Inference Systems with Corules for Fair Subtyping and Liveness Properties of Binary Session Types

Luca Ciccone
University of Torino, Italy

Luca Padovani
University of Torino, Italy

Abstract

Many properties of communication protocols stem from the combination of safety and liveness properties. Characterizing such combined properties by means of a single inference system is difficult because of the fundamentally different techniques (coinduction and induction, respectively) usually involved in defining and proving them. In this paper we show that Generalized Inference Systems allow for simple and insightful characterizations of (at least some of) these combined inductive/coinductive properties for dependent session types. In particular, we illustrate the role of corules in characterizing weak termination (the property of protocols that can always eventually terminate), fair compliance (the property of interactions that can always be extended to reach client satisfaction) and also fair subtyping, a liveness-preserving refinement relation for session types.

2012 ACM Subject Classification Theory of computation → Axiomatic semantics; Theory of computation → Type structures; Theory of computation → Program specifications

Keywords and phrases Inference systems, session types, safety, liveness, induction, coinduction

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.125

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Supplementary Material The Agda formalization of the notions and results presented in the paper is available at Model (Agda Formalization): https://github.com/boystrange/FairSubtypingAgda/tree/v1.0 archived at swh:1:rev:6d00f452a8380c2aacc659b2a69b9f4b3accfa27

Acknowledgements We are grateful to Francesco Dagnino for his feedback on an early version of this paper. The anonymous ICALP reviewers provided useful comments and suggestions that helped us improving the paper.

1 Introduction

Session types [21, 22, 4, 23] describe communication protocols at the type level. By making sure that processes use session channels according to their session type, a session type system enables the modular enforcement of various desirable properties, including the absence of communication errors, protocol fidelity and in some cases deadlock freedom. These are all examples of safety properties, which are informally identified by the motto “nothing bad ever happens” [28]. Less frequent are (session) type systems also enforcing liveness properties, those identified by the motto “something good eventually happens”. In a network of communicating processes, a typical example of liveness property is the fact that a protocol or a process can always eventually terminate. It is well known that characterizations and proofs of safety and liveness properties rely on fundamentally different (dual) techniques [24, 2, 3, 8]: safety properties are usually based on invariance (coinductive) arguments, whereas liveness properties are usually based on well foundedness (inductive) arguments. As a consequence, it is generally difficult to characterize and enforce complex properties that exhibit a mixture of safety and liveness by means of a single inference system (such as a session type system), in which the inference rules are interpreted either all inductively or all coinductively.
To tame such difficulty, in this work we advocate the use of Generalized Inference Systems (GISs) [5, 16]. A GIS allows for the definition of predicates that are a fixed point of the inference operator associated with the inference system, but not necessarily the least or the greatest one. This is made possible by coaxioms and corules, whose purpose is to provide an inductive definition of a space within which a coinductive definition is used. A recurring example in the literature of GISs is the predicate $\text{maxElem}(l, x)$, asserting that $x$ is the maximum element of a possibly infinite list $l$. If we consider the inference system

\[
\begin{array}{c}
\text{maxElem}(l, y) \\
\text{maxElem}(x :: \Lambda, x) \\
\text{maxElem}(x :: l, \text{max}\{x, y\})
\end{array}
\]

where $\Lambda$ denotes the empty list and :: is the constructor, we can give two natural interpretations to these rules. The inductive one, obtained by taking the least fixed point of the inference operator, restricts the set of derivable judgments to those for which there is a well-founded derivation tree. In this case, the $\text{maxElem}$ predicate is sound but not complete, since it does not hold for any infinite list, even those for which the maximum exists. The coinductive interpretation of these rules, obtained by taking the greatest fixed point of the inference operator, allows us to derive judgments by means of non-well-founded derivation trees. In this case, the $\text{maxElem}$ predicate is complete but not sound. In particular, it becomes possible to derive any judgment $\text{maxElem}(l, x)$ where $x$ is greater than the elements of the list, but is not an element of the list.

This is one situation in which the sought predicate is neither the least nor the greatest fixed point of a given inference operator, but is somewhere “in between” the two extremes. We can repair the above inference system by adding the following coaxiom:

\[
\text{maxElem}(x :: l, x)
\]

Read naively, this coaxiom seems to assert that the first element of any list is also its maximum. In the context of a GIS, however, its effect is that of ruling out those judgments $\text{maxElem}(l, x)$ in which $x$ is not an element of the list. In a sense, the coaxiom adds a well-foundedness element to the derivability of a judgment $\text{maxElem}(l, x)$, by requiring that $x$ must be found in – at some finite distance from the head of – the list $l$.

The main contribution of this work is the realization that corules can be used to easily characterize properties of session types involving a mixture of coinduction/safety and induction/liveness. We consider three such properties: weak termination (the property of protocols that can always eventually terminate), fair compliance (the property of client/server interactions that can always be extended to reach client satisfaction) and fair subtyping (a liveness-preserving refinement relation for session types). We show how to provide sound and complete characterizations of these properties just by adding a few corules to the inference systems of their “unfair” counterparts, those focusing on safety but neglecting liveness. Not only the added corules shed light on the liveness(-preserving) property of interest, but we can conveniently appeal to the bounded coinduction principle of GISs [5] to prove the completeness of the provided characterizations, thus factoring out a significant amount of work. We also make two side contributions. First, the aforementioned characterizations are given for a family of dependent session types [33, 34, 32, 14] in which the length and structure of the protocol may depend in non-trivial ways on the content of exchanged messages. Thus, we extend previously given characterizations of fair subtyping [29, 30] to a much larger class of protocols. Second, we provide an Agda [27] formalization of all the notions and results stated in the paper. In particular, we give the first machine-checked formalization of a liveness-preserving refinement relation for (dependent) session types.
The rest of the paper is structured as follows. We quickly recall the key definitions of GISs in Section 2 and describe syntax and semantics of (dependent) session types in Section 3. We define and characterize weak termination, fair compliance and fair subtyping in Sections 4–6 and conclude in Section 7. The Agda formalization is accessible from a public repository [15].

