IRIS AperTO

UNIVERSITÀ DEGLI STUDI DI TORINO

AperTO - Archivio Istituzionale Open Access dell'Università di Torino

On Projecting Processes into Session Types

This is the author's manuscript

Original Citation:

Availability:

This version is available http://hdl.handle.net/2318/80856 since 2016-11-21T09:17:27Z

Published version:

DOI:10.1017/S0960129511000405

Terms of use:

Open Access

Anyone can freely access the full text of works made available as "Open Access". Works made available under a Creative Commons license can be used according to the terms and conditions of said license. Use of all other works requires consent of the right holder (author or publisher) if not exempted from copyright protection by the applicable law.

(Article begins on next page)

19 March 2024



UNIVERSITÀ DEGLI STUDI DI TORINO

This is an author version of the contribution published on:

Luca Padovani
On Projecting Processes into Session Types
MATHEMATICAL STRUCTURES IN COMPUTER SCIENCE (2012)
22(2)
DOI: 10.1017/S0960129511000405

The definitive version is available at:
http://www.journals.cambridge.org/abstract_S0960129511000405

On Projecting Processes into Session Types

Luca Padovani

Dipartimento di Informatica, Università degli Studi di Torino

We define session types as projections of the behavior of processes with respect to the operations processes perform on channels. This calls for a parallel composition operator over session types denoting the simultaneous access to a channel by two or more processes. The proposed approach allows us to define a semantically grounded theory of session types that does not require the linear usage of channels. However, type preservation and progress can only be guaranteed for processes that never receive channels they already own. A number of examples show that the resulting framework validates existing session type theories and unifies them to some extent.

1. Introduction

Session types are an increasingly popular technique for describing structured communications in distributed systems. In these systems, processes engage into a conversation by first establishing a *private session* and then carrying on the conversation within the protected scope of the session. The session type *prescribes*, for each process involved in the session, the sequence and type of messages the process is allowed to send or expected to receive at each given time. Several theories of session types have been put forward, each characterized by a combination of features. The arity of the session is restricted to two processes in the seminal works by Honda [1993] and Honda *et al.* [1998], it has been relaxed to an arbitrary, but fixed number of processes in multi-party session types [Honda *et al.*, 2008], and it is unrestricted in conversation types [Caires and Vieira, 2009], where processes may dynamically join and leave the session. Processes may be modeled as bare terms of some process algebra, typically some dialect of the π-calculus, or as threads in functional languages [Vasconcelos *et al.*, 2006; Gay and Vasconcelos, 2007] and object-oriented calculi (see Drossopoulou *et al.* [2007]; Dezani-Ciancaglini *et al.* [2009] for just a few examples). Session type theories vary also in the properties they are able to enforce, ranging from basic type safety to local and global progress properties [Dezani-Ciancaglini *et al.*, 2008; Bettini *et al.*, 2008].

In this paper we take a step back and try to change the perspective: we *define* session types as *projections* of processes. In particular, we slice the behavior of a process according to the channels it uses and define the session type associated with a channel as the behavior of the process *restricted* to that channel, where the actual messages sent and received by the process are approximated by the corresponding types. To illustrate this idea consider a simplified version of the "two buyer protocol" from [Honda *et al.*, 2008] where two processes, $Buyer_1$ and $Buyer_2$,

cooperate for purchasing an item from a *Seller* process, which is modeled like this:

$$\mathit{Seller} = a?(x).x?(\mathit{item} : \texttt{string}).x!\mathit{price}(\mathit{item}).(x?\texttt{false} + x?\texttt{true}.x?(\mathit{addr} : \texttt{string}))$$

The seller waits for purchase requests on some public channel a, on which it receives a private channel x – the session channel – where the rest of the conversation takes place. On this channel the seller receives the name of an item the buyer wants to purchase, it sends out its price, and then waits for a decision from the buyer ($+$ denotes an external choice), which is signalled here with a boolean value: `false` means that the buyer has refused to buy the item and the conversation terminates immediately; `true` means that the buyer has agreed to buy the item, so the seller waits for the address to which the item should be shipped and then terminates.

By projecting *Seller* we obtain

$$\vdash \mathit{Seller} : \{a : ?\rho.\mathbf{1}\}$$

where $?\rho.\mathbf{1}$, the session type associated with a, states that *Seller* uses a for receiving another channel on which it commits to behave according to ρ. The $\mathbf{1}$ denotes that *Seller* no longer uses a after this action. The session type

$$\rho = ?\texttt{string}.!\texttt{real}.(?\texttt{false}.\mathbf{1} + ?\texttt{true}.?\texttt{string}.\mathbf{1})$$

is in fact the projection of *Seller*'s continuation after action $a?(x)$ with respect to x, but x is not visible in *Seller*'s projection since it is bound within the process. Observe that the session type only mentions the type of the messages exchanged by the process (in this particular example, `false` and `true` are singleton types corresponding to the boolean values).

The two buyers are modeled with the following terms:

$$\begin{aligned}\mathit{Buyer}_1 &= (\mathsf{new}\ c)a!c.c!\text{“The Origin of Species”}.b!c\\ \mathit{Buyer}_2 &= b?(x).x?(\mathit{price} : \texttt{real}).(x!\texttt{false} \oplus x!\texttt{true}.x!\mathit{address})\end{aligned}$$

The first buyer creates a fresh channel c – basically, a new session – and sends it to the seller. On c it also sends the name of the item to buy and then *delegates* c to the second buyer, which thus becomes responsible for carrying out the rest of the conversation. The second buyer, once it has received the session channel from the first buyer, waits for the price of the item and then decides ($\oplus$ denotes an internal choice) whether to buy the item, by sending `true` to the seller followed by the shipping address, or to refuse the item by sending `false` to the seller.

The projection of Buyer_2 is analogous to that of *Seller* and yields

$$\vdash \mathit{Buyer}_2 : \{b : ?\theta.\mathbf{1}\} \qquad \text{where} \qquad \theta = ?\texttt{real}.(!\texttt{false}.\mathbf{1} \oplus !\texttt{true}.!\texttt{string}.\mathbf{1})$$

whereas process Buyer_1 results in the projection

$$\vdash \mathit{Buyer}_1 : \{a : !\rho.\mathbf{1}, b : !\theta.\mathbf{1}\}$$

Channels a and b are used for delegating c. When c is delegated on a it is because Buyer_1 expects the receiver to behave according to the type ρ we have determined above, while when c is delegated on b it is because Buyer_1 expects the receiver to behave according to the type θ above. Since c is bound within Buyer_1 it does not appear in its in projection, nonetheless we may try to argue about the projection of the sub-process $a!c.c!\text{“}\ldots\text{”}.b!c$ within Buyer_1 with respect to c: this sub-process uses c directly just for sending a string. In addition, having delegated c to

other processes, it is as if the sub-process also implements the delegated behaviors by itself. In other words we have

$$\vdash a!c.c!\text{“The Origin of Species”}.b!c : \{a : !\rho.\mathbf{1}, b : !\theta.\mathbf{1}, c : \rho \mid !\mathtt{string}.(\theta \mid \mathbf{1})\}$$

Indeed, ρ is the behavior delegated to *Seller*. After this delegation, the process sends a string on channel c and finally it delegates the behavior θ to $Buyer_2$. When a process sends a channel to another process, the two processes must be running in parallel. Consequently, after the communication we end up with two (or more) processes acting on the delegated channel simultaneously. Thus, the overall projection of all of these processes on the channel c is the *parallel composition* of the projections of the single processes on c. Using the same argument we obtain the following projection of the whole system:

$$\vdash Seller \mid Buyer_1 \mid Buyer_2 : \{a : ?\rho.\mathbf{1} \mid !\rho.\mathbf{1}, b : !\theta.\mathbf{1} \mid ?\theta.\mathbf{1}\}$$

This projection exercise leads to a number of observations. The first one is that we can use projections for deducing properties of systems, such as the absence of communication errors. In this respect, observe that the session types associated with a and b have a symmetric structure and are made of complementary actions. It is thus natural to think of session types as if they were processes, where complementary actions synchronize, and derive the reductions

$$?\rho.\mathbf{1} \mid !\rho.\mathbf{1} \longrightarrow \mathbf{1} \mid \mathbf{1} \qquad \text{and} \qquad !\theta.\mathbf{1} \mid ?\theta.\mathbf{1} \longrightarrow \mathbf{1} \mid \mathbf{1}$$

proving that at no time a process sends a message that the other process is unwilling to receive and symmetrically no process starves from some message that is never sent. Both session types eventually reduce to parallel compositions of $\mathbf{1}$'s representing the fact that the channels they are associated with are no longer involved in any interaction. The channel c is more critical, since it is restricted and hence inaccessible by other processes. For this reason we expect that its session type is *complete* just like $?\rho.\mathbf{1} \mid !\rho.\mathbf{1}$ and $!\theta.\mathbf{1} \mid ?\theta.\mathbf{1}$ are in the sense described above. Indeed we have

$$\begin{aligned}
\rho \mid !\mathtt{string}.(\theta \mid \mathbf{1}) &\longrightarrow !\mathtt{real}.(?\mathtt{false}.\mathbf{1} + ?\mathtt{true}.?\mathtt{string}.\mathbf{1}) \mid \theta \mid \mathbf{1} \\
&\longrightarrow (?\mathtt{false}.\mathbf{1} + ?\mathtt{true}.?\mathtt{string}.\mathbf{1}) \mid (!\mathtt{false}.\mathbf{1} \oplus !\mathtt{true}.!\mathtt{string}.\mathbf{1}) \mid \mathbf{1} \\
&\longrightarrow (?\mathtt{false}.\mathbf{1} + ?\mathtt{true}.?\mathtt{string}.\mathbf{1}) \mid !\mathtt{false}.\mathbf{1} \mid \mathbf{1} \longrightarrow \mathbf{1} \mid \mathbf{1} \mid \mathbf{1}
\end{aligned}$$

and we obtain another successful derivation if, in the third reduction, we choose the right branch of the internal choice rather than the left one. Overall we may deduce, by looking at the projection, that the system is free from communication errors. What we cannot deduce is, in general, that the system enjoys progress (although the presented one does), since the projection abstracts away from the order in which different channels are used.

The second observation is that we can use the projection of a system to compare its implementation against some desired specification, or to relate it to a different implementation. For instance, we may argue that $\rho \mid !\mathtt{string}.(\theta \mid \mathbf{1})$ and $\rho \mid !\mathtt{string}.\theta$ are equivalent as far as completeness is concerned. While in the latter session type it is no longer evident that the channel is being delegated, from the point of view of *Seller* it makes no difference whether it is a single buyer engaged in the conversation or if there are many of them, provided that their overall behavior is compatible with that exposed by the seller. As another example, consider the process

$(\mathsf{new}\ c)Buyer_3$ where

$$Buyer_3 = a!c.c!\text{“The Origin of Species”}.c?(price : \mathtt{real}).c!\mathtt{true}.c!address$$

modeling a wealthy (or naive) buyer who always accepts the offer from the seller. Its projection with respect to channel c is

$$\vdash Buyer_3 : \{a : !\rho.\mathbf{1}, c : \rho \mid !\mathtt{string}.?\mathtt{real}.!\mathtt{true}.!\mathtt{string}.\mathbf{1}\}$$

which resembles the projection of $Buyer_1$ obtained above, but is not quite the same. In particular, it looks as if $!\mathtt{string}.?\mathtt{real}.!\mathtt{true}.!\mathtt{string}.\mathbf{1}$ is a more deterministic version of $!\mathtt{string}.(\theta \mid \mathbf{1})$ since the latter may refuse the *Seller*'s offer, while the former cannot. This comparison between different degrees of nondeterminism, which we call *subsession* and denote with

$$!\mathtt{string}.(\theta \mid \mathbf{1}) \preceq !\mathtt{string}.?\mathtt{real}.!\mathtt{true}.!\mathtt{string}.\mathbf{1}$$

in this paper, is very closely related to the notion of *subtyping* in programming languages, except that here we are comparing behaviors rather than sets of related values. Subtyping relations for session types have already been studied [Gay and Hole, 2005], but the particular framework that we are setting up here gives us the interesting opportunity to re-discover these relations semantically: if session types are processes, we can reason about them by applying and adapting well-known behavioral theories for processes.

The reader will have noticed that we have been extremely liberal in the interpretation of the word “session” and in the association of channels with the corresponding session types in the examples above. Standard session type theories define the session as a private context which is instantiated and used by means of dedicated linguistic constructs. Here instead we work with just processes and channels. For us, each channel, no matter if private or public, identifies a session, and the session type describes the actions performed on that channel and the order in which they are performed.

The most fundamental difference between our approach and standard session type theories is that we permit non-linear usage of private and public channels. We have already seen that process $Buyer_1$ above keeps using private channel c after it has been sent to the seller. Basically this is possible because the parallel composition operator $\mid$ over session types allows us to express arbitrarily complex compositions of simultaneous behaviors over the same channel. Nonetheless, the non-linear usage of channels may interfere badly with the idea of projected behavior. To illustrate the issues that may arise consider the process

$$P = a?(x).x?(y : \mathtt{int}).a!(y+1)$$

which receives some channel x on a on which it waits for an integer number y, and then sends $y+1$ on a before terminating. It is reasonable to expect that the continuation of P after the first action is projected in the following way:

$$\vdash x?(y : \mathtt{int}).a!(y+1) : \{x : ?\mathtt{int}.\mathbf{1}, a : !\mathtt{int}.\mathbf{1}\} \tag{1}$$

Now consider the reduction

$$a!a \mid P \to \mathsf{nil} \mid a?(y : \mathtt{int}).a!(y+1)$$

where process P synchronizes with the process $a!a$ and reduces to $a?(y : \mathtt{int}).a!(y+1)$ which

is obtained by instantiating the variable x in the continuation of P after $a?(x)$ with the actual channel a that is sent. Thus, in this particular reduction, the variable x before the instantiation is an *alias* for a. The residual of P after the synchronization would be projected like this:

$$\vdash a?(y:\texttt{int}).a!(y+1) : \{a : ?\texttt{int}.!\texttt{int}.\mathbf{1}\} \tag{2}$$

The projection (2) has little to do with the one we obtained in (1) and it is hard to imagine how to formalize the relation between the two. In general, our projection assumes that each channel variable is instantiated with a channel name that is different from any other channel already present in the projection. It may be argued that this example is somewhat contrived, since we are sending channel a over itself, but the problem is more general. For instance, consider the process

$$Q = a?(x).b?(y).x!1.y!2$$

and the following reduction

$$a!c \mid b!c \mid Q \to \mathsf{nil} \mid b!c \mid b?(y).c!1.y!2 \to \mathsf{nil} \mid \mathsf{nil} \mid c!1.c!2$$

where the channel variables x and y, which are distinct in Q, are both aliases for c.

In an earlier version of this paper [Padovani, 2009] the aliasing problem is avoided altogether by means of a draconian restriction on the projection of processes guarded by channel input prefix: there a process of the form $a?(x).P$ is well typed if P has no free name other than x. Imposing that P can only use the channel it has received obviously limits the applicability of our projection idea. In this paper we relax this condition at the cost of a slight complication of the typing rules. The idea is to devise a type system that prevents well-typed processes from receiving channels they already own. We enforce this property by means of a *strict channel order* $\prec$ such that $v \prec u$ means that channel v can be sent/received on u and by imposing, in a process like $a?(x).P$, that all the free channel names and channel variables in P other than x and a are strictly larger than a according to $\prec$. The process P defined above does not violate this constraint by itself, since the continuation after the $a?(x)$ prefix uses no channel other than x and a. However, the sender process $a!a$ is ill-typed since, by strictness of $\prec$, the relation $a \prec a$ does *not* hold: a channel cannot be sent over itself. The process Q, on the other hand, is ill-typed since the residual $x!1.y!2$ requires the relations $x \prec a \prec b \prec x$, which form a cycle: the relation $x \prec a$ arises since x is received from a; the relation $a \prec b$ arises since b occurs free in the continuation after the input action $a?(x)$; similarly, the relation $b \prec x$ arises since x occurs free in the continuation after $b?(y)$. In summary, we allow non-linear usage of channels in the sense that the same channel can be simultaneously used by many processes at the same time. However, a well-typed process is prevented from receiving a channel it already owns, for this could change its projection in an unpredictable way.

We can identify three main contributions of our approach: viewing session types as projections of process behaviors allows us to define session types as an algebraic language of processes that closely resembles value-passing CCS. In particular, alternative behaviors can be composed by means of internal and external choices and the simultaneous access to a channel by two or more processes is modeled by the parallel composition of session types. As an immediate consequence of this generalization, we show that session types can be studied semantically, rather than syntactically, using and possibly adapting well-known behavioral equivalences. Thus, we are able to semantically justify the fundamental concepts (duality, well-typedness, the subtyping

Table 1. *Syntax of session types.*

σ	$::=$		**session type**	α	$::=$		**action**
		$\mathbf{0}$	(error)			$?t$	(value input)
	$\mid$	$\mathbf{1}$	(success)		$\mid$	$!t$	(value output)
	$\mid$	$\alpha.\sigma$	(action)		$\mid$	$?\sigma$	(channel input)
	$\mid$	$\sigma + \sigma$	(external choice)		$\mid$	$!\sigma$	(channel output)
	$\mid$	$\sigma \oplus \sigma$	(internal choice)				
	$\mid$	$\sigma \mid \sigma$	(composition)				

relation) that are axiomatically or syntactically presented in other theories. In the end, we provide a unified framework of behavioral types that encompasses features not only of dyadic and multiparty session types, but also of channel types [Pierce and Sangiorgi, 1996] and of conversation types [Caires and Vieira, 2009].

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 pre-order – to reason about safe replacement of channels and well-behaving systems. We devote Section 3 to the study of the main properties of the subsession relation. In Section 4 we formally define a process language without any explicit construct dedicated to session-oriented interaction and show how to project processes in this language into session types by means of a type system. The section illustrates the idea of behavior projection and the features of the type system with several examples and concludes with the main properties (type preservation and safety) of the projection. Because of its relative complexity, we prefer developing the theory of session types before applying it to a process language, to stress the focus of this paper on the nature of session types rather than that of processes. In fact, the process language presented in Section 4 can be seen as just a case study for the developed theory, which may be applied to different languages as well. Nonetheless, the reader who is eagerly looking for the details of the projection may safely jump to Section 4 after reading Section 2. Section 5 discusses related work and Section 6 concludes. For the sake of readability, proofs and other supplementary material related to Sections 3 and 4 have been moved into Appendix A and B respectively.

2. Syntax and Semantics of Session Types

2.1. *Syntax*

Let us get started by fixing some conventions: we assume an unspecified set $\mathscr{V}$ of *basic values*, ranged over by $\mathtt{v}$, ... such as integer and real numbers, boolean values, and so on; *basic types*, ranged over by t, s, ..., are arbitrary subsets of $\mathscr{V}$ like $\emptyset$ (the empty type), $\mathtt{v}$ (the singleton type inhabited by $\mathtt{v}$ only), `int`, `real`, `bool`, and so forth; we write $\neg t$ for $\mathscr{V} \setminus t$. We interpret types as sets of values and say that t is a *subtype* of s when $t \subseteq s$. This setting may be generalized to

more expressive types as explained in [Castagna and Frisch, 2005; Frisch *et al.*, 2008]. We will sometimes say that v is of type t if $\mathsf{v} \in t$.

The syntax of session types is presented in Table 1: session types, ranged over by $\sigma, \tau, \ldots$, are projections of process behaviors with respect to a fixed channel. The session type $\mathbf{0}$ describes a faulty process which was involved in a communication error over the channel. No correct system should ever contain channels typed by $\mathbf{0}$, but having an explicit representation of faulty behaviors is useful in the theory that follows. In a sense, the existence of $\mathbf{0}$ witnesses that session types are behaviors and that it is perfectly feasible (although generally undesirable) to write processes that misbehave. The session type $\mathbf{1}$ describes a process that has finished using a channel. The session type $\alpha.\sigma$ describes a process that offers an action determined by the prefix α, and after the action is performed it behaves according to the residual session type σ. *Action prefixes*, ranged over by $\alpha, \ldots$, represent input/output operations on a channel: $?t$ and $!t$ represent respectively the input and the output of values of type t; $?\sigma$ and $!\sigma$ are similar but they concern input/output of channels of type σ. Session types can be composed by means of two behavioral choices, the *external choice* $+$ and the *internal choice* $\oplus$. The session type $\sigma + \tau$ describes a process that offers interacting processes two possible behaviors σ and τ. Interacting processes choose by offering complementary actions with respect to those offered in σ and τ. Dually, the session type $\sigma \oplus \tau$ describes a process that internally decides to behave according to either σ or τ. Interacting processes have no way of affecting this choice and must be prepared to handle both behaviors σ and τ. Finally, the session type $\sigma \mid \tau$ describes the simultaneous access to a channel by two processes, each of which behaves, with respect to that channel, according to the session types σ and τ.

We do not rely on any explicit syntax for describing recursive session types (hence potentially infinite behaviors). As in [Castagna *et al.*, 2009a], we define session types as the possibly infinite syntax trees generated by the productions for σ in Table 1 that satisfy the following conditions:

— every infinite branch of the tree has infinite occurrences of the action prefix operator;
— every tree has a finite number of different sub-trees.

The first is a *contractivity condition* to rule out trees such as those that are solutions of the equations $X = X + X$ or $X = X \oplus X$ which are not meaningful in the theory we are about to develop; the second is a *regularity condition* ensuring that the trees are *regular trees* [Courcelle, 1983]. In addition, we will focus on well-formed session types where every parallel composition occurring in some branch of an external choice must be guarded by a prefix. This condition arises naturally in practice (see Section 4) and spares us some annoying technicalities in the proofs.

