# Session Types = Intersection Types + Union Types

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. In particular, intersections and unions provide a native solution to the problem of computing joins and meets of session types. Also, the subtyping relation turns out to be a pre-congruence, while this is not always the case in related behavioral theories.

## 1 Introduction

Session types [10, 11, 12] 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 involved in the session, the sequence and the type of messages the process is allowed to send or expected to receive at each given time. For example, the session type $\overline{a}.\overline{a}.b$ associated with some channel $\mathsf{c}$ states that a process can use $\mathsf{c}$ 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 and so forth, 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 $\overline{a}.T \,\Box\, \overline{b}.S$ usually means that a process chooses to send either an $a$ message or a $b$ message and then behaves according to $T$ or $S$ depending on the message that it has sent; dually, the session type $a.T \,\Box\, b.S$ 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 $\Box$ like in the examples above, to emphasize this fact. This is the key intuition in [3, 14, 1] where session types are studied as proper terms of a simple process algebra with action prefixes and two choice operators: the *internal choice* $T \oplus S$ denotes that the process decides which branch, $T$ or $S$, to take and behaves accordingly; the *external choice* $T + S$ denotes that the process offers two possible behaviors, $T$ and $S$, 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, 14] 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 $\lesssim$ for session types as an adaptation of the well-known *must* pre-order for processes [6, 5]. Nonetheless, the resulting theory of session types suffers from a few shortcomings. First of all, the semantics of the external choice is a bit involved because in some contexts it is indistinguishable from that of the internal choice: the typical example, which is also one of the pivotal laws

E. Pimentel, B. Venneri and J. Wells (Eds.): Workshop on
Intersection Types and Related Systems 2010 (ITRS 2010).
EPTCS 45, 2011, pp. 71–89, doi:10.4204/EPTCS.45.6

© Luca Padovani
This work is licensed under the
Creative Commons Attribution License.

of the *must* pre-order, is $a.T + a.S \eqsim a.(T \oplus S)$ (we write $\eqsim$ for the equivalence relation induced by $\lesssim$). As a direct consequence of this, the subtyping relation $\lesssim$ fails to be a pre-congruence. Indeed we have $a.b \lesssim a.b + b.c$ but $a.b + b.d \not\lesssim a.b + b.c + b.d \eqsim a.b + b.(c \oplus d)$. This poses practical problems (one has to characterize the contexts in which subtyping is safe) as well as theoretical ones ($\lesssim$ is harder to characterize axiomatically). Finally, recent developments of session type theories have shown a growing interest toward the definition of *meet* and *join* operators over session types [13], 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 propose a language of session types which uses intersection types and union types for modeling branching points. The idea is that when some channel is typed by the intersection type $\overline{a}.T \wedge \overline{b}.S$ this means that the channel has both type $\overline{a}.T$ and also type $\overline{b}.S$, namely 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 $T$ and $S$. Dually, when some channel is typed by the union type $a.T \vee b.S$ this means that the process does not precisely know the type of the channel, which may be either $a.T$ or $b.S$. 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 $a.T \vee a.S \eqsim a.(T \vee S)$.

In summary, we argue that intersection and union types are natural, type theoretic alternatives for internal and external choices, respectively. Furthermore, they allow us to develop a decidable theory of session types that are natively equipped with join and meet operators, and where the semantically defined subtyping relation is a pre-congruence.

**Structure of the paper.** We devote Section 2 to presenting a process algebra, so that we can formalize processes and 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 includes the description of an algorithm for deciding the subtyping relation, a type system for checking whether a process conforms to a given session type, as well as 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. For the sake of simplicity, in this paper we restrict ourselves to finite processes and finite types. Indeed, the relationship between branching operators and intersection and union types is independent of the fact that processes may or may not be infinite. On the contrary, dealing with infinite behaviors introduces some technical difficulties, briefly touched upon in Section 4, that we plan to address in a forthcoming and more comprehensive work. For the sake of readability, proofs and other technical details have been postponed to sections A and B.

## 2 Processes

Let us fix some notation: we let $a$, $b$, ... range over some set $\mathscr{N}$ of *action names* whose meaning is left unspecified; we let $P$, $Q$, ... range over *processes* and $\alpha$, $\beta$, ... range over *actions*. We distinguish *input actions* of the form $a$ from *output actions* of the form $\overline{a}$; we say that $\overline{\alpha}$ is the *co-action* of $\alpha$ where $\overline{\overline{a}} = a$. 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 [6, 9] without relabeling, restriction, and parallel composition. The terms **0** and **1** denote idle processes that perform no further action. The former is deadlocked, while the latter represents a successfully terminated interaction (since we are going to give processes a testing semantics, we prefer denoting success by means of a dedicated

Table 1: Syntax of processes.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Process** | $P$ | $::=$ | $\mathbf{0}$ | (deadlock) | **Action** | $\alpha$ | $::=$ | $a$ (input) |
| | | $\mid$ | $\mathbf{1}$ | (termination) | | | $\mid$ | $\overline{a}$ (output) |
| | | $\mid$ | $\alpha.P$ | (prefix) | | | | |
| | | $\mid$ | $P \oplus P$ | (internal choice) | | | | |
| | | $\mid$ | $P + P$ | (external choice) | | | | |

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

$$\text{(R1)}\quad \mathbf{1} \xrightarrow{\checkmark} \mathbf{1} \qquad \text{(R2)}\quad \alpha.P \xrightarrow{\alpha} P \qquad \text{(R3)}\quad P \oplus Q \longrightarrow P$$

$$\text{(R4)}\quad \frac{P \longrightarrow P'}{P + Q \longrightarrow P' + Q} \qquad \text{(R5)}\quad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \text{(R6)}\quad \frac{P \xrightarrow{\overline{a}} P'}{P + Q \longrightarrow \overline{a}.P'}$$

term $\mathbf{1}$ rather than a special action as in other theories [5]). The term $\alpha.P$ denotes a process that performs the action $\alpha$ and then continues as $P$. The term $P \oplus Q$ 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 $Q$. In the following we will usually omit trailing $\mathbf{1}$'s and write, for example, $a.\overline{b}$ instead of $a.\overline{b}.\mathbf{1}$. We will also write $\mathscr{P}$ for the set of all processes.

The formal meaning of processes is given by a transition system, defined in Table 2 (symmetric rules have been omitted). The system consists of two relations, an unlabelled one $\longrightarrow$ and a labelled one $\xrightarrow{\mu}$ where $\mu$ is a *label* is an element of $\mathscr{N} \cup \overline{\mathscr{N}} \cup \{\checkmark\}$ and $\checkmark \notin \mathscr{N} \cup \overline{\mathscr{N}}$ is a flag denoting successful termination. We extend the $\overline{\cdot}$ involution to labels so that $\overline{\checkmark} = \checkmark$ and to sets of labels $\mathrm{A}$ so that $\overline{\mathrm{A}} = \{\overline{\mu} \mid \mu \in \mathrm{A}\}$. Intuitively $\longrightarrow$ represents *internal*, *invisible* transitions of a process, while $\xrightarrow{\mu}$ represents *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; rule (R2) states that a process $\alpha.P$ may execute the action $\alpha$ and reduce to $P$; rule (R3) (and the symmetric one) states that a process $P \oplus Q$ internally decides to reduce to either $P$ or $Q$; rule (R4) (and the symmetric one) 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 [6] from which out process language is inspired. Rule (R5) (and the symmetric one) states that an external choice offers any action that is offered by either branch of the choice. Rule (R6) and its symmetric 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. This is practically justified in our setting as we aim at modelling dyadic sessions. In the following we will sometimes use the following notation: we

write $\Longrightarrow$ for the reflexive and transitive closure of $\longrightarrow$; we let $\overset{\mu}{\Longrightarrow}$ be the composition $\Longrightarrow\overset{\mu}{\longrightarrow}\Longrightarrow$; we write $P \not\longrightarrow$ if there is no $P'$ such that $P \longrightarrow P'$; we write $P \overset{\mu}{\Longrightarrow}$ if $P \overset{\mu}{\Longrightarrow} P'$ for some $P'$; let $\mathsf{init}(P) \overset{\text{def}}{=} \{\mu \mid P \overset{\mu}{\Longrightarrow}\}$.

The next and final step is to describe how two processes "complete each other", in the sense that they interact 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 orthogonality relation between processes:

**Definition 2.1** (orthogonal processes)**.** *Let $\longrightarrow$ be the smallest relation between systems $P \mid Q$ of two processes such that:*

$$\frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q} \qquad \frac{Q \longrightarrow Q'}{P \mid Q \longrightarrow P \mid Q'} \qquad \frac{P \overset{\overline{\alpha}}{\longrightarrow} P' \qquad Q \overset{\alpha}{\longrightarrow} Q'}{P \mid Q \longrightarrow P' \mid Q'}$$

*and let $\Longrightarrow$ be the reflexive, transitive closure of $\longrightarrow$. We write $P \mid Q \not\longrightarrow$ if there are no $P'$ and $Q'$ such that $P \mid Q \longrightarrow P' \mid Q'$. We say that $P$ and $Q$ are* orthogonal*, notation $P \perp Q$, if $P \mid Q \Longrightarrow P' \mid Q' \not\longrightarrow$ implies $P' \overset{\checkmark}{\longrightarrow}$ and $Q' \overset{\checkmark}{\longrightarrow}$.* ■

As an example, consider the process $P \overset{\text{def}}{=} \overline{a}.(a+b)$. Then $a.\overline{a}$, $a.\overline{b}$, $a.(\overline{a} \oplus \overline{b})$ are all orthogonal to $P$. The processes $a$ and $P$ are *not* orthogonal because $a \mid P \longrightarrow \mathbf{1} \mid a+b \not\longrightarrow$ and $a+b \overset{\checkmark}{\not\longrightarrow}$ (both processes must be in a successfully terminated state when they reach a stable configuration). Also $a.(\overline{a} \oplus \overline{c})$ and $P$ are not orthogonal because $a.(\overline{a} \oplus \overline{c}) \mid P \longrightarrow \overline{a} \oplus \overline{c} \mid a+b \longrightarrow \overline{c} \mid a+b \not\longrightarrow$.

Orthogonality provides us with a notion of "test" that we can use for discriminating processes, in the spirit of the testing framework [5]. Informally, when $P \perp Q$ we can see $Q$ as a test that $P$ succeeds to pass (since orthogonality is symmetric, we can also reason the other way around and see $P$ as a test for $Q$). Equivalently, we can see $Q$ as a context that completes $P$. Then, we can say that two processes are equivalent if they pass the same tests, or if they are completed by the same contexts. In fact, it makes sense to interpret processes as the set of tests they pass and to define a pre-order between processes, which we call *refinement*, as the inclusion of their corresponding interpretations.

