Session Types at the Mirror

Luca Padovani
Istituto di Scienze e Tecnologie dell'Informazione, Università degli Studi di Urbino "Carlo Bo"
padovani@sti.uniurb.it

We (re)define session types as projections of process behaviors with respect to the communication channels they use. In this setting, we give session types a semantics based on fair testing. The outcome is a unified theory of behavioral types that share common aspects with conversation types and that encompass features of both dyadic and multi-party session types. The point of view we provide sheds light on the nature of session types and gives us a chance to reason about them in a framework where every notion, from well-typedness to the subtyping relation between session types, is semantically – rather than syntactically – grounded.

1 Introduction

The *leitmotif* in the flourishing literature on session types [14, 15, 16] is to associate every communication channel with a type that *constraints* how a process can use that channel. In this paper we take the opposite perspective: we *define* the session type associated with a channel as the *projection* of the behavior of the processes restricted to how that channel is used by them. As expected, this approach requires a language of session types that is more general than the ones we usually encounter in other works. But, and this is in summary the contribution of this work, the language we come up with is just a minor variation of well-known value passing process algebras that can be semantically characterized using well-known concepts and techniques.

To get acquainted with our approach, let us consider the following example written in π-calculus like language and which is a slightly simplified variant of the motivating example in [16]:

$$\begin{array}{rcl}
\text{Seller} & = & a?(x).x?(\mathit{title}:\texttt{String}).x!\mathit{price}.x?(\mathit{addr}:\texttt{Address}).x!\mathit{date} \\
\text{Buyer1} & = & (\nu c)a!c.c!\text{“The Origin of Species”}.c?(\mathit{price}:\texttt{Int}).(\nu d)b!d.d!\mathit{price}/2.d!c \\
\text{Buyer2} & = & b?(y).y?(\mathit{contrib}:\texttt{Int}).y?(z).z!\mathit{address}.z?(d:\texttt{Date})
\end{array}$$

Here we have two buyers that collaborate with each other in order to complete a transaction with a seller. Buyer1 creates a local channel c that it sends to Seller through the public channel a. The channel c is normally dubbed *session*: it is a fresh channel shared by Buyer1 and Seller on which the two can communicate privately. On c, Buyer1 sends to the Seller the name of a book, and Seller answers with its price. At this stage Buyer1 asks the collaboration of Buyer2: it creates another fresh channel d which it communicates to Buyer2 by means of the public channel b, it sends Buyer2 the amount of money Buyer2 should contribute, and finally it *delegates* the private channel c to Buyer2, so that Buyer2 can complete the transaction with the Seller. This implies sending the Seller a delivery address and receiving the estimated delivery date.

Let us focus on the public channels a and b: the former is used by Buyer1 for sending a channel of some type, say σ, and is used by Seller for receiving a channel of the same type. In our approach we say that the type of a is $?\sigma.\mathbf{1} \mid !\sigma.\mathbf{1}$, where $?\sigma.\mathbf{1}$ is the projected behavior of Seller on a, $!\sigma.\mathbf{1}$ is the projected behavior of Buyer1 on a, and $\mid$ denotes the composition of these two behaviors. In a similar way, b is used by Buyer1 and Buyer2 and has type $!\tau.\mathbf{1} \mid ?\tau.\mathbf{1}$, assuming that the channel exchanged between Buyer1 and Buyer2 has type τ. Channel c is more interesting: it is created by Buyer1, which uses it according to the type $\mathtt{!String.?Int}$. However, c is delegated to Seller right after its creation, and to Buyer2 when Buyer1 has finished using it. So, the true type of c is $\sigma \mid \mathtt{!String.?Int}.\rho$ where σ is the projection of Seller's behavior with respect to the channel c (after it has been received by Seller), and ρ is the projection of Buyer2's behavior with respect to the same channel after it has been received by Buyer2. By similar arguments, one can see that the type of d is $\tau \mid \mathtt{!Int.!}\rho.\mathbf{1}$ and the mentioned types σ, τ, and ρ are defined as $\mathtt{?String.!Int.?Address.!Date}.\mathbf{1}$, $\mathtt{?Int.?}\rho.\mathbf{1}$, and $\mathtt{!Address.?Date}.\mathbf{1}$, respectively. If we were to depict the projection we have operated for typing the channels in the example, we could summarize it as follows:

$$\begin{array}{rccccc}
 & \overbrace{\hphantom{\mathtt{?String.!Int.?Address.!Date}.\mathbf{1}}}^{\text{Seller}} & & \overbrace{\hphantom{\mathtt{!String.?Int}.\mathbf{1}}}^{\text{Buyer1}} & & \overbrace{\hphantom{\mathtt{!Address.?Date}.\mathbf{1}}}^{\text{Buyer2}} \\
a: & ?\sigma.\mathbf{1} & \mid & !\sigma.\mathbf{1} & & \\
b: & & & !\tau.\mathbf{1} & \mid & ?\tau.\mathbf{1} \\
c: & \mathtt{?String.!Int.?Address.!Date}.\mathbf{1} & \mid & \mathtt{!String.?Int}.\mathbf{1} & \mid & \mathtt{!Address.?Date}.\mathbf{1} \\
d: & & & \mathtt{!Int.!}\rho.\mathbf{1} & \mid & \mathtt{?Int.?}\rho.\mathbf{1}
\end{array}$$

Can we tell whether the system composed of Seller and the two buyers "behaves well"? Although at this stage we have not given a formal semantics to session types, by looking at the the types for the various channels involved in the example we can argue that they all eventually "reduce" to a parallel composition of $\mathbf{1}$'s. If we read the type $\mathbf{1}$ as the fact that a process stops using a channel with that type, this roughly indicates that all the conversations initiated in the example eventually terminate successfully.

Observe that the projection we have operated does not capture the temporal dependencies between communications occurring on different channels. This is a well-known source of problems if one is interested in global progress properties. In our approach, and unlike other presentations of session types, we do not even try to impose any linearity constraint on the channels being used, nor do we use *polarities* [11] or indexes [16, 1] for distinguishing different roles. For example, the process Buyer1 keeps using channel c after it has been delegated, and it delegates the channel once more before terminating. As a consequence, the projection we operate may not even capture the temporal dependencies between communications occurring *on the same channel*. This can happen if two distinct free variables are instantiated with the same channel during some execution. Thus, we must impose additional constraints on processes only to ensure the type preservation property. Interestingly, such additional constraints are exactly the same used for ensuring global progress [8, 1, 3].

We can identify three main contributions of this work: (1) we show that session types can be naturally generalized to an algebraic language of processes that closely resembles value-passing CCS; (2) as a consequence, we are able to work on session types reusing a vast toolkit of known results and techniques; in particular, we are able to semantically justify the fundamental concepts (duality, well-typedness, the subtyping relation) that are axiomatically or syntactically presented in other theories; (3) we provide a unified framework of behavioral types that encompasses features not only of dyadic and multi-party session types, but also of conversation types [2].

Structure of the paper. In Section 2 we define session types as a proper process algebra equipped with a labeled transition system and a testing semantics based on fair testing. This will immediately provide us with a semantically justified equivalence relation – actually, a precongruence – to reason about safe replacement of channels and well-behaving systems. In Section 3 we formally define a process language that is a minor variant of the π-calculus without any explicit construct that is dedicated to session-oriented interaction. We will show how to type processes in this language and illustrate the main features of the type system with several examples. Finally, we will prove the main properties (type preservation and local progress) of our typing relation. Section 4 concludes. For referee convenience, the Appendix contains proofs and auxiliary results.

Related work. Theories of dyadic session types can be traced back to the works of Honda [14] and Honda *et al.* [15]. Since then, the application of session types has been extended to functional languages [20, 12] and object-oriented languages (see [10, 9] for just a few examples). A major line of research is the one dealing with so-called multi-party session types, those describing sessions where multiple participants interact simultaneously [16, 1]. An in depth study of a subtyping relation for session types can be found in [11], while [19] provides an incremental tutorial presentation of the most relevant features of dyadic session types.

Conversation types [2] are a recently introduced formalism that aims at generalizing session types for the description of the behavior of processes that interact within and across the scope of structurally organized communications called conversations. Conversation types are very similar to the language of session types we propose here, for example they embed a parallel composition operator for representing the composed behavior of several processes simultaneously accessing a conversation. The difference with our approach mainly resides in the semantics of types: we treat session types as terms of a proper process algebra with a proper transition relation and all the relevant notions on types originate from here. In [2], the semantics of conversation types is given in terms of syntactically-defined notions of subtyping and *merging*. Also, [2] uses a process language that incorporates explicit constructs for dealing with conversations, while we emphasize the idea of *projected behavior* by working with the naked π-calculus.

Elsewhere [3] we have been advocating the use of a testing approach in order to semantically justify session types. Unlike [3], here we disallow branch selection depending on the type of channels. This reduces the expressiveness of types for the sake of a simplification of the technicalities in the resulting theory. Another difference is that in the present paper we adopt a *fair testing* approach [18].

Finally, it should be mentioned that the use of processes as types has already been proposed in the past, for example in [5, 17]. In particular, [17] uses a language close to value-passing CCS for defining an effect system for Concurrent ML.

2 Syntax and semantics of session types

Let us fix some conventions: σ, τ, ρ, ... range over *session types*; α, ... range over *actions*; t, s, ... range over *types*; v, ... range over an unspecified set $\mathscr{V}$ of *basic values*; $\mathtt{B}$, ... range over an unspecified set of *basic types* such as `Int`, `Bool`, `String`, and so on. The syntax of session types is defined by the grammar in Table 1. Types represent sets of related values: $\mathbb{0}$ is the empty type, the one inhabited by no value; basic types are arbitrary subsets of $\mathscr{V}$; for every $\mathsf{v} \in \mathscr{V}$ we write v for the *singleton type* whose only value is v itself. We will write $\mathsf{v} : t$ to state that v inhabits type t and we will sometimes say that v is of type t.

Actions represent input/output operations on a channel. An action $!t$ represents the sending of an arbitrary value of type t; an action $?t$ represents the receiving of an arbitrary value of type t; actions $!\sigma$ and $?\sigma$ are similar but they respectively represent the sending and receiving of a channel of type σ.

Although session types are used to classify channels, they describe the behavior of processes using those channels. Consistently with this observation, we will often present session types as characterizing processes rather than channels. In the explanation that follows, it is useful to keep in mind that, when a process uses a channel according to some protocol described by a session type, it expects to interact with other processes that use the same channel according to other protocols. For a communication to occur,

Table 1: Syntax of session types.

σ	$::=$		**session type**	α	$::=$		**action**	t	$::=$		**type**
		$\mathbf{0}$	(failure)			$?t$	(value input)			$\mathbb{0}$	(empty)
	$\mid$	$\mathbf{1}$	(success)		$\mid$	$!t$	(value output)		$\mid$	$\mathtt{v}$	(singleton)
	$\mid$	$\alpha.\sigma$	(action prefix)		$\mid$	$?\sigma$	(channel input)		$\mid$	$\mathtt{B}$	(basic type)
	$\mid$	$\sigma+\sigma$	(external choice)		$\mid$	$!\sigma$	(delegation)				
	$\mid$	$\sigma\oplus\sigma$	(internal choice)								
	$\mid$	$\sigma \mid \sigma$	(composition)								

the process must perform an action on the channel (say, sending a value of some type), and another process must perform the corresponding co-action (say, receiving a value of the same type). The session type $\mathbf{0}$ classifies a channel on which a communication error has occurred. No correct system should ever involve channels typed by $\mathbf{0}$, but we will see that it is useful to have an explicit term denoting a static error. The session type $\mathbf{1}$ describes a process that performs no further action on a channel. The session type $\alpha.\sigma$ describes a process that performs the action α, and then behaves according to the protocol σ. The session type $\sigma+\tau$ is the *external choice* of σ and τ and describes a process that offers interacting processes to behave according to one of the branches. Dually, the session type $\sigma\oplus\tau$ is the *internal choice* of σ and τ and describes a process that internally decides to behave according to one of the branches. The session type $\sigma \mid \tau$ describes the simultaneous access to a shared channel by two processes behaving according to σ and τ. If we have n processes sharing a common channel and each process behaves according to some protocol σ_i, then $\sigma_1 \mid \cdots \mid \sigma_n$ describes the overall protocol implemented by the processes on the channel.

We do not rely on any explicit syntax for describing recursive behaviors. We borrow the technique already used in [3] and define the set of session types as the set of possibly infinite syntax trees generated by the productions of the grammar in Table 1 that satisfy the following conditions:

1. the tree must contain a finite number of *different* subtrees;
2. on every infinite branch of the tree there must be infinite occurrences of the action prefix operator;
3. the tree must contain a finite number of occurrences of the parallel composition operator.

The first condition is a standard *regularity condition* imposing that the tree must be a *regular tree* [6]. The second one is a *contractivity condition* ruling out meaningless regular trees such as those generated by the equations $X = X+X$ or $X = X\oplus X$. Finally, it can be shown that the last condition enforces that the protocol described by a session type is “finite state”.

To familiarize with session types consider the following two examples: $?\mathtt{Int}.!\mathtt{String}.\mathbf{1}+?\mathtt{Bool}.!\mathtt{Real}.\mathbf{1}$ describes a process that waits for either an integer number or a Boolean value. If the process receives an integer number, it sends a string; if the process receives a Boolean value, it sends a real number. After that, in either case, the process stops using the channel. Instead, the session type $!\mathtt{Int}.\mathbf{1}\oplus!\mathtt{Bool}.\mathbf{1}$ describes a process that internally decides whether to send an integer or a Boolean value.

