

Foundations of Session Types

Giuseppe Castagna¹

Mariangiola Dezani-Ciancaglini²

Elena Giachino^{1,2}

Luca Padovani³

¹PPS (CNRS) - Université Denis Diderot - Paris, France

²Dipartimento di Informatica - Università degli Studi di Torino - Torino, Italy

³Istituto di Scienze e Tecnologie dell'Informazione - Università degli Studi di Urbino - Urbino, Italy

Abstract. We present a streamlined theory of session types based on a simple yet general and expressive formalism whose main features are semantically characterized and where each design choice is semantically justified. We formally define the semantics of session types and use it to devise the subsessioning relation. We give a coinductive characterization of subsessioning and describe algorithms to decide all the key relations defined in the article. We demonstrate the generality and expressivity of our framework by providing a session-based type system for a π -calculus variant that does not rely on any specialized construct for session-based communication. The type system is shown to guarantee absence of communication errors and global progress.

1. Introduction

Sessions are a common mechanism of interaction in distributed architectures. Two processes willing to interact establish a connection on a shared public channel. In this connection they agree on some private channel on which to have a conversation, dubbed session. The conversation follows a protocol describing the kind and order of the messages exchanged on the private channel. In general, a protocol does not specify a unique sequence of interactions. At any point of the interaction the rest of the conversation for a process may depend upon the kind of messages received by the process on the private channel and/or the internal state of the process. When the decision is exclusively based on the received messages one speaks of an *external choice*. When the decision is taken autonomously by the process one speaks of an *internal choice*. The messages exchanged during a session may be synchronization signals, basic values (e.g., integers, strings), names of public channels (those used to start sessions), or even names of private channels of already started sessions. In the last case one speaks of *delegation* since by sending to some other process the private channel of a session, the process delegates the receiver to continue that session.

In summary, session-based interaction is obtained from two ingredients, each ingredient being formed by two different components. The first ingredient is communication and takes place on two different kinds of channels, public channels used to establish a connection, and private channels created at the connection of the former and on which messages of different kinds are exchanged according to a given protocol. The second ingredient is control, and it is implemented by two different kinds of choices, internal choices and external ones.

Static descriptions of the behaviour of sessions (i.e., their protocols) should permit the detection of communication mismatches and session deadlocks. Types are a good candidate for such descriptions, except that typical type systems for process algebras are unfit to type the private channels on which sessions take place, since these channels can carry messages of different types. To obviate this limitation Honda *et al.* introduced *session types* [18, 20] that

describe the sequences of messages exchanged on a private session channel and their possible branching based on labels. To that end they enrich the language of types *and of processes* with specialized signals for dealing with connections, delegations, and signals carrying labels that drive choices in combination with label-based branching primitives. Since then, several variants of session types have been put forward. They vary according to the programming language they target, the type containment relations, and the specific features they aim to capture. As Honda *et al.*, they rely on label-based primitives that tie them to the particular problem they tackle and may hinder their adoption in general purpose languages.

In this work we present a basic and unified theory of session types that aims at being as much language independent as possible. To achieve language independence, we design our types around a process language that has been stripped off of any specific session-oriented linguistic construct: session connections, interactions, and delegations will be imagined as instances of π -calculus communications. We suppose branching as being implemented by classic process algebra internal and external choices [9], with just a single modification: we allow the branch of an external choice to be selected according to the type of the message being communicated, as opposed to the channel on which communication occurs. This modification fits nicely the session-based communication model, where messages are exchanged over a unique, private channel.

Our approach has many positive upshots. First, having dissociated control from a particular linguistic construct, that is label-driven branching, we can more easily type the native branching constructs of a language we want to endow with session types, thus avoiding clumsy language extensions. As an aside, language independence is further increased by the fact that all our definitions are semantic-based, rather than syntax-oriented. Second, we enhance compositionality of branching constructs because the result of the combination of different branches is automatically computed at the type level, without the need of introducing new labels or of renaming existing ones to avoid clashes on shared labels. Last but not least, replacing labels with values and types increases expressiveness: values are first class (so they can result from computations and communicated on channels) and types enable the definition of finer grained disciplines for branch selection.