Example 2.1. To familiarize with recursive session types consider the family of infinite behaviors that are uniquely determined by the following equations (uniqueness is ensured by the contractivity condition above [Courcelle, 1983]):

$$!t^\omega = !t.!t^\omega \qquad ?t^\omega = ?t.?t^\omega + ?\neg t.\mathbf{0} \qquad t^\omega = (?t.t^\omega + ?\neg t.\mathbf{0}) \oplus !t.t^\omega$$
$$!\rho^\omega = !\rho.!\rho^\omega \qquad ?\rho^\omega = ?\rho.?\rho^\omega \qquad \rho^\omega = ?\rho.\rho^\omega \oplus !\rho.\rho^\omega$$

The session type $!t^\omega$ describes the behavior of a process that sends an infinite number of values of type t. The session type $!\rho^\omega$ is similar, but in this case the process sends channels with type ρ rather than basic values. The session type $?t^\omega$ describes the behavior of a process that is capable of receiving an infinite number of values of type t and that fails as soon as it receives any value

that is not of type t. Again the session type $?\rho^\omega$ is similar, but regards the input capability of the process. The session type t^ω is a composition of $?t^\omega$ and $!t^\omega$ where, at each moment, the process internally decides whether to send a value of type t or to wait for a value of type t. Similarly, ρ^ω is a composition of $?\rho^\omega$ and $!\rho^\omega$.

It is instructive to compare the above behaviors with the following ones:

$$!t^* = \mathbf{1} \oplus !t.!t^* \qquad ?t^* = \mathbf{1} \oplus (?t.?t^* + ?\neg t.\mathbf{0}) \qquad t^* = \mathbf{1} \oplus (?t.t^* + ?\neg t.\mathbf{0}) \oplus !t.t^*$$

For example, the session type $!t^*$ describes a process that sends an arbitrary number of values of type t on some channel, but it may internally decide to quit using the channel at any time. So, $!t^\omega$ is a "more deterministic" behavior than $!t^*$ and it gives more guarantees to interacting processes. The precise sense in which $!t^\omega$ and $!t^*$ (as well as the other pairs of corresponding behaviors) are related will be formalized in Section 2.2. ■

It may appear 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 theories. There are three reasons for this generality: first, it allows us to express the typing rules (Section 4) in a compositional way, which is particularly important in our approach where we aim at capturing full, unconstrained process behaviors; second, it separates actions from choices, thus yielding a clean, algebraic type language with orthogonal features; third, it allows us to express (some) parallel compositions in terms of equivalent, sequential session types. For example, consider the session type $\sigma = !\texttt{int}.\mathbf{1} \,|\, !\texttt{bool}.\mathbf{1}$ which 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 simultaneously available. In other words, the composition σ is equivalent to the session type $!\texttt{int}.!\texttt{bool}.\mathbf{1} + !\texttt{bool}.!\texttt{int}.\mathbf{1}$, that is the interleaving of the actions in σ. Had we expanded σ to $!\texttt{int}.!\texttt{bool}.\mathbf{1} \oplus !\texttt{bool}.!\texttt{int}.\mathbf{1}$ instead, no interacting process would be able to decide which value, the integer or the boolean value, to read first. This ability to expand parallel compositions in terms of sequential choices is well studied in process algebra communities where it goes under the name of *expansion law* [De Nicola and Hennessy, 1987; Hennessy, 1988].

2.2. *Semantics*

According to the idea that session types are behaviors, the most natural way for giving some meaning to session types is to equip them with a transition relation that describes the actions performed by processes behaving according to these types. The 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 *action labels* ranged over by μ, ... and generated by the grammar:

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

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*. Thus, the language of session types and the transition system extend CCS without τ's [De Nicola and Hennessy,

Table 2. *Transitions of session types.*

$$\text{(R1)}\ \mathbf{1} \xrightarrow{\checkmark} \mathbf{1} \qquad \text{(R2)}\ \sigma \oplus \tau \longrightarrow \sigma \qquad \text{(R3)}\ !\mathsf{v}.\sigma \xrightarrow{!\mathsf{v}} \sigma \qquad \text{(R4)}\ \frac{\mathsf{v} \in t}{!t.\sigma \longrightarrow !\mathsf{v}.t} \qquad \text{(R5)}\ \frac{\mathsf{v} \in t}{?t.\sigma \xrightarrow{?\mathsf{v}} \sigma} \qquad \text{(R6)}\ !\rho.\sigma \xrightarrow{!\rho} \sigma$$

$$\text{(R7)}\ \frac{\rho \preceq \rho'}{?\rho'.\sigma \xrightarrow{?\rho} \sigma} \qquad \text{(R8)}\ \frac{\rho \not\preceq \rho'}{?\rho'.\sigma \xrightarrow{?\rho} \mathbf{0}} \qquad \text{(R9)}\ \frac{\sigma \longrightarrow \sigma'}{\sigma + \tau \longrightarrow \sigma' + \tau} \qquad \text{(R10)}\ \frac{\sigma \xrightarrow{\mu} \sigma'}{\sigma + \tau \xrightarrow{\mu} \sigma'} \qquad \text{(R11)}\ \frac{\sigma \longrightarrow \sigma'}{\sigma \mid \tau \longrightarrow \sigma' \mid \tau}$$

$$\text{(R12)}\ \frac{\sigma \xrightarrow{\mu} \sigma' \qquad \mu \neq \checkmark}{\sigma \mid \tau \xrightarrow{\mu} \sigma' \mid \tau} \qquad \text{(R13)}\ \frac{\sigma \xrightarrow{\mu} \sigma' \qquad \tau \xrightarrow{\overline{\mu}} \tau' \qquad \mu \neq \checkmark}{\sigma \mid \tau \longrightarrow \sigma' \mid \tau'} \qquad \text{(R14)}\ \frac{\sigma \xrightarrow{\checkmark} \sigma' \qquad \tau \xrightarrow{\checkmark} \tau'}{\sigma \mid \tau \xrightarrow{\checkmark} \sigma' \mid \tau'}$$

1987] to a value-passing calculus. Let us briefly comment the rules in Table 2. Rule (R1) states that the session type **1** emits a signal ✓ denoting successful termination and reduces to itself. By rule (R2), the session type $\sigma \oplus \tau$ can perform an internal transition to either σ or τ. Rules (R3), (R4), and (R5) deal with actions on basic values: the session type $!\mathsf{v}.\sigma$ performs the action $!\mathsf{v}$ denoting the emission of value v and reduces to σ; the session type $!t.\sigma$ may internally reduce to any $!\mathsf{v}.\sigma$ where v is of type t; the session type $?t.\sigma$ may perform any action of the form $?\mathsf{v}$ for any v of type t then reducing to σ. 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. Rules (R6), (R7), and (R8) deal with actions on channels: the session type $!\rho.\sigma$ performs an action $!\rho$ (the output of a channel of type ρ); rules (R7) and (R8) state that a session type $?\rho'.\sigma$ performs actions of the form $?\rho$ for any ρ. However, $?\rho.\sigma$ reduces to σ only if the type ρ of the received channel is a *subsession* of the expected type ($\rho \preceq \rho'$); if the type ρ of the received channel is *not* a subsession of the expected type ($\rho \not\preceq \rho'$), then an unrecoverable error occurs. This is signalled by the fact that the residual behavior is **0**. We will define the subsession relation between session types shortly. For the time being, the reader can comfort herself with the intuition that subsession is the behavioral equivalent of subtyping: if $\rho \preceq \rho'$ holds, then any channel with associated session type ρ may be safely used where a channel with associated session type ρ' is expected. Rule (R9) states that $+$ is indeed an external choice, thus internal moves 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. Rule (R10) states that external choices offer any action that is offered by either branch. Rules (R11), (R12), and (R13) express the evolution of compositions: each component of a composition is able to make autonomous progress by itself (R11); each action offered by a component is also offered by the whole composition (R12); two components may synchronize if they offer complementary actions. In this rule and in the following, $\overline{\mu}$ denotes the complement of action μ, for example $\overline{?\mathsf{v}} = !\mathsf{v}$ and $\overline{?\rho} = !\rho$; we assume that $\overline{\checkmark}$ is undefined. Finally, rule (R14) expresses the fact that a parallel composition is successfully terminated only if both its components are.

Before we move on to defining the subsession relation, we should discuss a fundamental design

decision that regards communication and external choices. On the one hand, rule (R5) shows that only values of the right type may be input by a process whose behavior is described by the session type $?t.\sigma$. As a consequence, values may drive the selection of the branch in external choices. For example, we have $?\mathtt{int}.\sigma + ?\mathtt{bool}.\tau \xrightarrow{?3} \sigma$ and $?\mathtt{int}.\sigma + ?\mathtt{bool}.\tau \xrightarrow{?\mathtt{true}} \tau$. The *type* of the value in the label uniquely determines the branch and thus the residual behavior of the process. This feature subsumes the label-driven branch selection that is found in standard session types theories. On the other hand, rules (R7) and (R8) show that a process waiting for a channel will input the channel regardless of its associated session type. If the channel that is received has the "right type" (a session type that is a subsession of the expected one), then the communication is successful and the process proceeds normally. If the channel that is read has the "wrong type" (a session type that is not a subsession of the expected one), then the communication yields an error. As a consequence, 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 then, assuming that ρ and ρ' are not related by the subsession relation, we also have the transitions $?\rho.\sigma + ?\rho'.\tau \xrightarrow{?\rho'} \mathbf{0}$ and $?\rho.\sigma + ?\rho'.\tau \xrightarrow{?\rho} \mathbf{0}$. Namely, a process interacting with another one whose behavior is described by $?\rho.\sigma + ?\rho'.\tau$ may safely send a channel of type ρ'' only if ρ'' is a subsession of both ρ and ρ'. In this case, the residual behavior is non-deterministically chosen to be either σ or τ. In summary, unlike in [Castagna *et al.*, 2009a] we do not allow dynamic dispatching according to the type of channels. Section 5 contains a more detailed discussion about this design decision and its consequences.

In the following we adopt standard conventions regarding the transition relations: we write $\Longrightarrow$ for the reflexive, transitive closure of $\longrightarrow$; let $\overset{\mu}{\Longrightarrow}$ be $\Longrightarrow\xrightarrow{\mu}\Longrightarrow$; we write $\sigma \xrightarrow{\mu}$ (respectively, $\sigma \overset{\mu}{\Longrightarrow}$) if there exists τ such that $\sigma \xrightarrow{\mu} \tau$ (respectively, $\sigma \overset{\mu}{\Longrightarrow} \tau$); we write $\not\longrightarrow$, $\xrightarrow{\mu}\!\!\!\!\!\!/\;$, $\overset{\mu}{\Longrightarrow}\!\!\!\!\!\!/\;$ for the usual negated relations; for example, $\sigma \not\longrightarrow$ means that σ does not perform internal transitions. We let $\mathsf{init}(\sigma) \stackrel{\text{def}}{=} \{\mu \mid \sigma \overset{\mu}{\Longrightarrow}\}$; if A is a set of actions, we write $\overline{\mathrm{A}}$ for the set of corresponding co-actions, namely $\overline{\mathrm{A}} \stackrel{\text{def}}{=} \{\overline{\mu} \mid \mu \in \mathrm{A} \setminus \{\checkmark\}\}$.

We have assumed that the subtyping relation $t \subseteq s$ for plain types is semantically defined as the inclusion between the sets of values that inhabit t and s. It would be nice if a similar approach could be adopted also for session types but, since session types are behaviors, it is not clear which values do inhabit them. It could be argued that session types are inhabited by channels, which are indeed values. However, this interpretation falls short of providing a suitable semantics of session types because channels, unlike basic values (such as, say, numbers or XML documents), are just names and, as such, they have no structure. There is nothing in a channel name that may somehow constraint or characterize the *behavior* of processes using that channel. Consequently, from a purely type-theoretic point of view we consider all channels as having one special type (in Section 3 we will actually introduce this type $\diamond$ for technical reasons). This provides further motivation for preventing dynamic dispatching depending on the "type" of channels, but it leaves open the issue of defining a suitable subsession relation for session types. If we insist on viewing session types as behaviors rather than actual types, we can adopt a *testing* approach [De Nicola and Hennessy, 1984], in this way: first we define some way of "testing" a session type, that we call completeness; then we declare two session types as being equivalent if they pass the same tests; in fact, we will say that σ is a subsession of τ if any test that σ passes is also passed by τ.

As we have informally stated in the introduction, *completeness* of a session type σ means that σ describes the behavior of some processes successfully interacting with each other. By "successfully" we mean that no process is ever left behind waiting for messages that never arrive, or trying to send messages that no one is willing to receive. Also, no communication error resulting from rule (R8) may ever occur in a complete session type. From this intuition it is clear the need to single out the components of σ and verify that either they emit $\checkmark$, indicating successful termination, or they can synchronize with some other component.

Definition 2.1 (liveness). We say that a session type σ is a *component* if every parallel composition in σ occurs underneath a prefix. Let $\cong$ be the least congruence that satisfies the laws:

$$\sigma \cong \mathbf{1} \mid \sigma \qquad \sigma \mid \tau \cong \tau \mid \sigma \qquad \sigma \mid (\tau \mid \rho) \cong (\sigma \mid \tau) \mid \rho$$

We say that σ is *live* if, for every σ_1 and σ_2, $\sigma \cong \sigma_1 \mid \sigma_2$ where σ_1 is a component implies either $\checkmark \in \mathsf{init}(\sigma_1)$ or $\overline{\mathsf{init}(\sigma_1)} \cap \mathsf{init}(\sigma_2) \neq \emptyset$.

We now formalize completeness as the preservation of liveness under internal transitions.

Definition 2.2 (completeness). We say that σ is *complete* if $\sigma \Longrightarrow \tau$ implies τ live.

Completeness implies that no communication error can occur (rule (R8) is never applied) for otherwise some component reduces to $\mathbf{0}$ and $\mathbf{0} \mid \sigma$ is *not* live regardless of σ. Therefore, because of the parallel composition operator for session types, our notion of completeness generalizes that of duality in other session type theories, and in particular that of Castagna *et al.* [2009a] which adopts a language of session types very close to the one used here, except for the lack of $\mid$.

Example 2.2. In the introduction we have argued that a session type such as $!\mathtt{int}.\mathbf{1} \mid ?\mathtt{int}.\mathbf{1}$ is complete because the actions offered by the two components are complementary. However, the operational semantics of session types works at the finer level of values, rather than at the level of actions. So, to be more precise, $!\mathtt{int}.\mathbf{1} \mid ?\mathtt{int}.\mathbf{1}$ is complete because any $!\mathsf{v}$ performed by $!\mathtt{int}.\mathbf{1}$ has a complementary one $?\mathsf{v}$ performed by $?\mathtt{int}.\mathbf{1}$. This makes the definition of completeness more sophisticated and the resulting theory more general, because we can deal with (partially) overlapping types. For example, assuming that $\mathtt{int} \subseteq \mathtt{real}$ we have that $!\mathtt{int}.\mathbf{1} \mid ?\mathtt{real}.\mathbf{1}$ is also complete, despite the two actions $!\mathtt{int}$ and $?\mathtt{real}$ are not exactly complementary. Similarly, $!\mathtt{real}.\mathbf{1} \mid (?\mathtt{int}.\mathbf{1} + ?(\mathtt{real} \setminus \mathtt{int}).\mathbf{1})$ is complete as well, even though in $?\mathtt{int}.\mathbf{1} + ?(\mathtt{real} \setminus \mathtt{int}).\mathbf{1}$ there is no single prefix that can accept all the actions $!\mathsf{v}$ that can be performed by $!\mathtt{real}.\mathbf{1}$. ■

We use completeness as the discriminating test for comparing session types extensionally and say that σ is a subsession of τ if every session type that completes σ completes τ as well.

Definition 2.3 (subsession). Let $[\![\sigma]\!] \stackrel{\text{def}}{=} \{\rho \mid (\rho \mid \sigma) \text{ complete}\}$. We say that σ is a *subsession* of τ, notation $\sigma \preceq \tau$, if $[\![\sigma]\!] \subseteq [\![\tau]\!]$. We write $\approx$ for the equivalence relation induced by $\preceq$, namely $\approx = \preceq \cap \preceq^{-1}$.

The fact that subsession is a sensible choice when interpreted as a subtyping relation may not be entirely obvious because, according to Definition 2.3, the subsession relation speaks about a

left-to-right substitutability: if $\sigma \preceq \tau$ holds, then it is safe to use a process that behaves according to τ in place of a process that behaves according to σ. That is, subsession is a behavioral relation which, in accordance with other standard behavioral pre-orders such as the *must* testing relation [De Nicola and Hennessy, 1984], has been defined in terms of the contexts ρ in which some behavior σ can be safely replaced by another behavior τ. On the other hand, the relation $\sigma \preceq \tau$ that we informally anticipated while describing the transition system of session types and according to the usual intuition behind every subtyping relation, speaks about a *right-to-left* substitutability: it is safe to use a channel "with associated type" σ where a channel "with associated type" τ is expected. The quotation marks remind us that σ and τ are behaviors while channels, by themselves, do not expose any behavior. It is the processes using those channels that behave according to σ or τ. This is the key for realizing that subsession does really make sense when interpreted as a subtyping relation: when a process P acts on some channel $c : \tau$ we can think of τ as the projection of P's behavior on c, while the context ρ is given by the combined behavior of all the other processes acting on c. By replacing c with $d : \sigma$ we are changing the context with which P interacts and the relation $\sigma \preceq \tau$ assures that any context that completes σ completes τ as well, hence the substitution is safe in that it preserves completeness.

The equational theory induced by the subsession relation is not obvious, although a few relations are easy to check: for example, $|$ is a commutative, associative operator with neutral element $\mathbf{1}$. According to the subsession relation, $+$ and $\oplus$ are also commutative and associative and $\mathbf{0}$ is neutral for $+$. Furthermore $\sigma \oplus \tau \preceq \sigma$, namely $\preceq$ embeds reduction of nondeterminism as a fundamental law, in the same spirit of the *must* testing relation for standard process algebras [De Nicola and Hennessy, 1984].

Example 2.3. The session types defined in Example 2.1 resemble classic channel types in typed theories of the π-calculus. In fact we can spot even more similarities when we try to relate them using the subsession relation. To begin with, it is possible to show that $?\cdot^*$ is covariant, $!\cdot^*$ is contravariant, and $\cdot^*$ is invariant in the respective arguments. More formally, the relations

$$t \subseteq s \iff ?t^* \preceq ?s^* \qquad s \subseteq t \iff !t^* \preceq !s^* \qquad t = s \iff t^* \approx s^*$$

do hold (to be precise the middle statement holds only when $s \neq \emptyset$, see Example 3.3).

Furthermore, it is safe to use a channel with "more capabilities" where a channel with "less capabilities" is expected, hence

$$t^* \preceq ?t^* \preceq ?t^\omega \qquad \text{and} \qquad t^* \preceq !t^* \preceq !t^\omega \qquad \text{and} \qquad t^* \preceq t^\omega$$

since it is safe to use a channel with both input and output capabilities where only one of them is needed. Also, it is safe to use a channel with associated type $\cdot^*$ where a channel with associated type $\cdot^\omega$ is expected. In this case, the process will never exercise the capability to stop using the channel, even though this is allowed by the session type associated with the actual channel it is using.

Finally, an interesting interplay arises between the internal choice operator and input/output actions:

$$?t^* \oplus ?s^* \approx ?(t \cap s)^* \qquad \text{and} \qquad !(t \cup s)^* \preceq !t^* \oplus !s^*$$

The first relation has been discovered previously [Castagna *et al.*, 2008; Carpineti *et al.*, 2009] in completely different settings. In general the equivalence $!(t \cup s)^* \approx !t^* \oplus !s^*$ does not hold

because the first value sent by a process that behaves according to $!t^* \oplus !s^*$ may give information about the type of the values in the subsequent communications. For example, consider $\sigma = \mathbf{1} + ?\mathtt{int}.\sigma$ and $\tau = \mathbf{1} + ?\mathtt{bool}.\tau$. Then $\mathbf{1} + ?\mathtt{int}.\sigma + ?\mathtt{bool}.\tau$ completes $!\mathtt{int}^* \oplus !\mathtt{bool}^*$ but not $!(\mathtt{int} \cup \mathtt{bool})^*$. ■

Other relations are those concerning errors: we have $\mathbf{0} \approx \alpha.\mathbf{0}$ and $!\emptyset.\sigma \approx ?\emptyset.\sigma \approx \mathbf{0}$. More generally, the relation $\sigma \approx \mathbf{0}$ means that σ cannot be completed in any way: σ describes an intrinsically flawed behavior that may reach a state in which it is not live regardless of its context. 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 2.4 (viability). We say that σ is *viable* if $[\![\sigma]\!] \neq \emptyset$.

Example 2.4. Observe that $?t.\sigma \preceq ?s.\sigma$ when $t \subseteq s$ does *not* hold in general. For example, consider $?\mathtt{int}.\mathbf{1}$ and $?\mathtt{real}.\mathbf{1}$. Then $!\mathtt{int}.\mathbf{1} + !\sqrt{2}.\mathbf{0}$ completes $?\mathtt{int}.\mathbf{1}$ because $?\mathtt{int}$ does not perform $?\sqrt{2}$. However, $!\mathtt{int}.\mathbf{1} + !\sqrt{2}.\mathbf{0}$ does not complete $?\mathtt{real}.\mathbf{1}$ because of the derivation $!\mathtt{int}.\mathbf{1} + !\sqrt{2}.\mathbf{0} \mid ?\mathtt{real}.\mathbf{0} \longrightarrow \mathbf{0} \mid \mathbf{1}$. In general, if $\sigma \preceq \tau$ holds, then τ should not expose any action that was not exposed by σ to avoid interferences.

