From Lock Freedom to Progress Using Session Types

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

Inspired by Kobayashi's type system for lock freedom, we define a behavioral type system for ensuring progress in a language of binary sessions. The key idea is to annotate actions in session types with priorities representing the urgency with which such actions must be performed and to verify that processes perform such actions with the required priority. Compared to related systems for session-based languages, the presented type system is relatively simpler and establishes progress for a wider range of processes.

1 Introduction

A system has the progress property if it does not accumulate garbage (messages that are produced and never consumed) and does not have dead code (processes that wait for messages that are never produced). For *session-based* systems, where processes interact by means of *sessions* through disciplined interaction patterns described by *session types*, the type systems by Dezani-Ciancaglini *et al.* [9, 2, 7] guarantee that well-typed processes have progress. These type systems analyze the dependencies between different (possibly interleaved) sessions and establish progress if no circular dependency is found. In a different line of work [11], Kobayashi defines a type system ensuring a lock-freedom property closely related to progress. Despite the similarities between the notions of progress and lock-freedom, however, the type systems in [9, 2, 7] and the one in [11] are difficult to compare, because of several major differences in both processes and types. In particular, the type systems in [2, 7] are defined for an asynchronous language with a native notion of session, while Kobayashi's type system is defined for a basic variant of the synchronous, pure π-calculus.

The natural approach for comparing these analysis techniques would require compiling a (well-typed) "source" session-based process into a "target" π-calculus process, and then using Kobayashi's type system for reasoning on progress of the source process in terms of lock-freedom of the target one. The problem of such compilation schemes (see [8] for an example) is that they produce target processes in which the communication topology is significantly more complex than that of the corresponding source ones because of explicit continuation channel passing and encoding of recursion. The net effect is that many well-typed source processes become ill-typed according to [11]. In this work we put forward a different approach: we lift the technique underlying Kobayashi's type system to a session type system for reasoning directly on the progress properties of processes. The results are very promising, because the type system we obtain is simpler than the ones defined in [9, 2, 7] and at the same time is capable of proving progress for a wider range of processes. As a welcome side effect, the structure given by sessions allows us to simplify some technical aspects of Kobayashi's original type system as well.

To sketch the key ideas of Kobayashi's type system applied to sessions, consider the term

$$a^+?(x).b^-!\langle 4\rangle \mid b^+?(y).a^-!\langle 3\rangle \tag{1}$$

which represents the parallel composition of two processes that communicate through two distinct sessions named a and b. Each session is accessed via its two endpoints, which we represent as the name of

Submitted to:
PLACES 2013

the session decorated with a *polarity* $+$ or $-$, along the lines of [10]. We say that a^- is the peer of a^+, and vice versa. In (1), the process on the left hand side of $|$ is waiting for a message from endpoint a^+, after which it sends 4 over endpoint b^-. The process on the right hand side of $|$ instead is waiting for a message from endpoint b^+, after which it sends 3 over endpoint a^-. Notice that the message that is supposed to be received from a^+ is the one sent over a^-, and the message that is supposed to be received from b^+ is the one sent over b^-. Clearly, as each send operation is guarded by a receive, the term denotes a process without progress. In particular, there is a circular dependency between the actions pertaining the two sessions a and b.

The mechanism used for detecting these circular dependencies consists in associating each action with an ordered pair of *priorities*. For instance, the receive action on a^+ would be associated with the pair $\langle\alpha,\beta\rangle$ where the first component (α) measures the urgency to perform the action by the process using a^+ and the second component (β) measures the urgency to perform the complementary (send) action by the process using the peer endpoint a^-. Because a^- is the peer of a^+, it is understood that such send action will be associated with a pair that has exactly the same two components as $\langle\alpha,\beta\rangle$, but in reverse order, namely $\langle\beta,\alpha\rangle$. Similarly, the two actions on b^+ and b^- will be associated with two pairs $\langle\gamma,\delta\rangle$ and $\langle\delta,\gamma\rangle$. In the example, the two parallel processes are performing the receive actions on the endpoints a^+ and b^+ first, therefore complying with their respective duties no matter of how high the priorities α and γ are. The send operation on b^-, on the other hand, is guarded by the receive action on a^+ and will not be performed until this action is completed, namely until a message is sent over endpoint a^-. So, the left subprocess is complying with its duty to perform the send action on b^- with priority δ provided that such priority is lower than that (β) to perform the action on a^-. At the same time, by looking at the right subprocess, we deduce that such process is complying with its duty to perform the action on a^- with priority β provided that β is lower than δ (the priority associated with the action on b^-). Overall, we realize that the two constraints "δ lower than β" and "β lower than δ" are not simultaneously satisfiable for any choice of β and δ, which is consistent with the fact that the system makes no progress.

An even simpler example of process without progress is

$$a^+?(x).a^-!\langle x\rangle \tag{2}$$

where the input action on a^+ guards the very send action that should synchronize with it. If we respectively associate the actions on a^+ and a^- with the pairs of priorities $\langle\alpha,\beta\rangle$ and $\langle\beta,\alpha\rangle$ we see that the structure of the process gives rise to the constraint "β lower than β", which is clearly unsatisfiable.

In summary, the type system that we are going to present relies on pairs of priorities associated with each action in the system, and verifies whether the relations originating between these priorities as determined by the structure of processes are satisfiable. If this is the case, it can be shown that the system has progress. All it remains to understand is the role played by session types. In fact, in all the examples above we have associated priorities with actions occurring within processes. Since a session type system determines a one-to-one correspondence between actions occurring in processes and actions occurring in session types, we let such pairs of priorities be part of the session types themselves. For instance, the left process in (1) would be well typed in an environment with the associations $a^+ : \langle\alpha,\beta\rangle?\mathit{int}.\mathtt{end}$ and $b^- : \langle\delta,\gamma\rangle!\mathit{int}.\mathtt{end}$ provided that "β is lower than δ".

We continue the exposition by defining a calculus of binary sessions in Section 2. We purposely use a minimal set of supported features to ease the subsequent formal development but we will be a bit more liberal in the examples. The type system is defined in Section 3, which also includes the soundness proof. Section 4 discusses a few extensions that can be accommodated with straightforward adjustments to the type system. Section 5 discusses related work and concludes.

Table 1: Syntax of processes.

P	$::=$		**Process**
		$\mathbf{0}$	(idle)
	$\mid$	X	(variable)
	$\mid$	$u?(x).P$	(input)
	$\mid$	$u!\langle v\rangle.P$	(output)
	$\mid$	$P \mid Q$	(composition)
	$\mid$	$(\nu a)P$	(session)
	$\mid$	$\texttt{rec}^{[\iota]}\, X.P$	(recursion)

2 Language

Syntax. We begin by fixing a few conventions: we use m, n, ... to range over natural numbers; we use a, b, ... to range over (countably many) *channels*; we use p, q, ... to range over the *polarities* $+$ and $-$; we define an involution $\overline{\cdot}$ over polarities such that $\overline{\mp} = -$; *endpoints* a^+, a^-, ... are channels decorated with a polarity; we use x, y, ... to range over (countably many) *variables*; we use u, v, ... to range over *names*, which are either variables or endpoints; we use ι, ... to range over *indices*, which are either natural numbers or ∞; we let $\infty + 1 = \infty$ and we extend the usual total order $<$ over natural numbers to indices so that $n < \infty$ for every n; we use X, Y, ... to range over (countably many) *process variables*; we use P, Q, ... to range over processes.

The language we work with is a simple variant of the synchronous π-calculus equipped with binary sessions. Each session takes place on a private channel a, which is represented as two *peer endpoints* a^+ and a^- so that a message sent over one of the endpoints is received from its peer. The syntax of processes is defined by the grammar in Table 1 and briefly described in the following paragraphs. The term $\mathbf{0}$ denotes the idle process, which performs no actions. The term $u?(x).P$ denotes a process that waits for a message from endpoint u, binds the message to the variable x, and then behaves as P. The term $u!\langle v\rangle.P$ denotes a process that sends message v over the endpoint u and then continues as P. In the prefixes $u?(x)$ and $u!\langle v\rangle$ we call u the *subject*. The term $P \mid Q$ denotes the conventional parallel composition of P and Q. The term $(\nu a)P$ denotes a session on channel a that is private to P. Within P the session can be accessed through the two endpoints a^+ and a^-. Terms X and $\texttt{rec}^{[\iota]}\, X.P$ are used for building recursive processes. The only unusual feature here is the index ι which, when finite, sets an upper bound to the number of unfoldings of the recursive term.

A term $u?(x).P$ binds the variable x in P, a term $(\nu a)P$ binds the endpoints a^+ and a^- in P, and a term $\texttt{rec}^{[\iota]}\, X.P$ binds the process variable X in P. Then, $\mathsf{fn}(P)$ denotes the set of free names of a generic process P. Similarly for $\mathsf{fpv}(P)$, but for free process variables. We sometimes write $\prod_{i=1}^{n} P_i$ for the composition $P_1 \mid \cdots \mid P_n$ and $(\nu \tilde{a})P$ for $(\nu a_1)\cdots(\nu a_n)P$. We write $\mathbb{P}^{[\iota]}$ for the set of all processes such that every $\texttt{rec}$ occurring in them has an index no greater than ι and we let $\mathbb{P}^{[\mathsf{fin}]} = \bigcup_{n\in\mathbb{N}} \mathbb{P}^{[n]}$. We say that P is a *user process* if $P \in \mathbb{P}^{[\infty]} \setminus \mathbb{P}^{[\mathsf{fin}]}$. That is, user processes only allow for unbounded unfoldings of recursions. The remaining processes are only useful for proving soundness of the type system.

