (done by dec). If the second summand equals 0, the first summand gives the required sum. Processes modelling this behaviour are the following:

$$\begin{aligned}
P_{\mathtt{cl}} &= \mathtt{add}!\ell_1(5).\mathtt{add}!\ell_2(4).\mathtt{add}?\ell_3(x).\mathbf{0}\\
P_{\mathtt{add}} &= \mathtt{cl}?\ell_1(y_1).\mathtt{cl}?\ell_2(y_2).\mu X.\text{if } y_2 = 0 \text{ then } \mathtt{inc}!\ell_4(\text{true}).\mathtt{dec}!\ell_4(\text{true}).\mathtt{cl}!\ell_3(y_1).\mathbf{0}\\
&\qquad\qquad \text{else } \mathtt{inc}!\ell_5(y_1).\mathtt{inc}?\ell_6(y_1).\mathtt{dec}!\ell_7(y_2).\mathtt{dec}?\ell_8(y_2).X\\
P_{\mathtt{inc}} &= \mu X.\mathtt{add}?\ell_4(y).\mathbf{0} + \mathtt{add}?\ell_5(y).\mathtt{add}!\ell_6(y+1).X\\
P_{\mathtt{dec}} &= \mu X.\mathtt{add}?\ell_4(y).\mathbf{0} + \mathtt{add}?\ell_7(y).\mathtt{add}!\ell_8(y-1).X.
\end{aligned}$$

We can extend addition to integers by changing the process $P_{\mathtt{add}}$ as follows:

$$\begin{aligned}
P'_{\mathtt{add}} = \mathtt{cl}?\ell_1(y_1).\mathtt{cl}?\ell_2(y_2).\mu X.&\text{if } y_2 = 0 \text{ then } \mathtt{inc}!\ell_4(\text{true}).\mathtt{dec}!\ell_4(\text{true}).\mathtt{cl}!\ell_3(y_1).\mathbf{0}\\
&\text{else if } y_2 > 0 \text{ then } \mathtt{inc}!\ell_5(y_1).\mathtt{inc}?\ell_6(y_1).\mathtt{dec}!\ell_7(y_2).\mathtt{dec}?\ell_8(y_2).X\\
&\text{else } \mathtt{inc}!\ell_5(y_2).\mathtt{inc}?\ell_6(y_2).\mathtt{dec}!\ell_7(y_1).\mathtt{dec}?\ell_8(y_1).X.
\end{aligned}$$

Process $P'_{\mathtt{add}}$ additionally checks if the second summand is positive. If it is not, the sum is calculated by successively increasing the second summand by 1 and decreasing the first summand by 1. The new multiparty session follows the global protocol:

$$\begin{aligned}
&\mathtt{cl} \to \mathtt{add} : \ell_1(\mathtt{int}).\mathtt{cl} \to \mathtt{add} : \ell_2(\mathtt{int}).\mu\mathbf{t}.\mathtt{add} \to \mathtt{inc} : \{\\
&\quad \ell_4(\mathtt{bool}) : \mathtt{add} \to \mathtt{dec} : \ell_4(\mathtt{bool}).\mathtt{add} \to \mathtt{cl} : \ell_3(\mathtt{int}).\mathtt{end},\\
&\quad \ell_5(\mathtt{int}).\mathtt{inc} \to \mathtt{add} : \ell_6(\mathtt{int}).\mathtt{add} \to \mathtt{dec} : \ell_7(\mathtt{int}).\mathtt{dec} \to \mathtt{add} : \ell_8(\mathtt{int}).\mathbf{t}\,\}.
\end{aligned}$$

Operational soundness of the subtyping guarantees that the summation of natural numbers will be safe after this change, as for $\mathtt{nat} \leqslant \mathtt{int}$ we have

$$\mathtt{add}!\ell_1(\mathtt{nat}).\mathtt{add}!\ell_2(\mathtt{nat}).\mathtt{add}?\ell_3(\mathtt{int}).\mathtt{end} \;\leqslant\; \mathtt{add}!\ell_1(\mathtt{int}).\mathtt{add}!\ell_2(\mathtt{int}).\mathtt{add}?\ell_3(\mathtt{int}).\mathtt{end}.$$

Moreover, our operational completeness result means that any extension of the subtyping relation is not correct. E.g., consider the following types:

$$\mathbb{T} = \mathtt{add}!\ell_1(\mathtt{int}).\mathtt{add}!\ell_2(\mathtt{int}).\mathtt{end} \;\not\leqslant\; \mathtt{add}!\ell_2(\mathtt{int}).\mathtt{add}!\ell_1(\mathtt{int}).\mathtt{end} = \mathbb{T}'.$$

They send messages with different labels in a different order, and thus, are not related by subtyping. By operational completeness, we know that we can construct processes $Q_{\mathtt{cl}} = \mathtt{add}!\ell_1(5).\mathtt{add}!\ell_2(4).\mathbf{0}$ of type $\mathbb{T}$ and $Q'_{\mathtt{cl}} = \mathtt{add}!\ell_2(4).\mathtt{add}!\ell_1(5).\mathbf{0}$ of type $\mathbb{T}'$ and a multiparty session

$$\mathcal{M} = \mathtt{add} \triangleleft \mathtt{cl}?\ell_2(x).\text{if } \mathtt{neg}(x) > 0 \text{ then } \mathtt{cl}?\ell_1(x).\mathbf{0} \text{ else } \mathtt{cl}?\ell_1(x).\mathbf{0}$$

such that $\mathtt{cl} \triangleleft Q'_{\mathtt{cl}} \mid \mathcal{M}$ is well typed (hence has progress, by Theorem 3.23) — whereas $\mathtt{cl} \triangleleft Q_{\mathtt{cl}} \mid \mathcal{M}$ is stuck, since the multiparty session

$$\mathtt{cl} \triangleleft \mathtt{add}!\ell_1(5).\mathtt{add}!\ell_2(4).\mathbf{0} \mid \mathtt{add} \triangleleft \mathtt{cl}?\ell_2(x).\text{if } \mathtt{neg}(x) > 0 \text{ then } \mathtt{cl}?\ell_1(x).\mathbf{0} \text{ else } \mathtt{cl}?\ell_1(x).\mathbf{0}$$

cannot reduce because of label mismatch. So, if we would extend subtyping to relate the types above, we would break the soundness of the subtyping (as there is some context where a corresponding process substitution causes an error (stuck)).

6. Denotational preciseness

In λ-calculus types are usually interpreted as subsets of the domains of λ-models whereas subtyping is interpreted as set-theoretic inclusion [4,30]. *Denotational preciseness* of subtyping is then:

$$\mathbb{T} \leqslant \mathbb{T}' \text{ if and only if } [\![\mathbb{T}]\!] \subseteq [\![\mathbb{T}']\!]$$

using $[\![\ ]\!]$ to denote type interpretation.

In the present context, let us interpret a session type $\mathbb{T}$ as the set of closed processes typed by $\mathcal{T}(\mathbb{T})$, i.e.

$$[\![\mathbb{T}]\!] = \{P \mid \vdash P : \mathcal{T}(\mathbb{T})\}$$

We show that the subtyping is denotationally precise. The denotational soundness follows from the subsumption rule [T-SUB] and the interpretation of subtyping as set-theoretic inclusion. In order to prove that denotational completeness holds, we show a more general result that operational completeness implies denotational completeness, along the lines of Dezani-Ciancaglini et al. [23].

Theorem 6.1. *The existence of characteristic terms implies denotational completeness.*

Proof. Denotational completeness follows from the following key property of characteristic processes:

$$\vdash \mathcal{P}(\mathbb{T}) : \mathcal{T}(\mathbb{T}') \text{ implies } \mathcal{T}(\mathbb{T}) \leqslant \mathcal{T}(\mathbb{T}')$$

If we could derive $\vdash \mathcal{P}(\mathbb{T}) : \mathcal{T}(\mathbb{T}')$ with $\mathcal{T}(\mathbb{T}) \not\leqslant \mathcal{T}(\mathbb{T}')$, then the multiparty session

$$\mathsf{r} \triangleleft \mathcal{P}(\mathbb{T}) \mid \prod_{1 \leq i \leq n} \mathsf{p}_i \triangleleft \mathcal{P}(\mathbb{T}_i)$$

where $\mathtt{pt}\{\mathbb{T}'\} = \{\mathsf{p}_i\}_{1 \leq i \leq n}$ and $\mathbb{G} = \mathcal{G}(\mathbb{T}', \mathsf{r})$ and $\mathbb{T}_i = \mathbb{G}\upharpoonright \mathsf{p}_i$ for $1 \leq i \leq n$, could be typed. Theorem 4.19 shows that this process is stuck, and this contradicts the soundness of the type system. We get the desired property, which implies denotational completeness, since if $\mathcal{T}(\mathbb{T}) \not\leqslant \mathcal{T}(\mathbb{T}')$ then $\mathcal{P}(\mathbb{T}) \in [\![\mathbb{T}]\!]$, but $\mathcal{P}(\mathbb{T}) \notin [\![\mathbb{T}']\!]$. □

As a consequence, we obtain the denotational preciseness of subtyping.

Theorem 6.2 *(Denotational preciseness). The subtyping relation is denotationally precise.*

Note that the two notions of preciseness, denotational and operational, are not equivalent. The usual subtyping in λ-calculus with intersection types (resp. intersection and union types) is known to be denotationally precise [4] (resp. [59]). On the other hand, operational completeness requires that all empty types, i.e., not inhabited types, are less than all inhabited types. This makes unfeasible an operationally complete subtyping for the pure λ-calculus with intersection and union types, since inhabitation is undecidable for intersection types and union types [21].

7. Conclusion and related work

The preciseness result of this paper shows that multiparty session subtyping $\leqslant$ (Definition 3.15) is a canonical notion of subtyping for the synchronous calculus presented in Section 2: it is both operationally sound (Theorem 3.21) and complete (Theorem 4.18), hence *operationally precise*. We also show that $\leqslant$ is *denotationally precise* (Section 6).

This relation $\leqslant$ is incorporated into most implementations of multiparty session types [36,37,50–52,56,58]; most implementations are based on Scribble [61], which is a protocol description language based on multiparty session types [34,35].

The main technical contribution of this paper is the definition of characteristic global types in Section 4. Given a session type $\mathbb{T}$ and a session participant p which does not occur in $\mathbb{T}$, the associated characteristic global type expresses the communications prescribed by $\mathbb{T}$ between p and the participants in $\mathbb{T}$. After each communication involving p, the characteristic global type creates a cyclic communication between all participants in $\mathbb{T}$. Such cyclic communications are essential to project the characteristic global type and to generate a deadlock when the subtyping relation is extended, as illustrated in Examples 4.12 and 4.13.

The subtyping considered here is sound but not complete for asynchronous multiparty session π-calculus in [34,35].

We conjecture the completeness of the subtyping defined in [49] for asynchronous multiparty sessions. We are currently working toward the preciseness for asynchronous multiparty sessions, including delegations (i.e. channel passing).

Related work

Preciseness of subtyping. Ligatti et al. [42] first define operational preciseness of subtyping and apply it to iso-recursive types. They consider a typed λ-calculus enriched with naturals, reals, pair and case constructors/destructors, and roll/unroll. The predicate bad(M) holds when M reduces to a stuck term, i.e. to an irreducible term which is not a value. They propose new algorithmic rules for subtyping iso-recursive types and show that they are operationally precise.

Dezani-Ciancaglini and Ghilezan [21] adapt the ideas of Ligatti et al. [42] to the setting of the concurrent λ-calculus with intersection and union types by Dezani-Ciancaglini et al. [20]. For the operational preciseness, they take the view that evaluation of well-typed terms always terminates. This means that the predicate bad coincides with non-termination. In this calculus, applicative contexts are enough. Notably, soundness and completeness are made more operational by asking that some applications converge instead of being typable. To sum up, the definition of operational preciseness becomes:

A subtyping $\leq$ is operationally precise when $\sigma \leq \tau$ if and only if there are no closed terms M, N such that:
(i) ML converges for all closed terms L of type τ, and
(ii) N has type σ, and
(iii) MN diverges.

The main result in [21] is the operational preciseness of the subtyping induced by the standard set theoretic interpretation of arrow, intersection and union types.

Chen et al. [14,15] first give a general formulation of preciseness for session calculi, where processes are typed by sets of pairs (channels, session types) as in Honda et al. [32]. The session types prescribe how the channels can be used for communications. The calculus of processes includes an error process and bad(P) holds when process P reduces to error. The typing judgements for closed processes are of the form $\vdash P \triangleright \{a : T\}$, assuring that the process P has a single free channel a whose type is T. The judgement $\vdash C[a : T] \triangleright \emptyset$ means that filling the hole of C with any process P typed by $a : T$ produces a well-typed closed process. They obtain:

A subtyping $\leq$ is precise when, for all session types T and S:

$$T \leqslant S \iff \begin{pmatrix}\text{there do not exist } C \text{ and } P \text{ such that:}\\ \vdash C[a : S] \triangleright \emptyset \text{ and } \vdash P \triangleright \{a : T\} \text{ and } C[P] \longrightarrow^* \text{error}\end{pmatrix}$$

When the only-if direction ($\Longrightarrow$) of this formula holds, we say that the subtyping is sound; when the if direction ($\Longleftarrow$) holds, we say that the subtyping is complete. The first result by Chen et al. [14,15] is that the well-known session subtyping, the branching-selection subtyping by Demangeon and Honda [18], is sound and complete for the synchronous dyadic calculus. Next, the authors show that in the asynchronous calculus, this subtyping is incomplete for type-safety: that is, there exist session types T and S such that T can safely be considered as a subtype of S, but $T \leq S$ is not derivable by the subtyping. They propose an asynchronous subtyping system (inspired by Mostrous et al. [49]) which is sound and complete for the asynchronous dyadic calculus. The method gives a general guidance to design rigorous channel-based subtypings respecting desired safety properties. We conjecture the preciseness of the subtyping of Demangeon and Honda [18] also for the full synchronous multiparty calculus, including multiple interleaved sessions — but we could not use the present approach for the proof, since well-typed interleaved sessions can be stuck [17].

(Un-)Decidability of session subtyping. A faithful implementation of the multiparty session subtyping algorithm in Table 6 (with added support for higher-order types) can be found in a software artifact by Scalas and Yoshida [58]. Table 6 is based on the subtyping algorithm for binary session types introduced by Gay and Hole [25] — who, in turn, adopt techniques inspired by Pierce and Sangiorgi [54]. Chen et al. [14,15] introduce a precise subtyping relation for *asynchronous* binary sessions, where messages can be buffered in FIFO queues; later, Lange and Yoshida [41] and Bravetti et al. [8,9] show that such a relation is undecidable, and study some decidable fragments. As observed in Section 4.4 of the paper by Bravetti et al. [8], these results imply that if Definition 3.15 is similarly extended to asynchronous subtyping, then the relation becomes undecidable.

Semantic subtyping. A framework which is closely related to the above described works is semantic subtyping. In semantic subtyping, each type is interpreted as the set of values having that type and subtyping is subset inclusion between type interpretations [12]. This gives a precise subtyping as soon as the calculus allows to distinguish operationally values of different types.

Semantic subtyping was first proposed by Castagna and Benzaken through the development of the CDuce project [24]. CDuce is a modern XML-oriented functional language. Distinctive features of CDuce are a powerful pattern matching, first class functions, over-loaded functions, a very rich type system (with arrow, sequence, pair, record, intersection, union, difference type constructs), precise type inference for patterns and error localisation, and a natural interpretation of types as sets of values. It is enriched also with some important implementation aspects: in particular, a dispatch algorithm that demonstrates how static type information can be used to obtain very efficient compilation schemas.

Semantic subtyping has been also studied in [10] for a π-calculus with a patterned input and in [11] for a session calculus with internal and external choices and typed input. Types are built using a rich set of type constructors including union, intersection and negation: they extend IO-types in [10] and session types in [11]. Semantic subtyping is precise for the calculi of [10,11,24], thanks to the type case constructor in [24], and to the blocking of inputs for values of "wrong" types in [10,11].

Session types and contracts. Various works study the relationship between *binary* session types and *contracts* [13]: they assign contract-based semantics to types (e.g., via an encoding), and study the correspondence between the syntactically-defined relations in the literature of session types (e.g., duality and subtyping) and the semantically-defined relations in the literature of contracts: see, e.g., Barbanera and de'Liguoro [2,3], Bartoletti et al. [5], Bernardi and Hennessy [7]. In particular, Theorem 4.10 of Bernardi and Hennessy [7] shows that binary session subtyping is fully abstract w.r.t. a behavioural pre-order on contracts, based on *peer compliance*. This can be related to our results on multiparty sessions: we investigate the canonicity of the syntactically-defined multiparty session subtyping, by reasoning on the semantics of typed (characteristic) processes, and the correctness of their interactions. In a recent work, Bernardi and Francalanza [6] provide a novel conductive characterisation of a preorder that relates client contracts, and argue its decidability; notably, client contracts feature mixed choices and non-determinism (both absent in session types). As future work, it would be interesting to integrate this rich language in multiparty session types, develop a corresponding subtyping relation, and study its canonicity (i.e., preciseness) with the approach presented in the present paper.

Acknowledgements

We would like to thank Mariangiola Dezani for her contribution to the workshop version of this paper [22], and the anonymous reviewers for their detailed comments and suggestions. This work has been partially sponsored by EPSRC EP/K011715/1, EP/K034413/1, EP/L00058X/1, EP/N027833/1 and EP/N028201/1, MPNTR ON174026, III044006, COST Action EUTYPES (CA15123), ICT COST Action BETTY (IC1201) and COST Action ARVI (IC1402), and European Horizon 2020 project COEMS under number 732016.

Appendix A. Additional definitions and proofs for Section 3

In this section, we reprise the global/local type trees introduced in Section 3 with a more detailed treatment, based on [53, Part IV]: this is developed in Appendix A.1 (for global types) and Appendix A.2 (for session types), thus giving a more formal basis to the graphical type tree representations in Figs. 1 and 3, and to Notation 3.2 and 3.5.

Then, we study the properties of type projection (Appendix A.3) and merging (Appendix A.4), substitution/congruence (Appendix A.5). We continue with the properties of the type system (Appendix A.6), and the proofs of Lemma 3.19 (Appendix A.7), subject reduction and progress (Appendix A.8), and termination of algorithmic subtyping (Appendix A.9).

Before starting, we observe that since we will define global and session type trees as partial functions (cf. Definitions A.4 and A.13), we can consider them equal under the standard equality between functions, formalised in Definition A.1 below.

Definition A.1 *(Extensional function equality).* Given two (partial) functions f and f', we write $f = f'$ (read: *f and f' are equal*) iff (1) $\mathrm{dom}(f) = \mathrm{dom}(f')$, and (2) $\forall x \in \mathrm{dom}(f) : f(x) = f'(x)$.

Proposition A.2. *The relation "$=$" between (partial) functions is symmetric, reflexive and transitive.*

A.1. Global type trees

Definition A.3 *(Payload/continuation labels and sequences).* From the set of labels in Notation 2.1, we define the sets of *continuation labels* $\{\ell^{\mathsf{C}} \mid \ell \in \text{labels}\}$, and *payload labels* $\{\ell^{\mathsf{P}} \mid \ell \in \text{labels}\}$. We write w to indicate a sequence of *continuation* labels, ϵ for the empty sequence, and $w{\cdot}\ell^{\mathsf{C}}$ for the chaining of w and ℓ^{C}. Further, we write $w{\cdot}\ell^{\mathsf{P}}$ for the sequence obtained by chaining w (consisting of continuation labels, only) with *just one payload* label ℓ^{P}.

Definition A.4 *(Global type trees).* Let f be a partial function from sequences w or $w{\cdot}\ell^{\mathsf{P}}$ (Definition A.3) to the set $\{\mathsf{p} \to \mathsf{q}, \mathtt{end}, S \mid \mathsf{p}, \mathsf{q} \in \text{roles}, S \in \text{sorts}\}$. We say that f is a *global type tree* if it satisfies the following constraints:

1. $f(\epsilon)$ is defined as either $\mathtt{end}$ or $\mathsf{p} \to \mathsf{q}$ (for some p, q);
2. if $f(w) = \mathtt{end}$, then for all ℓ, $f(w{\cdot}\ell^{\mathsf{C}})$ and $f(w{\cdot}\ell^{\mathsf{P}})$ are *un*defined;
3. if $f(w) = \mathsf{p} \to \mathsf{q}$ (for some p, q), then there is a finite set of labels $L \neq \emptyset$ such that, for all $\ell \in L$, both $f(w{\cdot}\ell^{\mathsf{C}})$ and $f(w{\cdot}\ell^{\mathsf{P}})$ are defined;
4. if $f(w{\cdot}\ell^{\mathsf{C}})$ is defined, then it is either $\mathsf{p} \to \mathsf{q}$ (for some p, q) or $\mathtt{end}$, and $f(w)$ is defined;
5. if $f(w{\cdot}\ell^{\mathsf{P}})$ is defined, then it is a sort S, and $f(w)$ is defined.

A partial function f satisfying Definition A.4 can be visually represented as a tree, as shown in Figs. 1 and 2: edges are marked by continuation/payload labels, and nodes are marked by either $\mathsf{p} \to \mathsf{q}$, $\mathtt{end}$, or S (for some $\mathsf{p}, \mathsf{q}, S$); a path along the tree is defined as a sequence w or $w{\cdot}\ell^{\mathsf{P}}$ for which f is defined.

Definition A.5. The function $\mathcal{T}$, from closed global types to type trees, is inductively defined as:

$$\begin{array}{rll}
\mathcal{T}(\mathtt{end})(\epsilon) & = \mathtt{end} & \\
\mathcal{T}(\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathbb{G}_i\}_{i\in I})(\epsilon) & = \mathsf{p} \to \mathsf{q} & \\
\mathcal{T}(\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathbb{G}_i\}_{i\in I})(\ell_k^{\mathsf{C}}{\cdot}w) & = \mathcal{T}(\mathbb{G}_k)(w) & \text{for all } k \in I \\
\mathcal{T}(\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathbb{G}_i\}_{i\in I})(\ell_k^{\mathsf{P}}) & = S_k & \text{for all } k \in I \\
\mathcal{T}(\mu\mathbf{t}.\mathbb{G})(w) & = \mathcal{T}(\mathbb{G}\{\mu\mathbf{t}.\mathbb{G}/\mathbf{t}\})(w) &
\end{array}$$

We define the *tree of* $\mathbb{G}$ as $\mathcal{T}(\mathbb{G})$.

