

# Session Types = Intersection Types + Union Types

(extended abstract)

Luca Padovani

Dipartimento di Informatica, Università di Torino  
Corso Svizzera 185, Torino, Italy  
padovani@di.unito.it

We propose a semantically grounded theory of session types which relies on intersection and union types. We argue that intersection and union types are natural candidates for modeling branching points in session types and we show that the resulting theory overcomes some important defects of related behavioral theories.

## 1 Introduction

Session types [12, 13, 14] are protocol descriptions that constrain the use of communication channels in distributed systems. In these systems, processes engage into a conversation by first establishing a *session* on some private channel and then carrying on the conversation within the protected scope of the session. The session type *prescribes*, for each process connected by means of the session, the sequence and type of messages the process is allowed to send or expected to receive at each given time. For example, the session type  $\bar{a}.a.b$  associated with some channel  $c$  states that a process can use  $c$  for sending two  $a$  messages and then waiting for a  $b$  message, *in this order*. Names  $a$  and  $b$  may stand for either message types, labels, method names depending on the process language one is considering.

In most session type theories it is possible to specify protocols with *branching points* indicating alternative behaviors: for example, the session type  $\bar{a}.\eta \square \bar{b}.\theta$  usually means that a process chooses to send either an  $a$  message or a  $b$  message and then behaves according to  $\eta$  or  $\theta$  depending on the message that is sent; dually, the session type  $a.\eta \square b.\theta$  usually means that a process waits for either an  $a$  message or a  $b$  message, and then behaves according to the respective continuation. In these examples, as in the session type theories cited above, one is making the implicit assumption that the process actively choosing to follow one particular branch is the one that sends messages, while the process passively waiting for the decision is the one that receives messages. In practice, it is appropriate to devise two distinct branching operators, instead of a single one  $\square$  like in the examples above, to emphasize this fact. This is the key intuition in [3, 16] where session types are studied as proper terms of a simple process algebra with action prefixes and two choice operators: the *internal choice*  $\eta \oplus \theta$  denotes that the process decides which branch,  $\eta$  or  $\theta$ , to take and behaves accordingly; the *external choice*  $\eta + \theta$  denotes that the process offers two possible behaviors,  $\eta$  and  $\theta$ , and leaves the decision as to which one to follow to the process at the other end of the communication channel.

The approach advocated in [3, 16] recasts session types into well-known formalisms (process algebras) by fully embracing their behavioral nature. This permits the definition of an elegant, semantically grounded subtyping relation  $\leq$  for session types as an adaptation of the well-known *must* pre-order relation for processes [7, 6]. Nonetheless, the resulting theory of session types suffers from a few shortcomings. First of all, the semantics of the external choice is sometimes indistinguishable from that of the internal choice: the typical example, which is also one of the pivotal laws of the *must* pre-order,

Table 1: Syntax of processes.

$P ::=$	<b>process</b>	$\alpha ::=$	<b>action</b>
	$\mathbf{1}$		$a$ (input)
	$\alpha.P$		$\bar{a}$ (output)
	$P \oplus P$		
	$P + P$		

is  $a.\eta + a.\theta \geq a.(\eta \oplus \theta)$  (we write  $\geq$  for the equivalence relation induced by  $\leq$ ). As a direct consequence of this, the subtyping relation  $\leq$  fails to be a pre-congruence. Indeed we have  $a.b \leq a.b + b.c$  but  $a.b + b.d \not\leq a.b + b.c + b.d \geq a.b + b.(c \oplus d)$ . This poses practical problems (one has to characterize the contexts in which subtyping is safe) as well as theoretical ones ( $\leq$  is harder to characterize axiomatically). In addition, recent developments of session type theories have shown a growing interest toward the definition of *meet* and *join* operators over session types [15], which must be defined in an *ad hoc* manner since these do not always correspond to the internal choice and the external choice.

In this paper we address all of these shortcomings by proposing a language of session types where intersection and union types model branching points. The idea is that when some channel is typed by the intersection type  $\bar{a}.\eta \wedge \bar{b}.\theta$  this means that the channel has both type  $\bar{a}.\eta$  and also type  $\bar{b}.\theta$ . As a consequence, a process conforming to this type can choose to send an  $a$  message or a  $b$  message and then use the channel as respectively prescribed by  $\eta$  and  $\theta$ . Dually, when some channel is typed by the union type  $a.\eta \vee b.\theta$  this means that the process does not precisely know the type of the channel, which may be either  $a.\eta$  or  $b.\theta$ . Hence it must be ready to receive both an  $a$  message and a  $b$  message. It is the message received from the channel that helps the process disambiguate the type of the channel. If the message does not provide enough information, the ambiguity is propagated, hence one pivotal law of our theory is  $a.\eta \vee a.\theta \geq a.(\eta \vee \theta)$ .

In summary, we argue that intersection and union types are natural, type theoretic alternatives for internal and external choices, respectively; we show that intersection and union types have the same expressive power as the corresponding behavioral operators and that they provide us with native meet and join operators. Finally, we show that the semantically defined subtyping relation for session types is a pre-congruence.