It is safe to equip τ with more possible behaviors only if these are guarded by actions for which σ is known to have a non-viable continuation. For instance, according to Example 2.1 we have defined $?t^\omega = ?t.?t^\omega + ?\neg t.\mathbf{0}$ and it is possible to prove that $t \subseteq s$ implies $?t^\omega \preceq ?s^\omega$. Now we realize that the $\neg t.\mathbf{0}$ branch plays a fundamental role in this respect. Indeed, no process interacting with another one behaving according to $?t^\omega$ will ever send any value that is of type $s \setminus t$, since this will lead to a communication error. Thus it is safe to use a channel of type $?t^\omega$ where one of type $?s^\omega$ is expected, since the additional behaviors exposed by $?s^\omega$ are shielded by non-viable behaviors in $?t^\omega$. ■

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: the subsession relation between session types with weight 0 is well defined, since rules (R7) and (R8) are never used; the subsession relation between session types with weight $n+1$ is also well defined, since the premises of rules (R7) and (R8) necessarily regard session types whose weight is at most n.

This stratification induces a hierarchy on channels: those whose associated session type has weight 0 are *first-order* channels, in the sense that they cannot be used for sending other channels; channels whose associated session type has weight 1 can only be used for sending first-order channels; and so forth. Not only this hierarchy seems natural in practice, but it plays a fundamental role also in the projection of processes that we will see in detail in Section 4. ■

3. Properties of the Subsession Relation

The purpose of this section is to deepen our understanding of the subsession relation and to pinpoint its main properties. While the best way to achieve this would be to present an axiomati-

zation for $\preceq$ we must face the fact that $\preceq$ is closely related to the fair testing relation [Natarajan and Cleaveland, 1995; Rensink and Vogler, 2007] which is notoriously difficult, if at all possible, to axiomatize.[†] To further complicate matters, in our setting parallel composition is a truly primitive operator (see Definition 2.2). For example the session types

$$\sigma = ?\mathtt{int}^\omega \,|\, !\mathtt{int}^\omega \qquad \text{and} \qquad \tau = ?\mathtt{int}.(?\mathtt{int}^\omega \,|\, !\mathtt{int}^\omega) + !\mathtt{int}.(?\mathtt{int}^\omega \,|\, !\mathtt{int}^\omega)$$

are observationally indistinguishable (they are weakly bisimilar) and yet they describe different scenarios: σ describes the behavior of *two* processes communicating integer numbers; τ describes the behavior of *one* process that is waiting for either sending or receiving an integer number, and thereafter it forks two processes communicating integer numbers. In particular, σ is complete, while τ is not.

For these reasons we proceed as follows: in Section 3.1 we provide a *local characterization* of the subsession relation restricted to components. We are able to extend these results to the general setting thanks to the precongruence properties of $\preceq$ which we study in Section 3.2. Finally, we sketch out the role of parallel composition and the fairness properties of $\preceq$ by means of examples in Section 3.3.

3.1. *Local Characterization of the Subsession Relation*

The local characterization of the subsession relation requires some auxiliary notation. We begin by defining *ground actions* as approximated actions where we abstract from the type of communicated channels:

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

With an abuse of notation we use μ for ranging over ground and non-ground actions without distinction and will make sure that no confusion may possibly arise. Given an action μ, we write $\lfloor\mu\rfloor$ for the ground action corresponding to μ: $\lfloor\cdot\rfloor$ is the identity except that $\lfloor\dagger\rho\rfloor = \dagger\diamond$ where $\dagger \in \{?, !\}$. We extend $\lfloor\cdot\rfloor$ to sets of actions so that $\lfloor \mathrm{A} \rfloor = \{\lfloor\mu\rfloor \mid \mu \in \mathrm{A}\}$.

Then, we define the *continuation* of a session type σ with respect to an action μ as the combination of all the possible residuals of σ after μ. This differs from the relation $\xrightarrow{\mu}$ which relates σ with *one particular* (not necessarily unique) residual of σ after μ. For example, consider $\sigma = ?\mathtt{int}.\sigma_1 + ?\mathtt{real}.\sigma_2$. On the one hand we have $\sigma \xrightarrow{?3} \sigma_1$ and also $\sigma \xrightarrow{?3} \sigma_2$ namely, there are two possibly different residuals of σ after ?3 due to two different branches of the external choice that may yield common actions. On the other hand, the (unique) continuation of σ after ?3 is $\sigma_1 \oplus \sigma_2$, which expresses the fact that both branches are possible. One simple way to think of the continuation of σ after μ is as the behavior perceived by the process(es) at the other end of the communication channel: the process sending !3 (the dual action of ?3) to $?\mathtt{int}.\sigma_1 + ?\mathtt{real}.\sigma_2$ does *not* know which branch (σ_1 or σ_2) has been taken, hence from its point of view it is as if the receiver behaves according to $\sigma_1 \oplus \sigma_2$. The actual definition of continuation is slightly more involved because we must abstract from the type of channels being communicated and we need to take into account the possibility of communication errors:

[†] As far as the author's knowledge goes, the definition of complete axiomatizations of fair testing equivalences is still an open problem [Rensink and Vogler, 2007].

Definition 3.1 (continuation). Let $\sigma \stackrel{\lfloor\mu\rfloor}{\Longrightarrow}$. The *continuation* of σ with respect to μ is defined as follows:

$$\sigma(\mu) \stackrel{\text{def}}{=} \begin{cases} \bigoplus_{\sigma \Longrightarrow \stackrel{\mu}{\longrightarrow} \sigma'} \sigma' & \text{if } \lfloor\mu\rfloor \neq !\diamond \\ \bigoplus_{\sigma \Longrightarrow \stackrel{!\rho'}{\longrightarrow} \sigma'} \sigma' & \text{if } \mu = !\rho \text{ and } \sigma \stackrel{!\rho'}{\Longrightarrow} \text{ implies } \rho' \preceq \rho \\ \mathbf{0} & \text{otherwise} \end{cases}$$

As long as $\mu = \lfloor\mu\rfloor$ the continuation $\sigma(\mu)$ is the internal choice of all possible residuals of σ after μ. This is also true when $\mu = ?\rho$ for some ρ, but recall that session types of the form $?\rho'.\sigma$ offer any action of the form $?\rho$. So for example, if $\sigma = ?\rho_1.\sigma_1 + ?\rho_2.\sigma_2$ we may have $\sigma(?\rho) = \sigma_1 \oplus \sigma_2$ or $\sigma(?\rho) = \mathbf{0} \oplus \sigma_2$ or $\sigma(?\rho) = \sigma_1 \oplus \mathbf{0}$ or $\sigma(?\rho) = \mathbf{0} \oplus \mathbf{0}$ according to whether $\rho \preceq \sigma_1$ and $\rho \preceq \sigma_2$ or $\rho \not\preceq \sigma_1$ and $\rho \preceq \sigma_2$ or $\rho \preceq \sigma_1$ and $\rho \not\preceq \sigma_2$ or $\rho \not\preceq \sigma_1$ and $\rho \not\preceq \sigma_2$, respectively. When $\mu = !\rho$ things are a little more complicated. Consider a sender process that behaves according to the session type $\sigma = !\rho_1.\sigma_1 \oplus !\rho_2.\sigma_2$. The receiver process must offer actions of the form $?\rho'$ for the synchronization to occur, for instance we may assume that it behaves according to the type $?\rho.\sigma$. If $\rho_1 \preceq \rho$ and $\rho_2 \preceq \rho$, then, no matter which action ($!\rho_1$ or $!\rho_2$) is offered by the sender, no error occurs and $\sigma(!\rho) = \sigma_1 \oplus \sigma_2$. If, however, we have $\rho_1 \not\preceq \rho$, then the synchronization occurs nonetheless, but it yields a communication error. According to rule

$$\text{(R8)}\ \frac{\rho \not\preceq \rho'}{?\rho'.\sigma \stackrel{?\rho}{\longrightarrow} \mathbf{0}}$$

in Table 2, the error is registered in the receiver, which reduces to $\mathbf{0}$. Since here we only have the sender's session type at our disposal, it is in sender's continuation that we keep track of the error, hence we let $\sigma(!\rho) = \mathbf{0}$. In summary, the continuation after $!\rho$ is the combination of all the residuals after all the actions $!\rho'$, but only if ρ is an upper bound for all the ρ'. If ρ is not an upper bound for any possible ρ' that is offered, the continuation is defined to be $\mathbf{0}$. This implies that, if there is no upper bound for all the ρ' offered by the sender, then no synchronization may occur at all.

The final auxiliary notion we need, that of ready set, gives us a tool for measuring the degree of nondeterminism of a session type.

Definition 3.2 (ready set). We say that σ has *ready set* R, notation $\sigma \Downarrow \mathrm{R}$, if there exists σ' such that $\sigma \Longrightarrow \sigma'$ and $\mathrm{R} = \{\lfloor\mu\rfloor \in \lfloor\mathsf{init}(\sigma')\rfloor \mid \sigma(\mu) \text{ viable}\}$.

In words, σ has ready set R, where R is a possibly infinite set of actions, if σ may independently evolve to some residual σ' such that all the actions offered by σ' whose continuation with respect to σ is viable are in R. From now on we use R, S, ... to range over ready sets.

A few examples will clarify the concept:

— $\alpha.\mathbf{0} \Downarrow \emptyset$ since, regardless of any action that $\alpha.\mathbf{0}$ may possibly emit, the corresponding continuation is not viable;
— $?\mathtt{int}.\mathbf{1}$ has just one ready set $\{?\mathsf{v} \mid \mathsf{v} \in \mathtt{int}\}$ which contains all the actions of the form $?\mathsf{v}$ for every v of type $\mathtt{int}$;
— $!\mathtt{int}.\mathbf{1}$ has as many ready sets as the number of values of type $\mathtt{int}$, each set having the form $\{!\mathsf{v}\}$ where $\mathsf{v} \in \mathtt{int}$, plus another set $\{!\mathsf{v} \mid \mathsf{v} \in \mathtt{int}\}$ which is obtained by taking $\sigma' = \sigma$.

Table 3. *Selected axioms of the subsession relation.*

(S1)	$\sigma \oplus \tau \preceq \sigma$	(SP1)	$?(t \cup s).\sigma \approx ?t.\sigma + ?s.\sigma$	
(S2)	$\sigma \preceq \mathbf{1} + \sigma$	(SP2)	$!(t \cup s).\sigma \approx !t.\sigma \oplus !s.\sigma$	if $t, s \neq \emptyset$
(S3)	$\alpha.\mathbf{0} \preceq \mathbf{0}$			
		(CH1)	$\dagger\rho.\sigma + \dagger\rho'.\tau \approx \dagger\rho.\sigma \oplus \dagger\rho'.\tau$	
(D1)	$\alpha.\sigma + \alpha.\tau \approx \alpha.(\sigma \oplus \tau)$	(CH2)	$?(\rho \oplus \rho').(\sigma \oplus \tau) \approx ?\rho.\sigma \oplus ?\rho'.\tau$	
(D2)	$\alpha.\sigma \oplus \alpha.\tau \approx \alpha.(\sigma \oplus \tau)$	(CH3)	$!(\rho \vee \rho').(\sigma \oplus \tau) \approx !\rho.\sigma \oplus !\rho'.\tau$	

Intuitively, the larger the number of ready sets of a session type, the less deterministic the session type is. Compare $?\texttt{true}.\mathbf{1} + ?\texttt{false}.\mathbf{1}$ and $?\texttt{true}.\mathbf{1} \oplus ?\texttt{false}.\mathbf{1}$: the former has just one ready set $\{?\texttt{true}, ?\texttt{false}\}$ saying that the session type is ready to receive *any* boolean value; the latter has three ready sets, $\{?\texttt{true}\}$, $\{?\texttt{false}\}$, and $\{?\texttt{true}, ?\texttt{false}\}$, saying that the session type may be willing to receive only one of `true` or `false` or both, depending on its internal state. Only actions whose continuation with respect to σ is viable are considered in the ready set. For example $?\texttt{int}^\omega \Downarrow \{?\mathsf{v} \mid \mathsf{v} \in \texttt{int}\}$ despite $?\texttt{int}^\omega \xrightarrow{?\mathsf{v}}$ for every $\mathsf{v} \in \mathscr{V}$ since $?\texttt{int}^\omega(?\mathsf{v}) = \mathbf{0}$ for every $\mathsf{v} \in \mathscr{V} \setminus \texttt{int}$. Beware of ground actions $?\diamond$ and $!\diamond$: if $?\diamond$ is in some ready set of σ, then there exists ρ such that $\sigma(?\rho)$ is viable; if $!\diamond$ is in some ready set of σ, then there exists ρ such that $\sigma(!\rho)$ is viable and $\sigma \stackrel{!\rho'}{\Longrightarrow}$ implies $\rho' \preceq \rho$.

We are now ready to provide the local characterization of $\preceq$:

Theorem 3.1. Let σ and τ be components. Then $\sigma \preceq \tau$ if and only if:

1 $\tau \Downarrow \mathrm{S}$ implies $\sigma \Downarrow \mathrm{R}$ and $\mathrm{R} \subseteq \mathrm{S}$;
2 σ viable and $\tau \stackrel{\lfloor\mu\rfloor}{\Longrightarrow}$ implies $\sigma \stackrel{\lfloor\mu\rfloor}{\Longrightarrow}$ and $\sigma(\mu) \preceq \tau(\mu)$ for every $\mu \neq \checkmark$.

Property (1) states that for every ready set of τ there is one of σ with fewer actions. As we have seen, this intuitively means that τ is more deterministic than σ. Property (2) states that, when σ is viable, any observable action offered by the larger session type must also be offered by the smaller one and that the continuations of σ and τ with respect to such actions must be related by $\preceq$. We have already motivated this property in Example 2.4 by showing that $?\texttt{int}.\mathbf{1} \not\preceq ?\texttt{real}.\mathbf{1}$ since the additional actions emitted by $?\texttt{real}.\mathbf{1}$ may interfere with some session types that complete $?\texttt{int}.\mathbf{1}$.

The local characterization of the subsession relation permits to formally verify a number of properties of $\preceq$. Some of these, such as commutativity and associativity of $+$ and $\oplus$, are fairly obvious. Less obvious is the fact that the two choices distribute over each other, just like in other testing equivalences [De Nicola and Hennessy, 1987; Hennessy, 1988]. In Table 3 we have collected other laws related to $\preceq$ and the induced equivalence relation which we regard as particularly interesting. We spend the next few paragraphs describing them in some more detail.

Axiom (S1) concerns reduction of nondeterminism: it is safe to replace some behavior $(\sigma \oplus \tau)$ with a more deterministic one (σ) or, equivalently, it is safe to replace a channel with fewer capabilities (σ) with another one having more capabilities $(\sigma \oplus \tau)$. As the substitution of the channel does not affect the process, the process will behave more deterministically than what the type of the actual channel prescribes. Axiom (S2) states that it is safe to replace a channel which associated type is $\mathbf{1} + \sigma$ with a channel which associated type is σ. Indeed, the session type

$\mathbf{1}+\sigma$ indicates that the process using the channel is ready to interact according to σ but is also satisfied if not requested to. The session type σ, on the other hand, indicates that the interaction will occur. Axiom (S3) states that the availability of actions emitted by a prefix α are irrelevant if the corresponding continuation is non-viable, hence $\alpha.\mathbf{0} \preceq \mathbf{0}$. Observe that (S3) is not sound in standard testing theories and the presence of an action α is never irrelevant. The fundamental reason why (S3) is sound for $\preceq$ is that the notion of testing we are using is *symmetric*.

Axioms (D1) and (D2) show the interaction between choices and actions. In particular, $\alpha.\sigma + \alpha.\tau \approx \alpha.(\sigma \oplus \tau)$ tells us that the session type $\alpha.\sigma + \alpha.\tau$ denotes an internal choice between the continuations σ and τ after the prefix α: since both branches are guarded by the same prefix, there is no way for an interacting party to select one particular branch. In a sense axiom (D1) suggests that the external choice operator is more delicate than the internal choice since some external choices are actually internal choices in disguise. The law $\alpha.\sigma \oplus \alpha.\tau \approx \alpha.(\sigma \oplus \tau)$ states that it is irrelevant whether the internal choice is done before or after the action, when the action in the two branches is the same.

Axioms (SP1) and (SP2) are splitting laws concerning the communication of plain values. In particular, $?(t \cup s).\sigma \approx ?t.\sigma + ?s.\sigma$ relates input actions with external choices (accepting values of type $t \cup s$ is the same as offering the interacting party to send values that inhabit either t or s) while $!(t \cup s).\sigma \approx !t.\sigma \oplus !s.\sigma$ relates output actions with internal choices (sending a value of type $t \cup s$ is the same as internally choosing to send a value of type t or of type s). Note that this axiom holds only if neither t nor s is $\emptyset$ for otherwise $!t.\sigma \oplus !s.\sigma$ is not viable, while $!(t \cup s).\sigma$ may be.

Axiom (CH1) states that, as far as the communication of channels is concerned, internal and external choices make no difference, provided that the action is of the same kind in both branches. This is a consequence of the rules

$$\text{(R7)}\ \frac{\rho \preceq \rho'}{?\rho'.\sigma \xrightarrow{?\rho} \sigma} \qquad \text{and} \qquad \text{(R8)}\ \frac{\rho \not\preceq \rho'}{?\rho'.\sigma \xrightarrow{?\rho} \mathbf{0}}$$

(see Table 2), implying that in our theory there is no dynamic dispatching depending on the type of channels. A process that behaves according to $?\rho'.\sigma$ will always accept any channel that is sent to it, regardless of the session type associated with the channel, which may thus be incompatible with ρ'. In other words, from the point of view of synchronizations it is as if channels all have the same type, say $\diamond$ (which, practically speaking, the reader may think of as the type of URLs or of IP addresses). Under this interpretation, rule (CH1) becomes $?\diamond.\sigma + ?\diamond.\tau \approx ?\diamond.\sigma \oplus ?\diamond.\tau$ which is a reformulation of axioms (D1) and (D2). Axiom (CH2) shows that, when multiple branches accepting channels are present, the only way to be sure that no communication error occurs is to send a channel whose associated type is smaller than the type expected in each branch (indeed, $\rho \oplus \rho'$ is the greatest lower bound of ρ and ρ'). Axiom (CH3) is the dual of axiom (CH2): when channels of possibly different session types are sent, the receiver must be able to deal with all of them. Thus, if the channels being sent have associated types ρ and ρ', the receiver must be able to accept channels with associated type $\rho \vee \rho'$, which stands for least upper bound of ρ and ρ'. For example, a process behaving according to $!(!\mathtt{int}^*).\mathbf{1} \oplus !(!\mathtt{real}^*).\mathbf{1}$ is either sending a channel on which integer numbers can be sent, or a channel on which real numbers can be sent. The receiver

of this channel can use it safely only for sending values that are in the intersection $\mathtt{int} \cap \mathtt{real}$. Indeed we have $!(!\mathtt{int}^*).\mathbf{1} \oplus !(!\mathtt{real}^*).\mathbf{1} \approx !(!\mathtt{int}^*).\mathbf{1}$ since $!\mathtt{int}^* \vee !\mathtt{real}^* \approx !\mathtt{int}^*$.

Example 3.1. In axiom (CH2), the behavior $\rho \oplus \rho'$ may be non-viable even when both ρ and ρ' are. For example, we have $?(?t^\omega).\mathbf{1} \oplus ?(?s^\omega).\mathbf{1} \approx ?(?t^\omega \oplus ?s^\omega).\mathbf{1} \approx ?(?(t \cap s)^\omega).\mathbf{1}$. A process that behaves as specified by $?(?t^\omega).\mathbf{1} \oplus ?(?s^\omega).\mathbf{1}$ is waiting for a channel, but it will internally decide whether to use that channel for receiving values of type t or values of type s. Consequently, the only safe channels that can be sent to this process are those that can only carry values in the intersection $t \cap s$. If this intersection turns out to be empty, then the only channels that can be sent are useless, since their associated session type must be non-viable.

In axiom (CH3) the behavior $\rho \vee \rho'$ may not exist, in which case, no process will ever be able to accept a delegated channel from another process that behaves according to $!\rho.\sigma \oplus !\rho'.\tau$. For example, consider the behavior $!(!\mathtt{int}^\omega).\mathbf{1} \oplus !(!\mathtt{bool}^\omega).\mathbf{1}$ describing a process that either sends a channel on which integer numbers can be sent or it sends a channel on which boolean values can be sent. Since the receiver does not know which channel is actually sent, it can safely use it only for sending values of type $\mathtt{int} \cap \mathtt{bool}$. So, on one side the receiver commits to using the channel according to its session type and yet there is no value that can be safely sent on it. ■

3.2. *Pre-congruence Properties of the Subsession Relation*

Since $\preceq$ is akin to a subtyping relation it should be natural to use regardless of the context in which it is used. For this reason it is important to study its pre-congruence properties. The following result shows that $\preceq$ is a pre-congruence for prefix, parallel composition, and internal choice.

Theorem 3.2. Let $\sigma \preceq \tau$. Then **(1)** $\alpha.\sigma \preceq \alpha.\tau$; **(2)** $\rho \mid \sigma \preceq \rho \mid \tau$; **(3)** $\rho \oplus \sigma \preceq \rho \oplus \tau$ for every ρ.