2 Generalized Inference Systems

In this section we briefly recall the key notions of Generalized Inference Systems (GISs). In particular, we see how GISs enable the definition of predicates whose purely (co)inductive interpretation does not yield the intended meaning and we review the canonical technique to prove the completeness of a defined predicate with respect to a given specification. Further details on GISs may be found in the existing literature [5, 16].

An inference system $I$ over a universe $U$ of judgments is a set of rules, which are pairs $\langle \text{pr}, j \rangle$ where $\text{pr} \subseteq U$ is the set of premises of the rule and $j \in U$ is the conclusion of the rule. A rule without premises is called axiom. Rules are typically presented using the syntax

$$
\frac{\text{pr}}{j}
$$

where the line separates the premises (above the line) from the conclusion (below the line).

▶ Remark 2.1. In many cases, and in this paper too, it is convenient to present inference systems using meta-rules instead of rules. A meta-rule stands for a possibly infinite set of rules, which are obtained by instantiating the meta-variables occurring in the meta-rule. For example, the rules for $\text{maxElem}$ discussed in Section 1 are meta-rules referring to the meta-variables $x, y$ and $l$. The actual rules of the discussed inference system result from all possible instantiations of such meta-variables with numbers and (possibly infinite) lists. In the rest of the paper we will not insist on this distinction and we will use “(co)rule” even when referring to meta-(co)rules. If necessary, we will use side conditions to constrain the valid instantiations of the meta-variables occurring in such meta-(co)rules.

An interpretation of an inference system $I$ identifies a subset of $U$ whose elements are called derivable judgments. To define the interpretation of an inference system $I$, consider the inference operator associated with $I$, which is the function $F_I : \wp(\wp U) \to \wp(\wp U)$ such that

$$
F_I(X) = \{ j \in U \mid \exists \text{pr} \subseteq X : \langle \text{pr}, j \rangle \in I \}
$$

for every $X \subseteq U$. Intuitively, $F_I(X)$ is the set of judgments that can be derived in one step from those in $X$ by applying a rule of $I$. Note that $F_I$ is a monotone endofunction on the complete lattice $\wp(U)$, hence it has least and greatest fixed points.

▶ Definition 2.2. The inductive interpretation $\text{ind}[I]$ of an inference system $I$ is the least fixed point of $F_I$ and the coinductive interpretation $\text{coind}[I]$ is the greatest one.

From a proof theoretical point of view, $\text{ind}[I]$ and $\text{coind}[I]$ are the sets of judgments derivable with well-founded and non-well-founded proof trees, respectively.

Generalized Inference Systems enable the definition of (some) predicates for which neither the inductive interpretation nor the coinductive one give the expected meaning.

▶ Definition 2.3 (generalized inference system). A generalized inference system is a pair $\langle I, I_{\text{co}} \rangle$ where $I$ and $I_{\text{co}}$ are inference systems whose elements are called rules and corules, respectively. The interpretation of a generalized inference system $\langle I, I_{\text{co}} \rangle$, denoted by $\text{Gen}[I, I_{\text{co}}]$, is the greatest post-fixed point of $F_I$ that is included in $\text{ind}[I \cup I_{\text{co}}]$. 

ICALP 2021
From a proof-theoretical point of view, a GIS $⟨I, I_{co}⟩$ identifies the set of judgments that are derivable with an arbitrary (not necessarily well-founded) proof tree in $I$ and whose nodes (the judgments occurring in the proof tree) are all derivable with a well-founded proof tree in $I \cup I_{co}$. Recalling $\text{maxElem}$ from Section 1, the judgments referring to a “maximum” which does not belong to the list are ruled out since they cannot be derived using a well-founded proof tree in the inference system with the coaxiom.

Consider now a specification $S \subseteq U$ that contains the valid judgments. We can relate $S$ to the interpretation of a (generalized) inference system using one of the following proof principles. The induction principle allows us to prove the soundness of an inductively defined predicate by showing that $S$ is closed with respect to $I$. That is, whenever the premises of a rule of $I$ are all in $S$, then the conclusion of the rule is also in $S$.

\textbf{Proposition 2.4.} If $F_I(S) \subseteq S$, then $\text{Ind}[I] \subseteq S$.

The coinduction principle allows us to prove the completeness of a coinductively defined predicate by showing that $S$ is consistent with respect to $I$. That is, every judgment of $S$ is the conclusion of a rule whose premises are also in $S$.

\textbf{Proposition 2.5.} If $S \subseteq F_I(S)$, then $S \subseteq \text{Colnd}[I]$.

The bounded coinduction principle allows us to prove the completeness of a predicate defined by a generalized inference system $⟨I, I_{co}⟩$. In this case, one needs to show not only that $S$ is consistent with respect to $I$, but also that $S$ is bounded by the inductive interpretation of the inference system $I \cup I_{co}$. Formally:

\textbf{Proposition 2.6.} If $S \subseteq \text{Ind}[I \cup I_{co}]$ and $S \subseteq F_I(S)$, then $S \subseteq \text{Gen}[I, I_{co}]$.

Proving the boundedness of $S$ amounts to proving the completeness of $I \cup I_{co}$ (inductively interpreted) with respect to $S$. All of the GISs that we are going to discuss in Sections 4–6 are proven complete using the bounded coinduction principle.

### 3 Syntax and Semantics of Dependent Session Types

We assume a set $V$ of values that can be exchanged in communications. This set may include booleans, natural numbers, strings, and so forth. Hereafter, we assume that $V$ contains at least two elements, otherwise branching protocols cannot be described and the theoretical development that follows becomes trivial. We use $x, y, z$ to range over the elements of $V$.

We define the set $S$ of dependent session types over $V$ using coinduction, to account for the possibility that session types (and the protocols they describe) may be infinite.

