

Fair Subtyping for Multi-Party Session Types

Luca Padovani

Dipartimento di Informatica, Università di Torino, Italy

Laboratoire Preuves, Programmes et Systèmes, Université Paris Diderot, France

e-mail: padovani@di.unito.it

Abstract

We study a theory of session types in which we add a liveness property to the familiar safety one. In this setting, some subtype relations between session types that hold in other theories and that are commonly regarded as harmless become unsound. We present various equivalent definitions of the subtyping relation, we relate it with the standard one, and we give algorithms for deciding it. Incidentally, 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

A session is a conversation between processes that interact with one another by means of a private channel. Each process uses the channel according to a session type [15, 16, 21] that describes the order and the type of messages that the process is allowed to send or expected to receive in the scope of the session. For example, the session $p : q!a.q!a.?b.\text{end} \mid q : ?a.?a.p!b.\text{end}$ describes a conversation between two participants identified by the tags p and q : p sends two a messages to q and then waits for a b message; q waits for two a messages and then sends b to p . This is an example of *correct session*, where every output of a participant corresponds to an input of another participant and vice versa.

More interesting session types describe alternative and recursive behaviors. For example, the session types satisfying the equations

$$T = q!a.T \oplus q!b.\text{end} \quad R = ?a.R + ?b.\text{end}$$

describe two processes that respectively send and receive arbitrarily long sequences of a messages followed by a b message. The operators \oplus and $+$ distinguish between two kinds of behavioral choice, often named *internal* and *external* choice, respectively: a sender internally chooses the type (a or b) of message to send; a receiver externally offers two types of messages (a and b) that it is capable to handle and lets the other process decide which one to send. It is intuitively clear that $p : T \mid q : R$ describes a correct session: no matter of the type and the number of messages that p sends, q is capable to receive all of them.

The most common way to add flexibility to a type theory is to introduce a notion of *subtyping* that defines an asymmetric compatibility between types.

Informally, when T is a subtype of S it is safe to use a channel of type T whenever a channel of type S is expected. For example, the session type T defined above is a subtype of $q!b.\text{end}$: using a channel of type $q!b.\text{end}$ means sending a b message to process 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 $q!b.\text{end}$ cannot compromise correctness. The reader may draw an insightful analogy between subtyping of session and record types: if “using a record” means selecting one of its fields, then it is safe to use a record in every context where a subset of its fields can be selected. By generalizing the example above we may deduce that T is a subtype of S if S is a variant of T where some internal choices have been reduced in width. According to this intuition it is easy to see that all the session types in the family

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

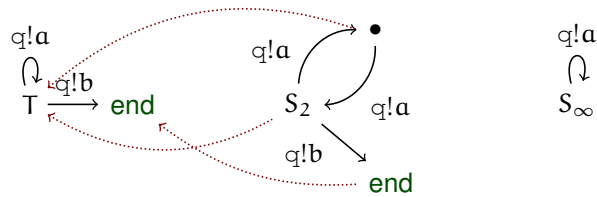
are supertypes of T . The type S_n allows sending a b message only after the number of sent a messages is multiple of n . The type S_∞ is somehow the limit of the sequence $\{S_i\}_{i \geq 2}$ and describes a process that only sends a messages. The fact that T is a subtype of S_∞ may be questionable: on the one hand, no process behaving as S_∞ can cause any harm, since no message of unexpected type is ever sent; on the other hand, while the sessions $p : S_i | q : R$ for $i \geq 2$ all have the potential for terminating (it is always possible that a b message is sent), the session $p : S_\infty | q : R$ is doomed to loop forever.

Type theories of session types with subtyping have been thoroughly studied, for example in [12, 6]. In these works the widely accepted point of view that drives the theory is that a session is correct when “nothing bad ever happens”. From this perspective, the session type T is a subtype of S_∞ . When we shift the focus from dyadic sessions (those with exactly two participants) to so-called *multy-party sessions* [17], where an arbitrary number of participants is involved, this viewpoint shows its limits, well beyond the questionable subtyping relation above. For example, in the session $p : S_\infty | q : R | r : ?c.\text{end}$ the participants p and q keep interacting with each other and neither of them cares to send the c message that r is waiting for. This session is correct, because no participant ever receives an unexpected message, but this is a poor consolation to r . Sure, by looking at the session types S_∞ and R one can tell that the problem is obvious, since r is never mentioned in them. However, S_∞ might just happen to be the supertype of some T' such that $p : T' | q : R | r : ?c.\text{end}$ does not exhibit the same problem for r (one such type is $T' = q!a.T' \oplus q!b.r!c.\text{end}$).

In this paper we develop a theory of session types that is centered around a stronger notion of correctness. In our theory, a correct session must preserve the potential for reaching a state where all of its participants are satisfied. As we have just seen, this stronger notion of correctness has an impact on the induced subtyping relation that cannot be neglected. For example, T and T' are no longer subtypes of S_∞ because $p : T | q : R$ and $p : T' | q : R | r : ?c.\text{end}$ are correct but $p : S_\infty | q : R$ and $p : S_\infty | q : R | r : ?c.\text{end}$ are not. 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 infinite 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

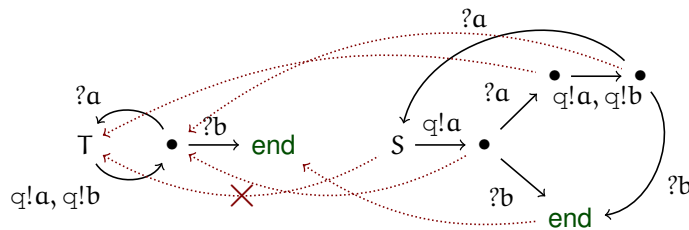
branch of an internal choice can be safely removed may involve a non-local check on the structure of the session types being compared. This results into a subtyping relation that is difficult, if at all possible, to axiomatize.

To illustrate the subtleties behind our subtyping relation, consider the session types T , S_2 , and S_∞ represented as the three automata below, where the initial states have been labelled with the name of the session type and the 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 above, 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 that S_2 has an intermediate state \bullet which lacks the outgoing $q!b$ -labelled transition that T has. The correspondence between T and this state of S_2 is safe if (and only if) there is no behavior R that makes $p : T \mid q : R$ correct and q is capable to loop the interaction starting from $p : S_2 \mid q : R$ in such a way that the \bullet state is visited infinitely often. If this were the case, 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 $p : S_2$ can always break the loop by sending 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 that completes) T , which we denote by $T \prec S_2$.