Reduction Semantics. The operational semantics of the calculus is expressed as usual as a combination of a structural congruence, which rearranges equivalent terms, and a reduction relation. Structural congruence is the least congruence that includes alpha renaming of bound names and process variables and the laws in Table 2. It is basically the same as that of the π-calculus, with the only exception

Table 2: Structural congruence for processes.

[S-PAR 1]	[S-PAR 2]	[S-PAR 3]	[S-RES 1]
$\mathbf{0} \mid P \equiv P$	$P \mid Q \equiv Q \mid P$	$P \mid (Q \mid R) \equiv (P \mid Q) \mid R$	$(\nu a)(\nu b)P \equiv (\nu b)(\nu a)P$

[S-RES 2]
$$\frac{a^+, a^- \notin \mathsf{fn}(Q)}{(\nu a)P \mid Q \equiv (\nu a)(P \mid Q)}$$

Table 3: Reduction of processes.

[R-COMM]
$$a^p!\langle c^q\rangle.P \mid a^{\overline{p}}?(x).Q \to P \mid Q\{c^q/x\}$$

[R-REC]
$$\texttt{rec}^{[\iota+1]}\, X.P \to P\{\texttt{rec}^{[\iota]}\, X.P/X\}$$

[R-RES]
$$\frac{P \to Q}{(\nu a)P \to (\nu a)Q}$$

[R-PAR]
$$\frac{P \to P'}{P \mid Q \to P' \mid Q}$$

[R-STRUCT]
$$\frac{P \equiv P' \qquad P' \to Q' \qquad Q' \equiv Q}{P \to Q}$$

of [S-RES 2] which changes the scope of *both endpoints* a^+ and a^- of a channel a. Reduction is the least relation defined by the rules in Table 3. It includes two axioms for communication [R-COMM] and recursion unfolding [R-REC], two context rules [R-RES] and [R-PAR], and a rule for reduction up to structural congruence [R-STRUCT]. Most rules are standard. In [R-COMM], a synchronization occurs only between two endpoints of the same channel with dual polarities and $Q\{c^q/x\}$ denotes the capture-avoiding substitution of endpoint c^q in place of the free occurrences of x within Q. Note, in particular, that $((\nu c)c^+!\langle x\rangle.\mathbf{0})\{c^-/x\}$ is undefined and that such a substitution is applicable only after alpha renaming the bound channel c by means of structural congruence. Unfolding of recursions is allowed only when the index associated with the recursive term is not zero. If different from ∞, the index is decremented by the unfolding. The notation $P\{Q/X\}$ denotes the capture-avoiding substitution of process Q in place of the free occurrences of the process variable X in P. For example, $((\nu a)X)\{a^+!\langle b^+\rangle/X\}$ is undefined.

In the following we write $\to^*$ for the reflexive, transitive closure of $\to$ and we say that P is in *normal form*, written $P \not\to$, if there is no Q such that $P \to Q$.

Progress. We conclude this section with the formalization of the progress property that we have alluded to in Section 1.

Definition 2.1 (progress)**.** We say that P has *progress* if:

1. $P \to^* (\nu\tilde{a})(a^p!\langle c^q\rangle.P' \mid Q)$ implies $Q \to^* (\nu\tilde{b})(a^{\overline{p}}?(x).Q' \mid R)$ where a does not occur in $\tilde{b}$;
2. $P \to^* (\nu\tilde{a})(a^p?(x).P' \mid Q)$ implies $Q \to^* (\nu\tilde{b})(a^{\overline{p}}!\langle c^q\rangle.Q' \mid R)$ where a does not occur in $\tilde{b}$.

Note in particular that our notion of progress differs from deadlock freedom in the sense that it is not sufficient for a process to be able to reduce in order for it to enjoy progress. For instance, $a^+!\langle b^-\rangle.\mathbf{0} \mid \texttt{rec}^{[\infty]}\, X.X$ does *not* have progress even if it admits an infinite sequence of reductions because the message b^- is never consumed (no prefix $a^-?(x)$ ever emerges).

Table 4: Syntax of session types.

T	$::=$		**Session Type**
		$\texttt{end}$	(termination)
	$\mid$	$\mathbf{t}$	(type variable)
	$\mid$	$\langle\alpha,\beta\rangle?S.T$	(input)
	$\mid$	$\langle\alpha,\beta\rangle!S.T$	(output)
	$\mid$	$\mu^{[\iota]}\mathbf{t}.T$	(recursion)

3 Session Types for Global Progress

Definitions. We use T, S, … to range over *session types*, $\mathbf{t}$, … to range over (countably many) *session type variables*, and α, β, … to range over *priorities*, which we concretely represent as natural numbers with the interpretation that "smaller number" means "higher priority", 0 denoting the highest priority. The syntax of session types is described by the grammar in Table 4. The term $\texttt{end}$ denotes an endpoint on which no further input/output operation is possible. The term $\langle\alpha,\beta\rangle?S.T$ denotes an endpoint that must be used with priority α for receiving a message of type S and according to T afterwards. Similarly, the term $\langle\alpha,\beta\rangle!S.T$ denotes an endpoint that must be used with priority α for sending a message of type S and according to T afterwards. Following Kobayashi [11], we sometimes call α *obligation* and β *capability*: the obligation α associated with an action of an endpoint expresses the *duty* to perform the action with priority α by the process owning the endpoint; the capability β associated with an action of an endpoint expresses the *guarantee* that the corresponding complementary action will be performed with priority β by the process owning the peer endpoint. Terms $\mathbf{t}$ and $\mu^{[\iota]}\mathbf{t}.T$ are used for building recursive session types, as usual. Like in processes, $\mu^{[\iota]}$'s are decorated with an index ι denoting the number of unfoldings allowed on this recursion, which is unbounded when $\iota = \infty$. The only binder for session type variables is μ, so the notions of free and bound type variables are as expected. We write $\mathsf{ftv}(T)$ for the set of free type variables of T.

We restrict session types to the terms generated by the grammar in Table 4 that satisfy the following conditions:

- there are no subterms of the form $\mu\mathbf{t}_1 \cdots \mu\mathbf{t}_n.\mathbf{t}_1$;
- the terms S in all prefixes $\langle\alpha,\beta\rangle?S$ and $\langle\alpha,\beta\rangle!S$ are *closed*.

The first condition ensures that session types are *contractive* and avoids meaningless terms such as $\mu\mathbf{t}.\mathbf{t}$. The second condition ensures that session types are *stratified* (a similar constraint can be found in [6, 1, 13]) and is imposed to simplify the notion of duality (Definition 3.1).

We take an *iso-recursive* point of view and distinguish between a recursive session type $\mu^{[\iota+1]}\mathbf{t}.T$ and its unfolding $T\{\mu^{[\iota]}\mathbf{t}.T/\mathbf{t}\}$, where $T\{S/\mathbf{t}\}$ denotes the capture-avoiding substitution of the free occurrences of $\mathbf{t}$ in T with S. Note that in the unfolding the index $\iota+1$ is decremented to ι, unless $\iota = \infty$ in which case it remains ∞.

A crucial notion of every theory of binary session types is that of *duality*, which relates the session types associated with the peer endpoints of a session. Informally, two session types are dual of each other if they specify complementary behaviors, whereby an input action with a message of type S in one session type is matched by an output action with a message of the same type in the dual session type. Formally, we define duality as follows:

Definition 3.1 (duality)**.** *Duality* is the least relation $\bowtie$ defined by the rules

$$\text{[D-END]}\ \ \texttt{end} \bowtie \texttt{end} \qquad \text{[D-VAR]}\ \ \mathbf{t} \bowtie \mathbf{t} \qquad \text{[D-PREFIX]}\ \frac{T \bowtie T'}{\langle \alpha,\beta\rangle ?S.T \bowtie \langle \beta,\alpha\rangle !S.T'} \qquad \text{[D-REC]}\ \frac{T \bowtie S}{\mu^{[\iota]}\mathbf{t}.T \bowtie \mu^{[\iota]}\mathbf{t}.S} \qquad \text{[D-UNFOLD]}\ \frac{T \bowtie \mu^{[\iota+1]}\mathbf{t}.S}{T \bowtie S\{\mu^{[\iota]}\mathbf{t}.S/\mathbf{t}\}}$$

plus the symmetric ones of [D-PREFIX] and [D-UNFOLD].

Rules [D-END], [D-VAR], and [D-REC] are standard from binary session type theories. Rule [D-PREFIX] is conventional except for the swapping of priorities decorating the actions that we have just discussed. Rule [D-UNFOLD] is necessary because our session types are iso-recursive. In particular, thanks to this rule we can derive that $\mu^{[\infty]}\mathbf{t}.\langle \alpha,\beta\rangle ?S.\mathbf{t} \bowtie \langle \beta,\alpha\rangle !S.\mu^{[\infty]}\mathbf{t}.\langle \beta,\alpha\rangle !S.\mathbf{t}$ where, in the second session type, we have unfolded the recursion once. This rule is necessary because the types associated with peer endpoints will in general be unfolded independently.

