Fair Subtyping for Multi-Party Session Types

Luca Padovani

Dipartimento di Informatica, Università di Torino, Italy
padovani@di.unito.it

Abstract. The standard subtyping relation used in dyadic session type theories may compromise the liveness of multi-party sessions. In this paper we define a *fair* subtyping relation for multi-party session types that preserves liveness, we relate it with the standard subtyping relation, and we give algorithms for deciding it. As a side effect, we provide an original and remarkably simple coinductive characterization of the fair testing preorder for nondeterministic, sequential processes consisting of internal choices of outputs and external choices of inputs.

1 Introduction

Type systems for dyadic sessions [15,16,22] require that, at any time, the two ends of a session must be used by exactly two processes and in complementary ways. These requirements enforce session correctness, namely *communication safety* (no message of unexpected type is ever sent) and *liveness* (whenever a message is exchanged, all of the processes involved in the session make progress). For example, the session $\mathtt{p} : T \mid \mathtt{q} : R$, where T and R are the session types defined by

$$T = \mathtt{q}!a.T \oplus \mathtt{q}!b.\mathsf{end} \qquad \text{and} \qquad R = \mathtt{p}?a.R + \mathtt{p}?b.\mathsf{end}\,,$$

is correct and describes a conversation between two processes identified by the tags $\mathtt{p}$ and $\mathtt{q}$: process $\mathtt{p}$ sends either an a message or a b message to $\mathtt{q}$; the decision as to which type of message is sent is taken by $\mathtt{p}$, whence the *internal choice* operator $\oplus$. Process $\mathtt{q}$ must be ready to receive either an a or a b message from $\mathtt{p}$, whence the *external choice* operator $+$. If an a message is exchanged, the two processes repeat this pattern; as soon as a b message is exchanged, the session ends.

The shift from dyadic to *multi-party sessions* [17] makes the definition of session correctness more subtle. First, it is no longer obvious what it means to use the ends of the session "in complementary ways" if the session involves more than two participants. Second, it is no longer reasonable to pretend that *all* of the involved participants make progress whenever a message is exchanged if communications are point-to-point and yet one would like to state that no participant is left behind. A natural formalization of correctness for multi-party sessions requires that, at any time, the session must have the possibility to reach a terminal configuration where all of its participants no longer use the session ends. For example, in the session $\mathtt{p} : T' \mid \mathtt{q} : R \mid \mathtt{r} : \mathtt{p}?c.\mathsf{end}$, where

$$T' = \mathtt{q}!a.T' \oplus \mathtt{q}!b.\mathtt{r}!c.\mathsf{end}\,,$$

the processes $\mathtt{p}$ and $\mathtt{q}$ may exchange an arbitrary number of a messages and, during their interaction, the process $\mathtt{r}$ does not make any progress. However, the session is

correct because, as long as a messages are exchanged, it is always *possible* (although not granted) for $\mathtt{p}$ to send a b message to $\mathtt{q}$ followed by a c message to $\mathtt{r}$. If this happens, all of the involved participants reach a terminal state and the session ends.

This difference between dyadic and multi-party sessions has dramatic effects on the subtyping relation for session types [12,6]. Subtyping defines an asymmetric compatibility between types such that, when T is a subtype of S, it is harmless to replace a channel with type S with another one with type T or, equivalently, it is harmless to replace a process that behaves according to T with another one that behaves according to S. For example, the session type T defined above is a subtype of $\mathtt{q}!b.\mathsf{end}$: using a channel of type $\mathtt{q}!b.\mathsf{end}$ means sending a b message to process $\mathtt{q}$. Since the session type T permits sending both an a message and a b message, using a channel with type T in place of another one with type $\mathtt{q}!b.\mathsf{end}$ does not compromise the correctness of the session. In general, we may deduce that T is a subtype of S if S is a variant of T where some branches of some internal choices have been pruned. According to this intuition every session type in the family

$$S_2 = \mathtt{q}!a.\mathtt{q}!a.S_2 \oplus \mathtt{q}!b.\mathsf{end} \quad \cdots \quad S_n = (\mathtt{q}!a.)^n S_n \oplus \mathtt{q}!b.\mathsf{end} \quad \cdots \quad S_\infty = \mathtt{q}!a.S_\infty$$

is a supertype of T. The type S_n allows sending a b message only after the number of sent a messages is a multiple of n. The type S_∞ is somehow the limit of the sequence $\{S_i\}_{i\geq 2}$ and describes a process that only sends a messages. The fact that T is a subtype of S_∞ may be questionable, because the sessions $\mathtt{p} : S_i \mid \mathtt{q} : R$ for $i \geq 2$ all have the potential to terminate (it is always possible that a b message is sent), while the session $\mathtt{p} : S_\infty \mid \mathtt{q} : R$ is doomed to loop forever. In a dyadic session like $\mathtt{p} : S_\infty \mid \mathtt{q} : R$ this is mitigated by the observation that *every* participant of the session makes indefinite progress. However, using the same arguments we might also deduce that S_∞ is a supertype of T', and now in the session $\mathtt{p} : S_\infty \mid \mathtt{q} : R \mid \mathtt{r} : \mathtt{p}?c.\mathsf{end}$ process $\mathtt{p}$ keeps interacting with $\mathtt{q}$ while c is stuck waiting for a message that is never sent. We conclude that the well-known subtyping relation for dyadic session types is unsound in multi-party theories because it may not preserve the liveness of multi-party sessions.

In this paper we study a sound subtyping relation for multi-party session types. Understanding when two session types are related by subtyping in our theory is a surprisingly complex business. First of all, the differences between the standard subtyping relation and ours emerge only when recursive session types are involved, while the two relations coincide on finite session types. Second, unlike the standard subtyping relation for session types, deciding whether some branch of an internal choice can be safely pruned may involve a non-local check on the structure of the session types being compared. This makes the subtyping relation particularly difficult to axiomatize. To illustrate the subtleties behind our subtyping relation, consider the session types T, S_2, and S_∞ represented as the three automata in Figure 1, where the initial states have been labelled with the name of the session type and the solid arcs with the actions performed by the processes that behave according to these types. The subtyping relation establishes a correspondence between states of two session types. In the figure, the correspondence is depicted as the three dotted arrows showing, for each state of S_2, the corresponding state of T. The fact that S_∞ *is not* a supertype of T can be easily detected since no end state is reachable from S_∞, but this does not explain why S_2 *is* a supertype of T. Observe

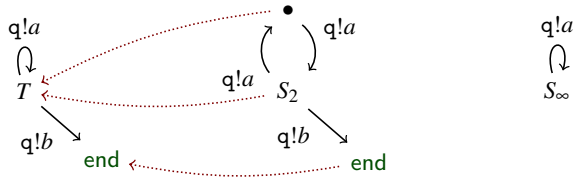

Fig. 1. Relation between $T = \mathtt{q}!a.T \oplus \mathtt{q}!b.\mathsf{end}$ and $S_2 = \mathtt{q}!a.\mathtt{q}!a.S_2 \oplus \mathtt{q}!b.\mathsf{end}$.

that S_2 has an intermediate state $\bullet$ which lacks the outgoing $\mathtt{q}!b$-labelled transition that T has. The correspondence between T and this state of S_2 is safe if (and only if) there is no session type R such that $\mathtt{p} : T \mid \mathtt{q} : R$ is a correct session and $\mathtt{q}$ is capable to loop the interaction starting from $\mathtt{p} : S_2 \mid \mathtt{q} : R$ in such a way that the $\bullet$ state is visited infinitely often. If this were the case, $\mathtt{q}$ could rely on the observation of a b message after having received an odd number of a messages to terminate successfully. This cannot happen in the example above because $\mathtt{p} : S_2$ can always break the loop by sending $\mathtt{q}$ an a message followed by a b one (the act of sending a message is irrevocably decided by the sender). We express this as the fact that S_2 *rules over* (every context, like $\mathtt{q} : R$, that completes) T, which we denote by $T \prec S_2$.