**Structure of the paper.** Section 2 is devoted to the formalization of processes and of correct process interactions in dyadic sessions (i.e., we consider sessions linking exactly two processes). We introduce session types in Section 3, where we use the formalization of processes from the previous section for defining their semantics. The section also includes an extended example motivating the need to compute meet and join of session types. We conclude in Section 4 with a summary of the paper and a few hints at future research directions.

## 2 Processes

Let us fix some notation: we let  $a, b, \dots$  range over some set  $\mathcal{N}$  of *action names* whose meaning is left unspecified; we let  $P, Q, \dots$  range over *processes* and  $\alpha, \beta, \dots$  range over *actions*. We distinguish *input actions* of the form  $a$  from *output actions* of the form  $\bar{a}$ ; we say that  $\bar{a}$  is the *co-action* of  $\alpha$  where  $\bar{\bar{a}} = a$ . We consider the simple language of sequential processes whose grammar is described in Table 1. Syntactically speaking the language is a minor variation of CCS without  $\tau$ 's [7, 11] without relabeling,

Table 2: Operational semantics of processes (symmetric rules omitted).

(R1) $\mathbf{1} \xrightarrow{\checkmark} \mathbf{1}$	(R2) $\alpha.P \xrightarrow{\alpha} P$	(R3) $P \oplus Q \longrightarrow P$
(R4) $\frac{P \longrightarrow P'}{P + Q \longrightarrow P' + Q}$	(R5) $\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$	(R6) $\frac{P \xrightarrow{\bar{a}} P'}{P + Q \longrightarrow \bar{a}.P'}$

restriction, and parallel composition. The term  $\mathbf{1}$  denotes an idle process that performs no further action; it represents a successfully terminated interaction. The term  $\alpha.P$  denotes a process that performs action  $\alpha$  and then continues as  $P$ . The term  $P \oplus Q$  is the internal choice of  $P$  and  $Q$  and denotes a process that internally decides whether to behave as  $P$  or as  $Q$ . Finally, the term  $P + Q$  is the external choice of  $P$  and  $Q$  and denotes a process that externally offers two behaviors,  $P$  and  $Q$ , and lets the environment decide which one it should follow. As we will see shortly, the decision of the environment is guided, as usual, by the initial actions performed by  $P$  and  $Q$ .

We do not rely on any explicit syntax for describing recursive processes (hence potentially infinite behaviors). Simply, we define processes as the possibly infinite syntax trees generated by the productions for  $P$  in Table 1 such that every infinite branch of the tree has infinitely many occurrences of the action prefix operator. This 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. For practical purposes we could also restrict our attention to *regular trees* [5], namely to finite-state processes. However this is not strictly necessary in this paper which is not concerned with algorithmic issues. In the following we will usually omit trailing  $\mathbf{1}$ 's and write, for example,  $a.\bar{b}$  instead of  $a.\bar{b}.\mathbf{1}$ .

**Example 2.1** Consider the processes  $P$  and  $Q$  that are solution of the equations  $P = \bar{a}.P \oplus \bar{b}$  and  $Q = a.Q + b$  (contractivity ensures that these equations have a unique solution [5]). The process  $P$  sends an arbitrary, possibly infinite number of ‘a’ messages and, if it stops sending ‘a’ messages it will send a ‘b’ message before terminating. The process  $Q$  is capable of receiving an arbitrary, possibly infinite number of ‘a’ messages and it terminates as soon as it receives a ‘b’ message. Intuitively, the processes  $P$  and  $Q$  are able to interact “smoothly” with each other, for some definition of “smoothly” that we postpone after having formally defined the operational semantics of processes. ■

The formal meaning of processes is defined by means of a labelled transition system, defined in Table 2 (to avoid clutter, symmetric rules have been omitted). *Labels* are elements of  $\mathcal{N} \cup \{\bar{a} \mid a \in \mathcal{N}\} \cup \{\checkmark\}$ , where  $\checkmark \notin \mathcal{N}$  denotes successful termination and  $\alpha$  denotes the execution of an action  $\alpha$ . We extend the  $\bar{\cdot}$  involution to labels so that  $\overline{\checkmark} = \checkmark$  and to sets of labels  $\Lambda$  so that  $\bar{\Lambda} = \{\bar{\mu} \mid \mu \in \Lambda\}$ . The transition system consists of two relations, an unlabelled one  $\longrightarrow$  representing *internal, invisible* transitions of a process and a labelled one  $\xrightarrow{\mu}$  representing *external, visible* transitions of a process. We briefly describe the meaning of the rules in the following paragraph: rule (R1) signals the fact that the process  $\mathbf{1}$  has terminated successfully. In this sense the label  $\checkmark$  should not be considered as a proper action, but simply a flag; rule (R2) states that a process  $\alpha.P$  may execute the action  $\alpha$  and reduce to  $P$ ; rule (R3) states that a process  $P \oplus Q$  internally decides to reduce to either  $P$  or  $Q$ ; rule (R4) states that internal decisions taken in some branch of an external choice do not preempt the other branch of the external choice. This rule is common in process algebras distinguishing between internal and external