It may seem that the syntax of session types is overly generic, and that external choices make sense only when they are guarded by input actions and internal choices make sense only when they are guarded by output actions. As a matter of facts, this is a common restriction in standard session type presentations. In our approach, this generality is actually *necessary*: a session type $\sigma = !\mathtt{Int}.\mathbf{1} \mid !\mathtt{Bool}.\mathbf{1}$ describes two processes trying to simultaneously send an integer and a Boolean value on the same channel. A process interacting with these two parties is allowed to read both values in either order, since both are available.

Table 2: Transitions of session types.

$$\mathbf{1} \xrightarrow{\checkmark} \mathbf{1} \qquad \sigma \oplus \tau \longrightarrow \sigma \qquad !\mathsf{v}.\sigma \xrightarrow{!\mathsf{v}} \sigma \qquad !\rho.\sigma \xrightarrow{!\rho} \sigma \qquad ?\rho.\sigma \xrightarrow{?\rho} \sigma$$

$$\frac{\mathsf{v} : t}{!t.\sigma \longrightarrow !\mathsf{v}.t} \qquad \frac{\mathsf{v} : t}{?t.\sigma \xrightarrow{?\mathsf{v}} \sigma} \qquad \frac{\sigma \longrightarrow \sigma'}{\sigma + \tau \longrightarrow \sigma' + \tau} \qquad \frac{\sigma \xrightarrow{\mu} \sigma'}{\sigma + \tau \xrightarrow{\mu} \sigma'} \qquad \frac{\sigma \longrightarrow \sigma'}{\sigma \mid \tau \longrightarrow \sigma' \mid \tau}$$

$$\frac{\sigma \xrightarrow{\mu} \sigma' \qquad \mu \neq \checkmark}{\sigma \mid \tau \xrightarrow{\mu} \sigma' \mid \tau} \qquad \frac{\sigma \xrightarrow{\checkmark} \sigma' \qquad \tau \xrightarrow{\checkmark} \tau'}{\sigma \mid \tau \xrightarrow{\checkmark} \sigma' \mid \tau'} \qquad \frac{\sigma \xrightarrow{!\mathsf{v}} \sigma' \qquad \tau \xrightarrow{?\mathsf{v}} \tau'}{\sigma \mid \tau \longrightarrow \sigma' \mid \tau'}$$

$$\frac{\sigma \xrightarrow{!\rho} \sigma' \qquad \tau \xrightarrow{?\rho'} \tau' \qquad \rho \preceq \rho'}{\sigma \mid \tau \longrightarrow \sigma' \mid \tau'} \qquad \frac{\sigma \xrightarrow{!\rho} \sigma' \qquad \tau \xrightarrow{?\rho'} \tau' \qquad \rho \not\preceq \rho'}{\sigma \mid \tau \longrightarrow \mathbf{0}}$$

In other words, the session type σ is equivalent to $\texttt{!Int.!Bool}.\mathbf{1} + \texttt{!Bool.!Int}.\mathbf{1}$, that is the interleaving of the actions in σ. Had we expanded σ to $\texttt{!Int.!Bool}.\mathbf{1} \oplus \texttt{!Bool.!Int}.\mathbf{1}$ instead, no interacting process would be able to decide which value, the integer or the Boolean value, to read first. The ability to express parallel composition in terms of choices is well studied in process algebra communities where it goes under the name of *expansion law* [7, 13]. This ability is fundamental in order to define complete proof systems and algorithms for deciding equivalences. Decidability issues aside, we envision two more reasons why this generality is appealing: first, it allows us to express the typing rules (Section 3) in a more compositional way, which is particularly important in our approach where we aim at capturing full, unconstrained process behaviors; second, it clearly separates communications (represented by actions) from choices, thus yielding a clean, algebraic type language with orthogonal features.

We equip session types with an operational semantics that mimics the actions performed by processes behaving according to these types. The labeled transition system of session types is defined by the rules in Table 2 plus the obvious symmetric rules of those concerning choices and parallel composition. Transitions make use of *labels* ranged over by $\mu, \ldots$ and generated by the grammar:

$$\mu \ ::= \ \checkmark \ \mid \ ?\mathsf{v} \ \mid \ !\mathsf{v} \ \mid \ ?\sigma \ \mid \ !\sigma$$

Strictly speaking, the transition system is defined by two relations: a labeled one $\xrightarrow{\mu}$ describing *external, visible actions* and an unlabeled one $\longrightarrow$ describing *internal, invisible actions*. The session type $\mathbf{1}$ emits a single action $\checkmark$ denoting successful termination of the protocol, and reduces to itself. The session type $\sigma \oplus \tau$ can perform an invisible transition to either σ or τ. The session type $!\mathsf{v}.\sigma$ emits the value v and reduces to σ. Similarly, $!\rho.\sigma$ emits a signal $!\rho$ (the output of a channel of type ρ) and $?\rho.\sigma$ emits a signal $?\rho$ (the input of a channel of type ρ). The first rule on the second line of Table 2 states that a process behaving according to $!t.\sigma$ internally chooses a value v of type t to send, and once has committed to such a value it reduces to $!\mathsf{v}.\sigma$. The next rule expresses a similar capability for input actions. However, observe that a process behaving according to $!t.\sigma$ commits to sending *one particular* value of type t, whereas a process behaving according to $?t.\sigma$ is able to receive *any* value of type t. The following rule states that $+$ is indeed an external choice, thus internal choices in either branch do not preempt the other branch. This is a typical reduction rule for those languages with two different choices, such as CCS without τ's [7]. The last two rules in the second line state obvious reductions for external choices, which offer any action that is offered in either branch, and parallel compositions, which allow either component to internally evolve independently. The first rule in the third line states that any action other than $\checkmark$ is offered by a parallel composition whenever it is offered by one of the components; the second rule in the row states that a parallel composition has successfully terminated only if both components have; the last rule states the obvious synchronization between components offering dual actions. The last line contains two more synchronization rules concerning channels: the first rule states that a process sending a channel of type ρ can synchronize with another process willing to receive a channel of type ρ', but only if $\rho \preceq \rho'$. Here $\preceq$ is a *subtyping relation* meaning that any channel of type ρ can be used where a channel of type ρ' is expected. We shall formally define $\preceq$ in a moment; for the time being we must content ourselves with this intuition. The last rule states that if the relation $\rho \preceq \rho'$ is not satisfied, the synchronization occurs nonetheless, but a communication error occurs.

Before we move on to the subtyping relation for session types, we should point out a fundamental design decision that relates communication and external choices. On the one hand, values other than channels may drive the selection of the branch in external choices. For example, we have $?\texttt{Int}.\sigma + ?\texttt{Bool}.\tau \xrightarrow{?3} \sigma$ while $?\texttt{Int}.\sigma + ?\texttt{Bool}.\tau \xrightarrow{?\texttt{true}} \tau$. The *type* of the value determines the branch, and this feature allows us to model the label-driven branch selection that is found in standard presentations. On the other hand, the last two rules in Table 2 show that branch selection cannot be affected by the type of the channel being communicated. It is true that $?\rho.\sigma + ?\rho'.\tau \xrightarrow{?\rho} \sigma$ and $?\rho.\sigma + ?\rho'.\tau \xrightarrow{?\rho'} \tau$, but when we compose $?\rho.\sigma + ?\rho'.\tau$ with $!\rho''.\theta$ any reduction is possible, and the residual may or may not be $\mathbf{0}$ depending on the relation between ρ, ρ', and ρ'':

$$\frac{\rho'' \preceq \rho}{?\rho.\sigma + ?\rho'.\tau \mid !\rho''.\theta \longrightarrow \sigma \mid \theta} \qquad\qquad \frac{\rho'' \not\preceq \rho}{?\rho.\sigma + ?\rho'.\tau \mid !\rho''.\theta \longrightarrow \mathbf{0}}$$

To be sure that the residual is not $\mathbf{0}$, it must be the case that $\rho'' \preceq \rho$ *and* $\rho'' \preceq \rho'$. In summary, we do not allow dynamic dispatching according to the type of a channel, namely *all channels are equal and indistinguishable*. This is not the only possible choice (see [3] for an alternative), but is one that significantly simplifies the theory.

In the following we adopt standard conventions regarding the transition relations: we write $\Longrightarrow$ for the reflexive, transitive closure of $\longrightarrow$; we write $\sigma \xrightarrow{\mu}$ (respectively, $\sigma \xRightarrow{\mu}$) if there exists τ such that $\sigma \xrightarrow{\mu} \tau$ (respectively, $\sigma \xRightarrow{\mu} \tau$); we write $\not\longrightarrow$, $\xrightarrow{\mu}\!\!\!\!\!/\;\;$, $\xRightarrow{\mu}\!\!\!\!\!/\;\;$ for the usual negated relations; for example, $\sigma \not\longrightarrow$ means that σ does not perform internal transitions.

The first characterization we give is that of *complete* session type, namely a session type that can always reach a successful state, no matter of its internal transitions.

Definition 2.1 (completeness)**.** We say that σ is *complete* if $\sigma \Longrightarrow \sigma'$ implies $\sigma' \xRightarrow{\checkmark}$.

Completeness roughly corresponds to the notion of *well-typed process* in standard presentations: it means that no evolution of the system may lead to an error state or a to state where one process insists on sending a message that no interacting party is willing to accept. $\mathbf{1}$ is the simplest complete session type; the session types $?\sigma.\mathbf{1} \mid !\sigma.\mathbf{1}$ and $\sigma \mid !\texttt{String}.?\texttt{Int}.\rho$ we have seen in the introduction are also complete, since every maximal transition leads to a successfully terminated state. The simplest example of incomplete session type is $\mathbf{0}$, another example being $?\texttt{Int}.\mathbf{1} \mid !\texttt{Real}.\mathbf{1}$ because of the maximal reduction $?\texttt{Int}.\mathbf{1} \mid !\texttt{Real}.\mathbf{1} \longrightarrow ?\texttt{Int}.\mathbf{1} \mid !\sqrt{2}.\mathbf{1} \not\longrightarrow$. If we take σ as the solution of the equation $X = ?\texttt{Int}.X$ and τ as the solution of the equation $Y = !\texttt{Int}.Y$ we have that $\sigma \mid \tau$ is *not* complete, despite it never reaches a deadlock state. In this sense the notion of completeness embeds a *fairness principle* that is typically found in fair testing theories [18]. In fact, we provide session types with a (fair) testing semantics: σ is "smaller than" τ if every session type that completes σ completes τ as well.

Definition 2.2 (subsession)**.** We say that σ is a *subsession* of τ, notation $\sigma \preceq \tau$, if $\sigma \mid \rho$ complete implies $\tau \mid \rho$ complete for every ρ. We write $\approx$ for the equivalence relation induced by $\preceq$, namely $\approx \, = \, \preceq \cap \succeq$.

The equational theory generated by this definition is not immediately obvious, and we will not develop it here because it falls out of the scope of the paper. Nonetheless, a few relations are easy to check: for example, $+$, $\oplus$, and $\mid$ are commutative, associative operators; $\mathbf{0}$ is neutral for $+$ and $\mathbf{1}$ is neutral for $\mid$; furthermore $\sigma \oplus \tau \preceq \sigma$. Namely, it is safe to use a channel with type $\sigma \oplus \tau$ where another one of type σ is expected. If the safety property mentioned here seems hard to grasp, one should resort to the intuition that the "type" of a channel actually is the behavior of a process communicating on that channel. A process that expects to receive a channel with type σ will behave on that channel according to σ; if we send that process a channel with type $\sigma \oplus \tau$, the receiving process will still behave according to σ, which is a more deterministic behavior than $\sigma \oplus \tau$, hence no problem may arise. As a special case of reduction of nondeterminism, we have $!\mathtt{Real}.\sigma \preceq \, !\mathtt{Int}.\sigma$ assuming that $\mathtt{Int}$ is a subtype of $\mathtt{Real}$. Other useful relations are those concerning failed processes: we have $\mathbf{0} \approx \alpha.\mathbf{0}$ and $!\mathbb{0}.\sigma \approx \, ?\mathbb{0}.\sigma \approx \mathbf{0}$. More generally, the relation $\sigma \approx \mathbf{0}$ means that there is no session type τ such that $\sigma \mid \tau$ is complete: σ is intrinsically flawed and cannot be remedied. The class of non-flawed session types will be of primary importance in the following, to the point that we reserve them a name.

Definition 2.3 (viability)**.** We say that σ is *viable* if $\sigma \mid \rho$ is complete for some ρ.