We can now observe that the graphical tree constructed by $\mathcal{T}(\mathbb{G})$ in Fig. 1 corresponds to the tree function yielded by $\mathcal{T}(\mathbb{G})$ in Definition A.5:

- a sequence of continuation labels w can be followed along the labelled edges of $\mathcal{T}(\mathbb{G})$ in Fig. 1, reaching a node with marking m (of the form $\mathsf{p} \to \mathsf{q}$ or $\mathtt{end}$), *if and only if* in Definition A.5 we have $\mathcal{T}(\mathbb{G})(w) = m$;
- similarly, a sequence $w{\cdot}\ell^{\mathsf{P}}$ can be followed along the edges of $\mathcal{T}(\mathbb{G})$ in Fig. 1, reaching a leaf node marked with sort S, *if and only if* in Definition A.5 we have $\mathcal{T}(\mathbb{G})(w{\cdot}\ell^{\mathsf{P}}) = S$.

Notation A.6 *(Global types as trees).* As in [53, p. 285], we save some notation by writing:

$$\mathtt{end} \quad \text{for} \quad \text{the type tree } f \text{ with } f(\epsilon) = \mathtt{end}$$

$$\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).g_i\}_{i\in I} \quad \begin{array}{l}\text{where } g_i \text{ is a global type tree } (\forall i{\in}I) \\ \text{and } \forall i, j \in I : i \neq j \text{ implies } \ell_i \neq \ell_j\end{array} \quad \text{for} \quad \begin{array}{l}\text{the type tree } f \text{ with:} \\ \text{1. (1) } f(\epsilon) = \mathsf{p} \to \mathsf{q}, \text{ and} \\ \text{2. (2) } \forall i{\in}I,\ f(\ell_i^{\mathsf{P}}){=}S_i \text{ and } f(\ell_i^{\mathsf{C}}{\cdot}w) = g_i(w)\end{array}$$

We will also write G as a shorthand for a tree $\mathcal{T}(\mathbb{G})$, for some $\mathbb{G}$ (cf. Notation 3.2).

In practice, two global type trees are equal when they are both $\mathtt{end}$, or denote a communication between a same pair of participants, with the same labels, and equal continuations. This is detailed in Definition A.7 and Lemma A.11 below.

Definition A.7 *(Global type tree isomorphism).* We say that a relation $\mathcal{R}$ between global type trees is a *global type tree isomorphism* iff it satisfies the following clauses, whenever $(f, f') \in \mathcal{R}$:

(i) if f is $\mathtt{end}$, then f' is $\mathtt{end}$;
(ii) if f is $\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).g_i\}_{i\in I}$ (for some $\mathsf{p}, \mathsf{q}, I, \ell_i, S_i, g_i$) then f' is $\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).g_i'\}_{i\in I}$ such that $\forall i \in I : (g, g_i') \in \mathcal{R}$.

We write $f \sim f'$ (read "*f and f' are isomorphic*") iff $f\ \mathcal{R}\ f'$ for some global type tree isomorphism $\mathcal{R}$. Given two global types $\mathbb{G}$ and $\mathbb{G}'$, we write $\mathbb{G} \sim \mathbb{G}'$ iff $\mathcal{T}(\mathbb{G}) \sim \mathcal{T}(\mathbb{G}')$.

Proposition A.8. *Assume $\mathsf{G} \sim \mathsf{G}'$, and take any sequence of continuation labels w such that $\mathsf{G}(w)$ is defined or $\mathsf{G}'(w)$ is defined. Then, $\mathsf{G}(w) = \mathsf{G}'(w)$.*

Proof. We prove a stronger statement (formalised below): intuitively, we show that when two global type trees G and G' are isomorphic (Definition A.7), and one of them is defined on a sequence of continuation labels w, then each label in the sequence w connects the pair $(\mathsf{G}, \mathsf{G}')$ to a pair of "continuation" type trees, that are also isomorphic.

Take any global type tree isomorphism $\mathcal{R}$, any G, G' such that $\mathsf{G}\ \mathcal{R}\ \mathsf{G}'$, and any sequence of continuation labels $w = \ell_1^{\mathsf{C}}{\cdot}\ell_2^{\mathsf{C}}{\cdot}\ldots{\cdot}\ell_n^{\mathsf{C}}$ such that $\mathsf{G}(w)$ is defined or $\mathsf{G}'(w)$ is defined. Letting:

$$\mathsf{G}_0 = \mathsf{G} \qquad \text{and} \qquad \mathsf{G}_0' = \mathsf{G}' \tag{1}$$

there is a sequence $(\mathsf{G}_0, \mathsf{G}_0'), (\mathsf{G}_1, \mathsf{G}_1'), \ldots, (\mathsf{G}_n, \mathsf{G}_n')$ such that:

(p1) $\forall i \in 0..n$: $(\mathsf{G}_i, \mathsf{G}_i') \in \mathcal{R}$;
(p2) $\mathsf{G}_n(\epsilon) = \mathsf{G}_n'(\epsilon)$;
(p3) $\forall i \in 0..n-1$: $\mathsf{G}_i = \mathsf{p} \to \mathsf{q} : \{\ell_k(S_k).\mathsf{G}_k\}_{k\in K}$, for some $\mathsf{p}, \mathsf{q}, K, \ell_k, S_k, \mathsf{G}_k$; moreover, for some $k \in K$, $\ell_k = \ell_i$ and $\mathsf{G}_k = \mathsf{G}_{i+1}$;
(p4) $\forall i \in 0..n-1$: $\mathsf{G}_i(\ell_{i+1}^{\mathsf{C}}{\cdot}\ell_{i+2}^{\mathsf{C}}{\cdot}\ldots{\cdot}\ell_n^{\mathsf{C}}) = \mathsf{G}_{i+1}(\ell_{i+2}^{\mathsf{C}}{\cdot}\ldots{\cdot}\ell_n^{\mathsf{C}})$ (where "$\ell_{i+2}^{\mathsf{C}}{\cdot}\ldots{\cdot}\ell_n^{\mathsf{C}}$" for $n = 1$ means ϵ);

(p5) $\forall i \in 0..n-1\colon\ \mathsf{G}'_i = \mathsf{p} \to \mathsf{q} : \{\ell_k(S_k).\mathsf{G}'_k\}_{k\in K}$, for some $\mathsf{p}, \mathsf{q}, K, \ell_k, S_k, \mathsf{G}'_k$; moreover, for some $k \in K$, $\ell_k = \ell_i$ and $\mathsf{G}'_k = \mathsf{G}'_{i+1}$;
(p6) $\forall i \in 0..n-1\colon\ \mathsf{G}'_i(\ell^{\mathsf{C}}_{i+1}\cdot\ell^{\mathsf{C}}_{i+2}\cdot\ldots\cdot\ell^{\mathsf{C}}_{n}) = \mathsf{G}'_{i+1}(\ell^{\mathsf{C}}_{i+2}\cdot\ldots\cdot\ell^{\mathsf{C}}_{n})$.

Assuming $\mathsf{G} = \mathsf{G}_0\ \mathcal{R}\ \mathsf{G}'_0 = \mathsf{G}'$, and assuming at least one between $\mathsf{G}(w)$ and $\mathsf{G}'(w)$ is defined, we proceed by induction on the length of w.

The base case $n = 0$ (i.e., $w = \epsilon$) is immediate: *(p1)* holds by hypothesis, *(p2)* holds by Definition A.7, and *(p3)–(p6)* hold vacuously.

In the inductive case $n = m + 1$, by the induction hypothesis, we have:

(ih1) $\forall i \in 0..m\colon\ (\mathsf{G}_i, \mathsf{G}'_i) \in \mathcal{R}$;
(ih2) $\mathsf{G}_m(\epsilon) = \mathsf{G}'_m(\epsilon)$;
(ih3) $\forall i \in 0..m-1\colon\ \mathsf{G}_i = \mathsf{p} \to \mathsf{q} : \{\ell_k(S_k).\mathsf{G}_k\}_{k\in K}$ (for some $\mathsf{p}, \mathsf{q}, K, \ell_k, S_k, \mathsf{G}_k$); moreover, for some $k \in K$, $\ell_k = \ell_i$ and $\mathsf{G}_k = \mathsf{G}_{i+1}$;
(ih4) $\forall i \in 0..m-1\colon\ \mathsf{G}_i(\ell^{\mathsf{C}}_{i+1}\cdot\ell^{\mathsf{C}}_{i+2}\cdot\ldots\cdot\ell^{\mathsf{C}}_{m}) = \mathsf{G}_{i+1}(\ell^{\mathsf{C}}_{i+2}\cdot\ldots\cdot\ell^{\mathsf{C}}_{m})$;
(ih5) $\forall i \in 0..m-1\colon\ \mathsf{G}'_i = \mathsf{p} \to \mathsf{q} : \{\ell_k(S_k).\mathsf{G}'_k\}_{k\in K}$ (for some $\mathsf{p}, \mathsf{q}, K, \ell_k, S_k, \mathsf{G}'_k$); moreover, for some $k \in K$, $\ell_k = \ell_i$ and $\mathsf{G}'_k = \mathsf{G}'_{i+1}$;
(ih6) $\forall i \in 0..m-1\colon\ \mathsf{G}'_i(\ell^{\mathsf{C}}_{i+1}\cdot\ell^{\mathsf{C}}_{i+2}\cdot\ldots\cdot\ell^{\mathsf{C}}_{m}) = \mathsf{G}'_{i+1}(\ell^{\mathsf{C}}_{i+2}\cdot\ldots\cdot\ell^{\mathsf{C}}_{m})$.

Then, we observe that since $(\mathsf{G}_m, \mathsf{G}'_m) \in \mathcal{R}$, the pair must satisfy clauses *(i)* and *(ii)* of Definition A.7. Thus, have the following cases:

- the pair $(\mathsf{G}_m, \mathsf{G}'_m)$ satisfies clause *(i)* non-vacuously — i.e., both G_m and G'_m are `end`. This case is impossible. In fact, if we admit it, we have that

$$\text{both } \mathsf{G}_m(\ell^{\mathsf{C}}_n) \text{ and } \mathsf{G}'_m(\ell^{\mathsf{C}}_n) \text{ are undefined} \qquad \text{(by Notation A.6 and Definition A.4)} \tag{2}$$

  From (2), we can prove that:

$$\forall i \in 0..m,\ \text{both } \mathsf{G}_i(\ell^{\mathsf{C}}_{i+1}\cdot\ell^{\mathsf{C}}_{i+2}\cdot\ldots\cdot\ell^{\mathsf{C}}_{n}) \text{ and } \mathsf{G}'_i(\ell^{\mathsf{C}}_{i+1}\cdot\ell^{\mathsf{C}}_{i+2}\cdot\ldots\cdot\ell^{\mathsf{C}}_{n}) \text{ are undefined} \tag{3}$$

  The proof of (3) is by induction on i, in the decreasing range $m..0$:
  - in the base case $i = m$, both $\mathsf{G}_i(\ell^{\mathsf{C}}_n)$ and $\mathsf{G}'_i(\ell^{\mathsf{C}}_n)$ are undefined, by (2);
  - in the inductive case $i = m - j$ (for $j \in 1..m$), by the induction hypothesis on (3), we know that $\mathsf{G}_{i+1}(\ell^{\mathsf{C}}_{i+2}\cdot\ell^{\mathsf{C}}_{i+3}\cdot\ldots\cdot\ell^{\mathsf{C}}_{n})$ and $\mathsf{G}'_{i+1}(\ell^{\mathsf{C}}_{i+2}\cdot\ell^{\mathsf{C}}_{i+3}\cdot\ldots\cdot\ell^{\mathsf{C}}_{n})$ are undefined. Therefore, by *(ih3)* and *(ih5)*, and by Notation A.6 and Definition A.5, also $\mathsf{G}_i(\ell^{\mathsf{C}}_{i+1}\cdot\ell^{\mathsf{C}}_{i+2}\cdot\ldots\cdot\ell^{\mathsf{C}}_{n})$ and $\mathsf{G}'_i(\ell^{\mathsf{C}}_{i+1}\cdot\ell^{\mathsf{C}}_{i+2}\cdot\ldots\cdot\ell^{\mathsf{C}}_{n})$ are undefined.

  Thus, by admitting that the pair $(\mathsf{G}_m, \mathsf{G}'_m)$ satisfies clause *(i)* non-vacuously, we have proved (3); however, this implies that both $\mathsf{G}(w) = \mathsf{G}_0(w)$ and $\mathsf{G}'(w) = \mathsf{G}'_0(w)$ are undefined — contradiction. Therefore, neither G_m nor G'_m is `end`;
- the pair $(\mathsf{G}_m, \mathsf{G}'_m)$ satisfies clause *(ii)* non-vacuously. Then, we have two sub-cases:
  - ℓ_n does not belong to the labels in the existential quantification (*"for some…"*) of clause *(ii)*. This case is impossible. In fact, if we admit it, then $\mathsf{G}_m(\ell^{\mathsf{C}}_n)$ and $\mathsf{G}'_m(\ell^{\mathsf{C}}_n)$ are undefined, and we obtain the same contradiction stemming from (2) above;
  - ℓ_n belongs to the labels in the existential quantification (*"for some…"*) of clause *(ii)*. In this case, by clause *(ii)* of Definition A.7,

$$\exists \mathsf{p}, \mathsf{q}, K :$$
$$\mathsf{G}_m = \mathsf{p} \to \mathsf{q} : \{\ell_k(S_k).\mathsf{G}_k\}_{k\in K} \tag{4}$$
$$\mathsf{G}'_m = \mathsf{p} \to \mathsf{q} : \{\ell_k(S_k).\mathsf{G}'_k\}_{k\in K} \tag{5}$$
$$\exists k \in K : \ell_k = \ell_n \text{ and } (\mathsf{G}_k, \mathsf{G}'_k) \in \mathcal{R} \tag{6}$$

  Therefore, letting:

$$(\mathsf{G}_n, \mathsf{G}'_n) = (\mathsf{G}_k, \mathsf{G}'_k) \text{ from (6)} \tag{7}$$

  we obtain a sequence $(\mathsf{G}_0, \mathsf{G}'_0), (\mathsf{G}_1, \mathsf{G}'_1), \ldots, (\mathsf{G}_n, \mathsf{G}'_n)$ which satisfies *(p1)–(p6)*, as follows:
  - *(p1)* holds by *(ih1)*, (7) and (6);
  - *(p2)* holds by (4) and (5);
  - *(p3)* is proved by cases on $i \in 1..n-1$ (reminding that $n-1 = m$):
    - if $i = m - j$ (for some $j \in 1..m$), the result follows by *(ih3)* and Notation A.6;
    - if $i = m = n - 1$, observe that $i + 1 = n$: the result follows by (7), (6) and (4);
  - *(p4)* is proved by taking any $i \in 1..n-1$, and G_i in item *(p3)*: the result follows by Notation A.6 and Definition A.5;

∗ *(p5)* is proved similarly to *(p3)*, using (5) instead of (4);
∗ *(p6)* is proved similarly to *(p4)*, using (5) instead of (4).

We have thus proved *(p1)–(p6)*. To conclude, notice that:

$$\forall i \in 0..n:\ \mathsf{G}_i(\ell_{i+1}^{\mathsf{C}} \cdot \ldots \cdot \ell_n^{\mathsf{C}}) = \mathsf{G}'_i(\ell_{i+1}^{\mathsf{C}} \cdot \ldots \cdot \ell_n^{\mathsf{C}}) \qquad \text{(by p2, p4 and p6)} \tag{8}$$

$$\mathsf{G}(w) = \mathsf{G}_0(w) = \mathsf{G}'_0(w) = \mathsf{G}'(w) \qquad \text{(by (8) (when } i = 0) \text{ and (1))}$$

which is the main thesis. □

Proposition A.9. *Assume* $\mathsf{G} \sim \mathsf{G}'$*, and take any sequence of continuation labels* w*, and any label* ℓ*, such that* $\mathsf{G}(w \cdot \ell^{\mathsf{P}})$ *is defined or* $\mathsf{G}'(w \cdot \ell^{\mathsf{P}})$ *is defined. Then,* $\mathsf{G}(w \cdot \ell^{\mathsf{P}}) = \mathsf{G}'(w \cdot \ell^{\mathsf{P}})$.

Proof. Similar to the proof of Proposition A.8, but considering the equality of the payload sorts pointed by ℓ^{P}, when checking the clauses of Definition A.7. □

Proposition A.10. *If* $\mathsf{G} \sim \mathsf{G}'$*, then:*

1. $\mathrm{dom}(\mathsf{G}) = \mathrm{dom}(\mathsf{G}')$*;*
2. *for all* $x \in \mathrm{dom}(\mathsf{G})$, $\mathsf{G}(x) = \mathsf{G}'(x)$.

Proof. Item 1 is direct consequence of Proposition A.8 and Proposition A.9: they imply that when $\mathsf{G}(w)$ (resp. $\mathsf{G}(w \cdot \ell^{\mathsf{P}})$), is defined, then also $\mathsf{G}'(w)$ (resp. $\mathsf{G}'(w \cdot \ell^{\mathsf{P}})$), is defined, and *vice versa*.

Item 2 follows by item 1, Proposition A.8 and Proposition A.9. □

Lemma A.11. *For all* G *and* G', $\mathsf{G} = \mathsf{G}'$ *if and only if* $\mathsf{G} \sim \mathsf{G}'$.

Proof. ($\Longrightarrow$) Consider the following relation:

$$\mathcal{R} = \left\{ (\mathsf{G}, \mathsf{G}') \;\middle|\; \mathsf{G} = \mathsf{G}' \right\}$$

Inspecting each pair $(\mathsf{G}, \mathsf{G}') \in \mathcal{R}$, by Definition A.1 we have $\mathrm{dom}(\mathsf{G}) = \mathrm{dom}(\mathsf{G}')$; moreover, for all $x \in \mathrm{dom}(\mathsf{G})$, $\mathsf{G}(x) = \mathsf{G}'(x)$. Thus, by Notation A.6, Definition A.5 and Definition A.4, we have two possibilities:

(a) both G and G' are $\mathtt{end}$; or
(b) G is $\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i \in I}$ (for some $\mathsf{p}, \mathsf{q}, I, \ell_i, S_i, \mathsf{G}_i$) and G' is $\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}'_i\}_{i \in I}$ (for some G'_i), with $\mathsf{G}_i = \mathsf{G}'_i$ (for all $i \in I$) — which means $(\mathsf{G}_i, \mathsf{G}'_i) \in \mathcal{R}$ (for all $i \in I$).

Therefore, $\mathcal{R}$ satisfies the clauses of Definition A.7, i.e., $\mathcal{R}$ is a global type tree isomorphism — hence, we conclude $\mathsf{G} \sim \mathsf{G}'$.

($\Longleftarrow$) Assume $\mathsf{G} \sim \mathsf{G}'$: the result is direct consequence of Proposition A.10 and Definition A.1. □

Remark A.12. We will take advantage of Lemma A.11 to use $=$ (Definition A.1) and $\sim$ (Definition A.7) interchangeably. I.e., when considering two equal type trees $\mathsf{G} = \mathsf{G}'$, we will sometimes implicitly leverage Lemma A.11, to reason on how the pair $(\mathsf{G}, \mathsf{G}')$ satisfies the clauses of Definition A.7. Symmetrically, we can show $\mathsf{G} = \mathsf{G}'$ by showing that $\mathsf{G} \sim \mathsf{G}'$ holds, and implicitly leveraging Lemma A.11.

A.2. Session type trees

Definition A.13 *(Session type trees).* Let f be a partial function from sequences w or $w \cdot \ell^{\mathsf{P}}$ (Definition A.3) to the set $\{\wedge\mathsf{p}?, \vee\mathsf{p}!, \mathtt{end}, S \mid \mathsf{p} \in \text{roles}, S \in \text{sorts}\}$. We say that f is a *session type tree* if it satisfies the following constraints:

1. $f(\epsilon)$ is defined as either $\mathtt{end}$, $\wedge\mathsf{p}?$, or $\vee\mathsf{p}!$ (for some p);
2. if $f(w) = \mathtt{end}$, then for all ℓ, $f(w \cdot \ell^{\mathsf{C}})$ and $f(w \cdot \ell^{\mathsf{P}})$ are *un*defined;
3. if $f(w) = \wedge\mathsf{p}?$ or $f(w) = \vee\mathsf{p}!$ (for some p), then there is a finite set of labels $L \neq \emptyset$ such that, for all $\ell \in L$, both $f(w \cdot \ell^{\mathsf{C}})$ and $f(w \cdot \ell^{\mathsf{P}})$ are defined;
4. if $f(w \cdot \ell^{\mathsf{C}})$ is defined, then it is either $\mathtt{end}$, $\wedge\mathsf{p}?$, or $\vee\mathsf{p}!$ (for some p), and $f(w)$ is defined;
5. if $f(w \cdot \ell^{\mathsf{P}})$ is defined, then it is a sort S, and $f(w)$ is defined.

A function f satisfying Definition A.13 can be visually represented as a tree, as shown in Fig. 3: edges are marked by continuation/payload labels, and nodes are marked by either $\wedge\mathsf{p}?$, $\vee\mathsf{p}!$, $\mathtt{end}$, or S (for some p, S); a path along the tree is defined as a sequence w or $w \cdot \ell^{\mathsf{P}}$ for which f is defined.

Definition A.14. The function $\mathcal{T}$, from closed session types to type trees, is inductively defined as:

$$\begin{array}{rcll} & & \mathcal{T}(\mathtt{end})(\epsilon) & = \mathtt{end} & \\ & & \mathcal{T}\big(\bigwedge_{i\in I}\mathsf{p}?\ell_i(S_i).\mathbb{T}_i\big)(\epsilon) & = \wedge\mathsf{p}? & \\ & & \mathcal{T}\big(\bigvee_{i\in I}\mathsf{p}!\ell_i(S_i).\mathbb{T}_i\big)(\epsilon) & = \vee\mathsf{p}! & \\ \mathcal{T}\big(\bigwedge_{i\in I}\mathsf{p}?\ell_i(S_i).\mathbb{T}_i\big)(\ell_k^{\mathsf{C}}\cdot w) & = & \mathcal{T}\big(\bigvee_{i\in I}\mathsf{p}!\ell_i(S_i).\mathbb{T}_i\big)(\ell_k^{\mathsf{C}}\cdot w) & = \mathcal{T}(\mathbb{T}_k)(w) & \text{for all } k\in I \\ \mathcal{T}\big(\bigwedge_{i\in I}\mathsf{p}?\ell_i(S_i).\mathbb{T}_i\big)(\ell_k^{\mathsf{P}}) & = & \mathcal{T}\big(\bigvee_{i\in I}\mathsf{p}!\ell_i(S_i).\mathbb{T}_i\big)(\ell_k^{\mathsf{P}}) & = S_k & \text{for all } k\in I \\ & & \mathcal{T}(\mu\mathbf{t}.\mathbb{T})(w) & = \mathcal{T}(\mathsf{T}\{\mu\mathbf{t}.\mathbb{T}/\mathbf{t}\})(w) & \end{array}$$

We define the *tree of* $\mathbb{T}$ as $\mathcal{T}(\mathbb{T})$.