The rest of the paper is organized as follows. Section 2 defines syntax and set-theoretic semantics of our session types. The subtyping and subsessioning relations that follow arise as natural consequences of our semantic-based framework. We provide a coinductive characterization of subsessioning that sheds light on the properties of subsessioning and we describe algorithms to decide all key relations defined in the article. In Section 3 we put our framework at work by defining a type system for a minimal π -calculus variant that is free of any session-based linguistic construct. We show that the type system enforces a suitably defined progress property. Section 4 summarizes the contributions of our work, draws connections

with some of the most closely related papers, and sketches future directions of research. Proofs, further details, and further technical content are included in the appendix, for referees convenience only.

2. Session types

2.1 Type syntax

As said in the introduction we have two kinds of channels: public channels and private ones. Public channels are used for connecting two processes that are willing to interact by means of a fresh, private channel exchanged during the connection. Private channels are used for the actual conversation between the two processes. At type level this distinction corresponds to two different syntactic categories. Public channels are associated with a *session type* of the form $\text{begin}.\eta$ that classifies channels on which conversations can be requested by processes behaving according to the description η . Private channels are classified by *session descriptors*, ranged over by η . Session descriptors and types are defined by the grammar:

$$\begin{aligned} \text{(types)} \quad t &::= \dots \mid \text{begin}.\eta \mid \neg t \mid t \wedge t \mid t \vee t \mid v \\ \text{(descriptors)} \quad \eta &::= \text{end} \mid \alpha.\eta \mid \eta \oplus \eta \mid \eta + \eta \\ \text{(actions)} \quad \alpha &::= !t \mid ?t \mid !\chi \mid ?\chi \\ \text{(sieves)} \quad \chi &::= \eta \mid \neg\chi \mid \chi \wedge \chi \mid \chi \vee \chi \end{aligned}$$

Participants of a session use their (private) session channel either to exchange values (of some type) or to delegate other session channels (of some descriptor). In descriptors we use $?t$ and $!t$ to denote that (the process using) the channel will respectively wait for and send some value of type t , and use $?\eta$ and $!\eta$ (actually, $?\chi$ and $!\chi$, see later on) to denote that (the process that uses) the channel will respectively wait for (i.e., catch) and send (i.e., delegate) some private channel that will be used abiding by the session descriptor η . A descriptor $\alpha.\eta$ states that (the process using) the channel will perform one of the communication actions α described above and then will behave according to η ; a descriptor end states that the session on the channel has successfully ended; a descriptor $\eta_1 \oplus \eta_2$ states that (the process that uses) the channel will internally choose to behave according to either η_1 or η_2 ; a descriptor $\eta_1 + \eta_2$ states that (the process that uses) the channel gives the communicating partner the choice to behave according to either η_1 or η_2 . In what follows we adopt the convention that the prefix operator has precedence over the choice operators and we will use parentheses to enforce precedence. For instance, $(!t.\eta) + \text{end}$ and $!t.\eta + \text{end}$ denote the same session descriptor, which is different from $!t.(\eta + \text{end})$. Types t are inherited from the host language (this is stressed in the grammar above by the ellipsis in the production for types), to which we add (unless they are already provided by the host) singleton types (denoted by a value v , the only one they contain), Boolean combinators (i.e., \vee , \wedge , and \neg), and *session types* of the form $\text{begin}.\eta$ which classify yet-to-be-used public channels. The interest of session types is that they can be used to type higher-order communications in which the names of public channels are communicated over other channels; session types will also extend the type system of the host language which can thus use names of public channels as first class values.

The importance of Boolean combinators for types is shown by the following example where we assume Int be a subtype of Real :

$$?\text{Real}.\text{!Int}.\text{end} + ?\text{Int}.\text{!Bool}.\text{end} \quad (1)$$

The session descriptor above declares that if a process (that uses a channel with that behaviour) receives a real number, then it will answer by sending an integer, while if it receives an integer it will answer by sending a Boolean. A partner process establishing a conversation on such a channel knows that if it sends a real that is not an integer, then it should be ready to receive an integer while if it sends an integer, then it must be ready to receive an integer

or a Boolean value (notice how the type of the message drives the selection of the external choice). That is, its conversation will be represented by the following descriptor ($t \setminus s$ stands for $t \wedge \neg s$):

$$!(\text{Real} \setminus \text{Int}).?\text{Int}.\text{end} + !\text{Int}?.(\text{Bool} \vee \text{Int}).\text{end} \quad (2)$$