choices, such as CCS without  $\tau$ 's [7] from which our process language is inspired. Rule (R5) states that an external choice offers any action that is offered by either branch of the choice. Rule (R6) is possibly the less familiar one. It states that a process performing an output action may preempt other branches of an external choice. This rule has been originally introduced in [4] where the message sent is detached from its corresponding continuation, which is thus immediately capable of interacting with the surrounding environment. Here, as in [3], we keep the message and its continuation attached together, so as to model an asynchronous form of communication where the order of messages is preserved. In the following we will sometimes use the following notation: we will write  $\Longrightarrow$  for the reflexive and transitive closure of  $\longrightarrow$ ; we will write  $P \not\rightarrow$  if there is no  $P'$  such that  $P \longrightarrow P'$ ; we write  $P \xrightarrow{\mu}$  if  $P \xrightarrow{\mu} P'$  for some  $P'$ ; let  $\text{init}(P) \stackrel{\text{def}}{=} \{\mu \mid P \xrightarrow{\mu}\}$ .

The next and final step is to describe how two processes interact and when they do so without errors. Informally,  $P$  and  $Q$  interact without errors if, regardless of the respective internal choices, they are always capable of synchronizing by means of complementary actions or they have both successfully terminated. We formalize this as the following duality relation between processes:

**Definition 2.1 (process duality)** *Let  $\longrightarrow$  be the smallest relation between systems  $P|Q$  of two processes such that:*

$$\frac{P \longrightarrow P'}{P|Q \longrightarrow P'|Q} \quad \frac{Q \longrightarrow Q'}{P|Q \longrightarrow P|Q'} \quad \frac{P \xrightarrow{\bar{\alpha}} P' \quad Q \xrightarrow{\alpha} Q'}{P|Q \longrightarrow P'|Q'}$$

and let  $\Longrightarrow$  be the reflexive, transitive closure of  $\longrightarrow$ . Let  $P|Q \not\rightarrow$  if there are no  $P'$  and  $Q'$  such that  $P|Q \longrightarrow P'|Q'$ . We say that  $P$  and  $Q$  are dual, notation  $P \bowtie Q$ , if  $P|Q \Longrightarrow P'|Q' \not\rightarrow$  implies  $P' \checkmark$  and  $Q' \checkmark$ .

**Example 2.2** *The processes  $P$  and  $Q$  in Example 2.1 are dual. Indeed every possible interaction has either the form  $P|Q \Longrightarrow \bar{a}.P|Q$  which reduces back to  $P|Q$ , or the form  $P|Q \Longrightarrow \bar{b}.1|Q \longrightarrow 1|1$  where both processes have successfully terminated. ■*

We conclude this section discussing some more subtle examples. First of all observe that the external choice is not always truly external, as we have anticipated in the introduction. Indeed we have

$$a.\bar{a} + a.\bar{b} | \bar{a}.a \longrightarrow \bar{b} | a \not\rightarrow$$

The problem here is that both branches of the external choice are guarded by the same action  $a$ , and since it is the *initial* performed action to determine the chosen branch the process  $a.\bar{a} + a.\bar{b}$  does not really offer an external choice, but an internal one. Indeed, it is possible to prove that  $a.\bar{a} + a.\bar{b}$  and  $a.(\bar{a} \oplus \bar{b})$  are indistinguishable (every dual of  $a.\bar{a} + a.\bar{b}$  is a dual of  $a.(\bar{a} \oplus \bar{b})$  and vice-versa). Second, rule (R6) shows that output actions may somewhat turn external choices into internal ones. For instance

$$\bar{a} + b | \bar{b} \longrightarrow \bar{a} | \bar{b} \not\rightarrow$$

since the process on the left hand side of  $|$  may autonomously decide to send message  $\bar{a}$  even though process  $\bar{b}$  is not ready to receive it. The interaction between  $\bar{a} + b | \bar{b} + a$  would also fail, since  $\bar{b} + a \longrightarrow \bar{b}$ . Finally, observe that there are pathological processes that are intrinsically flawed and cannot interact smoothly with any other process. For example,  $a \oplus b$  has no dual process since it is not possible to know which message,  $a$  or  $b$ , it is ready to receive. As another example the process  $P = a \oplus \bar{b}$  has no dual: no process interacting with it can send an  $a$  message, since  $P \longrightarrow \bar{b}$ ; at the same time, a process waiting for the  $b$  message from  $P$  may starve forever since  $P \longrightarrow a$ .

Table 3: Syntax of session types.

$\eta$	$::=$	<b>session type</b>
		<b>1</b> (termination)
		$\alpha.\eta$ (prefix)
		$\eta \wedge \eta$ (intersection)
		$\eta \vee \eta$ (union)