A more involved example where this is not the case is the following:



The only difference between T and S is that S lacks the outgoing $q!b$ -labelled transition that T has. Basically, $p : S$ may send a b message only after an odd number of a messages have been sent to q and an equal number of a messages have been received. Unlike the previous example, it is q that decides whether to terminate the interaction with p , by sending a b message, or to continue, by sending an a message. Consider now the participant $q : R$ where $R = ?a.p!a.(?a.p!a.R + ?b.p!a.R) + ?b.p!b.end$. It is easy to see that $p : T \mid q : R$ is correct and that $p : S \mid q : R$ loops through state S . In other words, q forces $p : S$

to go through state S hoping to receive a b message. This was possible with $p : T$, but not with $p : S$. The fact that a participant like $q : R$ 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 removing outputs is safe.

Related work. The framework we have depicted is known in concurrency theory as *fair testing* [18, 20]. 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 . The notion of “test” roughly corresponds to the property that “nothing bad ever happens” in “unfair” testing theories, and to the property that “something good will eventually happen” in fair testing theories. The idea is that, in a system that goes infinitely often through a state from which some action is possible, that action will eventually be observed. 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 unfair refinements 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 also been given in the literature, but we find them unsatisfactory in one way or another. The authors of [18] present a characterization based on sets of infinite strings, while [20] relies on a complex 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 [20], 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 [20] 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). The standard reference for subtyping of session types is [12], where the subtyping relation is coinductively presented and thus “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 fair theory of multi-party session types where the focus is on the eventual satisfaction of all the interacting participants of a session rather than on the absence of communication errors. 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] plus a “ruled by” relation for which we provide a relatively simple decision procedure based on a 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

Table 1: Syntax of session types and sessions.

$T ::=$	Session Type	$M ::=$	Session
fail	(failure)	$p : T$	(participant)
end	(termination)	$M M$	(composition)
$\sum_{i \in I} ?a_i.T_i$	(input)		
$\bigoplus_{i \in I} p_i!a_i.T_i$	(output)		

standard one, modulo 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 compare our subtyping relation with 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. For the sake of readability, proofs and auxiliary technical material have been collected in Appendix A.

2 Syntax and Semantics of Session Types

Table 1 defines the syntax of sessions and session types. A *session* is a finite composition $p_1 : T_1 | \dots | p_n : T_n$ made of a fixed number of participants that communicate with each other according to the session types T_i . We assume to work exclusively with well-formed sessions, where each participant is uniquely identified by a *tag* p_i ($i \neq j$ implies $p_i \neq p_j$). Session types are the possibly infinite, finitely-branching, regular trees generated by the nonterminal T in the grammar in Table 1 such that:

- $a_i = a_j$ implies $T_i = T_j$ for every subterm $\sum_{i \in I} ?a_i.T_i$ and $i, j \in I$;
- $p_i!a_i = p_j!a_j$ implies $T_i = T_j$ for every subterm $\bigoplus_{i \in I} p_i!a_i.T_i$ and $i, j \in I$.

These two conditions ensure that session types are unambiguous by requiring that every prefix of the form $?a$ or $p!a$ uniquely determines a continuation. The session type **end** describes a successfully terminated participant that no longer participates to the session. The session type $\sum_{i \in I} ?a_i.T_i$ describes a process that waits for a message: depending on the type of the message it receives, represented by an atomic name a_i , the process behaves according to the continuation T_i . The session type $\bigoplus_{i \in I} p_i!a_i.T_i$ describes a process that internally decides to send some message of type a_i to another participant of the session identified by tag p_i . After the output operation the process behaves as described in the session type T_i . It is convenient (although not necessary) to have a canonical term **fail** describing *failed participants* that are unable to terminate successfully. This happens, for example, if a participant receives an unexpected message. No correct session should ever involve a process that has failed. Let us fix a few notational conventions: p, q, \dots range over tags; a, b, \dots range over action names; T, S, \dots range over session types, and M, N, \dots over sessions. Sometimes we will write $?a_1.T_1 + \dots + ?a_n.T_n$ for $\sum_{i=1}^n ?a_i.T_i$ and

Table 2: Transition system of sessions.

(T-SUCCESS) $p : \mathbf{end} \xrightarrow{\checkmark} p : \mathbf{end}$	(T-OUTPUT) $p : q!a.T \xrightarrow{q!a} p : T$	(T-CHOICE) $\frac{k \in I}{p : \bigoplus_{i \in I} q_i!a_i.T_i \longrightarrow p : q_k!a_k.T_k}$
(T-INPUT) $\frac{k \in I}{p : \sum_{i \in I} ?a_i.T_i \xrightarrow{p?a_k} p : T_k}$	(T-FAIL) $\frac{a \neq a_i \ (i \in I)}{p : \sum_{i \in I} ?a_i.T_i \xrightarrow{p?a} p : \mathbf{fail}}$	(T-PAR CHOICE) $\frac{M \longrightarrow M'}{M N \longrightarrow M' N}$
(T-PAR ACTION) $\frac{M \xrightarrow{\alpha} M' \quad \alpha \neq \checkmark}{M N \xrightarrow{\alpha} M' N}$	(T-COMM) $\frac{M \xrightarrow{p!a} M' \quad N \xrightarrow{p?a} N'}{M N \longrightarrow M' N'}$	(T-PAR SUCCESS) $\frac{M \xrightarrow{\checkmark} M \quad N \xrightarrow{\checkmark} N}{M N \xrightarrow{\checkmark} M N}$

$p_1!a_1.T_1 \oplus \dots \oplus p_n!a_n.T_n$ for $\bigoplus_{i=1}^n p_i!a_i.T_i$. Recall that a regular tree is made of a *finite* number of distinct subtrees [8]. We will write $\mathit{trees}(T)$ for the finite set of subtrees that T is made of, including T itself.

