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 $\mathit{maxElem}(l, x)$, asserting that x is the maximum element of a possibly infinite list l. If we consider the inference system

$$\frac{}{\mathit{maxElem}(x :: \Lambda, x)} \qquad \frac{\mathit{maxElem}(l, y)}{\mathit{maxElem}(x :: l, \max\{x, y\})}$$

where Λ 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 *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 *maxElem* predicate is complete but not sound. In particular, it becomes possible to derive any judgment $\mathit{maxElem}(l, x)$ 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*:

$$\frac{}{\mathit{maxElem}(x :: l, x)}$$

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 $\mathit{maxElem}(l, x)$ 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 $\mathit{maxElem}(l, x)$, 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* [1] $\mathcal{I}$ over a *universe* $\mathcal{U}$ of *judgments* is a set of *rules*, which are pairs $\langle pr, j\rangle$ where $pr \subseteq \mathcal{U}$ is the set of *premises* of the rule and $j \in \mathcal{U}$ is the *conclusion* of the rule. A rule without premises is called *axiom*. Rules are typically presented using the syntax

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

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

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

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

▶ **Definition 2.2.** *The* inductive interpretation $\mathsf{Ind}[\![\mathcal{I}]\!]$ *of an inference system* $\mathcal{I}$ *is the least fixed point of* $F_{\mathcal{I}}$ *and the* coinductive interpretation $\mathsf{CoInd}[\![\mathcal{I}]\!]$ *is the greatest one.*

From a proof theoretical point of view, $\mathsf{Ind}[\![\mathcal{I}]\!]$ and $\mathsf{CoInd}[\![\mathcal{I}]\!]$ 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* $\langle \mathcal{I}, \mathcal{I}_{\mathsf{co}}\rangle$ *where* $\mathcal{I}$ *and* $\mathcal{I}_{\mathsf{co}}$ *are inference systems whose elements are called* rules *and* corules*, respectively. The interpretation of a generalized inference system* $\langle \mathcal{I}, \mathcal{I}_{\mathsf{co}}\rangle$*, denoted by* $\mathsf{Gen}[\![\mathcal{I}, \mathcal{I}_{\mathsf{co}}]\!]$*, is the greatest post-fixed point of* $F_{\mathcal{I}}$ *that is included in* $\mathsf{Ind}[\![\mathcal{I} \cup \mathcal{I}_{\mathsf{co}}]\!]$*.*

From a proof theoretical point of view, a GIS $\langle \mathcal{I}, \mathcal{I}_{\mathsf{co}} \rangle$ identifies the set of judgments that are derivable with an arbitrary (not necessarily well-founded) proof tree in $\mathcal{I}$ and whose nodes (the judgments occurring in the proof tree) are all derivable with a well-founded proof tree in $\mathcal{I} \cup \mathcal{I}_{\mathsf{co}}$. Recalling *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* $\mathcal{S} \subseteq \mathcal{U}$ that contains the *valid judgments*. We can relate $\mathcal{S}$ 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 $\mathcal{S}$ is *closed* with respect to $\mathcal{I}$. That is, whenever the premises of a rule of $\mathcal{I}$ are all in $\mathcal{S}$, then the conclusion of the rule is also in $\mathcal{S}$.

▶ **Proposition 2.4.** *If* $F_{\mathcal{I}}(\mathcal{S}) \subseteq \mathcal{S}$*, then* $\mathsf{Ind}[\![\mathcal{I}]\!] \subseteq \mathcal{S}$*.*

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

▶ **Proposition 2.5.** *If* $\mathcal{S} \subseteq F_{\mathcal{I}}(\mathcal{S})$*, then* $\mathcal{S} \subseteq \mathsf{Coind}[\![\mathcal{I}]\!]$*.*

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

▶ **Proposition 2.6.** *If* $\mathcal{S} \subseteq \mathsf{Ind}[\![\mathcal{I} \cup \mathcal{I}_{\mathsf{co}}]\!]$ *and* $\mathcal{S} \subseteq F_{\mathcal{I}}(\mathcal{S})$*, then* $\mathcal{S} \subseteq \mathsf{Gen}[\![\mathcal{I}, \mathcal{I}_{\mathsf{co}}]\!]$*.*

Proving the boundedness of $\mathcal{S}$ amounts to proving the completeness of $\mathcal{I} \cup \mathcal{I}_{\mathsf{co}}$ (inductively interpreted) with respect to $\mathcal{S}$. 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 $\mathbb{V}$ of *values* that can be exchanged in communications. This set may include booleans, natural numbers, strings, and so forth. Hereafter, we assume that $\mathbb{V}$ 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 $\mathbb{V}$.