**Definition 2.2** (process interpretation and refinement)**.** *Let $[\![P]\!] \overset{\text{def}}{=} \{Q \in \mathscr{P} \mid P \perp Q\}$. We say that $Q$ is a* refinement *of $P$, notation $P \lesssim Q$, if and only if $[\![P]\!] \subseteq [\![Q]\!]$. We write $\simeq$ for the equivalence relation induced by $\lesssim$, namely $\simeq \;=\; \lesssim \cap \lesssim^{-1}$.* ■

Intuitively, $Q$ is a refinement of $P$ if any test that $P$ passes is also passed by $Q$. Therefore, it is safe to replace $P$ with $Q$ as any context in which $P$ operates correctly will continue to do so also with $Q$. The equational theory induced by refinement is closely related to the *must* testing pre-order [5]. In particular, we have $P \oplus Q \lesssim P$ since $[\![P \oplus Q]\!] = [\![P]\!] \cap [\![Q]\!]$. This equivalence lets us appreciate the fact that the internal choice operator does correspond to an intersection when we interpret processes as the sets of their orthogonals. Alas, under this interpretation the external choice operator does *not* correspond to a union, for three reasons:

- There can be processes in $[\![P+Q]\!]$ that are not contained in $[\![P]\!] \cup [\![Q]\!]$. For example, $\overline{a} \oplus \overline{b} \in [\![a+b]\!] \setminus [\![a]\!] \cup [\![b]\!]$. This is fairly common in every framework that accounts for non-deterministic entities. In our case, $\overline{a} \oplus \overline{b}$ is orthogonal to $a+b$, but not to $a$ or $b$ alone.
- Sometimes $[\![P+Q]\!] = [\![P \oplus Q]\!] = [\![P]\!] \cap [\![Q]\!]$, namely the external choice can be internal choice in disguise. For example, we have $a.\overline{a} + a.\overline{b} \simeq a.\overline{a} \oplus a.\overline{b} \simeq a.(\overline{a} \oplus \overline{b})$. The problem is that both

Table 3: Syntax of session types.

| | | | | |
|---|---|---|---|---|
| **Session type** | $T$ | $::=$ | $\mathbb{0}$ | (bottom) |
| | | $\mid$ | $\mathbb{1}$ | (top) |
| | | $\mid$ | $\mathsf{end}$ | (termination) |
| | | $\mid$ | $\alpha.T$ | (prefix) |
| | | $\mid$ | $T \wedge T$ | (intersection) |
| | | $\mid$ | $T \vee T$ | (union) |

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 $a.\overline{a} + a.\overline{b}$ does not offer an external choice, but is actually performing an internal one. A different instance of this phenomenon occurs when both branches of an external choice are guarded by output actions, because of rule (R6). For example, we have $\overline{a} + \overline{b} \approx \overline{a} \oplus \overline{b}$.

- The fact that output actions can preempt branches of external choices can make such branches useless. For example $\overline{a} + b \approx \overline{a} + \mathbf{1} \approx \overline{a}$, since $\overline{a} + P \longrightarrow \overline{a}$ by rule (R6).

A direct consequence of these subtleties related with the external choice is that refinement fails to be a pre-congruence. In particular, we are now able to justify the (in)equivalences $a.b + b.d \not\lesssim a.b + b.c + b.d \approx a.b + b.(c \oplus d)$ that we have anticipated in the introduction.

Observe that there are pathological processes that are intrinsically flawed and cannot interact correctly with any other process. For example, $a \oplus b$ has no orthogonals since it is not possible to know which message, $a$ or $b$, it is ready to receive. As another example the process $P = a \oplus \overline{b}$ has no orthogonals: no process interacting with it can send an $a$ message, since $P \longrightarrow \overline{b}$; at the same time, a process waiting for the $b$ message from $P$ may starve forever since $P \longrightarrow a$.

# 3 Session Types

In this section we introduce our language of session types, we study their semantics, and we provide a subtyping algorithm and a type system for checking processes against session types.

## 3.1 Syntax

We let $T$, $S$, ... range over *session types*, which are defined by the grammar in Table 3. The types $\mathbb{0}$ and $\mathbb{1}$ characterize channels which cannot be successfully used for any interaction. We postpone a more detailed discussion about $\mathbb{0}$ and $\mathbb{1}$ when we will formally define their semantics. For the time being, it suffices to say that $\mathbb{0}$ and $\mathbb{1}$ represent the largest and smallest element in the lattice of session types we are about to define. The type $\mathsf{end}$ denotes channels on which no further action is possible. There is a fundamental distinction between $\mathsf{end}$ and the two types $\mathbb{0}$ and $\mathbb{1}$: $\mathsf{end}$ denotes a successfully terminated interaction, while $\mathbb{0}$ and $\mathbb{1}$ denote the impossibility to carry on any interaction; the type $\alpha.T$ denotes channels on which it is possible to perform an action $\alpha$. Actions are the same ones that occur within processes, but the point of view is slightly different: a process *executes* an action, while a session type indicates the possibility or the obligation for a process to execute an action. We will appreciate more concretely this difference in Section 3.4, where we will see that the same process can be successfully checked against different session types. The type $T \wedge S$ denotes channels that have *both* types $T$ and $S$.

For example $\overline{a}.\mathsf{end} \wedge \overline{b}.\mathsf{end}$ denotes a channel that has both type $\overline{a}.\mathsf{end}$ and also type $\overline{b}.\mathsf{end}$, namely it can be used for sending both messages $a$ and $b$. Finally, the type $T \vee S$ denotes channels that either have type $T$ or $S$. For instance the type $a.\mathsf{end} \vee b.\mathsf{end}$ associated with a channel means that a process using that channel 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 $a.\mathsf{end}$ or $b.\mathsf{end}$.[1] To avoid clutter, in the following we will omit trailing $\mathsf{end}$'s and write, for instance, $\overline{a} \wedge \overline{b}$ instead of $\overline{a}.\mathsf{end} \wedge \overline{b}.\mathsf{end}$ when this generates no ambiguity with the syntax of processes.

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 similar roles: 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. No such correspondence exists between $+$ and $\vee$. For instance, consider $P = a.b.\overline{a} + a.c.\overline{b}$ and $T = a.b.\overline{a} \vee a.c.\overline{b}$. The external choice in $P$ is guarded by the same action $a$, meaning that after performing action $a$ the process may reduce to either $b.\overline{a}$ or to $c.\overline{b}$, the choice being nondeterministic. As we have already remarked at the end of Section 2, one can show that $P$ is equivalent to $a.(b.\overline{a} \oplus c.\overline{b})$, where the nondeterministic choice between the two residual branches is explicit. The session type $T$, on the other hand, tells us something different: we do not know whether the channel we are using has type $a.b.\overline{a}$ or $a.c.\overline{b}$ 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 $b.\overline{a} \vee c.\overline{b}$. 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 $T$ specify quite different behaviors, and in fact while $T$ is perfectly reasonable, in the sense that there are processes that conform to $T$ and that can correctly interact with corresponding orthogonal processes, the reader may easily verify that $P$ has no orthogonals.

## 3.2 Semantics

Intuitively we want to define the semantics $[\![T]\!]$ of a session type $T$ as a set of processes, so that session types can be related by comparing the corresponding interpretations pretty much as we did for processes (Definition 2.2). To assist the reader with this intuition, consider the scenario depicted below


where the notation $T \vdash P$ means that $P$, which we will think of as the "server", is using the end point of channel $\mathsf{c}$ according to the session type $T$. We write $T \vdash P$ instead of $\mathsf{c} : T \vdash P$ since we assume that $P$ acts on one channel only. The idea is that the interpretation of $T$ is the set of "client" processes $Q$ that can interact correctly with $P$ when placed at the other end point of the channel $\mathsf{c}$.

Before we address the formal definition of $[\![T]\!]$ we must realize that not every set of processes makes sense when interpreted in this way:

- if a server is able to interact correctly with all of the clients in the set $X = \{\overline{a}, \overline{b}\}$, then it is also able to interact correctly with $\overline{a} \oplus \overline{b}$;
- no server is able to interact correctly with *all* of the clients in the set $Y = \{\overline{a}, b\}$ because this server would have to perform both an input on $a$ and an output on $b$ at the same time.

[1] 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.

We conclude that neither $X$ nor $Y$ above are *closed* sets of processes that can serve as proper denotations of a session type: $X$ and $X \cup \{\overline{a} \oplus \overline{b}\}$ are indistinguishable because every server $P$ that includes $X$ in its interpretation includes also $X \cup \{\overline{a} \oplus \overline{b}\}$; $Y$ and $\mathscr{P}$ are indistinguishable because there is no server that includes $Y$ in its interpretation just as there is no server that includes the whole $\mathscr{P}$ in its interpretation. We therefore need a closure operation over sets of processes, which we define in terms of *orthogonal sets*, defined as follows:

**Definition 3.1** (orthogonal set)**.** *Let $X \subseteq \mathscr{P}$. Then $X^\perp \stackrel{\text{def}}{=} \{P \in \mathscr{P} \mid X \subseteq [\![P]\!]\}$.* ■

Intuitively, the orthogonal of some set of processes $X$ is the set of those processes that include $X$ in their interpretation. If we go back to the problematic sets of processes described earlier, we have $X^\perp = \{a+b, a+b+c, a+b+c+d, \dots\}$ and $Y^\perp = \emptyset$. Clearly the orthogonal of a set $X$ flips the perspective, in the sense that if $X$ is a set of "clients", then $X^\perp$ is the set of "servers" of those clients. Therefore, we define the closure as the *bi-orthogonal* $(\cdot)^{\perp\perp}$. For instance we have $X^{\perp\perp} = \{\overline{a}, \overline{b}, \overline{a} \oplus \overline{b}, \dots\}$ and $Y^{\perp\perp} = \mathscr{P}$. We say that a set $X$ of processes is *closed* if it is equal to its closure, namely if $X = X^{\perp\perp}$. The fact that $(\cdot)^{\perp\perp}$ is indeed a closure operator is formalized by the following result:

**Proposition 3.1.** *The bi-orthogonal is a closure, namely it is extensive, monotonic, and idempotent:*

1. *$X \subseteq X^{\perp\perp}$;*
2. *$X \subseteq Y$ implies $X^{\perp\perp} \subseteq Y^{\perp\perp}$;*
3. *$X^{\perp\perp} = X^{\perp\perp\perp\perp}$.*

*Proof.* Observe that $X^\perp = \{P \in \mathscr{P} \mid \forall Q \in X : P \perp Q\}$. Then $((\cdot)^\perp, (\cdot)^\perp)$ is a Galois connection (more precisely, a polarity) between the posets $\langle 2^{\mathscr{P}}, \subseteq \rangle$ and $\langle 2^{\mathscr{P}}, \supseteq \rangle$. Then it is a known fact that $(\cdot)^{\perp\perp} = (\cdot)^\perp \circ (\cdot)^\perp$ is a closure operator on the poset $\langle 2^{\mathscr{P}}, \subseteq \rangle$. □

Then we define the interpretation of session types in terms of closures of sets of processes, where we interpret $\wedge$ and $\vee$ as set-theoretic intersections and unions.