Remark 2.1. At this stage we can appreciate the fact that subsession depends on the transition relation, and that the transition relation depends on subsession. This circularity can be broken by stratifying the definitions: a session type σ is given weight 0 if it contains no prefix of the form $?\rho$ or $!\rho$; a session type σ is given weight $n > 0$ if any session type ρ in any prefix of the form $?\rho$ or $!\rho$ occurring in σ has weight at most $n-1$. By means of this stratification, one can see that the definitions of the transition relation and of subsession are well founded. ■

It is fairly easy to see that $\preceq$ is a precongruence with respect to action prefix, internal choice, and parallel composition. The case of the action prefix is trivial. As regards the internal choice, it suffices to observe that $(\sigma \oplus \tau) \mid \rho$ is complete if and only if both $\sigma \mid \rho$ and $\tau \mid \rho$ are complete. Namely, $\oplus$ corresponds to a set-theoretic intersection between session types that complete σ and τ. As regards the parallel composition, the precongruence follows from the very definition of subsession, since $\sigma \mid \sigma' \preceq \tau \mid \sigma'$ if $(\sigma \mid \sigma') \mid \rho$ complete implies $(\tau \mid \sigma') \mid \rho$ complete, namely if $\sigma \mid (\sigma' \mid \rho)$ complete implies $\tau \mid (\sigma' \mid \rho)$ complete, that is if $\sigma \preceq \tau$. Because all the non-viable session types are $\approx$-equal, however, $\preceq$ is *not* a precongruence with respect to the external choice. For example, we have $\mathbf{0} \preceq \, !\mathtt{Int}.\mathbf{0}$ but $!\mathtt{Int}.\mathbf{1} + \mathbf{0} \not\preceq \, !\mathtt{Int}.\mathbf{1} + !\mathtt{Int}.\mathbf{0} \approx \mathbf{0}$. This is a major drawback of the subsession relation as it is defined, since it prevents $\preceq$ from being used in arbitrary contexts for replacing equals with equals (note that $\approx$ is *not* a congruence for the same reasons). We resort to a standard technique for defining the largest relation included in $\preceq$ that is a precongruence with respect to the external choice. We call this relation *strong subsession*:

Definition 2.4 (strong subsession)**.** Let $\sqsubseteq$ be the largest relation included in $\preceq$ that is a precongruence with respect to $+$, namely $\sigma \sqsubseteq \tau$ if and only if $\sigma + \rho \preceq \tau + \rho$ for every ρ. We write $\simeq$ for the equivalence relation induced by $\sqsubseteq$, namely $\simeq \, = \, \sqsubseteq \cap \sqsupseteq$.

We end this section with a few results about $\preceq$ and $\sqsubseteq$. First of all, we can use $\sqsubseteq$ for reasoning about viability and completeness of a session type:

Proposition 2.1. *The following properties hold:*

Table 3: Syntax of processes.

P	$::=$		**process**	π	$::=$		**action**
	$\mid$	0	(idle)		$\mid$	$u?(x:t)$	(value input)
	$\mid$	$\pi.P$	(action prefix)		$\mid$	$u!e$	(value output)
	$\mid$	$\star P$	(replication)		$\mid$	$u?(x)$	(channel input)
	$\mid$	$P+P$	(external choice)		$\mid$	$u!v$	(delegation)
	$\mid$	$P\oplus P$	(internal choice)				
	$\mid$	$P\mid P$	(parallel composition)				
	$\mid$	$(\nu c)P$	(restriction)				

1. *σ is not viable if and only if $\sigma \sqsubseteq \mathbf{0}$;*
2. *σ is complete if and only if $\mathbf{1}+\sigma \sqsubseteq \sigma$.*

Then, we prove that $\preceq$ and $\sqsubseteq$ are *almost* the same relation, in the sense that they coincide as soon as the smaller session type is viable. This means that for all practical purposes the use of $\sqsubseteq$ in place of $\preceq$ is immaterial, if not for the gained precongruence, since in no case we will be keen on replacing a channel with a viable type with one that is not viable.

Theorem 2.1. *$\sigma \preceq \tau$ if and only if either $\sigma \sqsubseteq \mathbf{0}$ or $\sigma \sqsubseteq \tau$.*

3 Processes

Processes are defined by the grammar in Table 3. We use P, Q, R, ... to range over processes; we use π, ... to range over action prefixes; we use a, b, c, ... to range over *channel names*; we let x, y, z, ... range over *variables* and u, v, ... range over channel names and variables (v should not be confused with v that we used to range over elements of $\mathscr{V}$); we let e, ... range over an unspecified language of *expressions*. The process language is a minor variation of the π-calculus, so we remark here only the differences: we have four action prefixes: $u?(x:t)$ denotes a receive action for a basic value x of type t on channel u; $u!e$ denotes a send action for the value of the expression e on channel u; $u?(x)$ denotes a receive action for a channel x on channel u; $u!v$ denotes a send action for a channel v on channel u. Consistently with the language of session types, actions denoting send/receive operations of channels are “untyped”. The process $\star P$ denotes unbounded replications of process P, and $P+Q$ and $P\oplus Q$ respectively denote the external and internal choice between P and Q. We will usually omit the 0 process; we will write $\mathtt{fn}(P)$ for the set of free channel names occurring in P (the only binder for channel names is restriction); we will write $P\{^m/_x\}$ for the process P where free occurrences of the variable x has been replaced by m.

The transition relation of processes is defined by an almost standard relation in Table 4, so we will not provide detailed comments here. In the table, we write $e \downarrow \mathsf{v}$ for the fact that expression e evaluates to v. Labels of the transition relation are ranged over by ℓ, ... and are generated by the following grammar:

$$\ell \ ::= \ c?m \ \mid \ c!m \ \mid \ c!(d)$$

where m, ... ranges over *messages*, namely basic values and channel names. Actions of the form $c?m$ and $c!m$ are often called *free inputs* and *free outputs* respectively. Actions of the form $c!(d)$ are called *bound outputs* and represent the extrusion of a private channel, d in this case. We use these actions to model session initiations, whereby a private channel is exchanged and subsequently used for the actual

Table 4: Transitions of processes.

$$P \oplus Q \longrightarrow P \qquad \star P \longrightarrow \star P \mid P \qquad c?(x).P \xrightarrow{c?d} P\{{}^{d}/_{x}\} \qquad c!d.P \xrightarrow{c!d} P \qquad \frac{\mathsf{v} : t}{c?(x:t).P \xrightarrow{c?\mathsf{v}} P\{{}^{\mathsf{v}}/_{x}\}}$$

$$\frac{e \downarrow \mathsf{v}}{c!e.P \xrightarrow{c!\mathsf{v}} P} \qquad \frac{P \longrightarrow P'}{P + Q \longrightarrow P' + Q} \qquad \frac{P \xrightarrow{\ell} P'}{P + Q \xrightarrow{\ell} P'} \qquad \frac{P \xrightarrow{c!m} P' \qquad Q \xrightarrow{c?m} Q'}{P \mid Q \longrightarrow P' \mid Q'}$$

$$\frac{P \xrightarrow{c!(d)} P' \qquad Q \xrightarrow{c?d} Q' \qquad d \not\in \mathtt{fn}(Q)}{P \mid Q \longrightarrow (\nu d)(P' \mid Q')} \qquad \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q} \qquad \frac{P \xrightarrow{\ell} P' \qquad \mathtt{bn}(\ell) \cap \mathtt{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\ell} P' \mid Q}$$

$$\frac{P \longrightarrow P'}{(\nu d)P \longrightarrow (\nu d)P'} \qquad \frac{P \xrightarrow{\ell} P' \qquad d \not\in \mathtt{fn}(\ell) \cup \mathtt{bn}(\ell)}{(\nu d)P \xrightarrow{\ell} (\nu d)P'} \qquad \frac{P \xrightarrow{c!d} P' \qquad c \neq d}{(\nu d)P \xrightarrow{c!(d)} P'}$$

interaction. Notions of free and bound names in labels are standard, with $\mathtt{fn}(c?d) = \mathtt{fn}(c!d) = \{c, d\}$, $\mathtt{fn}(c?\mathsf{v}) = \mathtt{fn}(c!\mathsf{v}) = \mathtt{fn}(c!(d)) = \{c\}$, and $\mathtt{bn}(c!(d)) = \{d\}$, the other sets being empty.

We remark only two distinctive features of the transition relation: (1) the replicated process $\star P$ evolves by means of an internal transition to $\star P \mid P$; technically this makes $\star P$ a *divergent process*, but the fact that we work with a fair semantics makes this only a detail; (2) similarly to the transition relation for session types, the transition relation for processes selects branches of external choices according to the type of the basic value being communicated. This is evident in the transitions for $c?(x:t).P$, which are labeled by values of type t.

The typing rules for the process language are inductively defined in Table 5. Judgments have the form $\Gamma \vdash P : \Delta$ or $\Gamma \vdash_u P : \Delta$, where Γ is a standard environment mapping variables to basic types and Δ is an environment mapping channel names and channel variables to session types. We write $\mathtt{dom}(\Delta)$ for the domain of Δ. The u annotation in some judgments is used to constraint the way actions can be composed. Rule (T-WEAK) allows one to enrich Δ with assumptions of the form $u : \mathbf{1}$, indicating that a process does not use the channel u. The premise $u \not\in \mathtt{dom}(\Delta)$ implies $u \not\in \mathtt{fn}(P)$ since it is always the case that $\mathtt{fn}(P) \subseteq \mathtt{dom}(\Delta)$. Rule (T-SUB) is an almost standard subsumption rule regarding the type of a channel u. The peculiarity is that it works "the other way round" by allowing a session type τ to become a smaller session type σ. The intuition is that P behaves according to τ on the channel u. Thus, it is safe to declare that the session type associated with u is even less deterministic than τ. We will see that this rule is fundamental in the type system since many other rules impose equality constraints on session types that can only be satisfied by finding a lower bound to two or more session types. It should also be appreciated the importance of using $\sqsubseteq$, which is a precongruence, since this allows us to apply rule (T-SUB) in arbitrary contexts. Rule (T-RES) types restrictions, by requiring the session type associated with the restricted channel to be of the form $\mathbf{1} + \sigma$. In light of rule (T-SUB) and of Proposition 2.1(2), this requirement imposes that the session type of a restricted channel c must be complete. Namely, there must not be communication errors on c. Rule (T-NIL) types the idle process $\mathtt{0}$ with the empty session environment. The process $\mathtt{0}$ should not be confused with the failed session type $\mathbf{0}$: the former is the successfully terminated process that does not use any channel; the latter denotes a communication error or a deadlock. Rules annotated with a channel u regard communications. The annotation indicates the

Table 5: Typing rules for processes.

$$\textsc{T-WEAK}\ \frac{\Gamma \vdash P : \Delta \qquad u \notin \mathtt{dom}(\Delta)}{\Gamma \vdash P : \Delta \cup \{u : \mathbf{1}\}} \qquad \textsc{T-SUB}\ \frac{\Gamma \vdash P : \Delta \cup \{u : \tau\} \qquad \sigma \sqsubseteq \tau}{\Gamma \vdash P : \Delta \cup \{u : \sigma\}} \qquad \textsc{T-RES}\ \frac{\Gamma \vdash P : \Delta \cup \{c : \mathbf{1} + \sigma\}}{\Gamma \vdash (\nu c)P : \Delta}$$

$$\textsc{T-NIL}\ \frac{}{\Gamma \vdash \mathtt{0} : \emptyset} \qquad \textsc{T-COMM}\ \frac{\Gamma \vdash_u P : \Delta}{\Gamma \vdash P : \Delta} \qquad \textsc{T-INPUT}\ \frac{\Gamma, x : t \vdash P : \Delta \cup \{u : \sigma\}}{\Gamma \vdash_u u?(x:t).P : \Delta \cup \{u : ?t.\sigma\}} \qquad \textsc{T-INPUTS}\ \frac{\Gamma \vdash P : \{x : \rho\}}{\Gamma \vdash_u u?(x).P : \{u : ?\rho.\mathbf{1}\}}$$

$$\textsc{T-OUTPUT}\ \frac{\Gamma \vdash e : t \qquad \Gamma \vdash P : \Delta \cup \{u : \sigma\}}{\Gamma \vdash_u u!e.P : \Delta \cup \{u : !t.\sigma\}} \qquad \textsc{T-OUTPUTS}\ \frac{\Gamma \vdash P : \Delta \cup \{u : \sigma, v : \tau\}}{\Gamma \vdash_u u!v.P : \Delta \cup \{u : !\rho.\sigma, v : \tau \mid \rho\}}$$

$$\textsc{T-EXT}\ \frac{\Gamma \vdash_u P : \Delta \cup \{u : \sigma\} \qquad \Gamma \vdash_u Q : \Delta \cup \{u : \tau\}}{\Gamma \vdash_u : P + Q : \Delta \cup \{u : \sigma + \tau\}} \qquad \textsc{T-INT}\ \frac{\Gamma \vdash P : \Delta \qquad \Gamma \vdash Q : \Delta}{\Gamma \vdash P \oplus Q : \Delta}$$

$$\textsc{T-BANG}\ \frac{\Gamma \vdash P : \{u_i : \sigma_i{}^{i \in I}\} \qquad \sigma_i \sqsubseteq \sigma_i \mid \sigma_i{}^{i \in I}}{\Gamma \vdash \star P : \{u_i : \sigma_i{}^{i \in I}\}} \qquad \textsc{T-PAR}\ \frac{\Gamma \vdash P : \{u_i : \sigma_i{}^{i \in I}\} \qquad \Gamma \vdash Q : \{u_i : \tau_i{}^{i \in I}\}}{\Gamma \vdash P \mid Q : \{u_i : \sigma_i \mid \tau_i{}^{i \in I}\}}$$