Proof. **(1)** is an immediate consequence of Theorem 3.1. As regards **(2)**, we have $\theta \mid (\rho \mid \sigma)$ complete if and only if $(\theta \mid \rho) \mid \sigma$ complete, which implies $(\theta \mid \rho) \mid \tau$ complete being equivalent to $\theta \mid (\rho \mid \tau)$ complete. We conclude $\rho \mid \sigma \preceq \rho \mid \tau$ since θ is arbitrary. As regards **(3)**, it suffices to observe that $\theta \mid (\rho \oplus \sigma)$ is complete if and only if both $\theta \mid \rho$ and $\theta \mid \sigma$ are complete. □

The existence of non-viable behaviors results in a large class of indistinguishable session types, those that are not viable. For example in $\mathbf{0} \approx ?\mathtt{int}.\mathbf{0}$ we are equating a totally unobservable behavior ($\mathbf{0}$) with an observable, although still non-viable, one ($?\mathtt{int}.\mathbf{0}$). This prevents $\preceq$ from being a pre-congruence with respect to the external choice, for instance, $\mathbf{0} + ?\mathtt{int}.\mathbf{1} \not\preceq ?\mathtt{int}.\mathbf{0} + ?\mathtt{int}.\mathbf{1}$ since we know from axiom (D1) that $?\mathtt{int}.\mathbf{0} + ?\mathtt{int}.\mathbf{1} \approx ?\mathtt{int}.(\mathbf{0} \oplus \mathbf{1})$ which is not viable whereas $\mathbf{0} + ?\mathtt{int}.\mathbf{1}$ is. We can overcome this problem by focusing on the largest relation included in $\preceq$ that is a pre-congruence for $+$:

Definition 3.3 (subsession pre-congruence). Let $\sigma \sqsubseteq \tau$ if and only if $\sigma + \rho \preceq \tau + \rho$ for every component ρ. We write $\simeq$ for the equivalence relation induced by $\sqsubseteq$, namely $\simeq \,= \sqsubseteq \cap \sqsubseteq^{-1}$.

As we restrict $\preceq$ to $\sqsubseteq$ the risk is to lose some potentially interesting relations. Fortunately this is not the case and in fact $\preceq$ and $\sqsubseteq$ almost coincide, as we can see from the following local characterization for $\sqsubseteq$:

Theorem 3.3. Let σ and τ be components. Then $\sigma \sqsubseteq \tau$ if and only if:

1 $\tau \Downarrow \mathrm{S}$ implies $\sigma \Downarrow \mathrm{R}$ and $\mathrm{R} \subseteq \mathrm{S}$;
2 $\tau \overset{\lfloor \mu \rfloor}{\Longrightarrow}$ implies $\sigma \overset{\lfloor \mu \rfloor}{\Longrightarrow}$ and $\sigma(\mu) \preceq \tau(\mu)$ for every $\mu \neq \checkmark$.

The only difference between $\preceq$ and $\sqsubseteq$ is that condition (2) is required to hold for σ viable in Theorem 3.1, whereas it must hold unconditionally in Theorem 3.3. In other words we have the following:

Corollary 3.1. Let σ viable and σ, τ be components. Then $\sigma \preceq \tau$ if and only if $\sigma \sqsubseteq \tau$.

In practice using $\sqsubseteq$ in place of $\preceq$ makes no difference since one is always interested in working with viable session types (we will formalize this clearly in Section 4). In what follows we put $\sqsubseteq$ at work on some examples of derivable relations.

Example 3.2 (decomposition). It is always possible to rewrite external choices of input actions and internal choices of output actions so that the branches are disjoint with respect to basic values. More precisely, the following laws are derivable:

$$\text{(SP-IN)}\quad ?t.\sigma + ?s.\tau \simeq ?(t \setminus s).\sigma + ?(s \setminus t).\tau + ?(t \cap s).(\sigma \oplus \tau)$$

$$\text{(SP-OUT1)}\quad \frac{t \setminus s \neq \emptyset \qquad s \setminus t \neq \emptyset \qquad t \cap s \neq \emptyset}{!t.\sigma \oplus !s.\tau \simeq !(t \setminus s).\sigma \oplus !(s \setminus t).\tau \oplus !(t \cap s).(\sigma \oplus \tau)} \qquad \text{(SP-OUT2)}\quad \frac{\emptyset \subsetneq s \subseteq t}{!t.\sigma \simeq !(t \setminus s).\sigma \oplus !s.\sigma}$$

Axiom (SP-IN) shows that sending a value v to a process behaving according to the session type $?t.\sigma + ?s.\tau$ may result in three different continuations: if v has type t but not s, then the only possible continuation is σ; if v has type s but not t, then the only possible continuation is τ; if v has type $t \cap s$, then either branch can be selected and either continuation is possible. The rule can be easily derived from (SP1) and (D1) and is valid also in case any of $t \setminus s$, $s \setminus t$, or $t \cap s$ is empty because $?\emptyset.\sigma \simeq \mathbf{0}$.

Rule (SP-OUT1) is somewhat the dual of axiom (SP-IN) and regards output actions. Its derivation is straightforward from rule (SP2) and axiom (D2). The only difference with (SP-IN) is the requirement of non-emptiness for the various types in the decomposition, which derives from analogous requirements of axiom (SP2). Rule (SP-OUT2) is a specialized instance of (SP-OUT1) where $s \subseteq t$ and is just as easily derived. ■

Example 3.3 (covariance and contravariance). Input and output obey respectively the following covariance and contravariance properties:

$$\text{(S-IN-VAL)}\quad \frac{t \subseteq s}{?t.\sigma + ?(s \setminus t).\mathbf{0} \sqsubseteq ?s.\sigma} \qquad \text{(S-OUT-VAL)}\quad \frac{\emptyset \subsetneq s \subseteq t}{!t.\sigma \sqsubseteq !s.\sigma} \qquad \text{(S-IN-CH)}\quad \frac{\rho \preceq \rho'}{?\rho.\sigma \sqsubseteq ?\rho'.\sigma} \qquad \text{(S-OUT-CH)}\quad \frac{\rho' \preceq \rho}{!\rho.\sigma \sqsubseteq !\rho'.\sigma}$$

Rule (S-IN-VAL) follows from rule (SP-IN), axiom (S3), pre-congruence for $+$, and states covariance of input actions. To avoid interferences, this is safe only if the behavior after any value in $s \setminus t$ is non-viable (see also Example 2.4). Rule (S-OUT-VAL) follows from (SP-OUT2) and (S1) and states contravariance of output actions.

Rules (S-IN-CH) and (S-OUT-CH) are similar but they regard channels. Rule (S-IN-CH) follows from axioms (CH2) and (S1). Indeed, $\rho \preceq \rho'$ implies $\rho \approx \rho \oplus \rho'$. Then $?\rho.\sigma \simeq ?(\rho \oplus \rho').\sigma \simeq ?\rho.\sigma \oplus ?\rho'.\sigma \sqsubseteq ?\rho'.\sigma$. Dually, rule (S-OUT-CH) follows from axioms (CH3) and (S1), thus: $\rho' \preceq \rho$ implies $\rho \vee \rho' \approx \rho$, hence $!\rho.\sigma \simeq !(\rho \vee \rho').\sigma \simeq !\rho.\sigma \oplus !\rho'.\sigma \sqsubseteq !\rho'.\sigma$. ■

3.3. *Fairness Properties of the Subsession Relation*

At the beginning of this section we have stated that $\preceq$ is closely related to the fair testing relation. Unlike other testing relations, fair testing is insensitive to divergence and is based on the consideration that, if some action is offered infinitely often by a process, then some other process may rely on the fact that this action will eventually happen. To illustrate this concept consider the session type

$$?\mathtt{int}^\omega \mid !\mathtt{int}^\omega \mid ?1984.\mathbf{1}$$

describing three processes simultaneously accessing a channel: two of them receive and send an infinite sequence of integers; the third one is waiting for the number 1984 to appear. It is not hard to verify that this composition is complete: even though the second process keeps sending numbers different from 1984, those numbers will be read by the first process. After each synchronization between the first and the second process it is *possible* (although not granted) that the third process will send 1984 ($!1984 \in \mathsf{init}(?\mathtt{int}^\omega \mid !\mathtt{int}^\omega)$) and this is enough to entail completeness.

According to the local characterizations of $\preceq$ and $\sqsubseteq$, axiom (S1) is sound. It follows that any finite number of applications of (S1) is also sound. However, this is no longer true "in the limit", if we are allowed to apply (S1) infinitely many times. If this were the case we would be able to build the derivation

$$\dfrac{\dfrac{!\mathtt{int}^\omega \sqsubseteq !0^\omega}{!\mathtt{int}.!\mathtt{int}^\omega \sqsubseteq !\mathtt{int}.!0^\omega}\,(\text{precongruence}) \qquad \dfrac{\vdots}{!\mathtt{int}.!0^\omega \sqsubseteq !0.!0^\omega}\,(\text{S-OUT-VAL})}{!\mathtt{int}.!\mathtt{int}^\omega \sqsubseteq !0.!0^\omega}\,(\text{transitivity})$$

which coinductively proves $!\mathtt{int}^\omega \sqsubseteq !0^\omega$. This relation apparently makes sense, since $!0^\omega$ is a more deterministic behavior than $!\mathtt{int}^\omega$ or, equivalently, a process using a channel with type $!\mathtt{int}^\omega$ for sending only 0 is behaving well. However, the composition

$$?\mathtt{int}^\omega \mid !0^\omega \mid ?1984.\mathbf{1}$$

is not complete because $?1984.\mathbf{1}$ no longer has the potential synchronization with $?\mathtt{int}^\omega \mid !0^\omega$.

The problem is that axiom (S1) may prune out branches from a session type, and the pruned branches may incidentally be the ones that ensure completeness in some contexts. Such contexts are difficult to characterize because they depend on the non-local structure of session types. To illustrate what we mean by "locality" here, consider the session types defined by the following equations:

$$\sigma = !\mathtt{bool}.\mathbf{1} \oplus !\mathtt{int}.\sigma \qquad \rho_1 = !\mathtt{int}.(!\mathtt{bool}.\mathbf{1} \oplus !\mathtt{int}.\rho_1) \qquad \rho_2 = !\mathtt{bool}.\mathbf{1} \oplus !\mathtt{int}.!\mathtt{int}.\rho_2$$

where ρ_1 (respectively, ρ_2) differs from σ because all the $!\mathtt{bool}.\mathbf{1}$ branches after an even (respectively, odd) number of $!\mathtt{int}$ prefixes have been removed. It turns out that $\sigma \preceq \rho_i$ for $i \in \{1,2\}$ hold, since both ρ_1 and ρ_2 are more deterministic variants of σ and they both go infinitely many

Table 4. *Syntax of processes.*

P	::=		**process**	e	::=		**expression**
		nil	(idle)			x	(variable)
	$\mid$	$u!e.P$	(output)		$\mid$	c	(channel)
	$\mid$	$u?(x).P$	(channel input)		$\mid$	$\ldots$	
	$\mid$	$\sum_{i\in I} u?(x:t_i).P_i$	(value input)				
	$\mid$	$P \oplus P$	(internal choice)				
	$\mid$	$P \mid P$	(parallel composition)				
	$\mid$	$(\mathsf{new}\ c)P$	(restriction)				

times through a state where some boolean value may be emitted. So while in principle *any* branch $!\mathtt{bool}.\mathbf{1}$ after *any* number of $!\mathtt{int}$ prefixes can be safely removed, if we remove *all* of them (or just enough so that only a finite number of them remains), we invalidate the fairness property. Indeed we have $\sigma \not\preceq !\mathtt{int}^\omega$.

Example 3.4. In light of the preceding discussion one can verify that $!t^\omega$ is invariant with respect to t and that neither $t^\omega \preceq ?t^\omega$ nor $t^\omega \preceq !t^\omega$ do hold, while $?t^\omega$ is still covariant with respect to t like $?t^*$ (in fact covariance of $?t^\omega$ increases viable traces of the session type). Note also that the relations $?t^* \preceq ?t^\omega$, $!t^* \preceq !t^\omega$, and $t^* \preceq t^\omega$ that we have already presented in Example 2.3 are valid despite the fact that in every case infinitely many $\mathbf{1}$ branches are removed. In general the instances $\mathbf{1} \oplus \sigma \sqsubseteq \sigma$ and $\mathbf{1} \oplus \sigma \sqsubseteq \mathbf{1}$ of axiom (S1) are sound even when they are applied infinitely many times. ■

Interestingly, the notions of completeness and of subsession relation are affected by fairness only in systems made of three or more participants, as in the example at the beginning of this section. The subsession relation defined in [Castagna *et al.*, 2009a] is oblivious to fairness since in that context only dyadic sessions (those connecting exactly two processes) are considered. By contrast, the systems in [Castagna and Padovani, 2009] involve an arbitrary number of participants and consequently both completeness and the refinement relation have fairness properties.

4. A Projection for Processes

In this section we show how to project processes into the session types we have defined and studied in the previous sections.

4.1. *Syntax and Semantics of Processes*

Processes are defined by the grammar in Table 4. We use P, Q, R, ... to range over processes; we use a, b, c, ... to range over *channel names* taken from some infinite set $\mathscr{N}$; we let x, y, z, ... range over *variables* taken from some set $\mathscr{X}$. In principle we should distinguish channel variables, which can only be instantiated by channel names, from value variables, which can only be instantiated by basic values. To keep things simple we generically use the term "variable" for both and will make sure that no ambiguity may possibly arise. We let u, v, ... range over channel names and channel variables (v should not be confused with v that we used to range over

Table 5. *Axioms of structural congruence for processes.*

$P \mid \mathsf{nil}$	$\equiv$	P		(S-UNIT)
$P \mid Q$	$\equiv$	$Q \mid P$		(S-COMM)
$P \mid (Q \mid R)$	$\equiv$	$(P \mid Q) \mid R$		(S-ASSOC)
$(\mathsf{new}\ a)(\mathsf{new}\ b)P$	$\equiv$	$(\mathsf{new}\ b)(\mathsf{new}\ a)P$		(S-SWITCH)
$(\mathsf{new}\ a)\mathsf{nil}$	$\equiv$	nil		(S-NIL)
$(\mathsf{new}\ a)(P \mid Q)$	$\equiv$	$P \mid (\mathsf{new}\ a)Q$	$a \notin \mathtt{fn}(P)$	(S-EXTR)
$P \oplus Q$	$\equiv$	$Q \oplus P$		(S-CHOICE)

elements of $\mathscr{V}$); we let $e, \ldots$ range over an underspecified language of *expressions* which include variables and channel names; finally, we use m to range over *messages*, which are elements of $\mathscr{V} \cup \mathscr{N}$. The language we consider is a minor variant of the π-calculus, so we remark here only the differences: there are three action prefixes: $u!e$ denotes an output on channel u of the value or channel to which the expression e evaluates; $u?(x)$ denotes an input on channel u of a channel x; the guarded sum $\sum_{i \in I} u?(x : t_i).P_i$ denotes an input on channel u of some value v. The branch P_k is selected according to the type t_k that v belongs to (the types in the branches need not be disjoint). In these prefixes we call u *subject*. The process $P \oplus Q$ denotes an internal choice between P and Q. We adopt the following usual conventions: we omit the nil process when it immediately follows a prefix; we write $\mathtt{fn}(P)$ for the set of free variables and channel names occurring in P (the binders are input prefixes and restriction); we write $P\{{}^{m}/_{x}\}$ for the process P where all free occurrences of the variable x have been replaced by m.

As we did for session types we do not equip processes with concrete syntax for expressing infinite behaviors: simply, infinite processes are infinite trees built with the grammar in Table 4. In the examples that follow we will define infinite processes by means of possibly parametrized equations of the form

$$X(\tilde{u}) = P$$

where X is a process variable and $\tilde{u} \subseteq \mathtt{fn}(P)$ its parameters. Intuitively, an occurrence of $X(\tilde{v})$ within P stands for an occurrence of $P\{{}^{\tilde{v}}/_{\tilde{u}}\}$. More formally, we will write $X(\tilde{v})$ for the (syntactic) solution of the above equation where the parameters $\tilde{u}$ have been instantiated with $\tilde{v}$ within P. To ensure that such solution exists and is unique and that the typing rules we are about to discuss yield contractive session types we require that every occurrence of X within P must be guarded by no less than one prefix and at least one prefix with subject v for every $v \in \mathtt{fn}(P)$. The rationale for this unusually strong restriction comes from the fact that the projection produces possibly recursive session types that are associated with the single channels used by the process. For example, we want to avoid a process like

$$X = u!v.X$$

since its projection yields the (perfectly fine) equation $\sigma = !\rho.\sigma$ for channel u but also the equation $\rho = \rho \mid \rho$ for channel v, which admits a non-contractive solution. Thus, requiring that process variables must be guarded by prefixes regarding all the free names in P makes sure that the session types for all these names in the resulting projection are contractive.

Following standard presentations, the operational semantics of processes is given by a combi-

Table 6. *Reduction of processes.*

$$\text{(R-COMM)}\quad \frac{e \downarrow \mathsf{v} \qquad \mathsf{v} \in t_k \qquad k \in I}{a!e.P \mid \sum_{i \in I} a?(x:t_i).Q_i\} \to P \mid Q_k\{^{\mathsf{v}}/_x\}} \qquad \text{(R-COMMS)}\quad \frac{}{a!c.P \mid a?(x).Q \to P \mid Q\{^{c}/_x\}} \qquad \text{(R-CHOICE)}\quad \frac{}{P \oplus Q \to P}$$

$$\text{(R-NEW)}\quad \frac{P \to P'}{(\mathsf{new}\ c)P \to (\mathsf{new}\ c)P'} \qquad \text{(R-PAR)}\quad \frac{P \to P'}{P \mid Q \to P' \mid Q} \qquad \text{(R-CONG)}\quad \frac{P \equiv P' \qquad P' \to Q' \qquad Q \equiv Q'}{P \to Q}$$

nation of structural congruence rules and a reduction relation. Structural congruence is the smallest congruence that contains alpha-conversion and the rules in Table 5 and is standard, the only uncommon rule possibly being (S-CHOICE) which states the commutativity of internal choice. The reduction relation is inductively defined by the rules in Table 6: rules (R-NEW) and (R-PAR) state unremarkable reductions under contexts; rule (R-CONG) is the standard reduction up to structural congruence; rule (R-CHOICE) (possibly used in conjunction with (S-CHOICE) and (R-CONG)) states that a process $P \oplus Q$ may autonomously reduce to either P or Q; finally, rules (R-COMM) and (R-COMMS) are the two synchronization rules, whereby a message is exchanged between processes exposing complementary actions. In (R-COMM) we write $e \downarrow \mathsf{v}$ to denote that the expression e evaluates to some basic value v. Observe that a synchronization occurs only if there is some branch of the guarded sum that is capable of receiving the value v being sent, whereas in rule (R-COMMS) the synchronization occurs regardless of the type associated with the channel c being exchanged. This is consistent with the semantics of our session types, which prevents any form of dynamic dispatching based on the type of channels. In the following we write $P \to$ if there is Q such that $P \to Q$.

4.2. *A Type System for Projection*

We now get to the main point of this paper, namely the projection of a process into session types. Since the theory of session types has already been developed and studied in the previous sections, we only have to take care of the aliasing problem we anticipated in the introduction. The problem can be summarized as follows: in a process $a?(x).P$ it is not known to which channel name the variable x will be instantiated at runtime. Therefore, the projection of P, which yields a session environment Δ mapping channel names and channel variables to the corresponding session types, has a distinct entry regarding the channel variable x. This implicitly assumes that the channel variable x will not be instantiated to any channel name c already occurring in the domain of Δ for otherwise we would have to merge the session types $\Delta(c)$ and $\Delta(x)$ respectively associated with c and x. This, in general, can only be obtained by projecting the instantiated process $P\{^{c}/_x\}$ anew. Instead, we want the projection of $P\{^{c}/_x\}$ to be simply Δ where the entry for x has turned into an entry for c. To avoid the problem we introduce a channel order $\prec$ imposing a stratification over channels such that channel a can be sent over channel b only if $a \prec b$ and then requiring that a process $a?(x).P$ is well typed only if every free channel name and channel variable in P other than a and x is (strictly) larger than a according to $\prec$. In particular, another well-typed process

Table 7. *Typing rules for processes.*

$$\text{(T-SUB)}\ \frac{\Gamma;\mathscr{C}\vdash P:\Delta \qquad \Delta'\preceq\Delta}{\Gamma;\mathscr{C}\vdash P:\Delta'} \qquad \text{(T-NIL)}\ \frac{}{\Gamma;\mathscr{C}\vdash \mathsf{nil}:\emptyset} \qquad \text{(T-OUTPUT)}\ \frac{\Gamma\vdash e:t \qquad \Gamma;\mathscr{C}\vdash P:\Delta\cup\{u:\sigma\}}{\Gamma;\mathscr{C}\vdash u!e.P:\Delta\cup\{u:!t.\sigma\}}$$

$$\text{(T-INPUT)}\ \frac{\Gamma,x:t_i;\mathscr{C}\vdash P_i:\Delta\cup\{u:\sigma_i\}^{\,i\in I}}{\Gamma;\mathscr{C}\vdash \sum_{i\in I} u?(x:t_i).P_i:\Delta\cup\{u:\sum_{i\in I}?t_i.\sigma_i\}} \qquad \text{(T-OUTPUTS)}\ \frac{\Gamma;\mathscr{C}\vdash P:\Delta\cup\{u:\sigma,v:\tau\} \qquad \mathscr{C}\vdash v\prec u}{\Gamma;\mathscr{C}\vdash u!v.P:\Delta\cup\{u:!\rho.\sigma,v:\tau\,|\,\rho\}}$$

$$\text{(T-INPUTS)}\ \frac{\Gamma;\mathscr{C},x\prec u\vdash P:\Delta\cup\{u:\sigma,x:\rho\} \qquad \mathscr{C}\vdash u\prec \mathtt{dom}(\Delta)}{\Gamma;\mathscr{C}\vdash u?(x).P:\Delta\cup\{u:?\rho.\sigma\}} \qquad \text{(T-CHOICE)}\ \frac{\Gamma;\mathscr{C}\vdash P:\Delta \qquad \Gamma;\mathscr{C}\vdash Q:\Delta}{\Gamma;\mathscr{C}\vdash P\oplus Q:\Delta}$$

$$\text{(T-PAR)}\ \frac{\Gamma;\mathscr{C}\vdash P:\Delta \qquad \Gamma;\mathscr{C}\vdash Q:\Delta'}{\Gamma;\mathscr{C}\vdash P\,|\,Q:\Delta\,|\,\Delta'} \qquad \text{(T-RES)}\ \frac{\Gamma;\mathscr{C}\vdash P:\Delta\cup\{c:\sigma\} \qquad c\notin\mathtt{dom}(\Delta) \qquad \sigma\text{ complete}}{\Gamma;\mathscr{C}\setminus c\vdash(\mathsf{new}\ c)P:\Delta}$$

$a!c$ must be such that $c\prec a$, hence the reduction

$$a!c\,|\,a?(x).P\to \mathsf{nil}\,|\,P\{^{c}/_{x}\}$$

yields a residual $P\{^{c}/_{x}\}$ where c itself is strictly smaller than (hence different from) a and, by transitivity of $\prec$, any other free channel name in P.