**Definition 3.2** (session type semantics)**.** *The semantics of a session type is inductively defined by the following equations:*

$$\begin{array}{rcl} [\![\mathbb{0}]\!] & = & \emptyset \\ [\![\mathbb{1}]\!] & = & \mathscr{P} \\ [\![\mathsf{end}]\!] & = & \{\mathbf{1}\}^{\perp\perp} \\ [\![\alpha.T]\!] & = & \{\overline{\alpha}.P \mid P \in [\![T]\!]\}^{\perp\perp} \\ [\![T_1 \wedge T_2]\!] & = & [\![T_1]\!] \cap [\![T_2]\!] \\ [\![T_1 \vee T_2]\!] & = & ([\![T_1]\!] \cup [\![T_2]\!])^{\perp\perp} \end{array}$$

As we comment on the definition of $[\![\cdot]\!]$, it is useful to think of $[\![T]\!]$ as of the set of clients that a server using a channel with type $T$ must be able to satisfy. Since $\mathbb{0}$ denotes the empty set of clients, a channel typed by $\mathbb{0}$ is the easiest to use for a server, for the server is not required to satisfy any process. Dually, a channel typed by $\mathbb{1}$ is the hardest to use, for the server is required to satify any process. As this is impossible to achieve (there is no process that is dual of every process in $\mathscr{P}$), no server can effectively use a channel typed by $\mathbb{1}$. From a type-theoretic point of view, $\mathbb{0}$ and $\mathbb{1}$ represent two dual notions of emptyness: $\mathbb{0}$ means absence of clients, $\mathbb{1}$ means absence of servers. Later on we will see that any session type different from $\mathbb{0}$ and $\mathbb{1}$ is *inhabited*, in the sense that it admits at least one client and at least one server. A channel typed by $\mathsf{end}$ represents those clients that are satisfied even if they do not receive any further message. The process $\mathbf{1}$ clearly is a client of $\mathsf{end}$, but it's not the only one: any process that guarantees the ✓ action is a client of $\mathsf{end}$. Hence we have $[\![\mathsf{end}]\!] = \{\mathbf{1}, \mathbf{1}+a, \mathbf{1}+a+b, \dots\}$.

In particular, no process that is immediately able to emit an output is included in this set. Regarding the session type $\alpha.T$, its clients are all those processes that perform the co-action $\overline{\alpha}$ and whose continuation after $\alpha$ is in $[\![T]\!]$. If $\alpha$ is some input action $a$ then any process in $[\![\alpha.T]\!]$ sends $\overline{a}$ (and only $\overline{a}$), whereas if $\alpha$ is some output action $\overline{a}$ then any process in $[\![\alpha.T]\!]$ guarantees the input action $a$. For example we have $a \in [\![\overline{a}.\mathsf{end}]\!]$ and $a+b \in [\![\overline{a}.\mathsf{end}]\!]$ but $\overline{a} \oplus \overline{b} \notin [\![a.\mathsf{end}]\!]$. Therefore, a server using a channel typed by $\alpha.T$ is required to provide action $\alpha$ and to continue the interaction as specified by $T$. The intersection type $T_1 \wedge T_2$ denotes those channels that have both type $T_1$ and type $T_2$. Therefore the servers using these channels have the freedom to use them according to either $T_1$ or $T_2$. That is why the clients of $T_1 \wedge T_2$ must be clients of both $T_1$ and $T_2$. The union type $T_1 \vee T_2$ can be explained in a dual way with respect to the intersection. In this case, the server is unsure whether the channel has type $T_1$ or $T_2$ and consequently it must be able to satisfy (at least) all the clients of $T_1$ and all the clients of $T_2$ as well. Overall we see that intersections and unions of session types match in a quite natural way their set-theoretic interpretation. However, note that $[\![T_1 \wedge T_2]\!] = [\![T_1]\!] \cap [\![T_2]\!]$ whereas in general we have $[\![T_1 \vee T_2]\!] \supseteq [\![T_1]\!] \cup [\![T_2]\!]$. For example, $\overline{a} \oplus \overline{b} \in [\![a.\mathsf{end} \vee b.\mathsf{end}]\!] \setminus ([\![a.\mathsf{end}]\!] \cup [\![b.\mathsf{end}]\!])$. There is no need to use the closure operator on $[\![T_1]\!] \cap [\![T_2]\!]$ since it can be shown that this set is already closed.

We use $[\![\cdot]\!]$ for comparing session types. In particular we say that $T$ is a *subtype* of $S$ when $T$'s clients are included in $S$'s clients:

**Definition 3.3** (subtype)**.** *We say that $T_1$ is a* subtype *of $T_2$, written $T_1 \lesssim T_2$, if $[\![T_1]\!] \subseteq [\![T_2]\!]$. We write $\approx$ for the equivalence relation induced by $\lesssim$, namely $\approx = \lesssim \cap \lesssim^{-1}$.* ■

Unlike the refinement relation, subtyping turns out to be a pre-congruence with respect to all the operators of the session type language.

**Proposition 3.2.** *$\lesssim$ is a pre-congruence.*

*Proof.* Immediate from the definition of $\lesssim$ and Proposition 3.1(2). □

Equally trivial is the fact that $\wedge$ and $\vee$ provide us with a native way of respectively computing the greatest lower bound and the least upper bound of two session types. As regards $\wedge$, this is obvious since $[\![T_1 \wedge T_2]\!] = [\![T_1]\!] \cap [\![T_2]\!]$ by definition. For $\vee$, it suffices to observe that $T_1 \lesssim S$ and $T_2 \lesssim S$ implies $[\![T_1]\!] \cup [\![T_2]\!] \subseteq [\![S]\!]$. Since $([\![T_1]\!] \cup [\![T_2]\!])^{\perp\perp}$ is the smallest closed set that includes $[\![T_1]\!] \cup [\![T_2]\!]$ and since $[\![S]\!]$ is closed, we conclude $[\![T_1 \vee T_2]\!] = ([\![T_1]\!] \cup [\![T_2]\!])^{\perp\perp} \subseteq [\![S]\!]$, namely $T_1 \vee T_2 \lesssim S$. 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 *[12, 2] are abstract descriptions of interactions between two or more participants from a neutral point of view. For example, the global type*

$$\mathtt{A} \xrightarrow{a} \mathtt{B};\mathtt{A} \xrightarrow{b} \mathtt{B} \,\Box\, \mathtt{A} \xrightarrow{a} \mathtt{B};\mathtt{A} \xrightarrow{c} \mathtt{B}$$

*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* $\mathtt{A} \xrightarrow{a} \mathtt{B}$ *indicates that* A *sends an 'a' message to* B*. Actions can be composed in sequences (with* ;*) and in alternative paths (with* □*). 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* □ *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 $\overline{a}.\overline{b}$ on the l.h.s. and $\overline{a}.\overline{c}$ 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* $\overline{a}.\overline{b} \wedge \overline{a}.\overline{c} \simeq \overline{a}.(\overline{b} \wedge \overline{c})$. *Since* B *is the only receiver, it must be prepared to receive the messages from* A *regardless of which messages* A *decides to send. Therefore, the correct projection of the global type on* B *is* $a.b \vee a.c \simeq a.(b \vee c)$*, 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. ♦

As we have anticipated, for a session type to make sense, its interpretation must be different from both $\emptyset$ and $\mathscr{P}$. This condition roughly corresponds to non-emptyness: a standard "value" type is inhabited if there exists one value of that type; a session type is inhabited if it has at least one server and at least one client. This explains why there are two distinct "empty" session types.

**Definition 3.4** (viable session type)**.** *We say that the session type* $T$ *is* viable *if* $T \not\simeq \mathbb{0}, \mathbb{1}$. ■

Viability is a necessary and sufficient condition for $T$ to be implementable: if $T \not\simeq \mathbb{0}$ take any $P \in [\![T]\!]$. From the hypothesis $T \not\simeq \mathbb{1}$ and the fact that $[\![T]\!]$ is closed we also know that $[\![T]\!]^\perp \neq \emptyset$, because $[\![T]\!]^\perp = \emptyset$ implies $[\![T]\!]^{\perp\perp} = \mathscr{P}$. Hence there exists $Q \in [\![T]\!]^\perp$. By definition of orthogonal set we conclude $P \perp Q$. This discussion about viability emphasizes the importance of the orthogonal operation since the sets $[\![T]\!]$ and $[\![T]\!]^\perp$ contain precisely those processes that interact correctly via a channel typed by $T$. We conclude this section by showing that the orthogonal operator over sets of processes corresponds to a syntactic duality operation over session types.

**Theorem 3.1** (dual session type)**.** *The* dual *of a session type* $T$ *is the session type* $\overline{T}$ *obtained from* $T$ *by turning every* $\mathbb{0}$ *into* $\mathbb{1}$*, every* $\mathbb{1}$ *into* $\mathbb{0}$*, every action* $\alpha$ *into the corresponding co-action* $\overline{\alpha}$*, every* $\wedge$ *into* $\vee$*, and every* $\vee$ *into* $\wedge$*. Inductively:*

$$\begin{array}{rcl} \overline{\mathbb{0}} & = & \mathbb{1} \\ \overline{\mathbb{1}} & = & \mathbb{0} \\ \overline{\mathsf{end}} & = & \mathsf{end} \\ \overline{\alpha.T} & = & \overline{\alpha}.\overline{T} \\ \overline{T_1 \wedge T_2} & = & \overline{T}_1 \vee \overline{T}_2 \\ \overline{T_1 \vee T_2} & = & \overline{T}_1 \wedge \overline{T}_2 \end{array}$$

*Then* $[\![\overline{T}]\!] = [\![T]\!]^\perp$.