### 3 Session Types

In this section we introduce our language of session types, which is syntactically very similar to that of processes, but semantically differs in the interpretation of the branching operators. We let  $\eta, \theta, \dots$  range over *session types*, which are defined by the grammar in Table 3. The term **1** is the type of channels on which no further action is possible; the term  $\alpha.\eta$  is the type of channels on which it is possible to perform an action  $\alpha$ . Actions are the same that occur within processes, but the point of view is slightly different: a process *executes* an action, while a session type *prescribes* the execution of an action. The term  $\eta \wedge \theta$  is the type of channels that have *both* types  $\eta$  and  $\theta$ . For example  $c : \bar{a}.\mathbf{1} \wedge \bar{b}.\mathbf{1}$  denotes that channel  $c$  has both type  $\bar{a}.\mathbf{1}$  and also type  $\bar{b}.\mathbf{1}$ , namely it can be used for sending both messages  $a$  and  $b$ . finally, the term  $\eta \vee \theta$  is the type of channel that either have type  $\eta$  or  $\theta$ . For instance  $d : a.\mathbf{1} \vee b.\mathbf{1}$  means that a process using channel  $d$  must be ready to receive both a message  $a$  and a message  $b$ , since it does not know whether the type of the channel is  $a.\mathbf{1}$  or  $b.\mathbf{1}$ .<sup>1</sup> As we did for processes, we do not introduce any explicit notation for denoting infinite types and instead we rely on infinite terms satisfying the same contractivity condition. We will also omit trailing **1**'s to avoid clutter.

Before giving a formal semantics to session types let us discuss a few examples to highlight similarities and differences between them and processes. It should be pretty obvious that  $\oplus$  and  $\wedge$  play the same role: the ability for a process  $P \oplus Q$  to autonomously decide which behavior,  $P$  or  $Q$ , to perform indicates that the session type associated with the channel it is using allows both alternatives, it has *both* types. Conversely,  $+$  and  $\vee$  may differ depending on the respective operands. For instance, consider  $P = a.b.\bar{a} + a.c.\bar{b}$  and  $\eta = a.b.\bar{a} \vee a.c.\bar{b}$ . The external choice in  $P$  is guarded by the same action  $a$ , meaning that after performing action  $a$  the process may either reduce to  $b.\bar{a}$  or to  $c.\bar{b}$ , the choice being nondeterministic. As we have already argued at the end of Section 2, one can show that  $P$  is equivalent to  $a.(b.\bar{a} \oplus c.\bar{b})$ , where the nondeterministic choice between the two residual branches is explicit. The session type  $\eta$ , on the other hand, tells us something different: we do not know whether the channel we are using has type  $a.b.\bar{a}$  or  $a.c.\bar{b}$  and receiving message  $a$  from it does not help to solve this ambiguity. Therefore, after the message  $a$  has been received, we are left with a channel whose associated session type is  $b.\bar{a} \vee c.\bar{b}$ . At this stage, depending on the message,  $b$  or  $c$ , that is received, we are able to distinguish the type of the channel, and to send the appropriate message (either  $a$  or  $b$ ) before terminating. In summary,  $P$  and  $\eta$  specify quite different behaviors, and in fact while  $\eta$  is perfectly reasonable, the reader may verify that  $P$  is intrinsically flawed and cannot interact successfully with any other process.

To formalize the semantics of session types we relate them with processes and to do so we equip them with the operational semantics defined in Table 4 (symmetric rules omitted). In the table we write  $\eta \xrightarrow{\alpha}$  if there is no unguarded prefix  $\alpha$  in  $\eta$  and we write  $\eta \xrightarrow{\surd}$  if there is no unguarded **1** in  $\eta$ . The relation  $\xrightarrow{\mu}$  allows us to understand which actions are permitted by a session type and to preserve the

<sup>1</sup>We are making the implicit assumption that “using a channel” means either sending a message on it or waiting a message from it and that no *type-case* construct is available for querying the actual type of a channel.

Table 4: Operational semantics of session types.

(T1) $\mathbf{1} \xrightarrow{\checkmark} \mathbf{1}$	(T2) $\alpha.\eta \xrightarrow{\alpha} \eta$	(T3) $\frac{\eta \xrightarrow{\mu} \eta' \quad \theta \xrightarrow{\mu} \theta'}{\eta \wedge \theta \xrightarrow{\mu} \eta'}$	(T4) $\frac{\eta \xrightarrow{\mu} \eta' \quad \theta \xrightarrow{\mu} \theta'}{\eta \vee \theta \xrightarrow{\mu} \eta'}$
	(T5) $\frac{\eta \xrightarrow{\mu} \eta' \quad \theta \xrightarrow{\mu} \theta'}{\eta \wedge \theta \xrightarrow{\mu} \eta' \wedge \theta'}$	(T6) $\frac{\eta \xrightarrow{\mu} \eta' \quad \theta \xrightarrow{\mu} \theta'}{\eta \vee \theta \xrightarrow{\mu} \eta' \vee \theta'}$	

information about the modalities in which actions are offered in the residual session type. Rules (T1) and (T2) are basically the same as the corresponding rules (R1) and (R2) for processes and are obvious. Rules (T3) and (T5) state that a session type  $\eta \wedge \theta$  enables a  $\mu$  action if so does either  $\eta$  or  $\theta$ : if both branches enable the action, then the residual session type is given by the conjunction of the residuals of the two branches; if only one branch enables the action, then the residual coincides with the residual of that branch. Rules (T4) and (T6) are the dual of rules (T3) and (T5) and they deal with the union type  $\eta \vee \theta$ . For example we have  $a.\eta \vee a.\theta \xrightarrow{a} \eta \vee \theta$  and  $a.\eta \vee b.\theta \xrightarrow{a} \eta$ .