\textbf{Definition 3.1.} Let $S$ be the largest set such that $T \in S$ implies either $T = \text{nil}$ or $T = ?f$ or $T = !f$ where $f \in V \rightarrow S$ is a total function from $V$ to $S$. We use $T$, $S$ and $R$ to range over elements of $S$ and $f, g$ to range over elements of $V \rightarrow S$. We say that $T \in S$ is a (dependent) session type over $V$ and that $f \in V \rightarrow S$ is a continuation over $V$.

A session type can be of three forms. An input session type $?f$ describes a channel used first for receiving a message $x \in V$ and then according to $f(x)$. An output session type $!f$ describes a channel used first for sending a message $x \in V$ and then according to $f(x)$. Finally, the session type $\text{nil}$ describes an unusable session channel. As we will see shortly, we use $\text{nil}$ in combination with input and output session types to describe unexpected inputs and impossible outputs. We call the functions $f$ continuations since they take as input a message (the one being exchanged on the session channel) and compute the session type that describes the usage of the channel after the exchange. Continuations allow us to describe
protocols whose structure depends on previously exchanged messages. We do not detail
the concrete language in which continuations are specified, but in practice they will be a
small subset of the computable functions. For example, in our Agda formalization [15, file
SessionType.agda] continuations are well-typed Agda functions. This way, we can leverage
on Agda and its library [27] for constructing dependent session types.

\textbf{Remark 3.2.} Session types as defined in Definition 3.1 are isomorphic to the possibly
infinite trees coinductively generated by the productions

$$S, T ::= \text{nil} \mid ?\{x : T_x\}_{x \in \mathbb{V}} \mid !\{x : T_x\}_{x \in \mathbb{V}}$$

where we represent continuations \(f\) with their graph \(\{x : f(x)\}_{x \in \mathbb{V}}\). The structure of session
types we have chosen in Definition 3.1, besides suggesting an effective representation of
dependent session types, is also aimed at streamlining as much as possible not only the
syntax but also the semantics of session types. In the end, this choice allowed us to reduce
the inevitable complexity bloat that we had to face when formalizing these notions and the
related proofs in Agda [15].

It is convenient to introduce some notation for presenting session types in a more readable
and familiar form. Given a set \(X \subseteq \mathbb{V}\) of values, we write \(X.T\) for the continuation

$$(X.T)(x) \overset{\text{def}}{=} \begin{cases} T & \text{if } x \in X \\ \text{nil} & \text{otherwise} \end{cases}$$

so that we can write session types like \(!\mathbb{B}.T\) (send a boolean and continue as \(T\)) and \(?\mathbb{N}.S\)
(receive a natural number and continue as \(S\)). We abbreviate \(\{x\}\) with \(x\) when no confusion
may arise. So we write \(!\text{true}.T\) instead of \(!\{\text{true}\}.T\). We let \(?\text{end} \overset{\text{def}}{=} \emptyset.\text{nil}\) and \(!\text{end} \overset{\text{def}}{=} !\emptyset.\text{nil}\). Both \(?\text{end}\) and \(!\text{end}\) describe session channels on which no further communications may occur,
although they differ slightly with respect to the session types they can be safely combined with
(more on this later). We also define a partial binary operation \(\sqcup\) on session types
such that \(\text{nil} \sqcup T = T \sqcup \text{nil} = T\) and that is undefined otherwise. We extend this operation
pointwise to continuations, so that \((f \sqcup g)(x) = f(x) \sqcup g(x)\). It is intended that \(f \sqcup g\) is
undefined if so is \(f(x) \sqcup g(x)\) for some \(x \in \mathbb{V}\). We can now express the familiar external and
internal choice of session types as the (partial) operations \(+\) and \(\oplus\) defined by

\[ ?f + ?g \overset{\text{def}}{=} ?(f \sqcup g) \quad !f \oplus !g \overset{\text{def}}{=} !(f \sqcup g) \]

It is easy to see that \(+\) and \(\oplus\) are commutative and associative and respectively have
\(?\text{end}\) and \(!\text{end}\) as units. We assume that they bind less tightly than the “.” in continuations.
Finally, we let \(\text{dom}(f) \overset{\text{def}}{=} \{x \in \mathbb{V} \mid f(x) \neq \text{nil}\}\) be the proper domain of the continuation \(f\),
namely the set of messages for which \(f\) yields a session type other than \(\text{nil}\).

\textbf{Example 3.3.} The session types \(T_1\) and \(S_1\) that satisfy the equations

\[ T_1 = !\text{true}!\mathbb{N}.T_1 \oplus !\text{false} ?\text{end} \quad S_1 = !\text{true}!\mathbb{N}^+ .S_1 \oplus !\text{false} ?\text{end} \]

both describe a channel used for sending a boolean. If the boolean is \text{false}, the communication
stops immediately (?\text{end}). If it is \text{true}, the channel is used for sending a natural number (a
strictly positive one in \(S_1\)) and then according to \(T_1\) or \(S_1\) again. Notice how the structure of the
protocol after the output of the boolean depends on the value of the boolean.

The session types \(T_2\) and \(S_2\) that satisfy the equations

\[ T_2 = ?\text{true}!\mathbb{N}.T_2 + ?\text{false} ?\text{end} \quad S_2 = ?\text{true}!\mathbb{N}^+ .S_2 + ?\text{false} ?\text{end} \]

differ from \(T_1\) and \(S_1\) in that the channel they describe is used initially for receiving a boolean.
As a final example, the session type $T_3 = !f$ where

$$f(0) = T_3 \quad f(n+1) = !\exists f(n)$$

describes an channel used for sending streams of sequences beginning with a natural number $n$ followed by $n$ boolean messages.

