Fair Termination of Binary Sessions

LUCA CICCONE, Università di Torino, Italy
LUCA PADOVANI, Università di Torino, Italy

A binary session is a private communication channel that connects two processes, each adhering to a protocol description called *session type*. In this work, we study the first type system that ensures the *fair termination* of binary sessions. A session fairly terminates if all of the infinite executions admitted by its protocol are deemed "unrealistic" because they violate certain *fairness assumptions*. Fair termination entails the eventual completion of all pending input/output actions, including those that depend on the completion of an unbounded number of other actions in possibly different sessions. This form of *lock freedom* allows us to address a large family of natural communication patterns that fall outside the scope of existing type systems. Our type system is also the first to adopt *fair subtyping*, a liveness-preserving refinement of the standard subtyping relation for session types that so far has only been studied theoretically. Fair subtyping is surprisingly subtle not only to characterize concisely but also to use appropriately, to the point that the type system must carefully account for all usages of fair subtyping to avoid compromising its liveness-preserving properties.

Additional Key Words and Phrases: session types, fair termination, fair subtyping, deadlock freedom

1 INTRODUCTION

Session type systems [Honda 1993; Honda et al. 1998; Hüttel et al. 2016] are an established formalism for the static analysis of communicating processes: a *binary session* is a private communication channel that connects two processes, each using one *endpoint* of the session; a *session type* is a type-level description of the sequences of input/output actions performed by a process with respect to a session endpoint. By making sure that the session types associated with the endpoints of a session complement each other and that processes do behave according to these types, it is possible to design type systems that enforce fundamental correctness properties such as communication safety, protocol fidelity, race and deadlock freedom. These are all instances of *safety properties*, guaranteeing that "nothing bad ever happens", but in general one is also interested in *liveness properties* guaranteeing that "something good eventually happens" [Owicki and Lamport 1982].

To illustrate a few examples of liveness properties, consider the process

$$A\langle x\rangle \mid B\langle x,y\rangle \mid C\langle y\rangle \qquad \text{where} \qquad \begin{array}{r@{\;}l} B(x,y) & \stackrel{\triangle}{=} x?\{\mathsf{add} : B\langle x,y\rangle, \mathsf{pay} : \dots y!\mathsf{ship}\dots\} \\ C(y) & \stackrel{\triangle}{=} y?\mathsf{ship}\dots \end{array} \tag{1}$$

that models an acquirer A purchasing items from a business B which interacts with a carrier C. The acquirer and the business are connected by a session x whereas the business and the carrier are connected by a session y. The process is intentionally incomplete, but we see that the business can receive from the acquirer an arbitrary number of add messages (each message modeling the fact that the acquirer has added an item to its shopping cart) or a single pay message. Only then the business sends the ship message and the carrier can make progress. Examples of liveness

Authors' addresses: Luca Ciccone, Dipartimento di Informatica, Università di Torino, Torino, Italy, luca.ciccone@unito.it; Luca Padovani, Dipartimento di Informatica, Università di Torino, Torino, Italy, luca.padovani@unito.it.

Manuscript submitted to ACM

properties are (L1) *the acquirer eventually sends* pay *to the business*, (L2) *the business eventually sends* ship *to the carrier*, and also (L3) *the sessions x and y eventually terminate*. Note that some of these properties are stronger than others. For example, if we know that (L3) holds, then both (L1) and (L2) hold too.

A type system ensuring properties like (L1)–(L3) for the process in Eq. (1) is missing. There exist (session) type systems ensuring somewhat related properties such as *deadlock freedom* [Caires et al. 2016; Dardha and Gay 2018; Padovani 2014; Wadler 2014], *lock freedom* [Kobayashi 2002; Kobayashi and Sangiorgi 2010; Padovani 2014; Scalas and Yoshida 2019] and *strong normalization* [Lindley and Morris 2016; Yoshida et al. 2004]. However, deadlock freedom is in general too weak to guarantee liveness properties such as (L1)–(L3). For example, if the acquirer keeps adding items to the cart but is never willing to pay the business, the process in Eq. (1) is deadlock-free even if the carrier cannot make any progress. At the same time, strong normalization implies (L1)–(L3) but is in general too strong, in the sense that there exist interesting processes that satisfy these properties but that are *not* strongly normalizing. For example, if the acquirer may buy arbitrarily many items, then the process in Eq. (1) admits, at least in principle, an infinite computation in which the acquirer keeps adding items to the cart even if it is willing to pay the business every now and then. Lock freedom is a liveness property meaning that every pending communication, such as the input y?ship, can eventually be completed. Hence, lock freedom is strong enough to entail *e.g.* (L2). However, currently available type systems ensuring lock freedom have an important limitation: they can only handle those processes in which the completion of pending actions on a channel is attainable *regardless of the content of messages exchanged in other channels*. This is not the case for the action y!ship on session y, which is performed only provided that a pay message is exchanged in session x. Note that B, far from being a contrived corner case, models a simple "while" loop that talks to A for an *arbitrarily long but supposedly finite amount of time* before turning the attention to C. In summary, Eq. (1) is representative of a large family of processes for which no available type system is able to provide strong liveness guarantees such as (L1)–(L3).

The type system we propose in this work ensures the fair termination of binary sessions. In general, fair termination means that a program terminates provided that some *fairness assumptions* are made on its executions [Francez 1986; Grumberg et al. 1984]. In our context, fair termination guarantees the eventual completion of every session under such assumptions. Fair termination is stronger than (dead)lock freedom but weaker than strong normalization. Unlike deadlock freedom, fair termination guarantees that every pending action may be completed. Unlike lock freedom, fair termination guarantees that every session may come to an end. Unlike strong normalization, fair termination does not rule out the existence of infinite executions, which are deemed "unrealistic" because they violate the fairness assumptions being made. There are two reasons why we focus on fair termination instead of lock freedom. First, the word "session" embodies the idea of an activity that lasts for a *finite period of time*, so (fair) session termination may be regarded as a desirable if not defining property of communicating sessions in the first place. Second, lock freedom does not scale well to multiple (chained, nested or interleaved) sessions. For instance, the stubborn acquirer that sends add messages forever results in a lock-free session x, but the action y!ship in the business that is meant to "unlock" the complementary action y?ship in the carrier can be performed only if the acquirer eventually sends pay. So, knowing that the session x is lock free does not help us to reason on the lock freedom of the session y. On the contrary, knowing that x fairly terminates is enough to deduce that pay can be sent, hence that y fairly terminates as well.

Among the several fairness notions that have been considered in the literature [Francez 1986; Kwiatkowska 1989; van Glabbeek and Höfner 2019] we assume *strong fairness*, namely we assume that a process that has infinitely many opportunities of making some choice will make that choice infinitely often. In the specific case of (1), this translates to the assumption that an acquirer periodically faced with the opportunity of paying the business eventually pays the business. The motivation for choosing strong fairness is that we need an assumption on the individual behavior of sequential processes and on the messages they choose to send, whereas weaker assumptions like *e.g.* justness [van Glabbeek 2019; van Glabbeek and Höfner 2019] only concern the way parallel processes may independently progress. The fairness assumption we make is an assumption in a literal sense; it cannot be guaranteed by the type system or by a *scheduler* [Apt et al. 1987; Francez 1986] because it concerns the internal behavior of the processes that partake in a session.

It should be noted that the mere *assumption* of strong fairness does not turn an ordinary session type system into one that ensures fair session termination because the correspondence imposed by the type system between the structure of processes and that of the protocols they implement is generally (often necessarily) a loose one. Indeed, processes may be "more accommodating" than the protocols they implement by handling more messages than those mentioned in the protocols. For example, the business in (1) could handle a search message in addition to add and pay, even if the session type associated with x does not mention search. At the same time, processes may also be "less demanding" than the protocols they implement by sending fewer messages than those allowed by the protocols. For example, the acquirer in (1) could always purchase an odd number of items, or at least n items, or no more than n items, even if the session type associated with x allows sending an arbitrary number of add messages. These mismatches between processes and protocols are usually reconciled by a *subtyping relation* for session types [Bernardi and Hennessy 2016; Gay and Hole 2005]. The problem is that this subtyping relation is *too coarse* because it has been conceived to preserve the *safety* properties of sessions but not termination, which is a *liveness* property: if session types are not sufficiently precise descriptions of the actual behavior of processes, a session that appears to be fairly terminating at the level of types may not terminate at all at the level of processes. To solve this problem we adopt *fair subtyping* [Bravetti et al. 2021; Padovani 2013, 2016], a *liveness-preserving* refinement of the subtyping relation defined by Gay and Hole [2005].

As it turns out, the strong fairness assumption and the adoption of fair subtyping are still insufficient to guarantee fair session termination. For example, a process could indefinitely delay the termination of a session if it is allowed to chain or nest an infinite number of other sessions, even if all the created sessions (fairly) terminate. An even subtler issue is that *fair subtyping can be easily abused*, in the sense that using it "infinitely often" in the typing derivation of a recursive process may compromise its liveness-preserving feature. To overcome these problems, the type system has to account for all the creations of new sessions and all the usages of fair subtyping, making sure that the overall effort required to terminate all open sessions remains finite.

Summary of contributions. We present a session type system that ensures the *fair termination* of binary sessions in a calculus that supports *general recursion, session interleaving, session delegation and dynamic session creation*. Fair termination implies that all pending communications, including those that are blocked by an unbounded number of other communications and that depend on the exchange of a particular message in possibly different sessions, can be completed in finite time. To the best of our knowledge, this is the first type system capable of ensuring this form of lock freedom for a family of processes large enough to include (1). We also solve the long-standing problem of devising a type system based on *fair subtyping*. We design the type system so that the usage of fair subtyping in a typing derivation is safe, uncovering a dangerous interaction between fair subtyping and delegation whereby a single usage of fair subtyping for higher-order session types may have the same overall effect of infinitely many usages. We show how to avoid this issue by reconsidering some established properties of (fair) subtyping for higher-order session types.

Structure of the paper. Section 2 provides a quick introduction to *generalized inference systems*, the formalism we use to define fair subtyping and the typing rules. Section 3 provides all the notions on session types that are necessary in this work, including fair subtyping and fair termination. Section 4 describes the process calculus and Section 5 motivates the key properties enforced by the type system through a series of examples. Section 6 formalizes the type system and states its soundness. In Section 7 we show the dangerous interaction between fair subtyping and higher-order session types. Section 8 provides a more detailed comparison with related work and Section 9 concludes. Proofs and further definitions and results can be found in the Appendix, which is published in the supplemental material section of the ACM Digital Library page for this paper.

2 GENERALIZED INFERENCE SYSTEMS IN A NUTSHELL

Inference systems [Aczel 1977] are ubiquitous in the definition of predicates, relations and typing rules. An *inference system* $\mathcal{I}$ over a *universe* $\mathcal{U}$ of *judgments* is a set of *rules* $\langle pr, j\rangle$ where $pr \subseteq \mathcal{U}$ is a set of *premises* and $j \in \mathcal{U}$ is the conclusion. A *derivation tree* of $\mathcal{I}$ is a tree such that each node is labeled with the conclusion of a rule in $\mathcal{I}$ and its children are labeled with the premises of the rule. We say that a judgment j is *derivable* in $\mathcal{I}$ if there exists a derivation tree with root j. An inference system can have different interpretations depending on the set of derivation trees that one considers. The *inductive interpretation* of an inference system is the set of judgments that are derivable with well-founded derivation trees, those having a finite depth. The *coinductive interpretation* of an inference system is the set of judgments that are derivable with arbitrary (finite- or infinite-depth) derivation trees. It is known that these two interpretations respectively coincide with the least and the greatest fixed points of the inference operator $\Phi_{\mathcal{I}}$ associated with the inference system $\mathcal{I}$:

$$\Phi_{\mathcal{I}}(X) \overset{\text{def}}{=} \{ j \in \mathcal{U} \mid \exists pr \subseteq X : \langle pr, j\rangle \in \mathcal{I} \} \qquad \text{for all } X \subseteq \mathcal{U}$$

In some cases, the desired set of derivable judgments is an intermediate fixed point of the inference operator other than the least/greatest one. Generalized inference systems [Ancona et al. 2017; Dagnino 2019] allow for the characterization of (some) intermediate fixed points. Specifically, a *generalized inference system* is a pair $(\mathcal{I}, \mathcal{I}_{\text{co}})$ of inference systems whose interpretation is the set of judgments having an arbitrary (finite- or infinite-depth) derivation tree using the rules in $\mathcal{I}$ but such that all the judgments in this derivation tree also have a finite-depth derivation tree using the rules in $\mathcal{I} \cup \mathcal{I}_{\text{co}}$. It can be shown that this interpretation coincides with the greatest fixed point of $\Phi_{\mathcal{I}}$ that is included in the least fixed point of $\Phi_{\mathcal{I}\cup\mathcal{I}_{\text{co}}}$. The elements of $\mathcal{I}_{\text{co}}$ are called *corules*.

Hereafter, we write (co)rules following standard conventions: we use *meta-variables* for specifying families of (co)rules in a compact way and we draw a horizontal line to separate the premises pr from the conclusion j of a rule $\langle pr, j\rangle$. We double the line to distinguish the corules.

Example 2.1. The generalized inference system below defines a predicate $maximum(l, x)$ asserting that x is the greatest element of a *possibly infinite* list l:

$$\frac{}{maximum(x :: [\,], x)} \qquad \frac{maximum(l, y)}{maximum(x :: l, \max\{x, y\})} \qquad \frac{}{\overline{maximum(x :: l, x)}}$$

The axiom on the left states that the greatest element of a list $x :: [\,]$ that contains only x is just x, whereas the rule in the middle states that the greatest element of a list $x :: l$ is the maximum among the head x and the greatest element of the tail l. The problem of these “plain” rules is that their inductive interpretation is sound but not complete (no infinite list has a greatest element), whereas their coinductive interpretation is complete but not sound (it is possible to derive the judgment $maximum(l, y)$ when y is a proper upper bound of all elements in l, even if y is not itself an element of

l). With the addition of the corule, we restrict the set of derivable judgments $maximum(l, x)$ to those also admitting a finite-depth derivation using one of the two axioms, imposing that x must be an element of l. ⌟

Because of their interpretation, generalized inference systems are convenient to define mixed safety/liveness properties: safety properties are usually based on an invariance argument and can be naturally captured by the (coinductively interpreted) rules of the inference system; liveness properties are usually based on a well-foundedness argument and can be naturally captured by the (inductively interpreted) rules and corules. We will use generalized inference systems to provide compact definitions of fair subtyping (Section 3.2) and of the typing rules (Section 6). The reader interested in the metatheory of generalized inference systems may refer to Ancona et al. [2017] and Dagnino [2019].

3 SESSION TYPES

3.1 Syntax and Semantics

We use l, a, b, $\dots$ to denote the elements of a given set $\mathcal{L}$ of *labels* which may include values with a specific interpretation such as booleans, natural numbers, and so forth. A session type describes the communication protocol that takes place over a channel, namely the allowed sequences of input/output actions performed by a process on that channel. We use *polarities* $p \in \{?, !\}$ to distinguish *input actions* (?) from *output actions* (!) and we write $p^\perp$ for the *opposite* or *dual* polarity of p so that $?^\perp = !$ and $!^\perp = ?$. Session types are the possibly infinite, regular trees [Courcelle 1983] coinductively generated by the grammar below:

$$\textbf{Session type} \qquad S, T \ ::= \ p\,\mathsf{end} \mid pS.T \mid p\{l_i : S_i\}_{i \in I}$$

Session types of the form $p\,\mathsf{end}$ describe channels used for exchanging a session termination signal and on which no further communication takes place. Session types of the form $pS.T$ describe channels used for exchanging another channel of type S and then according to T. Finally, session types of the form $p\{l_i : S_i\}_{i\in I}$ describe channels used for exchanging a label l_k and then according to S_k. Session types of the form $?\{l_i : S_i\}_{i\in I}$ and $!\{l_i : S_i\}_{i\in I}$ are sometimes referred to as *external* and *internal* choices respectively, to emphasize that the label being received or sent is always chosen by the sender process. In a session type $p\{l_i : S_i\}_{i\in I}$ we assume that I is not empty and that $i \neq j$ implies $l_i \neq l_j$ for every $i, j \in I$. Note that I is not necessarily finite, although regularity implies that there must be finitely many *distinct* S_i.

To improve readability we abbreviate $p\{l : S\}$ (when the choice is trivial) as $pl.S$ and we define two partial operations $+$ and $\oplus$ such that

$$?\{l : S_l\}_{l\in A} + ?\{l : S_l\}_{l\in B} = ?\{l : S_l\}_{l\in A\cup B} \quad \text{and} \quad !\{l : S_l\}_{l\in A} \oplus !\{l : S_l\}_{l\in B} = !\{l : S_l\}_{l\in A\cup B} \tag{2}$$

when $A, B \neq \emptyset$ and $A \cap B = \emptyset$. We use U and V in addition to S and T to range over session types. Hereafter we specify possibly infinite session types by means of equations $S = \dots$ where the right hand side of the equation may contain guarded occurrences of the metavariable S. Guardedness guarantees that a session type S satisfying the equation exists and is unique [Courcelle 1983].

We equip session types with a *labeled transition system* (LTS) that allows us to describe, at the type level, the sequences of actions performed by a process on a channel. We distinguish two kinds of transitions: *unobservable transitions* $S \longrightarrow T$ are made autonomously by the process; *observable transitions* $S \xrightarrow{\alpha} T$ are made by the process in cooperation with the one it is interacting with through the channel. The label α describes the kind of interaction and has either the form pS (indicating the exchange of a channel of type S) or the form pl (indicating the exchange of label l). The polarity p

indicates whether the message is received (?) or sent (!). The LTS is defined below:

$$?S.T \xrightarrow{?S} T \qquad !S.T \xrightarrow{!S} T \qquad ?\{l : S_l\}_{l\in A} \xrightarrow{?l} S_l \qquad !l.S \oplus T \longrightarrow !l.S \qquad !l.S \xrightarrow{!l} S$$

Note the different behaviors described by session types of the form $p\{l_i : S_i\}_{i\in I}$ depending on the polarity p. A process using a channel of type $?\{l_i : S_i\}_{i\in I}$ performs an observable transition for each of the labels l_i it is willing to receive. On the contrary, a process using a channel of type $!\{l_i : S_i\}_{i\in I}$ first *chooses* a particular label $l = l_k$ for some $k \in I$ (this choice is internal to the process and is therefore unobservable) and then *sends* the label l. As an example, the chain of transitions

$$!l.S \oplus T \longrightarrow !l.S \xrightarrow{!l} S$$

models a process that first chooses and then sends the label l. The choice of the label is irrevocable and not negotiable with the receiver process. Note that, according to the definition of $\oplus$ (*cf.* Eq. (2)), T must be an internal choice of labels different from l, hence $!l.S \oplus T$ is a non-trivial choice among two or more labels. Also, $!l.S$ admits no further unobservable transitions.

In the following we write $\Longrightarrow$ for the reflexive, transitive closure of $\longrightarrow$ and $\overset{\alpha}{\Longrightarrow}$ for the composition $\Longrightarrow\overset{\alpha}{\longrightarrow}$. We extend transitions to strings of actions so that $\overset{\alpha_1\cdots\alpha_1}{\Longrightarrow}$ stands for the composition $\overset{\alpha_1}{\Longrightarrow} \cdots \overset{\alpha_n}{\Longrightarrow}$. We let φ, ψ range over strings of actions, we write ε for the empty string of actions, $\leq$ for the usual prefix relation on strings, $S \overset{\varphi}{\Longrightarrow}$ if $S \overset{\varphi}{\Longrightarrow} T$ for some T and $S \overset{\varphi}{\not\Longrightarrow}$ if not $S \overset{\varphi}{\Longrightarrow}$.

Definition 3.1 (paths of a session type). We say that φ is a *path* of S if $S \overset{\varphi}{\Longrightarrow}$. We write paths$(S)$ for the (prefix-closed) set of paths of S, that is paths$(S) \overset{\text{def}}{=} \{\varphi \mid S \overset{\varphi}{\Longrightarrow}\}$.

Note the difference between the relations $\Longrightarrow$ and $\overset{\varepsilon}{\Longrightarrow}$. The former relation entails zero or more unobservable transitions, whereas the latter relation entails no transitions at all. For example, we have $!l.S \oplus T \Longrightarrow !l.S$ but $!l.S \oplus T \overset{\varepsilon}{\not\Longrightarrow} !l.S$. As a consequence, the session type T on the right hand side of a relation $S \overset{\varphi}{\Longrightarrow} T$ is guaranteed to be a *subtree* of S. This property is useful to uniquely identify a particular subtree of S by means of a path φ.

Definition 3.2 (residual of a session type). The *residual* of a session type S with respect to a path $\varphi \in$ paths(S), denoted by $S(\varphi)$, is the unique session type T such that $S \overset{\varphi}{\Longrightarrow} T$.

Remark 1. Recall that regular trees are made of finitely many *distinct* subtrees [Courcelle 1983]. In particular, the set of all the residuals of S, namely $\{S(\varphi) \mid \exists\varphi \in \text{paths}(S)\}$, is always finite for every S. However, this set does not necessarily include *all* of the subtrees of S, since session types U occurring in prefixes of the form $pU.T$ are not reachable through paths. ⌟

The family of session types describing protocols that can always eventually terminate are particularly important in this work. We say that a session type with this property is *bounded*.

Definition 3.3 (bounded session type). We say that S is *bounded* if, for every $\varphi \in$ paths(S), there exist ψ and p such that $S(\varphi\psi) = p\,\mathsf{end}$.

That is, every path of a bounded session type can be extended to a maximal one after which the protocol is ended. For example, the session type $S = !\mathsf{add}.S \oplus !\mathsf{pay}.!\mathsf{end}$ is bounded whereas $T = !\mathsf{add}.T$ is not. Note the difference between *finite* and bounded session types: every finite session type is bounded, but not every bounded session type is finite as illustrated by T above.

Table 1. Fair subtyping.

$$\text{[f-converge]}\quad \begin{array}{c} \forall\varphi \in \mathsf{paths}(S) \setminus \mathsf{paths}(T) : \exists\psi \leq \varphi, l \in \mathcal{L} : S(\psi!l) \leqslant T(\psi!l) \\ \hline\hline S \leqslant T \end{array} \qquad \text{[f-end]}\quad \frac{}{p\,\mathsf{end} \leqslant p\,\mathsf{end}}$$

$$\text{[f-channel]}\quad \frac{S \leqslant T}{pU.S \leqslant pU.T} \qquad \text{[f-label-in]}\quad \frac{S_i \leqslant T_i \ {}^{(i\in I)}}{?\{l_i : S_i\}_{i\in I} \leqslant ?\{l_j : T_j\}_{j\in J}}\ I \subseteq J \qquad \text{[f-label-out]}\quad \frac{S_j \leqslant T_j \ {}^{(j\in J)}}{!\{l_i : S_i\}_{i\in I} \leqslant !\{l_j : T_j\}_{j\in J}}\ J \subseteq I$$

3.2 Fair Subtyping

The key ingredient of our type system is *fair subtyping*, denoted by the symbol $\leqslant$. The basic intuition underlying the subtyping relation $S \leqslant T$ is given by the Liskov substitution principle [Liskov and Wing 1994]: if S is a subtype of T, channels of type S can be "safely" used where channels of type T are expected. We quote "safely" to stress that, in our work, subtyping is meant not only to preserve safety but also fair termination, which is a liveness property.

Fair subtyping is defined by the generalized interpretation of the inference system in Table 1. In addition to $\leqslant$, we write $\leqslant_{\mathsf{ind}}$ for the relation defined by the *inductive* interpretation of the same inference system and $\leqslant_{\mathsf{coind}}$ for the relation defined by the *coinductive* interpretation of the inference system in Table 1 excluding the corule [f-converge]. See Section 2 for the meaning of generalized and (co)inductive interpretations of a given set of inference rules.

To get a sense of fair subtyping, think of the relation $!\mathsf{a}.S \oplus !\mathsf{b}.T \leqslant !\mathsf{a}.S$ saying that a channel of type $!\mathsf{a}.S \oplus !\mathsf{b}.T$ can be safely used where a channel of type $!\mathsf{a}.S$ is expected. Indeed, a (well-typed) process that owns a channel of type $!\mathsf{a}.S$ will use it for sending an a label and then according to S. This behavior is also allowed by the protocol $!\mathsf{a}.S \oplus !\mathsf{b}.T$, hence safety is preserved by the substitution. Dually, the relation $?\mathsf{a}.S \leqslant ?\mathsf{a}.S + ?\mathsf{b}.T$ holds because a (well-typed) process that uses a channel of type $?\mathsf{a}.S + ?\mathsf{b}.T$ may receive either an a label or a b label, hence safety is preserved if the channel is replaced by another one of type $?\mathsf{a}.S$ from which only an a label can be received.

The relation $\leqslant_{\mathsf{coind}}$ – which we dub *unfair subtyping* – is essentially the same relation defined by Gay and Hole [2005], except that $\leqslant_{\mathsf{coind}}$ is *invariant* for higher-order session types (*cf.* [f-channel]). The reason for this restriction is that we need to account for *all* usages of fair subtyping and make sure that they are done in suitably identified regions of a process. Allowing variant subtyping for higher-order session types may introduce "undetected" usages of fair subtyping that can compromise fair termination. We will dedicate Section 7 to analyzing this problem more in detail. The subtle difference between fair and unfair subtyping is due to the corule [f-converge]. Since this corule is somewhat obscure, we explain it gradually starting with the following observations:

(1) Recall from Section 2 that $S \leqslant T$ implies $S \leqslant_{\mathsf{coind}} T$ and $S \leqslant_{\mathsf{ind}} T$. Hence, $\leqslant$ is a refinement of $\leqslant_{\mathsf{coind}}$ such that, for each pair of related session types S and T, there exists a *finite-depth* derivation tree for the judgment $S \leqslant T$ using the rules and possibly the corule [f-converge].

(2) When $S \leqslant_{\mathsf{coind}} T$ holds, it is not possible to establish a general correlation between $\mathsf{paths}(S)$ and $\mathsf{paths}(T)$. Indeed, [f-label-in] entails that some paths of T are not present in S if $I \subsetneq J$ and [f-label-out] entails that some paths of S are not present in T if $J \subsetneq I$.

(3) The judgment $S \leqslant T$ is trivially derivable using [f-converge] if the path inclusion relation $\mathsf{paths}(S) \subseteq \mathsf{paths}(T)$ holds. Since [f-label-out] is the only rule that allows T to have fewer paths than S, we deduce that [f-converge] limits (but does not always forbid) applications of [f-label-out] when $J \subsetneq I$.

(4) In general [f-converge] requires that, whenever a path φ of S is no longer present in T, it must be possible to find a prefix ψ of φ and an output $!l$ shared by both S and T such that $S(\psi!l)$ and $T(\psi!l)$ are one step closer to the region of S and T where path inclusion holds.

The reason why path inclusion plays such an important role in the definition of fair subtyping is that a process using a channel x of type T keeps using x according to T even if x is replaced by another channel of type $S \leqslant T$, without even realizing that the replacement has taken place. After all, this is what the "safe substitution principle" is based on. As a consequence, none of the paths in S that have disappeared in T will be offered to the process at the other end of the session x. If there are "too few" paths in T compared to S, then the replacement might compromise the termination of the process at the other end of the session, should it crucially rely on those paths to terminate. When $S \leqslant T$ (and therefore $S \leqslant_{\mathsf{ind}} T$) holds, the corule [f-converge] makes sure that the process using x of type S believing that x has type T is always at *finite distance* from the region where path inclusion between (some subtrees of) S and (the corresponding subtrees of) T holds. Moreover, this region is always reachable by means of *output actions* (those $!l$ mentioned in [f-converge]) which are performed actively by the process using x. In other words, the process using x is always able, in a finite amount of time and relying on choices and actions it can perform autonomously, to steer the interaction towards a region of the protocol where path inclusion holds, hence where a common path to session termination is guaranteed to exist.

As we will see in the examples below, there might be infinitely many paths in $\mathrm{paths}(S) \setminus \mathrm{paths}(T)$ even if $S \leqslant T$. Nonetheless, [f-converge] is guaranteed to have finitely many premises because of the regularity of session types (*cf.* Remark 1).

Example 3.4 (acquirer protocols). Consider the session types $S = !\mathsf{add}.S \oplus !\mathsf{pay}.!\mathsf{end}$ and $T = !\mathsf{add}.(!\mathsf{add}.T \oplus !\mathsf{pay}.!\mathsf{end})$ which might describe the protocols of potential acquirers in Eq. (1). The session type S describes an acquirer that purchases an arbitrary number of items, whereas the session type T describes an acquirer that always purchases an odd number of items.

Let us prove that $S \leqslant T$ holds. To this aim, we have to find a possibly infinite derivation tree for $S \leqslant T$ using the rules of Table 1 and, for each judgment $S' \leqslant T'$ in this derivation, also a finite derivation tree using the rules and the corule [f-converge]. The (infinite) derivation tree

$$\dfrac{\dfrac{\dfrac{\vdots}{S \leqslant T} \qquad \dfrac{}{!\mathsf{end} \leqslant !\mathsf{end}}\,[\text{f-end}]}{S \leqslant !\mathsf{add}.T \oplus !\mathsf{pay}.!\mathsf{end}}\,[\text{f-label-out}]}{S \leqslant T}\,[\text{f-label-out}]$$

proves that $S \leqslant_{\mathsf{coind}} T$ holds. There are three judgments occurring in this tree, namely $S \leqslant T$, $S \leqslant !\mathsf{add}.T \oplus !\mathsf{pay}.!\mathsf{end}$ and $!\mathsf{end} \leqslant !\mathsf{end}$, for which the following (finite) derivation trees can be obtained using [f-converge] and [f-end]:

$$\dfrac{\dfrac{}{!\mathsf{end} \leqslant !\mathsf{end}}\,[\text{f-end}]}{S \leqslant T}\,[\text{f-converge}] \qquad \dfrac{\dfrac{}{!\mathsf{end} \leqslant !\mathsf{end}}\,[\text{f-end}]}{S \leqslant !\mathsf{add}.T \oplus !\mathsf{pay}.!\mathsf{end}}\,[\text{f-converge}]$$

More generally, if $T_{n \in \mathbb{N}}$ is the family of session types such that

$$T_n = \underbrace{!\mathsf{add} \dots !\mathsf{add}}_{n}.(!\mathsf{add}.T_n \oplus !\mathsf{pay}.!\mathsf{end})$$

where each T_n has an initial sequence of n $!\mathsf{add}$ prefixes, it is possible to obtain similar derivation trees for $S \leqslant T_n$ for all $n \in \mathbb{N}$. However, if we consider the session type $T_\infty = !\mathsf{add}.T_\infty$. which is somehow the limit of the succession $T_{n\in\mathbb{N}}$, we see that $S \leqslant_{\mathsf{coind}} T_\infty$ holds but $S \leqslant T_\infty$ does not. Indeed, each $\varphi \in \mathsf{paths}(S) \setminus \mathsf{paths}(T_\infty)$ has the form $(!\mathsf{add})^k!\mathsf{pay}$ for some $k \in \mathbb{N}$ and there is no prefix ψ of φ and action $!l$ such that $S(\psi!l) \leqslant_{\mathsf{ind}} T_\infty(\psi!l)$. The difference between T_n and T_∞ is that an acquirer behaving as T_n periodically has an opportunity of sending pay, which is essential for the termination of the business, whereas an acquirer behaving as T_∞ keeps sending add forever. With this behavior, safety is preserved but fair termination is not. ⌟

Example 3.5 (random bit generator). In this example we see that there is no trivial correlation between fair subtyping and session type boundedness, contrarily to what the previous example might suggest. To this aim, imagine a service that generates random bits on demand. Its protocol could be described by the session type $S = ?\mathsf{more}.(!0.S \oplus !1.S) + ?\mathsf{stop}.!\mathsf{end}$ according to which the service sends a random bit if the client sends more and terminates if the client sends stop. Consider now a *fully biased* random bit generator that deterministically sends 0 on request. Its protocol is described by the session type $T = ?\mathsf{more}.!0.T + ?\mathsf{stop}.!\mathsf{end}$ and now we have that $S \leqslant_{\mathsf{coind}} T$ holds whereas $S \leqslant T$ does not. The fact that fair subtyping does not hold can be shown with an argument similar to that used in Example 3.4, although we will provide a much easier proof in Section 3.3. The point to notice here is that T is bounded just like S is. Interestingly, we have $S \leqslant_{\mathsf{coind}} T'$ and $S \nleqslant T'$ also if T' describes a *partially biased* random bit generator, which deterministically sends two (or more) 0s in succession:

$$T' = ?\mathsf{more}.(!0.(?\mathsf{more}.!0.T' + ?\mathsf{stop}.!\mathsf{end}) \oplus !1.T') + ?\mathsf{stop}.!\mathsf{end}$$

In summary, the contravariance of label outputs allowed by [F-LABEL-OUT] may be constrained in non-trivial ways by [F-CONVERGE], depending on how input and output actions alternate. ⌟

We conclude this overview of fair subtyping by stating two notable properties of $\leqslant$.