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

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

A session type can be of three forms. An *input session type* $?f$ describes a channel used first for receiving a message $x \in \mathbb{V}$ and then according to $f(x)$. An *output session type* $!f$ describes a channel used first for sending a message $x \in \mathbb{V}$ and then according to $f(x)$. Finally, the session type nil describes an *unusable* session channel. As we will see shortly, we use 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.

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

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

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

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

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 $!\mathsf{true}.T$ instead of $!\{\mathsf{true}\}.T$. We let $?\mathsf{end} \stackrel{\text{def}}{=} ?\emptyset.\mathsf{nil}$ and $!\mathsf{end} \stackrel{\text{def}}{=} !\emptyset.\mathsf{nil}$. Both $?\mathsf{end}$ and $!\mathsf{end}$ 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 $\mathsf{nil} \sqcup T = T \sqcup \mathsf{nil} = T$ 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

$$?f + ?g \stackrel{\text{def}}{=} ?(f \sqcup g) \qquad !f \oplus !g \stackrel{\text{def}}{=} !(f \sqcup g)$$

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

▶ **Example 3.3.** The session types T_1 and S_1 that satisfy the equations

$$T_1 = !\mathsf{true}.!\mathbb{N}.T_1 \oplus !\mathsf{false}.?\mathsf{end} \qquad S_1 = !\mathsf{true}.!\mathbb{N}^+.S_1 \oplus !\mathsf{false}.?\mathsf{end}$$

both describe a channel used for sending a boolean. If the boolean is false, the communication stops immediately ($?\mathsf{end}$). If it is 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

$$T_2 = ?\mathsf{true}.!\mathbb{N}.T_2 + ?\mathsf{false}.?\mathsf{end} \qquad S_2 = ?\mathsf{true}.!\mathbb{N}^+.S_2 + ?\mathsf{false}.?\mathsf{end}$$

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

$$f(0) = T_3 \qquad f(n+1) = !\mathbb{B}.f(n)$$

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 α, β, γ, 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:

[T-INPUT] [T-OUTPUT]

$$\frac{}{?f \xrightarrow{?x} f(x)} \qquad \frac{}{!f \xrightarrow{!x} f(x)}\ x \in \mathsf{dom}(f)$$

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 $x \in \mathsf{dom}(f)$ in [T-OUTPUT] and the lack thereof in [T-INPUT]: a process that uses a session channel according to $!f$ refrains from sending a message x if $x \notin \mathsf{dom}(f)$, 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 !end and ?end. While both describe a session endpoint on which no further communications may occur, !end is "more robust" than ?end since it has no transitions, whereas ?end is "more fragile" than !end since it performs transitions, all of which lead to nil. For this reason, we use !end to flag successful session termination (Section 5), whereas ?end only means that the protocol has ended.

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

4 Weak Termination

A session type is weakly terminating if it preserves the possibility of reaching !end or ?end along all of its transitions that do not lead to 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.

▶ **Definition 4.1** (traces and maximal traces). *The* traces *of* T *are defined as* $\mathtt{tr}(T) \stackrel{\text{def}}{=} \{\varphi \mid \exists S : T \stackrel{\varphi}{\Longrightarrow} S \neq \mathsf{nil}\}$. *We say that* $\varphi \in \mathtt{tr}(T)$ *is* maximal *if* $\varphi\psi \in \mathtt{tr}(T)$ *implies* $\psi = \varepsilon$.