We define the operational semantics of session types by means of a labeled transition system. Labels, ranged over by $\alpha$, $\beta$, $\gamma$, have either the form $?x$ (input of message $x$) or the form $!x$ (output of message $x$). Transitions $T \xrightarrow{\alpha} S$ are defined by the following axioms:

\[
\begin{align*}
[T\text{-input}] & \quad \exists f \xrightarrow{?x} f(x) \\
[T\text{-output}] & \quad !f \xrightarrow{!x} f(x) \quad x \in \text{dom}(f)
\end{align*}
\]

There is a fundamental asymmetry between send and receive operations: the act of sending a message is active – the sender may choose the message to send – while the act of receiving a message is passive – the receiver cannot cherry-pick the message being received. We model this asymmetry with the side condition $x \in \text{dom}(f)$ in $[T\text{-output}]$ and the lack thereof in $[T\text{-input}]$: a process that uses a session channel according to $!f$ refrains from sending a message $x$ if $x \notin \text{dom}(f)$, whereas a process that uses a session channel according to $?f$ cannot decide which message $x$ it will receive, but the session channel becomes unusable if an unexpected message arrives. These transition rules allow us to appreciate a little more the difference between $!\text{end}$ and $?\text{end}$. While both describe a session endpoint on which no further communications may occur, $!\text{end}$ is “more robust” than $?\text{end}$ since it has no transitions, whereas $?\text{end}$ is “more fragile” than $!\text{end}$ since it performs transitions, all of which lead to $\text{nil}$. For this reason, we use $!\text{end}$ to flag successful session termination (Section 5), whereas $?\text{end}$ only means that the protocol has ended.

To describe sequences of consecutive transitions performed by a session type we use another relation $\xrightarrow{\pi}$ where $\varphi$ and $\psi$ range over strings of labels. As usual, $\varepsilon$ denotes the empty string and juxtaposition denotes string concatenation. The relation $\xrightarrow{\pi}$ is the least one such that $T \xrightarrow{\pi} T$ and if $T \xrightarrow{\alpha} S$ and $S \xrightarrow{\pi} R$, then $T \xrightarrow{\alpha\pi} R$.

## 4 Weak Termination

A session type is weakly terminating if it preserves the possibility of reaching $!\text{end}$ or $?\text{end}$ along all of its transitions that do not lead to $\text{nil}$. Weak termination of $T$ does not necessarily imply that there exists an upper bound to the length of communications that follow the protocol $T$, but it guarantees the absence of “infinite loops” whereby the communication is forced to continue forever.

To formalize weak termination we need the notion of trace, which is a finite sequence of actions performed on a session channel while preserving usability of the channel.

\[
\begin{definition}[traces and maximal traces] The traces of $T$ are defined as $\text{tr}(T) \triangleq \{ \varphi \mid \exists S : T \xrightarrow{\psi} S \neq \text{nil} \}$. We say that $\varphi \in \text{tr}(T)$ is maximal if $\varphi\psi \in \text{tr}(T)$ implies $\psi = \varepsilon$.
\end{definition}

For example, we have $\text{tr}(\text{nil}) = \emptyset$ and $\text{tr}(!\text{end}) = \text{tr}(?\text{end}) = \{ \varepsilon \}$. Note that $!\text{end}$ and $?\text{end}$ have the same traces but different transitions (hence different behaviors). A maximal trace is a trace that cannot be extended any further. For example $\varepsilon$ is a maximal trace of both $!\text{end}$ and $?\text{end}$ but not of $!\exists$. $?\text{end}$ whereas $!\text{true}$ and $!\text{false}$ are maximal traces of $!\exists$.?
The notion of “successful interaction” that we consider is biased towards client satisfaction, with the witness messages of $T$.

That is, $\text{weakly terminating}$. Finally, $\text{maximal trace}$ of $T$.

Example 4.3. All of the session types presented in Example 3.3 except $T_3$ are weakly terminating. The session type $T_3$ is not weakly terminating because no trace of $T_3$ can be extended to a maximal one. Note that also $S_3 \not\equiv \text{true} \parallel \text{false} \odot \text{end}$ is not weakly terminating, even though there is a path leading to $\text{end}$, because weak termination must be preserved along all possible transitions of the session type, whereas $S_3$ is not weakly terminating. Finally, $\text{nil}$ is trivially weakly terminating since it has no trace.

To find an inference system for weak termination observe that the set $W$ of weak terminating session types is the largest one that satisfies the following two properties: (1) it must be possible to reach either $\text{end}$ or $\text{end}$ from every $T \in W \setminus \{\text{nil}\}$; (2) the set $W$ must be closed by transitions, namely if $T \in W$ and $T \xrightarrow{\alpha} S$ then $S \in W$. Neither of these two properties, taken in isolation, suffices to define $W$: the session type $S_3$ from Example 4.3 enjoys property (1) but is not weakly terminating; the set $S$ is obviously the largest one with property (2), but not every session type is weakly terminating. This suggests the definition of $W$ as the largest subset of $S$ satisfying (2) and whose elements are bounded by property (1), which is precisely what corules allow us to specify.

Figure 1 shows a GIS $\langle T, T_{co} \rangle$ for weak termination, where $T$ consists of all the (singly-lined) rules whereas $T_{co}$ consists of all the (doubly-lined) corules (we will follow these naming and syntactic conventions also in the subsequent GISs). The axiom $\text{nil} \Downarrow$ indicates that $\text{nil}$ is weakly terminating in a trivial way (it has no trace), while rules $\text{in}$ and $\text{out}$ indicate that weak termination is closed by transitions. Note that these three rules, interpreted coinductively, are satisfied by all session types, hence $\{T \mid T \Downarrow \in \text{Colnd}[T] \}=S$.

Theorem 4.4. $T$ is weakly terminating if and only if $T \Downarrow \in \text{Gen}[T, T_{co}]$.

The proof of the “if” part of Theorem 4.4 crucially relies on the corules to extend each trace of $T$ to a maximal one. Indeed, suppose $T \Downarrow \in \text{Gen}[T, T_{co}]$ and consider a trace $\varphi \in \text{tr}(T)$. That is, $T \xrightarrow{\varphi} S$ for some $S \neq \text{nil}$. Using $\text{in}$ and $\text{out}$ we deduce $S \Downarrow \in \text{Gen}[T, T_{co}]$ by means of a simple induction on $\varphi$. Now $S \Downarrow \in \text{Gen}[T, T_{co}]$ implies $S \Downarrow \in \text{Ind}[T \cup T_{co}]$ by Definition 2.3. Another induction on the (well-founded) derivation of this judgment, along with the witness messages of $\text{co-in}$ and $\text{co-out}$, allows us to find $\psi$ such that $\varphi\psi$ is a maximal trace of $T$.