## 3.3 Subtyping Algorithm

In this section we define an algorithm for deciding the subtyping relation. Since the interpretation of a session type is usually an infinite set of processes, we cannot hope to derive a brute force algorithm that is based directly on Definition 3.3. Fortunately, session types admit a particularly simple and intuitive normal form. Therefore, we split the decision algorithm in two parts: first we provide an effective procedure for rewriting every session type into an equivalent normal form, which happens to be unique up to commutativity and associativity of intersections and unions. Then, we provide a syntax-directed algorithm that decides the subtyping relation between session types in normal form. In what follows we will use $n$-ary intersections and unions of the form $\bigwedge_{i\in\{1,\ldots,n\}} T_i$ and $\bigvee_{i\in\{1,\ldots,n\}} T_i$ in place of $T_1 \wedge \cdots \wedge T_n$ and $T_1 \vee \cdots \vee T_n$, respectively; as usual, we let $\bigwedge_{i\in\emptyset} T_i = \mathbb{1}$ and $\bigvee_{i\in\emptyset} T_i = \mathbb{0}$ by definition. We will also write $T\{\wedge S\}_\phi$ to indicate that the $\wedge S$ part is present only when $\phi$ holds; similarly for $T\{\vee S\}_\phi$.

**Definition 3.5** (normal form)**.** *We say that a session type* $T$ *is in* normal form *if either*

$$T \equiv \bigwedge_{a\in \mathrm{A}} \overline{a}.T_a\{\wedge \mathsf{end}\}_{\checkmark\in \mathrm{A}} \qquad \textit{or} \qquad T \equiv \bigvee_{a\in \mathrm{A}} a.T_a\{\vee \mathsf{end}\}_{\checkmark\in \mathrm{A}}$$

*and* $T_a$ *is viable and in normal form for every* $a \in \mathrm{A}$.

Table 4: Simplification laws (symmetric and dual laws omitted).

(E-PREFIX) $\alpha.\mathbb{0} = \mathbb{0}$ (E-BOTTOM) $\mathbb{0} \wedge T = \mathbb{0}$ (E-TOP) $\mathbb{1} \wedge T = T$ (E-DIST) $\alpha.T \wedge \alpha.S = \alpha.(T \wedge S)$

(E-INPUT-END) $\dfrac{T_a \text{ viable}^{(a \in \mathrm{A})}}{\left(\bigvee_{a \in \mathrm{A}} a.T_a\right) \wedge \mathsf{end} = \mathbb{0}}$ (E-INPUT-OUTPUT) $\dfrac{T_a \text{ viable}^{(a \in \mathrm{A})} \quad S \text{ viable}}{\left(\bigvee_{a \in \mathrm{A}} a.T_a\right) \wedge \overline{b}.S = \mathbb{0}}$ (E-INPUT-OUTPUT-END) $\dfrac{T_a \text{ viable}^{(a \in \mathrm{A})} \quad S \text{ viable}}{\left(\bigvee_{a \in \mathrm{A}} a.T_a \vee \mathsf{end}\right) \wedge \overline{b}.S = \overline{b}.S \wedge \mathsf{end}}$

(E-INPUT-INPUT) $\left(\bigvee_{a \in \mathrm{A}} a.T_a\{\vee \mathsf{end}\}_{\checkmark \in \mathrm{A}}\right) \wedge \left(\bigvee_{b \in \mathrm{B}} b.S_b\{\vee \mathsf{end}\}_{\checkmark \in \mathrm{B}}\right) = \bigvee_{a \in \mathrm{A} \cap \mathrm{B}} a.(T_a \wedge S_a)\{\vee \mathsf{end}\}_{\checkmark \in \mathrm{A} \cap \mathrm{B}}$

A process using a channel whose associated session type is $\bigwedge_{a \in \mathrm{A}} \overline{a}.T_a\{\wedge \mathsf{end}\}_{\checkmark \in \mathrm{A}}$ may send any message $a \in \mathrm{A}$ and it may decide to terminate if $\checkmark \in \mathrm{A}$. After sending a message $a$, the process must continue using the channel as specified by $T_a$. In a dual fashion, a process using a channel whose associated session type is $\bigvee_{a \in \mathrm{A}} a.T_a\{\vee \mathsf{end}\}_{\checkmark \in \mathrm{A}}$ must be ready to receive any message $a \in \mathrm{A}$ and it must also be ready to terminate immediately if no such message is received and $\checkmark \in \mathrm{A}$. In case a message $a$ is received, the process must continue using the channel as specified by $T_a$.

The simplicity of normal forms is due to the fact that some behaviors (like sending a message and receiving a message) are incompatible, in the sense that their combination (intersection or union) yields non-viable session types. Table 4 presents a set of laws that are used (from left to right) as basic simplification steps in the computation of the normal form (symmetric and dual laws are omitted). Laws (E-PREFIX), (E-BOTTOM), and (E-TOP) state that non-viable types absorb prefixes and that $\mathbb{0}$ and $\mathbb{1}$ are respectively neutral for $\vee$ and $\wedge$, as expected. Law (E-DIST) shows that common actions can be factored while preserving the combining operator. In particular, the dual law $\alpha.T \vee \alpha.S \simeq \alpha.(T \vee S)$ distinguishes subtyping from refinement and from the *must* pre-order, where the law $\alpha.P + \alpha.Q \simeq \alpha.(P \oplus Q)$ holds. Rules (E-INPUT-END) and (E-INPUT-OUTPUT) show that no client that sends a message $a \in \mathrm{A}$ can be satisfied by a server that may decide to terminate the interaction or to send a message. This is because the action of sending a message is irrevocable (see rule (R6) in the transition system of processes). Rule (E-INPUT-OUTPUT-END) shows that among the clients that either send a message $a \in \mathrm{A}$ or terminate are those that can also receive message $b$. Finally, rule (E-INPUT-INPUT) shows that the clients of a server will send only messages that can surely be received by the server. For example, $(a \vee b \vee c) \wedge (b \vee c \vee d) \simeq b \vee c$. The dual law concerns messages that can be sent by the server. Thus $(\overline{a} \wedge \overline{b} \wedge \overline{c}) \vee (\overline{b} \wedge \overline{c} \wedge \overline{d}) \simeq \overline{b} \wedge \overline{c}$: if the server is unsure whether the type of the channel is $\overline{a} \wedge \overline{b} \wedge \overline{c}$ or $\overline{b} \wedge \overline{c} \wedge \overline{d}$, then it can only send those messages that can travel along the channel in both cases.

**Lemma 3.1.** *The laws in Table 4 are sound.*

The simplification laws, and the axiomatization of $\lesssim$ that we are about to present, would be simpler if one could prove that $\wedge$ and $\vee$ distribute over each other. We conjecture that the lattice of closed sets of processes ordered by set inclusion is indeed distributive (in the process language, the internal and external choices distribute over each other), but the proof appears to be non-trivial.

**Lemma 3.2** (normal form)**.** *For every session type $T$ there exists $S$ in normal form such that $T \simeq S$.*

The proof of the normal form lemma is constructive and provides an effective procedure for rewriting every session type in its normal form using the laws in Table 4. What remains to do now is to provide the subtyping algorithm for session types in normal form.

Table 5: Subtyping algorithm.

(S-BOTTOM) $\mathbb{0} \leqslant \bigwedge_{a\in A} \overline{a}.T_a\{\wedge \mathsf{end}\}_{\checkmark\in A}$

(S-TOP) $\bigvee_{a\in A} a.T_a\{\vee \mathsf{end}\}_{\checkmark\in A} \leqslant \mathbb{1}$

(S-END) $\bigwedge_{a\in A} \overline{a}.T_a \wedge \mathsf{end} \leqslant \bigvee_{b\in B} b.S_b \vee \mathsf{end}$

(S-INPUT) $$\frac{A \subseteq B \qquad T_a \leqslant S_a{}^{(a\in A)}}{\bigvee_{a\in A} a.T_a\{\vee \mathsf{end}\}_{\checkmark\in A} \leqslant \bigvee_{b\in B} b.S_b\{\vee \mathsf{end}\}_{\checkmark\in B}}$$

(S-OUTPUT) $$\frac{B \subseteq A \qquad T_a \leqslant S_a{}^{(a\in B)}}{\bigwedge_{a\in A} \overline{a}.T_a\{\wedge \mathsf{end}\}_{\checkmark\in A} \leqslant \bigwedge_{b\in B} \overline{b}.S_b\{\wedge \mathsf{end}\}_{\checkmark\in B}}$$

Table 6: Type checking rules.

(T-NIL) $$\frac{}{\mathbb{0} \vdash \mathbf{0}}$$

(T-END) $$\frac{}{\mathsf{end} \vdash \mathbf{1}}$$

(T-SEND) $$\frac{T \vdash P}{\overline{a}.T \vdash \overline{a}.P}$$

(T-RECEIVE) $$\frac{T_{a_i} \vdash P_i{}^{(i\in I)}}{\bigvee_{i\in I} a_i.T_{a_i} \vdash \sum_{i\in I} a_i.P_i}$$

(T-CHOICE) $$\frac{T \vdash P \qquad T \vdash Q}{T \vdash P \oplus Q}$$

(T-SUB) $$\frac{T \vdash P \qquad S \lesssim T}{S \vdash P}$$

**Definition 3.6** (algorithmic subtyping)**.** *Let $\leqslant$ be the least relation defined by axioms and rules in Table 5.*

Because of the interpretation of $\wedge$ and $\vee$ as respectively intersections and unions, the algorithm looks embarrassingly obvious although it states well-known properties of channel types. In particular, rule (S-INPUT) states that it is safe to replace a channel c having some input capabilities (B) with another one d having fewer input capabilities ($A \subseteq B$), because any process originally using c will be ready to handle any message $b \in B$. Dually, rule (S-OUTPUT) states that is safe to replace a channel c having some output capabilities (B) with another one d having greater output capabilities ($A \supseteq B$), since the process originally using c will exercise on d only a subset of the capabilities allowed on it. Observe that (S-OUTPUT) and (S-INPUT) are just specializations of the well-known laws $T \wedge S \leqslant T$ and $T \leqslant T \vee S$ concerning intersection and union types. Rules (S-BOTTOM) and (S-TOP) state obvious facts about $\mathbb{0}$ and $\mathbb{1}$ being the smallest and the largest session types, respectively. Observe that rule (S-INPUT) is the counterpart of rule (S-BOTTOM) when $A = \emptyset$ and the larger session type is a union. Dually, the rule (S-OUTPUT) is the counterpart of rule (S-TOP) when $B = \emptyset$ and the smallest session type is an intersection. Rule (S-END) is required for the algorithm to be complete: it basically states the reflexivity of $\leqslant$ on end.