We can now observe that the graphical tree constructed by $\mathcal{T}(\mathbb{T})$ in Fig. 3 corresponds to the tree function yielded by $\mathcal{T}(\mathbb{T})$ in Definition A.14:

- a sequence of continuation labels w can be followed along the labelled edges of $\mathcal{T}(\mathbb{T})$ in Fig. 3, reaching a node with marking m (of the form $\wedge\mathsf{p}?$, $\vee\mathsf{p}!$, or $\mathtt{end}$), *if and only if* in Definition A.14 we have $\mathcal{T}(\mathbb{T})(w) = m$;
- similarly, a sequence $w\cdot\ell^{\mathsf{P}}$ can be followed along the edges of $\mathcal{T}(\mathbb{T})$ in Fig. 3, reaching a leaf node marked with sort S, *if and only if* in Definition A.14 we have $\mathcal{T}(\mathbb{T})(w\cdot\ell^{\mathsf{P}}) = S$.

Notation A.15 *(Session types as trees).* Similarly to [53, p. 285], we will save some notation by writing:

$\mathtt{end}$ for the type tree f with $f(\epsilon) = \mathtt{end}$

$\bigwedge_{i\in I}\mathsf{p}?\ell_i(S_i).t_i$ where t_i is a session type tree $(\forall i\in I)$ and $\forall i, j\in I : i\neq j$ implies $\ell_i\neq\ell_j$ for the type tree f with (1) $f(\epsilon) = \wedge\mathsf{p}?$, and (2) $\forall i\in I$, $f(\ell_i^{\mathsf{P}}) = S_i$ and $f(\ell_i^{\mathsf{C}}\cdot w) = t_i(w)$

$\bigvee_{i\in I}\mathsf{p}?\ell_i(S_i).t_i$ where t_i is a session type tree $(\forall i\in I)$ and $\forall i, j\in I : i\neq j$ implies $\ell_i\neq\ell_j$ for the type tree f with (1) $f(\epsilon) = \vee\mathsf{p}!$, and (2) $\forall i\in I$, $f(\ell_i^{\mathsf{P}}) = S_i$ and $f(\ell_i^{\mathsf{C}}\cdot w) = t_i(w)$

We will also write T as a shorthand for a tree $\mathcal{T}(\mathbb{T})$, for some $\mathbb{T}$ (cf. Notation 3.5).

Definition A.16 *(Session type isomorphism).* We say that a relation $\mathcal{R}$ between session type trees is a *session type tree isomorphism* iff it satisfies the following clauses, whenever $(f, f')\in\mathcal{R}$:

(i) if f is $\mathtt{end}$, then f' is $\mathtt{end}$;
(ii) if f is $\bigwedge_{i\in I}\mathsf{p}?\ell_i(S_i).t_i$ (for some $\mathsf{p}, I, \ell_i, S_i, t_i$), then f' is $\bigwedge_{i\in I}\mathsf{p}?\ell_i(S_i).t_i'$ such that $\forall i\in I : (t_i, t_i')\in\mathcal{R}$;
(iii) if f is $\bigvee_{i\in I}\mathsf{p}!\ell_i(S_i).t_i$ (for some $\mathsf{p}, I, \ell_i, S_i, t_i$), then f' is $\bigvee_{i\in I}\mathsf{p}!\ell_i(S_i).t_i'$ such that $\forall i\in I : (t_i, t_i')\in\mathcal{R}$.

We write $f\sim f'$ (read "*f and f' are isomorphic*") iff $f\mathcal{R}f'$ for some session type tree isomorphism $\mathcal{R}$. Given two session types $\mathbb{T}$ and $\mathbb{T}'$, we write $\mathbb{T}\sim\mathbb{T}'$ iff $\mathcal{T}(\mathbb{T})\sim\mathcal{T}(\mathbb{T}')$.

Lemma A.17. *For all* T *and* T', $\mathsf{T} = \mathsf{T}'$ *(Definition A.1) if and only if* $\mathsf{T}\sim\mathsf{T}'$.

Proof. Similar to Lemma A.11, adapting Proposition A.10, Proposition A.8 and Proposition A.9 to use session type trees as per Definition A.13 and Notation A.15. □

Remark A.18. We will take advantage of Lemma A.17 to use $=$ (Definition A.1) and $\sim$ (Definition A.16) interchangeably. I.e., when considering two equal type trees $\mathsf{T} = \mathsf{T}'$, we will sometimes implicitly leverage Lemma A.17, to reason on how the pair $(\mathsf{T}, \mathsf{T}')$ satisfies the clauses of Definition A.16. Symmetrically, we will demonstrate $\mathsf{T} = \mathsf{T}'$ by showing that $\mathsf{T}\sim\mathsf{T}'$ holds, and implicitly leveraging Lemma A.17.

A.3. Properties of global type projection

The proofs below leverage the notion of *global type tree context*, introduced in Definition A.19: i.e., a global type tree whose continuation branches always reach a hole after traversing a *finite* number of edges (see Fig. A.4): this leads to the inductive definition in Definition A.19 (the approach follows [53, p. 285], for finite type trees). Hence, a global type tree context $\mathfrak{G}$ can be traversed by structural induction.

Definition A.19. A *global type tree context* $\mathfrak{G}$ is a partial function f similar to Definition A.4, but only defined for sequences w with a maximum length k (for some k) and returning a hole $[\,]^i$ (where i is an index) instead of $\mathtt{end}$. Using Notation A.6, $\mathfrak{G}$ can be defined inductively, with the following productions:

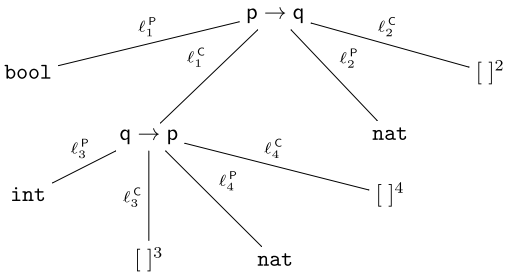

Fig. A.4. The type tree of the global type context: $\mathtt{p} \to \mathtt{q} : \Big\{ \ell_1(\mathtt{bool}).\, \mathtt{q} \to \mathtt{p} : \{\ell_3(\mathtt{int}).[\,]^3\,,\ \ell_4(\mathtt{nat}).[\,]^4\}\,,\ \ell_2(\mathtt{nat}).[\,]^2 \Big\}$.

$$\mathfrak{G} \ ::= \ \mathtt{p} \to \mathtt{q} : \{\ell_i(S_i).\mathfrak{G}_i\}_{i\in I} \ \mid \ [\,]^i$$

We write $\mathfrak{G}[\mathsf{G}_i]^i_{i\in I}$ to denote the global type tree obtained by grafting the tree G_i in the hole $[\,]^i$, for all $i \in I$, with the understanding that I indexes all the holes in $\mathfrak{G}$.

Proposition A.20. *Assume that* $\mathsf{G} \upharpoonright_{\mathtt{r}} \mathsf{T}$ *holds non-vacuously by clause* [PROJ-CONT] *of Definition 3.6. Then, there is* G' *such that* $\mathsf{G}' \upharpoonright_{\mathtt{r}} \mathsf{T}$ *holds non-vacuously by clause* [PROJ-END], [PROJ-IN], *or* [PROJ-OUT] *of Definition 3.6.*

Proof. Assume the hypothesis. By Definition 3.6, this means $\mathsf{G} = \mathtt{p} \to \mathtt{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}$ with $\mathtt{r} \notin \{\mathtt{p}, \mathtt{q}\}$. We have two possibilities:

- if $\mathtt{r} \notin \mathtt{pt}\{\mathsf{G}\}$, then by Definition 3.6 we must have $\mathsf{T} = \mathtt{end}$, and letting $\mathsf{G}' = \mathtt{end}$, we conclude that $\mathsf{G}' \upharpoonright_{\mathtt{r}} \mathsf{T}$ holds non-vacuously by clause [PROJ-END];
- if $\mathtt{r} \in \mathtt{pt}\{\mathsf{G}\}$, then by Definition 3.6 we have $\mathsf{T} = \bigsqcap_{i\in I} \mathsf{T}_i$ with $\mathsf{G}_i \upharpoonright_{\mathtt{r}} \mathsf{T}_i$. This means that, for all $i \in I$, there is $\mathfrak{G}_i$ such that $\mathtt{r} \notin \mathtt{pt}\{\mathfrak{G}_i\}$ and $\mathsf{G}_i = \mathfrak{G}_i[\mathsf{G}_j]^j_{j\in J_i}$, for some J_i; moreover, for some $\mathtt{p}$, either:
  (a) when $\mathsf{T} = \bigwedge_{l\in L} \mathtt{p}?\ell_l(S_l).\mathsf{T}'_l$, then for all $i \in I$ and $j \in J_i$, there are L_{ij} such that $L = \bigcup_{i\in I, j\in J} L_{ij}$ and $\mathsf{G}_j = \mathtt{p} \to \mathtt{r} : \{\ell_l(S_l).\mathsf{G}_l\}_{l\in L_{ij}}$ and $\forall l \in L : \mathsf{G}_l \upharpoonright_{\mathtt{r}} \mathsf{T}'_l$;
  (b) when $\mathsf{T} = \bigvee_{l\in L} \mathtt{p}!\ell_l(S_l).\mathsf{T}'_l$, for all $i \in I$ and $j \in J_i$, $\mathsf{G}_j = \mathtt{r} \to \mathtt{p} : \{\ell_l(S_l).\mathsf{G}_l\}_{l\in L}$ and $\forall l \in L : \mathsf{G}_l \upharpoonright_{\mathtt{r}} \mathsf{T}'_l$.
  Now, letting $J = \bigcup_{i\in I} J_i$, by structural induction on $\mathfrak{G}$ we can show that merging is defined only if $\mathsf{T} = \bigsqcap_{i\in I} \mathsf{T}_i = \bigsqcap_{j\in J} \mathsf{T}'_j$, with $\mathsf{G}_j \upharpoonright_{\mathtt{r}} \mathsf{T}'_j$ — i.e., by Definition 3.6, T is obtained by "skipping" $\mathfrak{G}$ (where $\mathtt{r}$ does not occur) and merging the projections of the types G_j grafted at its leafs (where $\mathtt{r}$ occurs). Then, by Definition 3.6, we have the following possibilities:
  - in case *(a)* above, letting $\mathsf{G}' = \mathtt{p} \to \mathtt{r} : \{\ell_l(S_l).\mathsf{G}_l\}_{l\in L}$, by Definition 3.6 we conclude that $\mathsf{G}' \upharpoonright_{\mathtt{r}} \mathsf{T}$ holds non-vacuously by clause [PROJ-IN];
  - in case *(b)* above, letting $\mathsf{G}' = \mathtt{r} \to \mathtt{p} : \{\ell_l(S_l).\mathsf{G}_l\}_{l\in L}$, by Definition 3.6 we conclude that $\mathsf{G}' \upharpoonright_{\mathtt{r}} \mathsf{T}$ holds non-vacuously by clause [PROJ-OUT]. □

Proposition 3.11. *The projection relation* $\upharpoonright_{\mathtt{r}}$ *is a partial function.*

Proof. To prove the statement, we need to show that if $\mathsf{G} \upharpoonright_{\mathtt{r}} \mathsf{T}$ and $\mathsf{G} \upharpoonright_{\mathtt{r}} \mathsf{T}'$ (for some G), then $\mathsf{T} = \mathsf{T}'$, by Definition A.1. We leverage Lemma A.17, to obtain the result by proving $\mathsf{T} \sim \mathsf{T}'$ (Definition A.16). We define the following relation:

$$\mathcal{R}' \ = \ \big\{(\mathsf{T}, \mathsf{T}') \ \big| \ \exists \mathsf{G} : \ \mathsf{G} \upharpoonright_{\mathtt{r}} \mathsf{T} \ \text{ and } \ \mathsf{G} \upharpoonright_{\mathtt{r}} \mathsf{T}'\big\}$$

and prove that $\mathcal{R}'$ is session type tree isomorphism, by Definition A.16. We proceed by taking each pair $(\mathsf{T}, \mathsf{T}') \in \mathcal{R}'$, and inspecting the corresponding G such that $\mathsf{G} \upharpoonright_{\mathtt{r}} \mathsf{T}$ and $\mathsf{G} \upharpoonright_{\mathtt{r}} \mathsf{T}'$:

- if $\mathtt{r} \notin \mathtt{pt}\{\mathsf{G}\}$, then (by clause [PROJ-END] of Definition 3.6) $\mathsf{T} = \mathsf{T}' = \mathtt{end}$. Hence, the pair $(\mathsf{T}, \mathsf{T}')$ satisfies clause i, and (vacuously) clauses ii and iii of Definition A.16;
- if $\mathsf{G} = \mathtt{p} \to \mathtt{r} : \{\ell_j(S_j).\mathsf{G}_j\}_{j\in J}$, then (by clause [PROJ-IN] of Definition 3.6) $\mathsf{T} = \bigwedge_{j\in J} \mathtt{p}?\ell_j(S_j).\mathsf{T}_j$, $\mathsf{T}' = \bigwedge_{j\in J} \mathtt{p}?\ell_j(S_j).\mathsf{T}'_j$, and for all $j \in J$, $\mathsf{G}_j \upharpoonright_{\mathtt{r}} \mathsf{T}_j$ and $\mathsf{G}_j \upharpoonright_{\mathtt{r}} \mathsf{T}'_j$: therefore, $\forall j \in J : (\mathsf{T}_j, \mathsf{T}'_j) \in \mathcal{R}'$. Hence, the pair $(\mathsf{T}, \mathsf{T}')$ satisfies clause ii, and (vacuously) clauses i and iii of Definition A.16;
- if $\mathsf{G} = \mathtt{r} \to \mathtt{q} : \{\ell_j(S_j).\mathsf{G}_j\}_{j\in J}$, then (by clause [PROJ-OUT] of Definition 3.6) $\mathsf{T} = \bigvee_{j\in J} \mathtt{p}!\ell_j(S_j).\mathsf{T}_j$, $\mathsf{T}' = \bigvee_{j\in J} \mathtt{p}!\ell_j(S_j).\mathsf{T}'_j$, and for all $j \in J$, $\mathsf{G}_j \upharpoonright_{\mathtt{r}} \mathsf{T}_j$ and $\mathsf{G}_j \upharpoonright_{\mathtt{r}} \mathsf{T}'_j$: therefore, $\forall j \in J : (\mathsf{T}_j, \mathsf{T}'_j) \in \mathcal{R}'$. Hence, the pair $(\mathsf{T}, \mathsf{T}')$ satisfies clause iii, and (vacuously) clauses i and ii of Definition A.16;
- if $\mathsf{G} = \mathtt{p} \to \mathtt{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}$ and $\mathtt{r} \notin \{\mathtt{p}, \mathtt{q}\}$, then $\mathsf{G} \upharpoonright_{\mathtt{r}} \mathsf{T}$ and $\mathsf{G} \upharpoonright_{\mathtt{r}} \mathsf{T}'$ hold non-vacuously by clause [PROJ-CONT] of Definition 3.6; hence, by Proposition A.20, there is G' such that $\mathsf{G}' \upharpoonright_{\mathtt{r}} \mathsf{T}$ and $\mathsf{G}' \upharpoonright_{\mathtt{r}} \mathsf{T}'$ hold non-vacuously by clause [PROJ-END], [PROJ-IN], or [PROJ-OUT] of Definition 3.6. Hence, we fall back into one of the previous three cases.

We have thus proved that $\mathcal{R}'$ is a session type tree isomorphism. Hence, since if $\mathsf{G} \upharpoonright_{\mathsf{r}} \mathsf{T}$ and $\mathsf{G} \upharpoonright_{\mathsf{r}} \mathsf{T}'$ (for some G) we have $(\mathsf{T}, \mathsf{T}') \in \mathcal{R}'$, we also have $\mathsf{T} \sim \mathsf{T}'$ (by Definition A.16), hence $\mathsf{T} = \mathsf{T}'$ (by Lemma A.17), and we conclude that the projection relation $\upharpoonright_{\mathsf{r}}$ is a partial function. □

A.4. Properties of session type merging

Proposition 3.8. *For two type trees* $\mathsf{T}' = \bigwedge_{i\in I} \mathsf{p}'?\ell_i(S_i).\mathsf{T}_i$ *and* $\mathsf{T}'' = \bigwedge_{j\in J} \mathsf{p}''?\ell_j(S_j).\mathsf{T}_j$, *we have that* $\mathsf{T}' \sqcap \mathsf{T}''$ *is defined* if and only if $\mathsf{p}' = \mathsf{p}''$ *and, whenever* $\ell_i = \ell_j$ *(for some* $i \in I$ *and* $j \in J$*),* $S_i = S_j$ *and* $\mathsf{T}_i = \mathsf{T}_j$.

Proof. ($\Longrightarrow$). Assuming $\mathsf{T} = \mathsf{T}' \sqcap \mathsf{T}''$, we proceed by cases on the merging rule in Definition 3.6 that yields T. If T is obtained by rule [MRG-ID], then we have $\mathsf{T} = \mathsf{T}' = \mathsf{T}''$, and the result is immediate. Otherwise, if T is obtained by rule [MRG-BRA], we know that there exist p, I', J' such that:

$$\mathsf{T}' = \bigwedge_{i\in I'} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i \text{ and } \mathsf{T}'' = \bigwedge_{j\in J'} \mathsf{p}?\ell_j(S_j).\mathsf{T}_j \text{ and } \mathsf{T} = \bigwedge_{k\in I'\cup J'} \mathsf{p}?\ell_k(S_k).\mathsf{T}_k \qquad \text{(by Definition 3.6)} \tag{9}$$

and this implies $\mathsf{p} = \mathsf{p}' = \mathsf{p}''$. Now, from the statement, assume $\ell_i = \ell_j$, for some $i \in I$ and $j \in J$; then pick the corresponding $\ell_{i'} = \ell_{j'} = \ell_i = \ell_j$, for some $i' \in I'$ and $j' \in J'$: we show that we must have $i' = j'$. We prove it by contradiction: if we assume $i' \neq j'$, then from (9) we have that, since $\{i', j'\} \subseteq I' \cup J'$, T is an invalid type tree with duplicated labels $\ell_{i'}$ and $\ell_{j'}$ – i.e., (9) does not allow to merge T' and T'' into T (contradiction). Therefore, whenever $\ell_{i'} = \ell_{j'}$, we must have $i' = j'$; from this, we get $S_{i'} = S_{j'}$ and $\mathsf{T}_{i'} = \mathsf{T}_{j'}$; and since $S_{i'} = S_i$, $\mathsf{T}_{i'} = \mathsf{T}_i$, $S_{j'} = S_j$, and $\mathsf{T}_{j'} = \mathsf{T}_j$, we conclude $S_i = S_j$ and $\mathsf{T}_i = \mathsf{T}_j$.

($\Longleftarrow$). Assume that $\mathsf{p}' = \mathsf{p}''$ and, whenever $\ell_{i'} = \ell_{j'}$ (for $i' \in I$ and $j' \in J$), $S_{i'} = S_{j'}$ and $\mathsf{T}_{i'} = \mathsf{T}_{j'}$. Take a pair of indexes i' and j' that satisfy the above condition. We can reindex T' by replacing i' with j', i.e.:

$$\mathsf{T}' = \bigwedge_{i\in(I\setminus i')\cup\{j'\}} \mathsf{p}'?\ell_i(S_i).\mathsf{T}_i$$

By repeating the procedure for all common labels of T' and T'', we obtain an indexing set I' for T' sharing a common index with J for each equal branch of T' and T''. Then, we can define $\mathsf{T} = \bigwedge_{k\in I'\cup J} \mathsf{p}'?\ell_k(S_k).\mathsf{T}_k$, and by Definition 3.6, we conclude that there exists $\mathsf{T} = \mathsf{T}' \sqcap \mathsf{T}''$. □

Proposition 3.7. *The merging operation is associative, i.e.:* $\mathsf{T} \sqcap (\mathsf{T}' \sqcap \mathsf{T}'') = (\mathsf{T} \sqcap \mathsf{T}') \sqcap \mathsf{T}''$.

Proof. By examining T, T' and T'', considering when their merging is defined by Definition 3.6. Note that if one of T, T' or T'' is `end` or an internal choice, then we must have $\mathsf{T} = \mathsf{T}' = \mathsf{T}''$, and we trivially conclude the proof. Otherwise we have, for some I, J, K:

- $\mathsf{T} = \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i$
- $\mathsf{T}' = \bigwedge_{j\in J} \mathsf{p}?\ell_j(S_j).\mathsf{T}_j$
- $\mathsf{T}'' = \bigwedge_{k\in K} \mathsf{p}?\ell_k(S_k).\mathsf{T}_k$

Then, by Definition 3.6, $\mathsf{T}' \sqcap \mathsf{T}'' = \bigwedge_{l\in J\cup K} \mathsf{p}?\ell_l(S_l).\mathsf{T}_l$, and $\mathsf{T} \sqcap (\mathsf{T}' \sqcap \mathsf{T}'') = \bigwedge_{l\in I\cup J\cup K} \mathsf{p}?\ell_l(S_l).\mathsf{T}_l$. Moreover, $(\mathsf{T} \sqcap \mathsf{T}') = \bigwedge_{m\in I\cup J} \mathsf{p}?\ell_m(S_m).\mathsf{T}_m$, and $(\mathsf{T} \sqcap \mathsf{T}') \sqcap \mathsf{T}'' = \bigwedge_{m\in I\cup J\cup K} \mathsf{p}?\ell_m(S_m).\mathsf{T}_m$. Therefore, we conclude $\mathsf{T} \sqcap (\mathsf{T}' \sqcap \mathsf{T}'') = (\mathsf{T} \sqcap \mathsf{T}') \sqcap \mathsf{T}''$. □

A.5. Substitution and congruence lemmas

Lemma A.21 *(Substitution lemma for process variables). If* $\Gamma, X : \mathsf{T} \vdash P : \mathsf{T}$ *and* $\Gamma \vdash Q : \mathsf{T}$, *then* $\Gamma \vdash P\{Q/X\} : \mathsf{T}$.

Proof. By structural induction on P. We assume that X occurs free in P and we proceed by cases on the last rule applied in the derivation of $\Gamma, X : \mathsf{T} \vdash P : \mathsf{T}$.

- [T-SUB] In this case by inversion of [T-SUB] we know that $\Gamma, X : \mathsf{T} \vdash P : \mathsf{T}'$ for $\mathsf{T}' \leqslant \mathsf{T}$. By induction hypothesis we get $\Gamma \vdash P\{Q/X\} : \mathsf{T}'$. We conclude the proof in this case from $\mathsf{T}' \leqslant \mathsf{T}$ and by an application of [T-SUB].
- [T- **0**] impossible since we assume that X occurs free in P.
- [T-VAR] In this case $P = X$. Then $P\{Q/X\} = Q$, so the proof is trivial.
- [T-REC] In this case $P = \mu Y.P'$ and $X \neq Y$. From Lemma A.23.(1).(d) we know that $\Gamma, X : \mathsf{T}, Y : \mathsf{T} \vdash P' : \mathsf{T}$. By induction hypothesis we get $\Gamma, Y : \mathsf{T} \vdash P'\{Q/X\} : \mathsf{T}$. We conclude the proof in this case by an application of [T-REC].
- The proof in cases [T-IN-CHOICE], [T-OUT], and [T-CHOICE] is straightforward by induction hypothesis. □

Lemma A.22.