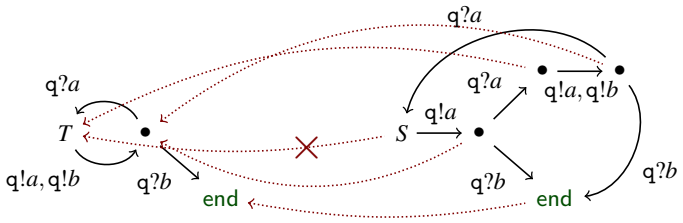

Fig. 2. Relation between $T = \mathtt{q}!a.(\mathtt{q}?a.T + \mathtt{q}?b.\mathsf{end}) \oplus \mathtt{q}!b.(\mathtt{q}?a.T + \mathtt{q}?b.\mathsf{end})$ and $S = \mathtt{q}!a.(\mathtt{q}?a.(\mathtt{q}!a.(\mathtt{q}?a.S + \mathtt{q}?b.\mathsf{end}) \oplus \mathtt{q}!b.(\mathtt{q}?a.S + \mathtt{q}?b.\mathsf{end})) + \mathtt{q}?b.\mathsf{end}$.

A more involved example is depicted in Figure 2. The only difference between T and S is that S lacks the outgoing $\mathtt{q}!b$-labelled transition that T has. Basically, $\mathtt{p} : S$ may send a b message only after an odd number of a messages have been sent to $\mathtt{q}$ and an equal number of a messages have been received. Unlike the previous example, it is $\mathtt{q}$ that decides whether to terminate the interaction with $\mathtt{p}$, by sending a b message, or to continue, by sending an a message. Consider now the participant $\mathtt{q} : R$ where

$$R = \mathtt{p}?a.\mathtt{p}!a.(\mathtt{p}?a.\mathtt{p}!a.R + \mathtt{p}?b.\mathtt{p}!a.R) + \mathtt{p}?b.\mathtt{p}!b.\mathsf{end}\,.$$

It is easy to see that $\mathtt{p} : T \mid \mathtt{q} : R$ is correct while $\mathtt{p} : S \mid \mathtt{q} : R$ loops through state S. In other words, $\mathtt{q}$ forces $\mathtt{p} : S$ to go through state S in hopes that a b message is received.

This was possible with $\mathtt{p} : T$, but not with $\mathtt{p} : S$. The fact that a participant like $\mathtt{q} : R$ exists means that T is not ruled by S, and therefore T is not a subtype of S. In this paper we show that the "ruled by" relation fully characterizes the contexts in which pruning outputs is safe.

Related work. The framework we have depicted is known in concurrency theory as *fair testing* [18,21]. Testing [10,9,14] is a general technique for defining refinement relations $\sqsubseteq$ between processes so that, when $P \sqsubseteq Q$ holds, the process Q can be safely used in place of process P because every "test" that P passes is passed also by Q. Fair testing adds a fairness assumption to standard testing: if a system goes infinitely often through a state from which some action is possible (like the action $\mathtt{q}!b$ from state T in Figure 2), a component of the system may rely upon the eventual observation of that action to terminate successfully. In the present paper, we instantiate fair testing to a context where processes are session types describing the behavior of participants of a multi-party session and the "test" is given by the correctness of a session.

Since the $\sqsubseteq$ relation is defined by universally quantifying over an infinite number of tests, a crucial aspect of every testing theory is the study of alternative, possibly effective characterizations of $\sqsubseteq$ or approximations of it. Alternative characterizations of refinements not considering fairness have been defined, for example, in [13,14] and later, in coinductive form, in [7] and in [4,1]. Alternative characterizations of fair refinements have already been given in the literature, but we find them unsatisfactory. The authors of [18] present a characterization based on sets of infinite strings, while [21] relies on a denotational model of processes. In both cases the characterizations are quite complex, if compared to those of corresponding unfair refinements, because they are semantically – rather than syntactically – based. In fact, as pointed out in [21], no complete axiomatization of these refinements is known at the present time. Recently, [2,3] have investigated subcontract relations for Web services which are closely related to fair subtyping of session types, but they refer to [21] when it comes to characterizing and deciding them. The authors of [4] provide a coinductive characterization that is not complete (for instance, it fails to assess that T is a subtype of S_2 in Figure 1). The standard reference for subtyping of session types is [12], where the subtyping relation is "unfair" by definition. A fair theory of multi-party session types has been developed in [19], but no alternative characterizations nor algorithms were given.

Contributions. This paper presents a self-contained theory of multi-party session types where the focus is on the eventual satisfaction of all the interacting participants. From a technical viewpoint, the main novelty is an alternative characterization of the fair subtyping relation which is expressed as the combination of the familiar, "unfair" subtyping relation [12] and a "ruled by" relation which can be expressed as a syntax-directed notion of behavioral difference between session types. This allows us to present a complete deduction system for the subtyping relation as a minor variation of the standard one, up to the use of the "ruled by" relation.

Structure of the paper. In Section 2 we formalize the language of (multi-party) session types, the notion of correct session, and subtyping as the relation that preserves correctness. We show that our subtyping differs from the standard one. Section 3 provides a

sound and complete coinductive characterization of subtyping based on the "ruled by" relation. Section 4 presents algorithms for deciding subtyping and related notions. Section 5 concludes. Proofs and auxiliary technical material are available in the appendix of the full version of the paper [20].

2 Syntax and Semantics of Session Types

We assume a set $\mathscr{R}$ of *role tags* ranged over by $\mathtt{p}$, $\mathtt{q}$, ..., a countable set $\mathscr{M}$ of *message types* ranged over by a, b, ..., and a countable set $\mathscr{X}$ of *recursion variables* ranged over by x, y, ... Table 1 defines the syntax of sessions and session types. *Sessions*, ranged over by M, N, ..., are finite compositions $\mathtt{p}_1 : T_1 \mid \cdots \mid \mathtt{p}_n : T_n$ made of a fixed number of participants that communicate with each other according to the session types T_i. We work exclusively with well-formed sessions, where each participant is uniquely identified by a tag $\mathtt{p}_i$ ($i \neq j$ implies $\mathtt{p}_i \neq \mathtt{p}_j$). Session types, ranged over by T, S, ..., are the closed terms generated by the grammar in Table 1 such that:

- every recursion variable is guarded by at least one (input or output) prefix, and
- in every subterm $\sum_{i\in I} \mathtt{p}?a_i.T_i$ or $\bigoplus_{i\in I} \mathtt{p}!a_i.T_i$ the a_i's are pairwise distinct.