The subtyping algorithm is correct and complete with respect to the set of session types in normal form:

**Theorem 3.2.** *Let $T$ and $S$ be in normal form. Then $T \lesssim S$ if and only if $T \leqslant S$.*

## 3.4 Type Checking

We conclude with the definition of a type checker to derive judgments of the form $T \vdash P$ meaning that $P$ is a well-typed process using a channel with type $T$. The type checker is defined by the axioms and rules in Table 6. We abbreviate $a_1.P_1 + \cdots + a_n.P_n$ with $\sum_{i\in\{1,\ldots,n\}} a_i.P_i$.

Because of the similarities between processes and session types, at first sight the type checker looks as stating a trivial correspondence between the two languages, but there are some lurking subtleties. Rules (T-NIL), (T-END), and (T-SEND) are indeed fairly obvious: the deadlocked server **0** can only use a channel typed by $\mathbb{0}$ since no client can interact with it; the terminated server **1** can use a channel typed

by end since it has successfully ended any interaction; the server $\overline{a}.P$ sending a message $a$ can use a channel typed by $\overline{a}.T$ if the continuation $P$ uses the channel according to $T$. Rule (T-RECEIVE) concerns servers waiting for a message from the set $\{a_i \mid i \in I\}$. Intuitively, these servers can use channels typed by $\bigvee_{i\in I} a_i.T_i$ where each continuation $P_i$ is well typed with respect to $T_i$. However, there is the possibility that two branches of the server are guarded by the same input action. Namely, it may be the case that $a_i = a_j$ for some $i, j \in I$ such that $i \neq j$. As we know, this translates into the server performing an internal choice on how to handle such a message, nondeterministically choosing between the continuations $P_i$ and $P_j$. Had we typed the server with respect to $\bigvee_{i\in I} a_i.T_i$, we would be stating that the server is capable of dealing with all the clients in the sets $[\![T_i \vee T_j]\!]$, which is not necessarily the case. Therefore, in order for this typing rule to be sound, we require that the continuations $P_i$ and $P_j$ of different branches guarded by the same input action $a_i = a_j$ must be typable with respect to the same type $T_{a_i} = T_{a_j}$. This way, no matter which continuation is selected, it will be well typed. Rule (T-CHOICE) presents a similar problem, since the server $P \oplus Q$ may independently reduce to either $P$ or $Q$. Therefore, we require both choices to be typable with respect to the same session type $T$. The attentive reader will have noticed a close relationship between this typing rule and standard type preservation results stating that (internal) reductions preserve the type: in this case, from the hypotheses $T \vdash P \oplus Q$ and either $P \oplus Q \longrightarrow P$ or $P \oplus Q \longrightarrow Q$ we easily deduce that the residual process is still well typed with respect to $T$. The last rule (T-SUB) is a standard subsumption rule, except that it deals with the type of the (implicit) channel used by the process and not with the type of the process itself. It states that if a process is well typed with respect to some session type $T$, then it is also well typed with respect to a smaller session type $S$. This is consistent with the intuition that it is safe to replace a value (in this case, a channel) with another one having a smaller type.

**Example 3.2.** *In the two derivations that follow, rule* (T-SUB) *is essential for rules* (T-RECEIVE) *and* (T-CHOICE) *to be applicable.*

$$\dfrac{\dfrac{\dfrac{\mathsf{end} \vdash \mathbf{1}}{\overline{a} \vdash \overline{a}} \quad \overline{a} \wedge \overline{b} \lesssim \overline{b}}{\overline{a} \wedge \overline{b} \vdash \overline{a}} \quad \dfrac{\dfrac{\mathsf{end} \vdash \mathbf{1}}{\overline{b} \vdash \overline{b}} \quad \overline{a} \wedge \overline{b} \lesssim \overline{b}}{\overline{a} \wedge \overline{b} \vdash \overline{b}}}{a.(\overline{a} \wedge \overline{b}) \vdash a.\overline{a} + a.\overline{b}} \qquad \dfrac{\dfrac{\dfrac{\mathsf{end} \vdash \mathbf{1}}{\overline{a} \vdash \overline{a}} \quad \overline{a} \wedge \overline{b} \lesssim \overline{b}}{\overline{a} \wedge \overline{b} \vdash \overline{a}} \quad \dfrac{\dfrac{\mathsf{end} \vdash \mathbf{1}}{\overline{b} \vdash \overline{b}} \quad \overline{a} \wedge \overline{b} \lesssim \overline{b}}{\overline{a} \wedge \overline{b} \vdash \overline{b}}}{\overline{a} \wedge \overline{b} \vdash \overline{a} \oplus \overline{b}}}{a.(\overline{a} \wedge \overline{b}) \vdash a.(\overline{a} \oplus \overline{b})}$$

*The fact that the two processes $a.\overline{a} + a.\overline{b}$ and $a.(\overline{a} \oplus \overline{b})$ are well typed with respect to the same type $a.(\overline{a} \wedge \overline{b})$ provides further evidence that they are equivalent, as informally argued in Section 1.* ♦

We conclude our study with a soundness result for the type system. If two processes are typed by dual session types, then they are orthogonal.

**Theorem 3.3.** *If $T \vdash P$ and $\overline{T} \vdash Q$, then $P \perp Q$.*

There is no hypothesis concerning the viability of $T$, but this is implied. The reader can easily verify that $T \vdash P$ implies $T \not\approx \mathbb{1}$, coherently with the observation that no process is able to satisfy *all* processes. As a consequence the hypotheses $T \vdash P$ and $\overline{T} \vdash Q$ are enough to ensure that $T$ and its dual are viable.

## 4 Concluding Remarks and Future Work

Previous formalizations of session types [3, 14, 1] are based on the observation that session types are behavioral types. As such, they are eligible for being studied by means of the numerous and well-developed techniques for process equivalence, and testing equivalence in particular [6, 5]. In this view

the different modalities in which actions are offered coincide with two known behavioral operators, the internal choice $\oplus$ and the external choice $+$. This approach, however natural and elegant, poses a few problems mostly due to the fact that the external choice is sometimes an internal choice in disguise: the language of session types may be difficult to understand to the programmer; the resulting subtyping relation is not a pre-congruence and is thus more difficult to use in practice; also, there are contexts where the computation of the greatest lower bound and of the least upper bound of session types arises naturally and these must be computed by means of meta-operators on session types [13].

In this work we propose an alternative language of session types which is not immediately related to some known process algebra. The basic idea is that the two choices can be naturally modeled by means of intersection and union types: the session type $T \wedge S$ describes a channel having *both* type $T$ *and* type $S$ and for this reason a process can freely use that channel as having either type; the session type $T \vee S$ describes a channel having *either* type $T$ *or* type $S$, therefore a process using that channel cannot make any assumption on it unless the exchanged messages provide enough information to disambiguate its type. The intersection and union operators are intuitive alternatives to internal and external choices, they provide a native mechanism to the computation of greatest lower bounds and least upper bounds, and the subtyping relation of the resulting theory turns out to be a pre-congruence.

It is worth noting that, in our theory, the semantics of session types solely depends on the process language, in particular on the adopted communication model and on the orthogonality relation. Any other concept or result is derived by these two. In this work we have adopted a partially asynchronous communication model, where output messages must be consumed before the sender can engage into any other activity, and a symmetric orthogonality relation where both processes involved in a communication must terminate successfully if the interaction reaches a stable state. These choices led us to rediscover a familiar theory of session types [8] but it is plausible to expect that different interesting theories can be developed by varying these two seminal notions. For example, using a truly asynchronous communication model, where an output action does not block subsequent actions, the relation $a.\overline{b} \lesssim \overline{b}.a$ would be sound because any "client" of $a.\overline{b}$ will eventually receive the $b$ message that the "server" of $\overline{b}.a$ sends ahead of time. Using a symmetric orthogonality relation might allow us to draw a closer comparison between our theory and more standard testing theories [5, 4], where the notion of "test" is asymmetric. We remark here just a few planned developments of our theory: first of all, we want to extend the presented framework to deal with possibly infinite session types. In principle this would amount to using a fix point operator for determining the semantics of recursive session types as sets of possibly infinite processes. However, the model presented in this work may need some further technical adjustments. To see why, consider the infinite session type determined by the equation $T = a.T$ which gives rise to the semantic equation $X = \{\overline{a}.P \mid P \in X\}^{\perp\perp}$. Both $\emptyset$ and $\mathscr{P}$ are solutions of the equation, meaning that the semantics of a session type may not be uniquely determined. At the same time, neither of $\emptyset$ and $\mathscr{P}$ is a satisfactory solution because they denote non-viable session types, while we would expect $[\![T]\!]$ to contain (recursive) processes that send an infinite number of $a$ messages. We plan to investigate whether the semantic model of types described in [15], which shares many properties with ours, can be used to give a proper semantics to infinite session types. The second extension to the presented framework is to consider non-atomic actions of the form $?t$ and $!t$ where $t$ is a *basic type* (such as `int`, `bool`, ...) and actions of the form $?T$ and $!T$ for describing delegations (the input and output of channels of type $T$). This will give rise to more interesting relations such as $!\texttt{int} \vee !\texttt{real} \eqsim !\texttt{int}$ assuming `int` is a subtype of `real`) and will allow us to compare more thoroughly our subtyping relation with the existing ones [8]. Finally, it looks like the presented approach can be easily extended to incroporate 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 [7].

**Acknowledgments.** I am grateful to the anonymous referees for the detailed comments and feedback on an earlier version of this paper. I wish to thank Mariangiola Dezani, Kohei Honda, and Nobuko Yoshida for the insightful discussions.

## References

[1] Franco Barbanera and Ugo de'Liguoro. Two notions of sub-behaviour for session-based client/server systems. In *Proceedings of PPDP'10*, pages 155–164. ACM, 2010.

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

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

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

[7] Simon Gay. Bounded polymorphism in session types. *MSCS*, 18(5):895–930, 2008.

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

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

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

[11] 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.

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

[13] 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.