(1) Let $\Gamma \vdash P : \mathsf{T}$ *and* $P \equiv Q$. *Then* $\Gamma \vdash Q : \mathsf{T}$
(2) Let $\vdash \mathcal{M} : \mathsf{G}$ *and* $\mathcal{M} \equiv \mathcal{M}'$. *Then* $\vdash \mathcal{M} : \mathsf{G}$

Proof. By case analysis on the derivation of $P \equiv Q$ and $\mathcal{M} \equiv \mathcal{M}'$

1. • The only case is [S-REC]. So, in this case, $P = \mu X.P'$ and $Q = P'\{\mu X.P'/X\}$. By Lemma A.23.(1).(d) we get $\Gamma, X : \mathsf{T} \vdash P' : \mathsf{T}$. From that, $\Gamma \vdash P : \mathsf{T}$ and Lemma A.21, we derive the proof.
2. • [S-MULTI] In this case $\mathcal{M} = \mathsf{p} \triangleleft P \mid \mathcal{M}_1$, $\mathcal{M}' = \mathsf{p} \triangleleft Q \mid \mathcal{M}_1$ and $P \equiv Q$. Let $\mathcal{M}_1 = \prod_{i \in I} \mathsf{p}_i \triangleleft P_i$. Then, by Lemma A.23.(2) we get $\mathtt{pt}\{\mathsf{G}\} \subseteq \{\mathsf{p}\} \cup \{\mathsf{p}_i \mid i \in I\}$ and $\vdash P : \mathsf{T}$ and $\mathsf{G} \upharpoonright_{\mathsf{p}} \mathsf{T}$ and $\vdash P_i : \mathsf{T}_i$ and $\mathsf{G} \upharpoonright_{\mathsf{p}_i} \mathsf{T}_i$ for all $i \in I$. So, from $\vdash P : \mathsf{T}$ and $P \equiv Q$, by Lemma A.22.1, we obtain $\vdash Q : \mathsf{T}$. We conclude the proof by an application of [T-SESS].
   • [S-PAR 1] In this case $\mathcal{M} = \mathsf{p} \triangleleft \mathbf{0} \mid \mathcal{M}_1$ and $\mathcal{M}' = \mathcal{M}_1$. Let $\mathcal{M}_1 = \prod_{i \in I} \mathsf{p}_i \triangleleft P_i$. Then, by Lemma A.23.(2) we get $\mathtt{pt}\{\mathsf{G}\} \subseteq \{\mathsf{p}\} \cup \{\mathsf{p}_i \mid i \in I\}$ and $\vdash \mathbf{0} : \mathsf{T}$ and $\mathsf{G} \upharpoonright_{\mathsf{p}} \mathsf{T}$ and $\vdash P_i : \mathsf{T}_i$ and $\mathsf{G} \upharpoonright_{\mathsf{p}_i} \mathsf{T}_i$ for all $i \in I$. From Lemma A.23.(1).(f) we know that $\mathsf{T} = \mathtt{end}$. We derive the proof from Lemma A.23.(2), Definition 3.6, $\mathsf{p} \notin \mathtt{pt}\{\mathcal{M}_1\}$ and an application of [T-SESS].
   • The proof in cases [S-PAR 2] and [S-PAR 3] is trivial. □

A.6. Properties of subtyping and type system

Lemma 3.16. *The subtyping relation* $\leqslant$ *is reflexive and transitive.*

Proof. Proof of reflexivity is straightforward and we give here only the proof of transitivity. Let $\mathsf{T}_1 \leqslant^+ \mathsf{T}_3$ if there exists T_2 such that $\mathsf{T}_1 \leqslant \mathsf{T}_2$ and $\mathsf{T}_2 \leqslant \mathsf{T}_3$. We are going to show that $\leqslant^+$ satisfies the rules from Definition 3.15 — and therefore, since $\leqslant$ is the largest relation satisfying such rules, $\leqslant^+ \subseteq \leqslant$. We have the following cases:

1. $\mathsf{T}_1 = \mathsf{T}_2 = \mathsf{T}_3 = \mathtt{end}$: In this case $\mathsf{T}_1 \leqslant^+ \mathsf{T}_3$ satisfies [SUB-END].
2. $\mathsf{T}_1 = \bigwedge_{i \in I \cup J \cup K} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i$ and $\mathsf{T}_2 = \bigwedge_{i \in I \cup J} \mathsf{p}?\ell_i(S'_i).\mathsf{T}'_i$ and $\mathsf{T}_3 = \bigwedge_{i \in I} \mathsf{p}?\ell_i(S''_i).\mathsf{T}''_i$ and
   $S'_i \leq: S_i$ and $\mathsf{T}_i \leqslant \mathsf{T}'_i$, for every $i \in I \cup J$, and $S''_i \leq: S'_i$ and $\mathsf{T}'_i \leqslant \mathsf{T}''_i$, for every $i \in I$.
   By transitivity of $\leq:$ and definition of $\leqslant^+$, we conclude that $S''_i \leq: S_i$ and $\mathsf{T}_i \leqslant^+ \mathsf{T}''_i$, for every $i \in I$. Thus, $\mathsf{T}_1 \leqslant^+ \mathsf{T}_3$ satisfies [SUB-IN].
3. $\mathsf{T}_1 = \bigvee_{i \in I} \mathsf{p}!\ell_i(S_i).\mathsf{T}_i$ and $\mathsf{T}_2 = \bigvee_{i \in I \cup J} \mathsf{p}?\ell_i(S'_i).\mathsf{T}'_i$ and $\mathsf{T}_3 = \bigvee_{i \in I \cup J \cup K} \mathsf{p}?\ell_i(S''_i).\mathsf{T}''_i$ and
   $S_i \leq: S'_i$ and $\mathsf{T}_i \leqslant \mathsf{T}'_i$, for every $i \in I$, and $S'_i \leq: S''_i$ and $\mathsf{T}'_i \leqslant \mathsf{T}''_i$, for every $i \in I \cup J$.
   By transitivity of $\leq:$ and definition of $\leqslant^+$, we conclude that $S_i \leq: S''_i$ and $\mathsf{T}_i \leqslant^+ \mathsf{T}''_i$, for every $i \in I$. Thus, $\mathsf{T}_1 \leqslant^+ \mathsf{T}_3$ satisfies [SUB-OUT].

At this point, we have shown that $\leqslant^+ \subseteq \leqslant$. Hence, take $\mathsf{T}_1, \mathsf{T}_2, \mathsf{T}_3$ such that $\mathsf{T}_1 \leqslant \mathsf{T}_2$ and $\mathsf{T}_2 \leqslant \mathsf{T}_3$: since $\mathsf{T}_1 \leqslant^+ \mathsf{T}_3$, from the above we have $\mathsf{T}_1 \leqslant \mathsf{T}_3$, and we conclude that $\leqslant$ is transitive. □

As usual, we start with inversion and substitution lemmas.

Lemma A.23 *(Inversion lemma). (1) Let* $\Gamma \vdash P : \mathsf{T}$.

(a) If $P = \sum_{i \in I} \mathsf{p}?\ell_i(x).Q_i$, *then* $\bigwedge_{i \in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i \leqslant \mathsf{T}$ *and* $\Gamma, x : S_i \vdash Q_i : \mathsf{T}_i$, *for every* $i \in I$.
(b) If $P = \mathsf{p}!\ell(\mathsf{e}).Q$, *then* $\Gamma \vdash \mathsf{e} : S$ *and* $\Gamma \vdash Q : \mathsf{T}'$ *and* $S' \leq: S$ *and* $\mathsf{p}!\ell(S').\mathsf{T}' \leqslant \mathsf{T}$.
(c) If $P = \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2$, *then* $\Gamma \vdash P_1 : \mathsf{T}_1$ *and* $\Gamma \vdash P_2 : \mathsf{T}_2$ *and* $\mathsf{T}_1 \leqslant \mathsf{T}$ *and* $\mathsf{T}_2 \leqslant \mathsf{T}$.
(d) If $P = \mu X.Q$, *then* $\Gamma, X : \mathsf{T} \vdash Q : \mathsf{T}$.
(e) If $P = X$, *then* $\Gamma = \Gamma', X : \mathsf{T}'$ *and* $\mathsf{T}' \leqslant \mathsf{T}$.
(f) If $P = \mathbf{0}$, *then* $\mathsf{T} = \mathtt{end}$.

(2) If $\vdash \prod_{i \in I} \mathsf{p}_i \triangleleft P_i : \mathsf{G}$, *then* $\mathtt{pt}\{\mathsf{G}\} \subseteq \{\mathsf{p}_i \mid i \in I\}$ *and* $\vdash P_i : \mathsf{G} \upharpoonright \mathsf{p}_i$ *for every* $i \in I$.

Proof. By induction on type derivations. □

Lemma A.24 *(Substitution lemma). If* $\Gamma, x : S \vdash P : \mathsf{T}$ *and* $\Gamma \vdash \mathsf{v} : S$, *then* $\Gamma \vdash P\{\mathsf{v}/x\} : \mathsf{T}$.

Proof. By structural induction on P. □

A.7. Subject reduction and type safety

Proposition A.25. *If* $\mathsf{T} = \bigsqcap_{i\in I} \mathsf{T}_i$ *and* $\forall i \in I : \mathsf{T}' \leqslant \mathsf{T}_i$, *then* $\mathsf{T}' \leqslant \mathsf{T}$.

Proof. Assume $\mathsf{T} = \bigsqcap_{i\in I} \mathsf{T}_i$ and take any T' such that $\forall i \in I : \mathsf{T}' \leqslant \mathsf{T}_i$. We have the following two cases:

- for some $i \in I$, T_i is an internal choice (resp. $\mathtt{end}$). This implies that, since $\bigsqcap_{i\in I} \mathsf{T}_i$ is defined, *for all* $i \in I$, T_i is an internal choice (resp. $\mathtt{end}$), and $\forall i, j \in I$, $\mathsf{T}_i = \mathsf{T}_j = \mathsf{T}$ (by Definition 3.6). Moreover, since $\mathsf{T}' \leqslant \mathsf{T}_i = \mathsf{T}$ (for all $i \in I$), by Lemma 3.16 we conclude $\mathsf{T} \leqslant \mathsf{T}'$;
- for some $i \in I$, T_i is an external choice. This implies that, since $\bigsqcap_{i\in I} \mathsf{T}_i$ is defined, *for all* $i \in I$ there is J_i such that $\mathsf{T}_i = \bigwedge_{j\in J_i} \mathsf{p}?\ell_j(S_j).\mathsf{T}_j$, and letting $J = \bigcup_{i\in I} J_i$, we have $\mathsf{T} = \bigwedge_{j\in J} \mathsf{p}?\ell_j(S_j).\mathsf{T}_j$ (by Definition 3.6). Moreover, since $\mathsf{T}' \leqslant \mathsf{T}'_i$ (for all $i \in I$), by Definition 3.15 (rule [SUB-IN]) we know that each T' is an external choice and, for some $J' \supseteq \bigcup_{i\in I} J_i = J$, we have $\mathsf{T}' = \bigwedge_{j\in J'} \mathsf{p}?\ell_j(S'_j).\mathsf{T}'_j$ and, $\forall j \in \bigcup_{i\in I} J_i = J$, $S_j \leq: S'_j$ and $\mathsf{T}'_j \leqslant \mathsf{T}_j$. Hence, by rule [SUB-IN] of Definition 3.15, we conclude $\mathsf{T}' \leqslant \mathsf{T}$. □

Proposition A.26. *If* $\mathsf{T} = \bigsqcap_{i\in I} \mathsf{T}_i$, *then for all* $i \in I$, $\mathsf{T} \leqslant \mathsf{T}_i$.

Proof. Assume $\mathsf{T} = \bigsqcap_{i\in I} \mathsf{T}_i$. We have the following two cases:

- T is an internal choice or $\mathsf{T} = \mathtt{end}$. Then, by Definition 3.6, $\bigsqcap_{i\in I} \mathsf{T}_i = \mathsf{T}_j = \mathsf{T}_k$ (for all $j, k \in I$), and by reflexivity of $\leqslant$ (Lemma 3.16) we conclude $\mathsf{T} \leqslant \mathsf{T}_i$ (for all $i \in I$);
- T is an external choice. Then, by Definition 3.6, for all $i \in I$ there is J_i such that $\mathsf{T}_i = \bigwedge_{j\in J_i} \mathsf{p}?\ell_j(S_j).\mathsf{T}_j$, and letting $J = \bigcup_{i\in I} J_i$, we have $\mathsf{T} = \bigwedge_{j\in J} \mathsf{p}?\ell_j(S_j).\mathsf{T}_j$. Therefore, each T_i ($i \in I$) has a subset of the branches of T; hence, by rule [SUB-IN] of Definition 3.15, we conclude $\mathsf{T} \leqslant \mathsf{T}_i$ (for all $i \in I$). □

Corollary A.27. *If* $\mathsf{T} = \bigsqcap_{i\in I} \mathsf{T}_i$, $\mathsf{T}' = \bigsqcap_{i\in I} \mathsf{T}'_i$, *and* $\forall i \in I : \mathsf{T}_i \leqslant \mathsf{T}'_i$, *then* $\mathsf{T} \leqslant \mathsf{T}'$.

Proof. Assume all the hypotheses. By Proposition A.26 we have $\forall i \in I : \mathsf{T} \leqslant \mathsf{T}_i$; hence, by Lemma 3.16, $\forall i \in I : \mathsf{T} \leqslant \mathsf{T}'_i$; thus, by Proposition A.25, we have $\mathsf{T} \leqslant \bigsqcap_{i\in I} \mathsf{T}'_i$, and by Lemma 3.16, we conclude $\mathsf{T} \leqslant \mathsf{T}'$. □

Definition A.28 below restates Definition 3.3 (balancing), under the formalism of Definition A.4.

Definition A.28 *(Balanced global type tree).* A *node of a global type tree* is a pair (f, w) where f is a global type tree (Definition A.4), and w is a sequence of continuation labels for which $f(w)$ is defined. Given a type tree node $n = (f, w)$, we say that n *involves* p if $f(w) = \mathsf{p} \to \mathsf{q}$ or $f(w) = \mathsf{q} \to \mathsf{p}$ (for some q).

We say that a type tree f *is balanced* iff:

for all nodes (f, w),
  if there is some p such that node $(f, w{\cdot}w')$ involves p (for some w'),
    then there is k finite such that,
      for all w' for which node $(f, w{\cdot}w')$ exists, and either (a) $f(w{\cdot}w') = \mathtt{end}$, or (b) w' has length k,
        there is w'' such that $w' = w''{\cdot}w_0$ (for some w_0)
          and node $(f, w{\cdot}w'')$ involves p.

Lemma A.29. *Assume:*

(i) $\mathsf{q}!\ell(S).\mathsf{T} \leqslant \mathsf{G}\upharpoonright\mathsf{p}$
(ii) either $\mathsf{p}?\ell(S').\mathsf{T}' \leqslant \mathsf{G}\upharpoonright\mathsf{q}$ *or* $\mathsf{p}?\ell(S').\mathsf{T}' \wedge \mathsf{T}'' \leqslant \mathsf{G}\upharpoonright\mathsf{q}$

Then, there are $\mathfrak{G}$, J, $\{\mathsf{G}_j\}_{j\in J}$ *such that:*

1. $\mathsf{G} = \mathfrak{G}[\mathsf{G}_j]^j_{j\in J}$
2. $\mathsf{p} \notin \mathtt{pt}\{\mathfrak{G}\} \not\ni \mathsf{q}$

Moreover, there is I *and* $\{\mathsf{G}'_{ji}\}_{j\in J, i\in I}$ *such that, for all* $j \in J$*:*

3. $\mathsf{G}_j = \mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}'_{ji}\}_{i\in I}$

Further, there is $k \in I$ *such that:*

4. $\ell = \ell_k$
5. $S \leq: S_k$ *and* $\forall j \in J : \mathsf{T} \leqslant \mathsf{G}'_{jk} \upharpoonright \mathsf{p}$
6. $S_k \leq: S'$ *and* $\forall j \in J : \mathsf{T}' \leqslant \mathsf{G}'_{jk} \upharpoonright \mathsf{q}$

Proof. Assume the hypothesis *(i)* of the statement. We have:

$$\exists I, k \in I : \begin{cases} \mathsf{G} \upharpoonright \mathsf{p} = \bigvee_{i \in I} \mathsf{q}!\ell_i(S_i).\mathsf{T}_i \\ \text{and } \ell = \ell_k, S \leq: S_k, \mathsf{T} \leqslant \mathsf{T}_k \end{cases} \quad \text{(by hyp. } (i) \text{ and Definition 3.15, rule [SUB-OUT])} \tag{10}$$

Moreover, we have:

$$\exists \mathfrak{G}, J : \begin{cases} \mathsf{G} = \mathfrak{G}[\mathsf{G}_j]^j_{j \in J} \text{ and } \mathsf{p} \notin \mathtt{pt}\{\mathfrak{G}\} \\ \text{and } \forall j \in J : \mathsf{G}_j = \mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}'_{j_i}\}_{i \in I} \\ \text{and } \forall j, j' \in J \text{ and } i \in I : \mathsf{T}_i = \mathsf{G}'_{ji} \upharpoonright \mathsf{p} = \mathsf{G}'_{j'i} \upharpoonright \mathsf{p} \end{cases} \tag{11}$$

To prove (11) above, observe that since G is balanced (Definitions 3.3 and A.28), we can always choose a $\mathfrak{G}, J$ such that $\mathsf{G} = \mathfrak{G}[\mathsf{G}_j]^j_{j \in J}$ and $\mathsf{p} \notin \mathtt{pt}\{\mathfrak{G}\}$, simply by extending $\mathfrak{G}$ along G until we reach, in each branch, a communication involving p; also note that such communication must exist in each branch of G, with the form $\mathsf{p} \to \mathsf{q} : \{\ldots\}$ – otherwise, the projection $\mathsf{G} \upharpoonright \mathsf{p}$ would not match (10) (i.e., we would get a contradiction). Furthermore, observe that in (10), $\mathsf{G} \upharpoonright \mathsf{p} = \bigvee_{i \in I} \mathsf{q}!\ell_i(S_i).\mathsf{T}_i$ can only hold if $\forall j \in J : \mathsf{G}_j \upharpoonright \mathsf{p} = \mathsf{G} \upharpoonright \mathsf{p} = \bigvee_{i \in I} \mathsf{q}!\ell_i(S_i).\mathsf{T}_i$: this can be proven by induction on the size of J, i.e., the number of holes in $\mathfrak{G}$, which corresponds to the number of projections $\mathsf{G}_j \upharpoonright \mathsf{p}$ that need to be merged to obtain $\mathsf{G} \upharpoonright \mathsf{p}$, by Definition 3.6. Now, notice that since we have obtained that $\forall j \in J : \mathsf{G}_j \upharpoonright \mathsf{p} = \bigvee_{i \in I} \mathsf{q}!\ell_i(S_i).\mathsf{T}_i$, then it must also be the case that, for all $j \in J$, $\mathsf{G}_j = \mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}'_{j_i}\}_{i \in I}$, for some $\{\mathsf{G}'_{ji}\}_{j \in J, i \in I}$ such that, $\forall j, j' \in J$ and $i \in I$, $\mathsf{T}_i = \mathsf{G}'_{ji} \upharpoonright \mathsf{p} = \mathsf{G}'_{j'i} \upharpoonright \mathsf{p}$ – otherwise, by Definition 3.6, there would be some $j \in J$ such that $\mathsf{G}_j \upharpoonright \mathsf{p} \neq \mathsf{G} \upharpoonright \mathsf{p}$ (contradiction). We have thus proved (11), and we also get:

$$\forall j, j' \in J : \mathsf{T} \leqslant \mathsf{T}_k = \mathsf{G}_{jk} \upharpoonright \mathsf{p} = \mathsf{G}_{j'k} \upharpoonright \mathsf{p} \qquad \text{(by (10) and (11))} \tag{12}$$

and the results above satisfy items 1, 3, 4, and 5 of the statement.

We are left to prove that, with $I, k, \mathfrak{G}, J$ above, we also satisfy items 2 and 6 of the statement. Assume the hypothesis *(ii)*; with a slight abuse of notation, we simplify the proof below by collapsing its two sub-cases: we allow T'' to be an "empty" external choice $\mathsf{T}'' = \bigwedge_{i \in \emptyset} \mathsf{p}?\ell_i(S''_i).\mathsf{T}''_i$ – and when it happens, we postulate $\mathsf{p}?\ell(S').\mathsf{T}' \wedge \mathsf{T}'' = \mathsf{p}?\ell(S').\mathsf{T}'$. We have:

$$\exists I' : \begin{cases} \mathsf{T}'' = \bigwedge_{i \in I'} \mathsf{p}?\ell_i(S''_i).\mathsf{T}''_i \\ \text{where } \forall i \in I' : \ell \neq \ell_i \end{cases} \quad \text{(by hyp. (ii), for the type to be well-formed)} \tag{13}$$

$$\mathsf{G} \upharpoonright \mathsf{q} = \bigwedge_{i \in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}'''_i \quad \text{(by hyp. (ii) and Definition 3.15, rule [SUB-IN])} \tag{14}$$

Now, observe that (14) implies $\mathsf{q} \notin \mathtt{pt}\{\mathfrak{G}\}$: otherwise, by (11), $\mathfrak{G}$ must contain some communication between q and $\mathsf{r} \neq \mathsf{p}$, which by Definition 3.6, would mean that $\mathsf{G} \upharpoonright \mathsf{q}$ is either undefined, or different from (14) (contradiction). Hence, we get:

$$\mathsf{p} \notin \mathtt{pt}\{\mathfrak{G}\} \not\ni \mathsf{q} \qquad \text{(by (11) and (14))}$$

which satisfies item 2 of the statement. Finally,

$$I' \cup \{k\} \supseteq I \text{ and } \begin{cases} \forall i \in I \setminus \{k\} : S_i \leq: S'_i \text{ and } \mathsf{T}'_i \leq: \mathsf{T}'''_i \\ \text{and } S_k \leq: S' \text{ and } \mathsf{T}' \leqslant \mathsf{T}'''_k \end{cases} \quad \left(\begin{array}{l} \text{by (10), (13), (14), hyp. (ii)} \\ \text{and Definition 3.15 [SUB-IN]} \end{array} \right) \tag{15}$$

$$\forall j, j' \in J : \mathsf{T}'''_k = \mathsf{G}_{jk} \upharpoonright \mathsf{q} = \mathsf{G}_{j'k} \upharpoonright \mathsf{q} \qquad \text{(by (11) and Definition 3.6)} \tag{16}$$

$$\forall j \in J : \mathsf{T}' \leqslant \mathsf{G}_{jk} \upharpoonright \mathsf{q} \qquad \text{(by (15), (16), and Lemma 3.16)} \tag{17}$$

and with (15) and (17) above, we satisfy the remaining item 6 of the statement, and conclude the proof. □

Lemma 3.19. *If* $\mathsf{q}!\ell(S).\mathsf{T} \leq \mathsf{G} \upharpoonright \mathsf{p}$ *and* $\mathsf{p}?\ell(S').\mathsf{T}' \leq \mathsf{G} \upharpoonright \mathsf{q}$, *then:*

(1) $\mathsf{T} \leqslant (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{p}$
(2) $\mathsf{T}' \leqslant (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{q}$; *and*
(3) $\mathsf{G} \upharpoonright \mathsf{r} \leqslant (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}) \upharpoonright \mathsf{r}$ *for* $\mathsf{r} \neq \mathsf{p}, \mathsf{r} \neq \mathsf{q}$.