We see that Boolean combinators immediately arise when describing the behaviour of an interacting process. They are also useful when considering equivalences. For instance, (1) is intuitively equivalent to

$$?(\text{Real} \setminus \text{Int}).!\text{Int}.\text{end} + ?\text{Int}?.(\text{Bool} \vee \text{Int}).\text{end} \quad (3)$$

The crucial role of Boolean combinators can further be shown by slightly modifying (1) so that it performs only input actions:

$$?\text{Real}?.\text{!Int}.\text{end} + ?\text{Int}?.\text{!Bool}.\text{end} \quad (4)$$

In this case the descriptor declares that after receiving an integer it will either wait for another integer or for a Boolean value. If an interacting process sends an integer, then in order to be sure that the conversation will not be stuck it must next send a value that is both an integer and a Boolean. Since there is no such a value, the only way to successfully interact with (4) is to make sure that interacting processes will only send reals that are not integers: $!(\text{Real} \setminus \text{Int}).!\text{Int}.\text{end}$. In conclusion, the only way to describe the sessions that can successfully interact with (4) is to use negation (for the sake of completeness note that (4) is equivalent to $?(\text{Real} \setminus \text{Int}). ? \text{Int} . \text{end} + ? \text{Int} . ? (\text{Bool} \wedge \text{Int}) . \text{end}$ which is equivalent to $?(\text{Real} \setminus \text{Int}). ? \text{Int} . \text{end}$ since the right summand of the previous choice can never successfully complete a conversation). A similar discussion can be done for delegation, that is, when actions are over session descriptors, rather than types. This is why we added Boolean combinations of session descriptors too (we dub them *sieves*) and actions have the form $?\chi$ and $!\chi$ rather than $?\eta$ and $!\eta$.

We want both types and session descriptors to be recursively definable. This is important for types since it allows us to represent recursive data structures (e.g., DTDs) while for session descriptors it allows us to represent services that provide an unbounded number of interactions such as (the service whose behaviour is the solution of the equation) $\eta = \text{end} + ?\text{Int}.\eta$ which describes a session that accepts as many integers as wished by the interacting process. In order to support recursive terms, we resort to a technique already used in [13, 6] where instead of introducing an explicit finite syntax for recursive terms, we directly work with possibly infinite regular term trees that satisfy some contractivity conditions; these conditions ensure that terms are semantically meaningful.

DEFINITION 2.1 (Types). *The types of our system are the possibly infinite regular trees coinductively generated by the productions in the grammar at the beginning of this section that satisfy the following conditions:*

1. on every infinite branch of a type there are infinitely many occurrences of “begin”;
2. on every infinite branch of a session descriptor there are infinitely many occurrences of “.” (the prefix constructor);
3. for every subterm of the form $\alpha.\eta$, the tree $\alpha.\eta$ is not a subtree of α .

The first two conditions are contractivity restrictions that rule out meaningless terms such as (the solutions of the equations) $t = t \vee t$ or $\eta = \eta \oplus \eta$; technically they say that the binary relation \triangleright defined by $t_1 \vee t_2 \triangleright t_i$, $t_1 \wedge t_2 \triangleright t_i$, $\neg t \triangleright t$, $\chi_1 \vee \chi_2 \triangleright \chi_i$, $\chi_1 \wedge \chi_2 \triangleright \chi_i$, $\neg\chi \triangleright \chi$, $\eta_1 + \eta_2 \triangleright \eta_i$, $\eta_1 \oplus \eta_2 \triangleright \eta_i$ is Noetherian (that is, strongly normalizing), which gives an induction principle on terms that we use in proofs without any further explicit reference to the relation \triangleright . The third condition states that recursion cannot escape prefixes and thus it rules out both not meaningful terms such as $\eta = !\eta.\text{end}$ and meaningful ones such as $\eta = !(\text{begin}.\eta).\text{end}$. In standard session calculi

with recursion [30] there is no such a restriction, so both terms are accepted. However, there is no value that inhabits the first type and this forbids delegation of a channel over itself [20, 30]. The type $\eta = !(\text{begin}.\eta).\text{end}$ instead classifies sessions that send a public channel whose type is the same as the type of the current session. Currently this third restriction is necessary to stratify the definitions of the subtyping and subsessioning relations, stratification we use in the proof of Theorem 2.6. We are investigating how to relax it and type the case that still escapes our framework.