For example, we have $\mathtt{tr}(\mathsf{nil}) = \emptyset$ and $\mathtt{tr}(!\mathsf{end}) = \mathtt{tr}(?\mathsf{end}) = \{\varepsilon\}$. Note that !end and ?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 ε is a maximal trace of both !end and ?end but not of $!\mathbb{B}.?\mathsf{end}$ whereas !true and !false are maximal traces of $!\mathbb{B}.?\mathsf{end}$.

$$[\text{NIL}]\ \frac{}{\mathsf{nil}\Downarrow} \qquad [\text{IN}]\ \frac{f(x)\Downarrow \ (\forall x \in \mathbb{V})}{?f\Downarrow} \qquad [\text{OUT}]\ \frac{f(x)\Downarrow \ (\forall x \in \mathbb{V})}{!f\Downarrow}$$

$$[\text{CO-IN}]\ \frac{f(x)\Downarrow}{?f\Downarrow}\ x \in \mathsf{dom}(f) \qquad [\text{CO-OUT}]\ \frac{f(x)\Downarrow}{!f\Downarrow}\ x \in \mathsf{dom}(f)$$

Figure 1 Generalized inference system $\langle \mathcal{T}, \mathcal{T}_{\mathsf{co}} \rangle$ for weak termination.

▶ **Definition 4.2** (weak termination). *We say that T is* weakly terminating *if, for every $\varphi \in \mathtt{tr}(T)$, there exists ψ such that $\varphi\psi \in \mathtt{tr}(T)$ and $\varphi\psi$ is maximal.*

▶ **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 $S_3 \stackrel{\text{def}}{=} !\mathsf{true}.T_3 \oplus !\mathsf{false}.?\mathsf{end}$ is not weakly terminating, even though there is a path leading to $?\mathsf{end}$, because weak termination must be *preserved* along all possible transitions of the session type, whereas $S_3 \xrightarrow{!\mathsf{true}} T_3$ and T_3 is not weakly terminating. Finally, nil is trivially weakly terminating since it has no trace. ⌟

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

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

▶ **Theorem 4.4.** *T is weakly terminating if and only if $T\Downarrow \in \mathsf{Gen}[\![\mathcal{T}, \mathcal{T}_{\mathsf{co}}]\!]$.*

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 $T\Downarrow \in \mathsf{Gen}[\![\mathcal{T}, \mathcal{T}_{\mathsf{co}}]\!]$ and consider a trace $\varphi \in \mathtt{tr}(T)$. That is, $T \stackrel{\varphi}{\Longrightarrow} S$ for some $S \neq \mathsf{nil}$. Using [IN] and [OUT] we deduce $S\Downarrow \in \mathsf{Gen}[\![\mathcal{T}, \mathcal{T}_{\mathsf{co}}]\!]$ by means of a simple induction on φ. Now $S\Downarrow \in \mathsf{Gen}[\![\mathcal{T}, \mathcal{T}_{\mathsf{co}}]\!]$ implies $S\Downarrow \in \mathsf{Ind}[\![\mathcal{T} \cup \mathcal{T}_{\mathsf{co}}]\!]$ by Definition 2.3. Another induction on the (well-founded) derivation of this judgment, along with the witness messages of [CO-IN] and [CO-OUT], allows us to find ψ 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

$$\frac{}{R \parallel T \to R' \parallel S'} \text{ if } R \xrightarrow{\overline{\alpha}} R' \text{ and } T \xrightarrow{\alpha} T'$$

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

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 nil only if an unexpected message is received). Formally:

▶ **Definition 5.1** (compliance). *We say that R is* compliant *with T if $R \parallel T \Rightarrow R' \parallel T' \not\to$ implies $R' = !\mathsf{end}$ and $T' \neq \mathsf{nil}$.*

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 $R \parallel T \Rightarrow R' \parallel T'$ implies $R' \parallel T' \Rightarrow !\mathsf{end} \parallel T''$ with $T'' \neq \mathsf{nil}$.*

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

$$R_1 = ?\mathsf{true}.?\mathbb{N}.R_1 + ?\mathsf{false}.!\mathsf{end} \qquad R_2 = !\mathsf{true}.(?0.!\mathsf{end} + ?\mathbb{N}^+.R_2)$$

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 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 true to the server in hope to receive 0, but 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 !end on the client side are not necessarily the same (complemented) traces that lead to ?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 $\langle \mathcal{C}, \mathcal{C}_{\mathsf{co}} \rangle$ for fair compliance. Rule [WIN] relates a satisfied client with a non-failed server. Rules [IN-OUT] and [OUT-IN] require that, no matter which message is exchanged between client and server, the respective continuations are still fair compliant. The side conditions $\mathsf{dom}(f) \neq \emptyset$ and $\mathsf{dom}(g) \neq \emptyset$ 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 [CO-IN-OUT] and [CO-OUT-IN] are essential.