The first condition forbids non-contractive session types such as $\mu x.x$, while the second condition ensures that session types are unambiguous by requiring that every prefix of the form $\mathtt{p}?a_i$ or $\mathtt{p}!a_i$ uniquely determines a continuation T_i. We consider session types modulo the folding and unfolding of recursive terms. Therefore, we assume $\mu x.T = T\{\mu x.T/x\}$ where $T\{\mu x.T/x\}$ denotes the session type obtained from T by replacing every free occurrence of x in T with $\mu x.T$ (μ is the only binder for recursion variables, and the notions of free and bound variables are defined as expected). In practice, this amounts to saying that session types are the possibly infinite, finitely-branching, regular trees [8] generated by the productions of the grammar in Table 1. Note that all the session types defined in the introduction can be finitely and uniquely expressed as possibly recursive terms generated by the grammar in Table 1.

Table 1. Syntax of session types and sessions.

$T ::=$	**Session Type**	$M ::=$	**Session**
fail	(failure)	$\mathtt{p} : T$	(participant)
$\mid \mathsf{end}$	(termination)	$\mid M \mid M$	(composition)
$\mid x$	(variable)		
$\mid \sum_{i\in I} \mathtt{p}?a_i.T_i$	(input)		
$\mid \bigoplus_{i\in I} \mathtt{p}!a_i.T_i$	(output)		
$\mid \mu x.T$	(recursion)		

The session type end describes a process that no longer participates to the session. The session type $\sum_{i\in I} \mathtt{p}?a_i.T_i$ describes a process that waits for a message from the source participant identified by tag $\mathtt{p}$: depending on the type a_i of the message it receives, the process behaves according to the continuation T_i. The session type

$\bigoplus_{i\in I} \mathtt{p}!a_i.T_i$ describes a process that internally decides to send a message of type a_i to the destination participant identified by tag $\mathtt{p}$. After the output operation the process behaves as described in the session type T_i. Terms x and $\mu x.T$ are used to build recursive session types. It is technically convenient (although not necessary) to have a canonical term fail describing *failed processes* that are unable to terminate successfully. This happens, for example, if a participant receives an unexpected message. Sometimes we will use the infix notation $\mathtt{p}?a_1.T_1 + \cdots + \mathtt{p}?a_n.T_n$ to denote $\sum_{i=1}^n \mathtt{p}?a_i.T_i$ and $\mathtt{p}!a_1.T_1 \oplus \cdots \oplus \mathtt{p}!a_n.T_n$ to denote $\bigoplus_{i=1}^n \mathtt{p}!a_i.T_i$. Note that mixed choices like $\mathtt{p}?a.T + \mathtt{p}!b.S$ and $\mathtt{p}?a.T + \mathtt{q}?a.S$ are forbidden. In particular, the source participant $\mathtt{p}$ and the destination participant $\mathtt{p}$ in $\sum_{i\in I} \mathtt{p}?a_i.T_i$ and $\bigoplus_{i=1}^n \mathtt{p}!a_i.T_i$ must be the same in all branches (all the examples in the introduction are consistent with these conventions). While slightly redundant, the syntax for inputs and outputs allows us to conveniently switch between the prefix forms and the corresponding infix forms. Also, we will write $\mathsf{trees}(T)$ for the finite set of subtrees that T is made of, including T itself (recall that a regular tree is made of a *finite* number of distinct subtrees [8]). Take for example $T = \mu x.(\mathtt{p}!a.\mathtt{q}?c.x \oplus \mathtt{p}!b.\mathsf{end})$. Then $\mathsf{trees}(T) = \{T, \mathtt{q}?c.T, \mathsf{end}\}$.

We express the evolution of a session by means of a transition system. The idea is that each participant of a session behaves as described by the corresponding session type and the session evolves by means of internal choices taken by the participants and by synchronizations occurring between them. Labels of the transition system, ranged over by $\hat\alpha$, are generated by the grammar

$$\hat\alpha \;::=\; \tau \;\mid\; \checkmark \;\mid\; \mathtt{p} : \mathtt{p}?a \;\mid\; \mathtt{p} : \mathtt{p}!a$$

and we use α to range over actions different from τ.

Table 2. Transition system of sessions.

$$\text{(T-SUCCESS)}\quad \mathtt{p} : \mathsf{end} \xrightarrow{\checkmark} \mathtt{p} : \mathsf{end} \qquad \text{(T-OUTPUT)}\quad \mathtt{p} : \mathtt{q}!a.T \xrightarrow{\mathtt{p}:\mathtt{q}!a} \mathtt{p} : T \qquad \text{(T-CHOICE)}\quad \frac{k \in I}{\mathtt{p} : \bigoplus_{i\in I} \mathtt{q}!a_i.T_i \xrightarrow{\tau} \mathtt{p} : \mathtt{q}!a_k.T_k}$$

$$\text{(T-INPUT)}\quad \frac{k \in I}{\mathtt{p} : \sum_{i\in I} \mathtt{q}?a_i.T_i \xrightarrow{\mathtt{p}:\mathtt{q}?a_k} \mathtt{p} : T_k} \qquad \text{(T-FAILURE)}\quad \frac{a \neq a_i \ ^{(i\in I)}}{\mathtt{p} : \sum_{i\in I} \mathtt{q}?a_i.T_i \xrightarrow{\mathtt{p}:\mathtt{q}?a} \mathtt{p} : \mathsf{fail}}$$

$$\text{(T-PAR ACTION)}\quad \frac{M \xrightarrow{\hat\alpha} M' \qquad \hat\alpha \neq \checkmark}{M \mid N \xrightarrow{\hat\alpha} M' \mid N} \qquad \text{(T-COMM)}\quad \frac{M \xrightarrow{\mathtt{p}:\mathtt{q}!a} M' \qquad N \xrightarrow{\mathtt{q}:\mathtt{p}?a} N'}{M \mid N \xrightarrow{\tau} M' \mid N'} \qquad \text{(T-PAR SUCCESS)}\quad \frac{M \xrightarrow{\checkmark} M \qquad N \xrightarrow{\checkmark} N}{M \mid N \xrightarrow{\checkmark} M \mid N}$$

Table 2 defines the transition system (symmetric rules omitted) in terms of a family of labelled relations $\xrightarrow{\hat\alpha}$. Rule (T-SUCCESS) states that end performs a $\checkmark$ action that