Let us now proceed to the formal definition of the type system.

Definition 4.1 (channel order). A *channel order* is a strict partial order over $\mathscr{N}\cup\mathscr{X}$.

We use $\mathscr{C},\dots$ to range over channel orders and we introduce the following notation:

— we write $\mathscr{C}\vdash u\prec v$ if $(u,v)\in\mathscr{C}$;
— we write $\mathscr{C}\{^{c}/_{x}\}$ for $\{(u\{^{c}/_{x}\},v\{^{c}/_{x}\})\mid(u,v)\in\mathscr{C}\}$;
— we write $\mathscr{C}\setminus c$ for the restriction of $\mathscr{C}$ on $(\mathscr{N}\setminus\{c\})\cup\mathscr{X}$;
— we write $\mathscr{C}\vdash M\prec N$ if $\mathscr{C}\vdash u\prec v$ for every $u\in M$ and $v\in N$ where $M,N\subseteq\mathscr{N}\cup\mathscr{X}$;
— we write $\mathscr{C},u\prec v$ for the smallest transitive relation that includes $\mathscr{C}\cup\{(u,v)\}$.

The typing rules for the process language are defined in Table 7. Judgments have the form $\Gamma;\mathscr{C}\vdash P:\Delta$, where Γ is a standard environment mapping value variables to basic types, $\mathscr{C}$ is a channel order, and Δ is a *session environment* mapping channel names and channel variables to session types. To avoid clutter, we will usually omit Γ and/or $\mathscr{C}$ when they are empty or unimportant. If X is an environment, we write $\mathtt{dom}(X)$ for the domain of X and $\emptyset$ for the empty environment. We assume, for every judgment $\Gamma;\mathscr{C}\vdash P:\Delta$, a basic form of hygiene by requiring $\mathtt{dom}(\Gamma)\cap\mathtt{dom}(\Delta)=\emptyset$. This implicitly imposes an unambiguous sorting of variables whereby the same variable x cannot simultaneously denote a basic value ($x\in\mathtt{dom}(\Gamma)$) and a channel ($x\in\mathtt{dom}(\Delta)$). The type system makes use of a $\preceq$ relation and a $|$ operator over session environments which naturally extend the same concepts over session types:

— let $\{u_i:\sigma_i\}^{i\in I}\,|\,\{u_i:\tau_i\}^{i\in I}$ be the session environment $\{u_i:\sigma_i\,|\,\tau_i\}^{i\in I}$;

— let $\{u_i : \sigma_i\}^{i\in I} \preceq \{u_j : \tau_j\}^{j\in J}$ if $J \subseteq I$ and $\sigma_i \preceq \tau_i$ for every $i \in J$ and $\sigma_i \preceq \mathbf{1}$ for every $i \in I \setminus J$.

The session environment $\Delta \,|\, \Delta'$, which is defined only when $\mathtt{dom}(\Delta) = \mathtt{dom}(\Delta')$, is obtained by pointwise composition of the session types in Δ and Δ'. The relation $\Delta \preceq \Delta'$ holds if every session type in Δ is $\preceq$-smaller than the corresponding session type in Δ', if there is one. Every session type in Δ which has no corresponding one in Δ' must be smaller than the successfully terminated session type $\mathbf{1}$. This relation is crucial in the definition of the sole non-structural rule of the type system, (T-SUB), which we describe first.

(T-SUB) This rule looks very much alike a subsumption rule, except that it seems to work "the other way round" in that it allows the session environment Δ associated with a process P to become a smaller Δ' in the conclusion. The apparently non-standard direction in which the subsession relation is applied in this rule can be explained by noting that, in our setting, "smaller" roughly means "less deterministic", hence less precise. Thus, if we view a judgment $P : \Delta$ as stating that Δ is the "type" of P, rule (T-SUB) allows us to give P a less precise "type" than the one it actually has. This is precisely what happens in a standard subsumption rule such as

$$\frac{\vdash e : t \qquad t \subseteq s}{\vdash e : s}$$

which decreases the precision of the type for e.

For instance, suppose $\vdash P : \{c : !\mathtt{int}.\mathbf{1}\}$, namely P sends integer values on channel c. Rule (T-SUB) states that it is safe to declare P as if it behaves according to $!\mathtt{real}.\mathbf{1}$ on c, even though we know that in reality P sends only values from a proper subset of `real`. By declaring that P is less deterministic than it actually is we are imposing stronger constraints on the environment in which P executes, as other processes waiting for messages on c will have to be ready to receive real numbers, not just integer ones. The second purpose of rule (T-SUB) is to enrich the session environment with session types for channels that are not used by P. For example, if $\vdash P : \emptyset$, meaning that P does not use any channel, we may also derive $\vdash P : \{c : \mathbf{1}\}$ which is a different way for stating the same thing. The ability to enrich session environment with explicit information about unused channels is essential in rules (T-INPUT), (T-CHOICE), and (T-PAR).

(T-NIL) The idle process nil does not perform any action on any channel, hence it is projected into the empty session environment. Observe that nil denotes a process that performs no action and that uses no channel, while the failed session type $\mathbf{0}$ denotes a communication error.

(T-OUTPUT) Rule (T-OUTPUT) types output actions of basic values of type t over channel u. 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. If the projection of P on u is σ, then the projection of $u!e.P$ on u is $!t.\sigma$. Just like e may evaluate to *any* value of type t, the behavior $!t.\sigma$ may internally reduce to $!\mathsf{v}.\sigma$ for every $\mathsf{v} \in t$.

(T-INPUT) Rule (T-INPUT) types guarded sums denoting input actions for basic values over channel u. If the projection of P_i on u is σ_i, then the projection of $\sum_{i\in I} u?(x : t_i).P_i$ on u is $\sum_{i\in I} ?t_i.\sigma_i$. Observe that the session environment of every branch must be the same, except possibly for the session type associated with u. This can be ensured by repeated applications of rule (T-SUB).

(T-OUTPUTS) Rule (T-OUTPUTS) types delegations, whereby a channel v is sent over another channel u. When delegating a channel, the process expects the receiver to behave on v according

to some session type ρ; at the same time, the process may continue using the delegated channel according to some behavior τ. Hence the overall behavior on v which appears in the resulting session environment is determined by the composition $\tau \mid \rho$, even though the ρ part of the behavior is actually taken care of by a different process. The projection of the process on u is similar to that in rule (T-OUTPUT), except that in this case the $!\rho$ prefix denotes the communication of a channel with associated type ρ. Rule (T-OUTPUTS) requires the delegated channel v to be strictly smaller than u according to the channel order $\mathscr{C}$. This ensures that $v \neq u$ and that v is not owned by the receiver process.

(T-INPUTS) Rule (T-INPUTS) types an input action for a channel x on channel u. The continuation P commits to behaving according to ρ on the received channel, and the projection of the process on u is similar to that in rule (T-INPUT), but the $?\rho$ prefix denotes the input of a channel. The premise $\mathscr{C} \vdash u \prec \mathtt{dom}(\Delta)$ imposes that every free channel variable or channel name in the continuation P must be strictly larger than u, and the continuation itself is typed using the enriched channel order $\mathscr{C}, x \prec u$ which records the assumption that x is strictly smaller than u. This assumption is in fact enforced on the sender in rule (T-OUTPUTS) and is sufficient to ensure that the received channel is not among the ones that P already owns.

(T-CHOICE) This rule types internal choices and requires the two branches P and Q of the choice to have the same session environment. In practice P and Q need not necessarily use the same channels, and those that are used in both branches need not be used according to the same behavior. The requirement imposed by rule (T-CHOICE) can always be satisfied by appropriate applications of rule (T-SUB). For example, suppose $\Gamma \vdash P : \Delta_1$ and $\Gamma \vdash Q : \Delta_2$ where $\Delta_1 = \{u : \sigma_1\}$ and $\Delta_2 = \{u : \sigma_2, v : \tau\}$ and $u \neq v$. Then $\{u : \sigma_1 \oplus \sigma_2, v : \mathbf{1} \oplus \tau\} \preceq \Delta_i$ for every $i \in \{1,2\}$. Note that, although it is always possible to find a lower bound to every pair of session environments, this does not guarantee that every session type in the resulting session environment is viable (we have discussed an instance of this problem in Example 3.1).

(T-PAR) This rule expresses more clearly than any other rule the idea of projected behavior we are pursuing. Unlike other session type systems, the rule allows (actually requires) both processes to use the same channels, whose corresponding projections are composed with $\mid$ (recall that $\Delta \mid \Delta'$ is defined only if $\mathtt{dom}(\Delta) = \mathtt{dom}(\Delta')$). Rule (T-SUB) can be used to make sure that the session environments for P and Q have the same domain, recalling that $\mathbf{1}$ is neutral for $\mid$.

(T-RES) In a restriction $(\mathsf{new}\ c)P$ the overall behavior of P projected on c must not require any external contribution in order to have progress since c is invisible from outside the restriction. This property is precisely captured by the notion of completeness, hence the behavior σ associated with the restricted channel c must be complete. The rule does not prevent c from being extruded but, as we have seen while commenting rule (T-OUTPUTS), the composition σ already takes into account any behavior implemented by external processes to which c is delegated. It is possible that the restricted channel c is communicated over other channels in P, just as it is possible that some channels are communicated over c in P. For these operations to be typed correctly, c must be ordered appropriately while typing P. This rule simply "guesses" the right channel order $\mathscr{C}$ for typing P, and requires the channel order $\mathscr{C} \setminus c$ in the conclusion to have no trace of c, but to be coherent with $\mathscr{C}$ otherwise.

Before studying the soundness properties of the type system let us have a look at a few examples of typing derivations. All of the presented examples focus on the main novelty of our

approach based on projected behavior, which allows the non-linear usage of (private) channels and the parallel composition of session types.

Example 4.1. In the introduction we have described a conversation between the processes *Seller*, $Buyer_1$, and $Buyer_2$. Using the typing rules in Table 7 we can now formally derive the projections we have intuitively obtained there. In particular, regarding the $Buyer_1$ process we have:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathscr{C}\vdash \mathsf{nil}:\emptyset}{\mathscr{C}\vdash \mathsf{nil}:\{a:\mathbf{1},b:\mathbf{1},c:\mathbf{1}\}}\,(\text{T-SUB}) \quad \mathscr{C}\vdash c\prec b}{\mathscr{C}\vdash b!c:\{a:\mathbf{1},b:!\theta.\mathbf{1},c:\mathbf{1}\mid\theta\}}\,(\text{T-OUTPUTS})}{\mathscr{C}\vdash c!\text{“}\ldots\text{”}.b!c:\{a:\mathbf{1},b:!\theta.\mathbf{1},c:!\mathtt{string}.(\mathbf{1}\mid\theta)\}}\,(\text{T-OUTPUT}) \quad \mathscr{C}\vdash c\prec a}{\mathscr{C}\vdash a!c.c!\text{“}\ldots\text{”}.b!c:\{a:!\rho.\mathbf{1},b:!\theta.\mathbf{1},c:\rho\mid!\mathtt{string}.(\mathbf{1}\mid\theta)\}}\,(\text{T-OUTPUTS})}{\vdash(\mathsf{new}\ c)a!c.c!\text{“}\ldots\text{”}.b!c:\{a:!\rho.\mathbf{1},b:!\theta.\mathbf{1}\}}\,(\text{T-RES})$$

for some appropriate ρ and θ such that $\rho\mid!\mathtt{string}.(\mathbf{1}\mid\theta)$ is complete. In this derivation we take $\mathscr{C}=\{(c,a),(c,b)\}$, which is the least channel order that is necessary for typing the restricted process in $Buyer_1$. Observe that $\mathscr{C}$ does not impose any particular relation between a and b, consequently $Buyer_1$ can be typed using the empty channel order. ■

Example 4.2 (buffer). To illustrate an example of typing derivation for an infinite process we consider $B(a,b)$ from [Milner, 1999], which is defined by

$$B(a,b) \quad = \quad a?(x:t).b!x.B(a,b)$$

and which represents a 1-place buffer parametrized on two channels a and b for respectively writing to and reading from the buffer values of type t. Observe that $B(a,b)$ satisfies the contractivity condition we mentioned when describing the process language. To type $B(a,b)$ we postulate an assumption about its projection, in particular

$$\vdash B(a,b):\{a:\sigma,b:\tau\}$$

having care to mention, in the session environment, only channels that occur within $B(a,b)$. Now we type the definition of $B(a,b)$ under this assumption:

$$\dfrac{\dfrac{\dfrac{x:t\vdash B(a,b):\{a:\sigma,b:\tau\}}{x:t\vdash b!x.B(a,b):\{a:\sigma,b:!t.\tau\}}\,(\text{T-OUTPUT})}{\vdash a?(x:t).b!x.B(a,b):\{a:?t.\sigma,b:!t.\tau\}}\,(\text{T-INPUT})$$

and we know from Example 2.1 that the (unique) solutions of the equations $\sigma=?t.\sigma$ and $\tau=!t.\tau$ are $?t^\omega$ and $!t^\omega$, respectively.

We may increase the capacity of the buffer by composing a suitable number of 1-place buffers. For example $(\mathsf{new}\ c)(B(a,c)\mid B(c,b))$ is a buffer with capacity 2 that can be typed as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash B(a,c):\{a:?t^\omega,c:!t^\omega\}}{\vdash B(a,c):\{a:?t^\omega,b:\mathbf{1},c:!t^\omega\}}\,(\text{T-SUB}) \quad \dfrac{\vdash B(c,b):\{b:!t^\omega,c:?t^\omega\}}{\vdash B(c,b):\{a:\mathbf{1},b:!t^\omega,c:?t^\omega\}}\,(\text{T-SUB})}{\vdash B(a,c)\mid B(c,b):\{a:?t^\omega\mid\mathbf{1},b:\mathbf{1}\mid!t^\omega,c:!t^\omega\mid?t^\omega\}}\,(\text{T-PAR})}{\vdash(\mathsf{new}\ c)(B(a,c)\mid B(c,b)):\{a:?t^\omega\mid\mathbf{1},b:\mathbf{1}\mid!t^\omega\}}\,(\text{T-RES})}{\vdash(\mathsf{new}\ c)(B(a,c)\mid B(c,b)):\{a:?t^\omega,b:!t^\omega\}}\,(\text{T-SUB})$$

where the topmost applications of (T-SUB) enrich the session environment with information on unused channels, so that the subsequent application of (T-PAR) is possible, (T-RES) is justified since $!t^{\omega} \mid ?t^{\omega}$ is a complete session type, and the application of (T-SUB) at the bottom gets rid of $\mathbf{1}$ subterms which are neutral for $\mid$. Observe that the projection of the two-place buffer coincides with that of the 1-place buffer: any information about the capacity of the buffer is abstracted away in its projection. ■

Example 4.3 (multi-party session). Consider the system

$$(\mathsf{new}\ c)(a!c \mid b!c \mid \mathit{Client}) \mid a?(x).\mathit{Server} \mid b?(x).\mathit{Server}$$

composed by a client and two servers defined as

$$\begin{array}{rcl} \mathit{Client} & = & c!\mathit{number}.\mathit{Client} \oplus c!\mathtt{false}.c!\mathtt{false} \\ \mathit{Server} & = & x?(y:\mathtt{int}).\mathit{Server} + x?(y:\mathtt{false}) \end{array}$$

The client process establishes a multi-party session by creating a fresh channel c that is forwarded to both servers located at a and b. Namely, the system reduces to

$$(\mathsf{new}\ c)(\mathit{Client} \mid \mathit{Server}\{{}^{c}/_{x}\} \mid \mathit{Server}\{{}^{c}/_{x}\})$$

where the private channel c is owned by all three processes in the system.

The client issues an unbounded number of requests on which the servers compete, so that multiple requests can be processed concurrently by the two servers, and the first that completes a request may begin serving the next one. At any time the client may close the session by sending a `false` signal to the servers. The client side of the system can be projected as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\mathscr{C} \vdash a!c : \{a : !\sigma.\mathbf{1}, c : \mathbf{1} \mid \sigma\}}}{\mathscr{C} \vdash a!c : \{a : !\sigma.\mathbf{1}, b : \mathbf{1}, c : \sigma\}}\ (\text{T-SUB}) \quad \dfrac{\dfrac{\vdots}{\mathscr{C} \vdash b!c : \{b : !\sigma.\mathbf{1}, c : \mathbf{1} \mid \sigma\}} \quad \dfrac{\dfrac{\vdots}{\mathscr{C} \vdash \mathit{Client} : \{c : \rho\}}}{\mathscr{C} \vdash \mathit{Client} : \{b : \mathbf{1}, c : \rho\}}\ (\text{T-SUB})}{\mathscr{C} \vdash b!c \mid \mathit{Client} : \{b : !\sigma.\mathbf{1} \mid \mathbf{1}, c : \mathbf{1} \mid \sigma \mid \rho\}}\ (\text{T-PAR})}{\mathscr{C} \vdash b!c \mid \mathit{Client} : \{a : \mathbf{1}, b : !\sigma.\mathbf{1}, c : \sigma \mid \rho\}}\ (\text{T-SUB})}{\mathscr{C} \vdash a!c \mid b!c \mid \mathit{Client} : \{a : !\sigma.\mathbf{1} \mid \mathbf{1}, b : \mathbf{1} \mid !\sigma.\mathbf{1}, c : \sigma \mid \sigma \mid \rho\}}\ (\text{T-PAR})$$

$$\dfrac{\mathscr{C} \vdash a!c \mid b!c \mid \mathit{Client} : \{a : !\sigma.\mathbf{1} \mid \mathbf{1}, b : \mathbf{1} \mid !\sigma.\mathbf{1}, c : \sigma \mid \sigma \mid \rho\}}{\mathscr{C} \vdash a!c \mid b!c \mid \mathit{Client} : \{a : !\sigma.\mathbf{1}, b : !\sigma.\mathbf{1}, c : \sigma \mid \sigma \mid \rho\}}\ (\text{T-SUB})$$

$$\dfrac{\mathscr{C} \vdash a!c \mid b!c \mid \mathit{Client} : \{a : !\sigma.\mathbf{1}, b : !\sigma.\mathbf{1}, c : \sigma \mid \sigma \mid \rho\}}{\vdash (\mathsf{new}\ c)(a!c \mid b!c \mid \mathit{Client}) : \{a : !\sigma.\mathbf{1}, b : !\sigma.\mathbf{1}\}}\ (\text{T-RES})$$

by taking $\mathscr{C} = \{(c,a),(c,b)\}$ and $\sigma = ?\mathtt{int}.\sigma + ?\mathtt{false}.\mathbf{1}$ and $\rho = !\mathtt{int}.\rho \oplus !\mathtt{false}.!\mathtt{false}.\mathbf{1}$ (observe that, in the application of rule (T-RES), the session type $\sigma \mid \sigma \mid \rho$, which combines the behavior of three processes running in parallel, is complete). The reader may easily complete the projection and verify that the whole system is well typed. ■

Example 4.4 (chat room service). The process *Chat* defined by

$$\begin{array}{rcl} \mathit{Chat} & = & (\mathsf{new}\ c)(\mathit{Join} \mid \mathit{Publish}) \\ \mathit{Join} & = & a!c.c!\text{“Welcome”}.\mathit{Join} \\ \mathit{Publish} & = & c?(x:\mathtt{string}).\mathit{Publish} \end{array}$$

models a chat room service as an unbounded multi-party session where an arbitrary number of participants may join and post messages. Each participant joins the chat room by receiving

channel c on the public channel a, and the service notifies this by posting a welcome message. On channel c each participant is allowed to send an arbitrary number of strings, which are collected by the process *Publish* and published in some unspecified way.