The judgments of the type system have the form $\Sigma;\Gamma;\Delta \vdash_\iota P$ where

$$\Delta ::= \emptyset \mid u : T \mid \Delta,\Delta \qquad \Gamma ::= \emptyset \mid X : \langle\Delta\rangle \mid \Gamma,\Gamma \qquad \Sigma ::= \emptyset \mid \mathbf{t} : \alpha \mid \Sigma,\Sigma$$

respectively define the *name environment* associating names u with session types T, the *process environment* Γ associating process variables X with name environments $\langle\Delta\rangle$, and the *type variable environment* Σ associating type variables $\mathbf{t}$ with priorities α. For all the environments we let $\mathsf{dom}(\cdot)$ be the function that returns their domain, we assume that composition through ‘,’ is defined only when the environments being composed have disjoint domains, and we identify environments modulo commutativity and associativity of ‘,’ and neutrality of $\emptyset$. We also write $\Gamma \setminus X$ for the restriction of Γ to $\mathsf{dom}(\Gamma)\setminus\{X\}$ and $\Gamma + X : \langle\Delta\rangle$ for $(\Gamma\setminus X), X : \langle\Delta\rangle$. Similarly for Σ.

We will need to compute the obligation of a type, which measures the urgency with which a value having that type must be used. Intuitively, the obligation of a session type T is given by the obligation of its topmost action. This leaves open the question as to what is the priority of T if T has no topmost action, in particular when T is `end` or a type variable. In the former case we should return a value that means “no urgency at all”, since an endpoint with type `end` should *not* be used. We will use the special value ∞ to this purpose. In the latter case we need a type variable environment that keeps track of the obligation associated with each type variable, as determined by the recursive structure of the session type in which it is bound. More precisely, whenever $\mathsf{ftv}(T) \subseteq \mathsf{dom}(\Sigma)$ we define $\mathsf{ob}_\Sigma(T)$ as:

$$\mathsf{ob}_\Sigma(T) \overset{\text{def}}{=} \begin{cases} \infty & \text{if } T = \texttt{end} \\ \Sigma(\mathbf{t}) & \text{if } T = \mathbf{t} \in \mathsf{dom}(\Sigma) \\ \alpha & \text{if } T = \langle\alpha,\beta\rangle ?S.T' \text{ or } T = \langle\alpha,\beta\rangle !S.T' \\ \mathsf{ob}_\Sigma(S) & \text{if } T = \mu^{[\iota]}\mathbf{t}.S \end{cases}$$

Note that $\mathsf{ob}_\Sigma(\mu^{[\iota]}\mathbf{t}.S)$ is well defined because session types are contractive.

In the following we will make abundant use of sequences. For example, $\tilde{u}$ denotes a (possibly empty) sequence $u_1,\dots,u_n$ of names. With some abuse of notation we also use sequences for denoting environments. For example, we write $\tilde{\mathbf{t}} : \tilde{\alpha}$ for $\mathbf{t}_1 : \alpha_1,\dots,\mathbf{t}_n : \alpha_n$ and $\tilde{u} : \mu^{[\iota]}\tilde{\mathbf{t}}.\tilde{T}$ for $u_1 : \mu^{[\iota]}\mathbf{t}_1.T_1,\dots,u_n : \mu^{[\iota]}\mathbf{t}_n.T_n$.

The typing rules for processes are defined in Table 5. Rule [T-IDLE] states that the idle process is well typed only in the empty name environment. This is because endpoints are linear entities and the ownership of an endpoint with a type different from `end` imposes its use.

Rules [T-INPUT] and [T-OUTPUT] deal with prefixes. They check that the process is entitled to receive/send a message on the endpoint u and does so with the required priority. In [T-INPUT], the received

Table 5: Type rules for processes.

[T-IDLE]
$$\Sigma;\Gamma;\emptyset \vdash_\iota \mathbf{0}$$

[T-VAR]
$$\Sigma,\tilde{\mathbf{t}}:\tilde{\alpha};\Gamma,X:\langle\tilde{u}:\tilde{\mathbf{t}}\rangle;\tilde{u}:\tilde{\mathbf{t}} \vdash_\iota X$$

[T-INPUT]
$$\frac{\Sigma;\Gamma;\Delta,u:T,x:S \vdash_\iota P \qquad \forall v\in\mathsf{dom}(\Delta):\beta<\mathsf{ob}_\Sigma(\Delta(v))}{\Sigma;\Gamma;\Delta,u:\langle\alpha,\beta\rangle?S.T \vdash_\iota u?(x).P}$$

[T-PAR]
$$\frac{\Sigma;\Gamma;\Delta_1 \vdash_\iota P \qquad \Sigma;\Gamma;\Delta_2 \vdash_\iota Q}{\Sigma;\Gamma;\Delta_1,\Delta_2 \vdash_\iota P\,|\,Q}$$

[T-OUTPUT]
$$\frac{\Sigma;\Gamma;\Delta,u:T \vdash_\iota P \qquad \beta<\mathsf{ob}_\Sigma(S) \qquad \forall v\in\mathsf{dom}(\Delta):\beta<\mathsf{ob}_\Sigma(\Delta(v))}{\Sigma;\Gamma;\Delta,u:\langle\alpha,\beta\rangle!S.T,v:S \vdash_\iota u!\langle v\rangle.P}$$

[T-END]
$$\frac{\Sigma;\Gamma;\Delta \vdash_\iota P}{\Sigma;\Gamma;\Delta,u:\mathtt{end} \vdash_\iota P}$$

[T-REC]
$$\frac{\Sigma+\tilde{\mathbf{t}}:\mathsf{ob}_\Sigma(\tilde{T});\Gamma+X:\langle\tilde{u}:\tilde{\mathbf{t}}\rangle;\tilde{u}:\tilde{T} \vdash_\iota P \qquad \iota'\leq\iota}{\Sigma;\Gamma;\tilde{u}:\mu^{[\iota']}\tilde{\mathbf{t}}.\tilde{T} \vdash_\iota \mathtt{rec}^{[\iota']}\,X.P}$$

[T-SESSION]
$$\frac{\Sigma;\Gamma;\Delta,a^+:T,a^-:S \vdash_\iota P \qquad T\bowtie S}{\Sigma;\Gamma;\Delta \vdash_\iota (\nu a)P}$$

message x becomes part of the receiver's name environment, as the receiver has acquired its ownership. In [T-OUTPUT], the sent message v is removed from the sender's name environment because its ownership has been transferred. The premise $\beta < \mathsf{ob}_\Sigma(\Delta(v))$ for every $v \in \mathsf{dom}(\Delta)$ can be explained in this way: a process of the form $u?(x).P$ blocks until a message is received from endpoint u. So, while this process is complying with its duty to use u regardless of the obligation α associated with it, it is also postponing the use of any endpoint in $\mathsf{dom}(\Delta)$ until this synchronization takes place. The capability β gives information about the priority with which the peer endpoint of u will be used elsewhere in the system. Therefore, the process is respectful of the priorities of the endpoints in $\mathsf{dom}(\Delta)$ if they are lower (hence numerically greater) than β. Three considerations: first of all notice that, the reasoning excludes that the peer endpoint of u is in $\mathsf{dom}(\Delta)$. If it were, its obligation would be β and the premise would require the unsatisfiable constraint $\beta < \beta$. This allows us to rule out configurations such as that exemplified in (2). Second, if a process has in the name environment an endpoint whose type has highest priority (hence obligation 0), the process must use such endpoint immediately. If the process has two or more endpoints with highest priority, the only way for the process to be well typed is to fork into as many different parallel subprocesses, each immediately using one of the endpoints with highest priority. Third, in the case of [T-OUTPUT] it is also required that β be strictly smaller than the obligation associated with the type of the sent message. This is because such message cannot be used until it is received, namely until the send operation is completed.

Rule [T-PAR] splits the name environment and distributes its content among the composed processes.

Rule [T-SESSION] deals with session restrictions and augments the name environment in the restricted process with the two peer endpoints of the session, which must be related by duality.

Rules [T-REC] and [T-VAR] deal with recursions. The former verifies that the name environment consists of endpoints with a recursive type, therefore imposing a correspondence between recursive processes and recursive types. Then, it checks that the body of the recursion is well typed where the type variable environment has been augmented with the obligations associated with the recursive type variables, the process environment has been augmented with the association that specifies the valid name environment that is expected whenever the recursion process variable is met, and the name environment is updated by opening up the recursive types. The rule also checks that the ι' indices in the recursive types and in the recursive process do not exceed the bound ι. Finally, rule [T-END] discards names from the name environment, provided that these have type `end`.

Basic Properties. Below we collect a few basic properties of the type system leading to the subject reduction result. We begin with two standard substitution results, one for processes and the other one for endpoints.