We express the evolution of a session by means of a labelled 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, ranged over by α, \dots , describe successful termination and input/output actions and are generated by the grammar that follows:

$$\alpha ::= \checkmark \mid p?a \mid p!a$$

Table 2 defines the transition system (symmetric rules omitted). Rule (T-SUCCESS) states that \mathbf{end} performs a \checkmark action that denotes successful termination and reduces to itself. Rules (T-OUTPUT) and (T-CHOICE) deal with outputs. The former one shows that a participant willing to send a a -message to participant q performs a $q!a$ 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. Rules (T-INPUT) and (T-FAIL) deal with inputs. The former one is standard and states that a participant p performs $p?a$ actions according to the type of messages it is willing to receive. The latter shows that a participant can receive an unexpected input, but in doing so it will reduce to an unrecoverable failed state. Rules (T-PAR CHOICE) and (T-PAR ACTION) are standard rules propagating transitions through compositions. (T-COMM) is the usual communication rule. Finally, (T-PAR SUCCESS) states that a composition has successfully terminated only if all of its participants have. In the following we adopt the following conventions: we write \Longrightarrow for the reflexive, transitive closure of \longrightarrow ; we write $\xrightarrow{\alpha}$ for $\Longrightarrow \xrightarrow{\alpha} \Longrightarrow$ and $\xrightarrow{\alpha_1 \dots \alpha_n}$ for the composition $\xrightarrow{\alpha_1} \dots \xrightarrow{\alpha_n}$; we let s, t, \dots range over finite strings of actions different from \checkmark ; we write $M \xrightarrow{\alpha}$ (respectively, $M \xrightarrow{\alpha}$) if there exists N such that $M \longrightarrow N$ (respectively, $M \xrightarrow{\alpha} N$); we write $M \not\xrightarrow{\alpha}$ if there exists no N such that $M \longrightarrow N$. We also extend the labelled transition relation and the above notation to session types so that, for example, $T \xrightarrow{\alpha} S$ if $p : T \xrightarrow{\alpha} p : S$ for some p .

We conclude this section with the definition of *correct session*. Intuitively, a session is correct if at any time it is always possible to reach a state where every participant of the session has successfully terminated. Formally we have:

Definition 2.1. We say that M is correct if $M \Longrightarrow N$ implies $N \xrightarrow{\checkmark}$.

As an example, consider the session $M = p : T \mid q : S \mid r : ?b.\text{end}$ where T and S are defined by the equations:

$$T = q!a.T \oplus r!b.T' \quad T' = q!a.T' \oplus q!c.\text{end} \quad S = ?a.S + ?c.\text{end}$$

It is easy to verify that M is a correct session: even though participant p keeps sending a messages to q , it always has the potential to send a b message to r . Once it has done so, it starts again sending a messages to q , but this time it can also terminate the interaction by sending a c message. At this point all the participants have successfully terminated. Perhaps more interesting are the *incorrect* sessions. For instance, $p : q!a.\text{end} \oplus q!b.\text{end} \mid q : ?a.\text{end}$ is not correct because p may decide to send a b message that q is not willing to receive. Even though at the beginning of the interaction there is one potential path leading to successful termination, the decision of sending b is made irrevocable by rule (T-CHOICE). There are also intrinsically flawed session types that can never be part of a correct session. For example, the session $M \mid p : \text{fail}$ is incorrect regardless of M , because fail is never able to perform \checkmark . The session type fail describes a participant that is unable to terminate successfully. Some sessions are incorrect despite that no deadlock occurs in them. This happens in the session $p : T \mid q : S$ where $T = q!a.T$ and $S = ?a.S$ because, even though the participants keep interacting with each other, they do not have the ability to terminate the interaction.

It is useful to remark a few easy properties of correctness: $p : \text{end}$ is the simplest correct session; the session $M \mid p : \text{end}$ 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 $M \mid p : T$ 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 $T \leq S$, if $M \mid p : T$ correct implies $M \mid p : S$ correct for every M . We write \leq for the equivalence relation induced by \leq , namely $\leq = \leq \cap \leq^{-1}$.

This definition may look surprising at first, because it speaks about left-to-right substitutability, while subtyping is usually concerned with right-to-left substitutability: it is safe to use a value of some subtype T wherever a value of a supertype S is expected. 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) while subtyping concerns the substitution of channels, not of processes. 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 with another channel d with type $T \leq S$, we are changing the set of processes that P is interacting with, which behave according to some M such that $M \mid p : T$ is correct. Replacing c with d does *not* affect the way P behaves:

P keeps using 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 $M \mid p : S$. Since $T \leq S$, we know that this session is correct.

A thorough study of the subtyping relation that solely relies on Definition 2.2 is not easy, because of the universal quantification over an infinite set of contexts M . Nonetheless, a few relations are easy to establish. For example, we have

$$(i) \ ?a.\text{end} \leq ?a.\text{end} + ?b.\text{end} \quad \text{and} \quad (ii) \ p!a.\text{end} \oplus q!b.\text{end} \leq p!a.\text{end}$$

namely \leq 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), any context M such that $M \mid p : ?a.\text{end}$ is correct must eventually send some message to p , and this message can only be a for otherwise p would fail because of rule (T-FAIL). Therefore, $M \mid p : ?a.\text{end} + ?b.\text{end}$ is also correct, since $?a.\text{end} + ?b.\text{end}$ is more receptive than $?a.\text{end}$. In (ii), any context M such that $M \mid r : p!a.\text{end} \oplus q!b.\text{end}$ is correct must be able to terminate successfully no matter which message (either a or b) is sent (to p or q , respectively). One such context is $M = p : ?a.q!c.\text{end} + ?c.\text{end} \mid q : ?b.p!c.\text{end} + ?c.\text{end}$. Therefore, nothing bad happens when we replace $p!a.\text{end} \oplus q!b.\text{end}$ with a more deterministic behavior, such as $p!a.\text{end}$. 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 is the largest relation \leq_U such that $T \leq_U S$ implies either:

1. $T = S = \text{end}$, or
2. $T = \sum_{i \in I} ?a_i.T_i$ and $S = \sum_{i \in I \cup J} ?a_i.S_i$ and $T_i \leq_U S_i$ for every $i \in I$, or
3. $T = \bigoplus_{i \in I \cup J} p_i!a_i.T_i$ and $S = \bigoplus_{i \in I} p_i!a_i.S_i$ and $T_i \leq_U S_i$ for every $i \in I$.

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 $\{p_i!a_i\}_{i \in I \cup J}$ can be sent if it never sends a message that is not in this set.

The relation \leq_U is appealing because of its simple and intuitive definition, but it is neither sound nor complete if compared with \leq . On the one hand, the \leq_U relation does not preserve correctness as by Definition 2.1. For instance, the reader may verify that $T \leq_U S_\infty$ holds for T and S_∞ defined in the introduction, but $T \not\leq S_\infty$. On the other hand, there exists a large class of equivalent session types that are syntactically unrelated. For instance, we have $S_\infty \leq \text{fail}$ and $S_\infty \not\leq_U \text{fail}$. Session types like **fail** or S_∞ are flawed because there is no correct

session in which they can occur. Therefore, they are the \leq -least elements and 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 simple, complete, coinductive characterization of \leq . 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 $T \leq S$ as the conjunction of two relations, the familiar (but unsafe) $T \leq_U S$ subtyping for session types (which is shown to include \leq when restricted on 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 how to reduce the $T \prec S$ relation to deciding 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 $M \mid p : T$ is correct for some M and p . We write \mathcal{T}_v for the set of viable session types.*

A session type T is *not* viable if and only if $T \leq \mathbf{fail}$. That is, being not viable means being “smaller than” the empty type. The existence of non-viable session types hinders the definition of coinductive characterizations of the subtyping relation such as the one in Definition 2.3. The reason is that 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. For the rest of this section we assume to work with session types in a normal form guaranteeing that every subtree in a session type in normal form is viable.