channel on which the communication occurs. Rule (T-INPUT) types an input action for basic values of type t. The assumption $x : t$ is moved into the environment Γ and if the session type associated with u in the continuation P is σ, then the overall behavior of P on u is described by $?t.\sigma$. Rule (T-OUTPUT) is similar, but regards output actions of basic values. We assume an unspecified set of deduction rules for judgments of the form $\Gamma \vdash e : t$, denoting that the expression e has type t in the environment Γ. Rule (T-INPUTS) types an input action for a channel x. The continuation P must be typed in a session environment of the form $\{x : \rho\}$, requiring that P must not refer to (free) channels other than the received one. Consequently, the whole process behaves according to the session type $?\rho.\mathbf{1}$. The severe restriction on the continuation process is necessary for type preservation, as we will see in Example 3.4 below. Rule (T-OUTPUTS) types delegations, whereby a channel v is sent over another channel u. This rule expresses clearly the idea of projection we are pursuing in our approach: the delegated channel v is used in the continuation P according to the session type τ (which may be $\mathbf{1}$ in case rule (T-WEAK) is applied); at the same time, the channel v is delegated to another process which will behave on it according to ρ. As a consequence, the overall behavior on v is expressed by the composition of τ and ρ, namely by $\tau \mid \rho$. If u is used in the continuation P according to σ, then its type is $!\rho.\sigma$ in the conclusion. Rule (T-EXT) types external choices. These can only be performed with respect to a single channel u by combining the session types associated with u in the two branches. The u annotation makes sure that both branches are guarded by actions involving u, as this is the only case where this type rule is sound (see example 3.3). Rule (T-COMM) strips the annotation u off the judgment, allowing one to re-enter the domain of judgments without annotations. Observe that the continuation process P in the rules (T-INPUT), (T-OUTPUT), (T-INPUTS), and (T-OUTPUTS) is typed by a judgment without the annotation, which is safe because P is guarded by a prefix. Rule (T-INT) types internal choices, but only when the two branches do have the same session environment. This can be achieved by repeated applications of rules (T-WEAK) and (T-SUB). Rule (T-PAR) types the parallel composition of processes. Again this rules shows the idea of projection and, unlike other session type systems, allows (actually requires) both processes to use exactly the same channels, whose corresponding session types are composed with $|$. In this context rule (T-WEAK) can be used to enforce that the session environments for P and Q be exactly the same, recalling that $\mathbf{1}$ is neutral for $|$. Finally, rule (T-BANG) types replicated processes: the basic idea is that a replicated process $\star P$ is well typed if any channel it uses is "unlimited" (in the terminology of [11]), which in our case translates to the property that it must be equal to (actually smaller than) two compositions of itself. $\mathbf{1}$ is the simplest session type with this property, but there are others as we will see in Example 3.1.

Example 3.1 (persistent service provider)**.** Consider the process

$$Q \equiv \star server?(x).P$$

which accepts an unbounded number of connection requests on the channel *server* and processes them in the process P. Assume we can type the non-replicated process as follows:

$$\frac{\Gamma \vdash P : \{x : \rho\}}{\Gamma \vdash server?(x).P : \{server : ?\rho.\mathbf{1}\}}$$

To apply rule (T-BANG) for Q we need *server* to have a type σ such that $\sigma \sqsubseteq \sigma \,|\, \sigma$, and $?\rho.\mathbf{1}$ clearly does not have this property. Consider the session type σ that is solution of the equation $X = \mathbf{1} \oplus ?\rho.X$. We have $\sigma \sqsubseteq ?\rho.\mathbf{1}$ and furthermore $\sigma \sqsubseteq \sigma \,|\, \sigma$. Hence we can now type Q with an application of rule (T-SUB) followed by (T-BANG). ■

Example 3.2 (multi-party session)**.** Intuitively, a multi-party session is a conversation taking place on a restricted channel that is shared between three or more participants. Consider a system $(\nu a)(P \,|\, P \,|\, Q)$ where

$$\begin{array}{rcl} P & \stackrel{\text{def}}{=} & a?(x).x?(y : \mathtt{Int}).(x!\mathit{isprime}(y) + x?(z : \mathtt{abort})) \\ Q & \stackrel{\text{def}}{=} & (\nu c)(a!c.c!n \,|\, a!c.c!n \,|\, c?(x : \mathtt{Bool}).c!\mathtt{abort}) \end{array}$$

the idea being that the two instances of P represent two servers checking whether a number is prime. The process Q establishes a connection by sending the two servers a fresh channel c and sending on this channel some integer number n. The two servers are thus able to process the number in parallel and the first one that succeeds sends the result back to Q. Upon reception of the result from one of the servers, Q notifies the other server by sending a dummy value `abort`, which we assume is a singleton type inhabited only by `abort` itself.

It is easy to verify that, within P, the channel x has type $\sigma = ?\mathtt{Int}.(!\mathtt{Bool}.\mathbf{1} + ?\mathtt{abort}.\mathbf{1})$ and a is used according to the type $?\sigma.\mathbf{1}$. In Q, a is used according to the type $!\sigma.\mathbf{1} \,|\, !\sigma.\mathbf{1}$ and c is used according to the type $\sigma \,|\, !\mathtt{Int}.\mathbf{1} \,|\, \sigma \,|\, !\mathtt{Int}.\mathbf{1} \,|\, ?\mathtt{Bool}.!\mathtt{abort}.\mathbf{1}$. Hence, the overall type of a is $!\sigma.\mathbf{1} \,|\, !\sigma.\mathbf{1} \,|\, ?\sigma.\mathbf{1} \,|\, ?\sigma.\mathbf{1}$ and the whole system is well typed since both a's type and c's type are complete. ■

The type system permits to find type derivations for processes using channels with a non-viable session type. Examples of such processes are $c?(x : \mathbb{0}).\mathtt{0}$. A non-viable session type indicates an intrinsic flaw in the process. It is thus unsurprising that viability of the session types in the session environment is an essential requirement for the type preservation property. We say that a session environment Δ is viable if so is every session type in its codomain.

Theorem 3.1 (subject reduction)**.** *Let* $\Gamma \vdash P : \Delta$ *and* $P \longrightarrow Q$ *and* Δ *viable. Then* $\Gamma \vdash Q : \Delta$.

Before addressing local progress, we justify by means of examples the two main constraints imposed by the type system in order to guarantee type preservation.

Example 3.3. To justify the use of u-annotations in rule (T-EXT), consider the process

$$P \stackrel{\text{def}}{=} a?(x:\mathtt{Int}).b?(y:\mathtt{Bool}) + b?(x:\mathtt{Int}).a?(y:\mathtt{Bool})$$

and suppose it well typed, where $a : ?\mathtt{Int}.\mathbf{1} + ?\mathtt{Bool}$ and $b : ?\mathtt{Bool}.\mathbf{1} + ?\mathtt{Int}.\mathbf{1}$. Apparently, *both* a and b are able to receive either an integer or a Boolean value and a system such as $(\nu a)(\nu b)(P \mid a!3 \mid b!3)$ would be well typed. Alas, the external choices in the types of a and b do not take into account the fact that any synchronization of P with another process may actually *disable* one branch in these choices. The reduction

$$(\nu a)(\nu b)(P \mid a!3 \mid b!3) \longrightarrow (\nu a)(\nu b)(b?(y:\mathtt{Bool}) \mid \mathtt{0} \mid b!3)$$

leads to an ill-typed process, since b has type $?\mathtt{Bool}.\mathbf{1} \mid !\mathtt{Int}.\mathbf{1}$ which is not complete. ■

Example 3.4. The severe constraint in the premise of rule (T-INPUTS) can be justified by looking at the following processes:

$$\begin{aligned} P &\stackrel{\text{def}}{=} a!c.a!c.c?(x:\mathtt{Int}).c?(y:\mathtt{Bool}) \\ Q &\stackrel{\text{def}}{=} a?(x).a?(y).y!\mathtt{true}.x!3 \end{aligned}$$

where P can be typed with a derivation like the following:

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash c?(x:\mathtt{Int}).c?(y:\mathtt{Bool}) : \{a:\mathbf{1}, c: ?\mathtt{Int}.?\mathtt{Bool}.\mathbf{1}\}}}{\Gamma \vdash a!c.c?(x:\mathtt{Int}).c?(y:\mathtt{Bool}) : \{a: !(!\mathtt{Bool}.\mathbf{1}).\mathbf{1}, c: !\mathtt{Bool}.\mathbf{1} \mid ?\mathtt{Int}.?\mathtt{Bool}.\mathbf{1}\}}}{\Gamma \vdash a!c.a!c.c?(x:\mathtt{Int}).c?(y:\mathtt{Bool}) : \{a: !(!\mathtt{Int}.\mathbf{1}).!(!\mathtt{Bool}.\mathbf{1}).\mathbf{1}, c: !\mathtt{Int}.\mathbf{1} \mid !\mathtt{Bool}.\mathbf{1} \mid ?\mathtt{Int}.?\mathtt{Bool}.\mathbf{1}\}}$$

The process P delegates the channel c twice on a. The first time, the delegated behavior is $!\mathtt{Int}.\mathbf{1}$, while the second time the delegated behavior is $!\mathtt{Bool}.\mathbf{1}$. Each time c is delegated, P assumes that the receiving process will implement the delegated behavior. However, as it can be clearly seen in the conclusion of the typing derivation above, the overall delegated behavior of c is $!\mathtt{Int}.\mathbf{1} \mid !\mathtt{Bool}.\mathbf{1}$, namely the parallel composition of the two behaviors that were separately delegated. This is fundamental for the completeness of c's type, since the input operations performed by the residual of P at the top of the typing derivation occur in a specific order.

The process Q, which receives both delegations, is unaware that x and y will be instantiated with the same channel. So, Q is well typed and x and y have respectively type $!\mathtt{Bool}.\mathbf{1}$ and $!\mathtt{Int}.\mathbf{1}$, as requested by P, but Q uses these channels in a specific order that is not captured by the projections. The process $P \mid Q$ deadlocks in two steps:

$$P \mid Q \Longrightarrow c?(x:\mathtt{Int}).c?(y:\mathtt{Bool}) \mid c!\mathtt{true}.c!3$$

where in the final state we have $c : ?\mathtt{Int}.?\mathtt{Bool}.\mathbf{1} \mid !\mathtt{Bool}.!\mathtt{Int}.\mathbf{1}$ which is not complete. By requiring, in the premise of rule (T-INPUTS), that the receiving process cannot use any channel other than the received one, we are basically imposing that the receiving process must handle every received channel in a thread of its own. ■

In judgments of the form $\Gamma \vdash P : \Delta$ the environment Δ is an approximation of P insofar as it describes the projections of P's behavior with respect to the channel it uses and delegates. It is well known that this approximation is unable to capture situations where well-typed processes deadlock because the interdependence between communications occurring on different channels are lost. Our approach is no exception, as shown by the following example.

Example 3.5 (deadlock)**.** Consider the system

$$(\nu a)(\nu b)(a!3.b?(x:\mathtt{Bool}) \mid b!\mathtt{true}.a?(x:\mathtt{Int}))$$

where the channels a and b have respectively type $\sigma = !\mathtt{Int}.\mathbf{1} \mid ?\mathtt{Int}.\mathbf{1}$ and $\tau = ?\mathtt{Bool}.\mathbf{1} \mid !\mathtt{Bool}.\mathbf{1}$. In both cases we have $\mathbf{1}+\sigma \sqsubseteq \sigma$ and $\mathbf{1}+\tau \sqsubseteq \tau$, hence the system is well typed but deadlock. ■

The progress property we are able to state is a *local* one, in the sense that a synchronization on some channel c is guaranteed to eventually occur provided that all processes awaiting for interactions on c are ready to do so. This notion of "readiness" is captured by the following definition:

Definition 3.1 (readiness)**.** We say that P is *ready* on c if $P \downarrow c$ is derivable by the rules:

$$\pi.P \downarrow \mathtt{subj}(\pi) \qquad \frac{c \notin \mathtt{fn}(P)}{P \downarrow c} \qquad \frac{P \downarrow c \qquad Q \downarrow c}{P+Q \downarrow c} \qquad \frac{P \downarrow c \qquad Q \downarrow c}{P \mid Q \downarrow c} \qquad \frac{P \downarrow c \qquad c \neq d}{(\nu d)P \downarrow c}$$

where we write $\mathtt{subj}(\pi)$ for the *subject* of π (the channel on which the communication occurs).

Intuitively, P is ready on c if either it does not use c, in which case it plays no role in any synchronization on c, or if P is prefixed by an action whose subject is c, or if every branch of P is ready on c. Observe that when $P \equiv P_1 + P_2$, *both* branches are required to be ready on c. This is not overly restrictive because, by rule (T-EXT), if either branch is prefixed by an action whose subject is c, so must be the other branch.

Theorem 3.2. *If* $\Gamma \vdash P : \Delta \cup \{c:\sigma\}$ *and* σ *complete and* $P \downarrow c$*, then either* $c \notin \mathtt{fn}(P)$ *or* $P \longrightarrow$*.*

4 Concluding remarks

It may sound obvious to say that session types are behavioral types. Yet, although session types are normally associated with channels, channels do not expose any behavior. The solution of this apparently innocuous paradox lays in the equally obvious observation that the session type associated with a channel *reflects* the behavior of a *process* concerning the input/output operations that the process performs on that channel. By taking this mirrored point of view we have been able to define a simple and, in our opinion, elegant theory of session types that generalizes, unifies, and semantically justifies many concepts that can be found scattered in the current literature: (multi-party) session types are terms of a suitably defined process algebra closely based on value-passing CCS; *completeness* expresses the property that a session is well-formed and never yields a communication error; *duality* [11] $\sigma \bowtie \tau$ is the special case where $\sigma \mid \tau$ is complete; *viability* captures the concept of well-typed process, namely of process that can be composed with others in order to implement complete sessions; the *subtyping relation* between session types arises semantically by relating those session types that preserve completeness in arbitrary contexts.