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

[15] Jerome Vouillon and Paul-André Melliès. Semantic types: a fresh look at the ideal model for types. *SIGPLAN Notices*, 39(1):52–63, 2004.

## A Supplement to Section 2

In this section we solely introduce some handy notation related to processes that will be useful for the proofs in Section B. First we define two relations, that we dub "may" and "must", distinguishing the fact that a process *may* output some message or is always capable to (i.e., *must*) perform some input or output action, regardless of its internal transitions.

**Definition A.1** (may/must)**.** *Let* $\mu \in \overline{\mathscr{N}} \cup \{\checkmark\}$. *We say that P* may output $\mu$*, notation* $P \downarrow \mu$*, if* $P \stackrel{\mu}{\Longrightarrow}$. *Let* $\mu \in \mathscr{N} \cup \overline{\mathscr{N}} \cup \{\checkmark\}$. *We say that P* must $\mu$*, notation* $P \Downarrow \mu$*, if* $P \Longrightarrow P'$ *implies* $P' \stackrel{\mu}{\Longrightarrow}$. *We say that P* may converge*, notation* $P\downarrow$*, if* $P \Longrightarrow P'$ *implies* $P \downarrow \mu$ *for some* $\mu$*; we say that P* must converge*, notation* $P\Downarrow$*, if there exists* $\mu$ *such that* $P \Downarrow \mu$.

We will sometimes say that a process $P$ guarantees action $\mu$ if $P \Downarrow \mu$.

Then, we define the continuation of a process $P$ with respect to an action $\mu$ as the combination of all the possible residuals of $P$ after $\mu$. This differs from the relation $\xrightarrow{\mu}$ which relates $P$ with *one particular* (not necessarily unique) residual of $P$ after $\mu$.

**Definition A.2** (continuation)**.** *Let $P \stackrel{\mu}{\Longrightarrow}$. The* continuation *of $P$ with respect to $\mu$ is defined as $P(\mu) \stackrel{\text{def}}{=} \bigoplus_{P \Longrightarrow \xrightarrow{\mu} Q} Q$.*

For example, consider $P = a.P_1 + b.P_2$. On the one hand we have $P \xrightarrow{a} P_1$ and also $P \xrightarrow{b} P_2$ namely, there are two possibly different residuals of $P$ after $a$ due to two different branches of the external choice that are guarded by the same action. On the other hand, the (unique) continuation of $P$ after $a$ is $P_1 \oplus P_2$, which expresses the fact that both branches are possible.

# B Supplement to Section 3

## B.1 Semantics

We begin by gaining some familiarity with the orthogonal and the bi-orthogonal operators and some of their properties, in particular we provide alternative characterizations for $X^\perp$ and $X^{\perp\perp}$, we prove that $(\cdot)^\perp$ is anti-monotonic, and we state some known properties regarding orthogonal set and set-theoretic operators.

**Proposition B.1.** *The following properties hold:*

1. $X^\perp = \bigcap_{P \in X} [\![P]\!]$;
2. $X^{\perp\perp} = \bigcap_{X \subseteq [\![P]\!]} [\![P]\!]$;
3. $X \subseteq Y$ *implies* $Y^\perp \subseteq X^\perp$;
4. $X^\perp$ *is closed;*
5. $(X \cup Y)^\perp = X^\perp \cap Y^\perp$.

*Proof.* We prove the items in order:

1. We have $Q \in X^\perp$ iff $X \subseteq [\![Q]\!]$ iff $P \perp Q$ for every $P \in X$ iff $Q \in [\![P]\!]$ for every $P \in X$ iff $Q \in \bigcap_{P \in X} [\![P]\!]$.
2. By item (1) we have $X^{\perp\perp} = \bigcap_{P \in X^\perp} [\![P]\!] = \bigcap_{X \subseteq [\![P]\!]} [\![P]\!]$.
3. By item (1) we have $Y^\perp = \bigcap_{P \in Y} [\![P]\!] \subseteq \bigcap_{P \in X} [\![P]\!] = X^\perp$.
4. From Proposition 3.1(1) we obtain $X^\perp \subseteq X^{\perp\perp\perp}$ by replacing $X$ with $X^\perp$. From the same proposition and item (3) we obtain $X^{\perp\perp\perp} \subseteq X^\perp$. We conclude $X^\perp = X^{\perp\perp\perp}$.
5. By item (1) we have $(X \cup Y)^\perp = \bigcap_{P \in X \cup Y} [\![P]\!] = \bigcap_{P \in X} [\![P]\!] \cap \bigcap_{P \in Y} [\![P]\!] = X^\perp \cap Y^\perp$. □

It should be observed that item (5) of the previous proposition can be generalized to arbitrary unions, namely that

$$\Big(\bigcup_{i \in I} X_i\Big)^\perp = \bigcap_{i \in I} X_i^\perp$$

for arbitrary, possibly infinite family of sets $X_i$. The reader may also verify that $\wedge$ and $\vee$ are indeed commutative and associative operators. These properties will be silently used in some of the proofs that follow.

We now present an auxiliary operator that is convenient in the definition of the semantics of session types. We write $\mathscr{G}_\alpha(X)$ for the set of processes that guarantee an $\alpha$ action and whose continuation after $\alpha$ is a process in $X$. Formally:

$$\mathscr{G}_\alpha(X) \stackrel{\text{def}}{=} \begin{cases} \mathscr{P} & \text{if } X^\perp = \emptyset \\ \{P \in \mathscr{P} \mid P \Downarrow \alpha \text{ and } P(\alpha) \in X\} & \text{otherwise} \end{cases}$$

Using $\mathscr{G}_\cdot(\cdot)$ one can equivalently define the interpretation of $\alpha.T$ as $[\![\alpha.T]\!] = \mathscr{G}_{\overline{\alpha}}([\![T]\!])$. In particular, the orthogonal of $\mathscr{G}_\alpha(X)$ can be computed simply by turning $\alpha$ into the corresponding co-action and by computing the orthogonal of $X$:

**Proposition B.2.** $\mathscr{G}_\alpha(X)^\perp = \mathscr{G}_{\overline{\alpha}}(X^\perp)$.

*Proof.* We distinguish three cases:

- ($X = \emptyset$) Then $X^\perp = \mathscr{P}$ and we conclude $\mathscr{G}_\alpha(X)^\perp = \emptyset^\perp = \mathscr{P} = \mathscr{G}_{\overline{\alpha}}(\mathscr{P}) = \mathscr{G}_{\overline{\alpha}}(X^\perp)$.
- ($X^\perp = \emptyset$) Then $\mathscr{G}_\alpha(X)^\perp = \mathscr{P}^\perp = \emptyset = \mathscr{G}_{\overline{\alpha}}(\emptyset) = \mathscr{G}_{\overline{\alpha}}(X^\perp)$.
- ($X \neq \emptyset$ and $X^\perp \neq \emptyset$) We have:

$$\begin{array}{lll} Q \in \mathscr{G}_\alpha(X)^\perp & \iff & \forall P \in \mathscr{G}_\alpha(X) : P \perp Q \qquad (X^\perp \neq \emptyset) \\ & \iff & \forall P \in \mathscr{G}_\alpha(X) : Q \Downarrow \overline{\alpha} \wedge Q(\overline{\alpha}) \perp P(\alpha) \\ & \iff & Q \Downarrow \overline{\alpha} \wedge \forall P \in \mathscr{G}_\alpha(X) : Q(\overline{\alpha}) \perp P(\alpha) \qquad (X \neq \emptyset) \\ & \iff & Q \Downarrow \overline{\alpha} \wedge Q(\overline{\alpha}) \in X^\perp \\ & \iff & Q \in \mathscr{G}_{\overline{\alpha}}(X^\perp) \end{array}$$

  namely $\mathscr{G}_\alpha(X)^\perp = \mathscr{G}_{\overline{\alpha}}(X^\perp)$. □

**Corollary B.1.** *$X$ closed implies $\mathscr{G}_\alpha(X)$ closed.*

*Proof.* By Proposition B.2 we have $\mathscr{G}_\alpha(X)^{\perp\perp} = \mathscr{G}_{\overline{\alpha}}(X^\perp)^\perp = \mathscr{G}_\alpha(X^{\perp\perp}) = \mathscr{G}_\alpha(X)$. □

We now have all the information for showing that $[\![T]\!]$ is a closed set of processes, so that we can rewrite $[\![T]\!]$ into $[\![T]\!]^{\perp\perp}$ and viceversa, whenever useful (Proof of Theorem 3.1).

**Proposition B.3.** *For every $T$, the set $[\![T]\!]$ is closed.*

*Proof.* An easy induction on $T$. The case when $T = \mathsf{end}$ follows from the fact that $\mathbf{1} \in [\![\mathsf{end}]\!]$, hence $[\![\mathsf{end}]\!]^\perp = [\![\mathsf{end}]\!]$. The case when $T = T_1 \wedge T_2$ is proved using Proposition B.1. □

**Theorem B.1** (Theorem 3.1)**.** *For every $T$, $[\![\overline{T}]\!] = [\![T]\!]^\perp$.*

*Proof.* By induction on $T$ and by cases on its shape:

- $[\![\overline{\mathbb{0}}]\!] = [\![\mathbb{1}]\!] = \mathscr{P} = \emptyset^\perp = [\![\mathbb{0}]\!]^\perp$.
- $[\![\overline{\mathbb{1}}]\!] = [\![\mathbb{0}]\!] = \emptyset = \mathscr{P}^\perp = [\![\mathbb{1}]\!]^\perp$.
- $[\![\overline{\mathsf{end}}]\!] = \{P \in \mathscr{P} \mid P \Downarrow \checkmark\} = [\![\mathsf{end}]\!]^\perp$.
- $[\![\overline{\alpha.S}]\!] = [\![\overline{\alpha}.\overline{S}]\!] = \mathscr{G}_{\overline{\alpha}}([\![\overline{S}]\!]) = \mathscr{G}_{\overline{\alpha}}([\![S]\!]^\perp) = \mathscr{G}_\alpha([\![S]\!])^\perp = [\![\alpha.S]\!]^\perp$.
- $[\![\overline{T_1 \wedge T_2}]\!] = [\![\overline{T}_1 \vee \overline{T}_2]\!] = ([\![\overline{T}_1]\!] \cup [\![\overline{T}_2]\!])^{\perp\perp} = ([\![T_1]\!]^\perp \cup [\![T_2]\!]^\perp)^{\perp\perp} = ([\![T_1]\!]^{\perp\perp} \cap [\![T_2]\!]^{\perp\perp})^\perp = ([\![T_1]\!] \cap [\![T_2]\!])^\perp = [\![T_1 \wedge T_2]\!]^\perp$.
- $[\![\overline{T_1 \vee T_2}]\!] = [\![\overline{T}_1 \wedge \overline{T}_2]\!] = [\![\overline{T}_1]\!] \cap [\![\overline{T}_2]\!] = [\![T_1]\!]^\perp \cap [\![T_2]\!]^\perp = ([\![T_1]\!] \cup [\![T_2]\!])^\perp = [\![T_1 \vee T_2]\!]^\perp$. □