flags successful termination and reduces to itself. Rules (T-OUTPUT) and (T-CHOICE) deal with outputs. The former one shows that a participant $\mathtt{p}$ willing to send an a message to participant $\mathtt{q}$ performs a $\mathtt{p} : \mathtt{q}!a$ action. The latter one states that a participant that is ready to send any message from a set internally and irrevocably chooses one particular message to send. In both rules we use the abbreviation $\mathtt{q}!a_0.T_0$ for $\bigoplus_{i\in\{0\}} \mathtt{q}!a_i.T_i$. Rules (T-INPUT) and (T-FAILURE) deal with inputs. The former one is standard and states that a participant $\mathtt{p}$ performs $\mathtt{p} : \mathtt{q}?a$ actions according to the type of messages it is willing to receive and the participant $\mathtt{q}$ from which it expects these messages to come. The latter shows that a participant can receive an unexpected input, but in doing so it will fail. Note the fundamental asymmetry between inputs and outputs: a participant autonomously commits to sending *one* particular message by means of rule (T-CHOICE), while it retains the ability to receive *any* message from a given set by means of rule (T-INPUT). Rule (T-PAR ACTION) propagates transitions through compositions and (T-COMM) is the usual communication rule. Finally, (T-PAR SUCCESS) states that a composition has successfully terminated if all of its participants have. In the following we adopt the following conventions: we write $\stackrel{\tau}{\Longrightarrow}$ for the reflexive, transitive closure of $\stackrel{\tau}{\longrightarrow}$; we write $\stackrel{\alpha}{\Longrightarrow}$ for $\stackrel{\tau}{\Longrightarrow}\stackrel{\alpha}{\longrightarrow}\stackrel{\tau}{\Longrightarrow}$ and $\stackrel{\alpha_1\cdots\alpha_n}{\Longrightarrow}$ for the composition $\stackrel{\alpha_1}{\Longrightarrow}\cdots\stackrel{\alpha_n}{\Longrightarrow}$; we let s, t, ... range over finite strings of actions different from $\checkmark$; we write $M \stackrel{\alpha}{\longrightarrow}$ (respectively, $M \stackrel{\alpha}{\Longrightarrow}$) if there exists N such that $M \stackrel{\alpha}{\longrightarrow} N$ (respectively, $M \stackrel{\alpha}{\Longrightarrow} N$); we write $M \stackrel{\tau}{\not\longrightarrow}$ (respectively, $M \stackrel{\alpha}{\not\Longrightarrow}$) if there exists no N such that $M \stackrel{\tau}{\longrightarrow} N$ (respectively, $M \stackrel{\alpha}{\Longrightarrow} N$). We also extend the labelled transition relation and the above notation to session types so that, for example, $T \stackrel{\alpha}{\longrightarrow} S$ if $\mathtt{p} : T \stackrel{\alpha}{\longrightarrow} \mathtt{p} : S$ for some $\mathtt{p}$.

Intuitively, a session is correct if the possibility to reach a state where every participant is successfully terminated is invariant under reductions. This can be formalized as follows:

Definition 2.1 (correct session). *We say that M is* correct *if $M \stackrel{\tau}{\Longrightarrow} N$ implies $N \stackrel{\checkmark}{\Longrightarrow}$.*

In Section 1 we have already seen a number of correct sessions, which the reader may now formally check against Definition 2.1. It is useful to discuss a few examples of *incorrect* sessions. For instance, $M = \mathtt{p} : \mathtt{q}!a.\mathsf{end} \oplus \mathtt{q}!b.\mathsf{end} \mid \mathtt{q} : \mathtt{p}?a.\mathsf{end}$ is not correct because $\mathtt{p}$ may decide to send a b message that $\mathtt{q}$ is not willing to receive. Even though at the beginning of the interaction there is one potential path leading to successful termination (indeed $M \stackrel{\checkmark}{\Longrightarrow}$), rule (T-CHOICE) can make the decision of sending b irrevocable (in this case $M \stackrel{\tau}{\longrightarrow} \mathtt{p} : \mathtt{q}!b.\mathsf{end} \mid \mathtt{q} : \mathtt{p}?a.\mathsf{end} \stackrel{\tau}{\longrightarrow} \mathtt{p} : \mathsf{end} \mid \mathtt{q} : \mathsf{fail} \stackrel{\checkmark}{\not\Longrightarrow}$). There are also intrinsically flawed session types that can never be part of correct sessions. For example, the session $M \mid \mathtt{p} : \mathsf{fail}$ is incorrect regardless of M, because fail is never able to perform $\checkmark$. Some sessions are incorrect despite that no fail term occurs in them. This happens in the session $\mathtt{p} : \mu x.\mathtt{q}!a.x \mid \mathtt{q} : \mu y.\mathtt{p}?a.y$ because, even though the participants $\mathtt{p}$ and $\mathtt{q}$ keep interacting with each other, they do not have the ability to terminate the interaction.

Some properties of correct sessions are easy to verify: $\mathtt{p} : \mathsf{end}$ is the simplest correct session; the session $M \mid \mathtt{p} : \mathsf{end}$ is correct if and only if M is correct; finally, correctness is preserved by reductions: if M is correct and $M \Longrightarrow N$, then N is also correct.

We define the subtyping relation for session types semantically as the relation that preserves correctness: we say that T is a *subtype* of S if every session $M \mid \mathtt{p} : T$ that is correct remains correct when we replace T with S. Formally:

Definition 2.2 (subtyping). *We say that T is a* subtype *of S, written $T \leqslant S$, if $M \mid \mathtt{p} : T$ correct implies $M \mid \mathtt{p} : S$ correct for every M. We write $\lesseqgtr$ for the equivalence relation induced by $\leqslant$, namely $\lesseqgtr = \leqslant \cap \leqslant^{-1}$.*

This definition may look surprising at first, because it speaks about left-to-right substitutability (of behaviors), while subtyping is concerned with right-to-left substitutability (of channels). The mismatch is only apparent, however, and is due to the fact that session types are behavioral types (they describe the behavior of processes using channels). To clarify this point, suppose that S is the type associated with a channel c and that some process P uses c as indicated by S. By replacing channel c in P with another channel d with type $T \leqslant S$, we are changing the set of processes that P is interacting with, which together behave according to some M such that $M \mid \mathtt{p} : T$ is correct. Replacing c with d does *not* affect the way P behaves: P uses channel d (whose actual type is T) as if it were channel c (thus according to S). This means that the actual implemented session is $M \mid \mathtt{p} : S$. Since $T \leqslant S$, we know that this session is correct.

A thorough study of the subtyping relation that solely relies on Definition 2.2 is hard, because of the universal quantification over an infinite set of contexts M. Nonetheless, a few relations are easy to establish. For example, we have

$$\textbf{(i)}\ \mathtt{p}?a.\mathsf{end} \leqslant \mathtt{p}?a.\mathsf{end} + \mathtt{p}?b.\mathsf{end} \qquad \text{and} \qquad \textbf{(ii)}\ \mathtt{p}!a.\mathsf{end} \oplus \mathtt{p}!b.\mathsf{end} \leqslant \mathtt{p}!a.\mathsf{end}$$

namely $\leqslant$ behaves covariantly with respect to inputs and contravariantly with respect to outputs, *on finite session types*. The two relations can be explained as follows: in **(i)**, every context M such that $M \mid \mathtt{q} : \mathtt{p}?a.\mathsf{end}$ is correct must eventually send some message to $\mathtt{q}$, and this message can only be a for otherwise $\mathtt{q}$ would fail because of rule (T-FAILURE). Therefore, $M \mid \mathtt{q} : \mathtt{p}?a.\mathsf{end} + \mathtt{p}?b.\mathsf{end}$ is also correct, since $\mathtt{p}?a.\mathsf{end} + \mathtt{p}?b.\mathsf{end}$ is more receptive than $\mathtt{p}?a.\mathsf{end}$. In **(ii)**, every context M such that $M \mid \mathtt{q} : \mathtt{p}!a.\mathsf{end} \oplus \mathtt{p}!b.\mathsf{end}$ is correct must be able to terminate successfully no matter which message (either a or b) is sent to $\mathtt{p}$. One such context is $M = \mathtt{p} : \mathtt{q}?a.\mathsf{end} + \mathtt{q}?b.\mathsf{end}$. Therefore, nothing bad happens when we replace $\mathtt{p}!a.\mathsf{end} \oplus \mathtt{p}!b.\mathsf{end}$ with a more deterministic behavior, such as $\mathtt{p}!a.\mathsf{end}$. As a general note, observe that relation **(i)** *increases* (and relation **(ii)** *decreases*) the number of paths along the session types that lead to end when one reads the relations from left to right. Since correctness concerns the reachability of a successfully terminated state, it is not obvious that reducing the number of paths leading to end is generally safe, as we have already argued in the introduction.