Unlike the transition relations of processes,  $\xrightarrow{\quad}$  is deterministic:  $\eta \xrightarrow{\mu} \eta_1$  and  $\eta \xrightarrow{\mu} \eta_2$  implies  $\eta_1 = \eta_2$ . For this reason  $\xrightarrow{\quad}$  says nothing about the modalities in which actions are offered, as determined by intersections and unions in the session types. For example,  $\bar{a} \wedge \bar{b}$  and  $\bar{a} \vee \bar{b}$  enable the same actions, but they prescribe quite different behaviors. Hence  $\xrightarrow{\quad}$  alone is not enough to characterize the semantics of session types. We capture the modality in which actions are offered by introducing a *readiness relation* which is inductively defined thus:

$$\mathbf{1} \downarrow \{\checkmark\} \quad \alpha.\eta \downarrow \{\alpha\} \quad \frac{\eta \downarrow A}{\eta \wedge \theta \downarrow A} \quad \frac{\eta \downarrow A \quad \theta \downarrow B}{\eta \vee \theta \downarrow A \cup B}$$

plus the symmetric of the rule for  $\wedge$ . Intuitively  $\eta \downarrow A$  means that a process conforming to  $\eta$  must be ready to perform any action in  $A$ . For example we have  $\bar{a} \wedge \bar{b} \downarrow \{\bar{a}\}$  and  $\bar{a} \wedge \bar{b} \downarrow \{\bar{b}\}$  while  $a \vee b \downarrow \{a, b\}$ . Thus a process conforming to  $\bar{a} \wedge \bar{b}$  has the choice to send either message  $a$  or message  $b$ ; a process conforming to  $a \vee b$  must be ready to receive both an  $a$  message and a  $b$  message.

We now have all the ingredients for formally defining the *services* of a session type, namely those processes that conform to a behavior which is (at least) as good as the one prescribed by the session type:

**Definition 3.1 (service)** We say that  $\mathcal{S}$  is a service relation if  $\eta \mathcal{S} Q$  implies: **(1)**  $Q \Longrightarrow Q'$  implies  $\eta \downarrow A$  for some  $A \subseteq \text{init}(Q')$ ; **(2)**  $Q \xrightarrow{\mu} Q'$  and  $\eta \xrightarrow{\mu} \eta'$  implies  $\eta' \mathcal{S} Q'$ . We say that  $Q$  is a service of  $\eta$ , written  $\eta \triangleleft Q$ , if  $\eta \mathcal{S} Q$  for some service relation  $\mathcal{S}$ .

When  $\eta \triangleleft Q$ , condition **(1)** imposes that any internal choice of the service ( $Q \Longrightarrow Q'$ ) must correspond to a readiness relation in the session type ( $\eta \downarrow A$ ). For example we have  $\bar{a} \triangleleft \bar{a}$  and  $\bar{a} \wedge \bar{b} \triangleleft \bar{a}$  and also  $\bar{a} \wedge \bar{b} \triangleleft \bar{a} \oplus \bar{b}$ , while  $\bar{a} \not\triangleleft \bar{a} \oplus \bar{b}$  and  $a \vee b \not\triangleleft a$ . Condition **(2)** imposes that any residual of  $\eta$  and  $Q$  after some common action  $\mu$  must be related by the service relation. For example we have  $a.b.\bar{a} \vee a.c.\bar{b} \triangleleft a.(b.\bar{a} + c.\bar{b})$  because  $a.b.\bar{a} \vee a.c.\bar{b} \xrightarrow{a} b.\bar{a} \vee c.\bar{b}$  and  $b.\bar{a} \vee c.\bar{b} \triangleleft b.\bar{a} + c.\bar{b}$  however  $a.b.\bar{a} \vee a.c.\bar{b} \not\triangleleft a.b.\bar{a} + a.c.\bar{b}$  because  $a.b.\bar{a} + a.c.\bar{b} \xrightarrow{a} b.\bar{a}$  and  $b.\bar{a} \vee c.\bar{b} \not\triangleleft b.\bar{a}$ .

In a dual fashion we define the *clients* of a session type  $\eta$  as those processes that interact smoothly with another one that conforms to  $\eta$ :

**Definition 3.2 (client)** We say that  $\mathcal{C}$  is a client relation if  $P \mathcal{C} \eta$  implies: **(1)**  $P \Longrightarrow P'$  and  $\eta \downarrow A$  implies  $\overline{\text{init}(P')} \cap A \neq \emptyset$ ; **(2)**  $P \xrightarrow{\bar{\mu}} P'$  and  $\eta \vdash^{\mu} \eta'$  implies  $P' \mathcal{C} \eta'$ . We say that  $P$  is a client of  $\eta$ , written  $P \triangleright \eta$ , if  $P \mathcal{C} \eta$  for some client relation  $\mathcal{C}$ .

Condition **(1)** imposes that, regardless of any internal choice of the client ( $P \Longrightarrow P'$ ) and regardless of any internal choice of the service of the session type ( $\eta \downarrow A$ ), client and service must be able to synchronize or they must have both successfully terminated ( $\overline{\text{init}(P')} \cap A \neq \emptyset$ ). Condition **(2)** requires the respective residuals of the process and of the session type to be in the client relation, as for the service relation. Note however that the process and the session type perform complementary actions. For example we have  $\bar{a} \triangleright a$  and  $\bar{a} \oplus \bar{b} \triangleright a \vee b$ , but  $\bar{a} \oplus \bar{b} \not\triangleright a$ .