## B.2 Subtyping Algorithm

**Lemma B.1.** *Let* $T \equiv \bigvee_{a\in \mathrm{A}} a.T_a\{\vee \mathsf{end}\}_{\checkmark\in \mathrm{A}}$ *and* $S \equiv \bigwedge_{a\in \mathrm{A}} \overline{a}.S_a\{\wedge \mathsf{end}\}_{\checkmark\in \mathrm{A}}$ *for some* $\mathrm{A} \subseteq \mathscr{N} \cup \{\checkmark\}$ *where* $T_a$ *and* $S_a$ *are viable for every* $a \in \mathrm{A}$*. Then the following properties hold:*

1. $P \in [\![T]\!]$ *if and only if* $P\downarrow$ *and* $\{\mu \mid P \downarrow \mu\} \subseteq \overline{\mathrm{A}}$ *and* $P \downarrow \overline{a}$ *implies* $P(\overline{a}) \in [\![T_a]\!]$*;*
2. $P \in [\![S]\!]$ *if and only if* $P\Downarrow$ *and* $\mathrm{A} \subseteq \{\mu \mid P \Downarrow \mu\}$ *and* $a \in \mathrm{A}$ *implies* $P(a) \in [\![S_a]\!]$*.*

*Proof.* We prove the two items in order:

1. Since $[\![T]\!]$ is closed we have $P \in [\![T]\!]$ if and only if $P \in [\![T]\!]^{\perp\perp}$ if and only if $[\![T]\!]^{\perp} \subseteq [\![P]\!]$. Now

$$[\![T]\!]^{\perp} = \left(\bigcup_{a\in \mathrm{A}} \mathscr{G}_{\overline{a}}([\![T_a]\!])\{\cup [\![\mathsf{end}]\!]\}_{\checkmark\in \mathrm{A}}\right)^{\perp} = \bigcap_{a\in \mathrm{A}} \mathscr{G}_{\overline{a}}([\![T_a]\!])^{\perp}\{\cap [\![\mathsf{end}]\!]^{\perp}\}_{\checkmark\in \mathrm{A}} = \bigcap_{a\in \mathrm{A}} \mathscr{G}_{a}([\![T_a]\!]^{\perp})\{\cap [\![\mathsf{end}]\!]\}_{\checkmark\in \mathrm{A}}$$

   in particular $\sum_{a\in \mathrm{A}} a.Q_a\{+\mathbf{1}\}_{\checkmark\in \mathrm{A}} \in [\![T]\!]^{\perp}$ for every $Q_a \in [\![T_a]\!]^{\perp}$. We deduce $P\downarrow$ and $P \downarrow \mu$ implies $\mu \in \overline{\mathrm{A}}$ and $P \downarrow \overline{a}$ implies $P(\overline{a}) \perp Q_a$. Since this holds for every $Q_a \in [\![T_a]\!]^{\perp}$ we have $[\![T_a]\!]^{\perp} \subseteq [\![P(\overline{a})]\!]$, which is equivalent to $P(\overline{a}) \in [\![T_a]\!]$.

2. We have

$$[\![S]\!] = \bigcap_{a\in \mathrm{A}} \mathscr{G}_{a}([\![S_a]\!])\{\cap [\![\mathsf{end}]\!]\}_{\checkmark\in \mathrm{A}}$$

   from which we deduce that $P \in [\![S]\!]$ if and only if $P\Downarrow$ and $\mu \in \mathrm{A}$ implies $P \Downarrow \mu$ and $a \in \mathrm{A}$ implies $P(a) \in [\![S_a]\!]$. □

**Lemma B.2** (Lemma 3.1)**.** *The laws in Table 4 are sound.*

*Proof.* Laws (E-PREFIX), (E-BOTTOM), and (E-TOP) are left as easy exercises for the reader. Regarding rule (E-DIST) we have $[\![\alpha.T \wedge \alpha.S]\!] = [\![\alpha.T]\!] \cap [\![\alpha.S]\!] = \mathscr{G}_{\overline{\alpha}}([\![T]\!]) \cap \mathscr{G}_{\overline{\alpha}}([\![S]\!]) = \{P \in \mathscr{P} \mid P \Downarrow \overline{\alpha} \wedge P(\overline{\alpha}) \in [\![T]\!]\} \cap \{P \in \mathscr{P} \mid P \Downarrow \overline{\alpha} \wedge P(\overline{\alpha}) \in [\![S]\!]\} = \{P \in \mathscr{P} \mid P \Downarrow \overline{\alpha} \wedge P(\overline{\alpha}) \in [\![T]\!] \cap [\![S]\!]\} = \mathscr{G}_{\overline{\alpha}}([\![T]\!] \cap [\![S]\!]) = \mathscr{G}_{\overline{\alpha}}([\![T \wedge S]\!]) = [\![\alpha.(T \wedge S)]\!]$. Regarding rule (E-INPUT-END), let $T \stackrel{\text{def}}{=} \bigvee_{a\in \mathrm{A}} a.T_a$ and suppose by contradiction that $P \in [\![T \wedge \mathsf{end}]\!]$. Then $P \in [\![T]\!]$ and $P \in [\![\mathsf{end}]\!]$ which implies $P\downarrow$ and $\{\mu \mid P \downarrow \mu\} \subseteq \overline{\mathrm{A}}$ and $P \Downarrow \checkmark$. Since $P \downarrow \overline{a}$ and $P \Downarrow \checkmark$ are incompatible properties we deduce $\mathrm{A} = \emptyset$. Then $T \approx \mathbb{0}$, which contradicts the hypothesis $P \in [\![T \wedge \mathsf{end}]\!]$. The proof that rule (E-INPUT-OUTPUT) is sound is similar, except that in this case $P \in [\![\overline{b}.S]\!]$ implies $P \Downarrow b$. Regarding rule (E-INPUT-OUTPUT-END), let $T \stackrel{\text{def}}{=} \bigvee_{a\in \mathrm{A}} a.T_a \vee \mathsf{end}$. We only need to prove $T \wedge \overline{b}.S \lesssim \mathsf{end}$ because $T \wedge \overline{b}.S \lesssim \overline{b}.S$ is obvious and $\overline{b}.S \wedge \mathsf{end} \lesssim T \wedge \overline{b}.S$ follows immediately from the fact that $\mathsf{end} \lesssim T$ and the pre-congruence of $\lesssim$. Let $P \in [\![T \wedge \overline{b}.S]\!]$. By Lemma B.1 we deduce $P \Downarrow b$ and $P\downarrow$. The only action in $\overline{\mathscr{N}} \cup \{\checkmark\}$ that may coexist with a guaranteed input action ($b$) is $\checkmark$. Since $P\downarrow$ we have $P \downarrow \mu$ implies $\mu = \checkmark$, hence $P \Downarrow \checkmark$. We conclude $P \in [\![\mathsf{end}]\!]$. Regarding rule (E-INPUT-INPUT) let $T \stackrel{\text{def}}{=} \bigvee_{a\in \mathrm{A}} a.T_a\{\vee \mathsf{end}\}_{\checkmark\in \mathrm{A}}$ and $S \stackrel{\text{def}}{=} \bigvee_{b\in \mathrm{B}} b.S_b\{\vee \mathsf{end}\}_{\checkmark\in \mathrm{B}}$. By Lemma B.1 we have $P \in [\![T \wedge S]\!]$ if and only if $P\downarrow$ and $\{\mu \mid P \downarrow \mu\} \subseteq \overline{\mathrm{A} \cap \mathrm{B}}$ and $P \downarrow \overline{a}$ implies $P(\overline{a}) \in [\![T_a]\!] \cap [\![S_a]\!]$ if and only if $P \in [\![\bigvee_{a\in \mathrm{A}\cap \mathrm{B}} a.(T_a \wedge S_a)\{\vee \mathsf{end}\}_{\checkmark\in \mathrm{A}\cap \mathrm{B}}]\!]$. □

Some of the proofs that follow are defined by induction on the *depth* of session types. By "depth" of a session type we mean the maximum number of nested actions in it. For example $a.\overline{b} \wedge c$ has depth 2, while $a.\overline{b}.c$ has depth 3. The session types $\mathbb{0}$, $\mathbb{1}$, and $\mathsf{end}$ all have depth 0.

**Lemma B.3** (Lemma 3.2)**.** *For every session type* $T$ *there exists* $S$ *in normal form such that* $T \approx S$*.*

*Proof.* By induction on the depth of $T$ and by cases on its shape.