Proof. Assume the two hypotheses. By Lemma A.29, we know that there is some $\mathfrak{G}$ such that $\mathsf{p} \notin \mathtt{pt}\{\mathfrak{G}\} \not\ni \mathsf{q}$ and $\mathsf{G} = \mathfrak{G}[\mathsf{G}_j]^j_{j \in J}$, and there is I such that (for all $j \in J$) $\mathsf{G}_j = \mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}'_i\}_{i \in I}$ and for some $k \in I$, $\ell_k = \ell$.

(Items (1) and (2)) We proceed by structural induction on $\mathfrak{G}$. In the base case $\mathfrak{G} = [\,]^j$ (i.e., when $\mathfrak{G}$ is just a hole with index j), we have:

$$\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q} \;=\; (\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}'_i\}_{i\in I}) \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q} \;=\; \mathsf{G}'_k \qquad \text{(by Definition 3.18)} \tag{18}$$

$$(\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{p} \;=\; \mathsf{G}'_k\upharpoonright\mathsf{p} \quad \text{and} \quad (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{q} \;=\; \mathsf{G}'_k\upharpoonright\mathsf{q} \qquad \text{(by (18) and Definition 3.6)} \tag{19}$$

$$\mathsf{T} \leqslant (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{p} \qquad \text{(by (19) and Lemma A.29(5))}$$

$$\mathsf{T}' \leqslant (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{q} \qquad \text{(by (19) and Lemma A.29(6))}$$

In the inductive case $\mathfrak{G} = \mathsf{s} \to \mathsf{s}' : \{\ell'_l(S_l).\mathfrak{G}_l\}_{l\in L}$, we have:

$$\mathsf{G} \;=\; \mathfrak{G}[\mathsf{G}_j]^j_{j\in J} \;=\; \mathsf{s} \to \mathsf{s}' : \{\ell'_l(S_l).\mathfrak{G}_l[\mathsf{G}_j]^j_{j\in J_l}\}_{l\in L} \quad \text{with } \textstyle\bigcap_{l\in L} J_l = \emptyset \text{ and } \bigcup_{l\in L} J_l = J \tag{20}$$

i.e., the set J indexing the holes of $\mathfrak{G}$ is partitioned among the sub-contexts $\mathfrak{G}_l$ ($l \in L$); and

$$\begin{aligned} \mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q} \;&=\; \mathfrak{G}[\mathsf{G}_j]^j_{j\in J} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q} \\ &=\; \mathsf{s} \to \mathsf{s}' : \{\ell'_l(S_l).(\mathfrak{G}_l[\mathsf{G}_j]^j_{j\in J_l} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\}_{l\in L} \end{aligned} \qquad \left(\begin{array}{l}\text{by (20) and Definition 3.18,}\\ \text{since } \mathsf{p} \notin \mathtt{pt}\{\mathfrak{G}\} \not\ni \mathsf{q},\\ \text{i.e., } \{\mathsf{s}, \mathsf{s}'\} \cap \{\mathsf{p}, \mathsf{q}\} = \emptyset)\end{array}\right) \tag{21}$$

Therefore:

$$(\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{p} \;=\; \prod_{l\in L}\Big((\mathfrak{G}_l[\mathsf{G}'_j]^j_{j\in J_l} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{p}\Big) \qquad \text{(by (21) and Definitions 3.6 and 3.18)} \tag{22}$$

$$(\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{q} \;=\; \prod_{l\in L}\Big((\mathfrak{G}_l[\mathsf{G}'_j]^j_{j\in J_l} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{q}\Big) \qquad \text{(by (21) and Definitions 3.6 and 3.18)} \tag{23}$$

$$\forall l \in L : \begin{cases} \mathsf{T} \leqslant (\mathfrak{G}_l[\mathsf{G}_j]^j_{j\in J_l} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{p} \quad \text{and} \\ \mathsf{T}' \leqslant (\mathfrak{G}_l[\mathsf{G}_j]^j_{j\in J_l} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{q} \end{cases} \qquad \text{(by the induction hypothesis)} \tag{24}$$

$$\mathsf{T} \leqslant (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{p} \qquad \text{(by (22), (24) and Proposition A.25)} \tag{25}$$

$$\mathsf{T}' \leqslant (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{q} \qquad \text{(by (23), (24) and Proposition A.25)}$$

(Item (3)) When $\mathsf{r} \notin \mathtt{pt}\{\mathsf{G}\}$, we trivially conclude $\mathsf{G}\upharpoonright\mathsf{r} = \mathtt{end} \leqslant \mathtt{end} = (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{r}$, by Definitions 3.6 and 3.18 and Lemma 3.16. Otherwise, when $\mathsf{r} \in \mathtt{pt}\{\mathsf{G}\}$, reminding that $\mathsf{G} = \mathfrak{G}[\mathsf{G}_j]^j_{j\in J}$, we proceed by structural induction on $\mathfrak{G}$. In the base case $\mathfrak{G} = [\,]^j$ (i.e., when $\mathfrak{G}$ is just a hole with index j), we have:

$$\mathsf{G}\upharpoonright\mathsf{r} = \mathsf{G}_j\upharpoonright\mathsf{r} = \big(\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}'_i\}_{i\in I}\big)\upharpoonright\mathsf{r} = \prod_{i\in I_j}(\mathsf{G}'_i\upharpoonright\mathsf{r}) \qquad \text{(by Definition 3.6)} \tag{26}$$

$$\exists k \in I :\; \ell_k = \ell \;\text{ and }\; (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{r} \;=\; (\mathsf{G}_j \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{r} \;=\; \big((\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}'_i\}_{i\in I}) \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q}\big)\upharpoonright\mathsf{r} \;=\; \mathsf{G}'_k\upharpoonright\mathsf{r} \qquad \left(\begin{array}{l}\text{since } \mathsf{p} \neq \mathsf{r} \neq \mathsf{q},\\ \text{and by Definition 3.18,}\\ \text{and Lemma A.29(4)}\end{array}\right) \tag{27}$$

$$\mathsf{G}\upharpoonright\mathsf{r} \;=\; \prod_{i\in I}(\mathsf{G}'_i\upharpoonright\mathsf{r}) \;\leqslant\; \mathsf{G}'_k\upharpoonright\mathsf{r} \;=\; (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{r} \qquad \text{(by (26), (27), Proposition A.26, and Lemma 3.16)}$$

In the inductive case $\mathfrak{G} = \mathsf{s} \to \mathsf{s}' : \{\ell'_l(S_l).\mathfrak{G}_l\}_{l\in L}$, we have (20) and (21) above, and the following sub-cases:

- $\mathsf{s} \neq \mathsf{r} \neq \mathsf{s}'$. We have:

$$\mathsf{G}\upharpoonright\mathsf{r} \;=\; \prod_{l\in L}\big(\mathfrak{G}_l[\mathsf{G}_j]^j_{j\in J_l}\upharpoonright\mathsf{r}\big) \qquad \text{(by (20) and Definition 3.6)} \tag{28}$$

$$(\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{r} \;=\; \prod_{l\in L}\Big((\mathfrak{G}_l[\mathsf{G}_j]^j_{j\in J_l} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{r}\Big) \qquad \text{(by (21) and Definitions 3.6 and 3.18)} \tag{29}$$

$$\forall l \in L :\quad \mathfrak{G}_l[\mathsf{G}_j]^j_{j\in J_l}\upharpoonright\mathsf{r} \;\leqslant\; (\mathfrak{G}_l[\mathsf{G}_j]^j_{j\in J_l} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{r} \qquad \text{(by the induction hypothesis)} \tag{30}$$

  and by (28), (29), (30) and Corollary A.27, we conclude $\mathsf{G}\upharpoonright\mathsf{r} \leqslant (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright\mathsf{r}$;
- $\mathsf{s} = \mathsf{r} \neq \mathsf{s}'$. Then, we have:

$$\mathsf{G}\upharpoonright\mathsf{r} \;=\; \bigvee_{l\in L} \mathsf{s}'!\ell'_l(S_l).\Big((\mathfrak{G}_l[\mathsf{G}_j]^j_{j\in J_l})\upharpoonright\mathsf{r}\Big) \qquad \text{(by (20) and Definition 3.6)} \tag{31}$$

$$(\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright \mathsf{r} = \Big(\mathsf{r} \to \mathsf{s}' : \{\ell'_l(S_l).(\mathfrak{G}_l[\mathsf{G}_j]^j_{j\in J_l} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\}_{l\in L}\Big)\upharpoonright \mathsf{r} = \bigvee_{l\in L} \mathsf{s}'!\ell'_l(S_l).\Big((\mathfrak{G}_l[\mathsf{G}_j]^j_{j\in J_l} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright \mathsf{r}\Big) \quad \text{(by (21) and Definitions 3.6 and 3.18)} \tag{32}$$

$$\forall l \in L: \quad \mathfrak{G}_l[\mathsf{G}_j]^j_{j\in J_l}\upharpoonright \mathsf{r} \leqslant (\mathfrak{G}_l[\mathsf{G}_j]^j_{j\in J_l} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright \mathsf{r} \quad \text{(by the induction hypothesis)} \tag{33}$$

and by (31), (32), (33) and Definition 3.15 (rule [SUB-OUT]), we conclude $\mathsf{G}\upharpoonright \mathsf{r} \leqslant (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright \mathsf{r}$;

- $\mathsf{s} \neq \mathsf{r} = \mathsf{s}'$. Similar to the previous case, except that $\mathsf{G}\upharpoonright \mathsf{r}$ and $(\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell} \mathsf{q})\upharpoonright \mathsf{r}$ trigger the third case of Definition 3.6 (i.e., they produce external choice types, with L-indexed branches), and we conclude by rule [SUB-IN] of Definition 3.15. □

A.8. Subject reduction and progress

Theorem 3.21 *(Subject Reduction). Let $\vdash \mathcal{M} : \mathsf{G}$. For all $\mathcal{M}'$, if $\mathcal{M} \longrightarrow \mathcal{M}'$, then $\vdash \mathcal{M}' : \mathsf{G}'$ for some G' such that $\mathsf{G} \Longrightarrow \mathsf{G}'$.*

Proof. The proof is by induction on the derivation of $\mathcal{M} \longrightarrow \mathcal{M}'$. According to reduction rules (Table 3), we have three base cases.

[R-COMM]: $\mathcal{M} = \mathsf{p} \triangleleft \sum_{i\in I} \mathsf{q}?\ell_i(x).P_i \mid \mathsf{q} \triangleleft \mathsf{p}!\ell_j(\mathsf{e}).Q \mid \prod_{l\in L} \mathsf{p}_l \triangleleft P_l, \mathcal{M}' = \mathsf{p} \triangleleft P_j\{\mathsf{v}/x\} \mid \mathsf{q} \triangleleft Q \mid \prod_{l\in L} \mathsf{p}_l \triangleleft P_l,\ \mathsf{e} \downarrow \mathsf{v}$

From $\vdash \mathcal{M} : \mathsf{G}$, by Lemma A.23(2),

$$\mathtt{pt}\{\mathsf{G}\} \subseteq \{\mathsf{p}, \mathsf{q}\} \cup \{\mathsf{p}_l : l \in L\} \tag{34}$$

$$\vdash \sum_{i\in I} \mathsf{q}?\ell_i(x).P_i : \mathsf{G}\upharpoonright \mathsf{p} \tag{35}$$

$$\vdash \mathsf{p}!\ell_j(\mathsf{e}).Q : \mathsf{G}\upharpoonright \mathsf{q} \tag{36}$$

$$\vdash P_l : \mathsf{G}\upharpoonright \mathsf{p}_l \text{ for every } l \in L \tag{37}$$

By Lemma A.23(1)(a), from (35),

$$\bigwedge_{i\in I} \mathsf{q}?\ell_i(S_i).\mathsf{T}_i \leqslant \mathsf{G}\upharpoonright \mathsf{p} \tag{38}$$

$$x : S_i \vdash P_i : \mathsf{T}_i \text{ for every } i \in I \tag{39}$$

By Lemma A.23(1)(b), from (36),

$$\mathsf{p}!\ell_j(S'_j).\mathsf{T}''_2 \leqslant \mathsf{G}\upharpoonright \mathsf{q} \tag{40}$$

$$\vdash Q : \mathsf{T}''_2 \tag{41}$$

$$\vdash e : S_j \tag{42}$$

$$S'_j \leq: S_j \tag{43}$$

By Lemma A.24, from (39), (42) and $\mathsf{e} \downarrow \mathsf{v}$,

$$\vdash P_j\{v/x\} : \mathsf{T}_j \tag{44}$$

By Lemma 3.19, from (38) and (40)

$$\mathsf{T}''_2 \leqslant (\mathsf{G} \setminus \mathsf{q} \xrightarrow{\ell_j} \mathsf{p})\upharpoonright \mathsf{q} \quad \mathsf{T}_j \leqslant (\mathsf{G} \setminus \mathsf{p} \xrightarrow{\ell_j} \mathsf{q})\upharpoonright \mathsf{p} \quad \mathsf{G}\upharpoonright \mathsf{p}_l \leqslant (\mathsf{G} \setminus \mathsf{q} \xrightarrow{\ell_j} \mathsf{p})\upharpoonright \mathsf{p}_l,\ l \in L \tag{45}$$

By [T-SUB] and [T-SESS], from (37), (41), (44) and (45),

$$\vdash \mathcal{M}' : \mathsf{G} \setminus \mathsf{q} \xrightarrow{\ell_j} \mathsf{p}.$$

[T-CONDITIONAL]: $\mathcal{M} = \mathsf{p} \triangleleft \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ P\ \mathsf{else}\ Q \mid \prod_{l\in L} \mathsf{p}_l \triangleleft P_l,\ \mathcal{M}' = \mathsf{p} \triangleleft P \mid \prod_{l\in L} \mathsf{p}_l \triangleleft P_l,\ e \downarrow \mathsf{true}.$

From $\vdash \mathcal{M} : \mathsf{G}$, by Lemma A.23(2),

$$\vdash \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ P\ \mathsf{else}\ Q : \mathsf{G}\upharpoonright \mathsf{p} \tag{46}$$

$$\vdash P_l : \mathsf{G}\upharpoonright \mathsf{p}_l,\ l \in L \tag{47}$$

By Lemma A.23(1)(c), from (46),

$$\vdash P : \mathsf{T}_1 \qquad \vdash Q : \mathsf{T}_2 \tag{48}$$

$$\mathsf{T}_1 \leqslant \mathsf{G}\upharpoonright \mathsf{p} \qquad \mathsf{T}_2 \leqslant \mathsf{G}\upharpoonright \mathsf{p} \tag{49}$$

By [T-SUB] and [T-SESS], from (47), (48) and (49), we derive $\vdash \mathcal{M}' : \mathsf{G}$.
[F-CONDITIONAL]: This case is similar to the previous one.

Assume that the last applied rule was [R-STRUCT]. Then, $\mathcal{M}_1' \longrightarrow \mathcal{M}_2'$ was derived from $\mathcal{M}_1 \longrightarrow \mathcal{M}_2$, where $\mathcal{M}_1 \equiv \mathcal{M}_1'$ and $\mathcal{M}_2 \equiv \mathcal{M}_2'$. By Lemma A.22(2), from the assumption $\vdash \mathcal{M}_1' : \mathsf{G}$, we deduce $\vdash \mathcal{M}_1 : \mathsf{G}$. By induction hypothesis, $\vdash \mathcal{M}_2 : \mathsf{G}'$ for some G' such that $\mathsf{G} \Longrightarrow \mathsf{G}'$. By Lemma A.22(2), $\vdash \mathcal{M}_2' : \mathsf{G}'$ for some G' such that $\mathsf{G} \Longrightarrow \mathsf{G}'$. □

Corollary 3.22. *Let* $\vdash \mathcal{M} : \mathsf{G}$. *If* $\mathcal{M} \longrightarrow^* \mathcal{M}'$, *then* $\vdash \mathcal{M}' : \mathsf{G}'$ *for some* G' *such that* $\mathsf{G} \Longrightarrow \mathsf{G}'$.

Proof. Assume that $\mathcal{M} = \mathcal{M}_0 \longrightarrow \mathcal{M}_1 \longrightarrow \cdots \longrightarrow \mathcal{M}_{n-1} \longrightarrow \mathcal{M}_n = \mathcal{M}'$, i.e., there are n reduction steps from $\mathcal{M}$ to $\mathcal{M}'$. We proceed by induction on n. The base case $n = 0$ is trivial, as $\mathcal{M} = \mathcal{M}'$. In the inductive case $n = m + 1$, by the induction hypothesis we obtain that there is G'' such that $\vdash \mathcal{M}_m : \mathsf{G}''$ and $\mathsf{G} \Longrightarrow \mathsf{G}''$; then, by Theorem 3.21, we conclude that there is G' such that $\vdash \mathcal{M}' : \mathsf{G}'$ and $\mathsf{G} \Longrightarrow \mathsf{G}'' \Longrightarrow \mathsf{G}'$, i.e., $\mathsf{G} \Longrightarrow \mathsf{G}'$. □

To show progress, a lemma on canonical forms is helpful. The proof easily follows from the inspection of the typing rules.

Lemma A.30 *(Canonical forms of processes). Let* $\Gamma \vdash P : \mathsf{T}$.

(1) *If* $\mathsf{T} = \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i$, *then*
  (a) $P \equiv \mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2$ *and* $\vdash P_1 : \mathsf{T}_1$ *and* $\vdash P_2 : \mathsf{T}_2$ *and* $\mathsf{T}_1 \leqslant \mathsf{T}$ *and* $\mathsf{T}_2 \leqslant \mathsf{T}$, *or*
  (b) $P \equiv \sum_{i\in I'} \mathsf{p}?\ell_i(x_i).P_i$ *and* $\Gamma, x_i : S_i' \vdash P_i : \mathsf{T}_i'$ *and* $S_i \leq: S_i'$ *and* $\mathsf{T}_i' \leqslant \mathsf{T}_i$ *and* $I \subseteq I'$.
(2) *If* $\mathsf{T} = \mathsf{p}!\ell(S').\mathsf{T}'$ *or* $\mathsf{T} = \mathsf{p}!\ell(S').\mathsf{T}' \vee \mathsf{T}''$, *then*
  (a) $P \equiv \mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2$ *and* $\vdash P_1 : \mathsf{T}_1$ *and* $\vdash P_2 : \mathsf{T}_2$ *and* $\mathsf{T}_1 \leqslant \mathsf{T}$ *and* $\mathsf{T}_2 \leqslant \mathsf{T}$, *or*
  (b) $P \equiv \mathsf{p}!\ell(\mathsf{e}).Q$ *and* $\Gamma \vdash \mathsf{e} : S_1'$ *and* $\Gamma \vdash Q : \mathsf{T}_1'$ *and* $\mathsf{T}_1' \leqslant \mathsf{T}'$ *and* $S_1' \leq: S'$.

Proof. The proof is by induction on the derivation $\Gamma \vdash P : \mathsf{T}$. □

Lemma A.31 *(Canonical forms of sessions). Let* $\vdash \mathcal{M} : \mathsf{G}$.

(1) *If* $\mathsf{G} = \mathtt{end}$, *then* $\mathcal{M} \equiv \mathsf{p} \triangleleft \mathbf{0}$.
(2) *If* $\mathsf{G} = \mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}$, *then* $\mathcal{M} \equiv \mathsf{p} \triangleleft P \mid \mathsf{q} \triangleleft Q \mid \mathcal{M}_1$ *and* $\vdash P : \bigvee_{i\in I} \mathsf{q}!\ell_i(S_i).(\mathsf{G}_i\upharpoonright \mathsf{p})$ *and* $\vdash Q : \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).(\mathsf{G}_i\upharpoonright \mathsf{q})$.

Proof. The proof holds by inversion of [T-SESS] and Definition 3.6. □

Theorem 3.23 *(Progress). If* $\vdash \mathcal{M} : \mathsf{G}$, *then either* $\mathcal{M} \equiv \mathsf{p} \triangleleft \mathbf{0}$ *or there is* $\mathcal{M}'$ *such that* $\mathcal{M} \longrightarrow \mathcal{M}'$.

Proof. We consider here G as a global type tree, since by [T-SESS] and the definition of the global type projection, $\mathcal{M}$ is typed considering G in that form.

1. If $\mathsf{G} = \mathtt{end}$, then $\mathcal{M} = \mathsf{p} \triangleleft \mathbf{0}$ by Lemma A.311.
2. If $\mathsf{G} = \mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}$, then $\mathcal{M} \equiv \mathsf{p} \triangleleft P \mid \mathsf{q} \triangleleft Q \mid \mathcal{M}_1$ and $\vdash P : \bigvee_{i\in I} \mathsf{q}!\ell_i(S_i).(\mathsf{G}_i\upharpoonright \mathsf{p})$ and $\vdash Q : \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).(\mathsf{G}_i\upharpoonright \mathsf{q})$. By Lemma A.30, we have the following cases:
  (a) $P \equiv \mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2$ and $\mathsf{e} \downarrow \mathsf{true}$: $\mathcal{M} \longrightarrow \mathsf{p} \triangleleft P_1 \mid \mathsf{q} \triangleleft Q \mid \mathcal{M}_1$, by [T-CONDITIONAL],
  (b) $P \equiv \mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2$ and $\mathsf{e} \downarrow \mathsf{false}$: $\mathcal{M} \longrightarrow \mathsf{p} \triangleleft P_2 \mid \mathsf{q} \triangleleft Q \mid \mathcal{M}_1$, by [F-CONDITIONAL],
  (c) $Q \equiv \mathsf{if}\ e\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2$ and $\mathsf{e} \downarrow \mathsf{true}$: $\mathcal{M} \longrightarrow \mathsf{p} \triangleleft P \mid \mathsf{q} \triangleleft Q_1 \mid \mathcal{M}_1$, by [T-CONDITIONAL],
  (d) $Q \equiv \mathsf{if}\ e\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2$ and $\mathsf{e} \downarrow \mathsf{false}$: $\mathcal{M} \longrightarrow \mathsf{p} \triangleleft P \mid \mathsf{q} \triangleleft Q_2 \mid \mathcal{M}_1$, by [F-CONDITIONAL],
  (e) $P \equiv \mathsf{q}!\ell_j(\mathsf{e}).P_j^{\mathsf{p}}$, $j \in I$, $\mathsf{e} \downarrow v$, and $Q \equiv \sum_{i\in I'} \mathsf{p}?\ell_i(x_i).P_i^{\mathsf{q}}$ and $I \subseteq I'$:
  $\mathcal{M} \longrightarrow \mathsf{p} \triangleleft P_j^{\mathsf{p}} \mid \mathsf{q} \triangleleft P_j^{\mathsf{q}}\{v/x\} \mid \mathcal{M}_1$, by [R-COMM]. □

As a consequence of subject reduction and progress, we get the safety property stating that a typed multiparty session will never get stuck.