The standard subtyping relation for session types [12], which we dub "unfair subtyping" to distinguish it from the one of Definition 2.2, is defined thus:

Definition 2.3 (unfair subtyping). *We say that $\mathscr{S}$ is a* coinductive subtyping *if $T \mathscr{S} S$ implies either:*

1. *$T = S = \mathsf{end}$, or*
2. *$T = \sum_{i \in I} \mathtt{p}?a_i.T_i$ and $S = \sum_{i \in I \cup J} \mathtt{p}?a_i.S_i$ and $T_i \mathscr{S} S_i$ for every $i \in I$, or*

3. $T = \bigoplus_{i\in I\cup J} \mathrm{p}!a_i.T_i$ *and* $S = \bigoplus_{i\in I} \mathrm{p}!a_i.S_i$ *and* $T_i \mathrel{\mathscr{S}} S_i$ *for every* $i \in I$.

Unfair subtyping*, denoted by* $\leqslant_{\mathsf{U}}$*, is the largest coinductive subtyping.*

Item (1) states that the only subtype of end is end. Item (2) is the standard *covariant* rule for input actions: it is safe for a process that is capable of handling a set $\{a_i\}_{i\in I\cup J}$ of incoming message types to wait for messages from a channel on which a subset $\{a_i\}_{i\in I}$ of message types can be received. Item (3) is dual of item (2) and deals with outputs. It states that a process can safely use a channel on which messages from the set $\{a_i\}_{i\in I\cup J}$ can be sent if it never sends a message that is not in this set.

The relation $\leqslant_{\mathsf{U}}$ is appealing because of its simple and intuitive definition, but it is neither sound nor complete if compared with $\leqslant$. On the one hand, the $\leqslant_{\mathsf{U}}$ relation does not preserve correctness as by Definition 2.1. For instance, the reader may verify that $T \leqslant_{\mathsf{U}} S_\infty$ holds for T and S_∞ defined in the introduction, but $T \not\leqslant S_\infty$ because S_∞ has no end subtree. On the other hand, there exists a large class of equivalent session types that are syntactically unrelated. For instance, we have $S_\infty \lesseqgtr \mathsf{fail}$ and $S_\infty \not\leqslant_{\mathsf{U}} \mathsf{fail}$. Session types like fail or S_∞ are flawed because there is no correct session in which they can occur. Therefore, they are the $\leqslant$-least elements and roughly correspond to the empty type in other type theories. Patching Definition 2.3 to take flawed session types into proper account is far from trivial (adding a case for dealing with fail session types is not enough, as S_∞ shows).

3 Coinductive Fair Subtyping

We devote this section to defining a complete, coinductive characterization of $\leqslant$. To ease the presentation, we proceed incrementally in three steps: **(1)** we introduce a normal form for session types that allows us to focus on the subclass of *viable* session types, those that can be part of correct sessions and that, consequently, are the most relevant in practice; **(2)** we express $T \leqslant S$ as the combination of two relations, the familiar (but unsafe) $T \leqslant_{\mathsf{U}} S$ subtyping for session types (which is shown to include $\leqslant$ when restricted to viable session types in normal form) and a $T \prec S$ relation that holds when the paths leading to successful termination in T that have disappeared from S do not endanger correctness; **(3)** we show that the $T \prec S$ relation is equivalent to the viability of a suitably defined $T - S$ session type, somehow representing the "behavioral difference" between T and S.

Normal form. At the end of Section 2 we have seen that there exist flawed session types that cannot occur in any correct session. Session types that *can* occur in correct sessions are our primary concern and we reserve a name for them.

Definition 3.1 (viability). *We say that* T *is* viable *if* $M \mid \mathrm{p} : T$ *is correct for some* M *and* p*. We write* $\mathscr{T}_{\mathsf{v}}$ *for the set of viable session types.*

A session type T is *not* viable if and only if $T \leqslant \mathsf{fail}$. That is, being not viable means being ($\leqslant$-smaller than) the empty type. The existence of non-viable session types hinders the coinductive characterization of the subtyping relation in the style of Definition 2.3 because these characterizations are based on the intuition that semantically

related session types must be syntactically similar, while we have shown that this is not necessarily true when non-viable session types are involved. We define a normal form that makes non-viable session types readily detectable and the syntax of viable ones meaningful in a sense that will be clarified shortly.

Definition 3.2 (normal form). *We say that T is in* normal form *if either $T = \mathsf{fail}$ or $\mathsf{end} \in \mathsf{trees}(S)$ for every $S \in \mathsf{trees}(T)$. We write $\mathscr{T}_{\mathsf{nf}}$ for the set of session types in normal form.*

The double indirection in Definition 3.2 imposes that an end leaf is included in *every* subtree of T when T is different from fail. For example $\mathtt{p}?a.\mathsf{end} + \mathtt{p}?b.\mathsf{fail}$ is not in normal form because $\mathsf{fail} \in \mathsf{trees}(\mathtt{p}?a.\mathsf{end} + \mathtt{p}?b.\mathsf{fail})$ and $\mathsf{end} \notin \mathsf{trees}(\mathsf{fail})$. The following proposition assures us that working with session types in normal forms is convenient and yet not restrictive: every session type has an $\leqslant$-equivalent one in normal form and every session type in normal form different from fail is viable.

Proposition 3.1. *The following properties hold: **(1)** for every $T \in \mathscr{T}$ there exists $S \in \mathscr{T}_{\mathsf{nf}}$ such that $T \leqslant S$; **(2)** $\mathscr{T}_{\mathsf{nf}} \setminus \{\mathsf{fail}\} \subseteq \mathscr{T}_{\mathsf{v}}$.*

For instance, $\mathtt{p}?a.\mathsf{end}$ is the normal form of $\mathtt{p}?a.\mathsf{end} + \mathtt{p}?b.\mathsf{fail}$. The syntax of session types in normal form is "meaningful" in the sense that $\leqslant_{\mathsf{U}}$ includes $\leqslant$ when we focus on viable session types in normal form.