PROPOSITION 3.6. *(1) $\leqslant$ is a preorder and (2) if S, T are finite, then $S \leqslant T$ if and only if $S \leqslant_{\mathsf{coind}} T$.*

Concerning Item 1, while the reflexivity of $\leqslant$ is trivial and the transitivity or $\leqslant_{\mathsf{coind}}$ is folklore [Gay and Hole 2005], the proof of transitivity of $\leqslant$ is made complex by the corule [F-CONVERGE]. In fact, the only proof we know of this fact relies on a semantic characterization of $\leqslant$ like the one we describe in Section 3.3. Item 2 shows that fair and unfair subtyping *coincide for finite session types*, hence the two relations only differ when infinite session types are considered. In particular, it can be shown that the corule [F-CONVERGE] is admissible if only finite session types are considered.

3.3 Compatibility

In this section we develop the notion of *session type compatibility*, which is instrumental to our theory for several reasons. First of all, compatibility is the relation assuring that the two endpoints of a session are used in complementary ways and that the amount of work that is necessary to terminate the session is finite (Section 3.4). This amount contributes to the definition of a *measure* at the level of processes and is a key element in the soundness proof of the type system (Section 6). Also, we use compatibility to formalize the relationship between fair subtyping (Section 3.2), the standard notion of *duality* for session types (Section 3.5) and the notion of *fair termination* found in the literature (Section 3.6). Finally, compatibility provides the semantic grounds to justify the corule [F-CONVERGE] in the definition of fair subtyping and consequently the technical machinery that we use to prove that fair subtyping is a preorder (Item 1 of Proposition 3.6).

Intuitively, S and T are *compatible* when they entail a "correct interaction" between the two processes that use the peer endpoints of a session, one of type T and the other of type S. If we use a term of the form $S \mid T$ to describe the session as a whole, with the two interacting processes behaving as S and T, then we can formalize their interaction (at the type level) using the LTS for session types and the reduction rules

$$\frac{S \longrightarrow S'}{S \mid T \longrightarrow S' \mid T} \qquad \frac{T \longrightarrow T'}{S \mid T \longrightarrow S \mid T'} \qquad \frac{S \xrightarrow{\alpha^\perp} S' \qquad T \xrightarrow{\alpha} T'}{S \mid T \longrightarrow S' \mid T'}$$

where we write $\alpha^\perp$ for the dual action of α, obtained by changing the polarity of α with the opposite one. A reduction occurs whenever one of the connected processes performs an unobservable transition or when the two processes exchange a message by proposing complementary actions. As usual, we let $\Longrightarrow$ stand for the reflexive, transitive closure of $\longrightarrow$ and we write $S \mid T \longrightarrow\!\!\!\!\!/\;$ if there are no S' and T' such that $S \mid T \longrightarrow S' \mid T'$.

Session type compatibility is the property saying that every finite interaction between two peer processes over a session can always be extended so as to successfully terminate the session.

Definition 3.7 (session type compatibility). We say that S and T are *compatible*, notation $S \sim T$, if $S \mid T \Longrightarrow S' \mid T'$ implies $S' \mid T' \Longrightarrow p^\perp \mathsf{end} \mid p\, \mathsf{end}$ for some p.

Note that compatibility implies the absence of communication errors, whereby a channel of unexpected type or an unexpected label is exchanged. Indeed, both $!\mathsf{a}.S \mid ?\mathsf{b}.T$ and $!U.S \mid ?V.T$ are stuck if $\mathsf{a} \neq \mathsf{b}$ and $U \neq V$. Compatibility also implies progress, for a session where both peers simultaneously attempt at receiving or sending a message is (or becomes) stuck.

Example 3.8. Consider the session types defined in Example 3.4 as well as $U = ?\mathsf{add}.U + ?\mathsf{pay}.?\mathsf{end}$. It is easy to see that U is compatible with both S and T_1 as well as with all the T_n for $n \in \mathbb{N}$. However, U is incompatible with T_∞ – despite the fact that U and T_∞ may interact forever without getting stuck – because $U \mid T_\infty$ cannot reach the state $?\mathsf{end} \mid !\mathsf{end}$. In other words, U provides a semantic justification for $S \not\leqslant T_\infty$: a business that behaves according to U can (fairly) terminate if it interacts with any acquirer that behaves according to S or any of the T_n, but not if it interacts with an acquirer that behaves according to T_∞. ⌟

The next two results formalize the tight relationship between compatibility and fair subtyping. First of all, fair subtyping preserves compatibility.

Theorem 3.9. *If $S \leqslant T$, then $U \sim S$ implies $U \sim T$ for every U.*

Theorem 3.9 seems to reverse the direction of the substitution principle that we mentioned when introducing fair subtyping (Section 3.2). The contradiction is only apparent, however, and is resolved by observing that Liskov's principle speaks of right-to-left substitutability *of values/channels* whereas Theorem 3.9 speaks of left-to-right substitutability *of behaviors/processes*. Gay [2016] discusses more in detail these two different yet related viewpoints.

Fair subtyping is also the *coarsest* subtyping relation between (bounded) session types that preserves compatibility. More precisely:

Theorem 3.10. *If S is bounded and $U \sim S$ implies $U \sim T$ for every U, then $S \leqslant T$.*

Theorem 3.10 shows that the corule [F-CONVERGE] is a *necessary condition* to turn unfair subtyping into a compatibility-preserving subtyping relation, at least when bounded session types are related by fair subtyping. In other words, if

$S \leqslant_{\text{coind}} T$ and $S \nleqslant T$, then it is possible to find U that is compatible with S but not with T, as we have done in Examples 3.8 and 3.11.

We can combine Theorem 3.9 and Theorem 3.10 to prove the transitivity of $\leqslant$ (among bounded session types). Indeed, suppose that $S \leqslant U$ and $U \leqslant T$ hold, where S is bounded, and consider an arbitrary V that is compatible with S. By Theorem 3.9 we deduce that V is compatible with U and therefore with T. By Theorem 3.10 we conclude $S \leqslant T$.

Example 3.11. Consider once again the session types S and T defined in Example 3.5, which are not related by fair subtyping, and let $U = !\mathsf{more}.(?\mathsf{0}.U + ?\mathsf{1}.!\mathsf{stop}.?\mathsf{end})$. Note that U stops the interaction as soon as it receives a $\mathsf{1}$ from the random bit generator. For this reason, we have $U \sim S$ but $U \nsim T$ since T never sends $\mathsf{1}$. By Theorem 3.9, we deduce $S \nleqslant T$ as we had already argued in Example 3.5. We can use a similar reasoning to show that $S \nleqslant T'$, except that the witness behavior that distinguishes S from T' is slightly more involved. The idea is to design a session type V that ends as soon as it receives a $\mathsf{1}$ immediately after it has received a $\mathsf{0}$:

$$V = !\mathsf{more}.(?\mathsf{0}.!\mathsf{more}.(?\mathsf{0}.V + ?\mathsf{1}.!\mathsf{stop}.?\mathsf{end}) + ?\mathsf{1}.V)$$

The proof of Theorem 3.10 is based on an effective construction of a discriminating session type (such as U or V above) that is compatible with S but not with T whenever $S \leqslant_{\text{coind}} T$ holds but $S \leqslant_{\text{ind}} T$ does not. ⌟

Remark 2. The reason why Theorem 3.10 does *not* hold in general, but only when S is bounded, is that the notion of session type compatibility that we consider (Definition 3.7) induces a large family of session types that are semantically equivalent (in the sense that they are not compatible with any other session type) but syntactically unrelated. As an example, the session types $S = ?\mathsf{add}.S$ and $T = !\mathsf{pay}.T$ are incompatible with any other session type (including themselves) simply because they do not contain an end leaf. In this case it is trivially true that any session type compatible with S is also compatible with T, but S and T cannot be related using the definition of $\leqslant$ as it stands. To make the correspondence between fair subtyping and compatibility preservation exact it is necessary to adopt a slightly different notion of session type compatibility that is biased towards the successful termination of one of the two session participants [Ciccone and Padovani 2021b; Padovani 2013]. This one-sided compatibility does not capture exactly the notion of "correct session termination" as we intend it, according to which *both* participants are required to successfully terminate, but is the one used in the proof that $\leqslant$ is transitive in general. ⌟

3.4 Rank of a Session

In the soundness proof of our type system we need to quantify the amount of work required to terminate a particular session in which one process behaves as S and the other as T. To this aim, we define the rank of this session as the smallest number of interactions that lead $S \mid T$ to termination.

Definition 3.12 (rank). The *rank* of S and T, written $\|S, T\|$, is the element of $\mathbb{N} \cup \{\infty\}$ defined as

$$\|S, T\| \overset{\text{def}}{=} \min\{1 + |\varphi| \mid \exists \varphi, p : S \overset{\varphi^\perp}{\Longrightarrow} p^\perp\,\mathsf{end}, T \overset{\varphi}{\Longrightarrow} p\,\mathsf{end}\}$$

where $|\varphi|$ denotes the length of φ, $\varphi^\perp$ is the string of actions obtained by dualizing all the actions in φ, and we postulate that $\min \emptyset = \infty$.

As we will see in Section 4, our process calculus requires an explicit message exchange for closing a session. This is the reason why we add 1 to the length of all paths that lead S and T to termination, so that the rank of S and T measures the actual number of synchronizations that are necessary to terminate the session. Note that the rank $\|S, T\|$

is generally unrelated to the lengths of the shortest paths of S and T that lead to termination. For example, if we take $S = ?\mathsf{a}.!\mathsf{c}.?\mathsf{a}.?\mathsf{end} + ?\mathsf{b}.?\mathsf{end}$ and $T = !\mathsf{a}.(?\mathsf{c}.!\mathsf{a}.!\mathsf{end} + ?\mathsf{d}.!\mathsf{end})$ we see that the shortest path φ such that $S(\varphi) = ?\mathsf{end}$ is $?\mathsf{b}$ of length 1 and the shortest path ψ such that $T(\psi) = !\mathsf{end}$ is $!\mathsf{a}?\mathsf{d}$ of length 2, but $\|S,T\| = 4$.

The rank of two compatible session types is always finite and varies in agreement with subtyping:

Theorem 3.13. *If $U \sim S$, then (1) $\|U,S\| \in \mathbb{N}$ and (2) $S \leqslant_{\mathsf{coind}} T$ implies $\|U,S\| \leq \|U,T\|$.*

Theorem 3.13 shows that every usage of (fair) subtyping may increase the amount of work that is necessary to terminate a session (Item 2), although such amount is guaranteed to remain finite as long as compatibility is preserved (Item 1). This property justifies the adoption of fair subtyping over unfair subtyping, since fair subtyping preserves compatibility (Theorem 3.9) whereas unfair subtyping in general does not (Examples 3.8 and 3.11). Theorem 3.13 also suggests that the finiteness of the rank can be guaranteed only when fair subtyping is used *finitely many times*. For this reason, we will have to be careful on *where* fair subtyping is used in the typing derivation of recursive processes to avoid that "too many" applications of fair subtyping end up having the same effect of unfair subtyping (Section 5.3).

3.5 Duality

In binary session type theories, the two endpoints of a session are associated with session types such that one is the *dual* of the other. The dual of a session type S has the same overall structure of S, but opposite polarities for the corresponding actions. Formally:

Definition 3.14. The *dual* of a session type S, written $S^\perp$, is corecursively defined by the equations

$$(p\,\mathsf{end})^\perp = p^\perp\,\mathsf{end} \qquad (pS.T)^\perp = p^\perp S.T^\perp \qquad (p\{l_i : S_i\}_{i\in I})^\perp = p^\perp\{l_i : S_i^\perp\}_{i\in I}$$

Although duality guarantees communication safety and progress, it does not imply compatibility in general. To see this, consider the session type T_∞ from Example 3.4 and note that

$$T_\infty^\perp \mid T_\infty \longrightarrow T_\infty^\perp \mid T_\infty \longrightarrow \cdots \not\Longrightarrow p^\perp\,\mathsf{end} \mid p\,\mathsf{end}$$

so two processes adhering to $T_\infty^\perp$ and T_∞ would be able to interact forever, but without hope of terminating the session. This kind of interaction must be forbidden in our setting if we are interested in fairly terminating sessions, hence $T_\infty^\perp \not\sim T_\infty$. Still, there is a connection between duality, boundedness and compatibility that is fundamental in the proof of Theorem 3.10, as it guarantees that every bounded session type is compatible with at least another one, its dual.

Theorem 3.15. *$U^\perp \sim U$ if and only if U is bounded.*

3.6 Fair Termination

In this section we relate compatibility with the notions of strong fairness and fair termination found in the literature [Apt et al. 1987; Francez 1986; van Glabbeek and Höfner 2019]. In general, fairness assumptions are made to rule out those infinite runs of a system that are considered unrealistic. In order to formulate strong fairness, we must therefore define a notion of "run" in our setting.

Definition 3.16 (run). A *run* of $S \mid T$ is a sequence of reductions

$$S \mid T \longrightarrow S_1 \mid T_1 \longrightarrow S_2 \mid T_2 \longrightarrow \cdots$$

and it is *maximal* if either it is infinite or if it ends with a term $S_n \mid T_n$ such that $S_n \mid T_n \not\longrightarrow$.

Among all possible runs, we identify the "fair" ones as those in which *reductions that are enabled infinitely often occur infinitely often*. In the taxonomy of fairness notions [Kwiatkowska 1989; van Glabbeek and Höfner 2019], this particular one is called *strong fairness*. Formally:

Definition 3.17 (fair run). A run π is *fair* if, for every $S \mid T$ that occurs infinitely often in π and every $S' \mid T'$ such that $S \mid T \longrightarrow S' \mid T'$, the reduction $S \mid T \longrightarrow S' \mid T'$ occurs infinitely often in π.

This notion of fairness is known to enjoy two properties that we use in our development: (1) every finite run is a fair run; (2) every finite run can be extended to a maximal fair run. The second property is considered to be an essential requirement for every fairness notion and is referred to in the literature as *machine closure* [Lamport 2000] or *feasibility* [Apt et al. 1987; van Glabbeek and Höfner 2019]. We can now define what it means for a session to be fairly terminating:

Definition 3.18 (fair termination). We say that a session described by the pair of session types $S \mid T$ *fairly terminates* if all of its maximal fair runs are finite.

Example 3.19. Consider T and T_∞ from Example 3.4 and the session type $U = ?\mathsf{add}.U + ?\mathsf{pay}.?\mathsf{end}$. We can depict all the runs of $U \mid T$ as the infinite tree

$$\begin{tikzcd}
U \mid T \arrow[r] & U \mid !\mathsf{add}.T \oplus !\mathsf{pay}.!\mathsf{end} \arrow[r] \arrow[d] & U \mid !\mathsf{add}.T \arrow[r] & U \mid T \arrow[r] & \cdots \\
& U \mid !\mathsf{pay}.!\mathsf{end} \arrow[r] & ?\mathsf{end} \mid !\mathsf{end} & &
\end{tikzcd}$$

where we observe that every run of $U \mid T$ ending in $?\mathsf{end} \mid !\mathsf{end}$ is maximal, finite and therefore fair, whereas the only infinite run of $U \mid T$ is unfair since the reduction $U \mid !\mathsf{add}.T \oplus !\mathsf{pay}.?\mathsf{end} \longrightarrow U \mid !\mathsf{pay}.!\mathsf{end}$ is infinitely often enabled but never performed. On the other hand, $U \mid T_\infty$ has only one maximal run $U \mid T_\infty \longrightarrow U \mid T_\infty \longrightarrow \cdots$, which is infinite and fair. In summary, the session $U \mid T$ is fairly terminating, whereas the session $U \mid T_\infty$ is not. ⌟

We can now establish the tight relationship between the compatibility of S and T and fair termination of $S \mid T$ without residual pending communications.

Theorem 3.20. *For every S and T we have that $S \sim T$ if and only if every maximal fair run of $S \mid T$ is finite and ends with $p^\perp\,\mathsf{end} \mid p\,\mathsf{end}$ for some p.*

The "only if" part of Theorem 3.20 (without the requirement that the final term of the fair run has the form $p^\perp\,\mathsf{end} \mid p\,\mathsf{end}$) is known as *liveness enhancing property* of the fairness assumption [Apt et al. 1987; van Glabbeek and Höfner 2019]. It shows that the fairness assumption affects the liveness properties that can be proved: some liveness properties (*e.g.* termination) do not hold in general (there exist infinite runs) but they do hold if the unfair runs are ruled out (all fair runs are finite).

4 LANGUAGE SYNTAX AND SEMANTICS

The syntax of processes makes use of an infinite set of *channel names*, ranged over by x, y and z, and of a finite set of *process names*, ranged over by A, B and C. Hereafter, we use $\overline{x}$ to denote a possibly empty tuple of names, extending the same notation to other entities. A *program* $\mathcal{P}$ is a finite set $\{A_i(\overline{x_i}) \stackrel{\triangle}{=} P_i\}_{i \in I}$ of *definitions* where each P_i is a *process* generated by the grammar below:

$$\begin{array}{llllllll}
\textbf{Process} & P, Q & ::= & \mathsf{done} & \text{termination} & | & A\langle \overline{x} \rangle & \text{invocation} \\
 & & | & \mathsf{wait}\ x.P & \text{signal input} & | & \mathsf{close}\ x & \text{signal output} \\
 & & | & x?(y).P & \text{channel input} & | & x!y.P & \text{channel output} \\
 & & | & xp\{l_i : P_i\}_{i \in I} & \text{label input/output} & | & P \oplus_k Q & \text{choice} \\
 & & | & (x)(P \mid Q) & \text{session} & | & \lceil x \rceil P & \text{cast}
\end{array}$$

The process done is terminated and performs no action. The invocation $A\langle \overline{x} \rangle$ behaves as P if $A(\overline{x}) \triangleq P$ is the definition of A. When $\overline{x}$ is empty, we write A and $A \triangleq P$ instead of $A\langle\rangle$ and $A() \triangleq P$. The process $\mathsf{wait}\ x.P$ waits for a signal from channel x indicating that the session x is being closed and then continues as P. The process $\mathsf{close}\ x$ sends the termination signal on x. The process $x?(y).P$ receives a channel y from channel x and then continues as P. Dually, $x!y.P$ sends y on x and then continues as P. The process $xp\{l_i : P_i\}_{i \in I}$ exchanges a label l_i on channel x and then continues as P_i. As for session types, we assume that the set I in these forms is always non-empty and that $i \neq j$ implies $l_i \neq l_j$ for every $i, j \in I$. Also, we write $xpl_i.P_i$ instead of $xp\{l_i : P_i\}_{i \in I}$ when I is a singleton $\{i\}$. A non-deterministic choice $P_1 \oplus_k P_2$ reduces to either P_1 or P_2. The annotation $k \in \{1, 2\}$ has no operational meaning, it is only used to record that P_k leads to the termination of the process and is omitted when irrelevant. A session $(x)(P \mid Q)$ is the parallel composition of P and Q connected by x. Finally, a *cast* $\lceil x \rceil P$ behaves exactly as P. This form simply records the fact that the type of x is subject to an application of fair subtyping in the typing derivation for P. As we will see in Sections 5.3 and 6, we use this form to precisely account for all places in (the typing derivation of) a process where fair subtyping is used. Occasionally we write $\lceil x_1 \cdots x_n \rceil P$ for $\lceil x_1 \rceil \cdots \lceil x_n \rceil P$.

The only binders are channel inputs $x?(y).P$ and sessions $(x)(P \mid Q)$. We write $\mathsf{fn}(P)$ and $\mathsf{bn}(P)$ for the sets of free and bound channel names occurring in P and we identify processes modulo renaming of bound names. The program $\mathcal{P}$ that provides the meaning to the process names occurring in processes is often left implicit. Sometimes we write a process definition $A(\overline{x}) \triangleq P$ as a proposition or side condition, intending that such definition is part of the implicit program $\mathcal{P}$.

Example 4.1. Let us revisit and complete the example we sketched in Section 1. We can model the whole system as the following set of process definitions:

$$\begin{array}{ll}
\mathit{Main} \triangleq (y)((x)(\lceil x \rceil A\langle x \rangle \mid B\langle x, y \rangle) \mid C\langle y \rangle) & B(x, y) \triangleq x?\{\mathsf{add} : B\langle x, y \rangle, \mathsf{pay} : \mathsf{wait}\ x.y!\mathsf{ship}.\mathsf{close}\ y\} \\
A(x) \triangleq x!\mathsf{add}.x!\{\mathsf{add} : A\langle x \rangle, \mathsf{pay} : \mathsf{close}\ x\} & C(y) \triangleq y?\mathsf{ship}.\mathsf{wait}\ y.\mathsf{done}
\end{array}$$

Note that the acquirer deterministically sends add to the business as the first message, whereas it chooses among add and pay every other interaction. After the acquirer has sent pay, it closes the session x with the business B. At this point, the business sends ship to the carrier C and closes the session y. The cast $\lceil x \rceil$ before the invocation of $A\langle x \rangle$ in *Main* is meant to account for the mismatch between the behavior of the acquirer, which always adds an odd number of items to the cart, and that of the business, which accepts any number of items added to the shopping cart. $\lrcorner$

The operational semantics of processes is defined using a structural pre-congruence relation $\preccurlyeq$ and a reduction relation $\longrightarrow$, both of which are defined in Table 2 and described hereafter. Rules [s-par-comm] and [s-par-assoc] express the usual commutativity and associativity of parallel composition. In the case of [s-par-assoc], the side condition $x \in \mathsf{fn}(Q)$ makes sure that the session $(x)(P \mid Q)$ we obtain on the right hand side does indeed connect P and Q through x. Also note that [s-par-assoc] only describes a right-to-left associativity of parallel composition and that left-to-right associativity is derivable. The remaining axioms are those that justify the use of a pre-congruence over a symmetric

Table 2. Structural pre-congruence and reduction of processes.

[s-par-comm]	$(x)(P \mid Q)$	$\preccurlyeq$	$(x)(Q \mid P)$	
[s-par-assoc]	$(x)(P \mid (y)(Q \mid R))$	$\preccurlyeq$	$(y)((x)(P \mid Q) \mid R)$	if $x \in \mathsf{fn}(Q)$
[s-cast-comm]	$\lceil x \rceil \lceil y \rceil P$	$\preccurlyeq$	$\lceil y \rceil \lceil x \rceil P$	
[s-cast-new]	$(x)(\lceil x \rceil P \mid Q)$	$\preccurlyeq$	$(x)(P \mid Q)$	
[s-cast-swap]	$(x)(\lceil y \rceil P \mid Q)$	$\preccurlyeq$	$\lceil y \rceil (x)(P \mid Q)$	if $x \neq y$
[s-call]	$A\langle \overline{x} \rangle$	$\preccurlyeq$	P	if $A(\overline{x}) \overset{\triangle}{=} P$
[r-choice]	$P_1 \oplus P_2$	$\longrightarrow$	P_k	if $k \in \{1, 2\}$
[r-signal]	$(x)(\mathsf{close}\ x \mid \mathsf{wait}\ x.P)$	$\longrightarrow$	P	
[r-channel]	$(x)(x!y.P \mid x?(y).Q)$	$\longrightarrow$	$(x)(P \mid Q)$	
[r-pick]	$(x)(x!\{l_i : P_i\}_{i \in I} \mid Q)$	$\longrightarrow$	$(x)(x!l_k.P_k \mid Q)$	if $k \in I$ and $\lvert I \rvert > 1$
[r-label]	$(x)(x!l_k.P \mid x?\{l_i : Q_i\}_{i \in I})$	$\longrightarrow$	$(x)(P \mid Q_k)$	if $k \in I$
[r-par]	$(x)(P \mid R)$	$\longrightarrow$	$(x)(Q \mid R)$	if $P \longrightarrow Q$
[r-cast]	$\lceil x \rceil P$	$\longrightarrow$	$\lceil x \rceil Q$	if $P \longrightarrow Q$
[r-struct]	P	$\longrightarrow$	Q	if $P \preccurlyeq P' \longrightarrow Q' \preccurlyeq Q$

congruence relation. Since each usage of fair subtyping may increase the amount of work that is necessary to terminate a session (*cf.* Theorem 3.13), axiom [s-cast-new] annihilates a cast on x nearby the binder for x, making sure that casts can only be removed and never added. Axioms [s-cast-comm] and [s-cast-swap] are used to move casts closer to their binder so that they can be annihilated with [s-cast-new]. Rule [s-call] unfolds process invocations to their definition.

The reduction rules are mostly unremarkable: [r-choice] models the non-deterministic choice between alternative behaviors; [r-pick] models a non-trivial choice among a set of labels to send; [r-signal], [r-label] and [r-channel] model synchronizations between a sender (on the left hand side of the parallel composition) and a receiver (on the right hand side of the parallel composition) with [r-signal] removing the binder of a closed session; [r-par], [r-cast] and [r-struct] close reductions under parallel compositions, under casts and by structural pre-congruence. In the following we write $\Longrightarrow$ for the reflexive, transitive closure of $\longrightarrow$ and $\Longrightarrow^{+}$ for $\Longrightarrow\longrightarrow$.

We can now define the property enforced by our type system.

Definition 4.2. We say that P is *fairly terminating* if $P \Longrightarrow Q$ implies $Q \preccurlyeq \mathsf{done}$ or $Q \Longrightarrow^{+} \mathsf{done}$.

Remark 3. The definitions of session type compatibility (Definition 3.7) and of fair process termination (Definition 4.2) are inspired to that of *successful computation* in fair testing theories [Natarajan and Cleaveland 1995; Rensink and Vogler 2007]. Rensink and Vogler [2007] show that these notions have a *built-in fairness assumption* that coincides with strong fairness, at least in the case of *finite-state processes* (in fact, Theorem 3.20 is a particular instance of this result for session types). But while defining fair runs for session types is doable with little effort (*cf.* Definition 3.17), the definition of fair runs for the π-calculus is much more involved [Bidinger and Compagnoni 2009; Cacciagrano et al. 2006, 2009; Kobayashi 2002]. Besides, none of the available definitions is directly applicable to our language since they are all based on choiceless versions of the π-calculus with replication instead of recursion. For these reasons, we adopt the formulation of fair process termination in Definition 4.2 for its appeal and simplicity: the reachability of done implies that every pending action (resp. open session) in Q may eventually be performed (resp. terminated). ⌟

Example 4.3. With the definitions given in Example 4.1, it is easy to see that there is an infinite reduction sequence starting from *Main* in which the acquirer keeps adding items to the cart:

$$
\begin{aligned}
(y)((x)(\lceil x\rceil A\langle x\rangle \mid B\langle x,y\rangle) \mid C\langle y\rangle) &\Longrightarrow (y)((x)(x!\{\mathsf{add} : A\langle x\rangle, \mathsf{pay} : \mathsf{close}\ x\} \mid B\langle x,y\rangle) \mid C\langle y\rangle)\\
&\longrightarrow (y)((x)(x!\mathsf{add}.A\langle x\rangle \mid B\langle x,y\rangle) \mid C\langle y\rangle)\\
&\Longrightarrow (y)((x)(A\langle x\rangle \mid B\langle x,y\rangle) \mid C\langle y\rangle) \longrightarrow \cdots
\end{aligned}
$$

Nonetheless, *Main* is fairly terminating. For example, we have:

$$
\begin{aligned}
&(y)((x)(x!\{\mathsf{add} : A\langle x\rangle, \mathsf{pay} : \mathsf{close}\ x\} \mid B\langle x,y\rangle) \mid C\langle y\rangle)\\
&\quad\longrightarrow (y)((x)(x!\mathsf{pay}.\mathsf{close}\ x \mid B\langle x,y\rangle) \mid C\langle y\rangle)\\
&\quad\Longrightarrow (y)((x)(\mathsf{close}\ x \mid \mathsf{wait}\ x.y!\mathsf{ship}.\mathsf{close}\ y) \mid C\langle y\rangle)\\
&\quad\longrightarrow (y)(y!\mathsf{ship}.\mathsf{close}\ y \mid C\langle y\rangle) \Longrightarrow (y)(\mathsf{close}\ y \mid \mathsf{wait}\ y.\mathsf{done}) \longrightarrow \mathsf{done}
\end{aligned}
$$

Note that in general it might be necessary for the acquirer to add one more item to the cart before it can send the payment to the business and the carrier receives a ship message. ⌟

5 THE TYPE SYSTEM BY EXAMPLES

In this section we motivate, through a series of examples, the key properties enforced by the type system that, taken together, guarantee fair termination. There are two families of problems that can compromise fair termination. First of all, the process (or part thereof) may be unable to reduce further but is not done. In our model, this can happen for many reasons, for example: a process attempts at sending a label on a session that the receiver is not willing to accept; a process attempts at sending a termination signal when the receiver expects a channel; the processes at the two ends of the same session are both waiting for a message from that session. These are all examples of *safety violations*, which are prevented by any ordinary session type system. In this section we focus instead on *liveness violations*. Roughly speaking, liveness is violated when a process (or part thereof) engages an infinite computation that cannot possibly terminate. In Section 3.2 we have introduced a fair subtyping relation that is liveness preserving but, as we will see in a moment, the adoption of fair subtyping alone is not enough to rule out all potential liveness violations. The type system must also enforce three properties that we call *action boundedness*, *session boundedness* and *cast boundedness* guaranteeing that the overall effort required to terminate the process is finite. In the rest of the section we describe informally these properties and we show that violating even just one of them may compromise fair process termination. In doing so, we assume that the reader has some familiarity with the basic features of session type systems, those that prevent the aforementioned safety violations. If not, it might be worth revisiting this section after reading Section 6.

5.1 Action Boundedness

We say that a process is *action bounded* if there is a finite upper bound to the number of actions it has to perform in order to terminate. An action-unbounded process cannot terminate. Compare

$$
A \triangleq A \oplus \mathsf{done} \qquad \text{and} \qquad B \triangleq B \oplus B \tag{3}
$$

and observe that A may always reduce to done, whereas B can only reduce forever into itself. So A is action bounded whereas B is not. We consider a parallel composition action bounded if so are *both* processes composed in parallel.

Action boundedness is a necessary condition for (fair) process termination, hence the type system must guarantee that well-typed processes are action bounded. As we will see in Section 6, this can be easily achieved by means of *typing corules*. Besides, action boundedness carries along two welcome side effects. The first one is that degenerate process definitions such as $A \stackrel{\triangle}{=} A$ are not action bounded and therefore are flagged as ill typed by the type system. This guarantees that finitely many unfoldings of recursive process invocations always suffice to expose some observable process behavior. The second is that action boundedness allows us to detect recursive processes that claim to use a channel in a certain way when in fact they never do so. As an example, compare

$$A(x,y) \stackrel{\triangle}{=} x!\mathsf{a}.A\langle x,y\rangle \oplus x!\mathsf{b}.\mathsf{close}\ x \qquad \text{and} \qquad B(x,y) \stackrel{\triangle}{=} x!\mathsf{a}.B\langle x,y\rangle$$

where $A\langle x,y\rangle$ is action bounded and $B\langle x,y\rangle$ is not. An ordinary session type system with coinductively interpreted typing rules would accept $B\langle x,y\rangle$ regardless of y's type on the grounds that y occurs once in the body of B, hence it is "used" linearly. This is unfortunate, since y is not used in any meaningful way other than being passed as an argument of B. In A, the same linearity check promptly detects that y is not used along the path to $\mathsf{close}\ x$ that proves the boundedness of $A\langle x,y\rangle$.