Theorem 3.24 *(Type Safety). If* $\vdash \mathcal{M} : \mathsf{G}$, *then it does not hold* $\mathtt{stuck}(\mathcal{M})$.

Proof. Direct consequence of Theorem 3.23, Corollary 3.22, and Definition 2.3. □

A.9. Algorithmic subtyping

Lemma 3.25. *The subtyping procedure in Table 6 is terminating.*

Proof. For a session type $\mathbb{T}$, we inductively define the set of *subterms of* $\mathbb{T}$, denoted by $\mathtt{subterm}(\mathbb{T})$:

$$\mathtt{subterm}(\mathtt{end}) = \{\mathtt{end}\} \qquad \mathtt{subterm}(\mathbf{t}) = \{\mathbf{t}\} \qquad \mathtt{subterm}(\mu\mathbf{t}.\mathbb{T}) = \{\mu\mathbf{t}.\mathbb{T}\} \cup \{\mathbb{T}'\{\mu\mathbf{t}.\mathbb{T}/\mathbf{t}\} \mid \mathbb{T}' \in \mathtt{subterm}(\mathbb{T})\}$$

$$\mathtt{subterm}\left(\bigwedge_{i\in I} \mathsf{p}?\ell(S_i).\mathbb{T}_i\right) = \bigcup_{i\in I} \mathtt{subterm}(\mathbb{T}_i) \cup \left\{\bigwedge_{i\in I} \mathsf{p}?\ell(S_i).\mathbb{T}_i\right\}$$

$$\mathtt{subterm}\left(\bigvee_{i\in I} \mathsf{p}!\ell(S_i).\mathbb{T}_i\right) = \bigcup_{i\in I} \mathtt{subterm}(\mathbb{T}_i) \cup \left\{\bigvee_{i\in I} \mathsf{p}!\ell(S_i).\mathbb{T}_i\right\}.$$

It is clear that the recursive function $\mathtt{subterm}(\mathbb{T})$ terminates, since it traverses $\mathbb{T}$ only once. Moreover, at each recursion step, the function yields a finite set of types, hence $\mathtt{subterm}(\mathbb{T})$ yields a finite set. E.g.,

$$\begin{aligned}
\text{letting } \mathbb{T} = \mu\mathbf{t}.\mathsf{p}!\ell(S).\mathbf{t}, \quad \mathtt{subterm}(\mathbb{T}) &= \{\mathbb{T}\} \cup \{\,\mathbb{T}'\{\mathbb{T}/\mathbf{t}\} \mid \mathbb{T}' \in \mathtt{subterm}(\mathsf{p}!\ell(S).\mathbf{t})\,\} \\
&= \{\mathbb{T}\} \cup \{\,\mathbb{T}'\{\mathbb{T}/\mathbf{t}\} \mid \mathbb{T}' \in \{\mathsf{p}!\ell(S).\mathbf{t}, \mathbf{t}\}\,\} \\
&= \{\mathbb{T}\} \cup \{\,\mathsf{p}!\ell(S).\mathbf{t}\{\mathbb{T}/\mathbf{t}\},\ \mathbf{t}\{\mathbb{T}/\mathbf{t}\}\,\} \\
&= \{\,\mathbb{T},\ \mathsf{p}!\ell(S).\mathbb{T}\,\}
\end{aligned}$$

Let us consider the application of the subtyping procedure to verify $\emptyset \vdash \mathbb{T}_1 \leqslant_a \mathbb{T}_2$, and let $\Theta \vdash \mathbb{T}_1' \leqslant_a \mathbb{T}_2'$ be a goal reached during its execution. Then, we have:

1. $\mathbb{T}_1', \mathbb{T}_2' \in \mathtt{subterm}(\mathbb{T}_1) \cup \mathtt{subterm}(\mathbb{T}_2)$,
2. if $(\mathbb{T}_1'', \mathbb{T}_2'') \in \Theta$ then $\mathbb{T}_1'', \mathbb{T}_2'' \in \mathtt{subterm}(\mathbb{T}_1) \cup \mathtt{subterm}(\mathbb{T}_2)$.

We can associate a measure to each goal reached during the execution of the procedure:

$$\mathtt{measure}\,(\Theta \vdash \mathbb{T}_1 \leqslant_a \mathbb{T}_2) = (n, m)$$

where n is the cardinality of Θ, and m is the maximum number of type constructors in $\mathbb{T}_1$ or $\mathbb{T}_2$. Measures are ordered as follows:

$$(n, m) > (n', m') \text{ if } n < n' \text{ or } (n = n' \text{ and } m > m').$$

The first component of the measure is upper bounded by $|(\mathtt{subterm}(\mathbb{T}_1) \cup \mathtt{subterm}(\mathbb{T}_2))|^2$, whilst the second one is lower bounded by 1. By examining the rules in Table 6, we can verify that in each state of the procedure, the measure of the current goal (in the rule conclusion) is bigger than the measures of its subgoals, if any (in the rule premises), directly implying that the subtyping procedure is terminating. □

Appendix B. Proofs for Section 4

B.1. Complementarity of relations $\leqslant_a$ *and* $\not\leqslant$

Lemma B.1. *If* $(\Theta_2 \vdash \mathbb{T}_2 \leqslant_a \mathbb{T}_2') \in \Phi(\Theta_1 \vdash \mathbb{T}_1 \leqslant_a \mathbb{T}_1')$, *then, for all* Θ *such that* $(\mathbb{T}_1, \mathbb{T}_1') \notin \Theta$, *we have* $(\Theta \cup \Theta_2 \vdash \mathbb{T}_2 \leqslant_a \mathbb{T}_2') \in \Phi(\Theta \cup \Theta_1 \vdash \mathbb{T}_1 \leqslant_a \mathbb{T}_1)$.

Proof. It follows directly from Definition 3.27. □

Lemma 4.5. *If* $\mathcal{T}(\mathbb{T}) \not\leqslant \mathcal{T}(\mathbb{T}')$, *then there is a failing derivation of* $\emptyset \vdash \mathbb{T} \leqslant_a \mathbb{T}'$.

Proof. By induction on the derivation of $\mathcal{T}(\mathbb{T}) \not\leqslant \mathcal{T}(\mathbb{T}')$.

We develop just two cases (the others are similar):

- base case [NSUB-DIFF-IN]. If $\mathcal{T}(\mathbb{T}) = \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathcal{T}(\mathbb{T}_i)$ and $\mathcal{T}(\mathbb{T}') = \bigwedge_{j\in J} \mathsf{q}?\ell'_j(S'_j).\mathcal{T}\left(\mathbb{T}'_j\right)$, then $\mathbb{T} \equiv \mu\mathbf{t}_1 \ldots \mu\mathbf{t}_{l_1}. \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathbb{T}_{1i}$ (or $\mathbb{T} \equiv \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathbb{T}_{1i}$) and $\mathbb{T}' \equiv \mu\mathbf{t}_1 \ldots \mu\mathbf{t}_{l_2}. \bigwedge_{j\in J} \mathsf{q}?\ell'_j(S'_j).\mathbb{T}'_{1j}$ (or $\mathbb{T}' \equiv \bigwedge_{j\in J} \mathsf{q}?\ell'_j(S'_j).\mathbb{T}'_{1j}$). We assume that both $\mathbb{T}$ and $\mathbb{T}'$ are μ-types (the other cases are simpler), and construct a failing derivation of $\emptyset \vdash \mathbb{T} \leqslant_a \mathbb{T}'$ as follows.

$$
\begin{aligned}
J_1 &= \emptyset \vdash \mathbb{T} \leqslant_a \mathbb{T}' \\
J_{k+1} &= \Theta_k \cup \{(\mu\mathbf{t}_k.\mathbb{T}_{2k}, \mathbb{T}')\} \vdash \mathbb{T}_{2k}\{^{\mu\mathbf{t}_k.\mathbb{T}_{2k}}/_{\mathbf{t}_k}\} \leqslant_a \mathbb{T}', k = 1, .., l_1, \text{ where} \\
&\quad \mathbb{T}_{2k}\{^{\mu\mathbf{t}_k.\mathbb{T}_{2k}}/_{\mathbf{t}_k}\} \equiv \mu\mathbf{t}_{k+1}.\mathbb{T}_{2(k+1)}, k \geq 2, \text{ and } \mathbb{T} \equiv \mu\mathbf{t}_1.\mathbb{T}_{21} \text{ and} \\
&\quad \Theta_{k+1} = \Theta_k \cup \{(\mu\mathbf{t}_k.\mathbb{T}_{2k}, \mathbb{T}')\} \text{ and } \Theta_1 = \emptyset \\
J_{l_1+m+1} &= \Theta_m \cup \{(\mu\mathbf{t}_{l_1}.\mathbb{T}_{2l_1}, \mu\mathbf{t}_m.\mathbb{T}'_{2m})\} \vdash \mathbb{T}_{2(l_1+1)} \leqslant_a \mathbb{T}'_{2m}\{^{\mu\mathbf{t}_m.\mathbb{T}'_{2m}}/_{\mathbf{t}_m}\}, m = 1, .., l_2 - 1, \text{ where} \\
&\quad \mathbb{T}_{2(l_1+1)} = \mathbb{T}_{2l_1}\{^{\mu\mathbf{t}_{l_1}.\mathbb{T}_{2l_1}}/_{\mathbf{t}_{l_1}}\} \text{ and } \mathbb{T}_{2l_1} \equiv \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathbb{T}^{l_1}_{1i} (\text{ for some } \mathbb{T}^{l_1}_{1i}, i \in I) \\
&\quad \mathbb{T}'_{2m}\{^{\mu\mathbf{t}_m.\mathbb{T}'_{2m}}/_{\mathbf{t}_m}\} \equiv \mu\mathbf{t}_{m+1}.\mathbb{T}'_{2(m+1)} \text{ and } \mathbb{T}' \equiv \mu\mathbf{t}_1.\mathbb{T}'_{21} \text{ and} \\
&\quad \Theta_{l_1+m+1} = \Theta_{l_1+m} \cup \{(\mu\mathbf{t}_{l_1+m}.\mathbb{T}_{2(l_1+m)}, \mathbb{T}')\} \\
J_{l_1+l_2+1} &= \Theta_{l_1+l_2} \cup \{(\mu\mathbf{t}_{l_1}.\mathbb{T}_{2l_1}, \mu\mathbf{t}_{l_2}.\mathbb{T}'_{2l_2})\} \vdash \mathbb{T}_{2(l_1+1)} \leqslant_a \mathbb{T}'_{l_1+l_2+1}, \text{ where} \\
&\quad \mathbb{T}'_{l_1+l_2+1} \equiv \mathbb{T}'_{2l_2}\{^{\mu\mathbf{t}_{l_2}.\mathbb{T}'_{l_2}}/_{\mathbf{t}_{l_2}}\} \text{ and } \mathbb{T}'_{2l_2} \equiv \bigwedge_{j\in J} \mathsf{q}?\ell'_j(S'_j).\mathbb{T}^{j_2}_j (\text{ for some } \mathbb{T}^{l_2}_{1j}, j \in J)
\end{aligned}
$$

  Note that, if $\mathbb{T}$ is not a μ-type, we erase components $J_2, \ldots, J_{l_1+1}$ − and similarly, if $\mathbb{T}'$ is not a μ-type, we erase the components $J_{l_1+2}, \ldots, J_{l_1+l_2+1}$. The proof for the other base cases [NSUB-DIFF-OUT], [NSUB-IN-OUT] and [NSUB-OUT-IN] is similar;
- inductive case [NSUB-CONT-IN]. Let $\mathcal{T}(\mathbb{T}) = \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathcal{T}(\mathbb{T}_i)$ and $\mathcal{T}(\mathbb{T}') = \bigwedge_{j\in J} \mathsf{p}?\ell'_j(S'_j).\mathcal{T}\left(\mathbb{T}'_j\right)$. For the sake of simplicity of the presentation, we show the proof for the case $\mathbb{T} \equiv \mu\mathbf{t}. \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathbb{T}_i$ and $\mathbb{T}' \equiv \bigwedge_{j\in J} \mathsf{p}?\ell'_j(S'_j).\mathbb{T}'_j$ with $k \in I$ and $s \in J$ such that $\ell_k = \ell'_s$ and $\mathcal{T}(\mathbb{T}_k\{^{\mathbb{T}}/_{\mathbf{t}}\}) \not\triangleleft \mathcal{T}(\mathbb{T}'_s)$. Other cases are similar. By the induction hypothesis, there is a failing derivation $(J'_1, \ldots, J'_m)$ of $\emptyset \vdash \mathbb{T}_k\{^{\mathbb{T}}/_{\mathbf{t}}\} \leqslant_a \mathbb{T}'_s$, where $J'_l = \Theta'_{l'} \vdash \mathbb{T}_{l'} \leqslant_a \mathbb{T}'_{l'}$ for $l' = 1, \ldots, m$. We can prove by Definition 3.27 and Lemma B.1 that the sequence $(J_1, \ldots, J_{m+3})$ with

$$
\begin{aligned}
J_1 &= \emptyset \vdash \mathbb{T} \leqslant_a \mathbb{T}' \\
J_2 &= \{(\mathbb{T}, \mathbb{T}')\} \vdash \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathbb{T}_i\{^{\mathbb{T}}/_{\mathbf{t}}\} \leqslant_a \bigwedge_{j\in J} \mathsf{p}?\ell'_j(S'_j).\mathbb{T}'_j \\
J_3 &= \{(\mathbb{T}, \mathbb{T}')\} \vdash \mathsf{p}?\ell_k(S_k).\mathbb{T}_k\{^{\mathbb{T}}/_{\mathbf{t}}\} \leqslant_a \mathsf{p}?\ell_k(S'_s).\mathbb{T}'_s \\
J_4 &= \{(\mathbb{T}, \mathbb{T}')\} \vdash \mathbb{T}_k\{^{\mathbb{T}}/_{\mathbf{t}}\} \leqslant_a \mathbb{T}'_s \\
J_{3+l'} &= \{(\mathbb{T}, \mathbb{T}')\} \cup \Theta'_{l'} \vdash \mathbb{T}_{l'} \leqslant_a \mathbb{T}'_{l'}, l' = 2, .., m.
\end{aligned}
$$

  is a failing derivation of $\emptyset \vdash \mathbb{T} \leqslant_a \mathbb{T}'$. (Notice that the general case would have instead of (J_1, J_2) the same components $(J_1, \ldots, J_{l_1+l_2+1})$ as in the base case, just with different processes $\mathbb{T}_{2l_1}$ and $\mathbb{T}'_{2l_2}$.) □

Lemma 4.6. *Let $(J_1, \ldots, J_n)$ be a failing derivation of $\emptyset \vdash \mathbb{T} \leqslant_a \mathbb{T}'$, where J_m has the form $\Theta_m \vdash \mathbb{T}_m \leqslant_a \mathbb{T}'_m$, $m \in \{1, \ldots, n\}$. Then, $\mathcal{T}(\mathbb{T}_m) \not\triangleleft \mathcal{T}(\mathbb{T}'_m)$ for every $m \in \{1, \ldots, n\}$.*

Proof. The proof is by induction on m, in decreasing order from n to 1:

- base case $m = n$: in this case, $(\mathbb{T}_m, \mathbb{T}'_m)$ has one of the forms listed in Corollary 3.30. We directly deduce $\mathcal{T}(\mathbb{T}) \not\triangleleft \mathcal{T}(\mathbb{T}')$ by applying one of the rules: [NSUB-ENDL], [NSUB-ENDR], [NSUB-DIFF-IN], [NSUB-DIFF-OUT], [NSUB-OUT-IN], [NSUB-IN-OUT], [NSUB-LABEL-IN], [NSUB-LABEL-OUT], [NSUB-EXCH-IN], or [NSUB-EXCH-OUT];
- inductive case $m < n$: by induction hypothesis, it holds that $\mathcal{T}(\mathbb{T}_{m+1}) \not\triangleleft \mathcal{T}(\mathbb{T}'_{m+1})$. By Definition 3.29(2), $\Theta_{m+1} \vdash \mathbb{T}_{m+1} \leqslant_a \mathbb{T}'_{m+1} \in \Phi(\Theta_m \vdash \mathbb{T}_m \leqslant_a \mathbb{T}'_m)$; therefore, by Definition 3.27, $(\mathbb{T}_m, \mathbb{T}'_m)$ has one of the following forms:
  - $(\mathbb{T}_m, \mu\mathbf{t}.\mathbb{T}''_m)$:
    then, $(\mathcal{T}(\mathbb{T}_m), \mathcal{T}(\mathbb{T}'_m)) = (\mathcal{T}(\mathbb{T}_m), \mathcal{T}(\mu\mathbf{t}.\mathbb{T}''_m)) = (\mathcal{T}(\mathbb{T}_m), \mathcal{T}(\mathbb{T}''_m\{^{\mu\mathbf{t}.\mathbb{T}''_m}/_{\mathbf{t}}\})) = (\mathcal{T}(\mathbb{T}_{m+1}), \mathcal{T}(\mathbb{T}'_{m+1}))$, and from the i.h. we conclude $\mathcal{T}(\mathbb{T}_m) \not\triangleleft \mathcal{T}(\mathbb{T}'_m)$;

- $(\mu \mathbf{t}.\mathbb{T}''_m, \mathbb{T}'_m)$:
  then, $(\mathcal{T}(\mathbb{T}_m), \mathcal{T}(\mathbb{T}'_m)) = (\mathcal{T}(\mu \mathbf{t}.\mathbb{T}''_m), \mathcal{T}(\mathbb{T}'_m)) = (\mathcal{T}(\mathbb{T}''_m\{^{\mu \mathbf{t}.\mathbb{T}''_m}/_{\mathbf{t}}\}), \mathcal{T}(\mathbb{T}'_m)) = (\mathcal{T}(\mathbb{T}_{m+1}), \mathcal{T}(\mathbb{T}'_{m+1}))$, and from the i.h. we conclude $\mathcal{T}(\mathbb{T}) \not\triangleleft \mathcal{T}(\mathbb{T}')$;
- $(\bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathbb{T}_i, \bigvee_{i\in J} \mathsf{p}!\ell_i(S'_i).\mathbb{T}'_i)$:
  then, $(\mathcal{T}(\mathbb{T}_m), \mathcal{T}(\mathbb{T}'_m)) = (\bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathcal{T}(\mathbb{T}_i), \bigvee_{i\in J} \mathsf{p}!\ell_i(S'_i).\mathcal{T}(\mathbb{T}'_i))$ and $\mathcal{T}(\mathbb{T}_{m+1}) \not\triangleleft \mathcal{T}(\mathbb{T}'_{m+1})$ with $m+1 \in I \cap J$. Hence, by the i.h. and [NSUB-CONT-OUT], we conclude $\mathcal{T}(\mathbb{T}_m) \not\triangleleft \mathcal{T}(\mathbb{T}'_m)$;
- $(\bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathbb{T}_i, \bigwedge_{j\in J} \mathsf{p}?\ell_j(S'_j).\mathbb{T}'_j)$:
  then, $(\mathcal{T}(\mathbb{T}_m), \mathcal{T}(\mathbb{T}'_m)) = (\bigwedge_{i\in I} \mathsf{p}!\ell_i(S_i).\mathcal{T}(\mathbb{T}_i), \bigwedge_{j\in J} \mathsf{p}!\ell_i(S'_i).\mathcal{T}(\mathbb{T}'_i))$ and $\mathcal{T}(\mathbb{T}_{m+1}) \not\triangleleft \mathcal{T}(\mathbb{T}'_{m+1})$ with $m+1 \in I \cap J$. Hence, by the i.h. and [NSUB-CONT-IN], we conclude $\mathcal{T}(\mathbb{T}_m) \not\triangleleft \mathcal{T}(\mathbb{T}'_m)$. □

B.2. Characteristic global types and characteristic contexts

Lemma B.2 *(Strengthening). If* $\Gamma, X : \mathsf{T}' \vdash P : \mathsf{T}$ *and* $X \notin \mathrm{fv}\{P\}$ *then* $\Gamma \vdash P : \mathsf{T}$.

Lemma B.3 *(Weakening). If* $\Gamma \vdash P : \mathsf{T}$ *and* $X \notin \mathrm{dom}(\Gamma)$ *then* $\Gamma, X : \mathsf{T}' \vdash P : \mathsf{T}$.

Lemma B.4. *If* $\Gamma, X_{\mathbf{t}} : \mathsf{T}_1 \vdash \mathcal{P}(\mathbb{T}) : \mathcal{T}(\mathbb{T}\sigma)$ *where* $\mathsf{T}_1 = \mathcal{T}(\mathbb{T}_1)$ *(for some* $\mathbb{T}_1$*), and* $\sigma = \{^{\Gamma(X_{\mathbf{t}'})}/_{\mathbf{t}'} \mid \mathbf{t}' \in \mathrm{fv}(\mathbb{T})\}$, *then* $\Gamma, X_{\mathbf{t}} : \mathsf{T}_2 \vdash \mathcal{P}(\mathbb{T}) : \mathcal{T}(\mathbb{T}\sigma')$, *for any* $\mathsf{T}_2 = \mathcal{T}(\mathbb{T}_2)$ *(for some* $\mathbb{T}_2$*) and* $\sigma' = (\sigma \setminus \{^{\mathbb{T}_1}/_{\mathbf{t}}\}) \cup \{^{\mathbb{T}_2}/_{\mathbf{t}}\}$.