The adoption of a fair testing semantics [18] for session types is original to the best of our knowledge. In fact, most presentations of session types rely on notions of duality or well-formed composition where the only concern is the absence of communication errors, while the fairness principle we adopt imposes an additional constraint: that at any time a conversation is always able to reach a so-called successful state. Whether or not this is desirable in practice, from a technical point of view there are both pros and cons: on the one hand, the fair subsession relation is more difficult to characterize coinductively and axiomatically because fairness escapes the mere structure of types; on the other hand, the subsession relation is an all-in-one tool that incorporates safe substitutability (rule (T-SUB)), viability, and completeness (Proposition 2.1). We have been unable to fully characterize completeness in terms of a non-fair subsession relation (see [4] for an attempt in the context of behavioral contracts).

The type system we have provided as a proof-of-concept in Section 3 may look excessively restrictive, in particular with respect to the rule (T-INPUTS) which demands that the continuation cannot use any (known) session if not the received one. We have three observations regarding this point: (1) this is a direct consequence of our focus on the idea of projected behavior, which allows a more liberal use of channels; (2) similar restrictions can be found in type systems guaranteeing global progress [8, 1, 3]; (3) the provided type system is very natural and simple, considering the freedom it leaves in the use of channels; this simplicity suggests that it can be smoothly extended with features such as polarities or roles which would likely help relaxing the constraints. We leave this extension as future work.

References

[1] L. Bettini, M. Coppo, L. D'Antoni, M. De Luca, M. Dezani-Ciancaglini, and N. Yoshida. Global Progress in Dynamically Interleaved Multiparty Sessions. In *CONCUR'08*, LNCS 5201. Springer, 2008.

[2] L. Caires and H. Vieira. Conversation types. In *ESOP'09*, LNCS 5502. Springer, 2009.

[3] G. Castagna, M. Dezani-Ciancaglini, E. Giachino, and L. Padovani. Foundations of session types. Available at `http://www.sti.uniurb.it/padovani/Papers/FoundationsSessionTypes.pdf`, 2009.

[4] G. Castagna and L. Padovani. Contracts for mobile processes. Available at `http://www.sti.uniurb.it/padovani/publications.html`, 2009.

[5] S. Chaki, S. K. Rajamani, and J. Rehof. Types as models: model checking message-passing programs. *SIGPLAN Not.*, 37(1):45–57, 2002.

[6] B. Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25:95–169, 1983.

[7] R. De Nicola and M. Hennessy. CCS without τ's. In *TAPSOFT'87/CAAP'87*, LNCS 249. Springer, 1987.

[8] M. Dezani-Ciancaglini, U. de' Liguoro, and N. Yoshida. On Progress for Structured Communications. In G. Barthe and C. Fournet, editors, *TGC'07*, LNCS 4912. Springer, 2008.

[9] M. Dezani-Ciancaglini, S. Drossopoulou, D. Mostrous, and N. Yoshida. Session Types for Object-Oriented Languages. *Information and Computation*, 207(5):595–641, 2009.

[10] S. Drossopoulou, M. Dezani-Ciancaglini, and M. Coppo. Amalgamating the Session Types and the Object Oriented Programming Paradigms. In *MPOOL'07*, 2007.

[11] S. Gay and M. Hole. Subtyping for session types in the π-calculus. *Acta Informatica*, 42(2-3):191–225, 2005.

[12] S. Gay and V. Vasconcelos. Asynchronous functional session types. Technical Report 2007–251, Department of Computing, University of Glasgow, 2007.

[13] M. Hennessy. *Algebraic Theory of Processes*. Foundation of Computing. MIT Press, 1988.

[14] K. Honda. Types for dyadic interaction. In *CONCUR'93*, LNCS 715, 1993.

[15] K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP'98*, LNCS 1381. Springer, 1998.

[16] K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL'08*, 2008.

[17] H. R. Nielson and F. Nielson. Higher-order concurrent programs with finite communication topology (extended abstract). In *POPL'94*, New York, NY, USA, 1994. ACM Press.

[18] A. Rensink and W. Vogler. Fair testing. *Inf. Comput.*, 205(2):125–198, 2007.

[19] V. Vasconcelos. Fundamentals of session types. In *SFM'09*, LNCS 5569. Springer, 2009.

[20] V. Vasconcelos, S. Gay, and A. Ravara. Type checking a multithreaded functional language with session types. *Theoretical Computer Science*, 368, 2006.

Acknowledgments. I wish to thank Giuseppe Castagna for having provided comments on an early draft of this paper.

A Supplement to Section 2

We introduce some additional notation that is used in this and the following appendixes: we write $\sum_{i\in\{1,\dots,n\}} \sigma_i$ in place of $\sigma_1 + \cdots + \sigma_n$ with the convention that $\sum_{i\in\emptyset} \sigma_i = \mathbf{0}$. Similarly, we write $\bigoplus_{i\in\{1,\dots,n\}} \sigma_i$ for $\sigma_1 \oplus \cdots \oplus \sigma_n$, assuming that $n > 0$ (the internal choice has no neutral element).

Definition A.1 (ground actions)**.** Let $\lfloor\cdot\rfloor$ be the function over actions that is the identity except that $\lfloor ?\rho \rfloor = ?\diamond$ and $\lfloor !\rho \rfloor = !\diamond$ for every ρ. We say that $\lfloor \mu \rfloor$ is the *ground action* of μ; we write $\sigma \xrightarrow{\lfloor\mu\rfloor} \sigma'$ if $\sigma \xrightarrow{\mu} \sigma'$.

The transition relation of session types in Table 2 is *subjective* in the sense that it expresses the behavior of a process from the point of view of the process itself. For example, we have $?\mathtt{Int}.\sigma + ?\mathtt{Real}.\tau \xrightarrow{?3} \sigma$. Namely, the process has read an integer value 3 and it knows that the left branch of the external choice has been selected. In the following it is useful to define an *objective* transition relation, from the point of view of the processes that are interacting with another process exposing a certain behavior. This objective transition relation will be expressed as a *continuation* of a session type with respect to a specified action. On the one hand, for example, we have $(?\mathtt{Int}.\sigma + ?\mathtt{Real}.\tau)(?3) = \sigma \oplus \tau$, namely the interacting processes do not know which branch has been selected. On the other hand, we have $(?\mathtt{Int}.\sigma + ?\mathtt{Real}.\tau)(?\sqrt{2}) = \tau$.

Definition A.2 (continuation)**.** Let $\sigma \overset{\mu}{\Longrightarrow}$. The *continuation* of σ with respect to μ is defined as $\sigma(\mu) \overset{\text{def}}{=} \bigoplus_{\sigma \overset{\lfloor\mu\rfloor}{\Longrightarrow} \sigma'} \sigma'$. We generalize this notion to finite sequences of actions $\mu_1,\dots,\mu_n$ so that $\sigma(\varepsilon) = \sigma$ and $\sigma(\mu\mu_1\cdots\mu_n) = \sigma(\mu)(\mu_1\cdots\mu_n)$.

Not every action exposed by a session type is important as far as viability is concerned. For example, $\sigma \overset{\text{def}}{=} \mathbf{1} + ?\mathtt{Int}.\mathbf{0}$ describes a process that is capable of reading any integer value, but after doing so there is no way to interact successfully with that process. In this sense, we say that σ is not ready on $?\mathsf{v}$ whenever $\mathsf{v} : \mathtt{Int}$. However, σ is ready on $\checkmark$, and this is what it makes σ viable. In general, we say that σ is ready on some (ground) action μ if there is a session type ρ that completes σ and may synchronize with σ when σ emits μ. To this aim we introduce a symmetric *duality relation* between (ground) actions such that:

$$\checkmark\#\checkmark \qquad ?\mathsf{v}\#!\mathsf{v} \qquad ?\rho\#!\rho' \qquad ?\diamond\#!\diamond$$

Observe that $?\rho$ and $!\rho'$ are dual regardless of ρ and ρ', since a synchronization is always possible, although the continuation is not necessarily viable.

Definition A.3 (readiness)**.** We say that σ is ready on μ, notation $\sigma \downarrow \mu$, if $\sigma \overset{\mu}{\Longrightarrow}$ and there exist ρ and μ' such that $\mu\#\mu'$ and $\rho \overset{\mu'}{\Longrightarrow}$ and $\sigma \,|\, \rho$ is complete.

Proof of Proposition 2.1(1). The "if" part is trivial. Suppose σ is not viable. We want to prove that $\sigma + \tau \preceq \tau$ for every τ. Suppose that this is not the case. Then there exists ρ such that $\sigma + \tau \,|\, \rho$ is complete and $\tau \,|\, \rho$ is not complete. Then $\sigma \,|\, \rho$ is complete, which contradicts σ not viable. □

Proposition A.1. *Let σ be viable. The following properties hold:*

1. *if $\sigma \Longrightarrow \tau$, then $\sigma \preceq \tau$;*
2. *if $\sigma \preceq \tau$ and τ is not complete, then σ is not complete;*
3. *if $\sigma \preceq \tau$ and $\tau \overset{\mu}{\Longrightarrow}$ and $\mu \neq \checkmark$, then $\sigma \overset{\lfloor\mu\rfloor}{\Longrightarrow}$ and $\sigma \downarrow \mu$ implies $\sigma(\mu) \preceq \tau(\mu)$.*

Proof. As regards item (1), observe that $\sigma \mid \rho$ complete implies $\tau \mid \rho$ complete, hence $\sigma \preceq \tau$.

As regards item (2), suppose that τ is not complete and σ is complete. Then $\sigma \mid \mathbf{1}$ is complete and $\tau \mid \mathbf{1}$ is not, which is absurd.

As regards item (3), from the hypothesis σ viable we know that there exists ρ such that $\sigma \mid \rho$ is complete. We reason by cases on μ for defining a session type ρ':

- if $\mu = \dagger \mathsf{v}$, then let $\rho' \stackrel{\text{def}}{=} \rho + \overline{\dagger}\mathsf{v}.\mathbf{0}$;
- if $\mu = \dagger \rho$, then let $\rho' \stackrel{\text{def}}{=} \rho + \overline{\dagger}\mathbf{0}.\mathbf{0}$.

Now suppose, by contradiction, that $\sigma \overset{\lfloor\mu\rfloor}{\not\Longrightarrow}$. We have that $\sigma \mid \rho'$ is complete and $\tau \mid \rho'$ is not, which is absurd. Hence $\sigma \overset{\lfloor\mu\rfloor}{\Longrightarrow}$.

Suppose $\sigma \downarrow \mu$ and let ρ be such that $\sigma \mid \rho$ is complete and $\rho \overset{\mu'}{\Longrightarrow}$ where $\mu \# \mu'$. Let ρ' be an arbitrary session type such that $\sigma(\mu) \mid \rho'$ is complete. The existence of ρ' is guaranteed by the existence of ρ. Let ρ'' be the same as ρ, except that every term $\alpha.\theta$ that generates μ' actions is replaced by $\alpha.\rho'$. We have $\sigma \mid \rho''$ complete, from which we deduce $\tau \mid \rho''$ complete. Hence, $\tau(\mu) \mid \rho'$ is complete. We conclude $\sigma(\mu) \preceq \tau(\mu)$. □

Proof of Theorem 2.1. ("if" part) If $\sigma \sqsubseteq \mathbf{0}$, then by Proposition 2.1(1) we have that σ is not viable, hence $\sigma \preceq \tau$. If $\sigma \sqsubseteq \tau$, then $\sigma + \mathbf{0} \preceq \tau + \mathbf{0}$ and we conclude by observing that $\sigma + \mathbf{0} \approx \sigma$ and $\tau + \mathbf{0} \approx \tau$.