5.2 Session Boundedness

We say that a process is *session bounded* if there is a finite upper bound to the number of sessions it has to create in order to terminate. It is easy to construct non-terminating processes by chaining together an infinite number of finite (or fairly terminating) sessions. For example, compare

$$A \stackrel{\triangle}{=} (x)(\mathsf{close}\ x \mid \mathsf{wait}\ x.A) \oplus \mathsf{done} \qquad \text{and} \qquad B_1 \stackrel{\triangle}{=} (x)(\mathsf{close}\ x \mid \mathsf{wait}\ x.B_1) \tag{4}$$

where A always has a possibility to terminate without creating new sessions (it is session bounded) while B_1 does not (it is session unbounded). It could be argued that B_1 is already ruled out because it is not action bounded. Indeed, while the left-hand side of the parallel composition in B_1 is finite, the right hand side is not (recall that we require *both* sides of a parallel composition to admit a finite path to either done or $\mathsf{close}\ x$). Below is a slightly more complex variation of B_1 that is action bounded and session unbounded. The trick is to have a finite branch on one side of the parallel composition matched by an infinite one on the other side:

$$B_2 \stackrel{\triangle}{=} (x)(x!\{\mathsf{a} : \mathsf{close}\ x, \mathsf{b} : \mathsf{wait}\ x.B_2\} \mid x?\{\mathsf{a} : \mathsf{wait}\ x.B_2, \mathsf{b} : \mathsf{close}\ x\}) \tag{5}$$

Eq. (4) shows that a session bounded process like A may still create an unbounded number of sessions. Below is another example of session bounded process that creates unboundedly many *nested* sessions, such that the first session being created is also the last one being completed:

$$(x)(C\langle x\rangle \mid \mathsf{wait}\ x.\mathsf{done}) \qquad \text{where} \qquad C(x) \stackrel{\triangle}{=} (y)(C\langle y\rangle \mid \mathsf{wait}\ y.\mathsf{close}\ x) \oplus \mathsf{close}\ x \tag{6}$$

While both A and C *may* create an arbitrary number of sessions, they do not *have to* do so in order to terminate. This is what sets them apart from B_1 and B_2.

5.3 Cast Boundedness

We say that a process is *cast bounded* if there is a finite upper bound to the number of casts it has to perform in order to terminate. Performing a cast means applying [S-CAST-NEW], which corresponds to a usage of fair subtyping. The reason why cast boundedness is fundamental is that the liveness-preserving property of fair subtyping holds as long as fair subtyping is used finitely many times. Conversely, infinitely many usages of fair subtyping may have the overall effect of a single usage of unfair subtyping (*cf.* Example 3.4). By "infinitely many usages" we mean usages of fair subtyping that occur within a loop in a recursive process. To illustrate the problem, let us consider the (non-terminating) process

$$(x)(A\langle x\rangle \mid B\langle x\rangle) \qquad \text{where} \qquad \begin{array}{l} A(x) \triangleq \lceil x\rceil x!\mathsf{add}.A\langle x\rangle \\ B(x) \triangleq x?\{\mathsf{add} : B\langle x\rangle, \mathsf{pay} : \mathsf{wait}\ x.\mathsf{done}\} \end{array} \tag{7}$$

and the session type $S = !\mathsf{add}.S \oplus !\mathsf{pay}.!\mathsf{end}$. It can be argued that the channel x is used according to S in $A(x)$ and according to $S^\perp$ in $B(x)$. Indeed, the structure of $B(x)$ matches perfectly that of $S^\perp$ and $x!\mathsf{add}.A(x)$ uses x according to $!\mathsf{add}.S$, which is a fair supertype of S accounted for by the cast $\lceil x\rceil$ in A. With this cast it is as if $A(x)$ promises to make a choice between sending add and sending pay at each iteration, but systematically favors add over pay. The overall effect of these unfulfilled promises is that the actual behavior of $A(x)$ over x is better described by the session type $T_\infty = !\mathsf{add}.T_\infty$, which is *not* a fair supertype of S as we have seen in Examples 3.4 and 3.8.

Although $A(x)$ could be rejected on the grounds that it is not action bounded, it is possible to find an action-bounded (but slightly more involved) variation of $A(x)$ and $B(x)$ in Eq. (7) in which the same phenomenon occurs. With the definitions

$$\begin{array}{l} A(x) \triangleq \lceil x\rceil x!\mathsf{more}.x?\{\mathsf{more} : A\langle x\rangle, \mathsf{stop} : \mathsf{wait}\ x.\mathsf{done}\} \\ B(x) \triangleq x?\{\mathsf{more} : \lceil x\rceil x!\mathsf{more}.B\langle x\rangle, \mathsf{stop} : \mathsf{wait}\ x.\mathsf{done}\} \end{array} \tag{8}$$

both $A(x)$ and $B(x)$ have a chance to continue or to terminate the session by sending either more or stop, except that they systematically favor more over stop. Now, if we consider the session type $S = !\mathsf{more}.(?\mathsf{more}.S + ?\mathsf{stop}.?\mathsf{end}) \oplus !\mathsf{stop}.!\mathsf{end}$, it can be argued that $A(x)$ uses x according to $S_A = !\mathsf{more}.(?\mathsf{more}.S + ?\mathsf{stop}.?\mathsf{end})$, which is a fair supertype of S, and that $B(x)$ uses x according to $S_B = ?\mathsf{more}.!\mathsf{more}.S^\perp + ?\mathsf{stop}.?\mathsf{end}$, which is a fair supertype of $S^\perp$. The two casts in Eq. (8) account for the differences between S and S_A in $A(x)$ and between $S^\perp$ and S_B in $B(x)$, but they occur within loops along paths that lead to process termination, hence A and B are not cast bounded.

It is worth discussing one last attempt to work around the problem, by moving the casts outward from within $A(x)$ and $B(x)$, as in

$$(x)(\lceil x\rceil A\langle x\rangle \mid \lceil x\rceil B\langle x\rangle) \qquad \text{where} \qquad \begin{array}{l} A(x) \triangleq x!\mathsf{more}.x?\{\mathsf{more} : A\langle x\rangle, \mathsf{stop} : \mathsf{wait}\ x.\mathsf{done}\} \\ B(x) \triangleq x?\{\mathsf{more} : x!\mathsf{more}.B\langle x\rangle, \mathsf{stop} : \mathsf{wait}\ x.\mathsf{done}\} \end{array} \tag{9}$$

Now $A(x)$ uses x according to $T_A = !\mathsf{more}.(?\mathsf{more}.T_A + ?\mathsf{stop}.?\mathsf{end})$ and $B(x)$ uses x according to $T_B = ?\mathsf{more}.!\mathsf{more}.T_B + ?\mathsf{stop}.?\mathsf{end}$, but while $S \leqslant_{\mathsf{coind}} T_A$ and $S^\perp \leqslant_{\mathsf{coind}} T_B$ both hold neither $S \leqslant T_A$ nor $S^\perp \leqslant T_B$ does. In summary, the non-terminating process in Eq. (9) is action bounded, session bounded and cast bounded, but it is typeable only using *unfair* subtyping.

6 THE TYPE SYSTEM, FORMALLY

The typing rules resemble those of a traditional session type system but differ in a few key aspects. First of all, they establish a tighter-than-usual correspondence between types and processes so that any discrepancy between actual and expected types is accounted for by explicit casts. This way, we make sure that actions leading to the termination of a session *at the type level* are matched by corresponding actions *at the process level*, a key property used in the soundness proof of the type system (Theorem 6.4). In addition, the typing rules enforce the boundedness properties informally described in the previous section. Action boundedness is enforced by specifying the typing rules as a generalized inference system and using two corules to make sure that every well-typed process is at finite distance from done or a close x. Concerning session and cast boundedness, we annotate typing judgments with a *rank*, that is an upper bound

Table 3. Typing rules.

$$
\text{[t-done]}\ \begin{array}{c} \\ \hline \emptyset \vdash^n \mathsf{done} \end{array}
\qquad
\text{[t-wait]}\ \begin{array}{c} \Gamma \vdash^n P \\ \hline \Gamma, x : ?\mathsf{end} \vdash^n \mathsf{wait}\ x.P \end{array}
\qquad
\text{[t-close]}\ \begin{array}{c} \\ \hline x : !\mathsf{end} \vdash^n \mathsf{close}\ x \end{array}
\qquad
\text{[t-channel-in]}\ \begin{array}{c} \Gamma, x : S, y : T \vdash^n P \\ \hline \Gamma, x : ?T.S \vdash^n x?(y).P \end{array}
$$

$$
\text{[t-channel-out]}\ \begin{array}{c} \Gamma, x : S \vdash^n P \\ \hline \Gamma, x : !T.S, y : T \vdash^n x!y.P \end{array}
\qquad
\text{[t-label]}\ \begin{array}{c} \Gamma, x : S_i \vdash^n P_i \ ^{(i \in I)} \\ \hline \Gamma, x : p\{l_i : S_i\}_{i \in I} \vdash^n xp\{l_i : P_i\}_{i \in I} \end{array}
\qquad
\text{[t-choice]}\ \begin{array}{c} \Gamma \vdash^{n_1} P \qquad \Gamma \vdash^{n_2} Q \\ \hline \Gamma \vdash^{n_k} P \oplus_k Q \end{array}
$$

$$
\text{[t-cast]}\ \begin{array}{c} \Gamma, x : T \vdash^n P \\ \hline \Gamma, x : S \vdash^{1+n} \lceil x \rceil P \end{array}\ S \leqslant T
\qquad
\text{[co-label]}\ \begin{array}{c} \Gamma, x : S_k \vdash^n P_k \\ \hline\hline \Gamma, x : p\{l_i : S_i\}_{i \in I} \vdash^n xp\{l_i : P_i\}_{i \in I} \end{array}\ k \in I
\qquad
\text{[co-choice]}\ \begin{array}{c} \Gamma \vdash^n P_k \\ \hline\hline \Gamma \vdash^n P_1 \oplus_k P_2 \end{array}
$$

$$
\text{[t-par]}\ \begin{array}{c} \Gamma, x : S \vdash^m P \qquad \Delta, x : T \vdash^n Q \\ \hline \Gamma, \Delta \vdash^{1+m+n} (x)(P \mid Q) \end{array}\ S \sim T
\qquad
\text{[t-call]}\ \begin{array}{c} \overline{x : S} \vdash^n P \\ \hline \overline{x : S} \vdash^{m+n} A\langle \overline{x} \rangle \end{array}\ A : [\overline{S}; n], A(\overline{x}) \triangleq P
$$

to the number of casts that must be performed and of sessions that must be created in order to terminate the process in the judgment.

The typing rules are defined by the generalized inference system in Table 3 and derive judgements of the form $\Gamma \vdash^n P$, meaning that P is well typed in the *typing context* Γ and has rank n. A typing context is a finite map from channels to session types written $x_1 : S_1, \dots, x_n : S_n$ or $\overline{x : S}$. We use Γ and Δ to range over typing contexts, we write $\emptyset$ for the empty context and Γ, Δ for the union of Γ and Δ when they have disjoint domains. We type check a program $\{A_i(\overline{x_i}) \triangleq P_i\}_{i \in I}$ under a global set of type assignments $\{A_i : [\overline{S_i}; n_i]\}_{i \in I}$ associating each process name A_i with a tuple of session types $\overline{S_i}$ and a rank n_i. The program is well typed if $\overline{x_i : S_i} \vdash^{n_i} P_i$ for every $i \in I$, establishing that the tuple $\overline{S_i}$ corresponds to the way the channels $\overline{x_i}$ are used by P_i and that n_i is a feasible rank annotation for P_i. Hereafter, we omit the rank from judgments when it is not important.

Let us look at the typing (co)rules in detail. [t-done] is the usual axiom requiring that the terminated process leaves no unused channels behind. Since done performs no casts and creates no sessions, it can have any rank. Rules [t-wait] and [t-close] concern the exchange of session termination signals. There is nothing remarkable here except noting once again that the rank of close x can be arbitrary. Rules [t-channel-in] and [t-channel-out] are similar, but they concern the exchange of channels. Note that, in [t-channel-out], the type T of the message y is required to match *exactly* that in the type of the channel x used for the communication, whereas Gay and Hole [2005] allow the type of y to be a subtype of T. This is one instance of the "tight correspondence" that we mentioned earlier. The rule [t-label] deals with the input/output of labels. As usual, any channel other than the one affected by the communication must be used in exactly the same way in every branch. However, the rule is stricter than that of Gay and Hole [2005] because it requires an exact correspondence between the labels that can be exchanged on x by the process and those in the type of x. The fact that a conclusion and premises are all annotated with the same rank n means that n is an upper bound for the rank of all branches of a label input/output. The corule [co-label] does not impose additional constraints compared to [t-label] and has *exactly one premise*, corresponding to one branch of the process in the conclusion. The effect of [co-label], when interpreted inductively together with the other rules, is to ensure the existence of a finite typing derivation whose leaves are applications of [t-done] or [t-close], hence action boundedness.

Rule [t-choice] is a standard typing rule for non-deterministic choices, requiring that both branches are well typed in exactly the same typing context. Notice that the rank of a choice $P_1 \oplus_k P_2$ is determined by the branch indexed by the k annotation, which is elected as the branch that leads to termination (see also Remark 4 for a comparison with [t-label]). Like [co-label], the associated corule [co-choice] ensures that the same branch gets closer to done or a $\mathsf{close}\ x$ to enforce action boundedness. Without this corule, it would not be possible to find a *finite-depth* derivation tree for an action-bounded process such as A in Eq. (3). Coherently with [t-choice], the same branch that leads to termination is also the one that determines the rank of the choice as a whole.

Rule [t-cast] is Liskov's substitution principle formulated as an inference rule. It states that a channel x of type S can be safely used where a channel of type T is expected, provided that $S \leqslant T$. The most important detail to notice here is that the rank of a cast is one plus that of the process in which the cast has effect. This way we account for this cast in the rank of the process so as to guarantee cast boundedness. Rule [t-par] concerns parallel composition and session creation. The rule is shaped after the cut rule of linear logic also adopted in other session type systems based on linear logic [Caires et al. 2016; Lindley and Morris 2016; Wadler 2014]. In particular, the parallel processes P and Q share no channel other than the session x that connects them, so as to prevent mutual dependencies between sessions and guarantee deadlock freedom. The side condition $S \sim T$ requires that the way in which P and Q use channel x is such that the session x can fairly terminate (*cf.* Definition 3.7). We *do not* require that S and T are dual to each other because reductions (*cf.* [r-pick]) and structural pre-congruence (*cf.* [s-cast-new]) do not necessarily preserve session type duality. Also, duality does not always imply compatibility (Section 3.5). The rank of a parallel composition is one plus that of the composed processes. By accounting for each occurrence of parallel compositions in the rank, we guarantee that well-typed processes are session bounded.

Finally, rule [t-call] states that a process invocation $A\langle\overline{x}\rangle$ is well typed provided that the types associated with $\overline{x}$ match those of the global assignment $A : [\overline{S}; n]$. Note that [t-call] is *not* an axiom: its premise (re)checks that the body P in the definition of A is coherent with the global type assignment $A : [\overline{S}; n]$. With this formulation of [t-call], the only axioms are [t-done] and [t-close] so that the inductive interpretation of the typing (co)rules ensures action boundedness. Note also that the rank of the conclusion may be greater than the rank n associated with A. This overapproximation grants more flexibility when typing different branches in [t-label].

Example 6.1. To show that the program defined in Example 4.1 is well typed, consider the session types $S = ?\mathsf{add}.S + ?\mathsf{pay}.?\mathsf{end}$ and $T = !\mathsf{add}.(!\mathsf{add}.T \oplus !\mathsf{pay}.!\mathsf{end})$ and the global type assignments $A : [T; 0]$, $B : [S, !\mathsf{ship}.!\mathsf{end}; 0]$, $C : [?\mathsf{ship}.?\mathsf{end}; 0]$ and $\mathit{Main} : [(); 3]$. We can obtain typing derivations for A, B and C using a null rank. In particular, we derive

$$\dfrac{\dfrac{\dfrac{\vdots}{x : T \vdash^0 A\langle x\rangle}\,[\textsc{t-call}] \qquad \dfrac{}{x : !\mathsf{end} \vdash^0 \mathsf{close}\ x}\,[\textsc{t-close}]}{x : !\mathsf{add}.T \oplus !\mathsf{pay}.!\mathsf{end} \vdash^0 x!\{\mathsf{add} : A\langle x\rangle, \mathsf{pay} : \mathsf{close}\ x\}}\,[\textsc{t-label}]}{x : T \vdash^0 x!\mathsf{add}.x!\{\mathsf{add} : A\langle x\rangle, \mathsf{pay} : \mathsf{close}\ x\}}\,[\textsc{t-label}]$$

for the definition of A and

$$\dfrac{\dfrac{\vdots}{x : S, y : !\mathsf{ship}.!\mathsf{end} \vdash^0 B\langle x, y\rangle}\,[\textsc{t-call}] \qquad \dfrac{\dfrac{\dfrac{}{y : !\mathsf{end} \vdash^0 \mathsf{close}\ y}\,[\textsc{t-close}]}{y : !\mathsf{ship}.!\mathsf{end} \vdash^0 y!\mathsf{ship}.\mathsf{close}\ y}\,[\textsc{t-label}]}{x : !\mathsf{end}, y : !\mathsf{ship}.!\mathsf{end} \vdash^0 \mathsf{wait}\ x.y!\mathsf{ship}.\mathsf{close}\ y}\,[\textsc{t-wait}]}{x : S, y : !\mathsf{ship}.!\mathsf{end} \vdash^0 x?\{\mathsf{add} : B\langle x, y\rangle, \mathsf{pay} : \mathsf{wait}\ x.y!\mathsf{ship}.\mathsf{close}\ y\}}\,[\textsc{t-label}]$$

for the definition of B. An analogous (but finite) derivation can be easily obtained for the body of process C and is omitted here for space limitations. Now we have

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{x : T \vdash^0 A\langle x\rangle}\,\text{[T-CALL]}}{x : S^\perp \vdash^1 \lceil x\rceil A\langle x\rangle}\,\text{[T-CAST]} \qquad \dfrac{\vdots}{x : S, y : !\mathsf{ship}.!\mathsf{end} \vdash^0 B\langle x, y\rangle}\,\text{[T-CALL]}}{y : !\mathsf{ship}.!\mathsf{end} \vdash^2 (x)(\lceil x\rceil A\langle x\rangle \mid B\langle x, y\rangle)}\,\text{[T-PAR]} \qquad \dfrac{\vdots}{y : ?\mathsf{ship}.?\mathsf{end} \vdash^0 C\langle y\rangle}}{\emptyset \vdash^3 (y)((x)(\lceil x\rceil A\langle x\rangle \mid B\langle x, y\rangle) \mid C\langle y\rangle)}$$

showing that *Main* too is well typed. In all cases, we have truncated the proof trees above the applications of [T-CALL]. Of course, for each judgment occurring in these proof trees, we also have to exhibit a finite proof tree possibly using [CO-LABEL] proving action boundedness. This can be easily achieved for the given process definitions, observing that none of A, B and C creates new sessions and that all of their typing derivations have a finite branch. ⌟

Example 6.2. In this example we demonstrate that well-typed processes may still create an unbounded number of (nested) sessions. To this aim, let us consider again the process C defined in Eq. (6). Notice that C is a choice whose left branch creates a new session and whose right branch does not. For this reason, we elect the right choice as the one that leads to termination, and therefore that determines the rank of the process. We derive

$$\dfrac{\dfrac{\dfrac{\vdots}{y : !\mathsf{end} \vdash^0 C\langle y\rangle}\,\text{[T-CALL]} \qquad \dfrac{\dfrac{}{x : !\mathsf{end} \vdash^0 \mathsf{close}\ x}\,\text{[T-CLOSE]}}{x : !\mathsf{end}, y : ?\mathsf{end} \vdash^0 \mathsf{wait}\ y.\mathsf{close}\ x}\,\text{[T-WAIT]}}{x : !\mathsf{end} \vdash^1 (y)(C\langle y\rangle \mid \mathsf{wait}\ y.\mathsf{close}\ x)}\,\text{[T-PAR]} \qquad \dfrac{}{x : !\mathsf{end} \vdash^0 \mathsf{close}\ x}}{x : !\mathsf{end} \vdash^0 (y)(C\langle y\rangle \mid \mathsf{wait}\ y.\mathsf{close}\ x) \oplus_2 \mathsf{close}\ x}$$

In a similar way, there exist well-typed processes that perform an unbounded number of casts but whose rank is finite. For example, it is easy to obtain a typing derivation for the following alternative definition of the process A discussed in Example 4.1:

$$A(x) \stackrel{\triangle}{=} x!\mathsf{add}.(\lceil x\rceil x!\mathsf{add}.A\langle x\rangle \oplus_2 \lceil x\rceil x!\mathsf{pay}.\mathsf{close}\ x)$$

Even though this process uses fair subtyping an unbounded number of times, the right branch of the choice has rank 1, which is all we need to conclude that the process has rank 1 overall. ⌟

Example 6.3 (random bit generator). Below we define a process $A(x)$ that implements the random bit generator protocol discussed in Examples 3.5 and 3.11 along with a client $B(x)$ that asks the service for random bits until it receives a $\mathsf{1}$:

$$A(x) \stackrel{\triangle}{=} x?\{\mathsf{more} : x!\{\mathsf{0} : A\langle x\rangle, \mathsf{1} : A\langle x\rangle\}, \mathsf{stop} : \mathsf{close}\ x\}$$
$$B(x) \stackrel{\triangle}{=} x!\mathsf{more}.x?\{\mathsf{0} : B\langle x\rangle, \mathsf{1} : x!\mathsf{stop}.\mathsf{wait}\ x.\mathsf{done}\}$$

These definitions are well typed under the global assignments $A : [S; 0]$ and $B : [U; 0]$ where S is defined as in Example 3.5 and U is defined as in Example 3.11. Notice that the termination of the parallel composition of A and B depends on a complex negotiation between A and B. Indeed, A terminates when it receives $\mathsf{0}$ from B and B terminates when it receives stop from A. This interaction pattern can only be modeled if both A and B are defined using general recursion. ⌟

Well-typed processes enjoy the expected properties, including typing preservation under structural pre-congruence and reduction. Most importantly, they fairly terminate:

Theorem 6.4. *If $\emptyset \vdash P$, then P is fairly terminating.*

The most intriguing aspect in the proof of Theorem 6.4 is that a closed, well-typed process admits a reduction sequence to done. Space constraints force us to relegate the details to the Appendix in the supplemental material, so here we only sketch the key elements of the proof, which is related to the *method of helpful directions* [Francez 1986]: we define a well-founded *measure* for (well-typed) processes and we prove that this measure decreases strictly as the result of "helpful" reductions. In our case, the measure of a (well-typed) process P is a lexicographically ordered pair (m, n) of natural numbers such that m is an upper bound to the number of sessions that P may need to create and of casts that P may need to perform *in the future* in order to terminate, whereas n is the cumulative rank of the sessions that P has created *in the past* and that are not terminated yet. A session terminates by [r-close]; a cast is performed when it is absorbed by the corresponding restriction, namely by [s-cast-new]. To distinguish between past and future of P we look at its structure: all sessions that occur unguarded in P have been created and are not terminated; all casts that occur in P are yet to be performed; all sessions that occur guarded in P have not been created yet. To compute the measure of a process, we introduce three refined typing rules to derive judgments of the form $\Gamma \vDash^{\mu} P$, stating that P is well typed in Γ and has measure μ:

$$\textsc{[mt-thread]}\ \frac{\Gamma \vdash^{n} P}{\Gamma \vDash^{(n,0)} P} \qquad \textsc{[mt-par]}\ \frac{\Gamma, x : S \vDash^{\mu} P \qquad \Delta, x : T \vDash^{\nu} P}{\Gamma, \Delta \vDash^{\mu+\nu+(0,\|S,T\|)} (x)(P \mid Q)}\ S \sim T \qquad \textsc{[mt-cast]}\ \frac{\Gamma, x : T \vDash^{\mu} P}{\Gamma, x : S \vDash^{\mu+(1,0)} \lceil x \rceil P}\ S \leqslant T$$

Rule [mt-thread] has *lower priority* than [mt-par] and [mt-cast], in the sense that it applies only to processes that are not a cast or a parallel composition. We call such processes *threads* and their measure is solely determined by their rank: every cast occurring in a thread is yet to be performed and every session occurring in a thread is yet to be created. Rule [mt-par] states that the measure of a parallel composition is the (pointwise) sum of the measures of the composed processes, taking into account the rank of the session x by which they are connected. Finally, [mt-cast] states that the measure of a cast is the same measure of the process in which the cast has effect, but with the first component increased by one to account for the fact that the cast is yet to be performed.

Note that, as a well-typed process reduces, its measure may vary arbitrarily. In particular, its measure *may increase* if the process chooses to create new sessions (*cf.* the left choice of process C in Eq. (6)) or if it picks a label that lengthens the shortest path leading to session termination (*cf.* Example 4.3). Nonetheless, the key lemma below assures that the measure of a well-typed process *may also decrease* following carefully chosen, "helpful" reductions.

Lemma 6.5. *If $\emptyset \vDash^{\mu} P$, then either $P \preccurlyeq \mathsf{done}$ or $P \Longrightarrow^{+} Q$ and $\emptyset \vDash^{\nu} Q$ for some Q and $\nu < \mu$.*

Remark 4. The rank of a non-deterministic choice $P \oplus Q$ can usually be chosen to be the minimum among those of the branches P and Q, so that the type system can handle processes like those in Example 6.2, which *may* create new sessions or perform casts but they need not do so in order to terminate. On the contrary, the rank of a label output $x!\{l_i : P_i\}_{i \in I}$ has to be an upper bound of that of all branches P_i. The motivation for such different ways of determining the rank of these process forms, despite both represent an *internal choice*, lies in the proof of Lemma 6.5. In $P \oplus Q$, both branches are typed in *exactly the same* typing context, meaning that the choice of one branch or the other has no substantial impact on the shortest paths that terminate the sessions used by P and Q. Thus, the "helpful" reduction can be solely driven by the rank of the chosen branch. In a label output $x!\{l_i : P_i\}_{i \in I}$ it could happen that all branches with minimum rank increase the length of the shortest path that leads to the termination of x. In this case, the choice of the "helpful" reduction must prioritize the termination of x, but then the rank of the whole process has to be an upper bound of that of the branches to be sure that the measure of the reduct decreases. ⌟

7 ON FAIR SUBTYPING AND HIGHER-ORDER SESSION TYPES

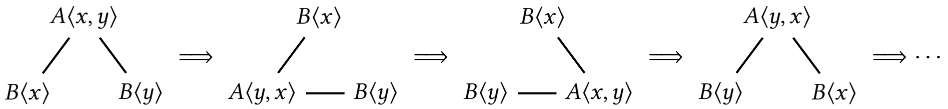
We have defined fair subtyping is such a way that higher-order session types are *invariant* with respect to the type of the channel being exchanged (*cf.* [F-CHANNEL]). This is a limitation compared to traditional presentations of unfair subtyping [Bernardi and Hennessy 2016; Castagna et al. 2009; Gay and Hole 2005], where the covariant/contravariant rules shown below are adopted:

$$\overset{[\text{U-CHANNEL-IN}]}{\dfrac{U \leqslant V \qquad S \leqslant T}{?U.S \leqslant ?V.T}} \qquad \overset{[\text{U-CHANNEL-OUT}]}{\dfrac{V \leqslant U \qquad S \leqslant T}{!U.S \leqslant !V.T}}$$

The problem of these rules is that a single application of fair subtyping allowing for co-/contra-variance of higher-order session types may have the same overall effect of infinitely many applications of fair subtyping on first-order session types and, as we have seen in Section 5.3, unbounded applications of fair subtyping may compromise fair termination. Below is an example that illustrates the problem. The example is not large *per se*, but it is a bit contrived because it has to involve two sessions (or else there would be no need for higher-order session types), it must be bounded (or else it could be ruled out by the action/session/cast boundedness requirements) and non-terminating:

$$(y)((x)(A\langle x,y\rangle \mid B\langle x\rangle) \mid B\langle y\rangle) \text{ where } \begin{array}{r} A(x,y) \overset{\triangle}{=} x!\mathsf{more}.x!y.B\langle x\rangle \\ B(x) \overset{\triangle}{=} x?\{\mathsf{more} : x?(y).A\langle y,x\rangle, \mathsf{stop} : \mathsf{wait}\ x.\mathsf{done}\} \end{array} \tag{10}$$

The process models a *master* $A\langle x,y\rangle$ connected with a *primary slave* $B\langle x\rangle$ and a *secondary slave* $B\langle y\rangle$ through the sessions x and y. The interaction among the three processes proceeds in rounds. At each round, the master may decide whether to continue or stop the interaction by sending either more or stop on the session x to the primary slave. If the master decides to continue the interaction (which it does deterministically), it also delegates y to the primary slave so that, at the next round, the roles of the three processes rotate: the master is downgraded to secondary slave, the primary slave is promoted to master, and the secondary slave becomes the primary one. Below is a graphical representation of the network topology modeled by the process and of its evolution:



It is clear that the process in Eq. (10) does not terminate since there is no close x to match the wait x. It is also relatively easy to infer the types of x and y from the structure of $A(x,y)$ and $B(x)$. In particular, if we call S_A and T_A the types of x and y in $A(x,y)$ and S_B the type of x in $B(x)$ we see that these types must satisfy the equations

$$S_A = !\mathsf{more}.!T_A.S_B \qquad S_B = ?\mathsf{more}.?S_A.T_A + ?\mathsf{stop}.?\mathsf{end} \qquad T_A = !\mathsf{more}.!T_A.S_B \oplus !\mathsf{stop}.!\mathsf{end}$$

Note that $T_A \leqslant S_A$ holds because T_A and S_A differ only for the topmost output. The validity of this relation is unquestionable as it relies on the definition of fair subtyping that we have given in Section 3.2, which is invariant with respect to higher-order session types. If fair subtyping allowed for covariance of higher-order inputs (*cf.* [U-CHANNEL-IN]), then $T_A^{\perp} \leqslant S_B$ would also hold and we would be able to establish that the process in Eq. (10) is well typed, provided that casts are placed appropriately. Below we show a version of the process in which we have annotated restrictions with the types $S \mid T$ of the two endpoints and casts with the target type of the channel affected by subtyping. The interested reader can find a full typing derivation in the supplemental material:

$$(y : T_A \mid T_A^\perp)((x : T_A \mid T_A^\perp)(\lceil x : S_A \rceil A\langle x, y\rangle \mid \lceil x : S_B \rceil B\langle x\rangle) \mid \lceil y : S_B \rceil B\langle y\rangle) \tag{11}$$

Dually, if fair subtyping allowed for contravariance of higher-order outputs (*cf.* [u-channel-out]) then $S_B^\perp \leqslant T_A$ would also hold (along with $S_B^\perp \leqslant S_A$ by transitivity of $\leqslant$) and we would be able to establish that the above process is well typed according to the type annotations shown below:

$$(y : S_B^\perp \mid S_B)((x : S_B^\perp \mid S_B)(\lceil x : S_A \rceil \lceil y : T_A \rceil A\langle x, y\rangle \mid B\langle x\rangle) \mid B\langle y\rangle)$$

By restricting fair subtyping of higher-order session types to invariant inputs and outputs, the only chance we have to build a typing derivation for the process in Eq. (10) is by casting y each time it is delegated, either before it is sent or after it is received, for example as in

$$\begin{aligned} A(x : U_A, y : V_A) &\triangleq x!\mathsf{more}.\lceil y : U_A \rceil x!y.B\langle x\rangle \\ B(x : U_B) &\triangleq x?\{\mathsf{more} : x?(y : U_A).A\langle y, x\rangle, \mathsf{stop} : \mathsf{wait}\ x.\mathsf{done}\} \end{aligned}$$

where

$$U_A = !\mathsf{more}.!U_A.U_B \qquad U_B = ?\mathsf{more}.?U_A.V_A + ?\mathsf{stop}.?\mathsf{end} \qquad V_A = !\mathsf{more}.!U_A.U_B \oplus !\mathsf{stop}.!\mathsf{end}$$

Note that $\lceil y : U_A \rceil$ is a "first-order" cast, in the sense that the relation $V_A \leqslant U_A$ holds for fair subtyping as defined in Section 3.2 without using [u-channel-in] or [u-channel-out], but the cast is now placed in a region within the definition of A that prevents finding a finite rank for A.

8 RELATED WORK

Deadlock freedom. The absence of deadlocks is a fundamental requirement for the proof of Theorem 6.4. In this work we ensure deadlock freedom by adopting the formulation of rule [t-par] inspired to linear logic, as in the works of Caires et al. [2016], Wadler [2014] and Lindley and Morris [2016]. This technique is simple and effective, but limits its applicability to tree-shaped network topologies. Another line of works makes use of a richer type structure associating input/output actions in types with "levels" or "priorities" so as to prevent circular dependencies. This approach has been pioneered by Kobayashi [2002, 2006] in the π-calculus and then ported to session-based calculi by Padovani [2014] and Dardha and Gay [2018]. Dardha and Pérez [2015] compare the two approaches. Balzer et al. [2019] relax the approach based on linear logic so as to allow controlled forms of channel sharing but still preserving deadlock freedom. Kobayashi and Laneve [2017] describe another approach based on behavioral types associated to processes (as opposed to channels) that is capable of addressing cyclic network topologies. Interestingly, the proof of Theorem 6.4 is independent of the technique used for ensuring deadlock freedom, hence our type system can be combined with other approaches addressing more complex network topologies.