5 Compliance

In this section we define and characterize two compliance relations for session types, which formalize the “successful” interaction between a client and a server connected by a session. The notion of “successful interaction” that we consider is biased towards client satisfaction, but see Remark 6.7 below for a discussion about alternative notions. To formalize compliance we need to model the evolution of a session as client and server interact. To this aim, we represent a session as a term $R \parallel T$ where $R$ describes the behavior of the client and $T$ that of the server. Sessions reduce according to the rule

$$
\begin{array}{c}
\text{nil} \Downarrow \\
\text{in} \Downarrow \\
\text{out} \Downarrow \\
\text{co-in} \Downarrow \\
\text{co-out} \Downarrow
\end{array}
\quad
\begin{array}{c}
f(x) \Downarrow \quad (\forall x \in \mathcal{V}) \\
f(x) \Downarrow \quad (\forall x \in \mathcal{V}) \\
x \in \text{dom}(f) \\
x \in \text{dom}(f)
\end{array}
$$

Figure 1 Generalized inference system $\langle T, T_{co} \rangle$ for weak termination.
\[ R \parallel T \rightarrow R' \parallel S' \] if \( R \xrightarrow{\alpha} R' \) and \( T \xrightarrow{\sigma} T' \)

where \( \sigma \) is the complementary action of \( \alpha \) defined by \( \overline{\sigma(x)} = !x \) and \( \overline{\sigma} = ?x \). We extend \( \sigma \) to traces in the obvious way and we write \( \Rightarrow \) for the reflexive, transitive closure of \( \rightarrow \). We write \( R \parallel T \rightarrow \) if \( R \parallel T \rightarrow R' \parallel T' \) for some \( R' \) and \( T' \) and \( R \parallel T \rightarrow \) if not \( R \parallel T \rightarrow \).

The first compliance relation that we consider requires that, if the interaction in a session stops, it is because the client “is satisfied” and the server “has not failed” (recall that a session type can turn into \( \text{nil} \) only if an unexpected message is received). Formally:

> **Definition 5.1** (compliance). We say that \( R \) is compliant with \( T \) if \( R \parallel T \Rightarrow R' \parallel T' \Rightarrow \) implies \( R' = \text{end} \) and \( T' \neq \text{nil} \).

This notion of compliance is an instance of safety property in which the invariant being preserved at any stage of the interaction is that either client and server are able to synchronize further, or the client is satisfied and the server has not failed.

The second compliance relation that we consider adds a liveness requirement namely that, no matter how long client and server have been interacting with each other, it is always possible to reach a configuration in which the client is satisfied and the server has not failed.

> **Definition 5.2** (fair compliance). We say that \( R \) is fair compliant with \( T \) if \( R \parallel T \Rightarrow R' \parallel T' \Rightarrow \) implies \( R' \parallel T' \Rightarrow \text{end} \) with \( T'' \neq \text{nil} \).

It is easy to show that fair compliance implies compliance, but there exist compliant session types that are not fair compliant, as illustrated in the following example.

> **Example 5.3.** Recall Example 3.3 and consider the session types \( R_1 \) and \( R_2 \) such that

\[ R_1 = ?\text{true.}\text{?N.} R_1 + ?\text{false.} \text{!end} \quad R_2 = \text{true.}(?\text{0.} \text{!end} + ?\text{N}.R_2) \]

Then \( R_1 \) is fair compliant with both \( T_1 \) and \( S_1 \) and \( R_2 \) is compliant with both \( T_2 \) and \( S_2 \). Even if \( S_1 \) exhibits fewer behaviors compared to \( T_1 \) (it never sends \( 0 \) to the client), at the beginning of a new iteration it can always send \text{false} and steer the interaction along a path that leads \( R_1 \) to success. On the other hand, \( R_2 \) is fair compliant with \( T_2 \) but not with \( S_2 \). In this case, the client insists on sending \text{true} to the server in hope to receive \( 0 \), while this is possible with the server \( T_2 \), the server \( S_2 \) only sends strictly positive numbers.

This example also shows that weak termination of both client and server is not sufficient, in general, to guarantee fair compliance. Indeed, both \( R_2 \) and \( S_2 \) are weakly terminating, but they are not fair compliant. The reason is that the sequences of actions leading to \( \text{!end} \) on the client side are not necessarily the same (complemented) traces that lead to \( \text{?end} \) on the server side. Fair compliance takes into account the synchronizations that can actually occur between client and server.

Figure 2 presents the GIS \( \langle \mathcal{C}, \mathcal{C}_{\alpha} \rangle \) for fair compliance. Rule \([\text{win}]\) relates a satisfied client with a non-failed server. Rules \([\text{in-out}]\) and \([\text{out-in}]\) require that, no matter which message is exchanged between client and server, the respective continuations are still fair compliant. The side conditions \( \text{dom}(f) \neq \emptyset \) and \( \text{dom}(g) \neq \emptyset \) guarantee progress by making sure that the sender is capable of sending at least one message. As we will see, the coinductive interpretation of \( \mathcal{C} \), which consists of these three rules, completely characterizes compliance (Definition 5.1). However, these rules do not ensure that the interaction between client and server can always reach a successful configuration as required by Definition 5.2. For this, the corules \([\text{co-in-out}]\) and \([\text{co-out-in}]\) are essential.
The notions of compliance given in Section 5 induce corresponding semantic notions of subtyping, which embed a substitution principle for session types. The key idea is the same used for defining testing equivalences for processes [26, 20, 31], except that we use the term “client” instead of the term “test”. Therefore, we would be able to derive the judgment $R_2 \vdash S_2$ despite the fact that $R_2$ is not fair compliant with $S_2$ (Example 5.3). At the same time, if we treated corules as plain rules, we would be able to derive the judgment $!N.?\!f \vdash ?0.?\!f$ despite the reduction $!N.?\!f \parallel ?0.?\!f \rightarrow !\!f \parallel \text{nil}$ since there exists an interaction that leads to the successful configuration $!\!f \parallel ?\!f$ (if the client sends 0) but none of the others does.