("only if" part) Suppose $\sigma \preceq \tau$ and $\sigma \not\sqsubseteq \mathbf{0}$ and suppose, by contradiction, that $\sigma \not\sqsubseteq \tau$. Namely, there exists σ' such that $\sigma + \sigma' \not\preceq \tau + \sigma'$. Then, there exists ρ such that $\sigma + \sigma' \mid \rho$ is complete and $\tau + \sigma' \mid \rho$ is not complete. This happens if there exists a derivation $\tau + \sigma' \mid \rho \Longrightarrow \tau'' \mid \rho' \overset{\checkmark}{\not\Longrightarrow}$. Without loss of generality, we may assume $\rho' \overset{\checkmark}{\Longrightarrow}$ and $\tau'' \overset{\checkmark}{\not\Longrightarrow}$. By unzipping this derivation we obtain that there exist two sequences of actions $\mu_1, \dots, \mu_n$ and $\mu_1', \dots, \mu_n'$ such that $\mu_i \# \mu_i'$ for every $1 \le i \le n$ and $\tau + \sigma' \overset{\mu_1' \cdots \mu_n'}{\Longrightarrow} \tau''$ and $\rho \overset{\mu_1 \cdots \mu_n}{\Longrightarrow} \rho'$. Suppose $n > 0$. By Proposition A.1 and from the hypotheses $\sigma \preceq \tau$ and σ viable we have that $\tau + \sigma' \overset{\lfloor\mu_1'\rfloor}{\Longrightarrow}$ implies $\sigma + \sigma' \overset{\lfloor\mu_1'\rfloor}{\Longrightarrow}$. From the hypotheses $\sigma + \sigma' \mid \rho$ complete and $\rho \overset{\lfloor\mu_1\rfloor}{\Longrightarrow}$ where $\mu_1 \# \mu_1'$ we deduce that $(\sigma + \sigma')(\mu_1') \mid \rho(\mu_1)$ is complete. Furthermore, $\tau \overset{\lfloor\mu_1'\rfloor}{\Longrightarrow}$ implies $\sigma \overset{\lfloor\mu_1'\rfloor}{\Longrightarrow}$ and $\sigma(\mu_1') \preceq \tau(\mu_1')$. From the hypothesis $\tau'' \mid \rho' \overset{\checkmark}{\not\Longrightarrow}$ we deduce that $(\tau + \sigma')(\mu_1') \mid \rho(\mu_1)$ is not complete, since $(\tau + \sigma')(\mu_1') \overset{\mu_2' \cdots \mu_n'}{\Longrightarrow} \tau''$ and $\rho(\mu_1) \overset{\mu_2 \cdots \mu_n}{\Longrightarrow} \rho'$. This is absurd, since we can show that $(\sigma + \sigma')(\mu_1') \preceq (\tau + \sigma')(\mu_1')$ recalling that $\preceq$ is a precongruence with respect to $\oplus$. We distinguish the following subcases:

- If $\tau \overset{\lfloor\mu_1'\rfloor}{\Longrightarrow}$ and $\sigma' \overset{\lfloor\mu_1'\rfloor}{\Longrightarrow}$, then $(\sigma + \sigma')(\mu_1') = \sigma(\mu_1') \oplus \sigma'(\mu_1') \preceq \tau(\mu_1') \oplus \sigma'(\mu_1') = (\tau + \sigma')(\mu_1')$.
- If $\tau \overset{\lfloor\mu_1'\rfloor}{\Longrightarrow}$ and $\sigma' \overset{\lfloor\mu_1'\rfloor}{\not\Longrightarrow}$, then $(\sigma + \sigma')(\mu_1') = \sigma(\mu_1') \preceq \tau(\mu_1') = (\tau + \sigma')(\mu_1')$.
- If $\tau \overset{\lfloor\mu_1'\rfloor}{\not\Longrightarrow}$ and $\sigma' \overset{\lfloor\mu_1'\rfloor}{\Longrightarrow}$ and $\sigma \overset{\lfloor\mu_1'\rfloor}{\Longrightarrow}$, then $(\sigma + \sigma')(\mu_1') = \sigma(\mu_1') \oplus \sigma'(\mu_1') \preceq \sigma'(\mu_1') = (\tau + \sigma')(\mu_1')$.
- If $\tau \overset{\lfloor\mu_1'\rfloor}{\not\Longrightarrow}$ and $\sigma' \overset{\lfloor\mu_1'\rfloor}{\Longrightarrow}$ and $\sigma \overset{\lfloor\mu_1'\rfloor}{\not\Longrightarrow}$, then $(\sigma + \sigma')(\mu_1') = \sigma'(\mu_1') = (\tau + \sigma')(\mu_1')$.

Suppose $n = 0$. Then $\tau + \sigma' \Longrightarrow \tau' + \sigma'' \equiv \tau''$. From $\tau'' \overset{\checkmark}{\not\Longrightarrow}$ we deduce $\tau' \overset{\checkmark}{\not\Longrightarrow}$ and $\sigma'' \overset{\checkmark}{\not\Longrightarrow}$. From Proposition A.1(2) and the hypotheses $\sigma \preceq \tau$ and σ viable we deduce that there exists σ''' such that $\sigma \Longrightarrow \sigma'''$ and $\sigma''' \overset{\checkmark}{\not\Longrightarrow}$. Then $\sigma + \sigma' \mid \rho \Longrightarrow \sigma''' + \sigma'' \mid \rho' \overset{\checkmark}{\not\Longrightarrow}$, contradicting the hypothesis $\sigma + \sigma' \mid \rho$ complete. □

Proof of Proposition 2.1(2). We prove that σ is complete if and only if $\mathbf{1}+\sigma \preceq \sigma$. The statement then follows from Theorem 2.1, since $\mathbf{1}+\sigma$ is viable ($\mathbf{1}+\sigma \mid \mathbf{1}$ is complete). As regards the "if" part, suppose by contradiction that $\mathbf{1}+\sigma \preceq \sigma$ and σ is not complete. Then $\mathbf{1} \mid \mathbf{1}+\sigma$ is complete and $\mathbf{1} \mid \sigma$ is not, which is absurd. Hence σ is complete. As regards the "only if" part, suppose by contradiction that σ is complete and $\mathbf{1}+\sigma \not\preceq \sigma$. Then, there exists ρ such that $\mathbf{1}+\sigma \mid \rho$ is complete and $\sigma \mid \mathbf{1}$ is not. Suppose that there exist μ and μ' such that $\mu\#\mu'$ and $\sigma \stackrel{\mu}{\Longrightarrow}$ and $\rho \stackrel{\mu'}{\Longrightarrow}$ and $\sigma(\mu) \mid \rho(\mu')$ is not complete. Then $\mathbf{1}+\sigma \mid \rho$ is not complete since $(\mathbf{1}+\sigma)(\mu) = \sigma(\mu)$, which is absurd. Hence, there exist σ' and ρ' such that $\sigma \Longrightarrow \sigma'$ and $\rho \Longrightarrow \rho'$ and σ' and ρ' never synchronize and $\sigma' \mid \rho' \stackrel{\checkmark}{\not\Longrightarrow}$, meaning either $\sigma' \stackrel{\checkmark}{\not\Longrightarrow}$ or $\rho' \stackrel{\checkmark}{\not\Longrightarrow}$. Then either σ is not complete or $\mathbf{1}+\sigma \mid \rho$ is not complete, which is absurd. Hence we conclude $\mathbf{1}+\sigma \preceq \sigma$. ☐

B Supplement to Section 3

Lemma B.1 (substitution)**.** *The following properties hold:*

1. *If* $\Gamma, x : t \vdash P : \Delta$ *and* $\mathsf{v} : t$*, then* $\Gamma \vdash P\{\mathsf{v}/x\} : \Delta$.
2. *If* $\Gamma \vdash P : \Delta \cup \{x : \sigma\}$ *and* $d \notin \mathtt{fn}(P)$*, then* $\Gamma \vdash P\{d/x\} : \Delta \cup \{d : \sigma\}$.

Proof. A simple induction on the typing derivation. ☐

Proposition B.1. *Let* $\Gamma \vdash P : \Delta \cup \{c : \mathbf{1}\}$ *and* $c \notin \mathtt{fn}(P)$*. Then* $\Gamma \vdash P : \Delta$.

Proof. A simple induction on the derivation of $\Gamma \vdash P : \Delta \cup \{c : \mathbf{1}\}$. ☐

Lemma B.2 (subject reduction)**.** *The following properties hold:*

1. *if* $\Gamma \vdash P : \Delta$ *and* $P \longrightarrow Q$ *and* Δ *is viable, then* $\Gamma \vdash Q : \Delta$*;*
2. *if* $\Gamma \vdash P : \Delta \cup \{c : \sigma\}$ *and* $P \xrightarrow{c\dagger\mathsf{v}} Q$*, then* $\sigma \stackrel{\dagger\mathsf{v}}{\Longrightarrow}$ *and* $\sigma \downarrow \dagger\mathsf{v}$ *implies* $\Gamma \vdash Q : \Delta \cup \{c : \sigma(\dagger\mathsf{v})\}$.
3. *if* $\Gamma \vdash P : \Delta \cup \{c : \sigma, d : \tau\}$ *and* $P \xrightarrow{c?d} Q$*, then there exists* ρ *such that* $\sigma \stackrel{?\rho}{\Longrightarrow}$ *and* $\sigma \downarrow ?\diamond$ *implies* $\Gamma \vdash Q : \Delta \cup \{c : \sigma(?\diamond), d : \tau \mid \rho\}$*;*
4. *if* $\Gamma \vdash P : \Delta \cup \{c : \sigma, d : \tau\}$ *and* $P \xrightarrow{c!d} Q$*, then there exist* τ' *and* ρ *such that* $\tau = \tau' \mid \rho$ *and* $\sigma \stackrel{!\rho}{\Longrightarrow}$ *and* $\sigma \downarrow !\diamond$ *implies* $\Gamma \vdash Q : \Delta \cup \{c : \sigma(!\diamond), d : \tau'\}$*;*
5. *if* $\Gamma \vdash P : \Delta \cup \{c : \sigma\}$ *and* $d \notin \mathtt{dom}(\Delta)$ *and* $P \xrightarrow{c!(d)} Q$*, then there exist* τ' *and* ρ *such that* $\tau' \mid \rho$ *is complete and* $\sigma \stackrel{!\rho}{\Longrightarrow}$ *and* $\sigma \downarrow !\diamond$ *implies* $\Gamma \vdash Q : \Delta \cup \{c : \sigma(!\diamond), d : \tau'\}$.

Proof. By induction on the derivation of $P \longrightarrow Q$ or $P \xrightarrow{\mu} Q$ and by cases on the structure of P. By Proposition A.1(3) we may assume that the typing derivation for P does not end with an application of rule (T-SUB) and consequently is syntax directed except possibly for applications of rule (T-WEAK). Also, note that the constraints on the domain of the session environment in items (3) and (5) are not restrictive. In item (3), if d does not belong to the session environment it can be added with an application of rule (T-WEAK). In item (3), if $d \in \mathtt{dom}(\Delta)$ then $d \notin \mathtt{fn}(P)$ for otherwise P would not be able to emit a bound output action $c!(d)$. Consequently, we can remove d from the session environment by means of Proposition B.1.

($P \equiv Q \oplus P' \longrightarrow Q$) From the hypothesis of rule (T-INT) we conclude $\Gamma \vdash Q : \Delta$.

($P \equiv \star P' \longrightarrow \star P' \mid P' \equiv Q$) From rule (T-BANG) we deduce $\sigma \sqsubseteq \sigma \mid \sigma$ for every $u : \sigma \in \Delta$. From (T-PAR) and (T-SUB) we conclude $\Gamma \vdash Q : \Delta$.

($P \equiv c?(x:t).P' \xrightarrow{c?\mathsf{v}} P'\{\mathsf{v}/_x\} \equiv Q$ **where** $\mathsf{v} : t$) From rule (T-INPUT) we deduce $\sigma = ?t.\sigma'$ and $\Gamma, x : t \vdash P' : \Delta \cup \{c : \sigma'\}$. By Lemma B.1 we obtain $\Gamma \vdash Q : \Delta \cup \{c : \sigma'\}$ and we conclude by observing that $\sigma(?\mathsf{v}) = (?t.\sigma')(?\mathsf{v}) = \sigma'$.

($P \equiv c?(x).P' \xrightarrow{c?d} P'\{d/_x\} \equiv Q$) From rules (T-INPUTS) and (T-WEAK) we deduce $\sigma = ?\rho.\mathbf{1}$ for some ρ and $\tau = \mathbf{1}$ and $u : \theta \in \Delta$ implies $\theta = \mathbf{1}$ and $\Gamma \vdash P' : \{x : \rho\}$. Since $d \notin \mathtt{fn}(P')$, by Lemma B.1 we obtain $\Gamma \vdash Q : \{d : \rho\}$. By repeated applications of rule (T-WEAK) we deduce $\Gamma \vdash Q : \Delta \cup \{c : \mathbf{1}, d : \tau \mid \rho\}$. We conclude by observing that $\sigma \overset{?\rho}{\Longrightarrow}$ and $\sigma(?\diamond) = \mathbf{1}$.

($P \equiv c!e.Q \xrightarrow{c!\mathsf{v}} Q$ **where** $e \downarrow \mathsf{v}$) From rule (T-OUTPUT) we deduce $\sigma = !t.\sigma'$ where $\Gamma \vdash e : t$ and $\Gamma \vdash Q : \Delta \cup \{c : \sigma'\}$. We conclude by observing that $\sigma \overset{!\mathsf{v}}{\Longrightarrow}$ and $\sigma(!\mathsf{v}) = \sigma'$.

($P \equiv c!d.Q \xrightarrow{c!d} Q$) From rule (T-OUTPUTS) we deduce $\sigma = !\rho.\sigma'$ there exists τ' such that $\tau = \tau' \mid \rho$ and $\Gamma \vdash Q : \Delta \cup \{c : \sigma', d : \tau'\}$. We conclude by observing that $\sigma(!\diamond) = \sigma'$.

($P \equiv P_1 + P_2 \longrightarrow P_1' + P_2 \equiv Q$ **where** $P_1 \longrightarrow P_1'$) From rule (T-EXT) we deduce that there exists c such that $\Delta = \Delta' \cup \{c : \sigma_1 + \sigma_2\}$ and $\Gamma \vdash_c P_1 : \Delta' \cup \{c : \sigma_1\}$ and $\Gamma \vdash_c P_2 : \Delta' \cup \{c : \sigma_2\}$. By induction hypothesis we deduce $\Gamma \vdash_c P_1' : \Delta' \cup \{c : \sigma_1\}$. From rule (T-EXT) we conclude $\Gamma \vdash_c Q : \Delta$.