According to Definitions 3.1 and 3.2 we can now define two semantics for the same session type  $\eta$ , respectively as the sets of clients and services of  $\eta$ :

**Definition 3.3 (session type semantics)** Let  $C[\eta] \stackrel{\text{def}}{=} \{P \mid P \triangleright \eta\}$  and  $S[\eta] \stackrel{\text{def}}{=} \{Q \mid \eta \triangleleft Q\}$ .

As we have already observed for processes, there exist intrinsically flawed behaviors. The non-flawed behaviors, namely those session types that can be implemented by services and consumed by clients, are important to the point that we reserve a name for them:

**Definition 3.4 (viability)** We say that  $\eta$  is viable if  $C[\eta] \neq \emptyset$  and  $S[\eta] \neq \emptyset$ .

In words, a session type is viable if there exists a service that conforms to it *and* if there exists a client that can interact with it. The two conditions  $C[\eta] \neq \emptyset$  and  $S[\eta] \neq \emptyset$  are *not* equivalent. For instance, take  $\eta = a \vee \bar{b}$ . Then  $\bar{a} \oplus b \triangleright \eta$ , namely there exists a client of  $\eta$ , but there is no service of  $\eta$ . Dually, we have that  $\theta = a \wedge b$  can be implemented by some services, for example  $\theta \triangleleft a \oplus b$ , but  $\theta$  has no client. By means of suitable combinations we can also obtain session types that have no clients and no services, such as  $a.\theta \vee \bar{b}.\theta$ .

The semantics of session types allows us to define the subtyping relation in terms of the sets of clients and services of the session types being related:

**Definition 3.5 (subsession)** We say that  $\eta$  is a subsession of  $\theta$ , notation  $\eta \leq \theta$ , if  $C[\eta] \subseteq C[\theta]$  and  $S[\theta] \subseteq S[\eta]$ . We write  $\eta \geq \theta$  for  $\eta \leq \theta$  and  $\theta \leq \eta$ .

The intuition behind  $\leq$  is the same that accompanies other standard subtyping relations: when  $\eta \leq \theta$  any channel of type  $\eta$  can be used where a channel of type  $\theta$  is expected. In our context, suppose we have  $c : \theta$  and  $d : \eta$  and that some service acts on  $c$  according to the type  $\theta$ . Then it is safe to replace  $c$  with  $d$  because:

- $C[\eta] \subseteq C[\theta]$ , namely every client of  $\eta$  using channel  $d$  is also a client of  $\theta$  (recall that, while replacing  $c$  with  $d$ , the process using  $c$  is unaware that the substitution has taken place, hence it will keep behaving on  $d$  as if it was acting on  $c$ );
- $S[\theta] \subseteq S[\eta]$ , namely every service of  $\theta$  using channel  $c$  is also a service of  $\eta$ .

A thorough study of the equational theory for  $\leq$  goes beyond the scope of this extended abstract. We discuss just a few examples, which are fairly easy to check and pinpoint the main characteristics of  $\leq$ :

$$\begin{array}{lll}
 \text{(1)} \quad \eta \wedge \theta \leq \eta & \text{(3)} \quad \eta_1 \wedge (\eta_2 \vee \eta_3) \geq (\eta_1 \wedge \eta_2) \vee (\eta_1 \wedge \eta_3) & \text{(5)} \quad \alpha.\eta \wedge \alpha.\theta \geq \alpha.(\eta \wedge \theta) \\
 \text{(2)} \quad \eta \leq \eta \vee \theta & \text{(4)} \quad \eta_1 \vee (\eta_2 \wedge \eta_3) \geq (\eta_1 \vee \eta_2) \wedge (\eta_1 \vee \eta_3) & \text{(6)} \quad \alpha.\eta \vee \alpha.\theta \geq \alpha.(\eta \vee \theta)
 \end{array}$$

The distributivity laws (3) and (4) permit to derive interesting absorption rules, for example:  $\bar{a} \vee (\bar{b} \wedge \bar{a}) \geq \bar{a}$  (if the service is unsure whether message  $\bar{b}$  can be sent, only  $\bar{a}$  can be sent); the dual version of this relation is  $(a \vee b) \wedge a \geq a$  (if the service cannot guarantee that message  $b$  can be received, only message  $a$  can). The laws (2) and (6) distinguish the subsession relation from the *must* pre-order [7, 6, 11] where  $P$  and  $P + Q$  are seldom related and  $\alpha.P + \alpha.Q$  is equivalent to  $\alpha.(P \oplus Q)$ . The laws (1) and (5) do hold also for processes and in the *must* pre-order.<sup>2</sup>

**Proposition 3.1** *The following properties hold:*

1.  $\leq$  is a pre-order and a pre-congruence;
2. if  $\eta \leq \theta_1$  and  $\eta \leq \theta_2$ , then  $\eta \leq \theta_1 \wedge \theta_2$ ;
3. if  $\eta_1 \leq \theta$  and  $\eta_2 \leq \theta$ , then  $\eta_1 \vee \eta_2 \leq \theta$ .