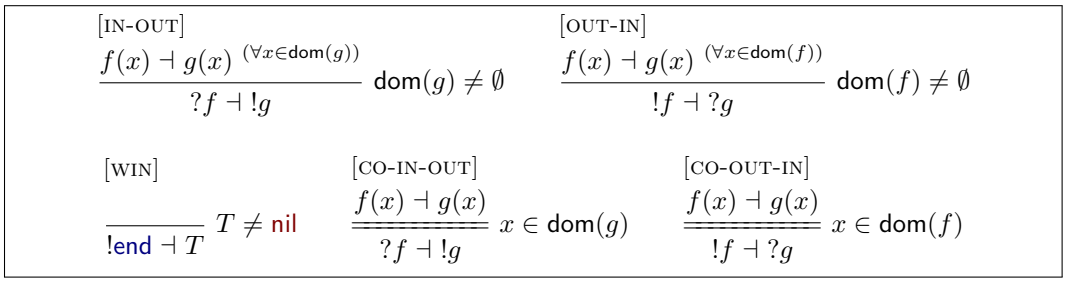

Figure 2 Generalized inference system $\langle \mathcal{C}, \mathcal{C}_{\mathsf{co}} \rangle$ for fair compliance.

▶ **Theorem 5.4** (compliance). *For every $R, T \in \mathbb{S}$, the following properties hold:*

1. *R is compliant with T if and only if $R \dashv T \in \mathsf{CoInd}[\![\mathcal{C}]\!]$;*
2. *R is fair compliant with T if and only if $R \dashv T \in \mathsf{Gen}[\![\mathcal{C}, \mathcal{C}_{\mathsf{co}}]\!]$.*

To illustrate the role of the corules, let us sketch the proof that the GIS in Figure 2 is sound with respect to fair compliance. Suppose that $R \dashv T \in \mathsf{Gen}[\![\mathcal{C}, \mathcal{C}_{\mathsf{co}}]\!]$ and consider a reduction $R \parallel T \Rightarrow R' \parallel T'$. An induction on the length of this reduction, along with [IN-OUT] and [OUT-IN], allows us to deduce $R' \dashv T' \in \mathsf{Gen}[\![\mathcal{C}, \mathcal{C}_{\mathsf{co}}]\!]$. Then we have $R' \dashv T' \in \mathsf{Ind}[\![\mathcal{C} \cup \mathcal{C}_{\mathsf{co}}]\!]$ by Definition 2.3. An induction on this (well-founded) derivation allows us to find a reduction $R' \parallel T' \Rightarrow !\mathsf{end} \parallel T''$ such that $T'' \neq \mathsf{nil}$.

Observe that the corules are at once essential and unsound. For example, without them we would be able to derive the judgment $R_2 \dashv S_2$ 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 $!\mathbb{N}.!\mathsf{end} \dashv ?0.?\mathsf{end}$ despite the reduction $!\mathbb{N}.!\mathsf{end} \parallel ?0.?\mathsf{end} \to !\mathsf{end} \parallel \mathsf{nil}$ since *there exists* an interaction that leads to the successful configuration $!\mathsf{end} \parallel ?\mathsf{end}$ (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

$$[\textsc{nil}]\ \frac{}{\mathsf{nil} \leqslant T} \qquad [\textsc{end}]\ \frac{}{p\,\mathsf{end} \leqslant T}\ T \neq \mathsf{nil}$$

$$[\textsc{converge}]\ \frac{\forall \varphi \in \mathtt{tr}(T) \setminus \mathtt{tr}(S) : \exists \psi \leq \varphi, x \in \mathbb{V} : T(\psi!x) \leqslant S(\psi!x)}{T \leqslant S}$$

$$[\textsc{in}]\ \frac{f(x) \leqslant g(x)\ ^{(\forall x \in \mathsf{dom}(f))}}{?f \leqslant ?g}\ \begin{array}{l}\mathsf{dom}(f) \neq \emptyset \\ \mathsf{dom}(f) \subseteq \mathsf{dom}(g)\end{array} \qquad [\textsc{out}]\ \frac{f(x) \leqslant g(x)\ ^{(\forall x \in \mathsf{dom}(g))}}{!f \leqslant !g}\ \begin{array}{l}\mathsf{dom}(g) \neq \emptyset \\ \mathsf{dom}(g) \subseteq \mathsf{dom}(f)\end{array}$$