6 Subtyping

The notions of compliance given in Section 5 induce corresponding semantic notions of subtyping, which embed a substitution principle for session types. The key idea is the same used for defining testing equivalences for processes [26, 20, 31], except that we use the term “client” instead of the term “test”. Therefore, $T$ is a subtype of $S$ if any client that successfully interacts with $T$ does so with $S$ as well.

- **Definition 6.1 (subtyping).** We say that $T$ is a subtype of $S$ if $R$ compliant with $T$ implies $R$ compliant with $S$ for every $R$.

- **Definition 6.2 (fair subtyping).** We say that $T$ is a fair subtype of $S$ if $R$ fair compliant with $T$ implies $R$ fair compliant with $S$ for every $R$.

According to these definitions, when $T$ is a (fair) subtype of $S$, a process that behaves according to $T$ can be replaced by a process that behaves according to $S$ without compromising (fair) compliance with the clients of $T$. At first sight this substitution principle appears to be just the opposite of the expected/intended one, whereby it is safe to use a session channel of type $T$ where a session channel of type $S$ is expected if $T$ is a subtype of $S$. The mismatch is only apparent, however, and can be explained by looking carefully at the entities being replaced in the substitution principles recalled above (processes in one case, session channels in the other). The interested reader may refer to Gay [18] for a nice study of these two
125:10 Inference Systems with Corules for Fair Subtyping

Therefore, it is of paramount importance to provide equivalent characterizations of these relations, particularly in the form of inference systems. The GIS \( \langle F, F_\converge \rangle \) for (fair) subtyping is shown in Figure 3 and described hereafter. Rule \([\text{NIL}]\) states that \text{NIL} is the least element of the subtyping preorder, which is justified by the fact that no client successfully interacts with \text{NIL}. Rule \([\text{END}]\) establishes that \(?\text{END}\) and \!\text{END}\) are the least elements among all session types different from \text{NIL}. In our theory, this relation arises from the asymmetric form of compliance we have considered: a server \text{pEND} satisfies only \!\text{END}, which successfully interacts with any server different from \text{NIL}. Rules \([\text{IN}]\) and \([\text{OUT}]\) indicate that inputs are covariant and outputs are contravariant. That is, it is safe to replace a server with another one that receives a superset of messages (\(\text{dom}(f) \subseteq \text{dom}(g)\) in \([\text{IN}]\)) and, dually, it is safe to replace a server with another one that sends a subset of messages (\(\text{dom}(g) \subseteq \text{dom}(f)\) in \([\text{OUT}]\)). The side condition \(\text{dom}(g) \neq \emptyset\) in \([\text{OUT}]\) is important to preserve progress: if the server that behaves according to the larger session type is unable to send any message, the client may get stuck waiting for a message that is never sent. On the other hand, the side condition \(\text{dom}(f) \neq \emptyset\) is unnecessary from a purely technical view point, since the rule \([\text{IN}]\) without this side condition is subsumed by \([\text{END}]\). We have included the side condition to minimize the overlap between different rules and for symmetry with respect to \([\text{OUT}]\). Overall, these rules are aligned with those of the subtyping relation for session types given by Gay and Hole [19] (see also Remark 6.7).

To discuss the corule \([\text{CONVERGE}]\) that characterizes fair subtyping we need to introduce one last piece of notation concerning session types.

**Definition 6.3 (residual of a session type).** Given a session type \(T\) and a trace \(\varphi\) of \(T\) we write \(T(\varphi)\) for the residual of \(T\) after \(\varphi\), namely for the session type \(S\) such that \(T \Rightarrow S\).

The notion of residual is well defined since session types are deterministic: if \(T \Rightarrow S_1\) and \(T \Rightarrow S_2\), then \(S_1 = S_2\). It is implied that \(T(\varphi)\) is undefined if \(\varphi \not\in \text{tr}(T)\).

For the sake of presentation we describe the corule \([\text{CONVERGE}]\) incrementally, showing how it contributes to the soundness proof of the GIS in Figure 3. In doing so, it helps bearing in mind that the relation \(T \leq S\) is meant to preserve fair compliance (Definition 5.2), namely the possibility that any client of \(T\) can terminate successfully when interacting with \(S\). As a first approximation observe that, when the traces of \(T\) are included in the traces of \(S\), the corule \([\text{CONVERGE}]\) boils down to the following coaxiom:

\[
\begin{array}{c}
\text{NIL} \leq T \\
p\text{END} \leq T \\
\exists \varphi \in \text{tr}(T) \setminus \text{tr}(S) : \exists \psi, x \in \mathcal{V} : T(\psi!x) \leq S(\psi!x) \\
T \leq S
\end{array}
\]

\[
\begin{array}{c}
f(x) \leq g(x) \quad \forall x \in \text{dom}(f) \\
?f \leq ?g \\
\text{dom}(f) \neq \emptyset \\
\text{dom}(g) \subseteq \text{dom}(f) \\
T \leq S
\end{array}
\]

\[
\begin{array}{c}
f(x) \leq g(x) \quad \forall x \in \text{dom}(g) \\
!f \leq !g \\
\text{dom}(g) \subseteq \text{dom}(f) \\
T \leq S
\end{array}
\]

\[
\begin{array}{c}
\forall x \in \text{dom}(f) \\
\exists \psi, x \in \mathcal{V} : T(\psi ! x) \leq S(\psi ! x)
\end{array}
\]
\[ \text{tr}(T) \subseteq \text{tr}(S) \]
\[ T \leq S \]