Lock freedom. Type systems ensuring this liveness property have been studied by Kobayashi [2002] and Kobayashi and Sangiorgi [2010] for the π-calculus, by Padovani [2014] for the linear π-calculus and calculi based on binary sessions, and by Padovani et al. [2014] for the conversation calculus, a calculus of multiparty sessions. A common trait of the type systems in these works, which is also the main distinguishing aspect with our own, is that they use explicit type annotations to capture the dependencies between different actions. In particular, the annotations in the works of Kobayashi [2002], Kobayashi and Sangiorgi [2010] and Padovani [2014] conservatively estimate the amount of reductions that are necessary in order to perform a given input/output action. If such amounts cannot be statically determined or if they are not finite, as is the case of the action $y!\mathsf{ship}$ in Eq. (1) and in general in all (recursive) processes where an action may be preceded by an arbitrary number of other actions, either the process is declared ill typed or the action is not guaranteed to be eventually performed. As a matter of fact, determining the amount of reductions that are necessary to unlock and perform a certain action is crucially important also in our work, but this information stays buried in the soundness proof of the type system as part of the measure and does not surface in types. The use of unannotated types allows our type system to address a broad family of processes – of which Eq. (1) is just one representative member – for which the existing type-based techniques are unable to guarantee lock freedom.

Termination by typing. Session type systems strictly based on linear logic [Caires et al. 2016; Wadler 2014] ensure session termination, but they do not allow for the specification of recursive protocols. Yoshida et al. [2004] present a type system for the π-calculus based on linear channel types that ensures strong normalization, hence lock freedom. Their type system applies also to binary sessions given the tight relationship between binary sessions and linear channel types [Dardha et al. 2017]. Deng and Sangiorgi [2006] and Demangeon et al. [2009] study type-based approaches for enforcing the termination of (mobile) processes with replication, namely the property that every reduction sequence is guaranteed to be finite. These type systems could be applied also to session-based calculi. Kobayashi and Sangiorgi [2010] show that the combination of deadlock freedom and termination results in lock freedom. Lindley and Morris [2016] study languages of binary sessions for which typing guarantees strong normalization, implying that well-typed programs are also lock free. Their type system is based on an extension of linear logic with fixed points [Baelde 2012; Doumane 2017] so that recursive types come into dual forms corresponding to least and greatest fixed points. Programs like those discussed in Examples 4.1 and 6.3, which fairly terminate but admit infinite computations, would be ill typed if modeled in their languages.

Multiparty sessions. Multiparty sessions [Honda et al. 2008, 2016] are a generalization of binary sessions to an arbitrary – sometimes variable – number of participants. In recent advancements to the theory of multiparty session types, Scalas and Yoshida [2019] show how to specify and enforce a family of safety and liveness properties of multiparty sessions as formulas in the modal μ-calculus. In particular, they define three predicates *live*, live^{+} and live^{++} closely related to lock freedom: while live^{+} and live^{++} specify "bounded" forms of lock freedom such that every pending communication is guaranteed to be performed in a finite number of steps, *live* specifies the eventual completion of *any* pending communication action. If we were to reason about process B in Eq. (1) using these predicates, we could say that B satisfies *live* but not live^{+} or live^{++}. Their type system is able to enforce live^{+} and live^{++}, but not *live* because they use an "unfair" subtyping relation that does not preserve (all) liveness properties. More recently, van Glabbeek et al. [2021] have proposed a type system for multiparty sessions that ensures lock freedom and that is not only sound (under suitable conditions) but also *complete*, in the sense that all lock-free sessions are well typed. Such a strong result is possible because the notion of lock freedom they consider is based on *justness* [van Glabbeek 2019; van Glabbeek and Höfner 2019], a "minimal fairness assumption that guarantees only that concurrent transitions cannot prevent each other from happening" [van Glabbeek et al. 2021]. We assume strong fairness, which is at the opposite end of the spectrum compared to justness. The stronger the fairness assumption, the larger the set of "unfair" executions that are ruled out, the larger the family of lock-free processes. For example, the process modeled in Eq. (1) is not lock free under justness. Another difference between our work and the one of van Glabbeek et al. [2021] is that they only consider single, first-order session systems (even though sessions can be multiparty). This simplification avoids all the issues arising from session creation and delegation (Sections 5 and 7) and from the interleaving of blocking actions from different sessions, which occurs in Eq. (1) and is one of the motivations for focusing on fair termination instead of lock freedom.

Fair subtyping. The original "unfair" subtyping relation for session types of Gay and Hole [2005] preserves communication safety but not necessarily (fair) termination. Analogous relations have been studied by Castagna et al. [2009] and Bernardi and Hennessy [2016]. Fair subtyping is a liveness-preserving refinement of unfair subtyping. Bravetti and Zavattaro [2009] study a *strong subcontract relation* for Web service contracts that shares the same liveness-preserving features of fair subtyping. Indeed, the notion of *strong contract composition* of Bravetti and Zavattaro [2009] that is preserved by strong subcontract is a first-order version of compatibility (Definition 3.7) for multiparty service compositions. Both strong subcontract and fair subtyping are closely related to divergence-insensitive refinements for processes called *fair testing* [Natarajan and Cleaveland 1995] and *should testing* [Rensink and Vogler 2007]. Inference systems for fair subtyping have been studied by Padovani [2013, 2016] and Ciccone and Padovani [2021b]. The generalized inference system for fair subtyping in Table 1 is a higher-order adaptation of the one presented by Ciccone and Padovani [2021b]. Bravetti et al. [2021] study a fair subtyping relation for *asynchronous* session types (in which output actions are non-blocking), showing that it is undecidable and proposing decidable approximations of it. It appears feasible that asynchronous fair subtyping could be used in our type system as a drop-in replacement of fair subtyping to further relax the correspondence between the structure of protocols and processes.

Fair termination. Grumberg et al. [1984] describe proof methods for proving the fair termination of communicating processes specified as CSP terms, hence for a process model comparable to that of our (first-order) session types. They point out how fair termination enables the characterization of behaviors like that of Eq. (1), where processes engaged in a finite but unbounded number of communications may (fairly) terminate. Cook et al. [2007] check liveness properties of C programs by means of a reduction to a fair termination problem, whereas Ganty et al. [2009] give a procedure for deciding fair termination of event-driven programs. Tassarotti et al. [2017] present an extension of separation logic to prove the correctness of an implementation of binary sessions. Interestingly, they also rely on a liveness-preserving refinement, although in their case the refinement is used to compare specification and implementation of the same program rather than protocols. The work of Cacciagrano et al. [2006, 2009] shows that a notion of fair process termination for the π-calculus formulated in terms of fair runs may differ from the one we have adopted in Definition 4.2. Given that the counterexample they use relies crucially on replication and shared channels, it might be interesting to investigate whether it is possible to restore the equivalence of the two formulations for our (typed) language, which is based on recursion and linearized channels.

9 CONCLUDING REMARKS

We have presented the first type system ensuring the fair termination of binary sessions (Theorem 6.4). The type system applies to a calculus that supports general recursion, session interleaving, session delegation and dynamic session creation, thus targeting a large family of processes with unboundedly many states and communicating in networks with variable topology. Fair termination is coarser than strong normalization or plain termination, since it does not rule out infinite interactions, and it is stricter than lock freedom, so it entails the eventual completion of *any* pending communication, including those blocked by an unbounded number of actions in possibly different (Eq. (1)) or yet-to-be-created (Eq. (6)) sessions. These interaction patterns fall outside the scope of existing type systems for lock freedom. Clearly, there exist lock-free processes that are not fairly terminating (Section 1), although the possibility that interactions may eventually terminate seems to be a natural requirement for sessions. For all of these reasons, we consider our type system a substantial leap forward in the study of type-based techniques for the enforcement of liveness properties of sessions.

A key element of the type system is fair subtyping, a liveness-preserving subtyping relation for session types that so far has only been studied theoretically but never applied in a type system. We have shown that fair subtyping must be used with care (Section 5.3). Indeed, our type system accounts for all the usages of fair subtyping, making sure that the overall effort required to terminate a process, along with all the sessions it has created, remains finite. We have also uncovered an unforeseen interaction between fair subtyping and higher-order sessions that may invalidate the liveness-preserving features of fair subtyping (Section 7). One simple way to avoid this issue is by imposing the invariance of higher-order session types (Section 3.2). While this is a restriction compared to the "unfair" subtyping relations for session types, we think it is unlikely to have a profound impact: higher-order channels in distributed programs are far less ubiquitous than, say, higher-order functions in functional ones, and co-/contra-variance of higher-order session types can be partly recovered by means of explicit casts, at least in those places where casts can be used.

We could not detail any algorithmic aspect of the type system in the main body of the paper because of space limits. In particular, the formulation of the typing rules in Table 3, in which the rank annotation is essentially guessed, allows for simple presentation and soundness proof of the type system, but does not say how to determine the rank annotation nor when it exists. We have defined an algorithm for computing the *minimum rank annotation* that is needed to type a process as well as necessary and sufficient conditions for its existence that are solely based on the structure of processes. This makes it possible to provide an equivalent and fully compositional set of typing rules with which we have proved that type checking is decidable if one assumes that bindings and casts are decorated with explicit type annotations. A proof-of-concept Haskell implementation of this type checking algorithm is available [Ciccone and Padovani 2021a]. We have also proved that the location of casts can be inferred automatically and have defined a sound and complete set of *co-contextual typing rules* [Erdweg et al. 2015] to reconstruct session types from unannotated processes. Details about these aspects can be found in the supplemental material associated with this paper.

Considering that fair subtyping for binary and multiparty session types share essentially the same characterization [Bravetti and Zavattaro 2009; Bugliesi et al. 2009; Padovani 2016], we expect the type system to scale smoothly to multiparty sessions [Honda et al. 2008, 2016]. It remains to be established if fair subtyping can also be applied in the general framework proposed by Scalas and Yoshida [2019], which is parametric in the liveness property one is interested to enforce, or if different versions of fair subtyping are necessary to enforce different liveness properties. As suggested by one reviewer, another interesting direction for future investigations is the application of the type system to higher-order session calculi such as GV [Gay and Vasconcelos 2010; Wadler 2014] and μGV [Lindley and Morris 2016]. In particular, establishing the rank of terms in a higher-order calculus appears to be less straightforward than in the process calculus that we consider.

ACKNOWLEDGMENTS

We are grateful to the anonymous POPL reviewers for their thoughtful comments and suggestions. A special mention goes to the members of the Artifact Evaluation Committee who evaluated our artifact. The extent and quality of their feedback has been invaluable to improve the usability and the documentation of the artifact.

REFERENCES

Peter Aczel. 1977. An Introduction to Inductive Definitions. In *Handbook of Mathematical Logic*, Jon Barwise (Ed.). Studies in Logic and the Foundations of Mathematics, Vol. 90. Elsevier, 739 – 782. https://doi.org/10.1016/S0049-237X(08)71120-0

Davide Ancona, Francesco Dagnino, and Elena Zucca. 2017. Generalizing Inference Systems by Coaxioms. In *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 (Lecture Notes in Computer Science, Vol. 10201), Hongseok Yang (Ed.). Springer, 29–55. https://doi.org/10.1007/978-3-662-54434-1_2

Krzysztof R. Apt, Nissim Francez, and Shmuel Katz. 1987. Appraising Fairness in Languages for Distributed Programming. In *Conference Record of the Fourteenth Annual ACM Symposium on Principles of Programming Languages, Munich, Germany, January 21-23, 1987*. ACM Press, 189–198. https://doi.org/10.1145/41625.41642

David Baelde. 2012. Least and Greatest Fixed Points in Linear Logic. *ACM Trans. Comput. Log.* 13, 1 (2012), 2:1–2:44. https://doi.org/10.1145/2071368.2071370

Stephanie Balzer, Bernardo Toninho, and Frank Pfenning. 2019. Manifest Deadlock-Freedom for Shared Session Types. In *Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11423)*, Luís Caires (Ed.). Springer, 611–639. https://doi.org/10.1007/978-3-030-17184-1_22

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

Philippe Bidinger and Adriana B. Compagnoni. 2009. Pict correctness revisited. *Theor. Comput. Sci.* 410, 2-3 (2009), 114–127. https://doi.org/10.1016/j.tcs.2008.09.014

Mario Bravetti, Julien Lange, and Gianluigi Zavattaro. 2021. Fair Refinement for Asynchronous Session Types. In *Foundations of Software Science and Computation Structures - 24th International Conference, FOSSACS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12650)*, Stefan Kiefer and Christine Tasson (Eds.). Springer, 144–163. https://doi.org/10.1007/978-3-030-71995-1_8

Mario Bravetti and Gianluigi Zavattaro. 2009. A theory of contracts for strong service compliance. *Math. Struct. Comput. Sci.* 19, 3 (2009), 601–638. https://doi.org/10.1017/S0960129509007658

Michele Bugliesi, Damiano Macedonio, Luca Pino, and Sabina Rossi. 2009. Compliance Preorders for Web Services. In *Web Services and Formal Methods, 6th International Workshop, WS-FM 2009, Bologna, Italy, September 4-5, 2009, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 6194)*, Cosimo Laneve and Jianwen Su (Eds.). Springer, 76–91. https://doi.org/10.1007/978-3-642-14458-5_5

Diletta Cacciagrano, Flavio Corradini, and Catuscia Palamidessi. 2006. Fair Pi. In *Proceedings of the 13th International Workshop on Expressiveness in Concurrency, EXPRESS 2006, Bonn, Germany, August 26, 2006 (Electronic Notes in Theoretical Computer Science, Vol. 175)*, Roberto M. Amadio and Iain Phillips (Eds.). Elsevier, 3–26. Issue 3. https://doi.org/10.1016/j.entcs.2006.10.051

Diletta Cacciagrano, Flavio Corradini, and Catuscia Palamidessi. 2009. Explicit fairness in testing semantics. *Log. Methods Comput. Sci.* 5, 2 (2009). https://doi.org/10.2168/LMCS-5(2:15)2009

Luís Caires, Frank Pfenning, and Bernardo Toninho. 2016. Linear logic propositions as session types. *Math. Struct. Comput. Sci.* 26, 3 (2016), 367–423. https://doi.org/10.1017/S0960129514000218

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

Luca Ciccone and Luca Padovani. 2021a. *FairCheck*. Retrieved October 1, 2021 from https://github.com/boystrange/FairCheck

Luca Ciccone and Luca Padovani. 2021b. Inference Systems with Corules for Fair Subtyping and Liveness Properties of Binary Session Types. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP'21) (LIPIcs, Vol. 198)*, Nikhil Bansal, Emanuela Merelli, and James Worrell (Eds.). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 125:1–125:16. https://doi.org/10.4230/LIPIcs.ICALP.2021.125

Byron Cook, Alexey Gotsman, Andreas Podelski, Andrey Rybalchenko, and Moshe Y. Vardi. 2007. Proving that programs eventually do something good. In *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007*, Martin Hofmann and Matthias Felleisen (Eds.). ACM, 265–276. https://doi.org/10.1145/1190216.1190257

Bruno Courcelle. 1983. Fundamental Properties of Infinite Trees. *Theor. Comput. Sci.* 25 (1983), 95–169. https://doi.org/10.1016/0304-3975(83)90059-2

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

Ornela Dardha and Simon J. Gay. 2018. A New Linear Logic for Deadlock-Free Session-Typed Processes. In *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 (Lecture Notes in Computer Science, Vol. 10803)*, Christel Baier and Ugo Dal Lago (Eds.). Springer, 91–109. https://doi.org/10.1007/978-3-319-89366-2_5

Ornela Dardha, Elena Giachino, and Davide Sangiorgi. 2017. Session types revisited. *Inf. Comput.* 256 (2017), 253–286. https://doi.org/10.1016/j.ic.2017.06.002

Ornela Dardha and Jorge A. Pérez. 2015. Comparing Deadlock-Free Session Typed Processes. In *Proceedings of the Combined 22th International Workshop on Expressiveness in Concurrency and 12th Workshop on Structural Operational Semantics, EXPRESS/SOS 2015, Madrid, Spain, 31st August 2015 (EPTCS, Vol. 190)*, Silvia Crafa and Daniel Gebler (Eds.). 1–15. https://doi.org/10.4204/EPTCS.190.1

Romain Demangeon, Daniel Hirschkoff, and Davide Sangiorgi. 2009. Mobile Processes and Termination. In *Semantics and Algebraic Specification, Essays Dedicated to Peter D. Mosses on the Occasion of His 60th Birthday (Lecture Notes in Computer Science, Vol. 5700)*, Jens Palsberg (Ed.). Springer, 250–273. https://doi.org/10.1007/978-3-642-04164-8_13

Yuxin Deng and Davide Sangiorgi. 2006. Ensuring termination by typability. *Inf. Comput.* 204, 7 (2006), 1045–1082. https://doi.org/10.1016/j.ic.2006.03.002

Amina Doumane. 2017. *On the infinitary proof theory of logics with fixed points. (Théorie de la démonstration infinitaire pour les logiques à points fixes)*. Ph. D. Dissertation. Paris Diderot University, France. https://tel.archives-ouvertes.fr/tel-01676953

Sebastian Erdweg, Oliver Bracevac, Edlira Kuci, Matthias Krebs, and Mira Mezini. 2015. A co-contextual formulation of type rules and its application to incremental type checking. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2015, part of SPLASH 2015, Pittsburgh, PA, USA, October 25-30, 2015*, Jonathan Aldrich and Patrick Eugster (Eds.). ACM, 880–897. https://doi.org/10.1145/2814270.2814277

Nissim Francez. 1986. *Fairness*. Springer. https://doi.org/10.1007/978-1-4612-4886-6

Pierre Ganty, Rupak Majumdar, and Andrey Rybalchenko. 2009. Verifying liveness for asynchronous programs. In *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, Zhong Shao and Benjamin C. Pierce (Eds.). ACM, 102–113. https://doi.org/10.1145/1480881.1480895

Simon J. Gay. 2016. Subtyping Supports Safe Session Substitution. In *A List of Successes That Can Change the World - Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday (Lecture Notes in Computer Science, Vol. 9600)*, Sam Lindley, Conor McBride, Philip W. Trinder, and Donald Sannella (Eds.). Springer, 95–108. https://doi.org/10.1007/978-3-319-30936-1_5

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

Simon J. Gay and Vasco Thudichum Vasconcelos. 2010. Linear type theory for asynchronous session types. *J. Funct. Program.* 20, 1 (2010), 19–50. https://doi.org/10.1017/S0956796809990268

Orna Grumberg, Nissim Francez, and Shmuel Katz. 1984. Fair Termination of Communicating Processes. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing* (Vancouver, British Columbia, Canada) *(PODC '84)*. Association for Computing Machinery, New York, NY, USA, 254–265. https://doi.org/10.1145/800222.806752

Kohei Honda. 1993. Types for Dyadic Interaction. In *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings (Lecture Notes in Computer Science, Vol. 715)*, Eike Best (Ed.). Springer, 509–523. https://doi.org/10.1007/3-540-57208-2_35

Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. 1998. Language Primitives and Type Discipline for Structured Communication-Based Programming. In *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 (Lecture Notes in Computer Science, Vol. 1381)*, Chris Hankin (Ed.). Springer, 122–138. https://doi.org/10.1007/BFb0053567

Kohei Honda, Nobuko Yoshida, and Marco Carbone. 2008. Multiparty asynchronous session types. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, George C. Necula and Philip Wadler (Eds.). ACM, 273–284. https://doi.org/10.1145/1328438.1328472

Kohei Honda, Nobuko Yoshida, and Marco Carbone. 2016. Multiparty Asynchronous Session Types. *J. ACM* 63, 1 (2016), 9:1–9:67. https://doi.org/10.1145/2827695

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. 2016. Foundations of Session Types and Behavioural Contracts. *ACM Comput. Surv.* 49, 1 (2016), 3:1–3:36. https://doi.org/10.1145/2873052

Naoki Kobayashi. 2002. A Type System for Lock-Free Processes. *Inf. Comput.* 177, 2 (2002), 122–159. https://doi.org/10.1006/inco.2002.3171

Naoki Kobayashi. 2006. A New Type System for Deadlock-Free Processes. In *CONCUR 2006 - Concurrency Theory, 17th International Conference, CONCUR 2006, Bonn, Germany, August 27-30, 2006, Proceedings (Lecture Notes in Computer Science, Vol. 4137)*, Christel Baier and Holger Hermanns (Eds.). Springer, 233–247. https://doi.org/10.1007/11817949_16

Naoki Kobayashi and Cosimo Laneve. 2017. Deadlock analysis of unbounded process networks. *Inf. Comput.* 252 (2017), 48–70. https://doi.org/10.1016/j.ic.2016.03.004

Naoki Kobayashi and Davide Sangiorgi. 2010. A hybrid type system for lock-freedom of mobile processes. *ACM Trans. Program. Lang. Syst.* 32, 5 (2010), 16:1–16:49. https://doi.org/10.1145/1745312.1745313

M.Z. Kwiatkowska. 1989. Survey of fairness notions. *Information and Software Technology* 31, 7 (1989), 371–386. https://doi.org/10.1016/0950-5849(89)90159-6

Leslie Lamport. 2000. Fairness and hyperfairness. *Distributed Comput.* 13, 4 (2000), 239–245. https://doi.org/10.1007/PL00008921

Sam Lindley and J. Garrett Morris. 2016. Talking bananas: structural recursion for session types. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016*, Jacques Garrigue, Gabriele Keller, and Eijiro Sumii (Eds.). ACM, 434–447. https://doi.org/10.1145/2951913.2951921

Barbara Liskov and Jeannette M. Wing. 1994. A Behavioral Notion of Subtyping. *ACM Trans. Program. Lang. Syst.* 16, 6 (1994), 1811–1841. https://doi.org/10.1145/197320.197383

V. Natarajan and Rance Cleaveland. 1995. Divergence and Fair Testing. In *Automata, Languages and Programming, 22nd International Colloquium, ICALP95, Szeged, Hungary, July 10-14, 1995, Proceedings (Lecture Notes in Computer Science, Vol. 944)*, Zoltán Fülöp and Ferenc Gécseg (Eds.). Springer, 648–659. https://doi.org/10.1007/3-540-60084-1_112

Susan S. Owicki and Leslie Lamport. 1982. Proving Liveness Properties of Concurrent Programs. *ACM Trans. Program. Lang. Syst.* 4, 3 (1982), 455–495. https://doi.org/10.1145/357172.357178

Luca Padovani. 2013. Fair Subtyping for Open Session Types. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 7966)*, Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg (Eds.). Springer, 373–384. https://doi.org/10.1007/978-3-642-39212-2_34

Luca Padovani. 2014. Deadlock and lock freedom in the linear π-calculus. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, Thomas A. Henzinger and Dale Miller (Eds.). ACM, 72:1–72:10. https://doi.org/10.1145/2603088.2603116

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

Luca Padovani, Vasco Thudichum Vasconcelos, and Hugo Torres Vieira. 2014. Typing Liveness in Multiparty Communicating Systems. In *Coordination Models and Languages - 16th IFIP WG 6.1 International Conference, COORDINATION 2014, Held as Part of the 9th International Federated Conferences on Distributed Computing Techniques, DisCoTec 2014, Berlin, Germany, June 3-5, 2014, Proceedings (Lecture Notes in Computer Science, Vol. 8459)*, eva Kühn and Rosario Pugliese (Eds.). Springer, 147–162. https://doi.org/10.1007/978-3-662-43376-8_10

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

Alceste Scalas and Nobuko Yoshida. 2019. Less is more: multiparty session types revisited. *Proc. ACM Program. Lang.* 3, POPL (2019), 30:1–30:29. https://doi.org/10.1145/3290343

Joseph Tassarotti, Ralf Jung, and Robert Harper. 2017. A Higher-Order Logic for Concurrent Termination-Preserving Refinement. In *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 (Lecture Notes in Computer Science, Vol. 10201)*, Hongseok Yang (Ed.). Springer, 909–936. https://doi.org/10.1007/978-3-662-54434-1_34

Rob van Glabbeek. 2019. Justness - A Completeness Criterion for Capturing Liveness Properties (Extended Abstract). In *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11425)*, Mikolaj Bojanczyk and Alex Simpson (Eds.). Springer, 505–522. https://doi.org/10.1007/978-3-030-17127-8_29

Rob van Glabbeek and Peter Höfner. 2019. Progress, Justness, and Fairness. *ACM Comput. Surv.* 52, 4 (2019), 69:1–69:38. https://doi.org/10.1145/3329125

Rob van Glabbeek, Peter Höfner, and Ross Horne. 2021. Assuming Just Enough Fairness to make Session Types Complete for Lock-freedom. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*. IEEE, 1–13. https://doi.org/10.1109/LICS52264.2021.9470531

Philip Wadler. 2014. Propositions as sessions. *J. Funct. Program.* 24, 2-3 (2014), 384–418. https://doi.org/10.1017/S095679681400001X

Nobuko Yoshida, Martin Berger, and Kohei Honda. 2004. Strong normalisation in the pi -calculus. *Inf. Comput.* 191, 2 (2004), 145–202. https://doi.org/10.1016/j.ic.2003.08.004

A SUPPLEMENT TO SECTION 3

A.1 Supplement to Section 3.2

For proving some of the results that follow it is convenient to provide a simpler characterization of $\leqslant_{\mathsf{ind}}$ that is solely based on [F-CONVERGE].

Lemma A.1. *Let $S \leqslant_{\mathsf{coind}} T$ and $\downarrow$ be the relation inductively defined by the following rule:*

$$\frac{\forall \varphi \in \mathsf{paths}(S) \setminus \mathsf{paths}(T) : \exists \psi \leq \varphi, l \in \mathcal{L} : S(\psi!l) \downarrow T(\psi!l)}{S \downarrow T}$$

Then $S \leqslant_{\mathsf{ind}} T$ if and only if $S \downarrow T$.

Proof. The "if" part is trivial since the sole rule defining $\downarrow$ is the same as [F-CONVERGE]. We prove the "only if" part by induction on the derivation of $S \leqslant_{\mathsf{ind}} T$ and by cases on the last rule applied.

Case [F-END]. Then $S = T = p\,\mathsf{end}$ for some p. We conclude $S \downarrow T$ observing that $\mathsf{paths}(S) \setminus \mathsf{paths}(T) = \emptyset$.

Case [F-CHANNEL]. Then $S = pU.S'$ and $T = pU.T'$ and $S' \leqslant_{\mathsf{ind}} T'$. Using the induction hypothesis we deduce that $S' \downarrow T'$. We conclude $S \downarrow T$ observing that $pU\varphi \in \mathsf{paths}(S) \setminus \mathsf{paths}(T)$ if and only if $\varphi \in \mathsf{paths}(S') \setminus \mathsf{paths}(T')$ and that $S(pU\varphi!l) = S'(\varphi!l)$ and $T(pU\varphi!l) = T'(\varphi!l)$ whenever $\varphi!l \in \mathsf{paths}(S') \cap \mathsf{paths}(T')$ by Definition 3.2.

Case [F-LABEL-IN]. Then $S = ?\{l_i : S_i\}_{i\in I}$ and $T = ?\{l_j : T_j\}_{j\in J}$ and $I \subseteq J$ and $S_i \leqslant_{\mathsf{ind}} T_i$ for every $i \in I$. Using the induction hypothesis we deduce that $S_i \downarrow T_i$ for every $i \in I$. We conclude $S \downarrow T$ observing that $\varphi \in \mathsf{paths}(S) \setminus \mathsf{paths}(T)$ implies $\varphi = ?l_k\psi$ for some $k \in I$ and $\psi \in \mathsf{paths}(S_k) \setminus \mathsf{paths}(T_k)$.

Case [F-LABEL-OUT]. Then $S = !\{l_i : S_i\}_{i\in I}$ and $T = !\{l_j : T_j\}_{j\in J}$ and $J \subseteq I$ and $S_j \leqslant_{\mathsf{ind}} T_j$ for every $j \in J$. Using the induction hypothesis we deduce that $S_j \downarrow T_j$ for every $j \in J$. In order to conclude $S \downarrow T$ we have to show that, for every $\varphi \in \mathsf{paths}(S) \setminus \mathsf{paths}(T)$, we are able to find $\psi \leq \varphi$ and $l \in \mathcal{L}$ such that $S(\psi!l) \downarrow T(\psi!l)$. We distinguish two sub-cases:

- Sub-case $\varphi = !l_j\varphi'$ where $\varphi' \in \mathsf{paths}(S_j) \setminus \mathsf{paths}(T_j)$ for some $j \in J$. From $S_j \downarrow T_j$ we deduce that there exist $\psi' \leq \varphi'$ and $l \in \mathcal{L}$ such that $S_j(\psi'!l) \downarrow T_j(\psi'!l)$. We conclude by taking $\psi \overset{\text{def}}{=} !l_j\psi'$ observing that $S(\psi!l) = S_j(\psi'!l)$ and $T(\psi!l) = T_j(\psi'!l)$.
- Sub-case $\varphi = !l_i\varphi'$ where $\varphi' \in \mathsf{paths}(S_i)$ for some $i \in I \setminus J$. We conclude by taking $\psi \overset{\text{def}}{=} \varepsilon$ and $l \overset{\text{def}}{=} l_j$ for some $j \in J$ and observing that $S(\psi!l) = S_j$ and $T(\psi!l) = T_j$.

Case [F-CONVERGE]. Then, for every $\varphi \in \mathsf{paths}(S) \setminus \mathsf{paths}(T)$, there exist $\psi \leq \varphi$ and $l \in \mathcal{L}$ such that $S(\psi!l) \leqslant_{\mathsf{ind}} T(\psi!l)$. We conclude immediately using the induction hypothesis to deduce that for each such ψ and l we have $S(\psi!l) \downarrow T(\psi!l)$. □

A.2 Supplement to Section 3.3

A.2.1 Generalized compatibility. We define a more general notion of compatibility, of which Definition 3.7 is a particular case, that is useful to prove the transitivity of fair subtyping. To this aim, we also give a more general notion of *liveness condition* that fair subtyping is meant to preserve.

Definition A.2 (liveness condition). A *liveness condition* is a binary relation $\mathcal{L}$ between session types. Hereafter, we assume that $\mathcal{L}$ stands for one of the following liveness conditions:

- $\mathcal{L}_1 = \{(!\mathsf{end}, ?\mathsf{end}), (?\mathsf{end}, !\mathsf{end})\}$, the one we adopted in the main body of the paper, or
- $\mathcal{L}_2 = \{(!\mathsf{end}, S) \mid S \text{ is a session type}\}$, according to which we only care about the successful termination of the "client", that is the session type U on the left hand side of a term $U \mid S$.

Proposition A.3. *When $\mathcal{L} \in \{\mathcal{L}_1, \mathcal{L}_2\}$, the following properties hold: (1) $U \mathrel{\mathcal{L}} S$ implies $U = p\,\mathsf{end}$ for some p; (2) if $S \leqslant T$, then $U \mathrel{\mathcal{L}} S$ if and only if $U \mathrel{\mathcal{L}} T$.*

Definition A.4 ($\mathcal{L}$-compatibility). We say that S is *$\mathcal{L}$-compatible* with T, notation $S \sim_{\mathcal{L}} T$, if $S \mid T \Longrightarrow S' \mid T'$ implies $S' \mid T' \Longrightarrow S'' \mid T''$ for some S'' and T'' such that $S'' \mathrel{\mathcal{L}} T''$.

Definition A.5 ($\mathcal{L}$-viability). We say that S is $\mathcal{L}$-viable if, for every $\varphi \in \mathsf{paths}(S)$, there exists U such that $U \sim_{\mathcal{L}} S(\varphi)$. If S is $\mathcal{L}$-viable, we write $\mathcal{L}^{\perp}(S)$ for a representative session type U such that $U \sim_{\mathcal{L}} S$. Note that every session type is $\mathcal{L}_2$-viable and that every bounded session type is $\mathcal{L}_1$-viable. In particular, we can take $\mathcal{L}_1^{\perp}(S) \stackrel{\text{def}}{=} S^{\perp}$ and $\mathcal{L}_2^{\perp}(S) \stackrel{\text{def}}{=} {!\mathsf{end}}$.