Figure 3 Generalized inference system $\langle \mathcal{F}, \mathcal{F}_{\mathsf{co}} \rangle$ for fair subtyping.

different, yet related viewpoints. What matters here is that the above notions of subtyping are "correct by definition" but do not provide any hint as to the shape of two session types T and S that are related by (fair) subtyping. This problem is well known in the semantic approaches for defining subtyping relations [17, 7] as well as in the aforementioned testing theories for processes [26, 20, 31], which the two definitions above are directly inspired from. Therefore, it is of paramount importance to provide equivalent characterizations of these relations, particularly in the form of inference systems.

The GIS $\langle \mathcal{F}, \mathcal{F}_{\mathsf{co}} \rangle$ for (fair) subtyping is shown in Figure 3 and described hereafter. Rule [NIL] states that nil is the least element of the subtyping preorder, which is justified by the fact that no client successfully interacts with nil. Rule [END] establishes that ?end and !end are the least elements among all session types different from nil. In our theory, this relation arises from the asymmetric form of compliance we have considered: a server $p\,\mathsf{end}$ satisfies only !end, which successfully interacts with any server different from nil. Rules [IN] and [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 ($\mathsf{dom}(f) \subseteq \mathsf{dom}(g)$ in [IN]) and, dually, it is safe to replace a server with another one that sends a subset of messages ($\mathsf{dom}(g) \subseteq \mathsf{dom}(f)$ in [OUT]). The side condition $\mathsf{dom}(g) \neq \emptyset$ in [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 $\mathsf{dom}(f) \neq \emptyset$ is unnecessary from a purely technical view point, since the rule [IN] without this side condition is subsumed by [END]. We have included the side condition to minimize the overlap between different rules and for symmetry with respect to [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 [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 φ of T we write $T(\varphi)$ for the* residual *of T after φ, namely for the session type S such that $T \stackrel{\varphi}{\Longrightarrow} S$.*

The notion of residual is well defined since session types are *deterministic*: if $T \stackrel{\varphi}{\Longrightarrow} S_1$ and $T \stackrel{\varphi}{\Longrightarrow} S_2$, then $S_1 = S_2$. It is implied that $T(\varphi)$ is undefined if $\varphi \notin \mathtt{tr}(T)$.

For the sake of presentation we describe the corule [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 $T \leqslant S$ 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 [CONVERGE] boils down to the following coaxiom:

$$\frac{}{T \leqslant S}\ \mathtt{tr}(T) \subseteq \mathtt{tr}(S)$$

Now consider a client R that is fair compliant with T. It must be the case that $R \parallel T \Rightarrow !\mathsf{end} \parallel T'$ for some $T' \neq \mathsf{nil}$, namely that $R \stackrel{\overline{\varphi}}{\Longrightarrow} !\mathsf{end}$ and $T \stackrel{\varphi}{\Longrightarrow} T'$ for some sequence φ of actions. The side condition $\mathtt{tr}(T) \subseteq \mathtt{tr}(S)$ ensures that φ is also a trace of S, therefore $R \parallel S \Rightarrow !\mathsf{end} \parallel S'$ for the $S' \neq \mathsf{nil}$ such that $S \stackrel{\varphi}{\Longrightarrow} S'$. 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

$$\forall \varphi \in \mathtt{tr}(T) \setminus \mathtt{tr}(S) : \exists \psi \leq \varphi, x \in \mathbb{V} : T(\psi!x) \leqslant S(\psi!x)$$

of [CONVERGE] make sure that, for every trace φ of T that is not a trace of S, there exists a common prefix ψ 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 $T \leqslant S$, to the residuals of T and S for which trace inclusion holds. The fact that ψ 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, $!0.?\mathsf{end} \leqslant !\mathbb{N}.?\mathsf{end}$ is derivable by [CONVERGE] since $\mathtt{tr}(!0.?\mathsf{end}) \subseteq \mathtt{tr}(!\mathbb{N}.?\mathsf{end})$, but $!0.?\mathsf{end}$ is *not* a subtype of $!\mathbb{N}.?\mathsf{end}$.