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

Observe that if $T \in \mathcal{T}_{nf}$, then $S \in \mathcal{T}_{nf}$ for every $S \in \mathbf{trees}(T)$. The following proposition assures us that working with session types in normal forms is convenient and not restrictive: every session type in normal form is viable, and every viable session type has an equivalent one in normal form.

Proposition 3.1. *The following properties hold: (1) $\mathcal{T}_{nf} \subseteq \mathcal{T}_v$; (2) for every $T \in \mathcal{T}_v$ there exists $S \in \mathcal{T}_{nf}$ such that $T \leq S$.*

Furthermore, the syntax of session types in normal form is “meaningful” in the sense that \leq_U includes \leq when we focus on session types in normal form.

Theorem 3.1. *Let $T, S \in \mathcal{T}_{nf}$. Then $T \leq S$ implies $T \leq_U S$.*

Unfair subtyping and \leq_U decomposition. Focusing on session types in normal form does not change the fact that \leq_U is unsound with respect to \leq . More precisely, \leq_U does not introduce deadlocks, but it can introduce livelocks:

Theorem 3.2. *Let $T, S \in \mathcal{T}_{\text{nf}}$. If $T \leq_U S$ and $M \mid_P : T$ is correct, then $M \mid_P : S \implies N \dashv\dashv$ implies $N \checkmark$.*

An immediate corollary of Theorems 3.1 and 3.2 is that \leq_U and \leq coincide when either of the session types being compared is finite. In particular:

Corollary 3.1. *Let $T, S \in \mathcal{T}_{\text{nf}}$ and $T \leq_U S$ and S be finite. Then $T \leq S$.*

Proof. Immediate since every maximal computation of $M \mid_P : S$ is finite. \square

Theorem 3.2 and Corollary 3.1 show that \leq_U is not too far away from being a sound characterization of \leq . Therefore, we seek for a decomposition of the relation $T \leq S$ as the conjunction of two relations: $T \leq_U S$, 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 S). The “ruled by” relation \prec is defined thus:

Definition 3.3. *We say that T is ruled by S , notation $T \prec S$, if $M \mid_P : T$ correct implies $M \mid_P : S \checkmark$ for every M .*

Clearly S may preclude successful termination only when some outputs are removed (increasing the inputs increases the paths leading to successful termination). The property $T \prec S$ states that *every* context M that completes T *must* take into account the possibility to successfully terminate by following a path of actions shared with S . In other words, there exists no M that solely relies on the outputs emitted by T but not by S in order to successfully terminate. We formalize the decomposition of \leq as the “conjunction” of \leq_U and \prec :

Definition 3.4 (coinductive fair subtyping). *Let \leq_C be the largest relation such that $T \leq_C S$ implies $T \leq_U S$ and $T \prec S$.*

The relation \leq_C is indeed the characterization of \leq we are looking for:

Theorem 3.3. *Let $T, S \in \mathcal{T}_{\text{nf}}$. Then $T \leq S$ if and only if $T \leq_C S$.*

Characterization of \prec and behavioral difference. We now shift the focus on the \prec relation. Suppose $T \not\prec S$. Then there exists some context M such that the correctness of $M \mid_P : T$ 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 S in three respects:

1. every **end** in S has been turned to a **fail** in $T - S$;
2. $T - S$ performs no more inputs than those performed by T ;
3. $T - S$ performs all the outputs performed by T .

Formally:

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

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

Because of its definition, $T - S$ is viable if there exists M such that the correctness of $M \mid p : T - S$ solely depends on the **end** leaves found in those branches of T that have been pruned in S by item (3) of Definition 2.3. To make acquaintance with ‘-’ let us revisit some of the examples we have seen in the introduction. Let $T = q!a.T \oplus q!b.\text{end}$ and $S_n = (q!a.)^n S_n \oplus q!b.\text{end}$. We have

$$T - S_n = \underbrace{q!a.(q!a.(\dots (q!a.(T - S_n) \oplus q!b.\text{end}) \dots) \oplus q!b.\text{end})}_{n} \oplus q!b.\text{fail}$$

and $T - S_\infty = T$. Observe that $T - S_\infty$ is viable, while no $T - S_n$ is because of the $q!b.\text{fail}$ branch. Also, when S is finite $T - S$ is never viable. For example, $T - q!b.\text{end} = q!a.T \oplus q!b.\text{fail}$ and $T - q!a.q!b.\text{end} = q!a.(q!a.T \oplus q!b.\text{fail}) \oplus q!b.\text{end}$. This is consistent with Corollary 3.1, showing that \leq_U and \leq coincide when the larger 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 $T \leq_U S$. Then $T \prec S$ if and only if $T - S$ is not viable.*

We conclude this section with 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 $T \leq S$ one has to decide whether $T - S \leq \text{fail}$. 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 $T \leq S$. For example, we have $q!a.T \oplus q!b.\text{end} \prec q!a.S \oplus q!b.\text{end}$ where $T = ?a.(q!a.T \oplus q!b.T) + ?b.\text{end}$ and $S = ?a.q!a.S + ?b.\text{end}$. Still, $q!a.T \oplus q!b.\text{end} \not\leq q!a.S \oplus q!b.\text{end}$ because $T \not\leq S$, 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 about the complexity of deciding 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 $S \in \text{trees}(T)$ 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 $\{\mathbf{V}_i^T\}_{i \in \mathbb{N}}$ of sets of session types defined in this way:

$$\begin{aligned} \mathbf{V}_0^T &= \text{trees}(T) \\ \mathbf{V}_{2i+1}^T &= \{S \in \mathbf{V}_{2i}^T \mid \exists s : S \xrightarrow{s} \text{end}, \forall t \leq s : S \xrightarrow{t} S' \in \text{trees}(T) \Rightarrow S' \in \mathbf{V}_{2i}^T\} \\ \mathbf{V}_{2i+2}^T &= \{\text{end}\} \cup \{\sum_{j \in I} ?a_j.T_j \in \mathbf{V}_{2i+1}^T \mid \exists j \in I : T_j \in \mathbf{V}_{2i+1}^T\} \\ &\quad \cup \{\bigoplus_{j \in I} p_j!a_j.T_j \in \mathbf{V}_{2i+1}^T \mid \forall j \in I : T_j \in \mathbf{V}_{2i+1}^T\} \end{aligned}$$

where \leq is the usual prefix relation between strings of actions.