Lemma 3.2 (weakening)**.** *If $\Sigma;\Gamma;\Delta \vdash_\iota P$ and $\Sigma \subseteq \Sigma'$ and $\Gamma \subseteq \Gamma'$, then $\Sigma';\Gamma';\Delta \vdash_\iota P$.*

Lemma 3.3 (process substitution)**.** *Let* **(1)** $\Sigma', \tilde{\mathbf{t}} : \tilde{\alpha}; \Gamma', X : \langle \tilde{u} : \tilde{\mathbf{t}} \rangle; \Delta, \tilde{u} : \tilde{T} \vdash_\iota P$ *and* **(2)** $\Sigma;\Gamma;\tilde{u} : \tilde{S} \vdash_\iota Q$ *where $\tilde{\alpha} = \mathsf{ob}_\Sigma(\tilde{S})$ and $\Sigma \subseteq \Sigma'$ and $\Gamma \subseteq \Gamma'$. Then $\Sigma';\Gamma';\Delta,\tilde{u} : \tilde{T}\{\tilde{S}/\tilde{\mathbf{t}}\} \vdash_\iota P\{Q/X\}$.*

Proof. By induction on the derivation of **(1)** and by cases on the last rule applied. We only show a few interesting cases:

$\boxed{\text{[T-VAR] when } P = X}$ Then $\Delta = \emptyset$, $\tilde{T} = \tilde{\mathbf{t}}$ and we conclude from **(2)** with an application of Lemma 3.2.

$\boxed{\text{[T-INPUT]}}$ We deduce:

- $\Delta, \tilde{u} : \tilde{T} = \Delta', u : \langle \alpha, \beta \rangle ?S.T$;
- $P = u?(x).P'$;
- $\Sigma', \tilde{\mathbf{t}} : \tilde{\alpha}; \Gamma', X : \langle \tilde{u} : \tilde{\mathbf{t}} \rangle; \Delta', u : T, x : S \vdash_\iota P'$;
- $\beta < \mathsf{ob}_{\Sigma', \tilde{\mathbf{t}}:\tilde{\alpha}}(\Delta'(v))$ for every $v \in \mathsf{dom}(\Delta')$.

We only consider the case in which $u \in \mathsf{dom}(\Delta)$, the case $u \in \tilde{u}$ being analogous. Then $\Delta' = \Delta'', \tilde{u} : \tilde{T}$ for some Δ''. By induction hypothesis we deduce $\Sigma';\Gamma';\Delta''', u : T, x : S \vdash_\iota P'\{Q/X\}$ where $\Delta''' = \Delta'', \tilde{u} : \tilde{T}\{\tilde{S}/\tilde{\mathbf{t}}\}$. Because of the hypothesis $\tilde{\alpha} = \mathsf{ob}_\Sigma(\tilde{S})$ we also have $\mathsf{ob}_{\Sigma', \tilde{\mathbf{t}}:\tilde{\alpha}}(\Delta'(v)) = \mathsf{ob}_{\Sigma'}(\Delta'''(v))$ for every $v \in \mathsf{dom}(\Delta') = \mathsf{dom}(\Delta''')$. We conclude $\Sigma';\Gamma';\Delta,\tilde{u} : \tilde{T}\{\tilde{S}/\tilde{\mathbf{t}}\} \vdash_\iota P\{Q/X\}$ with an application of [T-INPUT].

$\boxed{\text{[T-REC]}}$ We deduce:

- $\Delta, \tilde{u} : \tilde{T} = \tilde{v} : \mu^{[\iota']}\tilde{\mathbf{t}}'.\tilde{T}'$;
- $P = \texttt{rec}^{[\iota']}\, Y.P'$;
- $(\Sigma', \tilde{\mathbf{t}} : \tilde{\alpha}) + \tilde{\mathbf{t}}' : \mathsf{ob}_{\Sigma', \tilde{\mathbf{t}}:\tilde{\alpha}}(\tilde{T}'); (\Gamma', X : \langle \tilde{u} : \tilde{\mathbf{t}} \rangle) + Y : \langle \tilde{v} : \tilde{\mathbf{t}}' \rangle; \tilde{v} : \tilde{T}' \vdash_\iota P'$;
- $\iota' \leq \iota$.

We only consider the case in which $X \neq Y$ and $\tilde{\mathbf{t}} \cap \tilde{\mathbf{t}}' = \emptyset$ and $\tilde{u} = \tilde{v}$ (hence $\Delta = \emptyset$). Because of the hypothesis $\tilde{\alpha} = \mathsf{ob}_\Sigma(S)$ we know that $\mathsf{ob}_{\Sigma', \tilde{\mathbf{t}}:\tilde{\alpha}}(\tilde{T}') = \mathsf{ob}_{\Sigma'}(\tilde{T}'\{\tilde{S}/\mathbf{t}\})$. Therefore, by induction hypothesis we deduce $\Sigma' + \tilde{\mathbf{t}}' : \mathsf{ob}_{\Sigma'}(\tilde{T}'\{\tilde{S}/\tilde{\mathbf{t}}\}); \Gamma' + Y : \langle \tilde{u} : \tilde{\mathbf{t}}' \rangle; \tilde{u} : \tilde{T}'\{\tilde{S}/\tilde{\mathbf{t}}\} \vdash_\iota P'\{Q/X\}$. We conclude $\Sigma';\Gamma';\tilde{u} : (\mu^{[\iota]}\tilde{\mathbf{t}}'.\tilde{T}')\{\tilde{S}/\tilde{\mathbf{t}}\} \vdash_\iota P\{Q/X\}$ with an application of [T-REC]. □

Lemma 3.4 (value substitution)**.** *Let $\Sigma;\Gamma;\Delta, x : S \vdash_\iota P$ and $c^q \notin \mathsf{dom}(\Delta)$ and $P\{c^q/x\}$ is defined. Then $\Sigma;\Gamma;\Delta, c^q : S \vdash_\iota P\{c^q/x\}$.*

To prove subject reduction one must formulate it for processes which possibly have free endpoints. In doing so, it is necessary to impose, on the name environment used for typing such processes, that it enjoys a basic form of balancing, whereby the peer endpoints of the same session are associated with dual types. Formally:

Definition 3.5 (balanced context)**.** We say that Δ is *balanced* if $a^p, a^{\overline{p}} \in \mathsf{dom}(\Delta)$ implies $\Delta(a^p) \bowtie \Delta(a^{\overline{p}})$.

It is also necessary to determine an accurate correspondence between the name environment *before* the reduction, and the name environment *after* the reduction. For this reason, we define a reduction relation also for environments which takes into account the possible changes that can occur to the types in its range: either a recursive type is unfolded, or two corresponding actions from types associated with peer endpoints annihilate each other as the result of a communication.

Definition 3.6 (context reduction)**.** *Context reduction* is the least relation $\rightarrow$ defined by the rules

$$a^p : \mu^{[\iota+1]}\mathbf{t}.T \rightarrow a^p : T\{\mu^{[\iota]}\mathbf{t}.T/\mathbf{t}\} \qquad a^p : \langle\alpha,\beta\rangle!S.T, a^{\overline{p}} : \langle\beta,\alpha\rangle?S.T' \rightarrow a^p : T, a^{\overline{p}} : T'$$

and closed by context composition.

Context reductions preserve balancing.

Lemma 3.7. *Let Δ be balanced and $\Delta \rightarrow \Delta'$. Then Δ' is also balanced.*

Proof. By considering the two cases corresponding to the two possible reductions that can occur to Δ (Definition 3.6). We show one of them. Suppose $\Delta = \Delta'', a^p : \mu^{[\iota+1]}\mathbf{t}.T \rightarrow \Delta'', a^p : T\{\mu^{[\iota]}\mathbf{t}.T/\mathbf{t}\} = \Delta'$ and that $a^{\overline{p}} \in \mathsf{dom}(\Delta'')$. From the hypothesis that Δ is balanced we deduce $\Delta''(a^{\overline{p}}) \bowtie \mu^{[\iota+1]}\mathbf{t}.T$. We conclude $\Delta''(a^{\overline{p}}) \bowtie T\{\mu^{[\iota]}\mathbf{t}.T/\mathbf{t}\}$ by an application of [D-UNFOLD]. □

The property that typing is preserved by structural congruence is obvious.

Lemma 3.8. *Let $\Sigma;\Gamma;\Delta \vdash_\iota P$ and $P \equiv Q$. Then $\Sigma;\Gamma;\Delta \vdash_\iota Q$.*

Proof. Standard induction on $P \equiv Q$. □

Theorem 3.9 (subject reduction)**.** *Let $\Delta \vdash_\iota P$ and Δ balanced and $P \rightarrow Q$. Then $\Delta' \vdash_\iota Q$ for some Δ' such that $\Delta \rightarrow^* \Delta'$.*

Proof. By induction on the derivation of $P \rightarrow Q$ and by cases on the last rule applied. We only focus on the two base cases, the remaining ones follow by a simple induction argument and possibly Lemma 3.8.

$\boxed{\text{[R-COMM]}}$ Then $P = a^p!\langle c^q\rangle.P' \mid a^{\overline{p}}?(x).Q' \rightarrow P' \mid Q'\{c^q/x\} = Q$. From the hypothesis $\Delta \vdash_\iota P$ and [T-PAR] we deduce:

- $\Delta = \Delta_1, \Delta_2$;
- $\Delta_1 \vdash_\iota a^p!\langle c^q\rangle.P'$;
- $\Delta_2 \vdash_\iota a^{\overline{p}}?(x).Q'$.