For projecting *Join* we begin with the assumption

$$\mathscr{C} \vdash \mathit{Join} : \{a : !(!\mathtt{string}^*)^\omega, c : !\mathtt{string}^\omega\} \qquad \text{where} \qquad \mathscr{C} = \{(c,a)\}$$

and derive

$$\dfrac{\dfrac{\dfrac{\mathscr{C} \vdash \mathit{Join} : \{a : !(!\mathtt{string}^*)^\omega, c : !\mathtt{string}^\omega\}}{\mathscr{C} \vdash c!\text{“}\ldots\text{”}.\mathit{Join} : \{a : !(!\mathtt{string}^*)^\omega, c : !\mathtt{string}^\omega\}}\,(\text{T-OUTPUT}) \qquad \mathscr{C} \vdash c \prec a}{\mathscr{C} \vdash a!c.c!\text{“}\ldots\text{”}.\mathit{Join} : \{a : !(!\mathtt{string}^*)^\omega, c : !\mathtt{string}^\omega \mid !\mathtt{string}^*\}}\,(\text{T-OUTPUTS})}{\mathscr{C} \vdash \mathit{Join} : \{a : !(!\mathtt{string}^*)^\omega, c : !\mathtt{string}^\omega\}}\,(\text{T-SUB})$$

Since $!\mathtt{string}^\omega = !\mathtt{string}.!\mathtt{string}^\omega$ the session type associated with c does not change while going from the premise to the conclusion of rule (T-OUTPUT). In the application of (T-SUB) we use the property $!\mathtt{string}^\omega \preceq !\mathtt{string}^\omega \mid !\mathtt{string}^*$, which may be surprising. According to the session type $!\mathtt{string}^*$ associated with the exported channel c, each participant may send an arbitrary number of messages, but it may also decide to stop sending them at any time. This behavior gives less guarantees than $!\mathtt{string}^\omega$, and in fact we have seen that $!\mathtt{string}^* \preceq !\mathtt{string}^\omega$. However, the possible termination of each participant is compensated by the fact that unlimited participants can join the chat room at any time (recall that *Join* and each participant run in parallel). So, the assumption we have made in *Join* about channel c, which is typed with $!\mathtt{string}^\omega$, “absorbs” the $!\mathtt{string}^*$ behavior of each participant. In *Join* we could have assumed c with type $!\mathtt{string}^*$ as well, since $!\mathtt{string}^* \preceq !\mathtt{string}^* \mid !\mathtt{string}^*$ also holds. However, this assumption would have made *Chat* ill-typed, because $!\mathtt{string}^* \mid ?\mathtt{string}^\omega$, which is the composition of the projections for c of processes *Join* and *Publish*, is not complete. ■

4.3. *Properties of the Projection*

We now present a standard sequence of results proving that the provided type system is sound. First we show that the projection of a process is not affected by structural congruence rules:

Lemma 4.1. If $\Gamma; \mathscr{C} \vdash P : \Delta$ and $P \equiv Q$, then $\Gamma; \mathscr{C} \vdash Q : \Delta$.

Before showing that projections are preserved by reductions, we must realize that the existence of a projection for a process does not necessarily mean that the process “behaves well”. For example we have

$$\vdash c?(x : \emptyset) : \{c : ?\emptyset.\mathbf{1}\}$$

Not only $c?(x : \emptyset)$ cannot make any progress, but it cannot cooperate with any other process. This is witnessed by the existence, in its session environment, of the session type $?\emptyset.\mathbf{1}$, which is not viable. Viability of the session types in a session environment is really the property that characterizes *well-typedness* of processes and that we need as hypothesis to the type preservation theorem. We say that a session environment Δ is viable if so is every session type in the range of Δ.

Theorem 4.1 (type preservation). Let $\Gamma; \mathscr{C} \vdash P : \Delta$ and $P \to Q$ and Δ viable. Then $\Gamma; \mathscr{C} \vdash Q : \Delta$.

If compared with standard session type theories, the notion of *viability* looks like an additional complication of our setting. However, the rules in Table 7 should really be thought of as projection rules, rather than typing rules: they synthesize the behavior of a process into its session environment, and they do so by imposing as few constraints as possible on how processes act on channels. In standard session type theories there is no need for the concept of viability because session types are syntactically guaranteed to be viable and the (implicit) assumption is made that no basic type can be empty. The next example shows that viability of the session environment is necessary for Theorem 4.1 to hold.

Example 4.5. Consider the process $P \mid Q$ where

$$P = (\mathsf{new}\ c)a!c \qquad \text{and} \qquad Q = a?(x).x!3$$

Process P creates a fresh channel c, it sends c to Q, and does not use it anymore. Process Q sends an integer on the channel it receives from P. According to the rules in Table 7 we have

$$\vdash P \mid Q : \{a : !\mathbf{1}.\mathbf{1} \mid ?(!\mathtt{int}.\mathbf{1}).\mathbf{1}\}$$

In particular, the session type associated with a is not viable, because $\mathbf{1} \preceq !\mathtt{int}.\mathbf{1}$ does not hold. Indeed, we have the reduction

$$P \mid Q \to (\mathsf{new}\ c)(\mathsf{nil} \mid c!3)$$

where the residual process is ill typed, since c is associated with the session type $\mathbf{1} \mid !\mathtt{int}.\mathbf{1}$ which is not complete, hence it does not satisfy the premise of rule (T-RES). ■

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

Example 4.6 (deadlock). Consider the process

$$(\mathsf{new}\ a)(\mathsf{new}\ b)(a!3.b?(x : \mathtt{bool}) \mid b!\mathtt{true}.a?(x : \mathtt{int}))$$

where the channels a and b have respectively type $\sigma = !\mathtt{int}.\mathbf{1} \mid ?\mathtt{int}.\mathbf{1}$ and $\tau = ?\mathtt{bool}.\mathbf{1} \mid !\mathtt{bool}.\mathbf{1}$. Both σ and τ are complete, hence the process is well-typed. Nonetheless, it is unable to perform any reduction. ■

The safety property we are able to state guarantees that, if all the processes sharing some channel c are immediately ready to communicate on c, then they will eventually synchronize. Since in our reduction for processes synchronization is triggered not just by the channels on which messages are exchanged, but also by the type of the exchanged messages, the eventual synchronization translates to the fact that there is no communication error: it is never the case that there is a process willing to send a message of some type, and no other process is ever willing to receive messages of that particular type. The notion of "readiness" we mentioned is formalized thus:

Definition 4.2 (readiness). We say that P is *ready* on c if $P \downarrow c$ is derivable by the rules:

$$c!e.P \downarrow c \qquad c?(x).P \downarrow c \qquad \sum_{i\in I} c?(x:t_i).P_i \downarrow c \qquad \frac{P \downarrow c \qquad Q \downarrow c}{P \mid Q \downarrow c}$$

Intuitively, P is ready on c if c is the subject of every unguarded prefix in P. Also, a ready process does not have any unguarded internal choice, hence its only possibility to perform a reduction step is by means of a synchronization rule (either (T-COMM) or (T-COMMS)).

Theorem 4.2. If $\Gamma;\mathscr{C} \vdash P : \Delta \cup \{c : \sigma\}$ and σ complete and $P \downarrow c$, then $P \rightarrow$.

Since reductions propagate through parallel compositions (R-PAR) and restrictions (R-NEW), Theorem 4.2 can be generalized whenever P is some unguarded sub-process that is ready on c. In particular, if $\Gamma;\mathscr{C} \vdash P \mid Q : \Delta \cup \{c : \sigma\}$ and $c \notin \mathtt{fn}(Q)$ or if $\Gamma;\mathscr{C} \vdash (\mathsf{new}\ d)P : \Delta \cup \{c : \sigma\}$ and $c \neq d$, then $\Gamma;\mathscr{C} \vdash P : \Delta' \cup \{c : \sigma\}$ for some Δ'.

5. Related Work

Since their introduction session types have been extended in many ways and applied to several programming paradigms. Nonetheless, in each theory session types are roughly characterized by the same features: the ability to describe sequences of exchanged messages having possibly different types; some form of branching and branch selection; two dual branching modalities which we called internal and external choices in this work; the ability to express recursive behaviors. In our theory sequencing and branching are modeled by means of algebraic operators of a proper process calculus: prefix for sequencing, two behavioral operators corresponding to internal and external choice. We have avoided an explicit representation of recursive behaviors and worked instead with infinite trees, like in [Castagna *et al.*, 2009a].

Session type theories differentiate also by the basic mechanism that drives branch selection: it can be labels [Honda *et al.*, 1998; Gay and Hole, 2005; Vasconcelos *et al.*, 2006; Gay and Vasconcelos, 2007; Honda *et al.*, 2008], object types [Capecchi *et al.*, 2009; Drossopoulou *et al.*, 2007], or even the type of channels [Castagna *et al.*, 2009a]. In this work, as in [Castagna *et al.*, 2009a], we generalize the label-driven approach to a type-driven one, but we do not allow branching to be affected by the type of channels. This is a direct consequence of rule

$$\text{(R8)}\ \frac{\rho \not\preceq \rho'}{?\rho'.\sigma \xrightarrow{?\rho} \mathbf{0}}$$

in the operational semantics of session types, modelling the fact that the reception of channels with the "wrong type" yields an unrecoverable error. Hence, well-typed processes are guaranteed to exchange only channels with the "correct type". This approach limits the expressiveness of our language of session types, if compared to that in [Castagna *et al.*, 2009a], but results in a slightly simpler theory. From a practical point of view it is also simpler to implement, since it does not require session types to be represented and maintained at run-time (recall that the session type associated with a channel changes with time). This fact can already be appreciated in the

communication rules of our process calculus (Section 4)

$$\text{(R-COMM)}\quad \frac{e \downarrow \mathsf{v} \qquad \mathsf{v} \in t_k \qquad k \in I}{a!e.P \,|\, \sum_{i\in I} a?(x:t_i).Q_i\} \to P \,|\, Q_k\{{}^{\mathsf{v}}/_{x}\}} \qquad \text{and} \qquad \text{(R-COMMS)}\quad \frac{}{a!c.P \,|\, a?(x).Q \to P \,|\, Q\{{}^{c}/_{x}\}}$$

where it is clear that communication of basic values implies a run-time check for their type ($\mathsf{v} \in t_k$) whereas no such check is required for the communication of channels. Observe that if we equipped the transition system of session types with the rule

$$\frac{\mathsf{v} \notin t}{?t.\sigma \xrightarrow{?\mathsf{v}} \mathbf{0}}$$

which corresponds to (R8) but for basic values, we would disable (at the level of session types) dynamic dispatching also on the type of basic values, because any input operation would be able to receive *every* basic value, although some of these would lead to an unrecoverable error. When desired, as in Examples 2.1 and 2.3, this feature can be encoded by the behavior $?t.\sigma + ?\neg t.\mathbf{0}$.

The main difference between our approach and existing ones lies in the presence of a parallel composition operator for session types, in the spirit of conversation types by Caires and Vieira [2009]. In a sense, we have started from the idea that two (or more) participants must be combined together in order to communicate, which is common to all works on session types, and internalized it directly in the session type language by providing a composition operator.

Defining session types as a process language has allowed us to study their theory using well-known process algebraic techniques. This was already done in [Castagna *et al.*, 2009a] regarding the dyadic setting. Here we have generalized the approach to sessions with an arbitrary number of participants. This has required a notion of completeness (Definition 2.2) which is more involved than that of duality that we find in dyadic settings. Interestingly, the additional complication arises not so much from the number of participants, as from the fact that the most natural formulation of completeness induces a *fairness* property in the resulting subsession relation. Indeed, while the subsession relation in [Castagna *et al.*, 2009a] is akin to the *must* testing pre-order [De Nicola and Hennessy, 1984, 1987; Hennessy, 1988], the subsession relation in this work (Definition 2.3) shares some properties with the *should* testing pre-order [Natarajan and Cleaveland, 1995; Rensink and Vogler, 2007]. With respect to the earlier version of this paper [Padovani, 2009], however, we have adopted a different notion of *liveness* which is similar to a symmetric variant of the error-freedom relation in [Acciai and Boreale, 2008]. In [Padovani, 2009] a session type σ is live if $\sigma \stackrel{\checkmark}{\Longrightarrow}$, namely if σ has the ability to reach a successfully terminated state, hence completeness is somewhat related to *total correctness* (a final state is always reachable). In the present paper we have tried to relax this into *partial correctness* (no bad state is ever reached). For us, a bad state is one where some non-terminated participant is no longer capable of synchronizing with any other participant within the same session. We argue that the notion of completeness we use in this work fits more naturally a type-theoretic setting and allows us to type a large number of relatively simple processes that occur often in practice and that are considered well behaved. For example, in typing the combined buffer (Example 4.2) we have used the property that $!t^{\omega} \,|\, ?t^{\omega}$ is complete, but the same session type is non-viable in [Padovani, 2009] because $\checkmark \notin \mathsf{init}(!t^{\omega} \,|\, ?t^{\omega})$. Alas, the formulation of completeness in this work is far less elegant

and succinct than the one in [Padovani, 2009], and does not avoid the subtleties of fairness in the resulting subsession relation (Section 3.3). Alternative characterizations of fair/should testing relations have been defined by Natarajan and Cleaveland [1995] and Bugliesi *et al.* [2010]. In all these works the characterizations highlight the "non-local" nature of fairness, which makes it more awkward to apply in practice.

A distinctive feature of the language of processes we have used in [Padovani, 2009] and also in this paper (Section 4) is that we do not discriminate between private and public channels (sometimes referred to by the adjectives *live* and *shared* in other theories), there is no construct specifically dedicated to session-oriented communication, and we do not impose any linearity constraint on the use of channels. So, if on one hand the use we make of the term "session" may be improper as sessions were specifically introduced as a structuring construct, on the other hand we have shown that the essence of sessions can be captured purely at the level of types. This led us to re-discover and refine channel types [Pierce and Sangiorgi, 1996; Sangiorgi and Walker, 2001] as well as some known properties of their semantics [Castagna *et al.*, 2008; Carpineti *et al.*, 2009]. [Vasconcelos, 2009; Giunti and Vasconcelos, 2010] also provide type systems for pure π-calculus processes, but the linearly typed channels implicitly model dyadic sessions.

With respect to the process language in [Padovani, 2009], here we have operated several changes: syntactically, we use recursive definitions instead of replication for modeling infinite behaviors. As observed elsewhere [Honda *et al.*, 1998], recursion seems to better match concrete programming paradigms, but the results we have obtained here easily extend to a language with replication as we did in [Padovani, 2009]. Also, we have preferred a reduction semantics for processes to the labeled transition relation used in [Padovani, 2009], which turns out to be simpler to understand and results in simpler proofs. The most important difference with [Padovani, 2009] is that we have relaxed the type system, in particular rule (T-INPUTS), by means of a channel order that stratifies channels. The stratification is already enforced on the session types associated with channels (Remark 2.1), but in the type system it has to be made explicit because the typing rules only have a partial view of the session types. The aliasing phenomenon we avoid by means of a channel order is known to be source of trouble in general settings not involving any sort of typing on processes. For instance, it is partially responsible for the unpopularity of the mismatch operator in process calculi [Sangiorgi and Walker, 2001]. Within the more restricted scope of session type theories, the same phenomenon has been addressed in a number of ways (Yoshida and Vasconcelos [2007] provide a clear survey on this matter). In [Honda *et al.*, 1998] the synchronization rule modeling channel passing has the form

$$a!c.P \mid a?(c).Q \to P \mid Q$$

where alpha-conversion is implicitly used for renaming the channel c bound in the input prefix $a?(c)$ so that it matches the channel being sent. If alpha-conversion is not possible because c occurs free in Q, then no synchronization takes place. As pointed out by Yoshida and Vasconcelos [2007], this solution is not completely satisfactory since it implies a runtime check. Furthermore, if adopted in our context it would force us to weaken Theorem 4.2, for well-typedness and readiness of processes would no longer guarantee a synchronization involving a delegation. Gay and Hole [2005] use *polarities* for distinguishing the two ends of session channels, and impose a linear usage of these channels. This way, each end of a session channel is always owned by exactly one process at any time, thus no merging of session types is ever required. This solution

is unfeasible in our approach, which heavily relies on non-linear usage of channels (including private ones). A similar solution is also proposed by Vasconcelos [2009], where channels are not polarized but restrictions bind two (distinct) channels corresponding to the two ends of a session. Giunti and Vasconcelos [2010] address the problem in yet another way: they define a linear type system where session channels do not have polarities, and they enrich the type language with types of the form (σ, τ), so that a judgment $c : (\sigma, \tau) \vdash P$ denotes the fact that P owns *both ends* of the session identified by c, and the two ends are used according to the session types σ and τ. Then, a typing derivation such as

$$\frac{c : \sigma \vdash P \qquad c : \tau \vdash Q}{c : (\sigma, \tau) \vdash P \mid Q}$$

indicates that the two ends are owned by the two sub-processes P and Q. There is a strong similarity between the type (σ, τ) and the session type $\sigma \mid \tau$ in the present work. In fact, just as we require the session type of a restricted channel to be complete (rule (T-RES)), they require the two components of (σ, τ) to be dual. The difference is that the hypothesis $c : (\sigma, \tau)$ does not necessarily imply that the two ends are owned by *independent* processes running in parallel. For example, if in the system $P \mid Q$ above P sends c to Q, the system evolves to some $P' \mid Q'$ which is typed thus:

$$\frac{\vdash P' \qquad c : (\sigma', \tau') \vdash Q'}{c : (\sigma', \tau') \vdash P' \mid Q'}$$

for some appropriate session types σ' and τ'. Basically, the typing records the fact that now both ends of the session are owned by the sole process Q', which however may use c according to some sequential interleaving of σ' and τ' semantically different from $\sigma' \mid \tau'$. Again, this approach does not guarantee progress in the sense of Theorem 4.2. In the end the stratification we impose is original to the best of our knowledge, in that it solves the aliasing problem with a static type system (no runtime check is required) and it does not rely on polarities nor on linear usage of channels. It is not obvious whether this approach is more or less restrictive when compared against related ones; the topic should be further investigated. Other session type theories such as those developed in [Dezani-Ciancaglini *et al.*, 2008; Bettini *et al.*, 2008] impose an ordering on channels for ensuring global progress of well-typed systems. In these works the channel order determines the *temporal order* in which different channels can be used, whereas in our case the order determines a hierarchy of channels such that "smaller" channels can be sent/received over "larger" ones. In this sense our channel order resembles the nesting of arrows in the types of higher-order functions.

In recent years there has been considerable work on so-called *contracts* for distributed processes and Web services [Castagna *et al.*, 2009b; Castagna and Padovani, 2009]. Unlike session types, contracts capture the overall behavior of a process, including the order of communications occurring on different channels. For this reason, their behavioral nature has been explicitly recognized since the very beginning and their theories have been developed as refinements and extensions of existing process algebraic ones. Even for contracts the shift from dyadic theories, where only client/server interactions are considered, to multi-party theories has required the adoption of fair behavioral equivalences [Bravetti and Zavattaro, 2007a,b, 2009]. Laneve and Padovani [2008] show semantic-preserving, mutual encodings between contracts and session

types in the dyadic setting. Here we further clarify the relationship between the two formalisms in the general setting by pursuing the idea of projected behavior. In particular, we argue that the projection described in Section 4 would seamlessly work if applied to the contracts in [Castagna and Padovani, 2009] which capture a very shallow abstraction of processes.

The use of processes as types has already been proposed in ways unrelated to sessions and session types, for example by Chaki *et al.* [2002] and Nielson and Nielson [1994]. In particular, Nielson and Nielson [1994] use a language close to value-passing CCS for defining an effect system for Concurrent ML. Sumii and Kobayashi [1998] and Kobayashi *et al.* [2000] describe a type system for guaranteeing deadlock freedom of π-calculus processes. In their proposal channels are associated with types of the form $[t]/U$ where t is the type of messages transmitted over the channel while U is a term of a simple process algebra called *usage* which specifies the modalities (input/output) in which the channel is used by a process. As in our approach, usages can be composed in parallel to express the simultaneous access to a channel by several processes. In fact, this type system can be seen as projecting processes into usages. The main difference is that channels are only allowed to transmit one type of message, while in our case the same channel can be used for transmitting messages of different types. Also, usages are equipped with a subusage relation, but this is axiomatically defined whereas our subsession relation arises from completeness. Igarashi and Kobayashi [2004] propose a radically different (and more expressive) application of the "types-as-processes" paradigm, where π-calculus processes (*not* channels) are associated with *process types* which are expressed as CCS-like terms. Unlike [Sumii and Kobayashi, 1998; Kobayashi *et al.*, 2000] and session type theories including the one proposed in the present paper, the process type associated with a process captures the order in which communications on different channels occur while the actual content of messages is approximated into types. In this respect, process types are much more similar to contracts proposed by Castagna and Padovani [2009], except that contracts keep track of delegated channels for ensuring global progress.

In the present work we have been advocating a *behavioral* approach for the formalization of session types, but *logical* and *type-theoretic* approaches have also been investigated. In the logical approach Caires and Pfenning [2010] interpret session types as formulas of a suitable fragment of linear logic: internal and external choices are modeled as additive disjunction $\oplus$ and additive conjunction $\&$ respectively. In this view input actions are modeled as the linear implication $A \multimap B$, output actions are modeled as the tensor product $A \otimes B$, and exponentials are used for denoting non-linear resources, such as shared channels. Remarkably, the seminal work by Honda [1993] on session types was clearly inspired by linear logic, as witnessed by the use of the symbols $\oplus$ and $\&$ for denoting branching points. The type-theoretic approach [Padovani, 2010] is based on the observation that internal and external choices can be modeled as intersection and union types, respectively. For example, a channel typed by $!\texttt{int}.\sigma \wedge !\texttt{bool}.\tau$ can be used by a process that internally decides whether to send an integer or a boolean value, and then behaves according to σ or τ, respectively. This is equivalent to saying that the channel has *both* type $!\texttt{int}.\sigma$ and type $!\texttt{bool}.\tau$. Conversely, a channel typed by $?\texttt{int}.\sigma \vee ?\texttt{bool}.\tau$ has an "undefined" type: it can have either type $?\texttt{int}.\sigma$ or type $?\texttt{bool}.\tau$. The process will figure out the exact type of the channel from the type (either `int` or `bool`) of the first message received from it.