Proof. By induction on the structure of $\mathbb{T}$. □

Lemma B.5. *For every (possibly open) type* $\mathbb{T}$, *there is* Γ *and* σ *such that* $\mathrm{dom}(\Gamma) = \{X_{\mathbf{t}} \mid \mathbf{t} \in \mathrm{fv}(\mathbb{T})\}$ *and* $\Gamma \vdash \mathcal{P}(\mathbb{T}) : \mathcal{T}(\mathbb{T}\sigma)$, *where* σ *is a substitution such that* $\sigma = \{^{\mathbb{T}_{\mathbf{t}}}/_{\mathbf{t}} \mid \mathbf{t} \in \mathrm{fv}(\mathbb{T})$ *and* $\Gamma(X_{\mathbf{t}}) = \mathcal{T}(\mathbb{T}_{\mathbf{t}})\}$.

Proof. By induction on the structure of $\mathbb{T}$.

- $\mathbb{T} \equiv \mathtt{end}$: $\mathcal{P}(\mathtt{end}) = \mathbf{0}$ and, by [T-**0**], $\vdash \mathcal{P}(\mathtt{end}) : \mathtt{end}$.
- $\mathbb{T} \equiv \mathbf{t}$: $\mathcal{P}(\mathbf{t}) = X_{\mathbf{t}}$
  By [T-VAR], $X_{\mathbf{t}} : \mathcal{T}(\mathbb{T}') \vdash X_{\mathbf{t}} : \mathcal{T}(\mathbb{T}')$ for any $\mathbb{T}'$. For $\sigma = \{^{\mathbb{T}'}/_{\mathbf{t}}\}$, we have
  $$X_{\mathbf{t}} : \mathcal{T}(\mathbb{T}') \vdash \mathcal{P}(\mathbf{t}) : \mathcal{T}(\mathbb{T}\sigma)\,.$$
- $\mathbb{T} \equiv \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathbb{T}_i$: $\mathcal{P}(\mathbb{T}) = \sum_{i\in I} \mathsf{p}?\ell_i(x_i).\mathsf{if}\ \mathtt{expression}(x_i, S_i)\ \mathsf{then}\ \mathcal{P}(\mathbb{T}_i)\ \mathsf{else}\ \mathcal{P}(\mathbb{T}_i)$
  By the induction hypothesis, $\Gamma_i \vdash \mathcal{P}(\mathbb{T}_i) : \mathcal{T}(\mathbb{T}_i\sigma_i)$, where $\sigma_i = \{^{\mathbb{T}_{\mathbf{t}}}/_{\mathbf{t}} \mid \mathbf{t} \in \mathrm{fv}(\mathbb{T}_i)$ and $\mathcal{T}(\mathbb{T}_{\mathbf{t}}) = \Gamma_i(X_{\mathbf{t}})\}$ for some Γ_i, and every $i \in I$. Let us denote by Γ the environment consisting of assignments $X_{\mathbf{t}} : \mathcal{T}(\mathbb{T}_{\mathbf{t}})$ for arbitrarily chosen $\mathbb{T}_{\mathbf{t}}$, where $X_{\mathbf{t}} \in \mathrm{dom}(\Gamma_i)$ for some $i \in I$. By Lemma 4.9, $\Gamma, x_i : S_i \vdash \mathtt{expression}(x_i, S_i) : \mathtt{bool}$, for every $i \in I$. By Lemma B.4, [T-CHOICE] and weakening, for each $i \in I$, we have the judgements:
  $$\Gamma, x_i : S_i \vdash \mathsf{if}\ \mathtt{expression}(x_i, S_i)\ \mathsf{then}\ \mathcal{P}(\mathbb{T}_i)\ \mathsf{else}\ \mathcal{P}(\mathbb{T}_i) : \mathcal{T}(\mathbb{T}_i\sigma'_i)\ \text{ where }\ \sigma'_i = \left\{ {}^{\mathbb{T}_{\mathbf{t}}}/_{\mathbf{t}} \;\middle|\; \begin{array}{l} \mathbf{t} \in \mathrm{fv}(\mathbb{T}_i) \text{ and} \\ \mathcal{T}(\mathbb{T}_{\mathbf{t}}) = \Gamma(X_{\mathbf{t}}) \end{array} \right\}$$
  Now, by [T-INPUT-CHOICE], we have
  $$\Gamma \vdash \sum_{i\in I} \mathsf{p}?\ell(x_i).\mathsf{if}\ \mathtt{expression}(x_i, S_i)\ \mathsf{then}\ \mathcal{P}(\mathbb{T}_i)\ \mathsf{else}\ \mathcal{P}(\mathbb{T}_i) : \bigwedge_{i\in I} \mathsf{p}?\ell(S_i).\mathcal{T}(\mathbb{T}_i\sigma'_i)\,.$$
  We conclude this case by remarking that
  $$\bigwedge_{i\in I} \mathsf{p}?\ell(S_i).\mathcal{T}(\mathbb{T}_i\sigma'_i) = \mathcal{T}(\mathbb{T}\sigma) \qquad \text{for } \sigma = \cup_{i\in I}\sigma'_i = \{^{\mathbb{T}_{\mathbf{t}}}/_{\mathbf{t}} \mid \mathbf{t} \in \mathrm{fv}(\mathbb{T}) \text{ and } \mathcal{T}(\mathbb{T}_{\mathbf{t}}) = \Gamma(X_{\mathbf{t}})\}.$$
- $\mathbb{T} \equiv \bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathbb{T}_i$: $\mathcal{P}(\mathbb{T}) = \bigoplus_{i\in I} \mathsf{p}!\ell_i(\mathtt{value}(S_i)).\mathcal{P}(\mathbb{T}_i)$.
  By the induction hypothesis, $\Gamma_i \vdash \mathcal{P}(\mathbb{T}_i) : \mathcal{T}(\mathbb{T}_i\sigma_i)$, where $\sigma_i = \{^{\mathbb{T}_{\mathbf{t}}}/_{\mathbf{t}} \mid \mathbf{t} \in \mathrm{fv}(\mathbb{T}_i)$ and $\mathcal{T}(\mathbb{T}_{\mathbf{t}}) = \Gamma_i(X_{\mathbf{t}})\}$ for some Γ_i, and every $i \in I$. Let us denote by Γ the environment consisting of assignments $X_{\mathbf{t}} : \mathcal{T}(\mathbb{T}_{\mathbf{t}})$ for arbitrarily chosen $\mathbb{T}_{\mathbf{t}}$, where $X_{\mathbf{t}} \in \mathrm{dom}(\Gamma_i)$ for some $i \in I$. By Lemma B.4 and weakening, for each $i \in I$, we have the judgement:
  $$\Gamma \vdash \mathcal{P}(\mathbb{T}_i) : \mathcal{T}(\mathbb{T}_i\sigma'_i) \quad \text{where }\ \sigma'_i = \left\{ {}^{\mathbb{T}_{\mathbf{t}}}/_{\mathbf{t}} \;\middle|\; \begin{array}{l} \mathbf{t} \in \mathrm{fv}(\mathbb{T}_i) \\ \text{and }\ \mathcal{T}(\mathbb{T}_{\mathbf{t}}) = \Gamma(X_{\mathbf{t}}) \end{array} \right\}$$

Since $\mathsf{p}!\ell_i(S_i).\mathcal{T}(\mathbb{T}_i\sigma_i') \leqslant \bigvee_{i\in I}\mathsf{p}!\ell_i(S_i).\mathcal{T}(\mathbb{T}_i\sigma_i')$, for every $i \in I$, we get by [T-OUT] and [T-SUB] that

$$\Gamma \vdash \mathsf{p}!\ell_i(\mathtt{value}(S_i)).\mathcal{P}(\mathbb{T}_i) : \bigvee_{i\in I}\mathsf{p}!\ell_i(S_i).\mathcal{T}(\mathbb{T}_i\sigma_i')\,.$$

By successive application of [T-CHOICE], we conclude

$$\Gamma \vdash \bigoplus_{i\in I}\mathsf{p}!\ell_i(\mathtt{value}(S_i)).\mathcal{P}(\mathbb{T}_i) : \mathcal{T}(\mathbb{T}\sigma) \qquad \text{where } \sigma = \cup_{i\in I}\sigma_i'$$

- $\mathbb{T} \equiv \mu\mathbf{t}.\mathbb{T}'$: $\mathcal{P}(\mathbb{T}) = \mu X_{\mathbf{t}}.\mathcal{P}(\mathbb{T}')$
  By induction hypothesis, there is Γ' such that

  $$\Gamma' \vdash \mathcal{P}(\mathbb{T}') : \mathcal{T}(\mathbb{T}'\sigma') \qquad \text{where } \sigma' = \{\mathbb{T}_{\mathbf{t}'}/\mathbf{t}' \mid \mathbf{t}' \in \mathrm{fv}(\mathbb{T}') \text{ and } \mathcal{T}(\mathbb{T}_{\mathbf{t}'}) = \Gamma'(X_{\mathbf{t}'})\}$$

  We have two cases:
  (i) $\mathbf{t} \notin \mathrm{fv}(\mathbb{T}')$. In this case, $X_{\mathbf{t}} \notin \mathrm{fv}(\mathcal{P}(\mathbb{T}'))$ and $\Gamma'', X_{\mathbf{t}} : \mathcal{T}(\mathbb{T}'') \vdash \mathcal{P}(\mathbb{T}') : \mathcal{T}(\mathbb{T}'\sigma')$, for some $\mathbb{T}''$ (either $\Gamma' = \Gamma'', X_{\mathbf{t}} : \mathcal{T}(\mathbb{T}'')$ or it is obtained by weakening of Γ'). By [T-REC], we get $\Gamma'' \vdash \mu X_{\mathbf{t}}.\mathcal{P}(\mathbb{T}') : \mathcal{T}(\mathbb{T}'\sigma)$ with $\sigma' = \sigma$.
  (ii) $\mathbf{t} \in \mathrm{fv}(\mathbb{T}')$. In this case, $X_{\mathbf{t}} \in \mathrm{fv}(\mathcal{P}(\mathbb{T}'))$ and $\Gamma' = \Gamma'', X_{\mathbf{t}} : \mathcal{T}(\mathbb{T}'')$ for some $\mathbb{T}''$. By Lemma B.4,

  $$\Gamma'', X_{\mathbf{t}} : \mathcal{T}(\mathbb{T}'\sigma') \vdash \mathcal{P}(\mathbb{T}') : \mathcal{T}(\mathbb{T}'\sigma'')$$
  $$\text{where} \quad \sigma'' = \left\{\mathbb{T}_{\mathbf{t}'}/\mathbf{t}' \;\middle|\; \mathbf{t}' \in \mathrm{fv}(\mathbb{T}') \setminus \{\mathbf{t}\} \text{ and } \mathcal{T}(\mathbb{T}_{\mathbf{t}'}) = \Gamma''(X_{\mathbf{t}'})\right\} \cup \{\mathbb{T}'\sigma'/\mathbf{t}\}$$

  i.e., the difference between σ' and σ'' is that σ'' contains $\mathbb{T}'\sigma'/\mathbf{t}$ instead of $\mathbb{T}''/\mathbf{t}$. Then, by [T-REC], we conclude

  $$\Gamma'' \vdash \mu X_{\mathbf{t}}.\mathcal{P}(\mathbb{T}') : \mathcal{T}(\mathbb{T}'\sigma) \quad \text{where } \sigma = \{\mathbb{T}_{\mathbf{t}}/\mathbf{t} \mid \mathbf{t} \in \mathrm{fv}(\mathbb{T}) \text{ and } \mathcal{T}(\mathbb{T}_{\mathbf{t}}) = \Gamma''(X_{\mathbf{t}})\} = \sigma'' \setminus \{\mathbb{T}'\sigma'/\mathbf{t}\} \qquad \square$$

Proposition 4.10. *For all $\mathbb{T}$, it holds that $\vdash \mathcal{P}(\mathbb{T}) : \mathcal{T}(\mathbb{T})$.*

Proof. Direct consequence of Lemma B.5 (recalling that $\mathbb{T}$ is here assumed to be closed, cf. Definition 3.4). $\square$

Lemma 4.14. *For all session types $\mathbb{T}$ and $\mathsf{p} \notin \mathtt{pt}\{\mathbb{T}\}$:*

1. $\mathcal{T}(\mathcal{G}(\mathbb{T}, \mathsf{p}))\upharpoonright\mathsf{p} = \mathcal{T}(\mathbb{T})$;
2. $\mathcal{T}(\mathcal{G}(\mathbb{T}, \mathsf{p}))\upharpoonright\mathsf{q}$ *is defined for all* $\mathsf{q} \in \mathtt{pt}\{\mathbb{T}\}$.

Proof.

Item 1. We define a relation $\mathcal{R}$ containing the type trees $\mathcal{T}(\mathbb{T})$ and $\mathcal{T}(\mathcal{G}(\mathbb{T}, \mathsf{p}))\upharpoonright\mathsf{p} = \mathcal{G}_0(\mathbb{T}, \mathsf{p}, \mathtt{pt}\{\mathbb{T}\})$ (for any $\mathbb{T}$ and $\mathsf{p} \notin \mathtt{pt}\{\mathbb{T}\}$), and show that $\mathcal{R}$ is a session type tree isomorphism (Definition A.16). Let:

$$\mathcal{R} = \left\{ \left(\mathcal{T}(\mathbb{T}')\,,\, \mathcal{T}\big(\mathcal{G}_0(\mathbb{T}', \mathsf{p}, \mathtt{pt}\{\mathbb{T}\})\big)\upharpoonright\mathsf{p}\right) \;\middle|\; \mathsf{p} \notin \mathtt{pt}\{\mathbb{T}'\} \right\} \tag{1}$$

We proceed by inspecting each pair $(\mathsf{T}, \mathsf{T}') \in \mathcal{R}$, and showing that it satisfies the clauses of Definition A.16. We have the following cases:

- $\mathsf{T} = \mathcal{T}(\mathtt{end})$. Then, we have $\mathcal{T}(\mathcal{G}(\mathbb{T}, \mathsf{p})) = \mathtt{end}$, which means $\mathsf{T}' = \mathcal{T}(\mathcal{G}(\mathbb{T}, \mathsf{p}))\upharpoonright\mathsf{p} = \mathcal{T}(\mathtt{end})$. Hence, we conclude that the pair $(\mathsf{T}, \mathsf{T}')$ satisfies clause (i) and (vacuously) all other clauses of Definition A.16;
- $\mathsf{T} = \mathcal{T}(\mathbb{T}') = \bigwedge_{i\in I}\mathsf{p}_{j_0}?\ell_i(S_i).\mathcal{T}(\mathbb{T}_i')$, with $\mathtt{pt}\{\mathbb{T}\} = \{\mathsf{p}_j\}_{1\leq j\leq n}$.In this case, the pair $(\mathsf{T}, \mathsf{T}')$ vacuously satisfies all clauses of Definition A.16 — except for clause (ii), that we now prove. Observe that, since $\mathbb{T}'$ is a (possibly recursive) external choice, by Definition 4.11 we have:

  $$\mathsf{T}' = \mathcal{T}\Big(\mathsf{p}_{j_0} \to \mathsf{p} : \{\ell_i(S_i).\mathbb{G}_i^{j_0}\}_{i\in I}\Big)\upharpoonright\mathsf{p} \qquad \text{with } \mathbb{G}^{j_0} \text{ from Table 9}$$

  which, by Definition 3.6, corresponds to:

  $$\mathsf{T}' = \bigwedge_{i\in I}\mathsf{p}_{j_0}?\ell_i(S_i).\left(\mathcal{T}\left(\mathbb{G}_i^{j_0}\right)\upharpoonright\mathsf{p}\right) \tag{2}$$

  where, by Table 9, each $\mathbb{G}_i^{j_0}$ $(i \in I)$ consists of a sequence of communications *not* involving p (by (1)) that are skipped by projection (Definition 3.6), and lead to $\mathcal{G}_0(\mathbb{T}_i', \mathsf{p}, \mathtt{pt}\{\mathbb{T}\})$. Therefore, $\forall i \in I$, $\mathcal{T}\left(\mathbb{G}_i^{j_0}\right)\upharpoonright\mathsf{p} = \mathcal{T}\big(\mathcal{G}_0(\mathbb{T}_i', \mathsf{p}, \mathtt{pt}\{\mathbb{T}\})\big)\upharpoonright\mathsf{p}$, with $\mathsf{p} \notin \mathtt{pt}\{\mathbb{T}_i'\}$ — and thus, by (1), we have:

  $$\forall i \in I: \quad \left(\mathcal{T}(\mathbb{T}_i')\,,\, \mathcal{T}\big(\mathcal{G}_0(\mathbb{T}_i', \mathsf{p}, \mathtt{pt}\{\mathbb{T}\})\big)\right) \in \mathcal{R} \tag{3}$$

  and by (2) and (3), we obtain that the pair $(\mathsf{T}, \mathsf{T}')$ satisfies clause ii and (vacuously) all other clauses of Definition A.16;

• $\mathsf{T} = \mathcal{T}(\mathbb{T}') = \bigvee_{i\in I} \mathsf{p}_{j_0}?\ell_i(S_i).\mathcal{T}(\mathbb{T}'_i)$, with $\mathtt{pt}\{\mathbb{T}\} = \{\mathsf{p}_j\}_{1\le j\le n}$. Similar to the previous case.

We have thus shown that $\mathcal{R}$ in (1) is a session type isomorphism, containing the pair $\left(\mathcal{T}(\mathbb{T}) \ , \ \mathcal{T}(\mathcal{G}(\mathbb{T},\mathsf{p}))\upharpoonright\mathsf{p}\right)$. From this, by Definition A.16 we get $\mathcal{T}(\mathbb{T}) = \mathcal{T}(\mathcal{G}(\mathbb{T},\mathsf{p}))\upharpoonright\mathsf{p}$, and by symmetry (Proposition A.2), we obtain the thesis.

Item 2. We need to show that $\mathcal{T}(\mathcal{G}(\mathbb{T},\mathsf{p}))\upharpoonright\mathsf{q} = \mathcal{T}(\mathcal{G}_0(\mathbb{T},\mathsf{p},\mathtt{pt}\{\mathbb{T}\}))\upharpoonright\mathsf{q}$ is defined for all $\mathsf{q}\in\mathtt{pt}\{\mathbb{T}\}$. We prove a more general statement: $\mathcal{T}(\mathcal{G}_0(\mathbb{T},\mathsf{p},\mathcal{S}))\upharpoonright\mathsf{q}$ is defined, for any set of participants $\mathcal{S} = \{\mathsf{p}_j\}_{1\le j\le n} \supseteq \mathtt{pt}\{\mathbb{T}\}$, and for any $\mathsf{q}\in\mathcal{S}$. To this purpose, we first define a relation $\mathcal{R}$ between types and type trees: intuitively, the image of $\mathbb{T}$ in $\mathcal{R}$ is a set of type trees that over-approximate the one yielded by $\mathcal{T}(\mathcal{G}(\mathbb{T},\mathsf{p}))\upharpoonright\mathsf{q}$ (when the latter is defined). Fix q, and let $\mathcal{R}$ be the largest relation such that, whenever the pair $(\mathbb{T},\mathsf{T})$ belongs to $\mathcal{R}$, we have:

$$\mathcal{T}(\mathbb{T}) = \mathcal{T}(\mathtt{end}) \quad \text{implies } \mathsf{T} = \mathcal{T}(\mathtt{end})$$

$$\mathcal{T}(\mathbb{T}) = \textstyle\bigwedge_{i\in I} \mathsf{q}?\ell_i(S_i).\mathcal{T}(\mathbb{T}_i) \quad \text{implies } \begin{cases} \exists \mathsf{T}_i\ (\forall i\in I) : \mathsf{T} = \bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathsf{T}_i \\ \qquad \text{and } \forall i\in I : (\mathbb{T}_i,\mathsf{T}_i)\in\mathcal{R} \\ \text{or } \exists \mathsf{r},\mathsf{r}',\mathsf{T}_i\ (\forall i\in I) : \\ \qquad \mathsf{T} = \bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathsf{r}!\ell_i(\mathtt{bool}).\mathsf{r}'?\ell_i(\mathtt{bool}).\mathsf{T}_i \\ \qquad \text{and } \forall i\in I : (\mathbb{T}_i,\mathsf{T}_i)\in\mathcal{R} \end{cases}$$

$$\mathcal{T}(\mathbb{T}) = \textstyle\bigwedge_{i\in I} \mathsf{q}'?\ell_i(S_i).\mathcal{T}(\mathbb{T}_i) \ \ (\mathsf{q}'\neq\mathsf{q}) \quad \text{implies } \begin{cases} \exists \mathsf{r},\mathsf{r}',\mathsf{T}_i\ (\forall i\in I) : \\ \qquad \mathsf{T} = \bigsqcap_{i\in I} \mathsf{r}?\ell_i(\mathtt{bool}).\mathsf{r}'!\ell_i(\mathtt{bool}).\mathsf{T}_i \\ \qquad \text{and } \forall i\in I : (\mathbb{T}_i,\mathsf{T}_i)\in\mathcal{R} \end{cases} \tag{4}$$

$$\mathcal{T}(\mathbb{T}) = \textstyle\bigvee_{i\in I} \mathsf{q}!\ell_i(S_i).\mathcal{T}(\mathbb{T}_i) \quad \text{implies } \begin{cases} \exists \mathsf{T}_i\ (\forall i\in I) : \mathsf{T} = \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i \\ \qquad \text{and } \forall i\in I : (\mathbb{T}_i,\mathsf{T}_i)\in\mathcal{R} \\ \text{or } \exists \mathsf{r},\mathsf{r}',\mathsf{T}_i\ (\forall i\in I) : \\ \qquad \mathsf{T} = \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{r}!\ell_i(\mathtt{bool}).\mathsf{r}'?\ell_i(\mathtt{bool}).\mathsf{T}_i \\ \qquad \text{and } \forall i\in I : (\mathbb{T}_i,\mathsf{T}_i)\in\mathcal{R} \end{cases}$$

$$\mathcal{T}(\mathbb{T}) = \textstyle\bigvee_{i\in I} \mathsf{q}'!\ell_i(S_i).\mathcal{T}(\mathbb{T}_i) \ \ (\mathsf{q}'\neq\mathsf{q}) \quad \text{implies } \begin{cases} \exists \mathsf{r},\mathsf{r}',\mathsf{T}_i\ (\forall i\in I) : \\ \qquad \mathsf{T} = \bigsqcap_{i\in I} \mathsf{r}?\ell_i(\mathtt{bool}).\mathsf{r}'!\ell_i(\mathtt{bool}).\mathsf{T}_i \\ \qquad \text{and } \forall i\in I : (\mathbb{T}_i,\mathsf{T}_i)\in\mathcal{R} \end{cases}$$

We now show that, if we take $\mathcal{R}$ satisfying (4) for a given q:

(S1) $\mathcal{R}$ is left-total, i.e., for all $\mathbb{T}$, there is T such that $(\mathbb{T},\mathsf{T})\in\mathcal{R}$. To see why, notice that the only potential cases where, for some $\mathbb{T}$, there is no T such that $(\mathbb{T},\mathsf{T})\in\mathcal{R}$ are those where $\mathbb{T}$ is an internal/external choice to/from a participant $\mathsf{q}'\neq\mathsf{q}$: in such cases, (4) requires T to be a (potentially undefined) merging $\sqcap$ of type trees. But in such cases, T is obtained by merging singleton external choices from a same role r, with non-overlapping labels ℓ_i $(i\in I)$: therefore, the merging is always defined, by [MRG-BRA] in Definition 3.6;

(S2) if $\mathsf{T} = \mathcal{T}(\mathcal{G}_0(\mathbb{T},\mathsf{p},\mathcal{S}))\upharpoonright\mathsf{q}$ is defined (for some $\mathcal{S} = \{\mathsf{p}_j\}_{1\le j\le n} \supseteq \mathtt{pt}\{\mathbb{T}\}$), then $(\mathbb{T},\mathsf{T})\in\mathcal{R}$ — i.e., $\mathcal{R}$ relates $\mathbb{T}$ with the projection onto q of its characteristic global type. This is proved below;

(S3) if (by contradiction) we assume that $\mathcal{T}(\mathcal{G}_0(\mathbb{T},\mathsf{p},\mathcal{S}))\upharpoonright\mathsf{q}$ is *un*defined (for some $\mathbb{T}$ and $\mathcal{S} = \{\mathsf{p}_j\}_{1\le j\le n} \supseteq \mathtt{pt}\{\mathbb{T}\}$), then we obtain a contradiction. This is proved below;

(S4) hence, there is no $\mathbb{T}$ that satisfies the assumption of item *(S3)* above: this proves the thesis.