▶ **Example 6.4.** Consider once again the session types T_i and S_i of Example 3.3. It is easy to see that $T_i \leqslant S_i \in \mathsf{CoInd}[\![\mathcal{F}]\!]$ for $i = 1, 2$. In order to derive $T_i \leqslant S_i$ in the GIS $\langle \mathcal{F}, \mathcal{F}_{\mathsf{co}} \rangle$ we must find a well-founded proof tree of $T \leqslant S$ in $\mathcal{F} \cup \mathcal{F}_{\mathsf{co}}$ 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 φ of T_1 that is not a trace of S_1 has the form $(!\mathsf{true}!p_k)^k !\mathsf{true}!0 \dots$ where $p_k \in \mathbb{N}^+$. Thus, it suffices to take $\psi = \varepsilon$ and $x = 0$, noted that $T_1(!0) = S_1(!0) = ?\mathsf{end}$, to derive

$$\dfrac{\dfrac{}{?\mathsf{end} \leqslant ?\mathsf{end}}}{T_1 \leqslant S_1}$$

with two applications of [CONVERGE]. On the other hand, every trace $\varphi \in \mathtt{tr}(T_2) \setminus \mathtt{tr}(S_2)$ has the form $(?\mathsf{true}!p_k)^k ?\mathsf{true}!0 \dots$ 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 $(?\mathsf{true}!p_k)^k ?\mathsf{true}$ 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 $T_2 \leqslant S_2$ with a well-founded proof tree in $\mathcal{F} \cup \mathcal{F}_{\mathsf{co}}$. 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 $T \leqslant S$ may depend on the context in which T and S occur. For instance, in the non-well-founded derivation

$$\dfrac{\dfrac{\dfrac{\vdots}{T_1 \leqslant S_1}}{!\mathbb{N}.T_1 \leqslant !\mathbb{N}^+.S_1}\,[\text{OUT}] \qquad \dfrac{}{?\mathsf{end} \leqslant ?\mathsf{end}}\,[\text{END}]}{T_1 \leqslant S_1}\,[\text{OUT}] \tag{1}$$

the rule [OUT] is used infinitely many times to relate the output session types $!\mathbb{N}.T_1$ and $!\mathbb{N}^+.S_2$. In this context, rule [OUT] can be applied harmlessly. On the contrary, if we attempt to find a derivation for $T_2 \leqslant S_2$ we obtain the non-well-founded tree

$$\dfrac{\dfrac{\dfrac{\vdots}{T_2 \leqslant S_2}}{!\mathbb{N}.T_2 \leqslant !\mathbb{N}^+.S_2}\,[\text{OUT}] \qquad \dfrac{}{?\mathsf{end} \leqslant ?\mathsf{end}}\,[\text{END}]}{T_2 \leqslant S_2}\,[\text{IN}] \tag{2}$$

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

▶ Remark 6.5. As observed by one reviewer, rule [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 [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 [CONVERGE] in Figure 3:

$$[\text{CO-INC}]\ \frac{}{T \leqslant S}\ \mathtt{tr}(T) \subseteq \mathtt{tr}(S) \qquad [\text{CO-IN}]\ \frac{f(x) \leqslant g(x)\ ^{(\forall x \in \mathsf{dom}(f))}}{?f \leqslant ?g} \qquad [\text{CO-OUT}]\ \frac{f(x) \leqslant g(x)}{!f \leqslant !g}\ x \in \mathsf{dom}(f) \cap \mathsf{dom}(g)$$

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

▶ **Theorem 6.6.** *For every $T, S \in \mathbb{S}$ the following properties hold:*

1. *T is a subtype of S if and only if $T \leqslant S \in \mathsf{CoInd}[\![\mathcal{F}]\!]$;*
2. *T is a fair subtype of S if and only if $T \leqslant S \in \mathsf{Gen}[\![\mathcal{F}, \mathcal{F}_{\mathsf{co}}]\!]$.*

▶ 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 $T' \neq \mathsf{nil}$ into $T' = ?\mathsf{end}$ in Definition 5.1. The subtyping relation induced by symmetric compliance has essentially the same characterization of Definition 6.1, except that the axiom [END] is replaced by the more familiar $p\,\mathsf{end} \leqslant q\,\mathsf{end}$ [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 $T = ?\mathbb{N}.T$ and $S = !\mathbb{B}.S$, 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 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 $\mathbb{V} = \mathbb{B}$ 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 $\mathbb{V}$ 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

1 Peter Aczel. An introduction to inductive definitions. In Jon Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, pages 739–782. Elsevier, 1977. doi:10.1016/S0049-237X(08)71120-0.