($P \equiv P_1 + P_2 \xrightarrow{c\dagger\mathsf{v}} Q$ **where** $P_1 \xrightarrow{c\dagger\mathsf{v}} Q$) From rule (T-EXT) we deduce $\sigma = \sigma_1 + \sigma_2$ and $\Gamma \vdash_c P_i : \Delta \cup \{c : \sigma_i\}$ for $i \in \{1,2\}$. By induction hypothesis we deduce $\sigma_1 \overset{\dagger\mathsf{v}}{\Longrightarrow}$ and ($\sigma_1 \downarrow \dagger\mathsf{v}$ implies $\Gamma \vdash Q : \Delta \cup \{c : \sigma_1(\dagger\mathsf{v})\}$). We conclude by observing that $\sigma \overset{\dagger\mathsf{v}}{\Longrightarrow}$ and $\sigma \downarrow \dagger\mathsf{v}$ implies $\sigma_1 \downarrow \dagger\mathsf{v}$ and $\sigma(\dagger\mathsf{v}) \sqsubseteq \sigma_1(\dagger\mathsf{v})$ and by rule (T-SUB).

($P \equiv P_1 + P_2 \xrightarrow{c?d} Q$ **where** $P_1 \xrightarrow{c?d} Q$) From rule (T-EXT) we deduce $\sigma = \sigma_1 + \sigma_2$ where $\Gamma \vdash_c P_i : \Delta \cup \{c : \sigma_i, d : \tau\}$ for $i \in \{1,2\}$. By induction hypothesis we deduce that there exists ρ such that $\sigma_1 \overset{?\rho}{\Longrightarrow}$ and ($\sigma_1 \downarrow ?\diamond$ implies $\Gamma \vdash_c P_1' : \Delta \cup \{c : \sigma_1(?\diamond), d : \tau \mid \rho\}$). We observe that $\sigma \overset{?\rho}{\Longrightarrow}$ and $\sigma \downarrow ?\diamond$ implies $\sigma_1 \downarrow ?\diamond$ and $\sigma(?\diamond) \sqsubseteq \sigma_1(?\diamond)$. We conclude by rule (T-SUB).

$P \equiv P_1 + P_2 \xrightarrow{c!d} Q$ **where** $P_1 \xrightarrow{c!d} Q$ From rule (T-EXT) we deduce $\sigma = \sigma_1 + \sigma_2$ where $\Gamma \vdash_c P_i : \Delta \cup \{c : \sigma_i, d : \tau\}$ for $i \in \{1,2\}$. By induction hypothesis we deduce that there exist τ' and ρ such that $\tau = \tau' \mid \rho$ and $\sigma_1 \overset{!\rho}{\Longrightarrow}$ and ($\sigma_1 \downarrow !\diamond$ implies $\Gamma \vdash Q : \Delta \cup \{c : \sigma_1(!\diamond), d : \tau'\}$). Observe that $\sigma \overset{!\rho}{\Longrightarrow}$ and $\sigma \downarrow !\rho$ implies $\sigma_1 \downarrow !\rho$ and $\sigma(!\diamond) \sqsubseteq \sigma_1(!\diamond)$. We conclude by rule (T-SUB).

$P \equiv P_1 + P_2 \xrightarrow{c!(d)} Q$ **where** $P_1 \xrightarrow{c!(d)} Q$ From rule (T-EXT) we deduce $\sigma = \sigma_1 + \sigma_2$ where $\Gamma \vdash_c P_i : \Delta \cup \{c : \sigma_i\}$ for $i \in \{1,2\}$. By induction hypothesis we deduce that there exist τ' and ρ such that $\tau' \mid \rho$ is complete and $\sigma_1 \overset{!\rho}{\Longrightarrow}$ and ($\sigma_1 \downarrow !\diamond$ implies $\Gamma \vdash Q : \Delta \cup \{c : \sigma_1(!\diamond), d : \tau'\}$). Observe that $\sigma \overset{!\rho}{\Longrightarrow}$ and $\sigma \downarrow !\rho$ implies $\sigma_1 \downarrow !\rho$ and $\sigma(!\diamond) \sqsubseteq \sigma_1(!\diamond)$. We conclude by rule (T-SUB).

($P \equiv P_1 \mid P_2 \longrightarrow P_1' \mid P_2 \equiv Q$ **where** $P_1 \longrightarrow P_1'$) From rule (T-PAR) we deduce $\Gamma \vdash P_i : \{u_j : \sigma_{ij}\ {}^{j\in I}\}$ for $i \in \{1,2\}$. By induction hypothesis we deduce $\Gamma \vdash P_1' : \{u_j : \sigma_{ij}\ {}^{j\in I}\}$. By rule (T-PAR) we conclude $\Gamma \vdash Q : \Delta$.

($P \equiv P_1 \mid P_2 \longrightarrow P_1' \mid P_2'$ **where** $P_1 \xrightarrow{c!\mathsf{v}} P_1'$ **and** $P_2 \xrightarrow{c?\mathsf{v}} P_2'$) From rule (T-PAR) we deduce $\Delta = \{u_j : \sigma_{1j} \mid \sigma_{2j}\ {}^{j\in I}\} \cup \{c : \sigma_1 \mid \sigma_2\}$ and $\Gamma \vdash P_i : \{u_j : \sigma_{ij}\ {}^{j\in I}\} \cup \{c : \sigma_i\}$ for $i \in \{1,2\}$. By induction hypothesis we deduce $\sigma_1 \overset{!\mathsf{v}}{\Longrightarrow}$ and $\sigma_2 \overset{?\mathsf{v}}{\Longrightarrow}$ and ($\sigma_1 \downarrow !\mathsf{v}$ implies $\Gamma \vdash P_1' : \{u_j : \sigma_{1j}\ {}^{j\in I}\} \cup \{c : \sigma_1(!\mathsf{v})\}$) and

$(\sigma_2 \downarrow ?\mathsf{v}$ implies $\Gamma \vdash P_2' : \{u_j : \sigma_{2j}\ {}^{j\in I}\} \cup \{c : \sigma_2(?\mathsf{v})\})$. From $\sigma_1 \mid \sigma_2$ viable we deduce $\sigma_1 \downarrow !\mathsf{v}$ and $\sigma_2 \downarrow ?\mathsf{v}$. Observe that $\sigma_1 \mid \sigma_2 \sqsubseteq \sigma_1(!\mathsf{v}) \mid \sigma_2(?\mathsf{v})$. From (T-PAR) and (T-SUB) we conclude $\Gamma \vdash Q : \Delta$.

($P \equiv P_1 \mid P_2 \longrightarrow P_1' \mid P_2'$ where $P_1 \xrightarrow{c!d} P_1'$ and $P_2 \xrightarrow{c?d} P_2'$) From rule (T-PAR) we deduce $\Delta = \{u_j : \sigma_{1j} \mid \sigma_{2j}\ {}^{j\in I}\} \cup \{c : \sigma_1 \mid \sigma_2, d : \tau_1 \mid \tau_2\}$ and $\Gamma \vdash P_i : \{u_j : \sigma_j\ {}^{j\in I}\} \cup \{c : \sigma_i, d : \tau_i\}$ for $i \in \{1,2\}$. By induction hypothesis on P_1 we deduce that there exist τ_1' and ρ_1 such that $\tau_1 = \tau_1' \mid \rho_1$ and $\sigma_1 \overset{!\rho}{\Longrightarrow}$ and $(\sigma_1 \downarrow !\diamond$ implies $\Gamma \vdash P_1' : \{u_j : \sigma_{1j}\ {}^{j\in I}\} \cup \{c : \sigma_1(!\diamond), d : \tau_1'\})$. By induction hypothesis on P_2 we deduce that there exists ρ_2 such that $\sigma_2 \overset{?\rho_2}{\Longrightarrow}$ and $(\sigma_2 \downarrow ?\diamond$ implies $\Gamma \vdash P_2 : \{u_j : \sigma_{2j}\ {}^{j\in I}\} \cup \{c : \sigma_2(?\diamond), d : \tau_2 \mid \rho_2\})$. From $\sigma_1 \mid \sigma_2$ viable we deduce $\rho_1 \preceq \rho_2$. From τ_1 viable we deduce ρ_1 viable. By Theorem 2.1 we deduce $\rho_1 \sqsubseteq \rho_2$. By rule (T-PAR) we deduce $\Gamma \vdash Q : \{u_j : \sigma_{1j} \mid \sigma_{2j}\ {}^{j\in I}\} \cup \{c : \sigma_1(!\diamond) \mid \sigma_2(?\diamond), d : \tau_1' \mid \tau_2 \mid \rho_2\}$. We conclude from $\sigma_1 \mid \sigma_2 \sqsubseteq \sigma_1(!\diamond) \mid \sigma_2(?\diamond)$ and $\tau_1 \mid \tau_2 = \tau_1' \mid \rho_1 \mid \tau_2 \sqsubseteq \tau_1' \mid \tau_2 \mid \rho_2$ and rule (T-SUB).

($P \equiv P_1 \mid P_2 \longrightarrow (\nu d)(P_1' \mid P_2')$ where $P_1 \xrightarrow{c!(d)} P_1'$ and $P_2 \xrightarrow{c?d} P_2'$ and $d \notin \mathtt{fn}(P_2)$) From rule (T-PAR) we deduce $\Delta = \{u_j : \sigma_{1j} \mid \sigma_{2j}\ {}^{j\in I}\} \cup \{c : \sigma_1 \mid \sigma_2, d : \tau_1 \mid \tau_2\}$ and $\Gamma \vdash P_i : \{u_j : \sigma_{ij}\ {}^{j\in I}\} \cup \{c : \sigma_i, d : \tau_i\}$ for $i \in \{1,2\}$. Furthermore, it must be $\tau_1 = \tau_2 = \mathbf{1}$ since $d \notin \mathtt{fn}(P_1) \cup \mathtt{fn}(P_2)$. By induction hypothesis on P_1 we deduce that there exist τ_1' and ρ_1 such that $\tau_1' \mid \rho_1$ is complete and $\sigma_1 \overset{!\rho_1}{\Longrightarrow}$ and $(\sigma_1 \downarrow !\diamond$ implies $\Gamma \vdash P_1' : \{u_j : \sigma_{1j}\ {}^{j\in I}\} \cup \{c : \sigma_1(!\diamond), d : \tau_1'\})$. By induction hypothesis on P_2 we deduce that there exists ρ_2 such that $\sigma_2 \overset{?\rho_2}{\Longrightarrow}$ and $(\sigma_2 \downarrow ?\diamond$ implies $\Gamma \vdash P_2' : \{u_j : \sigma_{2j}\ {}^{j\in I}\} \cup \{c : \sigma_2(?\diamond), d : \tau_2 \mid \rho_2\})$. From $\sigma_1 \mid \sigma_2$ viable we deduce $\rho_1 \preceq \rho_2$ and $\sigma_1 \downarrow !\diamond$ and $\sigma_2 \downarrow ?\diamond$. From $\tau_1' \mid \rho_1$ complete we deduce ρ_1 viable. By Theorem 2.1 we deduce $\rho_1 \sqsubseteq \rho_2$. By rule (T-PAR) we deduce $\Gamma \vdash P_1' \mid P_2' : \{u_j : \sigma_{1j} \mid \sigma_{2j}\ {}^{j\in I}\} \cup \{c : \sigma_1(!\diamond) \mid \sigma_2(?\diamond), d : \tau_1' \mid \rho_2\}$. From $\tau_1' \mid \rho_1$ complete and $\rho_1 \sqsubseteq \rho_2$ we deduce $\tau_1' \mid \rho_2$ complete. We have $\sigma_1 \mid \sigma_2 \sqsubseteq \sigma_1(!\diamond) \mid \sigma_2(?\diamond)$. We conclude by rules (T-RES) and (T-SUB).

($P \equiv P_1 \mid P_2 \xrightarrow{c\dagger\mathsf{v}} P_1' \mid P_2 \equiv Q$ where $P_1 \xrightarrow{c\dagger\mathsf{v}} P_1'$) From rule (T-PAR) we deduce $\Delta = \{u_j : \sigma_{1j} \mid \sigma_{2j}\ {}^{j\in I}\}$ and $\sigma = \sigma_1 \mid \sigma_2$ and $\Gamma \vdash P_i : \{u_j : \sigma_{ij}\ {}^{j\in I}\} \cup \{c : \sigma_i\}$ for $i \in \{1,2\}$. By induction hypothesis we deduce $\sigma_1 \overset{\dagger\mathsf{v}}{\Longrightarrow}$ and $(\sigma_1 \downarrow \dagger\mathsf{v}$ implies $\Gamma \vdash P_1' : \{u_j : \sigma_{1j}\ {}^{j\in I}\} \cup \{c : \sigma_1(\dagger\mathsf{v})\})$. We have $\sigma \overset{\dagger\mathsf{v}}{\Longrightarrow}$ and $\sigma \downarrow \dagger\mathsf{v}$ implies $\sigma_1 \downarrow \dagger\mathsf{v}$ and $\sigma(\dagger\mathsf{v}) \sqsubseteq \sigma_1(\dagger\mathsf{v}) \mid \sigma_2$. We conclude by rules (T-PAR) and (T-SUB).