From [T-OUTPUT] we deduce:

- $\Delta_1 = \Delta_1', a^p : \langle\alpha,\beta\rangle!S.T, c^q : S$;
- $\Delta_1', a^p : T \vdash_\iota P'$.

From [T-INPUT] we deduce:

- $\Delta_2 = \Delta_2', a^{\overline{p}} : \langle\alpha',\beta'\rangle?S'.T'$;
- $\Delta_2', a^{\overline{p}} : T', x : S' \vdash_\iota Q'$.

From the hypothesis that Δ is balanced we also deduce that $S' = S$ and $T \bowtie T'$. By definition of Δ we know that $c^q \notin \mathsf{dom}(\Delta_2')$. By Lemma 3.4 we obtain $\Delta_2', a^{\overline{p}} : T', c^q : S \vdash_\iota Q'\{c^q/x\}$. Let $\Delta' = \Delta_1', a^p : T, \Delta_2', a^{\overline{p}} : T', c^q : S$ and observe that $\Delta \rightarrow \Delta'$. We conclude $\Delta' \vdash_\iota Q$ with an application of [T-PAR].

$\boxed{\text{[R-REC]}}$ Then $P = \mathtt{rec}^{[\iota+1]}\ X.P' \rightarrow P'\{\mathtt{rec}^{[\iota]}\ X.P'/X\} = Q$. Because of [T-END] we can assume that Δ does not contain bindings for endpoints with type $\mathtt{end}$. Under this assumption, from the hypothesis $\Delta \vdash_\iota P$ and [T-REC] we deduce:

- $\Delta = \tilde{u} : \mu^{[\iota+1]}\tilde{\mathbf{t}}.\tilde{T}$;
- $\tilde{\mathbf{t}} : \mathsf{ob}(\tilde{T}); X : \langle \tilde{u} : \tilde{\mathbf{t}} \rangle ; \tilde{u} : \tilde{T} \vdash_\iota P'$.

From the hypothesis $\Delta \vdash_\iota P$ we also deduce $\tilde{u} : \mu^{[\iota]}\tilde{\mathbf{t}}.\tilde{T} \vdash_\iota \mathtt{rec}^{[\iota]}\, X.P'$. Note that $\mathsf{ob}(\tilde{T}) = \mathsf{ob}(\mu^{[\iota]}\tilde{\mathbf{t}}.\tilde{T})$. Let $\Delta' = \tilde{u} : \tilde{T}\{\mu^{[\iota]}\tilde{\mathbf{t}}.\tilde{T}/\tilde{\mathbf{t}}\}$ and observe that $\Delta \rightarrow^* \Delta'$. We conclude $\Delta' \vdash_\iota Q$ by Lemma 3.3. □

Roadmap to Soundness. We sketch the proof that the type system is sound, namely that every well-typed process P enjoys the progress property. According to Definition 2.1, this amounts to showing that for every P' such that

$$P \rightarrow^* P'$$

every top-level prefix involving some endpoint a^p in P' is eventually consumed by a matching prefix involving the peer endpoint $a^{\overline{p}}$. In this respect, what is difficult to prove is the existence of a reduction sequence starting from P' that eventually exposes the matching prefix, because the peer endpoint $a^{\overline{p}}$ may be guarded by a number of prefixes involving other endpoints. In fact, in P' the endpoint $a^{\overline{p}}$ may also be "in transit" as a message exchanged within other sessions, hence the soundness proof should in principle follow all the delegations of $a^{\overline{p}}$ until $a^{\overline{p}}$ becomes the subject of another top-level prefix.

Instead of attempting this, we follow a radically different strategy. First of all, we observe that a well-typed process in normal form cannot have top-level prefixes (Lemma 3.18). The idea then is to prolong the derivation from P' to some P'' such that

$$P \rightarrow^* P' \rightarrow^* P'' \not\rightarrow$$

and to conclude that the top-level prefix with subject a^p in P' must have been consumed by a matching prefix that has emerged along the reduction from P' to P''. Unfortunately, it is not always possible to find such a P'' because in general well-typed processes (like P') are not weakly normalizing. However, since P' is a residual of P after a *finite* number of reductions, it is possible to find a *finite approximation* $Q \in \mathbb{P}^{[\mathsf{fin}]}$ of P that reduces to a finite approximation $Q' \in \mathbb{P}^{[\mathsf{fin}]}$ of P'. Since any process in $\mathbb{P}^{[\mathsf{fin}]}$ can be shown to be strongly normalizing (Corollary 3.17), then there exists a normal form Q'' that approximates P''. The strategy is summarized by the following diagram

$$\begin{tikzcd}
P \arrow[r, phantom, "\rightarrow^*"] & P' \arrow[r, phantom, "\rightarrow^*"] & P'' & \\
\sqcup\!\!| & \sqcup\!\!| & \sqcup\!\!| & \\
Q \arrow[r, phantom, "\rightarrow^*"] & Q' \arrow[r, phantom, "\rightarrow^*"] & Q'' \arrow[r, phantom, "\not\rightarrow"] &
\end{tikzcd}$$

where $\sqsubseteq$ denotes some approximation relation. Note that P'' is not, in general, in normal form. However, we will define $\sqsubseteq$ in such a way that a user process and its approximation share the same structure, except that the approximation has finite indices marking the $\mathtt{rec}^{[\infty]}$ terms. Therefore, if Q'' has no top-level prefix, then so does P''.

Approximations. Intuitively we say that P approximates Q if P and Q share the same overall structure, except that every recursion in Q is capable of at least as many unfoldings as the corresponding recursion in P. We formalize this notion by means of an order between processes:

Definition 3.10 (approximation)**.** The $\sqsubseteq$ be the least pre-congruence over processes induced by the rule

$$\frac{\iota \leq \iota' \qquad P \sqsubseteq Q}{\mathtt{rec}^{[\iota]}\, X.P \sqsubseteq \mathtt{rec}^{[\iota']}\, X.Q}$$

We say that P *approximates* Q if $P \sqsubseteq Q$.

The following Proposition establishes a simulation result between a process and its approximations. In particular, the reductions of the approximated process include those of its approximations.

Proposition 3.11. *Let $P \to^* P'$ and $P \sqsubseteq Q$. Then there exists Q' such that $Q \to^* Q'$ and $P' \sqsubseteq Q'$.*

Proof. An easy induction on the derivation of $P \to^* P'$, using the fact that Q in general allows more unfoldings of its own recursions compared to P. □

Our strategy for proving soundness relies on the ability to compute one particular approximation of an arbitrary user process P.

Definition 3.12 (ι-approximant)**.** The *ι-approximant* of a user process P, written $P^{[\iota]}$, is obtained by turning every $\mathtt{rec}^{[\infty]}$ in P to a $\mathtt{rec}^{[\iota]}$. We similarly define the ι-approximant $T^{[\iota]}$ of a session type T.

An essential assumption of the strategy is that each ι-approximant of a well-typed user process is itself well typed. Unfortunately, this is not always the case and we must slightly restrict the class of well-typed user processes for which we are able to prove progress using this strategy. To illustrate the issue, consider the user process

$$P \stackrel{\text{def}}{=} (\nu a)(a^+!\langle 3\rangle.\mathtt{rec}^{[\infty]}\, X.a^+!\langle 3\rangle.X \mid \mathtt{rec}^{[\infty]}\, Y.a^-?(x).Y)$$

which is well typed using the name environment $a^+ : T, a^- : S$ where

$$T \stackrel{\text{def}}{=} \langle \alpha,\beta\rangle!int.\mu^{[\infty]}\mathbf{t}.\langle \alpha,\beta\rangle!int.\mathbf{t} \qquad \text{and} \qquad S \stackrel{\text{def}}{=} \mu^{[\infty]}\mathbf{t}.\langle \beta,\alpha\rangle?int.\mathbf{t}$$

In particular, observe that the type T associated with a^+ has been unfolded to account for the fact that in P the endpoint a^+ is used once outside of the recursion. Consequently the proof of $T \bowtie S$ crucially relies on [D-UNFOLD] (Definition 3.1) for dealing with this unfolding. Now, it is easy to see that no n-approximant of P is well typed. In particular, it is not the case that $T^{[n]} \bowtie S^{[n]}$ because [D-UNFOLD] attempts to relate $S^{[n]}$ with the *folding* of $T^{[n]}$, namely $\mu^{[n+1]}\mathbf{t}.\langle \alpha,\beta\rangle!int.\mathbf{t}$. In this particular case one could find a more clever approximation of P where the leftmost $\mathtt{rec}$ is assigned index n and the rightmost one index $n+1$. However, because several endpoints can be used within the same recursion, it is possible to find other examples where *no index assignment* makes the process typable with finite indices.

In general, typability of every approximant of P is guaranteed if P is typable without ever using [D-UNFOLD] for relating dual session types. This is the case if $\vdash_0 P^{[0]}$.