Item (1) states that  $\leq$  behaves well with respect to all the operators (action prefix, intersection, and union) of the language of session types. Note that this is not true for processes, where it is unsafe, in general, to replace related processes when these occur in external choices (we have seen an instance of this problem in Section 1). Items (2) and (3) justify  $\wedge$  and  $\vee$  respectively as intersection and union operators. It must be noted, however, that they do not strictly induce an intersection or union in the semantics of session types. More specifically, we have that  $C[\eta] \cap C[\theta] = C[\eta \wedge \theta]$  (hence  $\eta \wedge \theta$  corresponds to the intersection of the clients) and  $C[\eta] \cup C[\theta]$  is a subset of, but not necessarily equal to,  $C[\eta \vee \theta]$ . The reason lies in the unavoidable fact that, among the clients of  $\eta \vee \theta$  are those of  $\eta$  and of  $\theta$ , but also those that internally decide to be clients of  $\eta$  or of  $\theta$ . These latter clients are not clients of  $\eta$  nor of  $\theta$  taken separately (unless, for example,  $\eta$  and  $\theta$  are equivalent). For instance, we have that  $\bar{a} \oplus \bar{b} \in C[a \vee b] \setminus (C[a] \cup C[b])$ . Dually, one can prove that  $S[\eta] \cap S[\theta] = S[\eta \vee \theta]$ , but again  $S[\eta] \cup S[\theta]$  is only included in  $S[\eta \wedge \theta]$ .

The following extended example shows the need to compute meets and joins of session types in some contexts. The availability of native unions and intersections within the language of session types makes this task trivial.

**Example 3.1 (global type projection)** Global types [14, 1] are abstract descriptions of interactions between two or more participants from a neutral point of view. For example, the global type

$$A \xrightarrow{a} B; A \xrightarrow{b} B \square A \xrightarrow{a} B; A \xrightarrow{c} B$$

specifies a system with two participants, here indicated by the tags  $A$  and  $B$ , which interact by exchanging messages ‘ $a$ ’, ‘ $b$ ’, and ‘ $c$ ’. In a global type, an action such as  $A \xrightarrow{a} B$  indicates that  $A$  sends an ‘ $a$ ’ message to  $B$ . Actions can be composed in sequences (with  $;$ ) and in alternative paths (with  $\square$ ). Overall, the global type describes which sequences of interactions are possible, but not who is responsible for which choices (hence the use of a single operator  $\square$  in branching points). The implementation of a global type begins by projecting it on each participant, so as to synthesize the session type that each participant must conform to. In this example we obtain the following projections: the projection on  $A$  is  $\bar{a}.\bar{b}$  on the l.h.s. and  $\bar{a}.\bar{c}$  on the r.h.s.; the projection on  $B$  is  $a.b$  on the l.h.s. and  $a.c$  on the r.h.s. Since  $A$  is the only sender, it is natural that its overall projection is  $\bar{a}.\bar{b} \wedge \bar{a}.\bar{c} \geq \bar{a}.\overline{(b \wedge c)}$ . Since  $B$  is the only receiver, it must be prepared to receive the messages from  $A$  regardless of which messages  $A$  decides to

<sup>2</sup>As an anonymous referee has pointed out, if one interprets  $a.\eta$  as the function type  $a \rightarrow \eta$ , then the laws (5) and (6) respectively correspond to  $(a \rightarrow \eta) \wedge (a \rightarrow \theta) \geq a \rightarrow (\eta \wedge \theta)$  and  $(a \rightarrow \eta) \vee (a \rightarrow \theta) \geq a \rightarrow (\eta \vee \theta)$  which do hold (at least partially) in more conventional type systems for functional languages with intersection and union types (see for example [8]).

send. Therefore, the correct projection of the global type on  $B$  is  $a.b \vee a.c \gtrsim a.(b \vee c)$ , which is the least upper bound of the projections on  $B$  of the two branches. In a language of session types with behavioral choices, this upper bound must be computed by an ad hoc operator, since  $a.b + a.c$  would be equivalent to  $a.(b \oplus c)$  which does not correspond to the correct projection for  $B$ . ■

We conclude this section with two results of correctness and completeness of the client/service characterizations we have given above:

**Theorem 3.1 (correctness and completeness)** *The following properties hold: (1)  $P \triangleright \eta$  and  $\eta \triangleleft Q$ , then  $P \bowtie Q$ ; (2) for every  $Q$  there exists  $\eta$  such that  $\eta \triangleleft Q$  and  $P \bowtie Q$  if and only if  $P \triangleright \eta$ .*

Correctness (1) states that two processes  $P$  and  $Q$  that are respectively client and service of the same session types are also dual. Completeness (2) has two important implications: the first one is that it is always possible to find the principal type of a service  $Q$ ; moreover, the principal type  $\eta$  of  $Q$  precisely captures the semantics of  $Q$ , in the sense that the set of duals of  $Q$  coincides with the clients of  $\eta$ .