Proposition A.6. *If S is $\mathcal{L}$-viable, then the following properties hold: (1) $S = pV.S'$ implies that S' is $\mathcal{L}$-viable; (2) $S = p\{l_i : S_i\}_{i \in I}$ implies that S_i is $\mathcal{L}$-viable for every $i \in I$.*

Proof. Immediate from the definition of $\mathcal{L}_1$ and $\mathcal{L}_2$. □

A.2.2 Soundness of fair subtyping. We begin by showing that both $\mathcal{L}$-compatibility and fair subtyping are preserved by reductions.

Lemma A.7. *If $U \sim_{\mathcal{L}} S$ and $U \mid S \Longrightarrow U' \mid S'$, then $U' \sim_{\mathcal{L}} S'$.*

Proof. Consider a reduction $U' \mid S' \Longrightarrow U'' \mid S''$. Then $U \mid S \Longrightarrow U'' \mid S''$ by transitivity of $\Longrightarrow$ and we conclude $U'' \mid S'' \Longrightarrow U''' \mid S'''$ for some U''' and S''' such that $U''' \mathrel{\mathcal{L}} S'''$ from the hypothesis $U \sim_{\mathcal{L}} S$. □

Lemma A.8. *If $S \leqslant T$ and $U \sim_{\mathcal{L}} S$ and $U \mid T \Longrightarrow U' \mid T'$, then $U \mid S \Longrightarrow U' \mid S'$ for some $S' \leqslant T'$.*

Proof. By induction on the length of reduction $U \mid T \Longrightarrow U' \mid T'$. In the base case $T' = T$ and we conclude immediately by taking $S' \stackrel{\text{def}}{=} S$. In the inductive case, we distinguish the following sub-cases.

Case $U = {!\{l_i : U_i\}_{i \in I}}$ and $U \mid T \longrightarrow {!l_k.U_k} \mid T \Longrightarrow U' \mid T'$ for some $k \in I$. From the hypothesis $U \sim_{\mathcal{L}} S$ and Lemma A.7 we deduce ${!l_k.U_k} \sim_{\mathcal{L}} S$. Using the induction hypothesis we deduce that there exists $S' \leqslant T'$ such that ${!l_k.U_k} \mid S \Longrightarrow U' \mid S'$. We conclude by observing that $U \mid S \longrightarrow {!l_k.U_k} \mid S$.

Case $T = {!\{l_j : T_j\}_{j \in J}}$ and $U \mid T \longrightarrow U \mid {!l_k.T_k} \Longrightarrow U' \mid T'$ for some $k \in J$. From the definition of $\leqslant$ we deduce $S = {!\{l_i : S_i\}_{i \in I}}$ with $J \subseteq I$ and $S_j \leqslant T_j$ for every $j \in J$. From the hypothesis $U \sim_{\mathcal{L}} S$ and Lemma A.7 we deduce $U \sim_{\mathcal{L}} {!l_k.S_k}$. Note that ${!l_k.S_k} \leqslant {!l_k.T_k}$, hence we can use the induction hypothesis to deduce that there exists $S' \leqslant T'$ such that $U \mid {!l_k.S_k} \Longrightarrow U' \mid S'$. We conclude by observing that $U \mid S \longrightarrow U \mid {!l_k.S_k}$.

Case $U = {!l_k.V}$ and $T = {?\{l_j : T_j\}_{j \in J}}$ and $U \mid T \longrightarrow V \mid T_k \Longrightarrow U' \mid T'$ for some $k \in J$. From the definition of $\leqslant$ we deduce $S = {?\{l_i : S_i\}_{i \in I}}$ and $I \subseteq J$ and $S_i \leqslant T_i$ for every $i \in I$. From the hypothesis $U \sim_{\mathcal{L}} S$ and Lemma A.7 we deduce $k \in I$ and $V \sim_{\mathcal{L}} S_k$. Using the induction hypothesis we deduce that there exists $S' \leqslant T'$ such that $V \mid S_k \Longrightarrow U' \mid S'$. We conclude by observing that $U \mid S \longrightarrow V \mid S_k$.

Case $U = {?\{l_i : U_i\}_{i \in I}}$ and $T = {!l_k.T''}$ and $U \mid T \longrightarrow U_k \mid T'' \Longrightarrow U' \mid T'$ for some $k \in I$. From the definition of $\leqslant$ we deduce $S = {!\{l_j : S_j\}_{j \in J}}$ and $k \in J$ and $S_k \leqslant T''$. From the hypothesis $U \sim_{\mathcal{L}} S$ and Lemma A.7 we deduce $U_k \sim_{\mathcal{L}} S_k$. Using the induction hypothesis we deduce that there exists $S' \leqslant T'$ such that $U_k \mid S_k \Longrightarrow U' \mid S'$. We conclude by observing that $U \mid S \Longrightarrow U \mid {!l_k.S_k} \longrightarrow U_k \mid S_k$.

Case $U = {?V.U''}$ and $T = {!V.T''}$ and $U \xrightarrow{?V} U''$ and $T \xrightarrow{!V} T''$ and $U \mid T \longrightarrow U'' \mid T'' \Longrightarrow U' \mid T'$. From the definition of $\leqslant$ we deduce $S = {!V.S''}$ and $S'' \leqslant T''$. From the hypothesis $U \sim_{\mathcal{L}} S$ and Lemma A.7 we deduce $U'' \sim_{\mathcal{L}} S''$. Using the induction hypothesis we deduce that there exists $S' \leqslant T'$ such that $U'' \mid S'' \Longrightarrow U' \mid S'$. We conclude by observing that $U \mid S \longrightarrow U'' \mid S''$.

Case $U = !V.U''$ *and* $T = ?V.T''$. Symmetric of the previous case. □

Next we show that fair subtyping preserves the reachability of a state that satisfies the liveness condition.

Lemma A.9. *If* $S \leqslant T$ *and* $U \sim_{\mathcal{L}} S$*, then* $U \mid T \Longrightarrow U' \mid T'$ *for some* U' *and* T' *such that* $U' \mathrel{\mathcal{L}} T'$.

Proof. By induction on the derivation of $S \leqslant_{\mathsf{ind}} T$. From the hypothesis $U \sim_{\mathcal{L}} S$ we deduce that $U \mid S \Longrightarrow U' \mid S'$ for some U' and S' such that $U' \mathrel{\mathcal{L}} S'$. That is, there exist φ and p such that $U \overset{\varphi^\perp}{\Longrightarrow} p\,\mathsf{end}$ and $S \overset{\varphi}{\Longrightarrow}$. We distinguish the following sub-cases:

Case $\varphi \in \mathsf{paths}(S) \cap \mathsf{paths}(T)$. From the hypothesis $S \leqslant T$ we deduce $S(\varphi) \leqslant T(\varphi)$. From $S(\varphi) \leqslant S'$ and Proposition A.3 we deduce $U' \mathrel{\mathcal{L}} S(\varphi)$ and therefore $U' \mathrel{\mathcal{L}} T(\varphi)$. We conclude by taking $T' \overset{\text{def}}{=} T(\varphi)$ and observing that $U \mid T \Longrightarrow U' \mid T'$.

Case $\varphi \in \mathsf{paths}(S) \setminus \mathsf{paths}(T)$. From Lemma A.1 we deduce that there exist ψ and $l \in \mathcal{L}$ such that $\psi \leq \varphi$ and $S(\psi!l) \leqslant_{\mathsf{ind}} T(\psi!l)$. From the hypothesis $S \leqslant T$ we deduce $S(\psi!l) \leqslant T(\psi!l)$. From the fact that $\psi \leq \varphi$ and $U \sim_{\mathcal{L}} S$, we deduce that there exists U'' such that $U \overset{\psi^\perp ?l}{\Longrightarrow} U''$ and $U'' \sim_{\mathcal{L}} S(\psi!l)$. Using the induction hypothesis we deduce that $U'' \mid S(\psi!l) \Longrightarrow U' \mid S'$ for some U' and S' such that $U' \mathrel{\mathcal{L}} S'$. We conclude $U \mid S \Longrightarrow U'' \mid S(\psi!l) \Longrightarrow U' \mid S'$. □

Lemma A.10. *If* $S \leqslant T$*, then* $U \sim_{\mathcal{L}} S$ *implies* $U \sim_{\mathcal{L}} T$ *for every* U.

Proof. Consider a run $U \mid T \Longrightarrow U'' \mid T''$. From Lemma A.8 we deduce that there exists $S'' \leqslant T''$ such that $U \mid S \Longrightarrow U'' \mid S''$. From the hypothesis $U \sim_{\mathcal{L}} S$ and Lemma A.7 we deduce $U'' \sim_{\mathcal{L}} S''$. From Lemma A.9 we conclude $U'' \mid T'' \Longrightarrow U' \mid T'$ for some U' and T' such that $U' \mathrel{\mathcal{L}} T'$. □

Theorem 3.9. *If* $S \leqslant T$*, then* $U \sim S$ *implies* $U \sim T$ *for every* U.

Proof. Special case of Lemma A.10. □

A.2.3 Completeness of fair subtyping. We prove that the preservation of compatibility, regardless of the particular liveness condition being considered, implies a relation that is slightly larger than *unfair* subtyping. In particular, let $\leqslant_{\mathsf{srv}}$ be the relation resulting from the coinductive interpretation of the rules in Table 1 together with the rule

$$\text{[U-END]}\ \frac{}{p\,\mathsf{end} \leqslant T}$$

establishing that $p\,\mathsf{end}$ is the least element of $\leqslant_{\mathsf{srv}}$. Notice that $\leqslant_{\mathsf{coind}} \subseteq \leqslant_{\mathsf{srv}}$.

Lemma A.11. *If* S *is* $\mathcal{L}$*-viable and* $U \sim_{\mathcal{L}} S$ *implies* $U \sim_{\mathcal{L}} T$ *for every* U*, then* $S \leqslant_{\mathsf{srv}} T$.

Proof. Let $\mathcal{R} \overset{\text{def}}{=} \{(S,T) \mid S \text{ is } \mathcal{L}\text{-viable and } U \sim_{\mathcal{L}} S \text{ implies } U \sim_{\mathcal{L}} T \text{ for every } U\}$ and observe that $S \mathrel{\mathcal{R}} T$ if S is $\mathcal{L}$-viable. We show $\mathcal{R} \subseteq \leqslant_{\mathsf{srv}}$ using the principle of coinduction. Namely we show that, whenever $S \mathrel{\mathcal{R}} T$, there exists a rule defining $\leqslant_{\mathsf{srv}}$ whose conclusion is $S \leqslant T$ and whose premises are all included in $\mathcal{R}$. We reason by cases on the shape of S, omitting symmetric cases.

Case $S = ?\mathsf{end}$. We conclude by observing that $S \leqslant T$ is the conclusion of the axiom [U-END].

Case $S = ?U.S'$. By definition of $\mathcal{R}$ we know that S is $\mathcal{L}$-viable, so S' is $\mathcal{L}$-viable as well by Proposition A.6. Let U' be an arbitrary session type such that $U' \sim_{\mathcal{L}} S'$. We know that such U' exists by definition of $\mathcal{L}$-viability. Now

$!U.U' \sim_{\mathcal{L}} S$ and therefore $!U.U' \sim_{\mathcal{L}} T$ by definition of $\mathcal{R}$. From Definition 3.7 we deduce that $T = ?U.T'$ and $U' \sim_{\mathcal{L}} T'$. Then $S' \mathrel{\mathcal{R}} T'$ since U' is arbitrary. We conclude by observing that $S \leqslant T$ is the conclusion of [f-channel].

Case $S = ?\{l_i : S_i\}_{i \in I}$. By definition of $\mathcal{R}$ we know that S is $\mathcal{L}$-viable, so S_i is $\mathcal{L}$-viable for every $i \in I$ by Proposition A.6. Let $U_{i \in I}$ be an arbitrary family of session types such that $U_i \sim S_i$ for every $i \in I$. We know that such family exists by definition of $\mathcal{L}$-viability. Let $V \overset{\text{def}}{=} !\{l_i : U_i\}_{i \in I}$. Now $V \sim_{\mathcal{L}} S$ and therefore $V \sim_{\mathcal{L}} T$ by definition of $\mathcal{R}$. From Definition A.4 we deduce that $T = ?\{l_j : T_j\}_{j \in J}$. Also, it must be the case that for each $i \in I$ there exists $j \in J$ such that $l_i = l_j$. Suppose by contradiction that this is not the case, namely that there exists $k \in I$ such that $l_j \neq l_k$ for every $j \in J$. Then we have $V \mid T \longrightarrow !l_k.U_k \mid T \not\longrightarrow$ which contradicts $V \sim_{\mathcal{L}} T$. Hence we may assume, without loss of generality, that $I \subseteq J$. Note that $S_i \mathrel{\mathcal{R}} T_i$ for every $i \in I$ since each U_i is arbitrary. We conclude by observing that $S \leqslant T$ is the conclusion of [f-label-in]. □

A slightly stronger result of Lemma A.11 can be obtained if we consider compatibility (Definition 3.7). In this case, the preservation of compatibility between S and T implies $S \leqslant_{\mathsf{coind}} T$.

Lemma A.12. *If S is bounded and $U \sim S$ implies $U \sim T$ for every U, then $S \leqslant_{\mathsf{coind}} T$.*

Proof. The proof is essentially the same as that of Lemma A.11, the only differences being the cases $S = ?\mathsf{end}$ and $S = !\mathsf{end}$. Let us discuss the first one. From the hypothesis $U \sim S$ we deduce $U = !\mathsf{end}$, implying $T = ?\mathsf{end}$ since this is the only session type compatible with U. Then we observe that $S \leqslant T$ is the conclusion of the axiom [f-end], which is the stricter version of [u-end]. □

Next we show that the preservation of compatibility between two session types S and T that are related by $\leqslant_{\mathsf{srv}}$ implies $S \leqslant_{\mathsf{ind}} T$.

Lemma A.13. *If $S \leqslant_{\mathsf{srv}} T$ and S is $\mathcal{L}$-viable and $U \sim_{\mathcal{L}} S$ implies $U \sim_{\mathcal{L}} T$ for every U, then $S \leqslant_{\mathsf{ind}} T$.*

Proof. Let $\uparrow$ be the largest relation such that $S \uparrow T$ implies that there exists $\varphi \in \mathsf{paths}(S) \setminus \mathsf{paths}(T)$ such that, for every $\psi \leq \varphi$ and $l \in \mathcal{L}$, if $S(\psi!l)$ is defined then $S(\psi!l) \uparrow T(\psi!l)$. From Lemma A.1 we know that $S \leqslant_{\mathsf{ind}} T$ if and only if $S \downarrow T$, hence $\uparrow$ happens to be the negation of $\leqslant_{\mathsf{ind}}$. We reason by contradiction, assuming that $U \sim_{\mathcal{L}} S$ implies $U \sim_{\mathcal{L}} T$ for every U and that $S \uparrow T$. Under the hypotheses $S \leqslant_{\mathsf{srv}} T$ and $S \uparrow T$, let $\mathcal{D}(S, T)$ be a session type corecursively defined as

$$\mathcal{D}(S,T) = \begin{cases} p^{\perp}U.\mathcal{D}(S',T') & \text{if } S = pU.S' \text{ and } T = pU.T' \\ !\{l_i : \mathcal{D}(S_i,T_i)\}_{i \in I, S_i \uparrow T_i} & \text{if } S = ?\{l_i : S_i\}_{i \in I}, T = ?\{l_j : T_j\}_{j \in J}, I \subseteq J \\ ?\{l_i : \mathcal{L}^{\perp}(S_i)\}_{i \in I \setminus J} + ?\{l_j : \mathcal{D}(S_j,T_j)\}_{j \in J} & \text{if } S = !\{l_i : S_i\}_{i \in I}, T = !\{l_j : T_j\}_{j \in J}, J \subseteq I \end{cases}$$

where the existence of $\mathcal{L}^{\perp}(S_i)$ in the last equation is guaranteed by the hypothesis that S is $\mathcal{L}$-viable. In defining $\mathcal{D}(S,T)$ we do not consider the case $S = p\,\mathsf{end}$ which is incompatible with the hypothesis $S \uparrow T$. We see from the definition of $\mathcal{D}(S,T)$ that $\mathcal{D}(S,T) \overset{\varphi^{\perp}}{\Longrightarrow} p\,\mathsf{end}$ for some p if and only if $T \overset{\varphi}{\not\Longrightarrow}$. Then we have $\mathcal{D}(S,T) \sim_{\mathcal{L}} S$ and $\mathcal{D}(S,T) \not\sim_{\mathcal{L}} T$, which contradicts the hypothesis. We conclude $S \leqslant_{\mathsf{ind}} T$ using Lemma A.1. □

Theorem 3.10. *If S is bounded and $U \sim S$ implies $U \sim T$ for every U, then $S \leqslant T$.*

Proof. From Lemma A.12 we deduce that there exists a derivation for $S \leqslant_{\mathsf{coind}} T$. Now we have to show that, for each judgment $S' \leqslant T'$ in this derivation, we have $S' \leqslant_{\mathsf{ind}} T'$. Observe that, from the existence of the judgment $S' \leqslant T'$, we deduce the existence of a string $\varphi \in \mathsf{paths}(S) \cap \mathsf{paths}(T)$ such that $S(\varphi) = S'$ and $T(\varphi) = T'$. Let V be an arbitrary session type such that $V \sim S'$ and consider the session type $U \overset{\text{def}}{=} \mathsf{client}(S, \varphi)$ defined inductively by the following equations:

$$
\begin{array}{rcll}
\mathsf{client}(S', \varepsilon) & = & V & \\
\mathsf{client}(pU.S', pU\varphi) & = & p^{\perp}U.\mathsf{client}(S', \varphi) & \\
\mathsf{client}(?\{l_i : S_i\}_{i \in I}, ?l_k\varphi) & = & !l_k.\mathsf{client}(S_k, \varphi) & k \in I \\
\mathsf{client}(!\{l_i : S_i\}_{i \in I}, !l_k\varphi) & = & ?\{l_i : S_i^{\perp}\}_{i \in I \setminus \{k\}} + ?l_k.\mathsf{client}(S_k, \varphi) & k \in I
\end{array}
$$

By construction of U and using the hypothesis that S is bounded we deduce $U \sim S$, hence $U \sim T$. Since V is arbitrary and $U \overset{\varphi^{\perp}}{\Longrightarrow}$ we deduce that $V \sim S'$ implies $V \sim T'$ for every V. We conclude $S' \leqslant_{\mathsf{ind}} T'$ using Lemma A.13, since $S \leqslant_{\mathsf{coind}} T$ implies $S' \leqslant_{\mathsf{coind}} T'$ which implies $S' \leqslant_{\mathsf{srv}} T'$. □

A.2.4 Transitivity of fair subtyping. We now have all the technical machinery to prove that $\leqslant$ is a pre-order.

Theorem A.14. *$\leqslant$ is transitive.*

Proof. Suppose $S \leqslant U$ and $U \leqslant T$. By definition of $\leqslant$ we have $S \leqslant_{\mathsf{coind}} U$ and $U \leqslant_{\mathsf{coind}} T$. It is a known fact that $\leqslant_{\mathsf{coind}}$ is transitive [Gay and Hole 2005], hence $S \leqslant_{\mathsf{coind}} T$ which implies $S \leqslant_{\mathsf{srv}} T$. Note that any session type is trivially $\mathcal{L}_2$-viable. Let V an arbitrary session type such that $V \sim_{\mathcal{L}_2} S$. Using Lemma A.10 twice we deduce $V \sim_{\mathcal{L}_2} U$ first and then $V \sim_{\mathcal{L}_2} T$. Since V is arbitrary, we have $S \leqslant_{\mathsf{ind}} T$ by Lemma A.13. We conclude $S \leqslant T$ since $\leqslant \, = \, \leqslant_{\mathsf{coind}} \cap \leqslant_{\mathsf{ind}}$ by definition of $\leqslant$. □

A.3 Supplement to Section 3.4

Theorem 3.13. *If $U \sim S$, then (1) $\|U, S\| \in \mathbb{N}$ and (2) $S \leqslant_{\mathsf{coind}} T$ implies $\|U, S\| \leq \|U, T\|$.*

Proof. Item 1 follows from the observation that $U \sim S$ implies $U \mid S \Longrightarrow p^{\perp}\,\mathsf{end} \mid p\,\mathsf{end}$ for some p, hence there exists φ such that $U \overset{\varphi^{\perp}}{\Longrightarrow} p^{\perp}\,\mathsf{end}$ and $S \overset{\varphi}{\Longrightarrow} p\,\mathsf{end}$. Item 2 is trivial to prove if we establish that

$$\{\varphi \mid U \overset{\varphi^{\perp}}{\Longrightarrow}, T \overset{\varphi}{\Longrightarrow}\} \subseteq \{\varphi \mid U \overset{\varphi^{\perp}}{\Longrightarrow}, S \overset{\varphi}{\Longrightarrow}\}$$

under the hypotheses $U \sim S$ and $S \leqslant_{\mathsf{coind}} T$. We prove that $U \overset{\varphi^{\perp}}{\Longrightarrow}$ and $T \overset{\varphi}{\Longrightarrow}$ implies $S \overset{\varphi}{\Longrightarrow}$ by induction on φ and by cases on its first action.

The base case $\varphi = \varepsilon$ is trivial. If $\varphi = pV\psi$, then $T = pV.T'$ for some T' and by definition of $\leqslant_{\mathsf{coind}}$ we deduce $S = pV.S'$ and $S' \leqslant_{\mathsf{coind}} T'$ for some S'. From $U \sim S$ we deduce $U = p^{\perp}V.U'$ for some $U' \sim S'$. Using the induction hypothesis we obtain $S' \overset{\psi}{\Longrightarrow}$, which is enough to conclude $S \overset{\varphi}{\Longrightarrow}$.

If $\varphi = !l\psi$, then $T = !\{l_j : T_j\}_{j \in J}$ and $l = l_k$ for some $k \in J$. By definition of $\leqslant_{\mathsf{coind}}$ we deduce $S = !\{l_i : S_i\}_{i \in I}$ for some $I \supseteq J$, hence $S \overset{!l}{\Longrightarrow}$. From $U \sim S$ we deduce $U = ?\{l_k : U_k\}_{k \in K}$ for some $K \supseteq I$ and $U_k \sim S_k$. Using the induction hypothesis we obtain $S_k \overset{\psi}{\Longrightarrow}$, from which we conclude $S \overset{\varphi}{\Longrightarrow}$.

If $\varphi = ?l\psi$, then $T = ?\{l_j : T_j\}_{j \in J}$ and $l = l_k$ for some $k \in J$. By definition of $\leqslant_{\mathsf{coind}}$ we deduce $S = ?\{l_i : S_i\}_{i \in I}$ for some $I \subseteq J$. From $U \sim S$ we deduce $U = !\{l_k : U_k\}_{k \in K}$ for some $K \subseteq I$. Also, it must be the case that $k \in K$ and $U_k \sim S_k$. Using the induction hypothesis we obtain $S_k \overset{\psi}{\Longrightarrow}$, from which we conclude $S \overset{\varphi}{\Longrightarrow}$. □

A.4 Supplement to Section 3.5

Theorem 3.15. *$U^{\perp} \sim U$ if and only if U is bounded.*

Proof. $\Rightarrow$. Let $\varphi \in \mathsf{paths}(U)$. Then $U^\perp \mid U \Longrightarrow U(\varphi)^\perp \mid U(\varphi)$. From the hypothesis that $U^\perp \sim U$ we deduce that there exists p such that $U(\varphi)^\perp \mid U(\varphi) \Longrightarrow p^\perp\,\mathsf{end} \mid p\,\mathsf{end}$. That is, there exists ψ such that $U(\varphi\psi) = p\,\mathsf{end}$. We conclude that U is bounded.

$\Leftarrow$. Consider a reduction $U^\perp \mid U \Longrightarrow S \mid T$. Using the fact that $S \mid T$ has been obtained by reducing a term consisting of session types that are one the dual of the other, it is always possible to extend this reduction so that $S \mid T \Longrightarrow V^\perp \mid V$ for some V that is a subtree of U. From the hypothesis that U is bounded we deduce that V is also bounded, hence $V \overset{\varphi}{\Longrightarrow} p\,\mathsf{end}$ for some φ and p. By definition of duality we have $V^\perp \overset{\varphi^\perp}{\Longrightarrow} p^\perp\,\mathsf{end}$. We conclude $S \mid T \Longrightarrow p^\perp\,\mathsf{end} \mid p\,\mathsf{end}$. □

A.5 Supplement to Section 3.6

Theorem 3.20. *For every S and T we have that $S \sim T$ if and only if every maximal fair run of $S \mid T$ is finite and ends with $p^\perp\,\mathsf{end} \mid p\,\mathsf{end}$ for some p.*

Proof. $\Rightarrow$. By contradiction, suppose that π is an infinite fair run of $S \mid T$. Since S and T are regular and have finitely many distinct subtrees, there must be a subtree S_1 of S and a subtree T_1 of T such that $S_1 \mid T_1$ occurs infinitely often in π. By definition of run and from the hypothesis $S \sim T$ we deduce that $S \mid T \Longrightarrow S_1 \mid T_1 \Longrightarrow p^\perp\,\mathsf{end} \mid p\,\mathsf{end}$. Note that $S_1 \mid T_1$ cannot be $p^\perp\,\mathsf{end} \mid p\,\mathsf{end}$ since this latter term is stuck. Hence, it must be the case that $\|S_1, T_1\| > 0$, namely $S_1 \mid T_1 \Longrightarrow S_2 \mid T_2 \Longrightarrow p^\perp\,\mathsf{end} \mid p\,\mathsf{end}$ for some S_2 and T_2 such that $\|S_1, T_1\| > \|S_2, T_2\|$. From the hypothesis that π is a fair run we deduce that the reduction $S_1 \mid T_1 \Longrightarrow S_2 \mid T_2$ occurs infinitely often in π, hence so does $S_2 \mid T_2$. Repeating these arguments we are able to find an infinite sequence of terms $S_i \mid T_i$ for $i = 1, 2, \dots$ occurring in π and having strictly decreasing distances from $p^\perp\,\mathsf{end} \mid p\,\mathsf{end}$, which is absurd. We conclude that there is no infinite fair run of $S \mid T$. As to the fact that every maximal fair run of $S \mid T$ ends with $p^\perp\,\mathsf{end} \mid p\,\mathsf{end}$, this follows immediately from the hypothesis $S \sim T$.

$\Leftarrow$. Let $S \mid T \Longrightarrow S' \mid T'$ and observe that this reduction corresponds to a finite run of $S \mid T$ ending with $S' \mid T'$. From the property that the notion of fair run is feasible we deduce that this run can be extended to a maximal fair run. From the hypothesis that every maximal fair run of $S \mid T$ is finite and ends with $p^\perp\,\mathsf{end} \mid p\,\mathsf{end}$ we conclude $S' \mid T' \Longrightarrow p^\perp\,\mathsf{end} \mid p\,\mathsf{end}$. □

B SUBJECT REDUCTION

The next result shows that typing is preserved by structural pre-congruence.

Lemma B.1. *If $\Gamma \vdash^n P$ and $P \preccurlyeq Q$, then $\Gamma \vdash^m Q$ for some $m \leq n$.*

Proof. The proof is by induction on the derivation of $P \preccurlyeq Q$ and by cases on the last rule applied. We only discuss a few representative cases, the remaining ones are analogous.

Case [s-par-comm]. Then $P = (x)(P_1 \mid P_2) \preccurlyeq (x)(P_2 \mid P_1) = Q$. From [t-par] we deduce that there exist $\Gamma_1, \Gamma_2, x, S_1, S_2, n_1$ and n_2 such that:

- $\Gamma = \Gamma_1, \Gamma_2$
- $\Gamma_i, x : S_i \vdash^{n_i} P_i$ for $i = 1, 2$
- $S_1 \sim S_2$
- $n = 1 + n_1 + n_2$

We conclude $\Gamma \vdash Q$ with one application of [t-par] by taking $m \overset{\text{def}}{=} n$.

Case [s-par-assoc]. Then $P = (x)(P_1 \mid (y)(P_2 \mid P_3)) \preccurlyeq (y)((x)(P_1 \mid P_2) \mid P_3) = Q$ and $x \in \mathsf{fn}(P_2)$. From [t-par] we deduce that there exist Γ_1, Γ_{23}, T_1, S_1, n_1 and n_{23} such that:

- $\Gamma = \Gamma_1, \Gamma_{23}$
- $\Gamma_1, x : T_1 \vdash^{n_1} P_1$
- $\Gamma_{23}, x : S_1 \vdash^{n_{23}} (y)(P_2 \mid P_3)$
- $T_1 \sim S_1$
- $n = 1 + n_1 + n_{23}$

From [t-par] we deduce that there exist Γ_2, Γ_3, T_2, S_2, n_2 and n_3 such that:

- $\Gamma_{23} = \Gamma_2, \Gamma_3$
- $\Gamma_2, x : S_1, y : T_2 \vdash^{n_2} P_2$
- $\Gamma_3, y : S_2 \vdash^{n_3} P_3$
- $T_2 \sim S_2$
- $n_{23} = 1 + n_2 + n_3$

Using [t-par] we derive $\Gamma_1, \Gamma_2, y : T_2 \vdash^{1+n_1+n_2} (x)(P_1 \mid P_2)$. We conclude $\Gamma \vdash^m (y)((x)(P_1 \mid P_2) \mid P_3)$ with another application of [t-par] by taking $m \overset{\text{def}}{=} n$.

Case [s-cast-new]. Then $P = (x)(\lceil x \rceil P_1 \mid P_2) \preccurlyeq (x)(P_1 \mid P_2)$. From [t-par] we deduce that there exist Γ_1, Γ_2, S_1, T, n_1 and n_2 such that:

- $\Gamma = \Gamma_1, \Gamma_2$
- $\Gamma_1, x : S_1 \vdash^{n_1} \lceil x \rceil P_1$
- $\Gamma_2, x : T \vdash^{n_2} P_2$
- $S_1 \sim T$
- $n = 1 + n_1 + n_2$

From [t-cast] we deduce that there exist S_2 and n_3 such that

- $\Gamma_1, x : S_2 \vdash^{n_3} P_1$
- $S_1 \leqslant S_2$
- $n_1 = 1 + n_3$

From $S_1 \sim T$ and $S_1 \leqslant S_2$ and Theorem 3.9 we deduce $S_2 \sim T$. We conclude with one application of [t-par] by taking $m \overset{\text{def}}{=} 1 + n_3 + n_2$.

Case [s-cast-swap]. Then $P = (x)(\lceil y \rceil P_1 \mid P_2) \preccurlyeq \lceil y \rceil (x)(P_1 \mid P_2)$ and $x \neq y$. From [t-par] we deduce that there exist Γ_1, S_1, S_2, T_1, n_1 and n_2 such that:

- $\Gamma = \Gamma_1, \Gamma_2, y : T_1$
- $\Gamma_1, x : S_1, y : T_1 \vdash^{n_1} \lceil y \rceil P_1$
- $\Gamma_2, x : S_2 \vdash^{n_2} P_2$
- $S_1 \sim S_2$
- $n = 1 + n_1 + n_2$

From [t-cast] we deduce that there exist T_2 and n_3 such that

- $T_1 \leqslant T_2$
- $\Gamma_1, x : S_1, y : T_2 \vdash^{n_3} P_1$
- $n_1 = 1 + n_3$

We derive $\Gamma_1, \Gamma_2, y : T_2 \vdash^{1+n_3+n_2} (x)(P_1 \mid P_2)$ with one application of [T-PAR] and we conclude with one application of [T-CAST] by taking $m \stackrel{\text{def}}{=} n$.

Case [S-CAST-COMM]. Then $P = \lceil x \rceil \lceil y \rceil P' \preccurlyeq \lceil y \rceil \lceil x \rceil P' = Q$. We can assume $x \neq y$ or else $P = Q$ and there is nothing to prove. From [T-CAST] we deduce that there exist Γ_1, S_1, S_2 and n_1 such that

- $\Gamma = \Gamma_1, x : S_1$
- $\Gamma_1, x : S_2 \vdash^{n_1} \lceil y \rceil P'$
- $S_1 \leqslant S_2$
- $n = 1 + n_1$

From [T-CAST] and the hypothesis $x \neq y$ we deduce that there exist Γ_2, T_1, T_2 and n_2 such that

- $\Gamma_1 = \Gamma_2, y : T_1$
- $\Gamma_2, x : S_2, y : T_2 \vdash^{n_2} P'$
- $T_1 \leqslant T_2$
- $n_1 = 1 + n_2$

We derive $\Gamma_2, x : S_1, y : T_2 \vdash^{n_1} \lceil x \rceil P'$ with one application of [T-CAST] and we conclude with another application of [T-CAST] by taking $m \stackrel{\text{def}}{=} n$.