Proposition 3.13. *Let P be a user process such that $\vdash_0 P^{[0]}$. Then $\vdash_\iota P^{[\iota]}$ for every ι.*

Proof. The derivation for $\vdash_\iota P^{[\iota]}$ can be obtained from that for $\vdash_\iota P$ by replacing every index ∞ occurring in $\mathtt{rec}^{[\infty]}$'s and $\mu^{[\infty]}$'s with ι. □

Given any finite reduction of a user process, it is possible to find an appropriate finite approximant that simulates the reduction.

Proposition 3.14. *Let P be a user process and $P \to^* P'$. Then $P^{[n]} \to^* Q \sqsubseteq P'$ for some n and $Q \in \mathbb{P}^{[\mathsf{fin}]}$.*

Proof. Just let n be the number of reductions in the derivation of $P \to^* P'$. Then it is possible to simulate the reduction $P \to^* P'$ starting from $P^{[n]}$ to reach some $Q \sqsubseteq P'$. □

Strong Normalization of Finite Approximants. Let us address the strong normalization property of the $\mathbb{P}^{[\mathsf{fin}]}$ fragment of the calculus. While this result is intuitively obvious because each recursion can be unfolded only finitely many times, the formal proof requires a rather complex "measure" for processes that decreases at each reduction step. As a first attempt, one might define the measure of a process P as the vector where the item at index i is the number of $\mathtt{rec}^{[i]}$ terms occurring in P. This measure does not take into account the fact that recursions with the highest index may increase in number, if they occur nested within other recursions. For instance, we have:

$$\mathtt{rec}^{[3]}\, X.(\mathtt{rec}^{[6]}\, Y.Y \mid X) \to \mathtt{rec}^{[6]}\, Y.Y \mid \mathtt{rec}^{[2]}\, X.(\mathtt{rec}^{[6]}\, Y.Y \mid X)$$

The example shows that the potential multiplicity of a recursive term should also depend on the indices of the recursions within which it is nested. Above, since the $\mathtt{rec}^{[6]}$ term occurs with a $\mathtt{rec}^{[3]}$ one, 3 unguarded instances of the $\mathtt{rec}^{[6]}$ term can be generated overall. But this is true in the example above only because the outermost recursion binds exactly one occurrence of the X variable. In general, recursion variables can occur non-linearly. For instance, we have

$$\mathtt{rec}^{[3]}\, X.(P \mid X \mid X) \to P \mid \mathtt{rec}^{[2]}\, X.(P \mid X \mid X) \mid \mathtt{rec}^{[2]}\, X.(P \mid X \mid X)$$

where the eventual number of unguarded P terms is 5. In essence, in computing the multiplicity of a term we must consider not only the indices of the recursions within which it is nested, but also the multiplicity of the process variables bound by such recursions.

Formally, we define an auxiliary function $\mathbf{V}_{-}(-)$ such that $\mathbf{V}_X(P)$ provides the *measure* of X in P, namely the number of occurrences of X in P, taking into account duplications caused by inner recursions:

$$\begin{aligned}
\mathbf{V}_X(X) &= 1 \\
\mathbf{V}_X(Y) &= 0 && \text{if } X \neq Y \\
\mathbf{V}_X(u!\langle v\rangle.P) = \mathbf{V}_X(u?(x).P) &= \mathbf{V}_X(P) \\
\mathbf{V}_X(P \mid Q) &= \mathbf{V}_X(P) + \mathbf{V}_X(Q) \\
\mathbf{V}_X(\mathtt{rec}^{[n]}\, X.P) &= 0 \\
\mathbf{V}_X(\mathtt{rec}^{[n]}\, Y.P) &= \mathbf{V}_X(P) \cdot \textstyle\sum_{k=0}^{n-1} \mathbf{V}_Y(P)^k && \text{if } X \neq Y
\end{aligned}$$

All equations but the last one are unremarkable. In order to compute the measure of X in a process $\mathtt{rec}^{[n]}\, Y.P$, we multiply the measure of X in P by the amount of duplication that X is subjected to in all the unfoldings of $\mathtt{rec}^{[n]}\, Y.P$. This is determined by the geometric progression $\sum_{k=0}^{n-1} \mathbf{V}_Y(P)^k$ which, by convention, is 0 when $n = 0$. In particular, variables guarded by a $\mathtt{rec}^{[0]}$ term do not count, which is consistent with the fact that such terms do not reduce (see [R-REC] in Table 3).

Once we know how to determine the measure of variables, the measure of terms follows similarly:

$$\begin{aligned}
\mathbf{E}(X) &= 0 \\
\mathbf{E}(u!\langle v\rangle.P) = \mathbf{E}(u?(x).P) &= 1 + \mathbf{E}(P) \\
\mathbf{E}(P \mid Q) &= \mathbf{E}(P) + \mathbf{E}(Q) \\
\mathbf{E}(\mathtt{rec}^{[n]}\, X.P) &= (1 + \mathbf{E}(P)) \cdot \textstyle\sum_{k=0}^{n-1} \mathbf{V}_X(P)^k
\end{aligned}$$

In computing $\mathbf{E}(P)$ we also take into account the prefixes of P, which may cause reductions by means of [R-COMM]. The measure of a recursive term $\mathtt{rec}^{[n]}\, X.P$ is 1 (given by the unfolding of the term) plus the measure of P (after the unfolding) multiplied by the amount of duplication that X is subjected to in the body of the recursion. As before, summations are empty when $n = 0$. In particular $\mathbf{E}(\mathtt{rec}^{[0]}\, X.P) = 0$ for every X and P.

The following crucial lemma shows that our notion of measure is well behaved with respect to process substitutions:

Lemma 3.15. *Let $P, Q \in \mathbb{P}^{[\mathsf{fin}]}$ and $P\{Q/X\}$ be defined. Then $\mathbf{E}(P\{Q/X\}) = \mathbf{E}(P) + \mathbf{E}(Q) \cdot \mathbf{V}_X(P)$.*

Proof. By induction on the structure of P. We only prove a few interesting cases.

$\boxed{P = u!\langle v\rangle.P'}$ We have:

$$\begin{aligned}
\mathbf{E}(P\{Q/X\}) &= \mathbf{E}(u!\langle v\rangle.P'\{Q/X\}) && \text{definition of substitution}\\
&= 1 + \mathbf{E}(P'\{Q/X\}) && \text{definition of } \mathbf{E}(-)\\
&= 1 + \mathbf{E}(P') + \mathbf{E}(Q) \cdot \mathbf{V}_X(P') && \text{induction hypothesis}\\
&= \mathbf{E}(P) + \mathbf{E}(Q) \cdot \mathbf{V}_X(P) && \text{definition of } \mathbf{E}(-) \text{ and } \mathbf{V}_-(-)
\end{aligned}$$

$\boxed{P = P_1 \mid P_2}$ We have:

$$\begin{aligned}
\mathbf{E}(P\{Q/X\}) &= \mathbf{E}(P_1\{Q/X\}) + \mathbf{E}(P_2\{Q/X\}) && \text{definition of } \mathbf{E}(-)\\
&= \mathbf{E}(P_1) + \mathbf{E}(Q) \cdot \mathbf{V}_X(P_1) + \mathbf{E}(P_2) + \mathbf{E}(Q) \cdot \mathbf{V}_X(P_2) && \text{induction hypothesis}\\
&= \mathbf{E}(P) + \mathbf{E}(Q) \cdot \mathbf{V}_X(P) && \text{definition of } \mathbf{E}(-) \text{ and } \mathbf{V}_-(-)
\end{aligned}$$

$\boxed{P = \mathtt{rec}^{[n]}\ Y.P' \text{ when } X \neq Y}$ We have:

$$\begin{aligned}
\mathbf{E}(P\{Q/X\}) &= \mathbf{E}(\mathtt{rec}^{[n]}\ Y.P'\{Q/X\}) && \text{definition of substitution}\\
&= (1 + \mathbf{E}(P'\{Q/X\})) \cdot \textstyle\sum_{k=0}^{n-1} \mathbf{V}_Y(P'\{Q/X\})^k && \text{definition of } \mathbf{E}(-)\\
&= (1 + \mathbf{E}(P'\{Q/X\})) \cdot \textstyle\sum_{k=0}^{n-1} \mathbf{V}_Y(P')^k && \text{because } Y \notin \mathsf{fpv}(Q)\\
&= (1 + \mathbf{E}(P') + \mathbf{E}(Q) \cdot \mathbf{V}_X(P')) \cdot \textstyle\sum_{k=0}^{n-1} \mathbf{V}_Y(P')^k && \text{induction hypothesis}\\
&= (1 + \mathbf{E}(P')) \cdot \textstyle\sum_{k=0}^{n-1} \mathbf{V}_Y(P')^k \\
&\qquad + \mathbf{E}(Q) \cdot \mathbf{V}_X(P') \cdot \textstyle\sum_{k=0}^{n-1} \mathbf{V}_Y(P')^k && \text{distributivity}\\
&= \mathbf{E}(P) + \mathbf{E}(Q) \cdot \mathbf{V}_X(P) && \text{definition of } \mathbf{E}(-) \text{ and } \mathbf{V}_-(-)
\end{aligned}$$

□

The main result of this section states that the measure of a process decreases at each reduction step.