Observe that, in computing \mathbf{V}_{2i+1}^T , 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 $S \xrightarrow{s} \text{end}$ has not been proved non-viable. Note also that the algorithm needs to go through a potentially infinite number of strings s such that $S \xrightarrow{t}$. However, it is enough to consider those paths such that the derivation $S \xrightarrow{t}$ never goes through the same subtree twice. Since regular session types have a finite number of distinct subtrees, it always suffices to check 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 $\mathbf{V}_k^T = \mathbf{V}_{k+1}^T = \mathbf{V}_{k+2}^T$. We denote the fixpoint of the sequence with $\text{viables}(T)$.

Theorem 4.1 (viability). $T \in \mathcal{T}_v$ if and only if $T \in \text{viables}(T)$.

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

$$\begin{aligned} \text{nf}(\text{end}) &= \text{end} \\ \text{nf}(\sum_{i \in I} ?a_i.T_i) &= \sum_{i \in I, T_i \in \text{viables}(T_i)} ?a_i.\text{nf}(T_i) \\ \text{nf}(\bigoplus_{i \in I} p_i!a_i.T_i) &= \bigoplus_{i \in I} p_i!a_i.\text{nf}(T_i) \end{aligned}$$

Theorem 4.2 (normal form). If $T \in \mathcal{T}_v$, then $\text{nf}(T)$ is in normal form and $T \preceq \text{nf}(T)$.

Fair subtyping. We present a sound and complete 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-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 with the algorithm above. It is enough to check the condition $T \prec S$ only when T and S are outputs and the latter has strictly fewer branches than the former. This is shown to imply that the condition holds whenever $T \preceq_A S$ is provable.

Theorem 4.3. Let $T, S \in \mathcal{T}_{\text{nf}}$. Then $T \preceq S$ if and only if $T \preceq_A S$.

It seems like the \prec relation does not admit a simple axiomatization. The problem is that the \preceq relation is not local, in the sense that the applicability

Table 3: Deduction system for the subtyping relation.

(FS-END)	
$\text{end} \leq_A \text{end}$	
(FS-INPUT)	(FS-OUTPUT)
$T_i \leq_A S_i \quad (i \in I)$	$T_i \leq_A S_i \quad (i \in I) \quad T - S \notin \text{viables}(T - S)$
$\frac{\sum_{i \in I} ?a_i.T_i \leq_A \sum_{i \in I \cup J} ?a_i.S_i}{\sum_{i \in I} ?a_i.T_i \leq_A \sum_{i \in I \cup J} ?a_i.S_i}$	$\frac{T = \bigoplus_{i \in I \cup J} p_i!a_i.T_i \leq_A \bigoplus_{i \in I} p_i!a_i.S_i = S}{T = \bigoplus_{i \in I \cup J} p_i!a_i.T_i \leq_A \bigoplus_{i \in I} p_i!a_i.S_i = S}$

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 $T = q!a.(?a.)^n(q!a.(?a.)^n T \oplus q!b.\text{end}) \oplus q!b.\text{end}$ and observe that the two $q!b$ branches can be arbitrarily distant according to the number n of input actions. Both the session types $S_1 = q!a.(?a.)^n(q!a.(?a.)^n S_1 \oplus q!b.\text{end})$ and $S_2 = q!a.(?a.)^n q!a.(?a.)^n S_2 \oplus q!b.\text{end}$ are supertypes of T and they differ from T because one of the two $q!b.\text{end}$ branches has been removed. However, removing both branches results into a non-viable session type. Therefore, one branch can be safely removed only if the other one is not.

One can now combine Theorems 4.1, 4.2, and 4.3 to define a complete algorithm for deciding $T \leq S$ in the general case:

1. if $T \notin \text{viables}(T)$, then T is not viable and $T \leq S$ holds;
2. if $T \in \text{viables}(T)$ and $S \notin \text{viables}(S)$, then $T \not\leq S$;
3. otherwise, $T \leq S$ holds if and only if $\text{nf}(T) \leq_A \text{nf}(S)$ does.

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 $\mathcal{R}(M) \stackrel{\text{def}}{=} \{N \mid M \Longrightarrow N\}$ is finite and can be computed in finite time by exploring every state reachable from M . Now M is correct if and only if for every $N \in \mathcal{R}(M)$ there exists $N' \in \mathcal{R}(N)$ such that $N' \xrightarrow{\checkmark}$.

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

$$\overline{\text{end}} = \text{end} \quad \overline{\sum_{i \in I} ?a_i.T_i} = \bigoplus_{i \in I} p!a_i.\bar{T}_i \quad \overline{\bigoplus_{i \in I} q!a_i.T_i} = \sum_{i \in I} ?a_i.\bar{T}_i$$

By definition of \leq , every session $p : T \mid q : S$ where $\bar{T} \leq S$ is also correct. However, the converse is not true. That is, there are cases where $p : T \mid q : S$ is correct and yet $\bar{T} \not\leq S$, for example when $T = q!a.(?a.T + ?b.T) \oplus q!b.\text{end}$ and $S = ?a.p!a.S + ?b.\text{end}$. This is in sharp contrast with the unfair theories [12, 6], where $p : T \mid q : S$ is correct (in the “unfair” sense) if and only if $\bar{T} \leq_U S$.

5 Conclusions

Applying the unfair subtyping relation in multi-party sessions may result into a subset of participants that starve for messages that are never sent. 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 session type theories that take into account some liveness property of sessions in addition to the standard safety. Fair subtyping relations (often referred to as refinements in concurrency theory) have rightfully gained the fame of being hard to characterize completely [18, 20] or even to approximate [4, 19]. The main contribution of this paper is the definition of a self-contained, comprehensive fair theory of session types which is presented as a relatively simple variation of more familiar ones.

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).

In our theory, checking whether a session is correct can be more expensive than in “unfair” theories (Section 4). This observation substantiates the effectiveness of the design-by-contract approach advocated in [5, 17], where the session types of a multi-party 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, subtyping may be used 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 correctness (in the sense of Definition 2.1) and not just safety. We plan to investigate the issues of type checking processes using a fair subtyping relation in a future work.

Acknowledgments. This report was mostly written while the author was maître de conférences invité at Laboratoire Preuves, Programmes et Systèmes in 2010. The author is grateful to Daniele Varacca for the enlightening discussions on fairness had during his stay there.

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] Arend Rensink and Walter Vogler. Fair testing. *Information and Computation*, 205(2):125–198, 2007.
- [21] Vasco T. Vasconcelos. Fundamentals of session types. In *SFM'09*, LNCS 5569, pages 158–186. Springer, 2009.