Case [S-CALL]. Then $P = A\langle \overline{x} \rangle \preccurlyeq Q$ and $A(\overline{x}) \stackrel{\triangle}{=} Q$. From [T-CALL] we conclude that there exist $\overline{S}$ and m such that $A : [\overline{S}; m]$ and $\Gamma = \overline{x : S}$ and $\overline{x : S} \vdash^m Q$ and $m \leq n$. □

Then we have subject reduction, stating that typing is preserved also by reductions. Note that in this case we are not able to establish a general relation between the rank of the reducible process and that of the reduct. In particular, the rank may increase.

Lemma B.2 (subject reduction). *If $\Gamma \vdash^n P$ and $P \longrightarrow Q$, then $\Gamma \vdash^m Q$ for some m.*

Proof. By induction on the derivation of $P \longrightarrow Q$ and by cases on the last rule applied.

Case [R-CHOICE]. Then $P = P_1 \oplus P_2 \longrightarrow P_k = Q$ where $k \in \{1, 2\}$. From [T-CHOICE] we deduce that $\Gamma \vdash^m Q$ for some m, which is all we need to conclude.

Case [R-PICK]. Then $P = (x)(x!\{l_i : P_i\}_{i \in I} \mid R) \longrightarrow (x)(x!l_k.P_k \mid R) = Q$ where $k \in I$ and $|I| > 1$. From [T-PAR] we deduce that there exist Γ_1, Γ_2, S, T, n_1 and n_2 such that

- $\Gamma = \Gamma_1, \Gamma_2$
- $\Gamma_1, x : S \vdash^{n_1} x!\{l_i : P_i\}_{i \in I}$
- $\Gamma_2, x : T \vdash^{n_2} R$
- $S \sim T$
- $n = 1 + n_1 + n_2$

From [T-LABEL] we deduce that there exists a family $S_{i \in I}$ such that:

- $S = !\{l_i : S_i\}_{i \in I}$
- $\Gamma_1, x : S_i \vdash^{n_1} P_i$ for every $i \in I$

From $S \sim T$ and $S \longrightarrow !l_k.S_k$ and Lemma A.7 we deduce $!l_k.S_k \sim T$. We conclude with one application of [T-LABEL] and one application of [T-PAR] by taking $m \stackrel{\text{def}}{=} n$.

Case [R-SIGNAL]. Then $P = (x)(\mathsf{close}\ x \mid \mathsf{wait}\ x.Q) \longrightarrow Q$. From [T-PAR], [T-CLOSE] and [T-WAIT] we deduce that there exist n' and m such that:

- $x : \mathsf{!end} \vdash^{n'} \mathsf{close}\ x$
- $\Gamma, x : \mathsf{?end} \vdash^{m} \mathsf{wait}\ x.Q$
- $\Gamma \vdash^{m} Q$
- $n = 1 + n' + m$

There is nothing left to prove.

Case [R-LABEL]. Then $P = (x)(x!l_k.R \mid x?\{l_i : Q_i\}_{i\in I}) \longrightarrow (x)(R \mid Q_k) = Q$ with $k \in I$. From [T-PAR] we deduce that there exist Γ_1, Γ_2, S, T, n_1 and n_2 such that:

- $\Gamma = \Gamma_1, \Gamma_2$
- $\Gamma_1, x : S \vdash^{n_1} x!l_k.R$
- $\Gamma_2, x : T \vdash^{n_2} x?\{l_i : Q_i\}_{i\in I}$
- $S \sim T$
- $n = 1 + n_1 + n_2$

From [T-LABEL] we deduce that there exists S_1 such that $S = !l_k.S_1$ and $\Gamma_1, x : S_1 \vdash^{n_1} R$. From [T-LABEL] we deduce that there exists a family $T_{i\in I}$ such that:

- $T = ?\{l_i : T_i\}_{i\in I}$
- $\Gamma_2, x : T_i \vdash^{n_2} Q_i$ for every $i \in I$

From $S \sim T$ we deduce $S_1 \sim T_k$. We conclude with one application of [T-PAR] by taking $m \overset{\text{def}}{=} n$.

Case [R-CHANNEL]. Then $P = (x)(x!y.P' \mid x?(y).Q') \longrightarrow (x)(P' \mid Q') = Q$. From [T-PAR] we deduce that there exist Γ_1, Γ_2, S, T, n_1 and n_2 such that

- $\Gamma = \Gamma_1, \Gamma_2$
- $\Gamma_1, x : S \vdash^{n_1} x!y.P'$
- $\Gamma_2, x : T \vdash^{n_2} x?(y).Q'$
- $S \sim T$
- $n = 1 + n_1 + n_2$

From [T-CHANNEL-OUT] we deduce that there exist Γ_1', S_1 and S_2 such that

- $\Gamma_1 = \Gamma_1', y : S_1$
- $S = !S_1.S_2$
- $\Gamma_1', x : S_2 \vdash^{n_1} P'$

From [T-CHANNEL-IN] we deduce that there exist T_1 and T_2 such that

- $T = ?T_1.T_2$
- $\Gamma_2, x : T_2, y : T_1 \vdash^{n_2} Q'$

From $S \sim T$ we deduce $S_1 = T_1$ and $S_2 \sim T_2$. We conclude with one application of [T-PAR] by taking $m \overset{\text{def}}{=} n$.

Case [R-CAST]. Then $P = \lceil x \rceil P' \longrightarrow \lceil x \rceil Q' = Q$ and $P' \longrightarrow Q'$. From [T-CAST] we deduce that there exist Γ', S, T and n' such that

- $\Gamma = \Gamma', x : S$
- $\Gamma', x : T \vdash^{n'} P'$
- $S \leqslant T$
- $n = 1 + n'$

Using the induction hypothesis on $\Gamma', x : T \vdash^{n'} P'$ and $P' \longrightarrow Q'$ we derive $\Gamma', x : T \vdash^{m'} Q'$ for some m'. We conclude with one application of [T-CAST] by taking $m \overset{\text{def}}{=} 1 + m'$.

Case [R-PAR]. Then $P = (x)(P' \mid R) \longrightarrow (x)(Q' \mid R) = Q$ and $P' \longrightarrow Q'$. From [T-PAR] we deduce that there exist Γ_1, Γ_2, S, T, n_1 and n_2 such that:

- $\Gamma = \Gamma_1, \Gamma_2$
- $\Gamma_1, x : S \vdash^{n_1} P'$
- $\Gamma_2, x : T \vdash^{n_2} R$
- $S \sim T$
- $n = 1 + n_1 + n_2$

Using the induction hypothesis we deduce that $\Gamma_1, x : S \vdash^{m_1} Q'$ for some m_1. We conclude with one application of [T-PAR] by taking $m \overset{\text{def}}{=} m_1 + n_2$.

Case [R-STRUCT]. Then $P \preccurlyeq P' \longrightarrow Q' \preccurlyeq Q$. From Lemma B.1 we deduce $\Gamma \vdash^{n'} P'$ for some $n' \leq n$. Using the induction hypothesis we deduce $\Gamma \vdash^{m'} Q'$ for some m'. We conclude using Lemma B.1 once more. □

In the following it will also be useful to have a more refined version of Lemma B.1 showing that structural pre-congruence does not increase the *measure* of a process, not just its rank. Before proving this result, we formalize the tight relationship between the "plain" typing judgments (Table 3) and those that allow us to compute the measure of a process.

Lemma B.3. *The following properties hold:*

(1) $\Gamma \vdash^{n} P$ *implies* $\Gamma \vDash^{\mu} P$ *for some* $\mu \leq (n, 0)$*;*

(2) $\Gamma \vDash^{\mu} P$ *implies* $\Gamma \vdash^{n} P$ *for some n such that* $\mu \leq (n, 0)$*.*

Proof. We prove item 1 by induction on the derivation of $\Gamma \vdash^{n}_{\text{ind}} P$ and by cases on the last rule applied. The proof of item 2 is analogous.

Case [T-PAR]. Then $P = (x)(P_1 \mid P_2)$ and $\Gamma = \Gamma_1, \Gamma_2$ and $\Gamma_i, x : S_i \vdash^{n_i} P_i$ for $i = 1, 2$ and $S_1 \sim S_2$ and $n = 1 + n_1 + n_2$. Using the induction hypothesis we deduce that there exist μ_1 and μ_2 such that $\Gamma_i, x : S_i \vDash^{\mu_i} P_i$ and $\mu_i \leq (n_i, 0)$ for $i = 1, 2$. We conclude with one application of [MT-PAR] by taking $\mu \overset{\text{def}}{=} \mu_1 + \mu_2 + (0, \|S_1, S_2\|)$ and observing that $\mu < (n_1, 0) + (n_2, 0) + (1, 0) = (n, 0)$.

Case [T-CAST]. Then $P = \lceil x \rceil Q$ and $\Gamma = \Delta, x : S$ and $\Delta, x : T \vdash^{m} Q$ and $S \preccurlyeq T$ and $n = 1 + m$. Using the induction hypothesis we deduce $\Delta, x : T \vDash^{\nu} Q$ for some $\nu \leq (m, 0)$. We conclude with one application of [MT-CAST] by taking $\mu \overset{\text{def}}{=} \nu + (1, 0)$ and observing that $\mu \leq (m, 0) + (1, 0) = (n, 0)$.

In all the other cases. We conclude with one application of [MT-THREAD] by taking $\mu \overset{\text{def}}{=} (n, 0)$. □

Lemma B.4. *If* $\Gamma \vDash^{\mu} P$ *and* $P \preccurlyeq Q$*, then there exists* $\nu \leq \mu$ *such that* $\Gamma \vDash^{\nu} Q$*.*

Proof. By induction on the derivation of $P \preccurlyeq Q$ and by cases on the last rule applied. We only consider the base cases.

Case [S-PAR-COMM]. Then $P = (x)(P_1 \mid P_2) \preccurlyeq (x)(P_2 \mid P_1) = Q$. From [MT-PAR] we deduce that there exist $\Gamma_1, \Gamma_2, S, T, \mu_1$ and μ_2 such that

- $\Gamma = \Gamma_1, \Gamma_2$
- $\mu = \mu_1 + \mu_2 + (0, \|S, T\|)$
- $\Gamma_1, x : S \vDash^{\mu_1} P_1$

- $\Gamma_2, x : T \vDash^{\mu_2} P_2$

We conclude with one application of [MT-PAR] by taking $\nu \overset{\text{def}}{=} \mu$.

Case [S-PAR-ASSOC]. Then $P = (x)(P_1 \mid (y)(Q_1 \mid Q_2)) \preccurlyeq (y)((x)(P_1 \mid Q_1) \mid Q_2) = Q$. From [MT-PAR] we deduce that there exist Γ_1, Δ_1, Δ_2, μ_1, ν_1, ν_2, S_1, S_2, T_1 and T_2 such that

- $\Gamma = \Gamma_1, \Delta_1, \Delta_2$
- $\mu = \mu_1 + (\nu_1 + \nu_2 + (0, \|T_1, T_2\|)) + (0, \|S_1, S_2\|)$
- $\Gamma_1, x : S_1 \vDash^{\mu_1} P_1$
- $\Delta_1, y : T_1, x : S_2 \vDash^{\nu_1} Q_1$
- $\Delta_2, y : T_2 \vDash^{\nu_2} Q_2$

We conclude with two applications of [MT-PAR] by taking $\nu \overset{\text{def}}{=} \mu$.

Case [S-CAST-COMM]. Then $P = \lceil x \rceil \lceil y \rceil P' \preccurlyeq \lceil x \rceil \lceil y \rceil P' = Q$. We only consider the case $x \neq y$ or else there is nothing interesting to prove. From [MT-CAST] we deduce that there exist Γ', μ', S_1, T_1, S_2 and T_2 such that

- $\Gamma = \Gamma', x : S_1, y : T_1$
- $\mu = \mu' + (1, 0) + (1, 0)$
- $\Gamma', x : S_2, y : T_2 \vDash^{\mu'} P'$
- $S_1 \preccurlyeq S_2$ and $T_1 \preccurlyeq T_2$

We conclude with two applications of [MT-CAST] by taking $\nu \overset{\text{def}}{=} \mu$.

Case [S-CAST-NEW]. Then $P = (x)(\lceil x \rceil P_1 \mid P_2) \preccurlyeq (x)(P_1 \mid P_2) = Q$. From [MT-PAR] we deduce that there exist Γ_1, Γ_2, μ_1, μ_2, S_1 and T such that:

- $\Gamma = \Gamma_1, \Gamma_2$
- $\mu = \mu_1 + \mu_2 + (0, \|S_1, T\|)$
- $\Gamma_1, x : S_1 \vDash^{\mu_1} \lceil x \rceil P_1$ and $\Gamma_2, x : T \vDash^{\mu_2} P_2$
- $S_1 \sim T$

From [MT-CAST] we deduce that there exist μ_1' and S_2 such that

- $\mu_1 = \mu_1' + (1, 0)$
- $\Gamma_1, x : S_2 \vDash^{\mu_1'} P_1$
- $S_1 \preccurlyeq S_2$

From $S_1 \sim T$ and $S_1 \preccurlyeq S_2$ and Theorem 3.9 we deduce $S_2 \sim T$. We conclude with one application of [MT-PAR] taking $\nu \overset{\text{def}}{=} \mu_1' + \mu_2 + (0, \|S_2, T\|) < \mu$.

Case [S-CAST-SWAP]. Then $P = (x)(\lceil y \rceil P_1 \mid P_2) \preccurlyeq \lceil y \rceil (x)(P_1 \mid P_2) = Q$ and $x \neq y$. From [MT-PAR] we deduce that there exist Γ_1, μ_1, μ_2, S_1, S_2 and T_1 such that:

- $\Gamma = \Gamma_1, \Gamma_2, y : T_1$
- $\mu = \mu_1 + \mu_2 + (0, \|S_1, S_2\|)$
- $\Gamma_1, x : S_1, y : T_1 \vDash^{\mu_1} \lceil y \rceil P_1$ and $\Gamma_2, x : S_2 \vDash^{\mu_2} P_2$
- $S_1 \sim S_2$

From [MT-CAST] we deduce that there exist μ_1' and T_2 such that

- $\mu_1 = \mu_1' + (1, 0)$
- $T_1 \preccurlyeq T_2$

- $\Gamma_1, x : S_1, y : T_2 \vdash P_1$

We derive $\Gamma_1, \Gamma_2, y : T_2 \vDash^{\mu'_1+\mu_2+(0,\|S_1,S_2\|)} (x)(P_1 \mid P_2)$ with one application of [mt-par] and we conclude with one application of [mt-cast] by taking $\nu \overset{\text{def}}{=} \mu$.

Case [s-call]. Then $P = A\langle\overline{x}\rangle \preccurlyeq Q$ where $A(\overline{x}) \overset{\triangle}{=} Q$. From [mt-thread] we deduce that $\Gamma \vdash^n A\langle\overline{x}\rangle$ for some n such that $\mu = (n, 0)$. Using Lemma B.1 we deduce that $\Gamma \vdash^m Q$ for some $m \leq n$. Using Lemma B.3 we deduce that $\Gamma \vDash^\nu Q$ for some $\nu \leq (m, 0)$. We conclude by observing that $\nu \leq (m, 0) \leq (n, 0) = \mu$. □

C NORMAL FORMS

In this section we introduce some normal forms that are instrumental to the soundness proof of the type system. To this aim, it is useful to also introduce *process contexts* as a convenient way of referring to sub-processes. A process context $\mathcal{C}$ is essentially a process with a *hole* denoted by $[\,]$:

$$\textbf{Process context} \quad \mathcal{C}, \mathcal{D} \ ::= \ [\,] \mid (x)(\mathcal{C} \mid P) \mid (x)(P \mid \mathcal{C}) \mid \lceil x \rceil \mathcal{C}$$

As usual, we write $\mathcal{C}[P]$ for the process obtained by replacing the hole in $\mathcal{C}$ with P. Note that this operation may capture channel names that occur free in P and that are bound by $\mathcal{C}$.

C.1 Choice normal form

Definition C.1 (choice normal form). We say that $P_1 \oplus P_2$ is an *unguarded choice* of P if there exists $\mathcal{C}$ such that $P \preccurlyeq \mathcal{C}[P_1 \oplus P_2]$. We say that P is in *choice normal form* if it has no unguarded choices.

We can reduce any well-typed process into a process that is in choice normal form. The fact that the original process is well typed guarantees that this reduction eventually terminates when all the unguarded choices have been resolved.

Lemma C.2. *If $\Gamma \vDash^\mu P$, then there exist Q in choice normal form and $\nu \leq \mu$ such that $P \Longrightarrow Q$ and $\Gamma \vDash^\nu Q$.*

Proof. We do an induction on $\Gamma \vdash_{\text{ind}} P$, which follows from the hypothesis $\Gamma \vDash^\mu P$ and Lemma B.3, and we reason by cases on the shape of P.

All cases where P is already in choice normal form. We conclude taking $Q \overset{\text{def}}{=} P$ and $\nu \overset{\text{def}}{=} \mu$.

Case $P = A\langle\overline{x}\rangle$ and $A(\overline{x}) \overset{\triangle}{=} R$. From [mt-thread] and [t-call] we deduce that $\Gamma = \overline{x : S}$ and $A : [\overline{S}; m]$ and $\Gamma \vdash^m R$ and $m \leq n$ and $\mu = (n, 0)$. From Lemma B.3 we deduce $\Gamma \vDash^{\mu'} P'$ for some $\mu' \leq (m, 0)$. Using the induction hypothesis we deduce that there exist Q in choice normal form and $\nu \leq \mu'$ such that $P' \Longrightarrow Q$ and $\Gamma \vDash^\nu Q$. We conclude by observing that $P \Longrightarrow Q$ using [s-call] and [r-struct] and that $\nu \leq \mu' \leq (m, 0) \leq (n, 0) = \mu$.

Case $P = P_1 \oplus_k P_2$. From [mt-thread] and either [t-choice] or [co-choice] we deduce $\mu = (n, 0)$ and $\Gamma \vdash^n P_k$. From Lemma B.3 we deduce $\Gamma \vDash^{\mu'} P_k$ for some $\mu' \leq (n, 0)$. Using the induction hypothesis we deduce that there exist Q in choice normal form and $\nu \leq \mu'$ such that $P_k \Longrightarrow Q$ and $\Gamma \vDash^\nu Q$. We conclude by observing that $P \longrightarrow P_k$ by [r-choice] and that $\nu \leq \mu' \leq (n, 0) = \mu$.

Case $P = (x)(P_1 \mid P_2)$. From [mt-par] we deduce $\Gamma = \Gamma_1, \Gamma_2$ and $\mu = \mu_1 + \mu_2 + (0, \|S_1, S_2\|)$ and $\Gamma_i, x : S_i \vDash^{\mu_i} P_i$ for $i = 1, 2$ and $S_1 \sim S_2$. Using the induction hypothesis we deduce that there exist Q_1 and Q_2 in choice normal form and $\nu_1 \leq \mu_1$ and $\nu_2 \leq \mu_2$ such that $P_i \Longrightarrow Q_i$ and $\Gamma_i, x : S_i \vDash^{\nu_i} Q_i$ for $i = 1, 2$. We conclude by taking $\nu \overset{\text{def}}{=} \nu_1 + \nu_2 + (0, \|S_1, S_2\|)$ with one application of [mt-par], observing that $\nu = \nu_1 + \nu_2 + (0, \|S_1, S_2\|) \leq \mu_1 + \mu_2 + (0, \|S_1, S_2\|) = \mu$.

Case $P = \lceil x \rceil P'$. Analogous to the previous case, just simpler. □

C.2 Thread normal form

We introduce a normal form that makes it easier to locate the components of a process that may interact with each other. Intuitively, a process is in *thread normal form* if it consists of an initial prefix of casts followed by a parallel composition of threads, where a thread is either done or a process waiting to perform an input/output action on some channel x. In this latter case, we say that the thread is an x-thread. Note that a process invocation $A\langle\overline{x}\rangle$ is *not* a thread. Formally:

Definition C.3 (thread normal form). A process is in *thread normal form* if it is generated by the grammar below:

$$\begin{array}{rcl} P^{nf}, Q^{nf} & ::= & \lceil x \rceil P^{nf} \mid P^{par} \\ P^{par}, Q^{par} & ::= & (x)(P^{par} \mid Q^{par}) \mid P^{th} \\ P^{th} & ::= & \mathsf{done} \mid \mathsf{close}\ x \mid \mathsf{wait}\ x.P \mid x!\{l_i : P_i\}_{i\in I} \mid x?\{l_i : P_i\}_{i\in I} \mid x!y.P \mid x?(y).P \end{array}$$

It is easy to rewrite any *well-typed* process that is in choice normal form into thread normal form using structural pre-congruence. The hypothesis that the process is well typed, at least according to the inductive interpretation of the typing rules with the corule [co-label], is necessary to guarantee that a process invocation may eventually be expanded to a term other than another process invocation. For example, the process A defined by $A \stackrel{\triangle}{=} A$ has no thread normal form and is ill typed. By combining this result with Lemma B.1 we can deduce that the obtained thread normal form is also well typed.

Lemma C.4. *If $\Gamma \vdash_{\mathsf{ind}} P$ and P is in choice normal form, then there exists P^{nf} such that $P \preccurlyeq P^{nf}$.*

Proof. By induction on $\Gamma \vdash_{\mathsf{ind}} P$ and by cases on the last rule applied.

Cases [t-choice] *and* [co-choice]. These cases are impossible from the hypothesis that P is in choice normal form.

Cases [t-done], [t-wait], [t-close], [t-channel-in], [t-channel-out], [t-label], [co-label]. Then P is a thread and is already in thread normal form and we conclude by reflexivity of $\preccurlyeq$.

Case [t-call]. Then there exist A, Q, $\overline{x}$ and $\overline{S}$ such that

- $P = A\langle\overline{x}\rangle$
- $A(\overline{x}) \stackrel{\triangle}{=} Q$
- $\Gamma = \overline{x : S}$
- $\overline{x : S} \vdash_{\mathsf{ind}} Q$

Using the induction hypothesis on $\overline{x : S} \vdash_{\mathsf{ind}} Q$ we deduce that there exists P^{nf} such that $Q \preccurlyeq P^{nf}$. We conclude $P \preccurlyeq P^{nf}$ using [s-call] and the transitivity of $\preccurlyeq$.

Case [t-par]. Then there exist x, P_1, P_2, Γ_1, Γ_2, S_1 and S_2 such that

- $P = (x)(P_1 \mid P_2)$
- $\Gamma = \Gamma_1, \Gamma_2$
- $\Gamma_i, x : S_i \vdash_{\mathsf{ind}} P_i$ for $i = 1, 2$

Using the induction hypothesis on $\Gamma_i, x : S_i \vdash_{\mathsf{ind}} P_i$ we deduce that there exists P_i^{nf} such that $P_i \preccurlyeq P_i^{nf}$ for $i = 1, 2$. By definition of thread normal form, it must be the case that $P_i^{nf} = \lceil \overline{x_i} \rceil P_i^{par}$ for some $\overline{x_i}$ and P_i^{par}. Let $\overline{y_i}$ be the same sequence as $\overline{x_i}$ except that occurrences of x have been removed. We conclude by taking $P^{nf} \stackrel{\text{def}}{=} \lceil \overline{y_1 y_2} \rceil (x)(P_1^{par} \mid P_2^{par})$

and observing that

$$\begin{array}{rcll} P & = & (x)(P_1 \mid P_2) & \text{by definition of } P \\ & \preccurlyeq & (x)(P_1^{nf} \mid P_2^{nf}) & \text{using the induction hypothesis} \\ & = & (x)(\lceil \overline{x_1} \rceil P_1^{par} \mid \lceil \overline{x_2} \rceil P_2^{par}) & \text{by definition of thread normal form} \\ & \preccurlyeq & \lceil \overline{y_1 y_2} \rceil (x)(P_1^{par} \mid P_2^{par}) & \text{by [s-cast-new], [s-cast-swap] and [s-par-comm]} \\ & = & P^{nf} & \text{by definition of } P^{nf} \end{array}$$

Case [t-cast]. Then there exist x, Q, Γ', S and T such that

- $P = \lceil x \rceil Q$
- $\Gamma = \Gamma', x : S$
- $\Gamma', x : T \vdash_{\mathsf{ind}} Q$
- $S \preccurlyeq T$

Using the induction hypothesis on $\Gamma', x : T \vdash_{\mathsf{ind}} Q$ we deduce that there exists Q^{nf} such that $Q \preccurlyeq Q^{nf}$. We conclude by taking $P^{nf} \stackrel{\text{def}}{=} \lceil x \rceil Q^{nf}$ using the fact that $\preccurlyeq$ is a pre-congruence. □

C.3 Proximity normal form

A process in proximity normal form is such that there exist at least two x-threads that are next to each other. Since each thread is waiting to perform an operation on the same session x, the two thread may potentially reduce if the operations are complementary ones.

Definition C.5 (proximity normal form). We say that P^{nf} is in *proximity normal form* if $P^{nf} = \mathcal{C}[(x)(P^{th} \mid Q^{th})]$ for some $\mathcal{C}$, x, P^{th} and Q^{th} where P^{th} and Q^{th} are x-threads.

In order to show that every well-typed, closed process in thread normal form can also be rewritten in proximity normal form we prove Lemma C.6, which pushes a restriction (x) next to a process in which x occurs free, which might as well be an x-thread.

Lemma C.6 (proximity). *If* $x \in \mathsf{fn}(P) \setminus \mathsf{bn}(\mathcal{C})$, *then* $(x)(\mathcal{C}[P] \mid Q) \preccurlyeq \mathcal{D}[(x)(P \mid Q)]$ *for some* $\mathcal{D}$.

Proof. By induction on the structure of $\mathcal{C}$ and by cases on its shape.

Case $\mathcal{C} = [\ ]$. We conclude by taking $\mathcal{D} \stackrel{\text{def}}{=} [\ ]$ using the reflexivity of $\preccurlyeq$.

Case $\mathcal{C} = (y)(\mathcal{C}' \mid R)$. From the hypothesis $x \in \mathsf{fn}(P) \setminus \mathsf{bn}(\mathcal{C})$ we deduce $x \neq y$ and $x \in \mathsf{fn}(P) \setminus \mathsf{bn}(\mathcal{C}')$. Using the induction hypothesis we deduce that there exists $\mathcal{D}'$ such that $(x)(\mathcal{C}'[P] \mid Q) \preccurlyeq \mathcal{D}'[(x)(P \mid Q)]$. Take $\mathcal{D} \stackrel{\text{def}}{=} (y)(\mathcal{D}' \mid R)$. We conclude

$$\begin{array}{rcll} (x)(\mathcal{C}[P] \mid Q) & = & (x)((y)(\mathcal{C}'[P] \mid R) \mid Q) & \text{by definition of } \mathcal{C} \\ & \preccurlyeq & (x)(Q \mid (y)(\mathcal{C}'[P] \mid R)) & \text{by [s-par-comm]} \\ & \preccurlyeq & (y)((x)(Q \mid \mathcal{C}'[P]) \mid R) & \text{by [s-par-assoc] and } x \in \mathsf{fn}(\mathcal{C}'[P]) \\ & \preccurlyeq & (y)((x)(\mathcal{C}'[P] \mid Q) \mid R) & \text{by [s-par-comm]} \\ & \preccurlyeq & (y)(\mathcal{D}'[(x)(P \mid Q)] \mid R) & \text{using the induction hypothesis} \\ & = & \mathcal{D}[(x)(P \mid Q)] & \text{by definition of } \mathcal{D} \end{array}$$

where, in using [s-par-assoc], we note that $x \in \mathsf{fn}(\mathcal{C}'[P])$ since $x \in \mathsf{fn}(P) \setminus \mathsf{bn}(\mathcal{C})$.

Case $\mathcal{C} = (y)(R \mid \mathcal{C}')$. Symmetric of the previous case.

Case $C = \lceil y \rceil C'$ *and* $x \neq y$. Using the induction hypothesis we deduce that there exists $\mathcal{D}'$ such that $(x)(C'[P] \mid Q) \preccurlyeq \mathcal{D}'[(x)(P \mid Q)]$. Take $\mathcal{D} \overset{\text{def}}{=} \lceil y \rceil \mathcal{D}'$. We conclude

$$\begin{array}{rcll} (x)(C[P] \mid Q) & = & (x)(\lceil y \rceil C'[P] \mid Q) & \text{by definition of } C \\ & \preccurlyeq & \lceil y \rceil (x)(C'[P] \mid Q) & \text{by [s-cast-swap] and } x \neq y \\ & \preccurlyeq & \lceil y \rceil \mathcal{D}'[(x)(P \mid Q)] & \text{using the induction hypothesis} \\ & = & \mathcal{D}[(x)(P \mid Q)] & \text{by definition of } \mathcal{D} \end{array}$$

Case $C = \lceil x \rceil C'$. Using the induction hypothesis we deduce that there exists $\mathcal{D}$ such that $(x)(C'[P] \mid Q) \preccurlyeq \mathcal{D}[(x)(P \mid Q)]$. We conclude

$$\begin{array}{rcll} (x)(C[P] \mid Q) & = & (x)(\lceil x \rceil C'[P] \mid Q) & \text{by definition of } C \\ & \preccurlyeq & (x)(C'[P] \mid Q) & \text{by [s-cast-new]} \\ & \preccurlyeq & \mathcal{D}[(x)(P \mid Q)] & \text{using the induction hypothesis} \end{array}$$

□

We can now prove the fact that every well-typed, closed process in thread normal form can be rewritten using structural pre-congruence either to done or to a process in proximity normal form. Note that this property is essentially a deadlock freedom result.

Lemma C.7. *If* $\emptyset \vDash^{\mu} P^{nf}$*, then* $P^{nf} = \mathsf{done}$ *or* $P^{nf} \preccurlyeq Q^{nf}$ *for some* Q^{nf} *in proximity normal form.*

Proof. A simple induction on the derivation of $\emptyset \vDash^{\mu} P^{nf}$ allows us to deduce that P^{nf} consists of k sessions and $k+1$ threads. If $k = 0$, then we conclude $P^{nf} = \mathsf{done}$. If $k > 0$, then each of the $k+1$ threads is an x_i-thread for some x_i. Since there are $k+1$ threads but only k distinct sessions, it must be the case that $x_i = x_j$ for some $1 \leq i < j \leq k+1$. In other words, there exist C, C_1, C_2, P_1^{th} and P_2^{th} such that P_1^{th} and P_2^{th} are x-threads and $P^{nf} = C[(x)(C_1[P_1^{th}] \mid C_2[P_2^{th}])]$. We conclude

$$\begin{array}{rcll} P^{nf} & = & C[(x)(C_1[P_1^{th}] \mid C_2[P_2^{th}])] & \text{by definition of } P^{nf} \\ & \preccurlyeq & C[\mathcal{D}_1[(x)(P_1^{th} \mid C_2[P_2^{th}])]] & \text{for some } \mathcal{D}_1 \text{ by Lemma C.6} \\ & \preccurlyeq & C[\mathcal{D}_1[(x)(C_2[P_2^{th}] \mid P_1^{th})]] & \text{by [s-par-comm]} \\ & \preccurlyeq & C[\mathcal{D}_1[\mathcal{D}_2[(x)(P_2^{th} \mid P_1^{th})]]] & \text{for some } \mathcal{D}_2 \text{ by Lemma C.6} \\ & \overset{\text{def}}{=} & Q^{nf} & \end{array}$$

where $x = x_i = x_j$. The fact that Q^{nf} is in thread normal form follows from the observation that P^{nf} does not have unguarded casts (it is a closed process in thread normal form) so the pre-congruence rules applied here and in Lemma C.6 do not move casts around. We conclude that Q^{nf} is in proximity normal form by its shape. □

D SOUNDNESS

Here we prove the soundness of the type system. As already hinted at in Section 6, the proof is loosely based on the method of helpful directions [Francez 1986], namely on the property that a (well-typed) process *may* reduce in such a way that its measure strictly decreases. Recall that this property is not true for every reduction. Here we assume that the reducing process is in proximity normal form. The same result will be generalized later on.

Lemma D.1. *If* $\Gamma \vDash^{\mu} P^{nf}$ *where* P^{nf} *is in proximity normal form, then there exist* Q *and* $\nu < \mu$ *such that* $P^{nf} \Longrightarrow^{+} Q$ *and* $\Gamma \vDash^{\nu} Q$.

Proof. From the hypothesis that P^{nf} is in proximity normal form we know that $P^{nf} = C[(x)(P^{th} \mid Q^{th})]$ for some C, x, P^{th} and Q^{th} such that both P^{th} and Q^{th} are x-threads. We reason by induction on C and by cases on its shape.