Theorem 3.16. *Let $P \in \mathbb{P}^{[\mathsf{fin}]}$ and $P \rightarrow Q$. Then $\mathbf{E}(Q) < \mathbf{E}(P)$.*

Proof. By induction on the derivation of $P \rightarrow Q$ and by cases on the last rule applied. Here we only show the two base cases, the others following by the inductive argument possibly using the fact that $\equiv$ and endpoint substitutions preserve the measure of processes and process variables.

$\boxed{\text{[R-COMM]}}$ Then $P = a^p!\langle c^q\rangle.P' \mid a^{\overline{p}}?(x).Q' \rightarrow P' \mid Q'\{c^q/x\} = Q$. We conclude:

$$\mathbf{E}(Q) = \mathbf{E}(P') + \mathbf{E}(Q'\{c^q/x\}) = \mathbf{E}(P) - 2$$

using the fact that endpoint substitutions do not alter the measure of a process.

$\boxed{\text{[R-REC]}}$ Then $P = \mathtt{rec}^{[n+1]}\ X.P' \rightarrow P'\{\mathtt{rec}^{[n]}\ X.P'/X\} = Q$. We derive:

$$\begin{aligned}
\mathbf{E}(Q) &= \mathbf{E}(P') + \mathbf{E}(\mathtt{rec}^{[n]}\ X.P') \cdot \mathbf{V}_X(P') && \text{Lemma 3.15}\\
&= \mathbf{E}(P') + (1 + \mathbf{E}(P')) \cdot \textstyle\sum_{k=0}^{n-1} \mathbf{V}_X(P')^k \cdot \mathbf{V}_X(P') && \text{definition of } \mathbf{E}(-)\\
&= \mathbf{E}(P') + (1 + \mathbf{E}(P')) \cdot (\textstyle\sum_{k=0}^{n} \mathbf{V}_X(P')^k - 1) && \text{geometric progression}\\
&= \mathbf{E}(P') + (1 + \mathbf{E}(P')) \cdot \textstyle\sum_{k=0}^{n} \mathbf{V}_X(P')^k - (1 + \mathbf{E}(P')) && \text{distributivity}\\
&= (1 + \mathbf{E}(P')) \cdot \textstyle\sum_{k=0}^{n} \mathbf{V}_X(P')^k - 1 \\
&= \mathbf{E}(P) - 1 && \text{definition of } \mathbf{E}(-)
\end{aligned}$$

□

Corollary 3.17. *Let $P \in \mathbb{P}^{[\mathsf{fin}]}$. Then P is strongly normalizing.*

Soundness Results. The last auxiliary result we need concerns the shape of well-typed processes in normal form, which are proved to have no pending prefixes at the top level.

Lemma 3.18. *Let $\vdash_\iota P$ and $P \not\rightarrow$. Then $P \equiv (\nu\tilde{a})\prod_{i\in I} \mathtt{rec}^{[0]}\, X_i.P_i$.*

Proof. Using the structural congruence rules of Table 2 it is clear that, whenever $P \not\rightarrow$, we have $P \equiv (\nu\tilde{a})P'$ for some P' such that

$$P' \;=\; \prod_{k\in K} \mathtt{rec}^{[0]}\, X_k.P_k \;\mid\; \prod_{i=1}^{m} a_i^{p_i}!\langle u_i\rangle.Q_i \;\mid\; \prod_{i=m+1}^{n} a_i^{p_i}?(x_i).R_i$$

We now prove that $n = 0$. From the hypothesis $\vdash_\iota P$ we deduce $\Delta \vdash_\iota P'$ for some Δ that is balanced. Let $\Delta(a_i^{p_i}) = T_i$ and $\Delta(a_i^{\overline{p}_i}) = \overline{T_i}$ for every $1 \le i \le n$. Let $\mathsf{cap}(T)$ be the capability of the topmost action in T, defined similarly to $\mathsf{ob}(T)$, and observe that $T \bowtie S$ implies $\mathsf{cap}(T) = \mathsf{ob}(S)$. We now proceed to show that for every $1 \le i \le n$ there exists $1 \le j \le n$ such that $\mathsf{cap}(T_j) < \mathsf{cap}(T_i)$. This is enough to conclude $n = 0$ because each $\mathsf{cap}(T_i)$ is finite.

Let $1 \le i \le n$. By [T-INPUT] and [T-OUTPUT] we deduce that T_i must begin with either an input or an output, so $a_i^{\overline{p}_i}$ cannot occur in any of the P_k because the type of endpoints occurring in P_k must begin with a $\mu^{[0]}$ by [T-REC]. Also, if $1 \le i \le m$, then $a_i^{\overline{p}_i}$ cannot be any of the $a_j^{p_j}$ for $m+1 \le j \le n$ and if $m+1 \le i \le n$, then $a_i^{\overline{p}_i}$ cannot be any of the $a_j^{p_j}$ for $1 \le j \le m$ because $P' \not\rightarrow$. Suppose that $a_i^{\overline{p}_i} \in \{u_j\} \cup \mathsf{fn}(Q_j)$ for some $1 \le j \le m$. By [T-OUTPUT] we deduce $\mathsf{cap}(T_j) < \mathsf{ob}(\overline{T_i}) = \mathsf{cap}(T_i)$. Suppose that $a_i^{\overline{p}_i} \in \mathsf{fn}(R_j)$ for some $m+1 \le j \le n$. By [T-INPUT] we deduce $\mathsf{cap}(T_j) < \mathsf{ob}(\overline{T_i}) = \mathsf{cap}(T_i)$. □

We conclude with the main result.

Theorem 3.19. *Every user process P such that $\vdash_0 P^{[0]}$ has progress.*

Proof. Consider a derivation of $P \rightarrow^* P'$ where $P' = (\nu\tilde{a})(a^p!\langle c^q\rangle.P_1 \mid P_2)$. By Proposition 3.14 there exist n and Q' such that $P^{[n]} \rightarrow^* Q'$ where $Q' \in \mathbb{P}^{[\mathsf{fin}]}$ and $Q' \sqsubseteq P'$. By Corollary 3.17 there exists Q'' such that $Q' \rightarrow^* Q'' \not\rightarrow$. From the hypothesis $\vdash_0 P^{[0]}$, Proposition 3.13, and Theorem 3.9 we deduce $\vdash_n Q''$. From Proposition 3.11 we deduce that there exists P'' such that $P' \rightarrow^* P''$ and $Q'' \sqsubseteq P''$. From Lemma 3.18 we deduce that Q'' does not contain unguarded prefixes, hence the same holds for P''. We conclude that $P_2 \rightarrow^* (\nu\tilde{b})(a^{\overline{p}}?(x).P_2' \mid Q) \rightarrow^* P''$ where a does not occur in $\tilde{b}$. □

Example 3.20 (forwarder)**.** Consider the process $P \stackrel{\text{def}}{=} \mathtt{rec}^{[\infty]}\, X.a^-?(x).b^+!\langle x\rangle.X$ which repeatedly receives a message from endpoint a^- and forwards it to endpoint b^+. Below is a derivation showing that $P^{[0]}$ is well typed in an appropriate name environment:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Sigma;\Gamma;a^- : \mathbf{t}, b^+ : \mathbf{t}' \vdash_0 X}\,[\text{T-VAR}] \qquad \delta < \mathsf{ob}_\Sigma(S) \qquad \delta < \alpha}{\Sigma;\Gamma;a^- : \mathbf{t}, b^+ : \langle\gamma,\delta\rangle!S.\mathbf{t}', x : S \vdash_0 b^+!\langle x\rangle.X}\,[\text{T-OUTPUT}] \qquad \beta < \gamma}{\Sigma;\Gamma;a^- : \langle\alpha,\beta\rangle?S.\mathbf{t}, b^+ : \langle\gamma,\delta\rangle!S.\mathbf{t}' \vdash_0 a^-?(x).b^+!\langle x\rangle.X}\,[\text{T-INPUT}]}{\emptyset;\emptyset;a^- : \mu^{[0]}\mathbf{t}.\langle\alpha,\beta\rangle?S.\mathbf{t}, b^+ : \mu^{[0]}\mathbf{t}'.\langle\gamma,\delta\rangle!S.\mathbf{t}' \vdash_0 \mathtt{rec}^{[0]}\, X.a^-?(x).b^+!\langle x\rangle.X}\,[\text{T-REC}]$$

In the derivation we let $\Sigma \stackrel{\text{def}}{=} \mathbf{t} : \alpha, \mathbf{t}' : \gamma$ and $\Gamma \stackrel{\text{def}}{=} X : \langle a^- : \mathbf{t}, b^+ : \mathbf{t}'\rangle$. Note that the constraints over priorities are satisfiable, taking for instance $\mathsf{ob}_\Sigma(S) = \alpha = \gamma = 1$ and $\delta = 0$. The interested reader can then extend the derivation to show that

$$(\nu a)(\nu b)(P^{[0]} \mid \mathtt{rec}^{[0]}\, Y.(\nu c)a^+!\langle c^+\rangle.Y \mid \mathtt{rec}^{[0]}\, Z.b^-?(y).Z)$$

is well typed (for instance, by taking $S = \texttt{end}$, the environment $c^+, c^- : \texttt{end}$ within the restriction (νc), and using [T-END] in two strategic places to discharge these endpoints), concluding that any process having this as 0-approximant has progress (Theorem 3.19). Incidentally, the same example also shows the importance of associating *two distinct priorities* to each action. If we were associating one single priority to each action, which essentially amounts to adding the constraints $\alpha = \beta$ and $\gamma = \delta$, this process would be ill typed because of the unsatisfiable chain of constraints $\alpha = \beta < \gamma = \delta < \alpha$. ■

4 Extensions

Both the calculus and the types can be easily extended to support labeled messages and label-driven branching. The type language can also be enriched with basic data types such as numbers, boolean values, etc. These values are not subject to any linearity constraint, so the $\mathsf{ob}(\cdot)$ function can be conservatively extended to basic types by returning ∞, meaning that [T-OUTPUT] does not require any constraint when sending messages of such types.