- If $T \equiv \mathbb{1}$ or $T \equiv \mathbb{0}$ or $T \equiv \mathsf{end}$, then $T$ is already in normal form.
- If $T \equiv \alpha.T'$, then by induction hypothesis there exists $S'$ in normal form such that $T' \simeq S'$. We reason by cases on $S'$ for finding $S$ in normal form such that $T \simeq S$:
  - if $S' \equiv \mathbb{1}$, then $T \simeq \alpha.\mathbb{1} \simeq \mathbb{1}$;
  - if $S' \equiv \mathbb{0}$, then $T \simeq \alpha.\mathbb{0} \simeq \mathbb{0}$;
  - in all the other cases we have $T \simeq \alpha.S'$ which is in normal form.
- If $T \equiv T_1 \wedge T_2$, then by induction hypothesis there exist $S_1$ and $S_2$ in normal form such that $T_1 \simeq S_1$ and $T_2 \simeq S_2$. We reason by cases on $S_1$ and $S_2$ (symmetric cases omitted):
  - if $S_1 \equiv \mathbb{0}$ we have $T \simeq \mathbb{0} \wedge S_2 \simeq \mathbb{0}$;
  - if $S_1 \equiv \mathbb{1}$ we have $T \simeq \mathbb{1} \wedge S_2 \simeq S_2$;
  - if $S_1 \equiv \bigvee_{a\in \mathrm{A}} a.S_{1,a}\{\vee \mathsf{end}\}_{\checkmark\in \mathrm{A}}$ and $S_2 \equiv \bigvee_{b\in \mathrm{B}} b.S_{2,b}\{\vee \mathsf{end}\}_{\checkmark\in \mathrm{B}}$, then by rule (E-INPUT-INPUT) we have $T \simeq S_1 \wedge S_2 \simeq \bigvee_{a\in \mathrm{A}\cap \mathrm{B}} a.(S_{1,a} \wedge S_{2,a})\{\vee \mathsf{end}\}_{\checkmark\in \mathrm{A}\cap \mathrm{B}}$. By induction hypothesis there exists $S_a$ in normal form such that $S_{1,a} \wedge S_{2,a} \simeq S_a$ for every $a \in A \cap B$, therefore $T \simeq \bigvee_{a\in \mathrm{A}\cap \mathrm{B}} a.S_a\{\vee \mathsf{end}\}_{\checkmark\in \mathrm{A}\cap \mathrm{B}}$.
  - if $S_1 \equiv \bigwedge_{a\in \mathrm{A}} \overline{a}.S_{1,a}\{\wedge \mathsf{end}\}_{\checkmark\in \mathrm{A}}$ and $S_2 \equiv \bigwedge_{b\in \mathrm{B}} \overline{b}.S_{2,a}\{\wedge \mathsf{end}\}_{\checkmark\in \mathrm{B}}$, then $T \simeq S_1 \wedge S_2 \simeq \bigwedge_{a\in \mathrm{A}\setminus \mathrm{B}} \overline{a}.S_{1,a} \wedge \bigwedge_{b\in \mathrm{B}\setminus \mathrm{A}} \overline{b}.S_{2,b} \wedge \bigwedge_{a\in \mathrm{A}\cap \mathrm{B}} \overline{a}.S_a\{\wedge \mathsf{end}\}_{\checkmark\in \mathrm{A}\cup \mathrm{B}}$ where $S_a$ is in normal form and $S_{1,a} \wedge S_{2,a} \simeq S_a$ for every $a \in \mathrm{A} \cap \mathrm{B}$.
  - if $S_1 \equiv \bigvee_{a\in \mathrm{A}} a.S_{1,a}$ and $S_2 \equiv \bigwedge_{b\in \mathrm{B}} \overline{b}.S_{2,a}\{\wedge \mathsf{end}\}_{\checkmark\in \mathrm{B}}$, then by rules (E-INPUT-END) and/or (E-INPUT-OUTPUT) we conclude $T \simeq \mathbb{0}$.
  - if $S_1 \equiv \bigvee_{a\in \mathrm{A}} a.S_{1,a} \vee \mathsf{end}$ and $S_2 \equiv \bigwedge_{b\in \mathrm{B}} \overline{b}.S_{2,a}\{\wedge \mathsf{end}\}_{\checkmark\in \mathrm{B}}$, then by rule (E-INPUT-OUTPUT-END) we conclude $T \simeq \bigwedge_{b\in \mathrm{B}} \overline{b}.S_{2,a} \wedge \mathsf{end}$.
- If $T \equiv T_1 \vee T_2$, then we reason in a dual fashion with respect to the previous case. □

**Theorem B.2** (Theorem 3.2)**.** *Let $T$ and $S$ be in normal form. Then $T \lesssim S$ if and only if $T \leqslant S$.*

*Proof.* The "if" part is trivial since $\leqslant$ axiomatizes obvious properties of $\lesssim$. Regarding the "only if" part, we proceed by induction on the depth of $T$ and $S$ and by cases on their (normal) form. We omit dual cases:

- ($T \equiv \mathbb{0}$) We conclude with an application of either (S-BOTTOM) or (S-INPUT) according to the form of $S$.
- ($S \equiv \mathbb{1}$) We conclude with an application of either (S-TOP) or (S-OUTPUT) according to the form of $T$.
- ($T \equiv \bigvee_{a\in \mathrm{A}} a.T_a\{\vee \mathsf{end}\}_{\checkmark\in \mathrm{A}}$ and $S \equiv \bigvee_{b\in \mathrm{B}} b.S_b\{\vee \mathsf{end}\}_{\checkmark\in \mathrm{B}}$) From the hypothesis $T \lesssim S$ and Lemma B.1 we deduce $\mathrm{A} \subseteq \mathrm{B}$ and $T_a \lesssim S_a$ for every $a \in \mathrm{A}$. By induction hypothesis we derive $T_a \leqslant S_a$ for every $a \in \mathrm{A}$, and we conclude with an application of rule (S-INPUT).
- ($T \equiv \bigvee_{a\in \mathrm{A}} a.T_a\{\vee \mathsf{end}\}_{\checkmark\in \mathrm{A}}$ and $S \equiv \bigwedge_{b\in \mathrm{B}} \overline{b}.S_b\{\wedge \mathsf{end}\}_{\checkmark\in \mathrm{B}}$) For every $P \in [\![T]\!]$ we have $\{\mu \mid P \downarrow \mu\} \subseteq \overline{\mathrm{A}}$ and $\emptyset \neq \mathrm{B} \subseteq \{\mu \mid P \Downarrow \mu\}$ from which we deduce $\mathrm{A} = \mathrm{B} = \{\checkmark\}$. We conclude with an application of rule (S-END).
- ($T \equiv \bigwedge_{a\in \mathrm{A}} \overline{a}.T_a\{\wedge \mathsf{end}\}_{\checkmark\in \mathrm{A}}$ and $S \equiv \bigwedge_{b\in \mathrm{B}} \overline{b}.S_b\{\wedge \mathsf{end}\}_{\checkmark\in \mathrm{B}}$) For every $P \in [\![T]\!]$ we have $\mathrm{A} \subseteq \{\mu \mid P \Downarrow \mu\}$ implies $\mathrm{B} \subseteq \{\mu \mid P \Downarrow \mu\}$ meaning $\mathrm{B} \subseteq \mathrm{A}$. Furthermore, $T_b \lesssim S_b$ for every $b \in \mathrm{B}$. By induction hypothesis we deduce $T_b \leqslant S_b$ for every $b \in \mathrm{B}$, and we conclude with an application of rule (S-OUTPUT).

- ($T \equiv \bigwedge_{a\in \mathrm{A}} \overline{a}.T_a\{\wedge \mathsf{end}\}_{\checkmark\in \mathrm{A}}$ and $S \equiv \bigvee_{b\in \mathrm{B}} b.S_b\{\vee \mathsf{end}\}_{\checkmark\in \mathrm{B}}$) For every $P \in [\![T]\!]$ we have $\mathrm{A} \subseteq \{\mu \mid P \Downarrow \mu\}$ implies $\{\mu \mid P\downarrow\mu\} \subseteq \overline{\mathrm{B}}$, from which we deduce $\checkmark \in \mathrm{A}\cap\mathrm{B}$. We conclude with an application of rule (S-END). □

## B.3 Type Checker

**Theorem B.3** (Theorem 3.3)**.** *If $T \vdash P$ and $\overline{T} \vdash Q$, then $P \perp Q$.*

*Proof.* It is sufficient to show that $T \vdash P$ implies $P \in [\![T]\!]^\perp$ for some generic $P$ and $T$. Then, by Theorem 3.1 we have $Q \in [\![\overline{T}]\!]^\perp = [\![T]\!]^{\perp\perp} = [\![T]\!]$ and we conclude $P \perp Q$ by definition of orthogonal set. We prove that $T \vdash P$ implies $P \in [\![T]\!]^\perp$ by induction on the derivation of $T \vdash P$ and by cases on the last rule applied:

- (T-NIL) Then $P = \mathbf{0}$ and $T = \mathbb{0}$ and we conclude $\mathbf{0} \in \mathscr{P} = \emptyset^\perp = [\![\mathbb{0}]\!]^\perp$.
- (T-END) Then $P = \mathbf{1}$ and $T = \mathsf{end}$ and we conclude $\mathbf{1} \in [\![\mathsf{end}]\!]^\perp = [\![\mathsf{end}]\!] = \{P \in \mathscr{P} \mid P \Downarrow \checkmark\}$.
- (T-SEND) Then $P = \overline{a}.Q$ and $T = \overline{a}.S$ for some $Q$ and $S$ such that $S \vdash Q$. By induction hypothesis we deduce $Q \in [\![S]\!]^\perp$. We conclude $P \in [\![T]\!]^\perp = \mathscr{G}_a([\![S]\!])^\perp = \mathscr{G}_{\overline{a}}([\![S]\!]^\perp)$ since $P \Downarrow \overline{a}$ and $P(\overline{a}) = Q \in [\![S]\!]^\perp$.
- (T-RECEIVE) Then $P = \sum_{i\in I} a_i.P_i$ and $T = \bigvee_{i\in I} a_i.T_{a_i}$ where $T_{a_i} \vdash P_i$ for every $i \in I$. By induction hypothesis we have $P_i \in [\![T_{a_i}]\!]^\perp$ for every $i \in I$. We conclude $P \in T^\perp = (\bigvee_{i\in I} a_i.T_{a_i})^\perp = (\bigcup_{i\in I} \mathscr{G}_{\overline{a}_i}([\![T_{a_i}]\!]))^{\perp\perp\perp} = (\bigcup_{i\in I} \mathscr{G}_{\overline{a}_i}([\![T_{a_i}]\!]))^\perp = \bigcap_{i\in I} \mathscr{G}_{\overline{a}_i}([\![T_{a_i}]\!])^\perp = \bigcap_{i\in I} \mathscr{G}_{a_i}([\![T_{a_i}]\!]^\perp)$ because $P \Downarrow a_i$ and $P(a_i) = \bigoplus_{a_i=a_j} P_j \in [\![T_{a_i}]\!]^\perp$ for every $i \in I$.
- (T-CHOICE) Then $P = P_1 \oplus P_2$ where $T \vdash P_i$ for $i \in \{1,2\}$. By induction hypothesis we deduce $P_i \in [\![T]\!]^\perp$ for $i \in \{1,2\}$, hence we conclude $P \in [\![T]\!]^\perp$ because $[\![T]\!]^\perp$ is closed.
- (T-SUB) Then $S \vdash P$ for some $S$ such that $T \lesssim S$. By induction hypothesis we have $P \in [\![S]\!]^\perp$ hence we conclude $P \in [\![T]\!]^\perp$ since $T \lesssim S$ implies $[\![S]\!]^\perp \subseteq [\![T]\!]^\perp$ by Proposition B.1(3). □