Theorem 3.1. *Let $T, S \in \mathscr{T}_{\mathsf{nf}} \setminus \{\mathsf{fail}\}$. Then $T \leqslant S$ implies $T \leqslant_{\mathsf{U}} S$.*

Unfair subtyping and $\leqslant$ decomposition. Focusing on viable session types in normal form does not change the fact that $\leqslant_{\mathsf{U}}$ is unsound with respect to $\leqslant$. More precisely, $\leqslant_{\mathsf{U}}$ does not introduce deadlocks, but it can introduce livelocks when recursive session types are involved:

Theorem 3.2. *Let $T, S \in \mathscr{T}_{\mathsf{nf}}$ and $T \leqslant_{\mathsf{U}} S$. Then:*

1. *T recursion-free implies $T \leqslant S$;*
2. *$M \mid \mathtt{p} : T$ correct and $M \mid \mathtt{p} : S \stackrel{\tau}{\Longrightarrow} N \stackrel{\tau}{\not\longrightarrow}$ imply $N \stackrel{\checkmark}{\longrightarrow}$.*

Theorem 3.2 shows that $\leqslant_{\mathsf{U}}$ is not too far away from being a sound characterization of $\leqslant$. Therefore, we attempt at characterizing $T \leqslant S$ as the combination of two relations: $T \leqslant_{\mathsf{U}} S$, expressing a *safety* property (S does not introduce deadlocks), and $T \prec S$, expressing a *liveness* property (S does not preclude the successful termination of any context that completes T). The "ruled by" relation $\prec$ is defined thus:

Definition 3.3. *Let $T, S \in \mathscr{T}_{\mathsf{nf}}$ and $T \leqslant_{\mathsf{U}} S$. We say that T is* ruled by *S, written $T \prec S$, if $M \mid \mathtt{p} : T$ correct implies $M \mid \mathtt{p} : S \stackrel{\checkmark}{\Longrightarrow}$ for every M.*

When $T \leqslant_{\mathsf{U}} S$, the behavior S may preclude successful termination of a context M that completes T only when some outputs in T have disappeared in S. The additional property $T \prec S$ prevents this from happening. Observe that $T \leqslant S$ implies $T \prec S$, but the converse is not true in general. In fact, $\prec$ precisely captures the difference between $\leqslant_{\mathsf{U}}$ and $\leqslant$, in the following sense:

Definition 3.4 (coinductive fair subtyping). *A coinductive subtyping $\mathscr{S}$ is* fair *if $T\ \mathscr{S}\ S$ implies $T \prec S$. We write $\leqslant_{\mathsf{C}}$ for the largest coinductive fair subtyping.*

The relation $\leqslant_{\mathsf{C}}$ is indeed the characterization of $\leqslant$ we are looking for:

Theorem 3.3. *Let $T, S \in \mathscr{T}_{\mathsf{nf}} \setminus \{\mathsf{fail}\}$. Then $T \leqslant S$ if and only if $T \leqslant_{\mathsf{C}} S$.*

Characterization of $\prec$ and behavioral difference. We now shift the focus to the $\prec$ relation. Suppose $T \leqslant_{\mathsf{U}} S$ and $T \not\prec S$. Then there exists some context M such that the correctness of $M \,|\, \mathtt{p} : T$ crucially depends on the outputs that T emits and that S does not. In order to find M, we define a session type $T - S$ that somehow represents the "difference" between T and S and that is viable if (and only if) such M does exist. The intuition is that $T - S$ differs from T and S in three respects:

1. Every end that lies on a path shared by T and S is turned to a fail in $T - S$. Therefore, any hypothetical context M such that $M \,|\, \mathtt{p} : T - S$ is correct can only count on those end leaves found in T that have disappeared in S.
2. $T - S$ performs no more inputs than those performed by T. In this way we stay assured that, if M exists, it does not use any additional input capability provided by S but not by T.
3. $T - S$ performs all the outputs performed by T.

Formally:

Definition 3.5 (session type difference). *Let $T \leqslant_{\mathsf{U}} S$. The* difference *of T and S, denoted by $T - S$, is coinductively defined by the following equations:*

$$\begin{aligned} \mathsf{end} - \mathsf{end} &= \mathsf{fail} \\ \textstyle\sum_{i\in I} \mathtt{p}?a_i.T_i - \sum_{i\in I\cup J} \mathtt{p}?a_i.S_i &= \sum_{i\in I} \mathtt{p}?a_i.(T_i - S_i) \\ \textstyle\bigoplus_{i\in I\cup J} \mathtt{p}!a_i.T_i - \bigoplus_{i\in I} \mathtt{p}!a_i.S_i &= \bigoplus_{i\in J\setminus I} \mathtt{p}!a_i.T_i \oplus \bigoplus_{i\in I} \mathtt{p}!a_i.(T_i - S_i) \end{aligned}$$

To make acquaintance with '$-$' let us revisit some of the examples in the introduction. Let $T = \mu x.(\mathtt{q}!a.x \oplus \mathtt{q}!b.\mathsf{end})$ and $S_n = \mu y.((\mathtt{q}!a.)^n y \oplus \mathtt{q}!b.\mathsf{end})$. We have

$$T - S_n = \mu z.(\underbrace{\mathtt{q}!a.(\mathtt{q}!a.(\cdots(\mathtt{q}!a.}_{n} z \oplus \underbrace{\mathtt{q}!b.\mathsf{end})\cdots)}_{n-1} \oplus \mathtt{q}!b.\mathsf{end}) \oplus \mathtt{q}!b.\mathsf{fail})$$

and $T - S_\infty = T$. Observe that $T - S_\infty$ is viable, while no $T - S_n$ is because of the $\mathtt{q}!b.\mathsf{fail}$ branch. Also, when either T or S is finite $T - S$ is never viable. For example, $T - \mathtt{q}!b.\mathsf{end} = \mathtt{q}!a.T \oplus \mathtt{q}!b.\mathsf{fail}$ and $T - \mathtt{q}!a.\mathtt{q}!b.\mathsf{end} = \mathtt{q}!a.(\mathtt{q}!a.T \oplus \mathtt{q}!b.\mathsf{fail}) \oplus \mathtt{q}!b.\mathsf{end}$. This is consistent with Theorem 3.2(1), showing that $\leqslant_{\mathsf{U}}$ and $\leqslant$ coincide when the $\leqslant_{\mathsf{U}}$-smaller session type is finite. In general, we can prove that $T \prec S$ holds if and only if the difference between T and S is not viable.

Theorem 3.4. *Let $T, S \in \mathscr{T}_{\mathsf{nf}}$ and $T \leqslant_{\mathsf{U}} S$. Then $T \prec S$ if and only if $T - S$ is not viable.*

On the practical side, Theorem 3.4 allows us to decide $T \prec S$ if we can decide the viability of a session type (we will address this in Section 4). On the theoretical side,

it highlights an interesting analogy between our framework and that of semantic subtyping [11], which also motivates the notation $T - S$. We have observed that "being not viable" is equivalent to "being smaller than fail", and that fail somehow represents the empty type in our theory. Therefore, a consequence of Theorems 3.3 and 3.4 is that in order to decide $T \leqslant S$ one has to decide whether $T - S \leqslant \mathsf{fail}$. This reformulation is precisely the one used in the framework of semantic subtyping, where types are interpreted as sets of values and deciding the subtyping relation $\sigma \subseteq \tau$ is equivalent to deciding the emptiness of $\sigma \setminus \tau$. Note however that $T \prec S$ alone does not imply $T \leqslant S$. For example, we have $\mathsf{q}!a.T \oplus \mathsf{q}!b.\mathsf{end} \prec \mathsf{q}!a.S \oplus \mathsf{q}!b.\mathsf{end}$ where $T = \mu x.(\mathsf{q}?a.(\mathsf{q}!a.x \oplus \mathsf{q}!b.T) + \mathsf{q}?b.\mathsf{end})$ and $S = \mu y.(\mathsf{q}?a.\mathsf{q}!a.y + \mathsf{q}?b.\mathsf{end})$. Still, $\mathsf{q}!a.T \oplus \mathsf{q}!b.\mathsf{end} \not\leqslant \mathsf{q}!a.S \oplus \mathsf{q}!b.\mathsf{end}$ because $T \not\leqslant S$, as we already know.