For simplicity our calculus is synchronous, but the type system applies with minimal changes also to asynchronous communication, which is more relevant in practice. In particular, since in an asynchronous communication model output operations are non-blocking, rule [T-OUTPUT] can avoid to enforce the sequentiality of the action with respect to the use of other endpoints.

Subtyping for session types has been widely studied in [10, 6, 12]. The decorations that are necessary for enforcing progress allow a natural form of subtyping, in accordance with the interpretation that a channel with type T can be safely used where a channel with type S is expected if $T \leqslant S$ (T is a subtype of S). Indeed, by looking at the typing rules, it is clear that obligations always occur on the right hand side of priority constraints, while capabilities always occur on the left hand side of these constraints. This means that subtyping can be covariant on capabilities and contravariant on obligations. More precisely, the core rules of subtyping would be formulated like this:

$$\text{[S-INPUT]}\ \frac{\alpha' \le \alpha \qquad \beta \le \beta' \qquad S \leqslant S' \qquad T \leqslant S'}{\langle \alpha, \beta \rangle ?S.T \leqslant \langle \alpha', \beta' \rangle ?S'.T'} \qquad \text{[S-OUTPUT]}\ \frac{\alpha' \le \alpha \qquad \beta \le \beta' \qquad S' \leqslant S \qquad T \leqslant S'}{\langle \alpha, \beta \rangle !S.T \leqslant \langle \alpha', \beta' \rangle !S'.T'}$$

Most session type theories support shared channel types that can be distributed non-linearly among processes. In [3] it was shown that shared channel types can be added with minimum effort by introducing a simple asymmetry between *service types*, which have the form $\langle \alpha, \infty \rangle ?S$ and only allow receiving messages of type S, and *client types*, which have the form $\langle \infty, \alpha \rangle !S$ and only allow sending messages of type S. Service endpoints must be used linearly like session endpoints, to make sure that no message sent over a client endpoint is lost. On the contrary, client endpoints can be safely shared between multiple processes or even left unused. As a result of this asymmetry, service endpoint types are given finite obligation and infinite capability (meaning that the owner of a service endpoint must use the channel, but is not guaranteed that it will receive any message from it), and dually client endpoint types are given infinite obligation and finite capability (meaning that the owner of a client endpoint may not use the endpoint, but if it does then it has the guarantee that the message will be eventually received). Because there is no guarantee that a message is sent over a client endpoint, the progress property (Definition 2.1) must be relaxed by allowing processes guarded by input actions on service endpoints.

5 Concluding Remarks

By adapting the type system for lock freedom described in [11] we have obtained a static analysis technique for ensuring progress in a calculus of sessions that is more fine grained than those described in [9, 2, 7]. For instance, the process shown in Example 3.20 is ill typed according to the type systems in [9, 2, 7] where it is not allowed to delegate a received channel. The increased precision of the approach presented here comes from associating pairs of priorities with each action in a session type, while in [2, 7] there is just one priority associated with the shared name on which the session is initiated. Following the ideas presented by Kobayashi [11] and adapted to sessions in the present work, Vieira and Vasconcelos [14] have defined a similar type system using abstract events instead of priorities, where events represent the temporal order with which actions should be performed. Their soundness result proves a weaker notion of progress, but it should be possible to strengthen it along the lines of Definition 2.1.

The aforementioned works can be classified as adopting a bottom-up approach, in the sense that they aim at verifying a global property (progress) of a compound system by checking properties of the system's constituents (the sessions). Other works adopt a top-down approach whereby well-typed or well-formed systems have progress by design. For example, Carbone and Montesi [5] advocate the use of a global programming model for describing systems of communicating processes such that, when the model is *projected* into the constituent processes, their parallel composition is guaranteed to enjoy progress. Caires and Pfenning [4] and subsequently Wadler [15] present type systems such that well-typed terms are deadlock-free. The result follows from the fact that the type system prevents the same process to interleave actions pertaining to different sessions.

A weakness of the type system presented here is that the priority constraints checked by rules [T-INPUT] and [T-OUTPUT] imply the knowledge of *every* endpoint used in the continuation of a process that follows a blocking action. This is feasible as long as processes are described as terms of an abstract calculus, but in a concrete programming language, processes are typically decomposed into functions, methods, objects, and modules. While type checking each of these entities in isolation, the type checker has only a partial knowledge about the possible continuations of the program, and of which endpoints are going to be used therein. We think that, in order for the approach to be applicable in practice, it is necessary to further enrich the structure of types. We are currently investigating this problem in a language with first-order functions and communication primitives.

Acknowledgments. I am grateful to Ilaria Castellani, Joshua Guttman and Philip Wadler, who encouraged me to reconsider (and eventually dismiss) the inaccurate interpretation of obligations and capabilities as timestamps that I used in an earlier version of this paper.

References

[1] Franco Barbanera & Ugo de'Liguoro (2010): *Two notions of sub-behaviour for session-based client/server systems*. In: *Proceedings of PPDP'10*, ACM, pp. 155–164, DOI: 10.1145/1836089.1836109.

[2] Lorenzo Bettini, Mario Coppo, Loris D'Antoni, Marco De Luca, Mariangiola Dezani-Ciancaglini & Nobuko Yoshida (2008): *Global Progress in Dynamically Interleaved Multiparty Sessions*. In: *Proceedings of CONCUR'08*, LNCS 5201, pp. 418–433, DOI: 10.1007/978-3-540-85361-9_33.

[3] Viviana Bono & Luca Padovani (2012): *Typing Copyless Message Passing*. *Logical Methods in Computer Science* 8, pp. 1–50, DOI: 10.2168/LMCS-8(1:17)2012.

[4] Luís Caires & Frank Pfenning (2010): *Session Types as Intuitionistic Linear Propositions*. In: *Proceedings of CONCUR'10*, LNCS 6269, pp. 222–236, DOI: 10.1007/978-3-642-15375-4_16.

[5] Marco Carbone & Fabrizio Montesi (2013): *Deadlock-freedom-by-design: multiparty asynchronous global programming*. In: *Proceedings of POPL'13*, ACM, pp. 263–274, DOI: `10.1145/2429069.2429101`.

[6] Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, Elena Giachino & Luca Padovani (2009): *Foundations of Session Types*. In: *Proceedings of PPDP'09*, ACM, pp. 219–230, DOI: `10.1145/1599410.1599437`.

[7] Mario Coppo, Mariangiola Dezani-Ciancaglini, Luca Padovani & Nobuko Yoshida (2013): *Inference of Global Progress Properties for Dynamically Interleaved Multiparty Sessions*. In: *Proceedings COORDINATION'13*, LNCS 7890, Springer, pp. 45–59, DOI: `10.1007/978-3-642-38493-6_4`.

[8] Ornela Dardha, Elena Giachino & Davide Sangiorgi (2012): *Session types revisited*. In: *Proceedings of PPDP'12*, ACM, pp. 139–150, DOI: `10.1145/2370776.2370794`.

[9] Mariangiola Dezani-Ciancaglini, Ugo de'Liguoro & Nobuko Yoshida (2008): *On Progress for Structured Communications*. In: *Proceedings of TGC'07*, LNCS 4912, pp. 257–275, DOI: `10.1007/978-3-540-78663-4_18`.

[10] Simon Gay & Malcolm Hole (2005): *Subtyping for session types in the π-calculus*. *Acta Informatica* 42(2-3), pp. 191–225, DOI: `10.1007/s00236-005-0177-z`.

[11] Naoki Kobayashi (2002): *A Type System for Lock-Free Processes*. *Information and Computation* 177(2), pp. 122–159, DOI: `10.1006/inco.2002.3171`.

[12] Luca Padovani (2011): *Session Types = Intersection Types + Union Types*. In: *Proceedings of ITRS'10*, EPTCS 45, pp. 71–89, DOI: `10.4204/EPTCS.45.6`.

[13] Luca Padovani (2012): *On Projecting Processes into Session Types*. *Mathematical Structures in Computer Science* 22, pp. 237–289, DOI: `10.1017/S0960129511000405`.

[14] Hugo Torres Vieira & Vasco Thudichum Vasconcelos (2013): *Typing Progress in Communication-Centred Systems*. In: *Proceedings of COORDINATION'13*, LNCS 7890, Springer, pp. 236–250, DOI: `10.1007/978-3-642-38493-6_17`.

[15] Philip Wadler (2012): *Propositions as sessions*. In: *Proceedings of ICFP'12*, ACM, pp. 273–286, DOI: `10.1145/2364527.2364568`.