Now consider a client \( R \) that is fair compliant with \( T \). It must be the case that \( R \parallel T \Rightarrow !\text{end} \parallel T' \) for some \( T' \neq \text{nil} \), namely that \( R \Rightarrow !\text{end} \) and \( T \Rightarrow T' \) for some sequence \( \varphi \) of actions. The side condition \( \text{tr}(T) \subseteq \text{tr}(S) \) ensures that \( \varphi \) is also a trace of \( S \), therefore \( R \parallel S \Rightarrow !\text{end} \parallel S' \) for the \( S' \neq \text{nil} \) such that \( S \Rightarrow \varphi \Rightarrow S' \). In general, we know from rule [out] that \( S \) may perform fewer outputs than \( T \), hence not every trace of \( T \) is necessarily a trace of \( S \). Writing \( \leq \) for the prefix order relation on traces, the premises

\[ \forall \varphi \in \text{tr}(T) \setminus \text{tr}(S) : \exists \psi \leq \varphi, x \in V : T(\psi!x) \leq S(\psi!x) \]

of [converge] make sure that, for every trace \( \varphi \) of \( T \) that is not a trace of \( S \), there exists a common prefix \( \psi \) of \( T \) and \( S \) and an output action \( !x \) shared by both \( T(\psi) \) and \( S(\psi) \) such that the residuals of \( T \) and \( S \) after \( \psi!x \) are one level closer, in the proof tree for \( T \leq S \), to the residuals of \( T \) and \( S \) for which trace inclusion holds. The fact that \( \psi \) must be followed by an output \( !x \) is fundamental, since the client \( R \) must be able to accept all the outputs of \( T \).

Note that the corule is unsound in general. For instance, \( !0.\text{end} \leq !N.\text{end} \) is derivable by [converge] since \( \text{tr}(!0.\text{end}) \subseteq \text{tr}(!N.\text{end}) \), but \( !0.\text{end} \) is not a subtype of \( !N.\text{end} \).

**Example 6.4.** Consider once again the session types \( T_i \) and \( S_i \) of Example 3.3. It is easy to see that \( T_i \leq S_i \in \text{Colnd}[F] \) for \( i = 1, 2 \). In order to derive \( T_i \leq S_i \) in the GIS \( \langle F, F_{\text{co}} \rangle \) we must find a well-founded proof tree of \( T \leq S \) in \( F \cup F_{\text{co}} \) and the only hope to do so is by means of [converge], since \( T_i \) and \( S_i \) share traces of arbitrary length. Observe that every trace \( \varphi \) of \( T_1 \) that is not a trace of \( S_1 \) has the form \( (\text{true}!p_k)^k \text{true}0\ldots \) where \( p_k \in \mathbb{N}^* \). Thus, it suffices to take \( \psi = \varepsilon \) and \( x = 0 \), noted that \( T_1(\varepsilon) = S_1(\varepsilon) = ?\text{end} \), to derive

\[ T_1 \leq S_1 \]

with two applications of [converge]. On the other hand, every trace \( \varphi \in \text{tr}(T_2) \setminus \text{tr}(S_2) \) has the form \( (?\text{true}!p_k)^k \text{true}0\ldots \) where \( p_k \in \mathbb{N}^* \). All the prefixes of such traces that are followed by an output and are shared by both \( T_2 \) and \( S_2 \) have the form \( (?\text{true}!p_k)^k \text{true} \) where \( p_k \in \mathbb{N}^* \), and \( T_2(\psi!p) = T_2 \) and \( S_2(\psi!p) = S_2 \) for all such prefixes and \( p \in \mathbb{N}^* \). It follows that we are unable to derive \( T_2 \leq S_2 \) with a well-founded proof tree in \( F \cup F_{\text{co}} \). This is consistent with the fact that, in Example 5.3, we have found a client \( R_2 \) that is fair compliant with \( T_2 \) but not with \( S_2 \). Intuitively, \( R_2 \) insists on poking the server waiting to receive 0. This can eventually happen with \( T_2 \), but not with \( S_2 \). In the case of \( T_1 \) and \( S_1 \) no such client can exist, since the server may decide to interrupt the interaction at any time by sending a false message to the client.

Example 6.4 also shows that fair subtyping is a context-sensitive relation in that the applicability of a rule for deriving \( T \leq S \) may depend on the context in which \( T \) and \( S \) occur. For instance, in the non-well-founded derivation

\[ \begin{array}{c}
T_1 \leq S_1 \\
\neg\neg T_1 \leq \neg\neg T_1 \setminus \text{OUT} \quad \text{[OUT]} \\
\text{OUT} \quad ?\text{end} \leq ?\text{end} \quad \text{[END]} \\
\neg\neg T_1 \leq S_1
\end{array} \]
the rule \([\text{out}]\) is used infinitely many times to relate the output session types \(\text{!}N.T_1\) and \(\text{!}N^+.S_2\). In this context, rule \([\text{out}]\) can be applied harmlessly. On the contrary, if we attempt to find a derivation for \(T_2 \leq S_2\) we obtain the non-well-founded tree

\[
\begin{array}{c}
T_2 \leq S_2 \\
\text{[out]} \\
\text{[end]} \\
\text{[in]}
\end{array}
\]

(2)

which is isomorphic to the one shown in Equation (1) with the difference that some applications of \([\text{out}]\) have been replaced by applications of \([\text{in}]\). Here too \([\text{out}]\) is used infinitely many times, but this time to relate the output session types \(\text{!}N.T_2\) and \(\text{!}N^+.S_2\). This derivation allows us to prove \(T_2 \leq S_2 \in \text{Colnd}[F,\mathcal{F}]\), but not \(T_2 \leq S_2 \in \text{Gen}[F,\mathcal{F}_\text{co}]\), because \([\text{out}]\) removes the \(0\) output from \(S_2\) that a client of \(T_2\) may depend upon.

\[\text{Remark 6.5.}\] As observed by one reviewer, rule \([\text{converge}]\) is hard not only to understand, but also to formalize in Agda. We have been unable to conceive a sound and complete GIS for fair subtyping that is based on simpler corules, but it should be noted that the property enforced by \([\text{converge}]\) is fundamentally non-local and therefore difficult to express in terms of immediate subtrees of a session type. To illustrate the point, consider the following alternative set of corules meant to replace \([\text{converge}]\) in Figure 3:

\[\begin{array}{c}
\text{[co-inc]} \\
T \leq S \\
\implies \text{tr}(T) \subseteq \text{tr}(S) \\
\text{[co-in]} \\
f(x) \leq g(x) \quad (\forall x \in \text{dom}(f)) \\
\implies ?f \leq ?g \\
\text{[co-out]} \\
f(x) \leq g(x) \\
\implies !f \not\leq !g \quad x \in \text{dom}(f) \cap \text{dom}(g)
\end{array}\]