6. Conclusion

It is obvious that session types are behavioral types. Still, session types are normally associated with channels and channels do not expose any behavior. These apparently inconsistent facts can be reconciled by observing that the session type associated with a channel is the projection of the behavior of a *process* restricted to the input/output operations that the process performs on that channel. Not surprisingly the session types defined in this way are more general – one might say more complicated – than the ones we usually encounter in other works. If only because, when two parallel sub-processes access the same channel, its associated session type will be the parallel composition of the session types obtained by projecting the two sub-processes separately. Nonetheless, by taking this alternative point of view we have been able to define a theory of session types that generalizes, clarifies, 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 close to value-passing CCS; *completeness* expresses the property that a session is well-formed and never yields a communication error; *duality* [Gay and Hole, 2005] $\sigma \bowtie \tau$ is the special case where $\sigma \mid \tau$ is complete; *viability* characterizes well-typed process and corresponds to the concept of inhabited (session) type; the *subtyping relation* between session types arises semantically by relating those session types that preserve completeness in arbitrary contexts.

We envision two main directions in which this work can be further developed. First of all, the semantics of session types naturally calls for some fairness property when moving from the dyadic to the multi-party scenario. Although the fair subsession relation is difficult to characterize in general, it may be interesting to investigate its relationship with global types of multi-party session type theories, from which session types are projected. Second, it would be interesting to provide a general session type inference algorithm for processes along the lines of the projection we have proposed. The aspects of the type system defined in Section 4 that make it non-algorithmic regard the application of the subsumption rule (T-SUB), the inference of channel orders, and above all the computation of the delegated behavior ρ in rule (T-OUTPUTS). The first two aspects seem to be addressable by means of standard techniques (in fact, while describing the typing rules we have already hinted at the critical places where subsumption is actually necessary). As regards the delegated behavior in rule (T-OUTPUTS), we conjecture that it can be inferred (in closed systems) by means of a standard unification algorithm exploiting axiom (CH3). Nonetheless, the combination of these aspects with the fact that we work with possibly infinite types makes the design of an inference algorithm quite challenging.

Acknowledgments. I wish to thank the anonymous referees for their detailed and thoughtful comments on early versions of this paper. I would also like to thank Giuseppe Castagna, Mariangiola Dezani, Kohei Honda, and Nobuko Yoshida for the insightful discussions.

References

Lucia Acciai and Michele Boreale. A type system for client progress in a service-oriented calculus. In *Concurrency, Graphs and Models: Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday*, LNCS 5065, pages 642–658. Springer, 2008.

Lorenzo Bettini, Mario Coppo, Loris D'Antoni, Marco De Luca, Mariangiola Dezani-

Ciancaglini, and Nobuko Yoshida. Global Progress in Dynamically Interleaved Multiparty Sessions. In *Proceedings of CONCUR'08*, LNCS 5201, pages 418–433. Springer, 2008.

Mario Bravetti and Gianluigi Zavattaro. Contract based multi-party service composition. In *Proceedings of FSEN'07*, LNCS 4767, pages 207–222. Springer, 2007.

Mario Bravetti and Gianluigi Zavattaro. Towards a unifying theory for choreography conformance and contract compliance. In *Proceedings of the 6th International Symposium on Software Composition*, LNCS 4829, pages 34–50. Springer, 2007.

Mario Bravetti and Gianluigi Zavattaro. A foundational theory of contracts for multi-party service composition. *Fundamenta Informaticae*, 89(4):451–478, 2009.

Michele Bugliesi, Damiano Macedonio, Luca Pino, and Sabina Rossi. Compliance preorders for Web Services. In *Proceedings of WS-FM'09*. Springer, 2010. To appear.

Luis Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *Proceedings of CONCUR'10*. Springer, 2010. To appear.

Luis Caires and Hugo Vieira. Conversation types. In *Proceedings of ESOP'09*, LNCS 5502, pages 285–300. Springer, 2009.

Sara Capecchi, Mario Coppo, Mariangiola Dezani-Ciancaglini, Sophia Drossopoulou, and Elena Giachino. Amalgamating Sessions and Methods in Object Oriented Languages with Generics. *Theoretical Computer Science*, 410:142–167, 2009.

Samuele Carpineti, Cosimo Laneve, and Luca Padovani. PiDuce - A project for experimenting Web services technologies. *Science of Computer Programming*, 74(10):777–811, 2009.

Giuseppe Castagna and Alain Frisch. A gentle introduction to semantic subtyping. In *Proceedings of PPDP'05*, pages 198–199. ACM, 2005.

Giuseppe Castagna and Luca Padovani. Contracts for mobile processes. In *Proceedings of CONCUR'09*, LNCS 5710, pages 211–228. Springer, 2009.

Giuseppe Castagna, Rocco De Nicola, and Daniele Varacca. Semantic subtyping for the pi-calculus. *Theoretical Computer Science*, 398(1-3):217–242, 2008.

Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, Elena Giachino, and Luca Padovani. Foundations of session types. In *Proceedings of PPDP'09*, pages 219–230. ACM, 2009.

Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A theory of contracts for Web services. *ACM Transactions on Programming Languages and Systems*, 31(5):1–61, 2009.

Sagar Chaki, Sriram K. Rajamani, and Jakob Rehof. Types as models: model checking message-passing programs. *SIGPLAN Notices*, 37(1):45–57, 2002.

Bruno Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25:95–169, 1983.

Rocco De Nicola and Matthew Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

Rocco De Nicola and Matthew Hennessy. CCS without τ's. In *Proceedings of TAPSOFT'87/CAAP'87*, LNCS 249, pages 138–152. Springer, 1987.

Mariangiola Dezani-Ciancaglini, Ugo de' Liguoro, and Nobuko Yoshida. On Progress for Structured Communications. In *Proceedings of TGC'07*, LNCS 4912, pages 257–275. Springer, 2008.

Mariangiola Dezani-Ciancaglini, Sophia Drossopoulou, Dimitris Mostrous, and Nobuko Yoshida. Session Types for Object-Oriented Languages. *Information and Computation*, 207(5):595–641, 2009.

Sophia Drossopoulou, Mariangiola Dezani-Ciancaglini, and Mario Coppo. Amalgamating the Session Types and the Object Oriented Programming Paradigms. In *Proceedings of MPOOL'07*, 2007.

Alain Frisch, Giuseppe Castagna, and Veronique Benzaken. Semantic subtyping: dealing set-theoretically with function, union, intersection, and negation types. *Journal of the ACM*, 55(4):1–64, 2008.

Simon Gay and Malcolm Hole. Subtyping for session types in the π-calculus. *Acta Informatica*, 42(2-3):191–225, 2005.

Simon Gay and Vasco T. Vasconcelos. Asynchronous functional session types. Technical Report 2007–251, Department of Computing, University of Glasgow, 2007.

Marco Giunti and Vasco T. Vasconcelos. A linear account of session types in the pi calculus. In *Proceedings of CONCUR'10*. Springer, 2010. To appear.

Matthew Hennessy. *Algebraic Theory of Processes*. Foundation of Computing. MIT Press, 1988.

Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language primitives and type disciplines for structured communication-based programming. In *Proceedings of ESOP'98*, LNCS 1381, pages 122–138. Springer, 1998.

Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *Proceedings of POPL'08*, pages 273–284. ACM, 2008.

Kohei Honda. Types for dyadic interaction. In *Proceedings of CONCUR'93*, LNCS 715, pages 509–523. Springer, 1993.

Atsushi Igarashi and Naoki Kobayashi. A generic type system for the Pi-calculus. *Theoretical Computer Science*, 311(1-3):121–163, 2004.

Naoki Kobayashi, Shin Saito, and Eijiro Sumii. An implicitly-typed deadlock-free process calculus. In *Proceedings of CONCUR'00*, LNCS 1877, pages 489–503. Springer, 2000.

Cosimo Laneve and Luca Padovani. The pairing of contracts and session types. In *Concurrency, Graphs and Models: Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday*, LNCS 5065, pages 681–700. Springer, 2008.

Robin Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1999.

V. Natarajan and Rance Cleaveland. Divergence and fair testing. In *Proceedings of ICALP '95*, LNCS 944, pages 648–659. Springer, 1995.

Hanne Riis Nielson and Flemming Nielson. Higher-order concurrent programs with finite communication topology (extended abstract). In *Proceedings of POPL'94*, pages 84–97. ACM, 1994.

Luca Padovani. Session types at the mirror. *EPTCS*, 12:71–86, 2009.

Luca Padovani. Session Types = Intersection Types + Union Types. In *Proceedings of ITRS'10*, 2010. To appear.

Benjamin Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, pages 376–385, 1996.

Arend Rensink and Walter Vogler. Fair testing. *Information and Computation*, 205(2):125–198, 2007.

Davide Sangiorgi and David Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

Eijiro Sumii and Naoki Kobayashi. A generalized deadlock-free process calculus. *Electronic Notes in Theoretical Computer Science*, 16(3):225–247, 1998.

Vasco T. Vasconcelos, Simon Gay, and Antonio Ravara. Type checking a multithreaded functional language with session types. *Theoretical Computer Science*, 368:64–87, 2006.

Vasco T. Vasconcelos. Fundamentals of session types. In *SFM'09*, LNCS 5569, pages 158–186. Springer, 2009.

Nobuko Yoshida and Vasco T. Vasconcelos. Language primitives and type discipline for structured communication-based programming revisited: Two systems for higher-order session communication. *Electronic Notes in Theoretical Computer Science*, 171(4):73–93, 2007.

Appendix A. Supplement to Section 3

Theorem A.1 (Theorem 3.1). Let σ and τ be components. Then $\sigma \preceq \tau$ if and only if:

1 $\tau \Downarrow \mathrm{S}$ implies $\sigma \Downarrow \mathrm{R}$ and $\mathrm{R} \subseteq \mathrm{S}$;
2 σ viable and $\tau \overset{\lfloor\mu\rfloor}{\Longrightarrow}$ implies $\sigma \overset{\lfloor\mu\rfloor}{\Longrightarrow}$ and $\sigma(\mu) \preceq \tau(\mu)$ for every $\mu \neq \checkmark$.

Proof. ("if" part) Let ρ be a session types that completes σ and consider a derivation $\rho \mid \tau \Longrightarrow \rho' \mid \tau'$. We must prove $\rho' \mid \tau'$ live. By unzipping the derivation we obtain a sequence φ of actions such that $\rho \overset{\overline{\varphi}}{\Longrightarrow} \rho'$ and $\tau \overset{\varphi}{\Longrightarrow} \tau'$. We distinguish two cases depending on the shape of φ.

($\varphi = \varepsilon$) From condition (1) we deduce that there exists σ' such that $\sigma \Longrightarrow \sigma'$ and $\{\lfloor\mu\rfloor \in \lfloor\mathsf{init}(\sigma')\rfloor \mid \sigma(\mu) \text{ viable}\} \subseteq \{\lfloor\mu\rfloor \in \lfloor\mathsf{init}(\tau')\rfloor \mid \tau(\mu) \text{ viable}\}$.
As regards the τ' component, from the hypothesis $\rho \mid \sigma$ complete we deduce $\rho' \mid \sigma'$ complete, in particular either $\checkmark \in \mathsf{init}(\sigma')$ or $\overline{\mathsf{init}(\rho')} \cap \mathsf{init}(\sigma') \neq \emptyset$. We conclude either $\checkmark \in \mathsf{init}(\tau')$ or $\overline{\mathsf{init}(\rho')} \cap \mathsf{init}(\tau') \neq \emptyset$ because $\{\lfloor\mu\rfloor \in \lfloor\mathsf{init}(\sigma')\rfloor \mid \rho' \overset{\lfloor\overline{\mu}\rfloor}{\Longrightarrow}\} \subseteq \{\lfloor\mu\rfloor \in \lfloor\mathsf{init}(\tau')\rfloor \mid \rho' \overset{\lfloor\overline{\mu}\rfloor}{\Longrightarrow}\}$.
As regards ρ' let $\rho' \cong \rho_1 \mid \rho_2$ where ρ_1 is a component and suppose $\checkmark \notin \mathsf{init}(\rho_1)$ for otherwise there is nothing to prove. We have $\rho' \mid \tau' \cong \rho_1 \mid (\rho_2 \mid \tau')$ and we distinguish two sub-cases:

($\overline{\mathsf{init}(\rho_2)} \cap \mathsf{init}(\tau') = \emptyset$) Then $\overline{\mathsf{init}(\rho_2)} \cap \mathsf{init}(\sigma') = \emptyset$ and from the hypothesis $\rho \mid \sigma$ complete we conclude $\emptyset \neq \overline{\mathsf{init}(\rho_1)} \cap \mathsf{init}(\rho_2 \mid \sigma') = \overline{\mathsf{init}(\rho_1)} \cap (\mathsf{init}(\rho_2) \cup \mathsf{init}(\sigma')) \subseteq \overline{\mathsf{init}(\rho_1)} \cap (\mathsf{init}(\rho_2) \cup \mathsf{init}(\tau')) = \overline{\mathsf{init}(\rho_1)} \cap \mathsf{init}(\rho_2 \mid \tau')$.

($\overline{\mathsf{init}(\rho_2)} \cap \mathsf{init}(\tau') \neq \emptyset$) Then there exist ρ_2', τ'', and μ such that $\rho_2 \overset{\overline{\mu}}{\Longrightarrow} \rho_2'$ and $\tau' \overset{\mu}{\Longrightarrow} \tau''$. From condition (2) we deduce $\sigma \overset{\lfloor\mu\rfloor}{\Longrightarrow}$ and $\sigma(\mu) \preceq \tau(\mu)$, hence $\rho_1 \mid (\rho_2' \mid \tau'')$ is complete. This means $\overline{\mathsf{init}(\rho_1)} \cap \mathsf{init}(\rho_2' \mid \tau'') \neq \emptyset$ and we conclude since $\mathsf{init}(\rho_2' \mid \tau'') \subseteq \mathsf{init}(\rho_2 \mid \tau')$.

($\varphi = \mu\varphi'$) Then $\rho \overset{\overline{\mu}}{\Longrightarrow} \rho'' \overset{\overline{\varphi}'}{\Longrightarrow} \rho'$ and $\tau \overset{\mu}{\Longrightarrow} \tau'' \overset{\varphi}{\Longrightarrow} \tau'$ for some ρ'' and τ''. From the hypothesis $\rho \mid \sigma$ complete we deduce σ viable. From condition (2) we deduce $\sigma \overset{\lfloor\mu\rfloor}{\Longrightarrow}$ and $\sigma(\mu) \preceq \tau(\mu)$ and $\rho'' \mid \sigma(\mu)$ complete, hence $\rho'' \mid \tau(\mu)$ complete. We conclude $\rho' \mid \tau'$ live since $\tau(\mu) \Longrightarrow \tau''$.

("only if" part) As regards condition (1), suppose by contradiction that there exists S such that $\tau \Downarrow \mathrm{S}$ and $\sigma \Downarrow \mathrm{R}$ implies $\mathrm{R} \not\subseteq \mathrm{S}$. Let $\mathrm{T} \overset{\text{def}}{=} \bigcup_{\sigma \Downarrow \mathrm{R}} \mathrm{R} \setminus \mathrm{S}$; let $C(\mu) \overset{\text{def}}{=} \{\sigma' \mid \sigma \Longrightarrow \overset{\mu}{\longrightarrow} \sigma'\}$; let $\mathrm{X} = \{\dagger_1 t_1, \ldots, \dagger_n t_n\}$ be a finite set such that:

1 $\dagger\mathsf{v} \in \mathsf{init}(\sigma)$ implies $\mathsf{v} \in t$ and $\dagger t \in \mathrm{X}$ for some t;
2 for every $\dagger t \in \mathrm{X}$ there exists $\mathsf{v} \in t$ such that $C(\dagger\mathsf{v}) \neq \emptyset$;
3 for every $\dagger t \in \mathrm{X}$ and for every $\mathsf{v}, \mathsf{w} \in t$ we have $C(\dagger\mathsf{v}) = C(\dagger\mathsf{w})$.

Observe that X always exists because the set $\{C(\dagger \mathsf{v}) \mid \dagger \in \{?,!\}, \mathsf{v} \in \mathscr{V}\}$ is a finite set since $\{\sigma' \mid \exists \mu : \sigma \Longrightarrow \xrightarrow{\mu} \sigma'\}$ is a subset of the set of σ's subtrees and the set of distinct σ's subtrees is finite by regularity of σ. Now let

$$\rho \stackrel{\text{def}}{=} \sum_{\dagger t \in \mathrm{X}} \overline{\dagger} t.\rho_{\dagger t} \{+ ?\theta.\rho_{!\diamond}\}_{!\diamond \in \mathrm{T}} \{+ !\mathbf{0}.\rho_{?\diamond}\}_{?\diamond \in \mathrm{T}} \{+ \mathbf{1}\}_{\checkmark \in \mathrm{T}} \tag{3}$$

where

— $\rho_{\dagger t}$ completes $\sigma(\dagger \mathsf{v})$ for every $\mathsf{v} \in t$;
— $\rho_{!\diamond}$ completes $\sigma(!\theta)$;
— $\rho_{?\diamond}$ completes $\sigma(?\mathbf{0})$

and the existence of θ is granted from the definition of ready set. By definition of ρ we have that $\rho \mid \sigma$ is complete while $\rho \mid \tau$ is not, which contradicts the hypothesis $\sigma \preceq \tau$.

As regards condition (2), assume σ viable and $\tau \stackrel{\lfloor \mu \rfloor}{\Longrightarrow}$ for some $\mu \neq \checkmark$ and suppose by contradiction that $\sigma \stackrel{\lfloor \mu \rfloor}{\not\Longrightarrow}$. Let $\rho \stackrel{\text{def}}{=} \rho' + \overline{\mu}.\mathbf{0}$ where ρ' is some arbitrary session type that completes σ. Without loss of generality we may assume that ρ' is a component. Indeed, if σ is viable we can build a component that completes it similar to that defined in (3) above. Then ρ completes σ but not τ, which is absurd. Hence $\sigma \stackrel{\lfloor \mu \rfloor}{\Longrightarrow}$. If $\sigma(\mu)$ is not viable there is nothing left to prove. If $\sigma(\mu)$ is viable, then consider an arbitrary ρ'' that completes it and let $\rho \stackrel{\text{def}}{=} \rho' + \overline{\mu}.\rho''$. From $\sigma \mid \rho$ complete and the hypothesis $\sigma \preceq \tau$ we deduce that $\rho'' \mid \tau(\mu)$ is complete. We conclude $\sigma(\mu) \preceq \tau(\mu)$ since ρ'' is arbitrary. □

Theorem A.2 (Theorem 3.3). Let σ and τ be components. Then $\sigma \sqsubseteq \tau$ if and only if:

1 $\tau \Downarrow \mathrm{S}$ implies $\sigma \Downarrow \mathrm{R}$ and $\mathrm{R} \subseteq \mathrm{S}$;
2 $\tau \stackrel{\lfloor \mu \rfloor}{\Longrightarrow}$ implies $\sigma \stackrel{\lfloor \mu \rfloor}{\Longrightarrow}$ and $\sigma(\mu) \preceq \tau(\mu)$ for every $\mu \neq \checkmark$.

Proof. ("if" part) It is enough to show that $\sigma + \rho$ and $\tau + \rho$ satisfy the conditions (1) and (2) of Theorem 3.1 for an arbitrary component ρ.

As regards condition (1), let $\tau + \rho \Downarrow \mathrm{S}$. By definition of ready set there exist τ' and ρ' such that $\tau \Longrightarrow \tau'$ and $\rho \Longrightarrow \rho'$ and $\mathrm{S} = \{\lfloor \mu \rfloor \in \lfloor \mathsf{init}(\tau' + \rho') \rfloor \mid (\tau + \rho)(\mu) \text{ viable}\}$. Let $\mathrm{S}' \stackrel{\text{def}}{=} \{\lfloor \mu \rfloor \in \lfloor \mathsf{init}(\tau') \rfloor \mid \tau(\mu) \text{ viable}\}$. From hypothesis (1) we deduce $\sigma \Downarrow \mathrm{R}'$ for some $\mathrm{R}' \subseteq \mathrm{S}'$. By definition of ready set we deduce that there exists σ' such that $\sigma \Longrightarrow \sigma'$ and $\mathrm{R}' = \{\lfloor \mu \rfloor \in \lfloor \mathsf{init}(\sigma') \rfloor \mid \sigma(\mu) \text{ viable}\}$. Let $\mathrm{R} \stackrel{\text{def}}{=} \{\lfloor \mu \rfloor \in \lfloor \mathsf{init}(\sigma' + \rho') \rfloor \mid (\sigma + \rho)(\mu) \text{ viable}\}$. To prove $\mathrm{R} \subseteq \mathrm{S}$, assume $\lfloor \mu \rfloor \in \mathrm{R}$ and observe that this implies $(\sigma + \rho)(\mu)$ viable. If we show that $\lfloor \mu \rfloor \in \lfloor \mathsf{init}(\tau' + \rho') \rfloor$ we are done, because $\preceq$ is a pre-congruence for $\oplus$ hence $(\tau + \rho)(\mu)$ is viable and we conclude $\lfloor \mu \rfloor \in \mathrm{S}$. The only interesting case is when $\lfloor \mu \rfloor \in \lfloor \mathsf{init}(\sigma') \setminus \mathsf{init}(\rho') \rfloor$. From $(\sigma + \rho)(\mu)$ viable we deduce $\sigma(\mu)$ viable, hence $\lfloor \mu \rfloor \in \mathrm{R}' \subseteq \mathrm{S}'$, which implies $\lfloor \mu \rfloor \in \lfloor \mathsf{init}(\tau') \rfloor$.

As regards condition (2) of Theorem 3.1 suppose $\sigma + \rho$ viable and let $\tau + \rho \stackrel{\lfloor \mu \rfloor}{\Longrightarrow}$ for some $\mu \neq \checkmark$. From hypothesis (2) we deduce $\sigma + \rho \stackrel{\lfloor \mu \rfloor}{\Longrightarrow}$ and we conclude $(\sigma + \rho)(\mu) \preceq (\tau + \rho)(\mu)$ because of pre-congruence of $\preceq$ with respect to $\oplus$.