4 Algorithms

In this section we define algorithms for deciding viability, for computing the normal form of viable session types, and for deciding subtyping. We also discuss the decidability of session correctness.

Viability. The viability of a session type T is tightly related to the reachability of end subtrees occurring in it. The algorithm we propose assumes initially that every subtree of T is viable and iteratively discards those subtrees for which this assumption is disproved. Each iteration performs three checks: a subtree $S \in \mathsf{trees}(T)$ is viable provided that end can be reached from it; input nodes are viable provided that there is at least one branch that is viable; output nodes are viable provided that every branch is viable. Formally, let the *viability sequence* for T be the sequence $\{\mathbf{V}^T_i\}_{i\in\mathbb{N}}$ of sets of session types defined in the following way, where $\leq$ is the usual prefix relation between strings of actions:

$$
\begin{aligned}
\mathbf{V}^T_0 &= \mathsf{trees}(T) \\
\mathbf{V}^T_{2i+1} &= \{S \in \mathbf{V}^T_{2i} \mid \exists s : S \stackrel{s}{\Longrightarrow} \mathsf{end}, \forall t \leq s : S \stackrel{t}{\Longrightarrow} S' \in \mathsf{trees}(T) \Rightarrow S' \in \mathbf{V}^T_{2i}\} \\
\mathbf{V}^T_{2i+2} &= \{\mathsf{end} \in \mathbf{V}^T_{2i+1}\} \cup \{\textstyle\sum_{j\in I} \mathsf{p}?a_j.T_j \in \mathbf{V}^T_{2i+1} \mid \exists j \in I : T_j \in \mathbf{V}^T_{2i+1}\} \\
&\qquad \cup \{\textstyle\bigoplus_{j\in I} \mathsf{p}!a_j.T_j \in \mathbf{V}^T_{2i+1} \mid \forall j \in I : T_j \in \mathbf{V}^T_{2i+1}\}
\end{aligned}
$$

Observe that, in computing $\mathbf{V}^T_{2i+1}$, it is not enough to be able to reach an end subtree from S to declare S viable. It must be the case that every subtree along the path $S \stackrel{s}{\Longrightarrow} \mathsf{end}$ has not been proved non-viable. Note also that, in principle, the computation of $\mathbf{V}^T_{2i+1}$ may need to consider an infinite number of strings s such that $S \stackrel{s}{\Longrightarrow}$. However, it is enough to consider those paths such that the derivation $S \stackrel{s}{\Longrightarrow}$ never goes through the same subtree twice. Since session types are regular trees and have a finite number of distinct subtrees, it always suffices to consider a finite number of paths. Every set in the sequence is finite and the sequence is decreasing. Therefore, there exists $k \in \mathbb{N}$ such that $\mathbf{V}^T_k = \mathbf{V}^T_{k+1} = \mathbf{V}^T_{k+2}$. We denote the fixpoint of the sequence with $\mathsf{viables}(T)$.

Theorem 4.1 (viability). *$T \in \mathscr{T}_{\mathsf{v}}$ if and only if $T \in \mathsf{viables}(T)$.*

Normal form. Once we know how to identify viable session types, computing their normal form is only a matter of pruning away those subtrees that are not viable. The normal form of T, denoted by $\mathsf{nf}(T)$, is defined coinductively by the following equations:

$$\begin{aligned} \mathsf{nf}(T) &= \mathsf{fail} && \text{if } T \notin \mathscr{T}_{\mathsf{v}} \\ \mathsf{nf}(\mathsf{end}) &= \mathsf{end} \\ \mathsf{nf}(\textstyle\sum_{i\in I} \mathtt{p}?a_i.T_i) &= \textstyle\sum_{i\in I, \mathsf{nf}(T_i)\neq\mathsf{fail}} \mathtt{p}?a_i.\mathsf{nf}(T_i) \\ \mathsf{nf}(\textstyle\bigoplus_{i\in I} \mathtt{p}!a_i.T_i) &= \textstyle\bigoplus_{i\in I} \mathtt{p}!a_i.\mathsf{nf}(T_i) \end{aligned}$$

(all the equations but the first one apply only to viable session types).

Theorem 4.2 (normal form). *For every T, $\mathsf{nf}(T)$ is in normal form and $T \leqslant \mathsf{nf}(T)$.*

Fair subtyping. We present a complete, algorithmic deduction system for the subtyping relation, which is coinductively defined in Table 3 (the corresponding inductive system can be obtained with standard memoization techniques). Rules (FS-END) and (FS-INPUT) are just the same as in well-known deduction systems for the unfair subtyping relation (see, e.g., [12]). Rule (FS-FAIL) states that fail is the least element according to $\leqslant_{\mathsf{A}}$. Rule (FS-OUTPUT) is similar to the familiar contravariant rule for outputs, except that it is applicable only when the smaller session type is ruled by the larger one, which can be determined by checking the viability of the difference of the two session types. It is enough to check the condition $T \prec S$ only when T and S are outputs. This is shown to imply that the condition holds whenever $T \leqslant_{\mathsf{A}} S$ is provable.

Table 3. Deduction system for the subtyping relation.

(FS-FAIL) $\mathsf{fail} \leqslant_{\mathsf{A}} T$

(FS-END) $\mathsf{end} \leqslant_{\mathsf{A}} \mathsf{end}$

(FS-INPUT)
$$\frac{T_i \leqslant_{\mathsf{A}} S_i \ ^{(i\in I)}}{\sum_{i\in I} \mathtt{p}?a_i.T_i \leqslant_{\mathsf{A}} \sum_{i\in I\cup J} \mathtt{p}?a_i.S_i}$$

(FS-OUTPUT)
$$\frac{T_i \leqslant_{\mathsf{A}} S_i \ ^{(i\in I)} \qquad \mathsf{nf}(T-S) = \mathsf{fail}}{T = \bigoplus_{i\in I\cup J} \mathtt{p}!a_i.T_i \leqslant_{\mathsf{A}} \bigoplus_{i\in I} \mathtt{p}!a_i.S_i = S}$$

Theorem 4.3. *$T \leqslant S$ if and only if $\mathsf{nf}(T) \leqslant_{\mathsf{A}} \mathsf{nf}(S)$.*

It seems like the $\prec$ relation does not admit a simple axiomatization. The problem is that the $\leqslant$ relation is not local, in the sense that the applicability of rule (FS-OUTPUT) may depend upon regions of the session types that are arbitrarily far away from the place where it is applied. Consider for instance the session type

$$T = \mu x.\mathtt{q}!a.(\mathtt{q}?a.)^n(\mathtt{q}!a.(\mathtt{q}?a.)^n x \oplus \mathtt{q}!b.\mathsf{end}) \oplus \mathtt{q}!b.\mathsf{end}$$

and observe that the two $\mathtt{q}!b$ branches can be arbitrarily distant depending on the number n of input actions. Both the session types

$$\begin{aligned} S_1 &= \mu x.\mathtt{q}!a.(\mathtt{q}?a.)^n(\mathtt{q}!a.(\mathtt{q}?a.)^n x \oplus \mathtt{q}!b.\mathsf{end}) \\ S_2 &= \mu y.(\mathtt{q}!a.(\mathtt{q}?a.)^n \mathtt{q}!a.(\mathtt{q}?a.)^n y \oplus \mathtt{q}!b.\mathsf{end}) \end{aligned}$$

are supertypes of T and they differ from T because one of the two $\mathtt{q}!b.\mathsf{end}$ branches has been pruned. However, pruning both branches results into a non-viable session type. Therefore, one branch can be safely removed only if the other one is not.