It is easy to see that these rules provide a sound approximation of \([\text{converge}]\), but they are not complete. Indeed, consider the session types \(T = ?\text{true}.T + ?\text{false}.(!\text{true}.?\text{end} \oplus !\text{false}.?\text{end})\) and \(S = ?\text{true}.S + ?\text{false}!.?\text{true}.?\text{end}\). We have \(T \leq S\) and yet \(T \not\leq \text{Ind} S\) cannot be proved with the above corules: it is not possible to prove \(T \leq \text{Ind} S\) using \([\text{co-inc}]\) because \(\text{tr}(T) \not\subseteq \text{tr}(S)\). If, on the other hand, we insist on visiting both branches of the topmost input as required by \([\text{co-in}]\), we end up requiring a proof of \(T \leq \text{Ind} S\) in order to derive \(T \leq \text{Ind} S\).

\[\text{Theorem 6.6.}\] For every \(T, S \in S\) the following properties hold:
1. \(T\) is a subtype of \(S\) if and only if \(T \leq S \in \text{Colnd}[F]\);
2. \(T\) is a fair subtype of \(S\) if and only if \(T \leq S \in \text{Gen}[F,\mathcal{F}_\text{co}]\).

\[\text{Remark 6.7.}\] Most session type theories adopt a symmetric form of session type compatibility whereby client and server are required to terminate the interaction at the same time. It is easy to define a notion of symmetric compliance (also known as peer compliance [7]) by turning \(T' \not= \text{nil}\) into \(T' = ?\text{end}\) in Definition 5.1. The subtyping relation induced by symmetric compliance has essentially the same characterization of Definition 6.1, except that the axiom \([\text{end}]\) is replaced by the more familiar \(p \text{end} \leq q \text{end}\) [19]. On the other hand, the analogous change in Definition 5.2 has much deeper consequences: the requirement that client and server must end the interaction at the same time creates a large family of session types that are syntactically very different, but semantically equivalent. For example, the session types \(T\) and \(S\) such that \(T = ?\text{!}N.T\) and \(S = \text{!}B.S\), which describe completely unrelated protocols, would be equivalent for the simple reason that no client successfully interacts with them (they are not weakly terminating, since they do not contain any occurrence of \(\text{end}\)). We have not investigated the existence of a GIS for fair subtyping induced by symmetric fair compliance. A partial characterization (which however requires various auxiliary relations) is given by Padovani [30].
7 Concluding Remarks

We have shown that generalized inference systems are an effective framework for defining sound and complete proof systems of (some) combined safety and liveness properties of (dependent) session types (Definitions 4.2 and 5.2), as well as of a liveness-preserving subtyping relation (Definition 6.2). We think that this achievement is more than a coincidence. One of the fundamental results in model checking states that every property can be expressed as the conjunction of a safety property and a liveness property [2, 3, 6]. The connections between safety and liveness on one side and coinduction and induction on the other make GISs appropriate for characterizing combined safety and liveness properties.

Murgia [25] studies a wide range of compliance relations for processes and session types, showing that many of them are fixed points of a functional operator, but not necessarily the least or the greatest ones. In particular, he shows that progress compliance, which is akin to our compliance (Definition 5.1), is a greatest fixed point and that should-testing compliance, which is akin to our fair compliance (Definition 5.2), is an intermediate fixed point. These results are consistent with Theorem 5.4. We have extended these results to subtyping (Definition 6.1) and fair subtyping (Definition 6.2). Previous alternative characterizations of fair subtyping and the related should-testing preorder either require several different relations [29, 30] or are denotational in nature [31] and therefore not as insightful as desirable. Using GISs, we have obtained complete characterizations of fair compliance and fair subtyping by simply adding a few corules to the proof systems of their “unfair” counterparts.

We have coded all the notions and results discussed in the paper in Agda [27], thus providing the first machine-checked formalization of liveness properties and liveness-preserving subtyping relations for dependent session types. Theorem 4.4 and item (2) of Theorems 5.4 and 6.6 are proved considering $V = B$ instead of an arbitrary set of values. This is because the version of the Agda library for GISs [11, 13] used for the formalization does not support (co)rules with infinitely many premises, which are necessary if $V$ is infinite. However, all of the key aspects of the characterizations of weak termination, fair compliance and fair subtyping already emerge in this simplified setting. The Agda formalization is not entirely constructive since it makes use of three postulates: the law of excluded middle, the extensionality axiom and the duality between a universally quantified, inductive characterization of convergence (see [converge] in Figure 3) and its negation, which is characterized using an existentially quantified, coinductive definition. Note that the Agda library for GISs is a standalone development [11, 13, 12], on top of which we have built our own [15]. This makes it easy to extend our results to other families of processes or to different properties.

In this paper we have focused on properties of session types alone. The most important piece of future work that we plan to carry out next is the development of a session type system making use of fair subtyping for the enforcement of liveness properties of processes. This problem has remained open for a long time [29, 30] because the integration of fair subtyping into a coinductively-interpreted session type system is (unsurprisingly) challenging. By contrast, session type systems making use of safety-preserving subtyping relations are quite widespread [19, 9, 23, 10]. The achievements described in this paper suggest that GISs could provide just the right framework for defining such type system. Somewhat connected with this future development is also the handling of delegation and therefore of higher-order session types. Previous developments [30] have shown that delegation is orthogonal to the characterizing features of fair subtyping, since the communication of session channels does not (usually) affect the branching structure of session types (but there are a few exceptions [9, 10]). For this reason, we think that this extension can be accounted for without substantial issues.
References