("only if" part) Since $\sigma \sqsubseteq \tau$ implies $\sigma \preceq \tau$ condition (1) follows immediately from Theorem 3.1. From the hypothesis $\sigma \sqsubseteq \tau$ we also deduce $\sigma + \mathbf{1} \preceq \tau + \mathbf{1}$. Observe that $\sigma + \mathbf{1}$ is viable. Suppose $\tau + \mathbf{1} \stackrel{\lfloor \mu \rfloor}{\Longrightarrow}$ for some $\mu \neq \checkmark$. From Theorem 3.1 we deduce $\sigma + \mathbf{1} \stackrel{\lfloor \mu \rfloor}{\Longrightarrow}$ and $(\sigma + \mathbf{1})(\mu) \preceq$

$(\tau+\mathbf{1})(\mu)$. We conclude $\sigma \overset{\lfloor\mu\rfloor}{\Longrightarrow}$ and $\sigma(\mu) \preceq \tau(\mu)$ since $\sigma(\mu) = (\sigma+\mathbf{1})(\mu) \preceq (\tau+\mathbf{1})(\mu) = \tau(\mu)$. □

Appendix B. Supplement to Section 4

In the type system and the proofs that follow we sometimes implicitly use some properties of channel orders that we collect below:

Proposition B.1. Let $\mathscr{C}$ be a channel order. Then the following properties hold:

1 if $a \neq c$, then $\mathscr{C}\{^c/_x\} \setminus a = (\mathscr{C} \setminus a)\{^c/_x\}$;
2 if $\mathscr{C} \vdash v \prec u$ does not hold, then $\mathscr{C}, u \prec v$ is a channel order;
3 if neither $\mathscr{C} \vdash c \prec x$ nor $\mathscr{C} \vdash x \prec c$ do hold, then $\mathscr{C}\{^c/_x\}$ is a channel order;
4 if $a \neq c$ and $(\mathscr{C} \setminus a)\{^c/_x\}$ is a channel order, then $\mathscr{C}\{^c/_x\}$ is a channel order.

Proof. We prove only item (1), the remaining items are easy exercises.

— ("only if" part) Suppose $(u,v) \in \mathscr{C}\{^c/_x\} \setminus a$. Then $u, v \neq a$ and $(u',v') \in \mathscr{C}$ for some u', v' such that $u = u'\{^c/_x\}$ and $v = v'\{^c/_x\}$. We deduce $u', v' \neq a$, hence $(u',v') \in \mathscr{C} \setminus a$. We conclude $(u,v) \in (\mathscr{C} \setminus a)\{^c/_x\}$.
— ("if" part) Suppose $(u,v) \in (\mathscr{C} \setminus a)\{^c/_x\}$. Then $u = u'\{^c/_x\}$ and $v = v'\{^c/_x\}$ for some u', v' such that $(u',v') \in \mathscr{C} \setminus a$. We deduce $u', v' \neq a$ and $(u',v') \in \mathscr{C}$, namely $(u,v) \in \mathscr{C}\{^c/_x\}$. We conclude $(u,v) \in \mathscr{C}\{^c/_x\} \setminus a$ because $u, v \neq a$. □

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

1 if $\Gamma, x : t; \mathscr{C} \vdash P : \Delta$ and $\mathsf{v} \in t$, then $\Gamma; \mathscr{C} \vdash P\{^{\mathsf{v}}/_x\} : \Delta$;
2 if $\Gamma; \mathscr{C} \vdash P : \Delta \cup \{x : \tau\}$ and $c \notin \mathtt{dom}(\Delta)$, then $\Gamma; \mathscr{C}\{^c/_x\} \vdash P\{^c/_x\} : \Delta \cup \{c : \tau\}$.

Proof. Item (1) follows immediately from the assumption that $(\mathtt{dom}(\Gamma) \cup \{x\}) \cap \mathtt{dom}(\Delta) = \emptyset$ and by assuming that the substitution lemma holds for the language of expressions e over which we have parametrized our language of processes. As regards item (2), we do an induction on P. Most cases are obvious so we only show here the interesting ones.

Suppose $P = u?(y).Q$ where we may assume, without loss of generality, that $x \neq y$. We distinguish two subcases:

($x = u$) Then $\Delta \cup \{x : \tau\} \preceq \Delta' \cup \{x : ?\rho.\sigma\}$ where **(1)** $\Gamma; \mathscr{C}, y \prec x \vdash Q : \Delta' \cup \{x : \sigma, y : \rho\}$ and **(2)** $\mathscr{C} \vdash x \prec \mathtt{dom}(\Delta')$ and **(3)** $\mathtt{dom}(\Delta') \cup \{x\} \subseteq \mathtt{dom}(\Delta) \cup \{x\}$. From the hypothesis $c \notin \mathtt{dom}(\Delta)$ and **(3)** we deduce $c \notin \mathtt{dom}(\Delta')$. From **(1)** and the induction hypothesis we have $\Gamma; \mathscr{C}\{^c/_x\}, y \prec c \vdash Q\{^c/_x\} : \Delta' \cup \{c : \sigma, y : \rho\}$. From **(2)** we deduce $\mathscr{C}\{^c/_x\} \vdash c \prec \mathtt{dom}(\Delta')$. By (T-INPUTS) we derive $\Gamma; \mathscr{C}\{^c/_x\} \vdash c?(y).Q\{^c/_x\} : \Delta' \cup \{c : ?\rho.\sigma\}$ and we conclude with an application of (T-SUB).

($x \neq u$) Then $\Delta \cup \{x : \tau\} \preceq \Delta' \cup \{x : \theta, u : ?\rho.\sigma\}$ where **(1)** $\Gamma; \mathscr{C}, y \prec u \vdash Q : \Delta' \cup \{x : \theta, u : \sigma, y : \rho\}$ and **(2)** $\mathscr{C} \vdash u \prec \mathtt{dom}(\Delta') \cup \{x\}$ and **(3)** $\mathtt{dom}(\Delta') \cup \{x, u\} \subseteq \mathtt{dom}(\Delta) \cup \{x\}$. From the hypothesis $c \notin \mathtt{dom}(\Delta)$ and **(3)** we deduce $c \notin \mathtt{dom}(\Delta') \cup \{u\}$. From **(1)** and the induction hypothesis we derive $\Gamma; \mathscr{C}\{^c/_x\}, y \prec u \vdash Q\{^c/_x\} : \Delta' \cup \{c : \theta, u : \sigma, y : \rho\}$. From **(2)** we deduce $\mathscr{C}\{^c/_x\} \vdash u \prec \mathtt{dom}(\Delta') \cup \{c\}$. By (T-INPUTS) we derive $\Gamma; \mathscr{C}\{^c/_x\} \vdash u?(y).Q\{^c/_x\} : \Delta' \cup \{c : \theta, u : ?\rho.\sigma\}$ and we conclude with an application of (T-SUB).

Suppose $P = u!v.Q$. We distinguish three subcases:

($x = u$) Then $\Delta \cup \{x : \tau\} \preceq \Delta' \cup \{x : !\rho.\sigma, v : \rho \mid \theta\}$ where **(1)** $\Gamma; \mathscr{C} \vdash Q : \Delta' \cup \{x : \sigma, v : \theta\}$ and **(2)** $\mathscr{C} \vdash v \prec x$ and **(3)** $\mathtt{dom}(\Delta') \cup \{x, v\} \subseteq \mathtt{dom}(\Delta) \cup \{x\}$. From the hypothesis $c \notin \mathtt{dom}(\Delta)$ and **(3)** we deduce $x \notin \mathtt{dom}(\Delta') \cup \{v\}$. From **(1)** and the induction hypothesis we derive $\Gamma; \mathscr{C}\{^c/_x\} \vdash Q\{^c/_x\} : \Delta' \cup \{c : \sigma, v : \theta\}$. From **(2)** we deduce $\mathscr{C}\{^c/_x\} \vdash v \prec c$. By (T-OUTPUTS) we derive $\Gamma; \mathscr{C}\{^c/_x\} \vdash c!v.Q\{^c/_x\} : \Delta' \cup \{c : !\rho.\sigma, v : \rho \mid \theta\}$ and we conclude with an application of (T-SUB).

($x = v$) Then $\Delta \cup \{x : \tau\} \preceq \Delta' \cup \{u : !\rho.\sigma, x : \rho \mid \theta\}$ where **(1)** $\Gamma; \mathscr{C} \vdash Q : \Delta' \cup \{u : \sigma, x : \theta\}$ and **(2)** $\mathscr{C} \vdash x \prec u$ and **(3)** $\mathtt{dom}(\Delta') \cup \{u, x\} \subseteq \mathtt{dom}(\Delta) \cup \{x\}$. From the hypothesis $c \notin \mathtt{dom}(\Delta)$ and **(3)** we deduce $c \notin \mathtt{dom}(\Delta') \cup \{u\}$. From **(1)** and the induction hypothesis we derive $\Gamma; \mathscr{C}\{^c/_x\} \vdash Q\{^c/_x\} : \Delta' \cup \{u : \sigma, c : \theta\}$. From **(2)** we deduce $\mathscr{C}\{^c/_x\} \vdash c \prec u$. By (T-OUTPUTS) we derive $\Gamma; \mathscr{C}\{^c/_x\} \vdash u!c.Q\{^c/_x\} : \Delta' \cup \{u : !\rho.\sigma, c : \rho \mid \theta\}$ and we conclude with an application of (T-SUB).

($x \neq u, v$) Then $\Delta \cup \{x : \tau\} \preceq \Delta' \cup \{x : \theta, u : !\rho.\sigma, v : \rho \mid \theta'\}$ where **(1)** $\Gamma; \mathscr{C} \vdash Q : \Delta' \cup \{x : \theta, u : \sigma, v : \theta'\}$ and **(2)** $\mathscr{C} \vdash v \prec u$ and **(3)** $\mathtt{dom}(\Delta') \cup \{x, u, v\} \subseteq \mathtt{dom}(\Delta) \cup \{x\}$. From the hypothesis $c \notin \mathtt{dom}(\Delta)$ and **(3)** we deduce $c \notin \mathtt{dom}(\Delta') \cup \{u, v\}$. From **(1)** and the induction hypothesis we derive $\Gamma; \mathscr{C}\{^c/_x\} \vdash Q\{^c/_x\} : \Delta' \cup \{c : \theta, u : \sigma, v : \theta'\}$. From **(2)** and by (T-OUTPUTS) we derive $\Gamma; \mathscr{C}\{^c/_x\} \vdash u!v.Q\{^c/_x\} : \Delta' \cup \{c : \theta, u : !\rho.\sigma, v : \rho \mid \theta'\}$ and we conclude with an application of (T-SUB).

Suppose $P = (\mathsf{new}\ a)Q$ where we may assume, without loss of generality, that $a \neq c$. Then $\mathscr{C} = \mathscr{C}' \setminus a$ and $\Delta \cup \{x : \tau\} \preceq \Delta' \cup \{x : \theta\}$ where **(1)** $\Gamma; \mathscr{C}' \vdash Q : \Delta' \cup \{x : \theta, a : \theta'\}$ and **(2)** θ' is complete and **(3)** $\mathtt{dom}(\Delta') \cup \{x\} \subseteq \mathtt{dom}(\Delta) \cup \{x\}$. From the hypothesis $c \notin \mathtt{dom}(\Delta)$ and **(3)** and $a \neq c$ we deduce $c \notin \mathtt{dom}(\Delta') \cup \{a\}$. From **(1)** and the induction hypothesis we derive $\Gamma; \mathscr{C}'\{^c/_x\} \vdash Q\{^c/_x\} : \Delta' \cup \{c : \theta, a : \theta'\}$. From **(2)** and by (T-RES) we derive $\Gamma; \mathscr{C}\{^c/_x\} \vdash Q\{^c/_x\} : \Delta' \cup \{c : \theta\}$ and we conclude with an application of (T-SUB). □

Lemma B.2 (Lemma 4.1). If $\Gamma; \mathscr{C} \vdash P : \Delta$ and $P \equiv Q$, then $\Gamma; \mathscr{C} \vdash Q : \Delta$.

Proof. We reason by induction on the derivation of $P \equiv Q$ and by cases on the last structural congruence rule applied, using the fact that $\preceq$ is a precongruence with respect to $|$, $\oplus$, and external choices when branches are guarded. Most cases are trivial, we focus on the interesting ones:

($P = Q \mid \mathsf{nil} \equiv Q$) Then $\Delta \preceq \{u_i : \sigma_i \mid \tau_i\}^{i \in I}$ where $\Gamma; \mathscr{C} \vdash Q : \{u_i : \sigma_i\}^{i \in I}$ and $\{u_i : \tau_i\}^{i \in I} \preceq \emptyset$. We deduce $\tau_i \preceq \mathbf{1}$ for every $i \in I$. We conclude by observing that $\Delta \preceq \{u_i : \sigma_i \mid \tau_i\}^{i \in I} \preceq \{u_i : \sigma_i \mid \mathbf{1}\}^{i \in I} \preceq \{u_i : \sigma_i\}^{i \in I}$.

($P \equiv P \mid \mathsf{nil}$) Let $\Delta' \stackrel{\text{def}}{=} \{u : \mathbf{1} \mid u \in \mathtt{dom}(\Delta)\}$ and observe that $\Gamma; \mathscr{C} \vdash \mathsf{nil} : \Delta'$. Since $\Delta \approx \Delta \mid \Delta'$ we conclude with an application of rule (T-PAR) followed by (T-SUB).

($P = (\mathsf{new}\ a)(P_1 \mid P_2) \equiv P_1 \mid (\mathsf{new}\ a)P_2 = Q$ **where** $a \notin \mathtt{fn}(P_1)$) Then $\Delta \preceq \Delta_1 \mid \Delta_2$ for some Δ_1 and Δ_2 such that $\Gamma; \mathscr{C} \vdash P_1 : \Delta_1 \cup \{a : \sigma_1\}$ and $\Gamma; \mathscr{C} \vdash P_2 : \Delta_2 \cup \{a : \sigma_2\}$ where $\sigma_1 \mid \sigma_2$ is complete. From $a \notin \mathtt{fn}(P)$ we deduce $\sigma_1 \preceq \mathbf{1}$, hence $\sigma_1 \mid \sigma_2 \preceq \mathbf{1} \mid \sigma_2 \approx \sigma_2$, namely σ_2 is complete. We deduce $\Gamma; \mathscr{C} \vdash (\mathsf{new}\ a)P_2 : \Delta_2$. We conclude by observing that $\Gamma; \mathscr{C} \vdash P_1 : \Delta_1$ since $a \notin \mathtt{fn}(P_1)$.

($P = P_1 \mid (\mathsf{new}\ a)P_2 \equiv (\mathsf{new}\ a)(P_1 \mid P_2) = Q$ **where** $a \notin \mathtt{fn}(P_1)$) Then $\Delta \preceq \Delta_1 \mid \Delta_2$ for some Δ_1 and Δ_2 such that $\Gamma; \mathscr{C} \vdash P_1 : \Delta_1$ and $\Gamma; \mathscr{C} \vdash P_2 : \Delta_2 \cup \{a : \sigma_2\}$ and $a \notin \mathtt{dom}(\Delta_1)$ and σ_2 is complete. By rule (T-SUB) we derive $\Gamma; \mathscr{C} \vdash P_1 : \Delta_1 \cup \{a : \mathbf{1}\}$. By rule (T-PAR) we derive $\Gamma; \mathscr{C} \vdash P_1 \mid P_2 : (\Delta_1 \mid \Delta_2) \cup \{a : \mathbf{1} \mid \sigma_2\}$. We conclude with an application of (T-RES) followed by (T-SUB) since $\mathbf{1} \mid \sigma_2$ is complete. □

Theorem B.1 (Theorem 4.1). Let $\Gamma;\mathscr{C} \vdash P : \Delta$ and $P \to Q$ and Δ viable. Then $\Gamma;\mathscr{C} \vdash Q : \Delta$.

Proof. By induction on the derivation of $P \to Q$ and by cases on the last rule applied. The cases for rules (R-NEW) and (R-PAR) follow by a simple induction, the latter one since $\preceq$ is preserved by $|$. The case for rule (R-CHOICE) is trivial because of the typing rule (T-CHOICE) and the case for rule (R-CONG) follows from the inductive hypothesis and Lemma 4.1.

As regards rule (R-COMM), we have $P = a!e.P' \,|\, \sum_{i\in I} a?(x:t_i).P_i \to P' \,|\, P_k\{^{\mathsf{v}}/_x\} = Q$ where $e \downarrow \mathsf{v}$ and $\mathsf{v} \in t_k$ for some $k \in I$. We deduce $\Delta \preceq (\Delta' \,|\, \Delta'') \cup \{a : !t.\sigma \,|\, \sum_{i\in I} ?t_i.\sigma_i\}$ where $\Gamma \vdash e : t$ and $\Gamma;\mathscr{C} \vdash P' : \Delta' \cup \{a : \sigma\}$ and $\Gamma, x : t_k;\mathscr{C} \vdash P_k : \Delta'' \cup \{a : \sigma_k\}$. By Lemma B.1(1) we deduce $\Gamma;\mathscr{C} \vdash P_k\{^{\mathsf{v}}/_x\} : \Delta'' \cup \{a : \sigma_k\}$. We conclude by observing that $!t.\sigma \,|\, \sum_{i\in I} ?t_i.\sigma_i \longrightarrow !\mathsf{v}.\sigma \,|\, \sum_{i\in I} ?t_i.\sigma_i \longrightarrow \sigma \,|\, \sigma_k$, hence $!t.\sigma \,|\, \sum_{i\in I} ?t_i.\sigma_i \preceq \sigma \,|\, \sigma_k$.

As regards rule (R-COMMS), we have $P = a!c.P_1 \,|\, a?(x).P_2 \to P_1 \,|\, P_2\{^c/_x\} = Q$. We deduce $\Delta \preceq (\Delta_1 \,|\, \Delta_2) \cup \{a : !\rho_1.\sigma_1 \,|\, ?\rho_2.\sigma_2, c : \tau \,|\, \rho_1\}$ where **(1)** $\Gamma;\mathscr{C} \vdash P_1 : \Delta_1 \cup \{a : \sigma_1, c : \tau\}$ and **(2)** $\mathscr{C} \vdash c \prec a$ and **(3)** $\Gamma;\mathscr{C}, x \prec a \vdash P_2 : \Delta_2 \cup \{a : \sigma_2, x : \rho_2\}$ and **(4)** $\mathscr{C} \vdash a \prec \mathtt{dom}(\Delta_2)$. From **(2)** and **(4)** we derive $c \notin \mathtt{dom}(\Delta_2)$. From **(2)** we also derive that $(\mathscr{C}, x \prec a)\{^c/_x\} = \mathscr{C}$. From **(3)** and by Lemma B.1(2) we deduce $\Gamma;\mathscr{C} \vdash P_2\{^c/_x\} : \Delta_2 \cup \{a : \sigma_2, c : \rho_2\}$. From the hypothesis Δ viable we deduce $\rho_1 \preceq \rho_2$. By (T-PAR) we derive $\Gamma;\mathscr{C} \vdash Q : (\Delta_1 \,|\, \Delta_2) \cup \{a : \sigma_1 \,|\, \sigma_2, c : \tau \,|\, \rho_2\}$. We conclude by observing that $!\rho_1.\sigma_1 \,|\, ?\rho_2.\sigma_2 \longrightarrow \sigma_1 \,|\, \sigma_2$ hence $!\rho_1.\sigma_1 \,|\, ?\rho_2.\sigma_2 \preceq \sigma_1 \,|\, \sigma_2$ and furthermore $\tau \,|\, \rho_1 \preceq \tau \,|\, \rho_2$ since $\preceq$ is a precongruence for $|$ and $\rho_1 \preceq \rho_2$. □

Theorem B.2 (Theorem 4.2). If $\Gamma;\mathscr{C} \vdash P : \Delta \cup \{c : \sigma\}$ and σ complete and $P \downarrow c$, then $P \to$.

Proof. We have $P \equiv P_1 \,|\, \cdots \,|\, P_n$ where each P_i is guarded by a prefix which subject is c. We deduce that there exist $\Delta_1, \ldots, \Delta_n$ and $\sigma_1, \ldots, \sigma_n$ such that $\sigma \preceq \sigma_1 \,|\, \cdots \,|\, \sigma_n$ and $\Gamma;\mathscr{C} \vdash P_i : \Delta_i \cup \{c : \sigma_i\}$ for every $i \in \{1, \ldots, n\}$ and each σ_i is the session type directly determined by the conclusion of one of the rules (T-OUTPUT), (T-INPUT), (T-OUTPUTS), or (T-INPUTS). Let $I \subseteq \{1, \ldots, n\}$ be the set of indexes such that $i \in I$ implies $P_i = c!e_i.P_i'$ for some e_i and P_i'. Observe that $e_i \downarrow \mathsf{v}_i$ implies $\sigma_i \Longrightarrow !\mathsf{v}_i.\sigma_i'$ for every $i \in I$. From the hypothesis σ complete we deduce that $\sigma_1 \,|\, \cdots \,|\, \sigma_n$ is also complete hence there exist $i \in I$ and $j \in \{1, \ldots, n\} \setminus I$ such that either $e_i \downarrow \mathsf{v}_i$ and $\sigma_j \xrightarrow{?\mathsf{v}_i}$ or $e_i = a$ and $\sigma_j \xrightarrow{?\diamond}$. We deduce $P_i \,|\, P_j \to$ which implies $P \to$. □