Correctness. We conclude this section with a few considerations on the decidability of correctness. Observe that, since session types are regular and finite branching, the set $\mathscr{R}(M) \stackrel{\text{def}}{=} \{N \mid M \stackrel{\tau}{\Longrightarrow} N\}$ is finite and can be computed in finite time by exploring every session reachable from M. Now M is correct if and only if for every $N \in \mathscr{R}(M)$ there exists $N' \in \mathscr{R}(N)$ such that $N' \stackrel{\checkmark}{\longrightarrow}$.

In the special case of binary sessions, when only two participants $\mathtt{p}$ and $\mathtt{q}$ are involved, the session $\mathtt{p} : T \mid \mathtt{q} : \overline{T}$ is always correct, assuming that $\mathtt{q}$ is the only role occurring in T, that T is in normal form, and that $\overline{T}$ is the *dual* of T coinductively defined by:

$$\overline{\mathsf{end}} = \mathsf{end} \qquad \overline{\textstyle\sum_{i\in I} \mathtt{q}?a_i.T_i} = \bigoplus_{i\in I} \mathtt{p}!a_i.\overline{T_i} \qquad \overline{\textstyle\bigoplus_{i\in I} \mathtt{q}!a_i.T_i} = \sum_{i\in I} \mathtt{p}?a_i.\overline{T_i}$$

By definition of $\leqslant$, every session $\mathtt{p} : T \mid \mathtt{q} : S$ where $\overline{T} \leqslant S$ is also correct. However, the converse is not true. That is, there are correct sessions $\mathtt{p} : T \mid \mathtt{q} : S$ where $\overline{T} \not\leqslant S$, for example when $T = \mu x.(\mathtt{q}!a.(\mathtt{q}?a.x + \mathtt{q}?b.x) \oplus \mathtt{q}!b.\mathsf{end})$ and $S = \mu y.(\mathtt{p}?a.\mathtt{p}!a.y + \mathtt{p}?b.\mathsf{end})$. This is in sharp contrast with the unfair theories [12,6], where $\mathtt{p} : T \mid \mathtt{q} : S$ is correct (in the "unfair" sense) if and only if $\overline{T} \leqslant_{\mathsf{U}} S$.

5 Conclusions

The standard subtyping relation for session types may compromise liveness of multiparty sessions. Even in dyadic sessions it might be desirable not to lose the ability to reach successful termination of the interacting parties. These scenarios naturally call for the definition of (multi-party) session type theories where every participant preserves the possibility to reach a successfully terminated state.

Fair subtyping relations (often referred to as refinements in concurrency theory) have rightfully gained the fame of being hard to characterize completely [18,21] or even to approximate [4,19]. In this paper we have fully characterized the fair subtyping relation as a simple variation of standard subtyping [12,6]. It is not entirely clear how much the characterization of the subtyping relation we have given owes to the fact that we work with a very primitive process language. The proof of the characterization (Theorem 3.3) only needs the semantic definition of $\prec$ (Definition 3.3) and therefore should be generalizable to full-featured process languages. It is not obvious, and thus subject to future investigation, whether the same holds for the notion of difference (Definition 3.5).

Checking whether a multi-party session is correct can be more expensive than in dyadic theories (Section 4). This observation substantiates the effectiveness of the design-by-contract approach advocated in [5,17], where the session types of a multiparty session are obtained as projections of a global type associated with the session. The approach guarantees that the resulting session is correct by construction. However, it may be necessary to use subtyping both during the projection as well as while type checking processes against the session types of the channels they use. Therefore, it is fundamental for subtyping to preserve session liveness (in the sense of Definition 2.1). Type checking processes using a fair subtyping relation seems to pose interesting technical problems, because of the interplay between coinductive typing of recursive processes and the liveness property we want to enforce on sessions. We leave these issues for future investigations.

Acknowledgments. The author is grateful to Daniele Varacca for the discussions on fairness and to the anonymous referees who helped improving the paper. This work was partially supported by a visiting professor position of the Université Paris Diderot.

References

1. Franco Barbanera and Ugo de'Liguoro. Two notions of sub-behaviour for session-based client/server systems. In *Proceedings of PPDP'10*, pages 155–164. ACM, 2010.
2. Mario Bravetti and Gianluigi Zavattaro. A foundational theory of contracts for multi-party service composition. *Fundamenta Informaticae*, 89(4):451–478, 2009.
3. Mario Bravetti and Gianluigi Zavattaro. A theory of contracts for strong service compliance. *Mathematical Structures in Computer Science*, 19:601–638, 2009.
4. Michele Bugliesi, Damiano Macedonio, Luca Pino, and Sabina Rossi. Compliance preorders for Web Services. In *Proceedings of WS-FM'09*, LNCS 6194, pages 76–91. Springer, 2010.
5. Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centred programming for web services. In *Proceedings of ESOP'07*, LNCS 4421, pages 2–17, 2007.
6. Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, Elena Giachino, and Luca Padovani. Foundations of session types. In *Proceedings of PPDP'09*, pages 219–230. ACM, 2009.
7. Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A theory of contracts for Web services. *ACM Transactions on Programming Languages and Systems*, 31(5):1–61, 2009.
8. Bruno Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25:95–169, 1983.
9. Rocco De Nicola and Matthew Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.
10. Rocco De Nicola and Matthew Hennessy. CCS without τ's. In *Proceedings of TAPSOFT'87/CAAP'87*, LNCS 249, pages 138–152. Springer, 1987.
11. Alain Frisch, Giuseppe Castagna, and Veronique Benzaken. Semantic subtyping: dealing set-theoretically with function, union, intersection, and negation types. *Journal of the ACM*, 55(4):1–64, 2008.
12. Simon Gay and Malcolm Hole. Subtyping for session types in the π-calculus. *Acta Informatica*, 42(2-3):191–225, 2005.
13. Matthew Hennessy. Acceptance trees. *Journal of the ACM*, 32(4):896–928, 1985.
14. Matthew Hennessy. *Algebraic Theory of Processes*. Foundation of Computing. MIT Press, 1988.
15. Kohei Honda. Types for dyadic interaction. In *Proceedings of CONCUR'93*, LNCS 715, pages 509–523. Springer, 1993.
16. Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language primitives and type disciplines for structured communication-based programming. In *Proceedings of ESOP'98*, LNCS 1381, pages 122–138. Springer, 1998.
17. Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *Proceedings of POPL'08*, pages 273–284. ACM, 2008.
18. V. Natarajan and Rance Cleaveland. Divergence and fair testing. In *Proceedings of ICALP '95*, LNCS 944, pages 648–659. Springer, 1995.
19. Luca Padovani. Session types at the mirror. *EPTCS*, 12:71–86, 2009.
20. Luca Padovani. Fair subtyping for multi-party session types. Full version available at `http://www.di.unito.it/~padovani/Papers/FairSessionTypes.pdf`, 2011.
21. Arend Rensink and Walter Vogler. Fair testing. *Information and Computation*, 205(2):125–198, 2007.
22. Vasco T. Vasconcelos. Fundamentals of session types. In *SFM'09*, LNCS 5569, pages 158–186. Springer, 2009.