## 4 Concluding Remarks and Future Work

We can identify both *behavioral* and *logical* approaches to the formalization of session types in the current literature. The behavioral approach advocated in [3, 16] is based on the observation that session types are behavioral types. As such, they are eligible for being studied by means of well-developed techniques for process equivalence, in particular testing equivalence [7, 6]. In this approach the different modalities in which actions are offered coincide with two known behavioral operators, the internal choice  $\oplus$  and the external choice  $+$ . The logical approach [2] interprets 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  $A \multimap B$  while output actions are modeled as the tensor product  $A \otimes B$ , hence there is no need for a “prefix” constructor as in our proposal. Remarkably, the seminal work of Honda [12] on session types was clearly inspired by linear logic, as witnessed by the use of  $\oplus$  and  $\&$  for modeling branching points.

In this work we propose a *type-theoretic* approach to the formalization of session types. The basic idea is that branching points correspond to intersection and union types: the session type  $\eta \wedge \theta$  describes a channel having *both* type  $\eta$  and type  $\theta$  and for this reason a process can freely use that channel as having either type; the session type  $\eta \vee \theta$  describes a channel having *either* type  $\eta$  or type  $\theta$ , therefore a process using that channel cannot make any assumption on it unless the exchanged messages provide enough information to disambiguate its type. With respect to the behavioral approach, our proposal addresses and solves a number of problems of practical interest: intersection and union operators subsume internal and external choices (without any loss in the expressiveness of session types, see Theorem 3.1(2)), they provide a native mechanism to the computation of greatest lower bounds and least upper bounds (Proposition 3.1(2,3)), and the subtyping relation of the resulting theory turns out to be a pre-congruence (Proposition 3.1(1)). Our proposal and the logical one in [2] share the property that branching points are modeled using operators whose semantics solely depends on the operators and not on the terms being combined. This is a key feature for having (pre)congruence in the resulting (in)equational theories. We postpone a more in-depth comparison between the two approaches to future work, when they will have both reached a higher degree of maturity. At the present time our session types accommodate recursive behaviors naturally using recursive types, whereas [2] only deals with finite formulas and hence finite behaviors; on the other hand, our session types are built using atomic actions and cannot express delegations, which are supported naturally by the logical approach proposed in [2].

This work may spread out as a fruitful line of research, of which we remark here just a few potential milestones: first of all, we plan to extend the simple language of session types presented here with more complex actions of the form  $?t$  and  $!t$  where  $t$  is a *basic type* (such as `int`, `bool`, ...) and actions of the form  $?\eta$  and  $!\eta$  for describing delegations (the input and output of channels of type  $\eta$ ). This will give rise to more interesting relations such as  $!\text{int} \vee !\text{real} \geqslant !\text{int}$  (assuming `int` is a subtype of `real`) and will allow us to compare more thoroughly our subsession relation with the existing ones [10]. Second, we plan to provide a complete axiomatization of  $\leqslant$  and a corresponding algorithm, which will necessarily be restricted to regular session types. Finally, it looks like the presented approach may be easily extended to incorporate universal and existential quantifiers in session types, so as to model polymorphism and data encapsulation. In this way we hope to provide semantic foundations to polymorphic session types [9].

**Acknowledgments.** I am grateful to the anonymous referees for the insightful comments and feedback and to Mariangiola Dezani for her encouragement and advice while I was writing this work.

## References

- [1] Mario Bravetti and Gianluigi Zavattaro. A foundational theory of contracts for multi-party service composition. *Fundamenta Informaticae*, 89(4):451–478, 2009.
- [2] Luis Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *CONCUR'10*. Springer, 2010 (to appear).
- [3] Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, Elena Giachino, and Luca Padovani. Foundations of session types. In *Proceedings of PPDP'09*, pages 219–230. ACM, 2009.
- [4] Ilaria Castellani and Matthew Hennessy. Testing theories for asynchronous languages. In *Proceedings of FSTTCS'98*, pages 90–101. Springer, 1998.
- [5] Bruno Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25:95–169, 1983.
- [6] Rocco De Nicola and Matthew Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.
- [7] Rocco De Nicola and Matthew Hennessy. CCS without  $\tau$ 's. In *Proceedings of TAPSOFT'87/CAAP'87*, LNCS 249, pages 138–152. Springer, 1987.
- [8] Joshua Dunfield and Frank Pfenning. Type assignment for intersections and unions in call-by-value languages. In *FoSSaCS'03*, LNCS 2620, pages 250–266. Springer, 2003.
- [9] Simon Gay. Bounded polymorphism in session types. *MSCS*, 18(5):895–930, 2008.
- [10] Simon Gay and Malcolm Hole. Subtyping for session types in the  $\pi$ -calculus. *Acta Informatica*, 42(2-3):191–225, 2005.
- [11] Matthew Hennessy. *Algebraic Theory of Processes*. Foundation of Computing. MIT Press, 1988.
- [12] Kohei Honda. Types for dyadic interaction. In *Proceedings of CONCUR'93*, LNCS 715, pages 509–523. Springer, 1993.
- [13] 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.
- [14] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *Proceedings of POPL'08*, pages 273–284. ACM, 2008.
- [15] Leonardo G. Mezzina. How to infer finite session types in a calculus of services and sessions. In *Proceedings of COORDINATION'98*, pages 216–231, 2008.
- [16] Luca Padovani. Session types at the mirror. *EPTCS*, 12:71–86, 2009.