Case $C = [\,]$. From [mt-par] and [mt-thread] we deduce that there exist $\Gamma_1, \Gamma_2, S_1, S_2, n_1$ and n_2 such that $\Gamma = \Gamma_1, \Gamma_2$ and $\Gamma_1, x : S_1 \vdash^{n_1} P^{th}$ and $\Gamma_2, x : S_2 \vdash^{n_2} Q^{th}$ and $S_1 \sim S_2$ and $\mu = (n_1 + n_2, \|S_1, S_2\|)$. We now reason by cases on the shape of S_1 and S_2, considering only those cases that are compatible with the fact that $S_1 \sim S_2$ and omitting symmetric cases.

- *Case* $S_1 = !\mathsf{end}$ *and* $S_2 = ?\mathsf{end}$. Then $\Gamma_1 = \emptyset$ and $P^{th} = \mathsf{close}\ x$ and $Q^{th} = \mathsf{wait}\ x.Q$ and $\Gamma_2 \vdash^{n_2} Q$ and $\|S_1, S_2\| = 1$. By Lemma B.3 we deduce that $\Gamma_2 \vDash^{\nu} Q$ for some $\nu \leq (n_2, 0)$. We conclude by observing that $P^{nf} \longrightarrow Q$ and that $\nu \leq (n_2, 0) < (n_1 + n_2, \|S_1, S_2\|) = \mu$.
- *Case* $S_1 = !\{l_i : S_i\}_{i \in I}$ *and* $S_2 = ?\{l_j : T_j\}_{j \in J}$. Because of the relation $S_1 \sim S_2$ we can assume, without loss of generality, that $I \subseteq J$. From the same relation we also deduce $S_i \sim T_i$ for every $i \in I$. From [t-label] we deduce that $P^{th} = x!\{l_i : P_i\}_{i \in I}$ and $Q^{th} = x?\{l_j : Q_j\}_{j \in J}$ and $\Gamma_1, x : S_i \vdash^{n_1} P_i$ for every $i \in I$ and $\Gamma_2, x : T_j \vdash^{n_2} Q_j$ for every $j \in J$. From Definition 3.12 we deduce that there exist φ and p such that $S_1 \overset{\varphi^{\perp}}{\Longrightarrow} p^{\perp}\ \mathsf{end}$ and $S_2 \overset{\varphi}{\Longrightarrow} p\ \mathsf{end}$ and $\|S_1, S_2\| = 1 + |\varphi|$. Because of the shape of S_1 and S_2, it must be the case that $\varphi = ?l_k\psi$ for some $k \in I$ and ψ such that $\|S_k, T_k\| = 1 + |\psi| = \|S_1, S_2\| - 1$. Let $Q \overset{\text{def}}{=} (x)(P_k \mid Q_k)$ and observe that $P^{nf} \Longrightarrow^{+} Q$. From Lemma B.3 we deduce that there exist $\mu_1 \leq (n_1, 0)$ and $\mu_2 \leq (n_2, 0)$ such that $\Gamma_1, x : S_k \vDash^{\mu_1} P_k$ and $\Gamma_2, x : T_k \vDash^{\mu_2} Q_k$. Let $\nu \overset{\text{def}}{=} \mu_1 + \mu_2 + (0, \|S_k, T_k\|)$. We conclude with one application of [mt-par] by observing that $\nu = \mu_1 + \mu_2 + (0, \|S_k, T_k\|) \leq (n_1 + n_2, \|S_k, T_k\|) < (n_1 + n_2, \|S_1, S_2\|) = \mu$.
- *Case* $S_1 = !U.S$ *and* $S_2 = ?U.T$. Analogous to (but simpler than) the previous case.

Case $C = (y)(\mathcal{D} \mid R^{par})$. Let $P_1^{nf} \overset{\text{def}}{=} \mathcal{D}[(x)(P^{th} \mid Q^{th})]$ and observe that P_1^{nf} is in proximity normal form. From [mt-par] we deduce that there exist $\Gamma_1, \Gamma_2, T_1, T_2, \mu_1$ and μ_2 such that $\Gamma = \Gamma_1, \Gamma_2$ and $\Gamma_1, y : T_1 \vDash^{\mu_1} P_1^{nf}$ and $\Gamma_2, y : T_2 \vDash^{\mu_2} R^{par}$ and $\mu = \mu_1 + \mu_2 + (0, \|T_1, T_2\|)$. Using the induction hypothesis we deduce that there exist Q_1' and $\nu_1 < \mu_1$ such that $P_1^{nf} \Longrightarrow^{+} Q_1'$ and $\Gamma_1, y : T_1 \vDash^{\nu_1} Q_1'$. We conclude by taking $Q \overset{\text{def}}{=} (y)(Q_1' \mid R^{par})$ and $\nu \overset{\text{def}}{=} \nu_1 + \mu_2 + (0, \|T_1, T_2\|)$ and observing that $\nu < \mu$.

Case $C = (y)(R^{par} \mid \mathcal{D})$. Symmetric of the previous case. □

Here we prove that any well-typed, closed process can be either rewritten to done using structural pre-congruence or reduced so as to obtain a strictly smaller measure.

Lemma 6.5. *If $\emptyset \vDash^{\mu} P$, then either $P \preccurlyeq \mathsf{done}$ or $P \Longrightarrow^{+} Q$ and $\emptyset \vDash^{\nu} Q$ for some Q and $\nu < \mu$.*

Proof. Using Lemma C.2 we deduce that there exist P' in choice normal form such that $P \Longrightarrow P'$ and $\emptyset \vDash^{\mu'} P'$ and $\mu' \leq \mu$. By Lemma B.3 we deduce $\emptyset \vdash P'$. Using Lemma C.4 we deduce that there exist P^{nf} such that $P' \preccurlyeq P^{nf}$. If $P^{nf} = \mathsf{done}$ there is nothing left to prove. If $P^{nf} \neq \mathsf{done}$, by Lemma C.7 we deduce $P^{nf} \preccurlyeq Q^{nf}$ for some Q^{nf} in proximity normal form. From Lemma B.4 we deduce $\emptyset \vDash^{\mu''} Q^{nf}$ for some $\mu'' \leq \mu'$. Using Lemma D.1 we conclude that $Q^{nf} \Longrightarrow^{+} Q$ and $\emptyset \vDash^{\nu} Q$ for some Q and $\nu < \mu'' \leq \mu' \leq \mu$. □

Using Lemma 6.5 we can prove that a well-typed, closed process weakly terminates. Namely, that there exists a finite reduction sequence to done.

Lemma D.2 (weak termination). *If $\emptyset \vdash^{n} P$, then either $P \preccurlyeq \mathsf{done}$ or $P \Longrightarrow^{+} \mathsf{done}$.*

Proof. From Lemma B.3 we deduce that there exists $\mu \leq (n, 0)$ such that $\emptyset \vDash^{\mu} P$. We proceed doing an induction on the lexicographically ordered pair μ. From Lemma 6.5 we deduce either $P \preccurlyeq \mathsf{done}$ or $P \Longrightarrow^{+} Q$ and $\emptyset \vDash^{\nu} Q$ for some $\nu < \mu$. In the first case there is nothing left to prove. In the second case we use the induction hypothesis to deduce that either $Q \preccurlyeq \mathsf{done}$ or $Q \Longrightarrow^{+} \mathsf{done}$. We conclude using either [r-struct] or the transitivity of $\Longrightarrow^{+}$, respectively. □

The soundness of the type system is now a simple corollary.

Theorem 6.4. *If $\emptyset \vdash P$, then P is fairly terminating.*

Proof. Immediate consequence of Lemmas B.2 and D.2. □

E SUPPLEMENT TO SECTION 7

Below is the partial proof tree showing that $A\langle x, y\rangle$ is well typed. Each judgment is implicitly annotated with the rank 0:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{x : T_B \vdash B\langle x\rangle}\;[\textsc{t-call}]}{x : !S_A.T_B, y : S_A \vdash x!y.B\langle x\rangle}\;[\textsc{t-channel-out}]}{x : T_A, y : S_A \vdash x!\mathsf{a}.x!y.B\langle x\rangle}\;[\textsc{t-label}]}{x : T_A, y : S_A \vdash A\langle x, y\rangle}\;[\textsc{t-call}]$$

Below is the partial proof tree showing that $B\langle x\rangle$ is well typed. Each judgment is implicitly annotated with the rank 0:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{x : S_A, y : T_A \vdash A\langle y, x\rangle}\;[\textsc{t-call}]}{x : ?T_A.S_A \vdash x?(y).A\langle y, x\rangle}\;[\textsc{t-channel-in}] \qquad \dfrac{\dfrac{}{\emptyset \vdash \mathsf{done}}\;[\textsc{t-done}]}{x : ?\mathsf{end} \vdash \mathsf{wait}\ x.\mathsf{done}}\;[\textsc{t-wait}]}{x : T_B \vdash x?\{\mathsf{a} : x?(y).A\langle y, x\rangle, \mathsf{b} : \mathsf{wait}\ x.\mathsf{done}\}}\;[\textsc{t-label}]}{x : T_B \vdash B\langle x\rangle}\;[\textsc{t-call}]$$

Finally, here is the partial proof tree showing that the process shown in Eq. (11) is well typed:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{x : T_A, y : S_A \vdash^0 A\langle x, y\rangle}\;[\textsc{t-call}]}{x : S_A, y : S_A \vdash^1 \lceil x\rceil A\langle x, y\rangle}\;[\textsc{t-cast}] \qquad \dfrac{\dfrac{\vdots}{x : T_B \vdash^0 B\langle x\rangle}\;[\textsc{t-call}]}{x : S_A^\perp \vdash^1 \lceil x\rceil B\langle x\rangle}\;[\textsc{t-cast}]}{y : S_A \vdash^3 (x)(\lceil x\rceil A\langle x, y\rangle \mid \lceil y\rceil B\langle x\rangle)}\;[\textsc{t-par}] \qquad \dfrac{\dfrac{\vdots}{y : T_B \vdash^0 B\langle y\rangle}\;[\textsc{t-call}]}{y : S_A^\perp \vdash^1 \lceil y\rceil B\langle y\rangle}\;[\textsc{t-cast}]}{\emptyset \vdash^5 (y)((x)(\lceil x\rceil A\langle x, y\rangle \mid \lceil x\rceil B\langle x\rangle) \mid \lceil y\rceil B\langle y\rangle)}\;[\textsc{t-par}]$$

F ALGORITHMS

F.1 Minimum rank of a process

In this section we develop a function to compute the *minimum rank* of a process, namely the least quantity that is necessary in order to find a typing derivation for P. The function is defined below.

Definition F.1 (minimum rank of a process). The *minimum rank* of a process P, written $\|P\|$, is the least upper bound to the number of casts that P may need to perform and of sessions that P may need to create in order to terminate.

Formally, let $\|P\|_{\mathcal{A}}$ be the function inductively defined by the following equations, where $\mathcal{A}$ is a set of process names:

$$\begin{aligned}
\|\mathsf{done}\|_{\mathcal{A}} = \|\mathsf{close}\ x\|_{\mathcal{A}} &= 0 \\
\|A\langle\overline{x}\rangle\|_{\mathcal{A}} &= 0 && \text{if } A \in \mathcal{A} \\
\|A\langle\overline{x}\rangle\|_{\mathcal{A}} &= \|P\|_{\mathcal{A}\cup\{A\}} && \text{if } A \notin \mathcal{A} \text{ and } A(\overline{x}) \triangleq P \\
\|\pi.P\|_{\mathcal{A}} &= \|P\|_{\mathcal{A}} \\
\|\lceil x\rceil P\|_{\mathcal{A}} &= 1 + \|P\|_{\mathcal{A}} \\
\|(x)(P \mid Q)\|_{\mathcal{A}} &= 1 + \|P\|_{\mathcal{A}} + \|Q\|_{\mathcal{A}} \\
\|xp\{l_i : P_i\}_{i\in I}\|_{\mathcal{A}} &= \textstyle\bigsqcup_{i\in I} \|P_i\|_{\mathcal{A}} \\
\|P_1 \oplus_k P_2\|_{\mathcal{A}} &= \|P_k\|_{\mathcal{A}} && \text{if } k \in \{1,2\}
\end{aligned}$$

We write $\|P\|$ for $\|P\|_{\emptyset}$.

Now we have to show that this function allows us to compute the minimum rank that is necessary in a typing derivation. The first step is to provide a characterization of those processes that play a primary role in computing $\|\cdot\|$. We do so introducing an order relation $\sqsubseteq$ on processes.

Definition F.2 (termination path). Let $\sqsubseteq$ be the least preorder such that

$$\frac{k \in \{1,2\}}{P_k \sqsubseteq P_1 \oplus_k P_2} \qquad P \sqsubseteq \pi.P \qquad \frac{k \in I}{P_k \sqsubseteq xp\{l_i : P_i\}_{i\in I}} \qquad \frac{A(\overline{x}) \triangleq P}{P \sqsubseteq A\langle\overline{x}\rangle}$$

We say that P is on a *termination path* of Q if $P \sqsubseteq Q$.

Intuitively, $P \sqsubseteq Q$ means that the rank of Q may be affected by the rank of P, because P occurs along a path that leads Q to termination. Hereafter, we often omit the arguments of a process invocation involved in a process order relation and write, for example, $A \sqsubseteq P$ and $P \sqsubseteq A$ instead of $A\langle\overline{x}\rangle \sqsubseteq P$ and $P \sqsubseteq A\langle\overline{x}\rangle$. The notation $A \sqsubseteq P \sqsubseteq A$, shortcut for $A \sqsubseteq P$ and $P \sqsubseteq A$, means that P is found in between two invocations of the same definition A. These "loops" in termination paths are the dangerous places in which no casts can be performed and no sessions can be created.

Definition F.3 (safe program). We say that a program $\{A_i(\overline{x_i}) \triangleq P_i\}_{i\in I}$ is *safe* if $A_i \sqsubseteq P \sqsubseteq A_i$ implies that P is not a cast or a session for every P and $i \in I$.

Note that it is straightforward to define an algorithm that checks whether a program is safe, since the number of processes is finite and so is the number of process names.

Now we show that, in a safe program, the rank of a process invocation corresponds to that of its unfolding. This requires some auxiliary results that allow us to express the relationship between the ranks of a process depending on the set $\mathcal{A}$ or process names used. First, we show that the larger the set of process names, the smaller the rank. Intuitively, this is because in Definition F.1 every invocation of a process name that occurs in $\mathcal{A}$ results in a null rank.

Lemma F.4. *If $\mathcal{A} \subseteq \mathcal{B}$, then $\|P\|_{\mathcal{B}} \leq \|P\|_{\mathcal{A}}$.*

Proof. By induction on the definition of rank and by cases on the shape of P. We only discuss the base case in which $P = A\langle\overline{x}\rangle$, distinguishing three sub-cases: if $A \in \mathcal{A}$, then we conclude $\|P\|_{\mathcal{B}} = 0 = \|P\|_{\mathcal{A}}$; if $A \in \mathcal{B} \setminus \mathcal{A}$, then we conclude $\|P\|_{\mathcal{B}} = 0 \leq \|P\|_{\mathcal{A}}$; if $A \notin \mathcal{B}$, then we conclude $\|P\|_{\mathcal{B}} = \|Q\|_{\mathcal{B}\cup\{A\}} \leq \|Q\|_{\mathcal{A}\cup\{A\}} = \|P\|_{\mathcal{A}}$ using the induction hypothesis and $A(\overline{x}) \triangleq Q$. □

Next we show that the rank of a process P does not depend on the presence or absence of A in the set $\mathcal{A}$ if $A \not\sqsubseteq P$ if there is no invocation to A along any termination path of P.

Lemma F.5. *If $A \not\sqsubseteq P$, then $\|P\|_{\mathcal{A}} = \|P\|_{\mathcal{A} \cup \{A\}}$.*

Proof. By induction on the definition of $\|P\|_{\mathcal{A}}$ and by cases on the shape of P.

Cases $P = \mathsf{done}$ and $P = \mathsf{close}\ x$. We conclude $\|P\|_{\mathcal{A}} = \|P\|_{\mathcal{A} \cup \{A\}} = 0$.

Case $P = B\langle\overline{x}\rangle$ where $B(\overline{x}) \stackrel{\triangle}{=} Q$. From the hypothesis $A \not\sqsubseteq P$ we deduce $B \neq A$. We distinguish two sub-cases. If $B \in \mathcal{A}$, then we conclude

$$\begin{array}{rcll}
\|P\|_{\mathcal{A}} & = & \|B\langle\overline{x}\rangle\|_{\mathcal{A}} & \text{by definition of } P \\
& = & 0 & \text{by definition of } \|\cdot\| \\
& = & \|B\langle\overline{x}\rangle\|_{\mathcal{A} \cup \{A\}} & \text{by definition of } \|\cdot\| \\
& = & \|P\|_{\mathcal{A} \cup \{A\}} & \text{by definition of } P
\end{array}$$

If $B \notin \mathcal{A}$, then from the hypothesis $A \not\sqsubseteq P$ we deduce $A \not\sqsubseteq Q$ and we conclude

$$\begin{array}{rcll}
\|P\|_{\mathcal{A}} & = & \|B\langle\overline{x}\rangle\|_{\mathcal{A}} & \text{by definition of } P \\
& = & \|Q\|_{\mathcal{A} \cup \{B\}} & \text{by definition of } \|\cdot\| \\
& = & \|Q\|_{\mathcal{A} \cup \{A,B\}} & \text{using the induction hypothesis} \\
& = & \|B\langle\overline{x}\rangle\|_{\mathcal{A} \cup \{A\}} & \text{by definition of } \|\cdot\| \\
& = & \|P\|_{\mathcal{A} \cup \{A\}} & \text{by definition of } P
\end{array}$$

Case $P = P_1 \oplus_k P_2$ where $k \in \{1, 2\}$. From the hypothesis $A \not\sqsubseteq P$ we deduce $A \not\sqsubseteq P_k$. We conclude

$$\begin{array}{rcll}
\|P\|_{\mathcal{A}} & = & \|P_1 \oplus_k P_2\|_{\mathcal{A}} & \text{by definition of } P \\
& = & \|P_k\|_{\mathcal{A}} & \text{by definition of } \|\cdot\| \\
& = & \|P_k\|_{\mathcal{A} \cup \{A\}} & \text{using the induction hypothesis} \\
& = & \|P_1 \oplus_k P_2\|_{\mathcal{A} \cup \{A\}} & \text{by definition of } \|\cdot\| \\
& = & \|P\|_{\mathcal{A} \cup \{A\}} & \text{by definition of } P
\end{array}$$

Case $P = \pi.Q$ for some prefix π. From the hypothesis $A \not\sqsubseteq P$ we deduce $A \not\sqsubseteq Q$. We conclude

$$\begin{array}{rcll}
\|P\|_{\mathcal{A}} & = & \|\pi.Q\|_{\mathcal{A}} & \text{by definition of } P \\
& = & \|Q\|_{\mathcal{A}} & \text{by definition of } \|\cdot\| \\
& = & \|Q\|_{\mathcal{A} \cup \{A\}} & \text{using the induction hypothesis} \\
& = & \|\pi.Q\|_{\mathcal{A} \cup \{A\}} & \text{by definition of } \|\cdot\| \\
& = & \|P\|_{\mathcal{A} \cup \{A\}} & \text{by definition of } P
\end{array}$$

Case $P = \lceil x \rceil Q$. From $A \not\sqsubseteq P$ we deduce $A \not\sqsubseteq Q$. We conclude

$$\begin{array}{rcll}
\|P\|_{\mathcal{A}} & = & \|\lceil x \rceil Q\|_{\mathcal{A}} & \text{by definition of } P \\
& = & 1 + \|Q\|_{\mathcal{A}} & \text{by definition of } \|\cdot\| \\
& = & 1 + \|Q\|_{\mathcal{A} \cup \{A\}} & \text{using the induction hypothesis} \\
& = & \|\lceil x \rceil Q\|_{\mathcal{A} \cup \{A\}} & \text{by definition of } \|\cdot\| \\
& = & \|P\|_{\mathcal{A} \cup \{A\}} & \text{by definition of } P
\end{array}$$

Case $P = (x)(P_1 \mid P_2)$. From $A \not\sqsubseteq P$ we deduce $A \not\sqsubseteq P_i$ for $i = 1, 2$. We conclude

$$\begin{array}{rcll}
\|P\|_{\mathcal{A}} & = & \|(x)(P_1 \mid P_2)\|_{\mathcal{A}} & \text{by definition of } P \\
& = & 1 + \|P_1\|_{\mathcal{A}} + \|P_2\|_{\mathcal{A}} & \text{by definition of } \|\cdot\| \\
& = & 1 + \|P_1\|_{\mathcal{A}\cup\{A\}} + \|P_2\|_{\mathcal{A}\cup\{A\}} & \text{using the induction hypothesis} \\
& = & \|(x)(P_1 \mid P_2)\|_{\mathcal{A}\cup\{A\}} & \text{by definition of } \|\cdot\| \\
& = & \|P\|_{\mathcal{A}\cup\{A\}} & \text{by definition of } P
\end{array}$$

Case $P = xp\{l_i : P_i\}_{i\in I}$. From the hypothesis $A \not\sqsubseteq P$ we deduce $A \not\sqsubseteq P_i$ for every $i \in I$. We conclude

$$\begin{array}{rcll}
\|P\|_{\mathcal{A}} & = & \|xp\{l_i : P_i\}_{i\in I}\|_{\mathcal{A}} & \text{by definition of } P \\
& = & \bigsqcup_{i\in I} \|P_i\|_{\mathcal{A}} & \text{by definition of } \|\cdot\| \\
& = & \bigsqcup_{i\in I} \|P_i\|_{\mathcal{A}\cup\{A\}} & \text{using the induction hypothesis} \\
& = & \|xp\{l_i : P_i\}_{i\in I}\|_{\mathcal{A}\cup\{A\}} & \text{by definition of } \|\cdot\| \\
& = & \|P\|_{\mathcal{A}\cup\{A\}} & \text{by definition of } P
\end{array}$$

□

Finally, we can show that the rank of a process P such that $A \sqsubseteq P \sqsubseteq A$ cannot exceed the rank of A. Recall that $A \sqsubseteq P$ means that there is an invocation to A along a termination path of P and that $P \sqsubseteq A$ means that P occurs along a termination path of A.

Lemma F.6. *In a safe program, if* $A \sqsubseteq P \sqsubseteq A$ *and* $A(\overline{x}) \triangleq Q$, *then* $\|P\|_{\mathcal{A}} \leq \|P\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\|$.

Proof. By induction on $\|P\|_{\mathcal{A}}$ and by cases on the shape of P. Note that P cannot be a cast or a session because of the hypothesis that the program is safe.

Case $P = B\langle\overline{x}\rangle$ *and* $B \in \mathcal{A}$. We conclude

$$\begin{array}{rcll}
\|P\|_{\mathcal{A}} & = & \|B\langle\overline{x}\rangle\|_{\mathcal{A}} & \text{by definition of } P \\
& = & 0 & \text{by definition of } \|\cdot\| \\
& = & \|B\langle\overline{x}\rangle\|_{\mathcal{A}\cup\{A\}} & \text{by definition of } \|\cdot\| \\
& \leq & \|B\langle\overline{x}\rangle\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{property of } \sqcup \\
& = & \|P\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{by definition of } P
\end{array}$$

Case $P = A\langle\overline{y}\rangle$ *and* $A \notin \mathcal{A}$. We may assume, without loss of generality, that $\overline{y} = \overline{x}$ since the rank of a process invocation does *not* depend on channel names. We conclude

$$\begin{array}{rcll}
\|P\|_{\mathcal{A}} & = & \|A\langle\overline{x}\rangle\|_{\mathcal{A}} & \text{by definition of } P \\
& = & \|Q\|_{\mathcal{A}\cup\{A\}} & \text{by definition of } \|\cdot\| \\
& \leq & \|Q\|_{\{A\}} & \text{by Lemma F.4} \\
& = & \|A\langle\overline{x}\rangle\| & \text{by definition of } \|\cdot\| \\
& = & 0 \sqcup \|A\langle\overline{x}\rangle\| & \text{property of } \sqcup \\
& = & \|A\langle\overline{x}\rangle\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{by definition of } \|\cdot\| \\
& = & \|P\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{by definition of } P
\end{array}$$

Case $P = B\langle\overline{x}\rangle$ and $B \notin \mathcal{A} \cup \{A\}$ where $B(\overline{x}) \triangleq R$. From the hypotheses $A \sqsubseteq P \sqsubseteq A$ and $B \notin \mathcal{A} \cup \{A\}$ we deduce $A \sqsubseteq R \sqsubseteq A$.

$$\begin{array}{rcll}
\|P\|_{\mathcal{A}} & = & \|B\langle\overline{x}\rangle\|_{\mathcal{A}} & \text{by definition of } P \\
& = & \|R\|_{\mathcal{A}\cup\{B\}} & \text{by definition of } \|\cdot\| \\
& \leq & \|R\|_{\mathcal{A}\cup\{A,B\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{using the induction hypothesis} \\
& = & \|B\langle\overline{x}\rangle\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{by definition of } \|\cdot\| \text{ and the hypothesis } B \notin \mathcal{A} \cup \{A\} \\
& = & \|P\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{by definition of } P
\end{array}$$

Case $P = P_1 \oplus_k P_2$ where $k \in \{1, 2\}$. From the hypotheses $A \sqsubseteq P \sqsubseteq A$ we deduce $A \sqsubseteq P_k \sqsubseteq A$. We conclude

$$\begin{array}{rcll}
\|P\|_{\mathcal{A}} & = & \|P_1 \oplus_k P_2\|_{\mathcal{A}} & \text{by definition of } P \\
& = & \|P_k\|_{\mathcal{A}} & \text{by definition of } \|\cdot\| \\
& \leq & \|P_k\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{using the induction hypothesis} \\
& = & \|P_1 \oplus_k P_2\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{by definition of } \|\cdot\| \\
& = & \|P\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{by definition of } P
\end{array}$$

Case $P = \pi.Q$. From the hypotheses $A \sqsubseteq P \sqsubseteq A$ we deduce $A \sqsubseteq Q \sqsubseteq A$. We conclude

$$\begin{array}{rcll}
\|P\|_{\mathcal{A}} & = & \|\pi.Q\|_{\mathcal{A}} & \text{by definition of } P \\
& = & \|Q\|_{\mathcal{A}} & \text{by definition of } \|\cdot\| \\
& \leq & \|Q\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{using the induction hypothesis} \\
& = & \|\pi.Q\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{by definition of } \|\cdot\| \\
& = & \|P\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{by definition of } P
\end{array}$$

Case $P = xp\{l_i : P_i\}_{i\in I}$. For every $i \in I$ we distinguish two possibilities, depending on whether $A \not\sqsubseteq P_i$ or $A \sqsubseteq P_i$. In the first case we deduce $\|P_i\|_{\mathcal{A}} = \|P_i\|_{\mathcal{A}\cup\{A\}}$ using Lemma F.5. In the second case we deduce $\|P_i\|_{\mathcal{A}} \leq \|P_i\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\|$ using the induction hypothesis. Either way, we have $\|P_i\|_{\mathcal{A}} \leq \|P_i\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\|$ for every $i \in I$. We conclude

$$\begin{array}{rcll}
\|P\|_{\mathcal{A}} & = & \|xp\{l_i : P_i\}_{i\in I}\|_{\mathcal{A}} & \text{by definition of } P \\
& = & \bigsqcup_{i\in I} \|P_i\|_{\mathcal{A}} & \text{by definition of } \|\cdot\| \\
& \leq & \bigsqcup_{i\in I}(\|P_i\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\|) & \text{using Lemma F.5 or the induction hypothesis} \\
& = & (\bigsqcup_{i\in I} \|P_i\|_{\mathcal{A}\cup\{A\}}) \sqcup \|A\langle\overline{x}\rangle\| & \text{distributivity of } \sqcup \\
& = & \|xp\{l_i : P_i\}_{i\in I}\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{by definition of } \|\cdot\| \\
& = & \|P\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{by definition of } P
\end{array}$$
□

Next is the key lemma stating that the given notion of rank (Definition F.1) behaves well, in the sense that it is preserved by unfolding a process invocation.

Lemma F.7. *In a safe program such that $A(\overline{x}) \triangleq P$ we have $\|P\| = \|A\langle\overline{x}\rangle\|$.*

Proof. Clearly $P \sqsubseteq A$ since P is the body of the definition of process A. We distinguish two sub-cases that cover all possibilities. If $A \not\sqsubseteq P$, then using Lemma F.5 we deduce $\|P\| = \|P\|_{\emptyset} = \|P\|_{\{A\}} = \|A\langle\overline{x}\rangle\|$. If $A \sqsubseteq P$, then using Lemma F.5 we deduce $\|P\| = \|P\|_{\emptyset} \leq \|P\|_{\{A\}} \sqcup \|A\langle\overline{x}\rangle\| = \|A\langle\overline{x}\rangle\| \sqcup \|A\langle\overline{x}\rangle\| = \|A\langle\overline{x}\rangle\|$. Using Lemma F.4 we conclude $\|A\langle\overline{x}\rangle\| = \|P\|_{\{A\}} \leq \|P\|$. □

In order to show that $\|P\|$ is indeed the minimum rank of P that allows us to find a typing derivation for P, provided there is one, the first thing to do is to prove that a well-typed program is also safe. So from now on, when we reason about well-typed processes, we may assume that they belong to a safe program.

Lemma F.8. *A well-typed program is safe.*

Proof. First of all observe that $\Gamma \vdash^m P$ and $\Delta \vdash^n Q$ and $P \sqsubseteq Q$ imply $m \leq n$. This follows by considering the base cases of $P \sqsubseteq Q$ and looking at the typing rules in which Q is in the conclusion and P is one of the premises. Then, $A \sqsubseteq P \sqsubseteq A$ implies that P occurs in a typing judgment having exactly the same rank annotation as that of A. It follows that P cannot be a cast or a session, for these forms strictly increase the rank annotation. □

Next we show that $\|P\|$ is no greater than any rank that may appear is a typing derivation for P.

Lemma F.9. *If $\Gamma \vdash^n P$, then $\|P\| \leq n$.*

Proof. We prove that $\Gamma \vdash^n P$ implies $\|P\|_{\mathcal{A}} \leq n$ by a straightforward induction on the definition of $\|P\|_{\mathcal{A}}$. The conclusion $\|P\| \leq n$ is then just the particular case when $\mathcal{A} = \emptyset$. □

Finally, we show that if a program is well typed under *some* assignment then it is also well typed under the assignment that uses the minimum ranks. With Lemma F.9, this result justifies the definition of $\|P\|$ as *minimum rank* of P.

Theorem F.10. *If $\{A_i\langle\overline{x_i}\rangle \stackrel{\triangle}{=} P_i\}_{i\in I}$ is well typed under the global assignment $\{A_i : [\overline{S_i}; n_i]\}_{i\in I}$, then it is well typed also under the global assignment $\{A_i : [\overline{S_i}; \|P_i\|]\}_{i\in I}$.*

Proof. First we use the coinduction principle to show that every judgment in $\mathcal{R} \stackrel{\text{def}}{=} \{\Gamma \vdash^m P \mid \Gamma \vdash^n_{\mathsf{coind}} P, \|P\| \leq m\}$ is the conclusion of a rule in Table 3 whose premises are also in $\mathcal{R}$. This allows us to deduce that $\Gamma \vdash^n_{\mathsf{coind}} P$ implies $\Gamma \vdash^{\|P\|}_{\mathsf{coind}} P$. Suppose $\Gamma \vdash^m P \in \mathcal{R}$. Then $\Gamma \vdash^n_{\mathsf{coind}} P$ and $\|P\| \leq m$. We reason by cases on the last rule used in the derivation of $\Gamma \vdash^n_{\mathsf{coind}} P$. We only discuss two representative cases.

Case [t-call]. Then $P = A_k\langle\overline{x}\rangle$ and $A_k(\overline{x}) \stackrel{\triangle}{=} Q$ and $\Gamma \vdash^{n'}_{\mathsf{coind}} Q$ for some $n' \leq n$ and some $k \in I$. From the definition of rank we deduce $\|P\| = \|Q\| = \|Q_k\|$ since the rank of a process does not depend on its free names. From Lemma F.9 we deduce $\|Q_k\| \leq n_k$. We conclude by observing that $m \geq n_k$ and that $\Gamma \vdash^m Q$ is the conclusion of [t-call].