A Proofs

We introduce some convenient notation used in this section: we write $\prod_{i=1}^n p_i : T_i$ for $p_1 : T_1 \mid \dots \mid p_n : T_n$; given a session M , we write $\text{dom}(M)$ for the tags of the participants in M , that is $\text{dom}(\prod_{i \in I} p_i : T_i) = \{p_i \mid i \in I\}$ and we write $M(p)$ for the session type associated with tag $p \in \text{dom}(M)$. We also write $\{p_1, \dots, p_n\}!a.T$ for $p_1!a \cdots p_n!a.T$ when the order of the outputs does not matter.

A.1 Proofs of Section 3

Theorem A.1 (Theorem 3.1). *Let $T, S \in \mathcal{T}_{\text{nt}}$. Then $T \leq S$ implies $T \leq_U S$.*

Proof. We show that \leq satisfies the conditions of Definition 2.3. Suppose $T \leq S$. We distinguish three possibilities according to the shape of T :

($T = \text{end}$) We have $p : \text{end} \mid q : T$ correct, hence $p : \text{end} \mid q : S$ is also correct. We conclude $S = \text{end}$, therefore item (1) of Definition 2.3 is satisfied.

($T = \sum_{i \in I} ?a_i.T_i$) Suppose $M \mid q : T_k$ correct for some $k \in I$, let $\text{dom}(M) = \{q_1, \dots, q_m\}$, and consider

$$N \stackrel{\text{def}}{=} p : q!a_k. \underbrace{?ack \cdots ?ack}_m . \text{end} \mid \prod_{i=1}^m q_i : p!ack.M(q_i)$$

where $p \notin \{q, q_1, \dots, q_m\}$ and ack is some name that does not occur anywhere. We have $N \mid q : T$ correct and $N \mid q : T \implies p : \text{end} \mid M \mid q : T_k$. From the hypothesis $T \leq S$ we deduce that $N \mid q : S$ is correct, therefore $S = \sum_{i \in I \cup J} ?a_i.S_i$. Furthermore, $N \mid q : S \implies p : \text{end} \mid M \mid q : S_k$, hence $T_i \leq S_i$ for every $i \in I$ because $k \in I$ and M are arbitrary. We conclude that item (2) of Definition 2.3 is satisfied.

($T = \bigoplus_{i \in I} p_i!a_i.T_i$) Let $\{M_i\}_{i \in I}$ be a family of systems such that $M_i \mid q : T_i$ is correct for every $i \in I$. Without loss of generality we may assume that $\text{dom}(M_i) = \text{dom}(M_j) = \{q_1, \dots, q_m\}$ for every $i, j \in I$ (if this is not the case, appropriate $p : \text{end}$ participants can be added wherever necessary). Observe that $\{p_i \mid i \in I\} \subseteq \{q_1, \dots, q_m\}$. For every $j \in \{1, \dots, m\}$, let

$$N_j \stackrel{\text{def}}{=} q_j : \sum_{i \in I, p_i = q_j} ?a_i. \{q_1, \dots, q_m\} \setminus \{q_j\}!ack_j. \\ \{q_1, \dots, q_m\} \setminus \{q_j\}!a_i.r!ack.M_i(q_j) \\ + \sum_{i \in I, p_i \neq q_j} ?ack_i. \sum_{k \in I, p_k = p_i} ?a_k.r!ack.M_k(q_j)$$

and let

$$N \stackrel{\text{def}}{=} p : \underbrace{?ack \cdots ?ack}_m . \text{end} \mid \prod_{i=1}^m N_i$$

where $p \notin \{q, q_1, \dots, q_m\}$ and $\{ack, ack_1, \dots, ack_m\}$ do not occur anywhere. We have that $N \mid q : T$ is correct and $N \mid q : T \implies p : \text{end} \mid M_i \mid q : T_i$ for every $i \in I$. From the hypothesis $T \leq S$ we deduce that $N \mid q : S$ is correct, therefore $S = \bigoplus_{i \in J} p_i!a_i.S_i$ and $J \subseteq I$. Furthermore, $N \mid q : S \implies p : \text{end} \mid M_i \mid q : S_i$ for every $i \in J$, hence $T_i \leq S_i$ for every $i \in J$ because each M_i is arbitrary. We conclude that item (3) of Definition 2.3 is satisfied. \square

Lemma A.1. Let $T, S \in \mathcal{T}_{\text{nt}}$ and $T \leq_U S$ and $M|_p : T$ correct and $M|_p : S \implies M'|_p : S'$. Then $M|_p : T \implies M'|_p : T'$ for some T' such that $T' \leq_U S'$.

Proof. By unzipping the derivation $M|_q : S \implies M'|_q : S'$ we deduce that $M \xrightarrow{\bar{s}} M'$ and $S \xrightarrow{\bar{s}} S'$ for some string s of actions. We show that there exists T' such that $T \xrightarrow{\bar{s}} T'$ and $T' \leq_U S'$ by induction on s :

- In the base case we have $s = \varepsilon$, and we conclude by taking $T' = T$ (note that S' may be a residual of S after an application of rule (T-CHOICE) but even in this case $T' \leq_U S'$ still holds).
- Suppose $s = q?as'$. Then $M \xrightarrow{q!a} M'' \xrightarrow{\bar{s}'} M'$ and $S \xrightarrow{q?a} S'' \xrightarrow{\bar{s}'} S'$ for some M'' and S'' . We deduce $S = \sum_{i \in J} ?a_i.S_i$ and $a = a_k$ and $S'' = S_k$ for some $k \in J$. From item (2) of Definition 2.3 we deduce $T = \sum_{i \in I} ?a_i.T_i$ with $I \subseteq J$. It must be the case that $k \in I$, for otherwise $M|_q : T \implies M''|_q$: fail which is incorrect. We deduce $T_k \leq_U S_k$ and we conclude by induction hypothesis.
- Suppose $s = p!as'$. Then $M \xrightarrow{p?a} M'' \xrightarrow{\bar{s}'} M'$ and $S \implies p!a S'' \xrightarrow{\bar{s}'} S'$ for some M'' and S'' . We deduce $S = \bigoplus_{i \in J} p_i!a_i.S_i$ and $p = p_k$ and $a = a_k$ for some $k \in J$. From item (3) of Definition 2.3 we deduce $T = \bigoplus_{i \in I} p_i!a_i.T_i$ and $J \subseteq I$. We deduce $T_k \leq_U S_k$ and we conclude by induction hypothesis. \square

Theorem A.2 (Theorem 3.2). Let $T, S \in \mathcal{T}_{\text{nt}}$. If $T \leq_U S$ and $M|_p : T$ is correct, then $M|_p : S \implies N \dashrightarrow$ implies $N \checkmark$.