We now prove *(S2)*. Fix q, and consider the relation:

$$\mathcal{R}' \ = \ \{\ (\mathbb{T},\mathsf{T}) \ \mid \ \mathsf{T} = \mathcal{T}(\mathcal{G}_0(\mathbb{T},\mathsf{p},\mathcal{S}))\upharpoonright\mathsf{q} \ \text{ with } \ \mathsf{q}\in\mathcal{S}\supseteq\mathtt{pt}\{\mathbb{T}\}\ \} \tag{5}$$

We now show that $\mathcal{R}'$ satisfies the clauses of (4), which implies $\mathcal{R}'\subseteq\mathcal{R}$ (since $\mathcal{R}$ is the largest relation satisfying such clauses). For each pair $(\mathbb{T},\mathsf{T})\in\mathcal{R}'$ and the corresponding $\mathcal{S}$, we have the cases:

• $\mathcal{T}(\mathbb{T}) = \mathtt{end}$. Then, $\mathsf{T} = \mathcal{T}(\mathcal{G}_0(\mathbb{T},\mathsf{p},\mathcal{S}))\upharpoonright\mathsf{q} = \mathcal{T}(\mathtt{end})$, as required by (4);

• $\mathcal{T}(\mathbb{T}) = \bigwedge_{i\in I} \mathsf{p}_{j_0}?\ell_i(S_i).\mathcal{T}(\mathbb{T}_i)$, for some $\mathsf{p}_{j_0}\in\mathcal{S}$. Then, by Definition 4.11, we have:

$$\mathsf{T} \ = \ \mathcal{T}\Big(\mathsf{p}_{j_0} \to \mathsf{p} : \{\ell_i(S_i).\mathbb{G}_i^{j_0}\}_{i\in I}\Big)\upharpoonright\mathsf{q} \qquad \text{with } \mathbb{G}^{j_0} \text{ from Table 9} \tag{6}$$

and two sub-cases:

– $\mathsf{p}_{j_0} = \mathsf{q}$. Then, from (6) and Definition 3.6, we have either:

$$\mathsf{T} = \bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathsf{T}_i \qquad \text{where } \forall i\in I : \mathsf{T}_i = \mathcal{T}\Big(\mathbb{G}_i^{j_0}\Big)\upharpoonright\mathsf{q} \tag{7}$$

$$\text{or } \exists \mathsf{r},\mathsf{r}' : \ \mathsf{T} = \bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathsf{r}!\ell_i(\mathtt{bool}).\mathsf{r}'?\ell_i(\mathtt{bool}).\mathsf{T}_i \qquad \text{where } \forall i\in I : \mathsf{T}_i = \mathcal{T}\Big(\mathbb{G}_i^{j_0}\Big)\upharpoonright\mathsf{q} \tag{8}$$

where case (7) holds when $\mathcal{S} = \{\mathsf{q}\}$, while case (8) holds when $\mathcal{S}\supsetneq\{\mathsf{q}\}$ (we have no further cases, since $\mathsf{q}\in\mathcal{S}$ by (5)). Now, notice that, by Table 9, each $\mathsf{G}_i^{j_0}$ $(i\in I)$ consists of a sequence of communications *not*

involving p, that are skipped by projection (Definition 3.6), and lead to $\mathcal{G}_0(\mathbb{T}_i, \mathsf{p}, \mathcal{S})$: therefore, $\forall i \in I$, we have $\mathsf{T}_i = \mathcal{T}\left(\mathbb{G}_i^{j_0}\right)\upharpoonright\mathsf{q} = \mathcal{T}(\mathcal{G}_0(\mathbb{T}_i, \mathsf{p}, \mathcal{S}))\upharpoonright\mathsf{q}$. This means that, by (5) we have $\forall i \in I : (\mathbb{T}_i, \mathsf{T}_i) \in \mathcal{R}'$ — and this, together with either (7) or (8), satisfies the clauses of (4);

– $\mathsf{p}_{j_0} \neq \mathsf{q}$. In this case, by Definition 3.6, we have:

$$\mathsf{T} = \prod_{i \in I} \left(\mathcal{T}\left(\mathbb{G}_i^{j_0}\right)\upharpoonright\mathsf{q}\right) \tag{9}$$

Notice that, by Table 9, each $\mathsf{G}_i^{j_0}$ $(i \in I)$ consists of a sequence of communications where q *first* receives a message with label ℓ_i from some participant r, and immediately resends it to some other participant r' (with $\mathsf{r}, \mathsf{r}' \in \mathcal{S}$, by clause $\mathcal{S} \supseteq \mathtt{pt}\{\mathbb{T}\}$ in (5) and Table 9). More in detail, we have:

$$\exists \mathsf{r}, \mathsf{r}' : \forall i \in I : \mathcal{T}\left(\mathsf{G}_i^{j_0}\right)\upharpoonright\mathsf{q} = \mathsf{r}?\ell_i(\mathtt{bool}).\mathsf{r}'!\ell_i(\mathtt{bool}).\mathsf{T}_i \quad \text{where} \quad \mathsf{T}_i = \mathcal{T}(\mathcal{G}_0(\mathbb{T}_i, \mathsf{p}, \mathcal{S}))\upharpoonright\mathsf{q} \tag{10}$$

This means that, by (5) we have $\forall i \in I : (\mathbb{T}_i, \mathsf{T}_i) \in \mathcal{R}'$ — and this, together with (9) and (10), satisfies the clauses of (4);

• $\mathcal{T}(\mathbb{T}) = \bigvee_{i \in I} \mathsf{p}_{j_0}?\ell_i(S_i).\mathcal{T}(\mathbb{T}_i)$, for some $\mathsf{p}_{j_0} \in \mathcal{S}$. Similar to the previous case.

We have thus proved $\mathcal{R}' \subseteq \mathcal{R}$; and since $\forall \mathbb{T}$, if $\mathsf{T} = \mathcal{T}(\mathcal{G}_0(\mathbb{T}, \mathsf{p}, \mathcal{S}))\upharpoonright\mathsf{q}$ is defined then $(\mathbb{T}, \mathsf{T}) \in \mathcal{R}' \subseteq \mathcal{R}$, we conclude that *(S2)* holds.

We now prove *(S3)*. By contradiction, assume that there is some $\mathbb{T}$ such that $\mathcal{T}(\mathcal{G}_0(\mathbb{T}, \mathsf{p}, \mathcal{S}))\upharpoonright\mathsf{q}$ is *un*defined (for some $\mathcal{S} = \{\mathsf{p}_j\}_{1 \leq j \leq n} \supseteq \mathtt{pt}\{\mathbb{T}\}$). This can only occur because the projection onto q requires an undefined merging. By Definitions 3.6 and 4.11, merging is only used when an (unfolded) subterm $\mathbb{T}'$ of $\mathbb{T}$ is an internal/external choice to/from a participant $\mathsf{p}_{j_0} \neq \mathsf{q}$: this leads to a merging involving some participants $\mathsf{r}, \mathsf{r}' \in \mathcal{S}$, similarly to (9) and (10) in the proof of *(S2)* above — except that the merging is now *un*defined. But if such a merging is undefined for r and r', then it is also undefined if we replace r, r' with any other pair of participants: this is because the merging rules of Definition 3.6 do not depend on the participants' identities. Now, recall that $\mathcal{R}$ is left-total by *(S1)*: therefore, there is some T' such that $(\mathbb{T}', \mathsf{T}') \in \mathcal{R}$ — however, such T' *cannot* be obtained by merging the type trees as described in (4): this is because, under our initial (contradictory) assumption, we know that such merging is always undefined, so we cannot satisfy the existential quantifications "$\exists \mathsf{r}, \mathsf{r}' \ldots$" in the clauses of (4) that use $\sqcap$. Thus, the pair $(\mathbb{T}', \mathsf{T}') \in \mathcal{R}$ does *not* satisfy the clauses in (4), i.e., $\mathcal{R}$ is *not* the largest relation whose elements satisfy such clauses — contradiction. Therefore, as per *(S4)*, we conclude that $\mathcal{T}(\mathcal{G}_0(\mathbb{T}, \mathsf{p}, \mathcal{S}))\upharpoonright\mathsf{q}$ is defined for all $\mathbb{T}$: this proves the thesis. □

Proposition 4.16. *Take any* $\mathbb{T}'$ *and* $\mathsf{r} \notin \mathtt{pt}\{\mathbb{T}'\}$. *If* $\vdash Q : \mathcal{T}(\mathbb{T}')$, *then* $\vdash C_{\mathsf{r},\mathbb{T}'_1}[\mathsf{r} \triangleleft Q] : \mathcal{T}(\mathcal{G}(\mathbb{T}', \mathsf{r}))$, *for any characteristic context* $C_{\mathsf{r},\mathbb{T}'_1}$ *such that* $\mathcal{T}(\mathbb{T}') = \mathcal{T}(\mathbb{T}'_1)$.

Proof. There are two cases.

- $\mathbb{T}' = \mathtt{end}$: If $\vdash Q : \mathtt{end}$ then $Q = \mathbf{0}$ and, by [T-SESS], $\vdash \mathsf{r} \triangleleft \mathbf{0} : \mathtt{end}$.
- $\mathtt{pt}\{\mathbb{T}'\} = \mathtt{pt}\{\mathbb{T}'_1\} = \{\mathsf{p}_1, \ldots, \mathsf{p}_n\}, n \geq 1$:
  $C_{\mathsf{r},\mathbb{T}'_1}[\mathsf{r} \triangleleft Q] = \mathsf{r} \triangleleft Q \mid \prod_{1 \leq i \leq n} \mathsf{p}_i \triangleleft \mathcal{P}(\mathbb{T}_i),\ \mathcal{T}(\mathbb{T}_i) = \mathcal{T}(\mathcal{G}(\mathbb{T}'_1, \mathsf{r}))\upharpoonright\mathsf{p}_i = \mathcal{T}(\mathcal{G}(\mathbb{T}', \mathsf{r}))\upharpoonright\mathsf{p}_i$
  By Lemma 4.14(2), $\mathcal{T}(\mathbb{T}'_1) = \mathcal{T}(\mathcal{G}(\mathbb{T}', \mathsf{r}))\upharpoonright\mathsf{r}$.
  By Proposition 4.10, $\vdash \mathcal{P}(\mathbb{T}_i) : \mathcal{T}(\mathbb{T}_i)$ i.e. $\vdash \mathcal{P}(\mathbb{T}_i) : \mathcal{T}(\mathcal{G}(\mathbb{T}', \mathsf{r}))\upharpoonright\mathsf{p}_i$.
  By [T-SESS], $\vdash C_{\mathsf{r},\mathbb{T}'_1}[\mathsf{r} \triangleleft Q] : \mathcal{T}(\mathcal{G}(\mathbb{T}', \mathsf{r}))$. □

References

[1] S. Bakel, M. Dezani-Ciancaglini, U. de' Liguoro, Y. Motohama, The Minimal Relevant Logic and the Call-by-Value Lambda Calculus, Technical Report TR-ARP-05-2000, The Australian National University, 2000.
[2] F. Barbanera, U. de'Liguoro, Two notions of sub-behaviour for session-based client/server systems, in: PPDP, ACM, 2010, pp. 155–164.
[3] F. Barbanera, U. de'Liguoro, Sub-behaviour relations for session-based client/server systems, Math. Struct. Comput. Sci. 25 (2015) 1339–1381, https://doi.org/10.1017/S096012951400005X.
[4] H. Barendregt, M. Coppo, M. Dezani-Ciancaglini, A filter lambda model and the completeness of type assignment, J. Symb. Log. 48 (1983) 931–940, https://doi.org/10.2307/2273659.
[5] M. Bartoletti, A. Scalas, R. Zunino, A semantic deconstruction of session types, in: CONCUR, Springer, 2014, pp. 402–418.
[6] G. Bernardi, A. Francalanza, Full-abstraction for must testing preorders - (extended abstract), in: COORDINATION, 2017, pp. 237–255.
[7] G. Bernardi, M. Hennessy, Using higher-order contracts to model session types, Log. Methods Comput. Sci. 12 (2016), https://doi.org/10.2168/LMCS-12(2:10)2016.
[8] M. Bravetti, M. Carbone, G. Zavattaro, Undecidability of asynchronous session subtyping, Inf. Comput. 256 (2017) 300–320, https://doi.org/10.1016/j.ic.2017.07.010.
[9] M. Bravetti, M. Carbone, G. Zavattaro, On the boundary between decidability and undecidability of asynchronous session subtyping, Theor. Comput. Sci. 722 (2018) 19–51, https://doi.org/10.1016/j.tcs.2018.02.010.
[10] G. Castagna, R. De Nicola, D. Varacca, Semantic subtyping for the pi-calculus, Theor. Comput. Sci. 398 (2008) 217–242.

[11] G. Castagna, M. Dezani-Ciancaglini, E. Giachino, L. Padovani, Foundations of session types, in: PPDP, ACM, 2009, pp. 219–230.
[12] G. Castagna, A. Frisch, A gentle introduction to semantic subtyping, in: ICALP, Springer, 2005, pp. 30–34.
[13] G. Castagna, N. Gesbert, L. Padovani, A theory of contracts for web services, ACM Trans. Program. Lang. Syst. 31 (2009), https://doi.org/10.1145/1538917.1538920, 19:1–19:61.
[14] T.C. Chen, M. Dezani-Ciancaglini, A. Scalas, N. Yoshida, On the preciseness of subtyping in session types, Log. Methods Comput. Sci. 13 (2017) 1–62.
[15] T.C. Chen, M. Dezani-Ciancaglini, N. Yoshida, On the preciseness of subtyping in session types, in: PPDP, ACM Press, 2014, pp. 135–146.
[16] M. Coppo, M. Dezani-Ciancaglini, L. Padovani, N. Yoshida, A gentle introduction to multiparty asynchronous session types, in: SFM, Springer, 2015, pp. 146–178.
[17] M. Coppo, M. Dezani-Ciancaglini, N. Yoshida, L. Padovani, Global progress for dynamically interleaved multiparty sessions, Math. Struct. Comput. Sci. 26 (2016) 238–302, https://doi.org/10.1017/S0960129514000188.
[18] R. Demangeon, K. Honda, Full abstraction in a subtyped pi-calculus with linear types, in: CONCUR, Springer, 2011, pp. 280–296.
[19] P. Deniélou, N. Yoshida, A. Bejleri, R. Hu, Parameterised multiparty session types, Log. Methods Comput. Sci. 8 (2012), https://doi.org/10.2168/LMCS-8(4:6)2012.
[20] M. Dezani-Ciancaglini, U. de'Liguoro, A. Piperno, A filter model for concurrent lambda-calculus, SIAM J. Comput. 27 (1998) 1376–1419, https://doi.org/10.1137/S0097539794275860.
[21] M. Dezani-Ciancaglini, S. Ghilezan, Preciseness of subtyping on intersection and union types, in: RTATLCA, Springer, 2014, pp. 194–207.
[22] M. Dezani-Ciancaglini, S. Ghilezan, S. Jaksic, J. Pantovic, N. Yoshida, Precise subtyping for synchronous multiparty sessions, in: PLACES, 2015, pp. 29–43.
[23] M. Dezani-Ciancaglini, S. Ghilezan, S. Jaksic, J. Pantovic, N. Yoshida, Denotational and operational preciseness of subtyping: a roadmap, in: Theory and Practice of Formal Methods: Essays Dedicated to Frank de Boer on the Occasion of His 60th Birthday, Springer, 2016, pp. 155–172.
[24] A. Frisch, G. Castagna, V. Benzaken, Semantic subtyping: dealing set-theoretically with function, union, intersection, and negation types, J. ACM 55 (2008) 1–64.
[25] S. Gay, M. Hole, Subtyping for session types in the pi calculus, Acta Inform. 42 (2005) 191–225, https://doi.org/10.1007/s00236-005-0177-z.
[26] S.J. Gay, 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, Springer International Publishing, 2016, pp. 95–108.
[27] S.J. Gay, M. Hole, Types and subtypes for client–server interactions, in: ESOP, 1999, pp. 74–90.
[28] R. Harper, Practical Foundations for Programming Languages, Cambridge University Press, 2013.
[29] M. Hennessy, The Distributed π-Calculus, Cambridge University Press, 2007.
[30] J.R. Hindley, The completeness theorem for typing lambda-terms, Theor. Comput. Sci. 22 (1983) 1–17, https://doi.org/10.1016/0304-3975(83)90136-6.
[31] K. Honda, M. Tokoro, An object calculus for asynchronous communication, in: ECOOP, Springer, 1991, pp. 133–147.
[32] K. Honda, V.T. Vasconcelos, M. Kubo, Language primitives and type disciplines for structured communication-based programming, in: ESOP, Springer, 1998, pp. 22–138.
[33] K. Honda, N. Yoshida, On reduction-based process semantics, Theor. Comput. Sci. 151 (1995) 437–486.
[34] K. Honda, N. Yoshida, M. Carbone, Multiparty asynchronous session types, in: POPL, ACM Press, 2008, pp. 273–284.
[35] K. Honda, N. Yoshida, M. Carbone, Multiparty asynchronous session types, J. ACM 63 (2016) 1–67.
[36] R. Hu, N. Yoshida, Hybrid session verification through endpoint API generation, in: FASE, Springer, 2016, pp. 401–418.
[37] R. Hu, N. Yoshida, Explicit connection actions in multiparty session types, in: FASE, Springer, 2017, pp. 116–133.
[38] H. Ishihara, T. Kurata, Completeness of intersection and union type assignment systems for call-by-value lambda-models, Theor. Comput. Sci. 272 (2002) 197–221, https://doi.org/10.1016/S0304-3975(00)00351-0.
[39] D. Kouzapas, N. Yoshida, Globally governed session semantics, in: CONCUR, Springer, 2013, pp. 395–409.
[40] D. Kouzapas, N. Yoshida, Globally governed session semantics, Log. Methods Comput. Sci. 10 (2015).
[41] J. Lange, N. Yoshida, On the undecidability of asynchronous session subtyping, in: FoSSaCs, 2017, pp. 441–457.
[42] J. Ligatti, J. Blackburn, M. Nachtigal, On subtyping-relation completeness, with an application to iso-recursive types, ACM Trans. Program. Lang. Syst. 39 (2017) 4:1–4:36, https://doi.org/10.1145/2994596.
[43] J. Ligatti, J. Blackburn, M. Nachtigal, On subtyping-relation completeness, with an application to iso-recursive types, ACM Trans. Program. Lang. Syst. 39 (2017) 4:1–4:36, https://doi.org/10.1145/2994596.
[44] B. Liskov, Keynote address – data abstraction and hierarchy, ACM SIGPLAN Not. 23 (1988) 17–34.
[45] B.H. Liskov, J.M. Wing, A behavioral notion of subtyping, TOPLAS 16 (1994) 1811–1841, https://doi.org/10.1145/197320.197383.
[46] R. Milner, Communication and Concurrency, Prentice-Hall, Inc., 1989.
[47] R. Milner, J. Parrow, D. Walker, A Calculus of Mobile Processes, Parts I and II, Inf. Comput. 100 (1992) 1–77.
[48] R. Milner, D. Sangiorgi, Barbed bisimulation, in: ICALP, Springer, 1992, pp. 685–695.
[49] D. Mostrous, N. Yoshida, K. Honda, Global principal typing in partially commutative asynchronous sessions, in: ESOP, Springer, 2009, pp. 316–332.
[50] R. Neykova, N. Yoshida, Let it recover: multiparty protocol-induced recovery, in: Compiler Construction, ACM, 2017, pp. 98–108.
[51] N. Ng, J.G. de Figueiredo Coutinho, N. Yoshida, Protocols by default – safe MPI code generation based on session types, in: Compiler Construction, Springer, 2015, pp. 212–232.
[52] N. Ng, N. Yoshida, Pabble: parameterised scribble, Serv. Oriented Comput. Appl. 9 (2015) 269–284, https://doi.org/10.1007/s11761-014-0172-8.
[53] B.C. Pierce, Types and Programming Languages, MIT Press, 2002.
[54] B.C. Pierce, D. Sangiorgi, Typing and subtyping for mobile processes, Math. Struct. Comput. Sci. 6 (1996) 409–453.
[55] D. Sangiorgi, D. Walker, The π-Calculus: a Theory of Mobile Processes, Cambridge University Press, 2001.
[56] A. Scalas, O. Dardha, R. Hu, N. Yoshida, A linear decomposition of multiparty sessions for safe distributed programming, in: ECOOP, Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2017, pp. 24:1–24:31.
[57] A. Scalas, N. Yoshida, Less is more: multiparty session types revisited, ACM Program. Lang. 3 (POPL) (2019) 30, https://doi.org/10.1145/3290343, to appear, preprint: http://mrg.doc.ic.ac.uk/publications/less-is-more-multiparty-session-types-revisited/preprint.pdf.
[58] A. Scalas, N. Yoshida, Less is more: multiparty session types revisited (artifact). https://doi.org/10.1145/3291638. Peer-reviewed artifact of [57] (to appear). Latest version available at https://alcestes.github.io/mpstk, 2019.
[59] J. Vouillon, Subtyping union types, in: CSL, 2004, pp. 415–429.
[60] N. Yoshida, P. Deniélou, A. Bejleri, R. Hu, Parameterised multiparty session types, in: FOSSACS, Springer, 2010, pp. 128–145.
[61] N. Yoshida, R. Hu, R. Neykova, N. Ng, The scribble protocol language, in: TGC, Springer, 2013, pp. 22–41.