Case [t-label]. Then $P = xp\{l_i : Q_i\}_{i\in I}$ and $\Gamma = \Delta, x : p\{l_i : S_i\}_{i\in K}$ and $\Delta, x : S_i \vdash^n_{\mathsf{coind}} Q_i$ for every $i \in I$. From the definition of rank we have $\|P\| = \bigsqcup_{i\in I} \|Q_i\|$. From $m \geq \|P\|$ we deduce $m \geq \|Q_i\|$ for every $i \in I$. Then $\Delta, x : S_i \vdash^m Q_i \in \mathcal{R}$ for every $i \in I$ by definition of $\mathcal{R}$ and we conclude by observing that $\Gamma \vdash^m P$ is the conclusion of [t-label].

To show that $\Gamma \vdash^n_{\mathsf{ind}} P$ implies $\Gamma \vdash^{\|P\|}_{\mathsf{ind}} P$ it suffices a straightforward induction on the derivation of $\Gamma \vdash^n_{\mathsf{ind}} P$. By the bounded coinduction principle this is enough to conclude. □

F.2 Decidability of type checking

In this section we show how to obtain an alternative version of the typing rules from which it is easy to derive a type checking algorithm, provided that bound names and casts are explicitly annotated with session types. There are three aspects that make the type system presented in Table 3 not strictly algorithmic: (1) the fact that typing derivations are potentially infinite; (2) the need for building finite derivations using the corules [co-choice] and [co-label], which overlap with [t-choice] and [t-label] respectively (3) the rank annotation to be used in each typing judgment.

$$\text{[A-DONE]}\;\frac{}{\emptyset \vdash \mathsf{done}} \qquad \text{[A-CALL]}\;\frac{}{\overline{x : S} \vdash A\langle\overline{x}\rangle}\;A : [\overline{S}; n] \qquad \text{[A-CHOICE]}\;\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \oplus Q} \qquad \text{[A-CLOSE]}\;\frac{}{x : !\mathsf{end} \vdash \mathsf{close}\ x}$$

$$\text{[A-WAIT]}\;\frac{\Gamma \vdash P}{\Gamma, x : ?\mathsf{end} \vdash \mathsf{wait}\ x.P} \qquad \text{[A-CHANNEL-IN]}\;\frac{\Gamma, x : S, y : T \vdash P}{\Gamma, x : ?T.S \vdash x?(y).P} \qquad \text{[A-LABEL]}\;\frac{\Gamma, x : S_i \vdash P_i \ ^{(i \in I)}}{\Gamma, x : p\{l_i : S_i\}_{i \in I} \vdash xp\{l_i : P_i\}_{i \in I}}$$

$$\text{[A-CHANNEL-OUT]}\;\frac{\Gamma, x : S \vdash P}{\Gamma, x : !T.S, y : T \vdash x!y.P} \qquad \text{[A-PAR]}\;\frac{\Gamma, x : S \vdash P \qquad \Delta, x : T \vdash Q}{\Gamma, \Delta \vdash (x)(P \mid Q)}\;S \sim T \qquad \text{[A-CAST]}\;\frac{\Gamma, x : T \vdash P}{\Gamma, x : S \vdash \lceil x \rceil P}\;S \leqslant T$$

Table 4. Algorithmic typing rules for processes.

Concerning Item 3, in Appendix F.1 we have seen how the rank annotation can be computed for any process in a safe program. So here we focus on Items 1 and 2.

Table 4 presents an (inductively interpreted) set of typing rules that are a "stripped down" version of those given in Table 3. There are two main differences between these rules and those given in the main body of the paper: first, there is no rank annotation in typing judgments; second, [A-CALL] is an *axiom*, unlike [T-CALL]. The remaining structure of the rules and the kind of constraints they impose is exactly the same as before. Henceforth, we write $\Gamma \vdash_{\mathsf{alg}} P$ if $\Gamma \vdash P$ is (inductively) derivable using the typing rules in Table 4.

Lemma F.11. *Let $\{A_i(\overline{x_i}) \triangleq P_i\}_{i \in I}$ be a safe program and $\{A_i : [\overline{S_i}; n_i]\}_{i \in I}$ be a global assignment. The following properties are equivalent:*

(1) $\overline{x_i : S_i} \vdash^{n_i}_{\mathsf{coind}} P_i$ for every $i \in I$;
(2) $\overline{x_i : S_i} \vdash_{\mathsf{alg}} P_i$ for every $i \in I$.

Proof. 1 ⟹ 2. Just observe that a (finite) derivation for $\Gamma \vdash_{\mathsf{alg}} P$ can be obtained from a (possibly infinite) derivation for $\Gamma \vdash_{\mathsf{coind}} P$ by truncating each application of [T-CALL] in the latter derivation to an application of [A-CALL] with the same conclusion.

2 ⟹ 1. Let $\mathcal{R} \overset{\text{def}}{=} \{\Gamma \vdash^n P \mid \Gamma \vdash_{\mathsf{alg}} P, \|P\| \leq n\}$. Using the coinduction principle it suffices to show that each judgment found in $\mathcal{R}$ is the conclusion of a rule in Table 3 whose premises are also in $\mathcal{R}$. Let $\Gamma \vdash^n P \in \mathcal{R}$, meaning that $\Gamma \vdash_{\mathsf{alg}} P$ and $\|P\| \leq n$. We reason by cases on the rule used to derive $\Gamma \vdash_{\mathsf{alg}} P$. We only discuss a few cases.

Case [A-DONE]. We conclude observing that P is the conclusion of [T-DONE].

Case [A-CALL]. Then $P = A\langle\overline{x}\rangle$ for some $A(\overline{x}) \triangleq Q$. Note that $n \geq \|P\| = \|Q\|$. From the hypothesis we know that $\Gamma \vdash_{\mathsf{alg}} Q$ and we conclude by observing that $\Gamma \vdash^n P$ is the conclusion of [T-CALL] and that $\Gamma \vdash^n Q \in \mathcal{R}$ by definition of $\mathcal{R}$.

Case [A-PAR]. Then $P = (x)(P_1 \mid P_2)$ and $\Gamma = \Gamma_1, \Gamma_2$ and $\Gamma_i, x : S_i \vdash_{\mathsf{alg}} P_i$ for $i = 1, 2$ and $S_1 \sim S_2$. Note that $n \geq \|P\| = 1 + \|P_1\| + \|P_2\|$. Hence, there exist n_1 and n_2 such that $n = 1 + n_1 + n_2$ and $\|P_i\| \leq n_i$ for $i = 1, 2$. We conclude by observing that $\Gamma \vdash^n P$ is the conclusion of [T-PAR] and that $\Gamma_i, x : S_i \vdash^{n_i} P_i \in \mathcal{R}$ by definition of $\mathcal{R}$.

Case [A-LABEL]. Then $P = xp\{l_i : P_i\}_{i \in I}$ and $\Gamma = \Gamma', x : p\{l_i : S_i\}_{i \in I}$ and $\Gamma', x : S_i \vdash_{\mathsf{alg}} P_i$ for every $i \in I$. Note that $n \geq \|P\| = \bigsqcup_{i \in I} \|P_i\|$, hence $n \geq \|P_i\|$ for every $i \in I$. We conclude by observing that $\Gamma \vdash^n P$ is the conclusion of [T-LABEL] and that $\Gamma', x : S_i \vdash^n P_i \in \mathcal{R}$ by definition of $\mathcal{R}$. □

$$\frac{}{\mathcal{A} \Vdash \mathsf{done}} \qquad \frac{}{\mathcal{A} \Vdash \mathsf{close}\ x} \qquad \frac{\mathcal{A} \cup \{A\} \Vdash P}{\mathcal{A} \Vdash A\langle\overline{x}\rangle}\ A \notin \mathcal{A}, A(\overline{x}) \triangleq P \qquad \frac{\mathcal{A} \Vdash P_k}{\mathcal{A} \Vdash P_1 \oplus_k P_2}\ k \in \{1,2\}$$

$$\frac{\mathcal{A} \Vdash P}{\mathcal{A} \Vdash \pi.P} \qquad \frac{\mathcal{A} \Vdash P_k}{\mathcal{A} \Vdash xp\{l_i : P_i\}_{i\in I}}\ k \in I \qquad \frac{\mathcal{A} \Vdash P_i\ {}^{(i=1,2)}}{\mathcal{A} \Vdash (x)(P_1 \mid P_2)} \qquad \frac{\mathcal{A} \Vdash P}{\mathcal{A} \Vdash \lceil x \rceil P}$$

Table 5. Algorithmic rules for action boundedness.

To filter out those judgments derivable in the algorithmic type system for which there is no finite derivation using the original type system with the corules [CO-CHOICE] and [CO-LABEL], we separately define the (inductive) inference system shown in Table 5 for action boundedness. Note that this inference system can be trivially turned into an algorithm by checking whether, for a process of the form $xp\{l_i : P_i\}_{i\in I}$, there is at least one branch for which $\mathcal{A} \Vdash P_i$ is derivable.

LEMMA F.12. *If $\Gamma \vdash^n_{\mathsf{coind}} P$, then $\Gamma \vdash^n_{\mathsf{ind}} P$ if and only if $\emptyset \Vdash P$.*

PROOF. For the "if" part we prove that $\mathcal{A} \Vdash P$ implies $\Gamma \vdash^{\mathcal{A}}_{\mathsf{ind}} P$ by induction on the derivation of $\mathcal{A} \Vdash P$. For the "only if", we first prove that if $\Gamma \vdash^{A}_{\mathsf{ind}} P$ and none of the process names occurring in the [T-CALL] applications of this derivation is in $\mathcal{A}$, then $\mathcal{A} \Vdash P$. Then, the result follows by considering the *smallest* derivation $\Gamma \vdash^{A}_{\mathsf{ind}} P$, in which no process definition is expanded twice. □

THEOREM F.13. *Let $\{A_i(\overline{x_i}) \triangleq P_i\}_{i\in I}$ be a safe program and $\{A_i : [\overline{S_i}; n_i]\}_{i\in I}$ be a global assignment. The following properties are equivalent:*

(1) $\overline{x_i : S_i} \vdash^{n_i} P_i$ for every $i \in I$;
(2) $\overline{x_i : S_i} \vdash_{\mathsf{alg}} P_i$ for every $i \in I$ and $\emptyset \Vdash Q$ is derivable for every Q occurring in the derivations.

PROOF. Consequence of Lemmas F.11 and F.12. □

F.3 Cast inference

Here we show that the regions of a process in which casts may be necessary can be automatically inferred. The way we do this is by means of a $\mathsf{recast}(P, A)$ function that computes a process that is structurally similar to P, except that all casts already present in P have been removed and new casts have been inserted. The function $\mathsf{recast}(P, A)$ is inductively defined by the equations

$$\begin{array}{rcl}
\mathsf{recast}(\lceil x \rceil P, A) & = & \mathsf{recast}(P, A) \\
\mathsf{recast}(\bigoplus_{i=1,2} P_i, A) & = & \bigoplus_{i=1,2} \begin{cases} \mathsf{recast}(P_i, A) & \text{if } A \sqsubseteq P_i \\ \lceil \overline{y} \rceil \mathsf{recast}(P_i, A) & \text{if } A \not\sqsubseteq P_i \text{ and } \{\overline{y}\} = \mathsf{fn}(P_i) \end{cases} \\
\mathsf{recast}(xp\{l_i : P_i\}_{i\in I}, A) & = & xp \left\{ \begin{array}{ll} l_i : \mathsf{recast}(P_i, A) & \text{if } A \sqsubseteq P_i \\ l_i : \lceil \overline{y} \rceil \mathsf{recast}(P_i, A) & \text{if } A \not\sqsubseteq P_i \text{ and } \{\overline{y}\} = \mathsf{fn}(P_i) \end{array} \right\}_{i\in I}
\end{array}$$

and extended homomorphically to all the remaining process forms. We also extend the function to process definitions, as shown below:

$$\mathsf{recast}(A(\overline{x}) \triangleq P) \overset{\text{def}}{=} \begin{cases} A(\overline{x}) \triangleq \mathsf{recast}(P, A) & \text{if } A \sqsubseteq P \\ A(\overline{x}) \triangleq \lceil \overline{x} \rceil \mathsf{recast}(P, A) & \text{if } A \not\sqsubseteq P \end{cases}$$

Since $\mathsf{recast}(P, A)$ erases any cast that may be present in P, it can be applied to a process *without* casts to produce another process in which casts that *may* be necessary have been automatically inserted. Note that such insertion does not specify the type associated with the cast.

Let us now show that $\mathsf{recast}(P, A)$ is well typed if so is P.

Lemma F.14. *If $\Gamma \vdash_{\mathsf{alg}} P$ and $P \sqsubseteq A$, then there exists Δ such that $\Gamma \leqslant \Delta$ and $\Delta \vdash_{\mathsf{alg}} \mathsf{recast}(P, A)$ and $A \sqsubseteq P$ implies $\Gamma = \Delta$.*

Proof. By induction on the derivation of $\Gamma \vdash_{\mathsf{alg}} P$ and by cases on the last rule applied. We only consider two interesting cases.

Case [a-cast]. Then $P = \lceil x \rceil Q$ for some x and Q. We deduce that there exist Γ', S and T such that $\Gamma = \Gamma', x : S$ and $S \leqslant T$ and $\Gamma', x : T \vdash_{\mathsf{alg}} Q$. Using the induction hypothesis we deduce that there exist Δ' and T' such that $\Delta', x : T' \vdash_{\mathsf{alg}} \mathsf{recast}(Q, A)$ and $\Gamma' \leqslant \Delta'$ and $T \leqslant T'$. From the assumption that the program is safe and the hypothesis $P \sqsubseteq A$ we deduce that $A \not\sqsubseteq P$. We conclude by taking $\Delta \overset{\text{def}}{=} \Delta', x : T'$ observing that $\Gamma = \Gamma', x : S \leqslant \Gamma', x : T \leqslant \Delta', x : T' = \Delta$ and that $\mathsf{recast}(P, A) = \mathsf{recast}(Q, A)$.

Case [a-label] *when $A \sqsubseteq P$*. Then $P = x p\{l_i : P_i\}_{i \in I}$. We deduce that there exist Γ' and a family $S_{i \in I}$ such that $\Gamma = \Gamma', x : p\{l_i : S_i\}_{i \in I}$ and $\Gamma', x : S_i \vdash_{\mathsf{alg}} P_i$ for every $i \in I$. Using the induction hypothesis we deduce that, for every $i \in I$, we have $\Delta_i, x : T_i \vdash_{\mathsf{alg}} \mathsf{recast}(P_i, A)$ for some Δ_i and T_i such that $\Gamma' \leqslant \Delta_i$ and $S_i \leqslant T_i$. Furthermore, $A \sqsubseteq P_i$ implies $S_i = T_i$ and $\Gamma' = \Delta_i$. Using repeated applications of [a-cast] we derive $\Gamma', x : S_i \vdash_{\mathsf{alg}} \lceil \overline{y} \rceil \mathsf{recast}(P_i, A)$ where $\{\overline{y}\} = \mathsf{fn}(P_i)$ for every $i \in I$ such that $A \not\sqsubseteq P_i$. We conclude $\Gamma \vdash_{\mathsf{alg}} \mathsf{recast}(P, A)$ with one application of [a-label]. □

Finally, we can prove that the recasting function preserves typing:

Theorem F.15. *If $\Gamma \vdash_{\mathsf{alg}} P$, then $\Gamma \vdash_{\mathsf{alg}} \mathsf{recast}(P, A)$.*

Proof. Straightfoward consequence of Lemma F.14. In the case $A \not\sqsubseteq P$, it may be necessary to use [a-cast] to derive the final judgment with the same Γ occurring in the original judgment. □

F.4 Type reconstruction

We now present a type reconstruction algorithm that infers the session types of channels of a process in which there is no type annotation. The algorithm is based on the *co-contextual formulation of the typing rules*, a technique discussed by Erdweg et al. [2015] and used in many others type inference algorithms. The algorithm works with a finite syntax for session types enriched with *session type variables* X, of which we assume to have an infinite supply. The co-contextual typing rules derive judgments of the form $P \triangleright \Gamma; \mathfrak{C}$ where $\mathfrak{C}$ is a finite set of *constraints* of the form

$$S \mathrel{\hat{=}} T \qquad \text{or} \qquad S \mathrel{\hat{\leqslant}} T \qquad \text{or} \qquad S \mathrel{\hat{\sim}} T$$

respectively representing type equality constraints, subtyping constraints and compatibility constraints. Note that S and T are finite session types possibly using session type variables, even if we use the same meta-variables S and T that stand for session types in the rest of the paper. The intuitive meaning of the judgment $P \triangleright \Gamma; \mathfrak{C}$ is that P is well typed in Γ provided that the constraints $\mathfrak{C}$ are "solvable", in a sense that we make precise shortly.

The co-contextual typing rules are shown in Table 6. We do not discuss each rule in detail, since the rules are in one-to-one correspondence with those of the (algorithmic) type system. The general idea is to use session type variables wherever the process does not provide sufficient information to (completely) infer the session type associated with a

$$\textsc{[i-done]}\ \dfrac{}{\mathsf{done} \triangleright \emptyset; \emptyset} \qquad \textsc{[i-call]}\ \dfrac{}{A\langle\overline{x}\rangle \triangleright \overline{x : X}; \emptyset}\ A : [\overline{X}; n] \qquad \textsc{[i-choice]}\ \dfrac{P \triangleright \overline{x : S}; \mathfrak{C} \qquad Q \triangleright \overline{x : T}; \mathfrak{C}'}{P \oplus Q \triangleright \overline{x : S}; \mathfrak{C} \cup \mathfrak{C}' \cup \{\overline{S \mathrel{\hat{=}} T}\}}$$

$$\textsc{[i-cast]}\ \dfrac{P \triangleright \Gamma, x : S; \mathfrak{C}}{\lceil x \rceil P \triangleright \Gamma, x : X; \mathfrak{C} \cup \{X \mathrel{\hat{\leqslant}} S\}}\ X \text{ fresh} \qquad \textsc{[i-channel-out]}\ \dfrac{P \triangleright \Gamma, x : S; \mathfrak{C}}{x!y.P \triangleright \Gamma, x : !X.S, y : X; \mathfrak{C}}\ X \text{ fresh}$$

$$\textsc{[i-close]}\ \dfrac{}{\mathsf{close}\ x \triangleright x : !\mathsf{end}; \emptyset} \qquad \textsc{[i-label]}\ \dfrac{P_i \triangleright \overline{y : T_i}, x : S_i; \mathfrak{C}_i \ ^{(i \in I)}}{xp\{l_i : P_i\}_{i \in I} \triangleright \overline{y : X}, x : p\{l_i : S_i\}_{i \in I}; \bigcup_{i \in I} \mathfrak{C}_i \cup \{\overline{X \mathrel{\hat{=}} T_i} \mid i \in I\}}\ \overline{X} \text{ fresh}$$

$$\textsc{[i-wait]}\ \dfrac{P \triangleright \Gamma; \mathfrak{C}}{\mathsf{wait}\ x.P \triangleright \Gamma, x : ?\mathsf{end}; \mathfrak{C}} \qquad \textsc{[i-channel-in]}\ \dfrac{P \triangleright \Gamma, x : S, y : T; \mathfrak{C}}{x?(y).P \triangleright \Gamma, x : ?T.S; \mathfrak{C}} \qquad \textsc{[i-par]}\ \dfrac{P \triangleright \Gamma, x : S; \mathfrak{C} \qquad Q \triangleright \Delta, x : T; \mathfrak{C}'}{(x)(P \mid Q) \triangleright \Gamma, \Delta; \mathfrak{C} \cup \mathfrak{C}' \cup \{S \mathrel{\hat{\sim}} T\}}$$

Table 6. Constraint generation for type reconstruction.

channel, and to accumulate constraints for any relation that is imposed by the typing rules in Table 3. We only point out that [I-PAR] makes use of dualized type variables $X^\perp$, which stand for the dual of the session type represented by X.

We now show that the constraint generator is sound and complete with respect to the algorithmic version of the type system. To this aim, we use σ to range over finite maps from session type variables to session types and we write $\sigma(S)$ for the session type obtained by replacing any variable X occurring in S with $\sigma(X)$. Of course we expect to have $\sigma(X^\perp) = \sigma(X)^\perp$. We also assume that $\sigma(S)$ is undefined if $X \notin \mathsf{dom}(\sigma)$ for some X occurring in S and we extend this notation to contexts, writing $\sigma(\Gamma)$ for the context obtained by applying σ to all the session types in Γ.

Definition F.16 (solution of a constraint set). We say that σ is a *solution* of $\mathfrak{C}$ if $S \mathrel{\hat{\leqslant}} T \in \mathfrak{C}$ implies $\sigma(S) \leqslant \sigma(T)$ and $S \mathrel{\hat{=}} T \in \mathfrak{C}$ implies $\sigma(S) = \sigma(T)$ and $S \mathrel{\hat{\sim}} T \in \mathfrak{C}$ implies $\sigma(S) \sim \sigma(T)$.

Now we can prove that, whenever $P \triangleright \Gamma; \mathfrak{C}$ is derivable and $\mathfrak{C}$ is solvable, we can obtain a derivation for $\Delta \vdash_{\mathsf{alg}} P$ where Δ is closely related to Γ.

THEOREM F.17 (SOUNDNESS). *If $P \triangleright \Gamma; \mathfrak{C}$ and σ is a solution of $\mathfrak{C}$, then $\sigma(\Gamma) \vdash_{\mathsf{alg}} P$.*

PROOF. By induction on the derivation of $P \triangleright \Gamma; \mathfrak{C}$ and by cases on the last rule applied. We omit the discussion [I-CHOICE], [I-CHANNEL-IN] and [I-CHANNEL-OUT] which are not substantially different from other cases.

Case [I-DONE]. Then $P = \mathsf{done}$ and $\Gamma = \emptyset$ and $\mathfrak{C} = \emptyset$. We conclude with one application of [A-DONE].

Case [I-CALL]. Then $P = A\langle\overline{x}\rangle$ and $\Gamma = \overline{x : X}$ and $\mathfrak{C} = \emptyset$. We conclude with one application of [A-CALL].

Case [I-WAIT]. Then $P = \mathsf{wait}\ x.Q$ and $\Gamma = \Delta, x : ?\mathsf{end}$ and $Q \triangleright \Delta; \mathfrak{C}$. Using the induction hypothesis we deduce that $\sigma(\Delta) \vdash_{\mathsf{alg}} Q$. We conclude with one application of [A-WAIT] by observing that $\sigma(\Gamma) = \sigma(\Delta, x : ?\mathsf{end}) = \sigma(\Delta), x : ?\mathsf{end}$.

Case [I-CLOSE]. Then $P = \mathsf{close}\ x$ and $\Gamma = x : !\mathsf{end}$ and $\mathfrak{C} = \emptyset$. We conclude with one application of [A-CLOSE].

Case [I-LABEL]. Then $P = xp\{l_i : P_i\}_{i \in I}$ and $\Gamma = \overline{y : X}, x : p\{l_i : S_i\}_{i \in I}$ and $\mathfrak{C} = \bigcup_{i \in I} \mathfrak{C}_i \cup \{\overline{X \mathrel{\hat{=}} T_i} \mid i \in I\}$ and $P_i \triangleright \overline{y : T_i}, x : S_i; \mathfrak{C}_i$ for every $i \in I$. From the hypothesis that σ is a solution of $\mathfrak{C}$ we deduce that $\overline{\sigma(X) = \sigma(T_i)}$ and σ is a solution of $\mathfrak{C}_i$ for every $i \in I$. Using the induction hypothesis we deduce that $\overline{y : \sigma(T_i)}, x : \sigma(S_i) \vdash_{\mathsf{alg}} P_i$ for every $i \in I$. We conclude with one application of [A-LABEL].

Case [i-par]. Then $P = (x)(P_1 \mid P_2)$ and $\Gamma = \Gamma_1, \Gamma_2$ and $\mathfrak{C} = \mathfrak{C}_1 \cup \mathfrak{C}_2 \cup \{S_1 \mathrel{\hat{\sim}} S_2\}$ and $P_i \triangleright \Gamma_i, x : S_i; \mathfrak{C}_i$ for $i = 1, 2$. From the hypothesis that σ is a solution of $\mathfrak{C}$ we deduce that σ is a solution of $\mathfrak{C}_1$ and $\mathfrak{C}_2$. Also, it must be the case that $\sigma(S_1) \sim \sigma(S_2)$. Using the induction hypothesis we deduce that $\sigma(\Gamma_i), x : \sigma(S_i) \vdash_{\mathsf{alg}} P_i$ for $i = 1, 2$. We conclude with one application of [a-par] observing that $\sigma(\Gamma) = \sigma(\Gamma_1, \Gamma_2) = \sigma(\Gamma_1), \sigma(\Gamma_2)$.

Case [i-cast]. Then $P = \lceil x \rceil Q$ and $\Gamma = \Delta, x : X$ and $\mathfrak{C} = \mathfrak{C}' \cup \{X \mathrel{\hat{\leqslant}} S\}$ and $Q \triangleright \Delta, x : S; \mathfrak{C}'$. From the hypothesis that σ is a solution of $\mathfrak{C}$ we deduce that σ is a solution of $\mathfrak{C}'$ and also that $\sigma(X) \leqslant \sigma(S)$. Using the induction hypothesis we deduce that $\sigma(\Delta), x : \sigma(S) \vdash_{\mathsf{alg}} P$. We conclude with one application of [a-cast] observing that $\sigma(\Gamma) = \sigma(\Delta), x : \sigma(X) \leqslant \sigma(\Delta), x : \sigma(S)$. □

We can also prove that type reconstruction is complete, namely that whenever $\Gamma \vdash_{\mathsf{alg}} P$ is derivable, there exists a derivation using the co-contextual typing rules that yields a solvable set of constraints.

Theorem F.18 (completeness). *If $\Gamma \vdash_{\mathsf{alg}} P$, then there exist Δ, $\mathfrak{C}$ and σ solution of $\mathfrak{C}$ such that $P \triangleright \Delta; \mathfrak{C}$ and $\Gamma = \sigma(\Delta)$.*

Proof. By induction on the derivation of $\Gamma \vdash_{\mathsf{alg}} P$ and by cases on the last rule applied. We omit the discussion of [a-choice], [a-channel-in], [a-channel-out] which are not substantially different from other cases.

Case [a-done]. Then $P = \mathsf{done}$ and $\Gamma = \emptyset$. We conclude by taking $\Delta \overset{\text{def}}{=} \emptyset$, $\mathfrak{C} \overset{\text{def}}{=} \emptyset$ and $\sigma \overset{\text{def}}{=} \emptyset$ with one application of [i-done].

Case [a-call]. Then $P = B\langle \overline{x} \rangle$ and $\Gamma = \overline{x : S}$ and $A : \overline{S}$. Let $\overline{X}$ be a tuple of fresh type variables with the same length of $\overline{S}$, let $\Delta \overset{\text{def}}{=} \overline{x : X}$, $\mathfrak{C} \overset{\text{def}}{=} \emptyset$ and $\sigma \overset{\text{def}}{=} \{\overline{X \mapsto S}\}$. We conclude with one application of [i-call] observing that $\Gamma = \overline{x : S} = \overline{x : \sigma(X)} = \sigma(\Delta)$.

Case [a-close]. Then $P = \mathsf{close}\ x$ and $\Gamma = x : !\mathsf{end}$. We conclude by taking $\Delta \overset{\text{def}}{=} x : !\mathsf{end}$, $\mathfrak{C} \overset{\text{def}}{=} \emptyset$ and $\sigma \overset{\text{def}}{=} \emptyset$ with one application of [i-close].

Case [a-wait]. Then $P = \mathsf{wait}\ x.Q$ and $\Gamma = \Gamma', x : ?\mathsf{end}$ and $\Gamma' \vdash_{\mathsf{alg}} Q$. Using the induction hypothesis we deduce that there exist Δ', $\mathfrak{C}$ and σ solution of $\mathfrak{C}$ such that $Q \triangleright \Delta; \mathfrak{C}$ and $\Gamma' = \sigma(\Delta')$. We conclude by taking $\Delta \overset{\text{def}}{=} \Delta', x : ?\mathsf{end}$ with one application of [i-wait] observing that $\Gamma = \Gamma', x : ?\mathsf{end} = \sigma(\Delta'), x : \sigma(?\mathsf{end})$.

Case [a-label]. Then $P = xp\{l_i : P_i\}_{i \in I}$ and $\Gamma = \overline{y : U}, x : S_i$ and $\overline{y : U}, x : S_i \vdash_{\mathsf{alg}} P_i$ for every $i \in I$. Using the induction hypothesis we deduce that there exist a family $\Delta_{i \in I}$, a family $T_{i \in I}$, a family $\mathfrak{C}_{i \in I}$ and a family $\sigma_{i \in I}$ such that σ_i is a solution of $\mathfrak{C}_i$ and $P_i \triangleright \Delta_i, x : T_i; \mathfrak{C}_i$ and $\overline{y : U} = \sigma_i(\Delta_i)$ and $S_i = \sigma_i(T_i)$ for every $i \in I$. Without loss of generality we may assume that the σ_i have pairwise disjoint domains for we have an infinite supply of session type variables. Let $\overline{X}$ be a tuple of as many fresh type variables as the number of channels in $\overline{y}$, let $\Delta \overset{\text{def}}{=} \{\overline{y : X}, x : p\{l_i : T_i\}_{i \in I}\}$ and $\mathfrak{C} \overset{\text{def}}{=} \bigcup_{i \in I} \mathfrak{C}_i \cup \{\overline{X \mathrel{\hat{=}} U_i} \mid i \in I\}$ and $\sigma = \bigcup_{i \in I} \sigma_i \cup \{\overline{X \mapsto U}\}$ and observe that σ is a solution of $\mathfrak{C}$. We conclude with one application of [i-label] observing that $\Gamma = \overline{y : U}, x : p\{l_i : S_i\}_{i \in I} = \overline{y : \sigma(X)}, x : p\{l_i : \sigma(T_i)\}_{i \in I} = \sigma(\Delta)$.

Case [a-par]. Then $P = (x)(P_1 \mid P_2)$ and $\Gamma = \Gamma_1, \Gamma_2$ and $\Gamma_i, x : S_i \vdash_{\mathsf{alg}} P_i$ for $i = 1, 2$ and $S_1 \sim S_2$. Using the induction hypothesis we deduce that there exist Δ_1, Δ_2, T_1, T_2, $\mathfrak{C}_1$, $\mathfrak{C}_2$, σ_1 solution of $\mathfrak{C}_1$ and σ_2 solution of $\mathfrak{C}_2$ such that $P_i \triangleright \Delta_i, x : T_i; \mathfrak{C}_i$ and $\Gamma_i = \sigma(\Delta_i)$ and $S_i = \sigma(T_i)$ for $i = 1, 2$. Without loss of generality, we may assume that σ_1 and σ_2 have disjoint domains for we have an infinite supply of type variables. Let $\Delta \overset{\text{def}}{=} \Delta_1, \Delta_2$ and $\mathfrak{C} \overset{\text{def}}{=} \mathfrak{C}_1 \cup \mathfrak{C}_2 \cup \{T_1 \mathrel{\hat{\sim}} T_2\}$ and $\sigma \overset{\text{def}}{=} \sigma_1 \cup \sigma_2$ and observe that σ is a solution of $\mathfrak{C}$. We conclude with one application of [i-par] observing that $\Gamma = \Gamma_1, \Gamma_2 = \sigma(\Delta_1), \sigma(\Delta_2) = \sigma(\Delta)$.

Case [a-cast]. Then $P = \lceil x \rceil Q$ and $\Gamma = \Gamma', x : S$ and $\Gamma', x : T \vdash_{\mathsf{alg}} Q$ and $S \leqslant T$. Using the induction hypothesis we deduce that there exist Δ', T', $\mathfrak{C}'$ and σ' solution of $\mathfrak{C}'$ such that $Q \triangleright \Delta', x : T'; \mathfrak{C}'$ and $\Gamma' = \sigma(\Delta')$ and $T = \sigma(T')$.

Let X be a fresh type variable, let $\Delta \overset{\text{def}}{=} \Delta', x : X$, $\mathfrak{C} \overset{\text{def}}{=} \mathfrak{C}' \cup \{X \hat{\leqslant} T'\}$ and $\sigma \overset{\text{def}}{=} \sigma' \cup \{X \mapsto S\}$. We conclude with one application of [I-CAST] observing that σ is a solution of $\mathfrak{C}$ and $\Gamma = \Gamma', x : S = \sigma(\Delta'), x : \sigma(X) = \sigma(\Delta)$. □

The type reconstruction algorithm must then be completed by defining a suitable solver for constraint sets. We leave this aspect to future work, noting that similar solvers (also for behavioral type systems) have already been defined.