Proof. We have $N = M'|_p : S'$ for some M' and S' . From the hypotheses $T \leq_U S$ and $M|_p : T$ correct and Lemma A.1 we deduce that $M|_p : T \implies M|_p : T'$ for some T' such that $T' \leq_U S'$. From the hypothesis $N \dashrightarrow$ we deduce $T' = S' = \text{end}$. Again from the hypothesis $M|_p : T$ correct we have $M' \checkmark$, therefore $N \checkmark$. \square

Theorem A.3 (Theorem 3.3). Let $T, S \in \mathcal{T}_{\text{nt}}$. Then $T \leq S$ if and only if $T \leq_C S$.

Proof. (“only if” part) We must show that $\leq \subseteq \leq_U \cap \prec$. The relation $\leq \subseteq \leq_U$ has already been proved in Theorem 3.1. Regarding the relation $\leq \subseteq \prec$, suppose $T \leq S$ and $M|_p : T$ correct. Then $M|_p : S \checkmark$ for otherwise $M|_p : S$ would be incorrect.

(“if” part) Suppose $T \leq_C S$ and let $M|_q : T$ be a correct session. Consider a derivation $M|_p : S \implies M'|_p : S'$. From Lemma A.1 we deduce $M|_p : T \implies M'|_p : T'$ for some T' such that $T' \leq_C S'$. From the hypothesis $M|_p : T$ correct we deduce $M'|_p : T'$ correct. By definition of \leq_C we have $T' \prec S'$, therefore we conclude $M|_p : S' \checkmark$. \square

Theorem A.4 (Theorem 3.4). Let $T \leq_U S$. Then $T \prec S$ if and only if $T - S$ is not viable.

Proof. (“only if” part) Suppose by contradiction that $T - S$ is viable and let $M|_p : T - S$ be a correct session. Then $M|_p : T$ is also correct and, from the hypothesis $T \prec S$, we deduce $M|_p : S \checkmark$. Then $M \xrightarrow{\bar{s}\checkmark}$ and $S \xrightarrow{\bar{s}}$ end for

some string s . By definition, we deduce $T - S \xrightarrow{s} \text{fail}$, thus contradicting the hypothesis that $T - S$ is viable.

(“if” part) Let $M \mid_P : T$ be a correct session and $\{\mathbf{V}_k^{T-S}\}_{k \in \mathbb{N}}$ be the viability sequence for $T - S$. We prove that $T' - S' \notin \mathbf{V}_k^{T-S}$ implies $M \mid_P : S' \xrightarrow{\checkmark}$ by induction on k .

- ($k = 0$) Since $T - S \in \mathbf{V}_0^{T-S}$ we conclude $M \mid_P : S \xrightarrow{\checkmark}$ (*ex falso quodlibet*).
- ($k > 0$ and k is odd) Then for every s such that $T - S \xrightarrow{s} \text{end}$ there exists $t \leq s$ such that $T - S \xrightarrow{t} R$ and $R \notin \mathbf{V}_{k-1}^{T-S}$. From the hypothesis $M \mid_P : T$ correct we deduce $M \xrightarrow{\bar{s}\checkmark} M'$ and $T \xrightarrow{\bar{s}\checkmark}$ for some s . Suppose that $S \xrightarrow{\bar{s}\checkmark}$, for otherwise there is nothing to prove. Then $T - S \xrightarrow{s} \text{end}$. We deduce that $T - S \xrightarrow{t} R$ with $R \notin \mathbf{V}_{k-1}^{T-S}$ for some $t \leq s$. Since T is in normal form and R is not viable, it must be the case that $R = T' - S'$ for some T' and S' such that $T \xrightarrow{t} T'$ and $S \xrightarrow{t} S'$. By induction hypothesis we deduce $M' \mid_P : S' \xrightarrow{\checkmark}$. We conclude by observing that $M \mid_P : S \xrightarrow{\checkmark} M' \mid_P : S'$.
- ($k > 0$ and k is even) We distinguish two subcases:
 - Suppose $T = \sum_{i \in I} ?a_i.T_i$ and $S = \sum_{i \in I \cup J} ?a_i.S_i$ and $T_i - S_i \notin \mathbf{V}_{k-1}^{T-S}$ for every $i \in I$. From the hypothesis $M \mid_P : T$ correct we deduce $M \xrightarrow{p!a_i} M'$ for some $i \in I$ and $M' \mid_P : T_i$ correct. By induction hypothesis we deduce $M' \mid_P : S_i \xrightarrow{\checkmark}$. We conclude by observing that $M \mid_P : S \xrightarrow{\checkmark} M' \mid_P : S_i$.
 - Suppose $T = \bigoplus_{i \in I \cup J} p_i!a_i.T_i$ and $S = \bigoplus_{i \in I} p_i!a_i.S_i$ and $T_i - S_i \notin \mathbf{V}_{k-1}^{T-S}$ for some $i \in I$, because T_i is viable for every $i \in J \setminus I$ since T is in normal form. From the hypothesis $M \mid_P : T$ correct we deduce $M \xrightarrow{?p_i a_i} M'$ for some M' such that $M' \mid_P : T_i$ is correct. By induction hypothesis we deduce $M' \mid_P : S_i \xrightarrow{\checkmark}$. We conclude by observing that $M \mid_P : S \xrightarrow{\checkmark} M' \mid_P : S_i$. \square

A.2 Proofs of Section 4

Theorem A.5 (Theorem 4.1). $T \in \mathcal{T}_V$ if and only if $T \in \text{viables}(T)$.

Proof. (“only if” part) We show that $S \in \text{trees}(T) \setminus \mathbf{V}_i^T$ implies that S is not viable by induction on i .

- ($i = 0$) There is nothing to prove since $\text{trees}(T) \setminus \mathbf{V}_0^T = \emptyset$.
- ($i > 0$ and i is odd) Then for all s such that $S \xrightarrow{s} \text{end}$ there exist $t \leq s$ and S' such that $S \xrightarrow{t} S'$ and $S' \in \text{trees}(T) \setminus \mathbf{V}_{i-1}^T$. Suppose, by contradiction, that S is viable. Then there exists M such that $M \mid_P : S$ is correct, hence $M \mid_P : S \xrightarrow{\checkmark} N \mid_P : \text{end}$ where $N \xrightarrow{\checkmark}$. We deduce that $M \xrightarrow{\bar{s}\checkmark} N$ and $S \xrightarrow{\bar{s}\checkmark} \text{end}$ for some finite string s of actions. Then there exists $t \leq s$ such that $S \xrightarrow{t} S'$ and $S' \notin \mathbf{V}_{i-1}^T$. By induction hypothesis we deduce that S' is not viable. This contradicts the hypothesis that $M \mid_P : S$ is correct, hence that S is viable.