2 Bowen Alpern and Fred B. Schneider. Defining liveness. *Inf. Process. Lett.*, 21(4):181–185, 1985. doi:10.1016/0020-0190(85)90056-0.

3 Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Comput.*, 2(3):117–126, 1987. doi:10.1007/BF01782772.

4 Davide Ancona, Viviana Bono, Mario Bravetti, Joana Campos, Giuseppe Castagna, Pierre-Malo Deniélou, Simon J. Gay, Nils Gesbert, Elena Giachino, Raymond Hu, Einar Broch Johnsen, Francisco Martins, Viviana Mascardi, Fabrizio Montesi, Rumyana Neykova, Nicholas Ng, Luca Padovani, Vasco T. Vasconcelos, and Nobuko Yoshida. Behavioral types in programming languages. *Found. Trends Program. Lang.*, 3(2-3):95–230, 2016. doi:10.1561/2500000031.

5 Davide Ancona, Francesco Dagnino, and Elena Zucca. Generalizing inference systems by coaxioms. In Hongseok Yang, editor, *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10201 of *Lecture Notes in Computer Science*, pages 29–55. Springer, 2017. doi:10.1007/978-3-662-54434-1_2.

6 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

7 Giovanni Bernardi and Matthew Hennessy. Using higher-order contracts to model session types. *Log. Methods Comput. Sci.*, 12(2), 2016. doi:10.2168/LMCS-12(2:10)2016.

8 Julian C. Bradfield and Colin Stirling. Modal mu-calculi. In Patrick Blackburn, J. F. A. K. van Benthem, and Frank Wolter, editors, *Handbook of Modal Logic*, volume 3 of *Studies in logic and practical reasoning*, pages 721–756. North-Holland, 2007. doi:10.1016/s1570-2464(07)80015-2.

9 Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, Elena Giachino, and Luca Padovani. Foundations of session types. In António Porto and Francisco Javier López-Fraguas, editors, *Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, September 7-9, 2009, Coimbra, Portugal*, pages 219–230. ACM, 2009. doi:10.1145/1599410.1599437.

10 Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, Elena Giachino, and Luca Padovani. Foundations of session types: 10 years later. In Ekaterina Komendantskaya, editor, *Proceedings of the 21st International Symposium on Principles and Practice of Programming Languages, PPDP 2019, Porto, Portugal, October 7-9, 2019*, pages 1:1–1:3. ACM, 2019. doi:10.1145/3354166.3356340.

11 Luca Ciccone. Flexible coinduction in agda. Master's thesis, DIBRIS, Università di Genova, Italy, 2020. arXiv:2002.06047.

12 Luca Ciccone, Francesco Dagnino, and Elena Zucca. Flexible coinduction in Agda. In Schloss Dagstuhl Leibniz-Zentrum für Informatik, editor, *Proceedings of the 12th conference on Interactive Theorem Proving, ITP 2021*, 2021. to appear.

13 Luca Ciccone, Francesco Dagnino, and Elena Zucca. Inference Systems in Agda, 2021. URL: https://github.com/LcicC/inference-systems-agda [cited Feb 1, 2021].

14 Luca Ciccone and Luca Padovani. A Dependently Typed Linear π-Calculus in Agda. In *PPDP'20: 22nd International Symposium on Principles and Practice of Declarative Programming, Bologna, Italy, 9-10 September, 2020*, pages 8:1–8:14. ACM, 2020. doi:10.1145/3414080.3414109.

15 Luca Ciccone and Luca Padovani. Fair Subtyping in Agda, 2021. URL: https://github.com/boystrange/FairSubtypingAgda/tree/v1.0 [cited May 1, 2021].