($P \equiv P_1 \mid P_2 \xrightarrow{c?d} P_1' \mid P_2 \equiv Q$ where $P_1 \xrightarrow{c?d} P_1'$) From rule (T-PAR) we deduce $\Delta = \{u_j : \sigma_{1j} \mid \sigma_{2j}\ {}^{j\in I}\}$ and $\sigma = \sigma_1 \mid \sigma_2$ and $\tau = \tau_1 \mid \tau_2$ and $\Gamma \vdash P_i : \{u_j : \sigma_{ij}\ {}^{j\in I}\} \cup \{c : \sigma_i, d : \tau_i\}$ for $i \in \{1,2\}$. By induction hypothesis there exists ρ such that $\sigma_1 \overset{?\rho}{\Longrightarrow}$ and $(\sigma_1 \downarrow ?\diamond$ implies $\Gamma \vdash P_1' : \{u_j : \sigma_{1j}\ {}^{j\in I}\} \cup \{c : \sigma_1(?\diamond), d : \tau_1 \mid \rho\})$. Observe that $\sigma \overset{?\rho}{\Longrightarrow}$ and $\sigma \downarrow ?\diamond$ implies $\sigma_1 \downarrow ?\diamond$ and $\sigma(?\diamond) \sqsubseteq \sigma_1(?\diamond)$. We conclude by rules (T-PAR) and (T-SUB).

($P \equiv P_1 \mid P_2$ where $P_1 \xrightarrow{c!d} P_1'$ and $P_1' \mid P_2 \equiv Q$) From rule (T-PAR) we deduce $\Delta = \{u_j : \sigma_{1j} \mid \sigma_{2j}\ {}^{j\in I}\}$ and $\sigma = \sigma_1 \mid \sigma_2$ and $\tau = \tau_1 \mid \tau_2$ and $\Gamma \vdash P_i : \{u_j : \sigma_{ij}\ {}^{j\in I}\} \cup \{c : \sigma_i, d : \tau_i\}$ for $i \in \{1,2\}$. By induction hypothesis we deduce that there exist τ_1' and ρ such that $\tau_1 = \tau_1' \mid \rho$ and $\sigma_1 \overset{!\rho}{\Longrightarrow}$ and $(\sigma_1 \downarrow !\diamond$ implies $\Gamma \vdash P_1' : \{u_j : \sigma_{ij}\ {}^{j\in I}\} \cup \{c : \sigma_1(!\diamond), d : \tau_1'\})$. Observe that $\sigma \overset{!\rho}{\Longrightarrow}$ and $\sigma \downarrow !\diamond$ implies $\sigma_1 \downarrow !\diamond$ and $\sigma(!\diamond) \sqsubseteq \sigma_1(!\diamond)$. We conclude by rules (T-PAR) and (T-SUB) and by taking $\tau' = \tau_1' \mid \tau_2$.

($P \equiv P_1 \mid P_2$ where $P_1 \xrightarrow{c!(d)} P_1'$ and $P_1' \mid P_2 \equiv Q$) From rule (T-PAR) we deduce $\Delta = \{u_j : \sigma_{1j} \mid \sigma_{2j}\ {}^{j\in I}\}$ and $\sigma = \sigma_1 \mid \sigma_2$ and $\Gamma \vdash P_i : \{u_j : \sigma_{ij}\ {}^{j\in I}\} \cup \{c : \sigma_i\}$ for $i \in \{1,2\}$. By induction hypothesis we deduce that there exist τ' and ρ such that $\tau' \mid \rho$ is complete and $\sigma_1 \overset{!\rho}{\Longrightarrow}$ and $(\sigma_1 \downarrow !\diamond$ implies $\Gamma \vdash P_1' : \{u_j : \sigma_{ij}\ {}^{j\in I}\} \cup \{c : \sigma_1(!\diamond), d : \tau'\})$. From the hypothesis $d \notin \mathtt{dom}(\Delta)$ we deduce $d \notin \mathtt{fn}(P)$, hence

$d \notin \mathtt{fn}(P_2)$. From rule (T-WEAK) we deduce $\Gamma \vdash P_2 : \{u_j : \sigma_{2j}\ {}^{j\in I}\} \cup \{c : \sigma_2, d : \mathbf{1}\}$. Observe that $\sigma \overset{!\rho}{\Longrightarrow}$ and $\sigma \downarrow {!\diamond}$ implies $\sigma_1 \downarrow {!\diamond}$ and $\sigma(!\diamond) \sqsubseteq \sigma_1(!\diamond)$. We conclude by rules (T-PAR) and (T-SUB).

($P \equiv (\nu c)P' \longrightarrow (\nu c)P'' \equiv Q$ where $P' \longrightarrow P''$) From rule (T-RES) we deduce that $\Gamma \vdash P' : \Delta \cup \{c : \mathbf{1} + \sigma\}$ for some σ. By induction hypothesis we deduce $\Gamma \vdash P'' : \Delta \cup \{c : \mathbf{1} + \sigma\}$. By rule (T-RES) we conclude $\Gamma \vdash Q : \Delta$.

($P \equiv (\nu d)P' \xrightarrow{c\dagger \mathsf{v}} (\nu d)P'' \equiv Q$ where $c \neq d$ and $P' \xrightarrow{c\dagger \mathsf{v}} P''$) From rule (T-RES) we deduce that $\Gamma \vdash P' : \Delta \cup \{d : \mathbf{1} + \tau\}$ for some τ. By induction hypothesis we deduce $\sigma \overset{\dagger \mathsf{v}}{\Longrightarrow}$ and ($\sigma \downarrow \dagger\mathsf{v}$ implies $\Gamma \vdash P'' : \Delta \cup \{c : \sigma(\dagger\mathsf{v}), d : \mathbf{1} + \tau\}$. We conclude by rule (T-RES).

($P \equiv (\nu d)P' \xrightarrow{c!(d)} Q$ where $P' \xrightarrow{c!d} Q$ and $c \neq d$) From rule (T-RES) we deduce that there exists τ complete such that $\Gamma \vdash P' : \Delta \cup \{c : \sigma, d : \tau\}$. By induction hypothesis we conclude that there exist τ' and ρ such that $\tau = \tau' \mid \rho$ and $\sigma \overset{!\rho}{\Longrightarrow}$ and ($\sigma \downarrow {!\diamond}$ implies $\Gamma \vdash Q : \Delta \cup \{c : \sigma(!\diamond), d : \tau'\}$).

($P \equiv (\nu d')P' \xrightarrow{c?d} (\nu d')P'' \equiv Q$ where $P' \xrightarrow{c?d} P''$ and $d' \neq c, d$) From rule (T-RES) we deduce that there exists τ' complete such that $\Gamma \vdash P' : \Delta \cup \{c : \sigma, d : \tau, d' : \tau'\}$. By induction hypothesis we deduce that there exists ρ such that $\sigma \overset{?\rho}{\Longrightarrow}$ and ($\sigma \downarrow {?\diamond}$ implies $\Gamma \vdash P'' : \Delta \cup \{c : \sigma(?\diamond), d : \tau \mid \rho, d' : \tau'\}$). We conclude by rule (T-RES).

($P \equiv (\nu d')P' \xrightarrow{c!d} (\nu d')P'' \equiv Q$ where $P' \xrightarrow{c!d} P''$ and $d' \neq c, d$) From rule (T-RES) we deduce that there exists τ'' complete such that $\Gamma \vdash P' : \Delta \cup \{c : \sigma, d : \tau, d' : \tau''\}$. By induction hypothesis we deduce that there exist τ' and ρ such that $\tau = \tau' \mid \rho$ and $\sigma \overset{!\rho}{\Longrightarrow}$ and ($\sigma \downarrow {!\diamond}$ implies $\Gamma \vdash P'' : \Delta \cup \{c : \sigma(!\diamond), d : \tau', d' : \tau''\}$. We conclude by rule (T-RES).

($P \equiv (\nu d')P' \xrightarrow{c!(d)} (\nu d')P''$ where $P' \xrightarrow{c!(d)} P''$ and $d' \neq c, d$) Similar to the previous case. □

Theorem 3.1 is a special case of Lemma B.2. Before proving Theorem 3.2 we need to show that among all the possible typings for a process there is one that is more precise than the others, which we call *principal typing*. Unsurprisingly, this is obtained when rule (T-SUB) is never used at the bottom of the typing derivation for a process. In order to prove that every process has a principal type, we need to define the projection of the visible actions of a process P with respect to a specified channel c, which we denote with $\mathtt{init}(P)|_c$:

$$\mathtt{init}(P)|_c \overset{\text{def}}{=} \{\dagger\mathsf{v} \mid P \overset{c\dagger\mathsf{v}}{\Longrightarrow}\} \cup \{\dagger\diamond \mid \exists d : P \overset{c\dagger d}{\Longrightarrow}\} \cup \{!\diamond \mid \exists d : P \overset{c!(d)}{\Longrightarrow}\}$$

Lemma B.3. *If $\Gamma \vdash P : \Delta \cup \{c : \sigma\}$ and $P \downarrow c$ and $P \not\longrightarrow$, then there exists τ such that $\sigma \sqsubseteq \tau$ and $\tau \not\longrightarrow$ and $\Gamma \vdash P : \Delta \cup \{c : \tau\}$ and $\mathtt{init}(\tau) = \mathtt{init}(P)|_c \cup \{\checkmark\}$.*

Proof. By induction on the derivation of $\Gamma \vdash P : \Delta \cup \{c : \sigma\}$ and by cases on the last typing rule applied. Only two cases are shown. Cases for action prefixes are base cases. The other cases can be solved by trivial inductions.

(T-WEAK) Then either $\sigma = \mathbf{1}$, in which case we conclude by taking $\tau = \sigma$, or we conclude by induction hypothesis.

(T-SUB) Then there exists τ' such that $\sigma \sqsubseteq \tau'$ and $\Gamma \vdash P : \Delta \cup \{c : \tau'\}$. We conclude by induction hypothesis and transitivity of $\sqsubseteq$. □

Proof of Theorem 3.2. We prove that if $\Gamma \vdash P : \Delta \cup \{c : \sigma\}$ and $P \downarrow c$ and $P \longrightarrow\!\!\!\!\!\!/\;\;$ and σ complete, then $c \not\in \mathtt{fn}(P)$. We proceed by induction on P. Trivial cases are omitted.

$(P \equiv \pi.P')$ From σ complete and $P \downarrow c$ we conclude $\mathtt{subj}(\pi) \neq c$ and $c \not\in \mathtt{fn}(P)$.

$(P \equiv P_1 + P_2)$ Then $\Delta \cup \{c : \sigma\} = \Delta' \cup \{d : \sigma_1 + \sigma_2\}$ for some Δ' and $\Gamma \vdash_d P_i : \Delta' \cup \{d : \sigma_i\}$ (if d were equal to c the session type σ would not be complete, because no prefix action involving c is typed by a complete session type). From the hypotheses $P \downarrow c$ and $P \longrightarrow\!\!\!\!\!\!/\;\;$ we deduce $P_i \downarrow c$ and $P_i \longrightarrow\!\!\!\!\!\!/\;\;$ for $i \in \{1,2\}$. By induction hypothesis we derive that $c \not\in \mathtt{fn}(P_i)$ for $i \in \{1,2\}$, hence we conclude $c \not\in \mathtt{fn}(P)$.

$(P \equiv P_1 \,|\, P_2)$ Then $\Delta = \{u_j : \sigma_{1j} \,|\, \sigma_{2j}\ {}^{j \in I}\}$ and $\sigma = \sigma_1 \,|\, \sigma_2$ and $\Gamma \vdash P_i : \{u_j : \sigma_{ij}\ {}^{j \in I}\} \cup \{c : \sigma_i\}$ for $i \in \{1,2\}$. From Lemma B.3 we deduce that there exist τ_1 and τ_2 such that $\sigma_i \sqsubseteq \tau_i$ and $\tau_i \longrightarrow\!\!\!\!\!\!/\;\;$ and $\mathtt{init}(\tau_i) = \mathtt{init}(P_i)|_c \cup \{\checkmark\}$ and $\Gamma \vdash P_i : \{u_j : \sigma_{ij}\ {}^{j \in I}\} \cup \{c : \tau_i\}$. We deduce $\sigma_1 \,|\, \sigma_2 \sqsubseteq \tau_1 \,|\, \tau_2$. From the hypotheses $P \downarrow c$ and $P \longrightarrow\!\!\!\!\!\!/\;\;$ we deduce $P_i \downarrow c$ and $P_i \longrightarrow\!\!\!\!\!\!/\;\;$ for every $i \in \{1,2\}$. From the hypothesis $P \longrightarrow\!\!\!\!\!\!/\;\;$ we deduce that P_1 and P_2 cannot synchronize with each other. In other terms, $\mu \in \mathtt{init}(P_1)|_c$ and $\mu' \in \mathtt{init}(P_2)|_c$ implies not $\mu \# \mu'$. This means that the same holds for the actions other than $\checkmark$ in $\mathtt{init}(\tau_1)$ and $\mathtt{init}(\tau_2)$. From σ complete we deduce $\tau_1 \,|\, \tau_2$ complete. From $\tau_i \longrightarrow\!\!\!\!\!\!/\;\;$ and the fact that τ_1 and τ_2 cannot synchronize with each other we deduce τ_i complete for every $i \in \{1,2\}$. By induction hypothesis we deduce $c \not\in \mathtt{fn}(P_i)$ for every $i \in \{1,2\}$. We conclude $c \not\in \mathtt{fn}(P)$. □