- ($i > 0$ and i is even) Suppose $S \in \mathbf{V}_{i-1}^T \setminus \mathbf{V}_i^T$. The session type S cannot be **end**, for **end** subtrees are always preserved along the sequence $\{\mathbf{V}_i^T\}_{i \in \mathbb{N}}$. Suppose, by contradiction, that S is viable and that $M|_p : S$ is correct. We distinguish two subcases according to the form of S :
 - ($S = \sum_{j \in I} ?a_j.S_j$ and $S_j \notin \mathbf{V}_{i-1}^T$ for every $j \in I$). It must be the case that $M \xrightarrow{p!a_k} M''$ for some $k \in I$ and $M''|_p : S_k$ is correct. But this is absurd, because by induction hypothesis we deduce that S_k is not viable.
 - ($S = \bigoplus_{j \in I} p_j!a_j.S_j$ and $S_k \notin \mathbf{V}_{i-1}^T$ for some $k \in I$). Dual of the previous case.

(“if” part) We must define a system M such that $M|_p : T$ is correct under the hypothesis $T \in \text{viables}(T)$. Let $\{p_1, \dots, p_n\}$ be the set of roles occurring in T and let q be another role different from them (and from p) which will be a controller participant. We also need a set $\{ack_1, \dots, ack_n\}$ of distinct names that do not occur anywhere else. The controller is defined by the equations:

$$\begin{aligned} \text{end} \downarrow q &= \text{end} \\ \sum_{i \in I} ?a_i.T_i \downarrow q &= \bigoplus_{i \in I, T_i \in \text{viables}(T_i)} \{p_1, \dots, p_n\}!a_i.p!a_i.(T_i \downarrow q) \\ \bigoplus_{i \in I} p_i!a_i.T_i \downarrow q &= \sum_{i=1}^n ?ack_i. \sum_{j \in I, p_i=p_j} ?a_j.\{p_1, \dots, p_n\} \setminus \{p_i\}!ack_i. \\ &\quad \{p_1, \dots, p_n\} \setminus \{p_i\}!a_j.(T_j \downarrow q) \end{aligned}$$

Basically, q is the only one sending messages to p and it notifies every p_i with the same messages it sends to p . Dually, every p_i immediately notifies the controller of the message it has received and the controller propagates the information to all the other roles. Each p_k is defined by the equations:

$$\begin{aligned} \text{end} \downarrow p_k &= \text{end} \\ \sum_{i \in I} ?a_i.T_i \downarrow p_k &= \sum_{i \in I, T_i \in \text{viables}(T_i)} ?a_i.(T_i \downarrow p_k) \\ \bigoplus_{i \in I} p_i!a_i.T_i \downarrow p_k &= \sum_{i \in I, p_i=p_k} ?a_i.q!ack_k.q!a_i.(T_i \downarrow p_k) \\ &\quad + \sum_{i=1}^n ?ack_i. \sum_{j \in I, p_i=p_j} ?a_j.(T_j \downarrow p_k) \end{aligned}$$

Observe that every projection is well defined because of the definition of $\text{viables}(T)$. Now consider

$$M \stackrel{\text{def}}{=} \prod_{i=1}^n p_i : T \downarrow p_i \mid q : T \downarrow q$$

It is a simple exercise to verify that $M|_p : T$ is correct and this concludes the proof. \square

Theorem A.6 (Theorem 4.2). *If $T \in \mathcal{T}$, then $\text{nf}(T)$ is in normal form and $T \leq \text{nf}(T)$.*

Proof. Let $\text{traces}(T) \stackrel{\text{def}}{=} \{s \mid T \xrightarrow{s} \text{end}\}$. We have $\text{traces}(\text{nf}(T)) \subseteq \text{traces}(T)$, therefore $\text{nf}(T) \leq T$. Consider M such that $M|_p : T$ is correct and consider a derivation $M|_p : \text{nf}(T) \Longrightarrow M'|_p : R$. Then there exists s such that $M \xrightarrow{s} M'$ and $\text{nf}(T) \xrightarrow{s} R$. From $\text{traces}(\text{nf}(T)) \subseteq \text{traces}(T)$ we deduce that there exists T' such that $T \xrightarrow{s} T'$ and $R = \text{nf}(T')$. From the hypothesis $M|_p : T$ correct we deduce that $M'|_p : T' \xrightarrow{\checkmark}$, namely that $M' \xrightarrow{\checkmark}$ and $T' \xrightarrow{\checkmark}$ for some

string t . Furthermore, $T' \xrightarrow{s'} T''$ implies T'' viable for every $s' \leq t$. Therefore $\text{nf}(T') \xrightarrow{t\checkmark} \text{nf}(T'')$ by definition of $\text{nf}(T')$. We conclude $M' \upharpoonright_{\mathcal{P}} : R \xrightarrow{\checkmark}$. \square

Theorem A.7 (Theorem 4.3). *Let $T, S \in \mathcal{T}_{\text{nf}}$. Then $T \leq S$ if and only if $T \leq_A S$.*

Proof. The “only if” part is immediate from Theorem 3.3. Regarding the “if” part, it is obvious that $\leq_A \subseteq \leq_U$, therefore we only have to show that whenever $T \leq_A S$ is derivable we have $T \prec S$. By Theorem 3.4, this is equivalent to checking that $T - S$ is not viable. From $T \leq_A S$ we deduce that every $T' - S'$ subtree occurring in $T - S$ and corresponding to an application of rule (FS-OUTPUT) is not viable. Therefore, $T - S \leq R$ where R is the same as $T - S$ where every such subtree has been replaced by **fail**. Now $\text{end} \notin \text{trees}(R)$ by definition of $T - S$, therefore $T - S$ is not viable. \square

Proposition A.1 (Proposition 3.1). *The following properties hold: (1) $\mathcal{T}_{\text{nf}} \subseteq \mathcal{T}_v$; (2) for every $T \in \mathcal{T}_v$ there exists $S \in \mathcal{T}_{\text{nf}}$ such that $T \leq S$.*

Proof. Item (2) is an immediate consequence of Theorem 4.2. Regarding item (1), let $\{\mathbf{V}_i^T\}_{i \in \mathbb{N}}$ be the sequence of sets resulting from the construction in Section 4. It is easy to verify that $\mathbf{V}_i^T = \text{trees}(T)$ for every $i \in \mathbb{N}$. We conclude that T is viable. \square