16 Francesco Dagnino. Coaxioms: flexible coinductive definitions by inference systems. *Log. Methods Comput. Sci.*, 15(1), 2019. doi:10.23638/LMCS-15(1:26)2019.

17 Alain Frisch, Giuseppe Castagna, and Véronique Benzaken. Semantic subtyping: Dealing set-theoretically with function, union, intersection, and negation types. *J. ACM*, 55(4):19:1–19:64, 2008. doi:10.1145/1391289.1391293.

18 Simon J. Gay. Subtyping supports safe session substitution. In Sam Lindley, Conor McBride, Philip W. Trinder, and Donald Sannella, editors, *A List of Successes That Can Change the World - Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday*, volume 9600 of *Lecture Notes in Computer Science*, pages 95–108. Springer, 2016. doi:10.1007/978-3-319-30936-1_5.

19 Simon J. Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Informatica*, 42(2-3):191–225, 2005. doi:10.1007/s00236-005-0177-z.

20 Matthew Hennessy. *Algebraic theory of processes*. MIT Press series in the foundations of computing. MIT Press, 1988.

21 Kohei Honda. Types for dyadic interaction. In Eike Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993. doi:10.1007/3-540-57208-2_35.

22 Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In Chris Hankin, editor, *Programming Languages and Systems - ESOP'98, 7th European Symposium on Programming, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 1998. doi:10.1007/BFb0053567.

23 Hans Hüttel, Ivan Lanese, Vasco T. Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniélou, Dimitris Mostrous, Luca Padovani, António Ravara, Emilio Tuosto, Hugo Torres Vieira, and Gianluigi Zavattaro. Foundations of session types and behavioural contracts. *ACM Comput. Surv.*, 49(1):3:1–3:36, 2016. doi:10.1145/2873052.

24 Dexter Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983. doi:10.1016/0304-3975(82)90125-6.

25 Maurizio Murgia. A note on compliance relations and fixed points. In Massimo Bartoletti, Ludovic Henrio, Anastasia Mavridou, and Alceste Scalas, editors, *Proceedings 12th Interaction and Concurrency Experience, ICE 2019, Copenhagen, Denmark, 20-21 June 2019*, volume 304 of *EPTCS*, pages 38–47, 2019. doi:10.4204/EPTCS.304.3.

26 Rocco De Nicola and Matthew Hennessy. Testing equivalences for processes. *Theor. Comput. Sci.*, 34:83–133, 1984. doi:10.1016/0304-3975(84)90113-0.

27 Ulf Norell. *Towards a Practical Programming Language Based on Dependent Type Theory*. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, Göteborg, Sweden., 2007. URL: http://www.cse.chalmers.se/~ulfn/papers/thesis.pdf.

28 Susan S. Owicki and Leslie Lamport. Proving liveness properties of concurrent programs. *ACM Trans. Program. Lang. Syst.*, 4(3):455–495, 1982. doi:10.1145/357172.357178.

29 Luca Padovani. Fair subtyping for open session types. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, volume 7966 of *Lecture Notes in Computer Science*, pages 373–384. Springer, 2013. doi:10.1007/978-3-642-39212-2_34.

30 Luca Padovani. Fair subtyping for multi-party session types. *Math. Struct. Comput. Sci.*, 26(3):424–464, 2016. doi:10.1017/S096012951400022X.

31 Arend Rensink and Walter Vogler. Fair testing. *Inf. Comput.*, 205(2):125–198, 2007. doi:10.1016/j.ic.2006.06.002.

32 Peter Thiemann and Vasco T. Vasconcelos. Label-dependent session types. *Proc. ACM Program. Lang.*, 4(POPL):67:1–67:29, 2020. doi:10.1145/3371135.

33 Bernardo Toninho, Luís Caires, and Frank Pfenning. Dependent session types via intuitionistic linear type theory. In Peter Schneider-Kamp and Michael Hanus, editors, *Proceedings of the 13th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 20-22, 2011, Odense, Denmark*, pages 161–172. ACM, 2011. doi:10.1145/2003476.2003499.

34 Bernardo Toninho and Nobuko Yoshida. Depending on session-typed processes. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, volume 10803 of *Lecture Notes in Computer Science*, pages 128–145. Springer, 2018. doi:10.1007/978-3-319-89366-2_7.