We do not specify any particular property for the types of the host language. If the host language has some type constructors (e.g., products, arrows, etc.) the first contractivity condition can be relaxed to requiring that on every infinite branch there are infinitely many occurrences of type constructors. The only condition that we impose on the host language is on values, which must satisfy the following *strong disjunction property* for unions:

$$\vdash v : t_1 \vee t_2 \iff \vdash v : t_1 \text{ or } \vdash v : t_2 \quad (5)$$

This condition *may* be restrictive only in the case that the host language already provides a union type combinator since, otherwise, it can be easily enforced by requiring that every session channel is associated with exactly one (most specific, because of subtyping) session type.

Henceforward, we will use t to range over *types*, θ and η to range over *session descriptors*, χ to range over *sieves*, ψ to range over all of them, and often omit the word “session” when speaking of session descriptors. We reserve v for values, whose definition and typing is left unspecified. We assume that values for a session type $\text{begin}.\eta$ are channels explicitly associated with or tagged by that type (or, because of subtyping, by a $\text{begin}.\eta'$ subtype of $\text{begin}.\eta$: more about that later on).

We do not include in our session descriptors a construct for parallel composition (as opposed to [22, 3], for example). Since we assume an interleaving semantics of parallel composition, then having two different choices is enough for faithfully describing several scenarios of concurrent actions by means of well-known expansion laws.

2.2 Semantics of types and descriptors

The semantics of both session descriptors and types—and more generally most of the constructions of this work—crucially relies on the notion of *duality*. In this section we first informally define duality to outline a denotational semantics for types and descriptors, then we give the formal definition of duality in terms of a labelled transition system for descriptors.

2.2.1 Set-theoretic interpretations

In the previous section we argued that a complete set of Boolean combinators must be used if we want to describe the set of partners that safely interact with a given descriptor. Since we want the semantics of Boolean combinators to be intuitive and easy to understand we base their definition on a set-theoretic interpretation. In particular, we interpret every type constructor as the set of its values and the Boolean combinators as the corresponding set-theoretic operations. In other terms, we seek for an interpretation of types $[\cdot]$ such that $[\mathbf{t}] = \{v \mid \vdash v : t\}$ and that $[\mathbf{t} \wedge \mathbf{s}] = [\mathbf{t}] \cap [\mathbf{s}]$, $[\mathbf{t} \vee \mathbf{s}] = [\mathbf{t}] \cup [\mathbf{s}]$, and $[\neg \mathbf{t}] = \mathcal{V} \setminus [\mathbf{t}]$ (where \mathcal{V} denotes the set of all values). The same interpretation can then be used to *define* the subtyping relation (denoted by “ $<$ ”) as follows:

$$t <: s \stackrel{\text{def}}{\iff} [\mathbf{t}] \subseteq [\mathbf{s}]$$

The technical machinery to define an interpretation with such properties and solve the problems its definition raises (e.g., the circularity between the subtyping relation and the typing of values) already exists and can be found in the work on Semantic Subtyping [13]:

we take it for granted and no longer bother about it if not for session types that are dealt with in Section 2.3. This interpretation of types justifies the use we do henceforward of the notation $v \in t$ to denote that v has type t .

The next problem is to give a set-theoretic interpretation to session descriptors, as we have Boolean combinations on them too. This interpretation is not required to be precise or mathematically meaningful but only to ensure that conversations do not get stuck. To this aim, rather than giving the set of values (or whatever they would be, since session descriptors classify just “chunks” of conversation) contained in a descriptor, it suffices to characterize all the possible behaviours common to all channels that implement a given session. In other terms, the semantics of a session descriptor can be characterized by the set of partners with whom the interaction will never get stuck (a sort of realizability semantics). This is captured by the notion of *duality*: two session descriptors η and θ are *dual* if any conversation on a private channel shared by two processes which follow respectively the prescriptions of η and θ will never get stuck. So, for instance, the descriptor (1) in the previous section is dual to the descriptor (2). But $!\text{Int}.\text{?}(\text{Bool} \vee \text{Int}).\text{end}$ is dual to (1), too.

Note also that some session descriptors have no dual, for example $\text{?}(\text{Bool} \wedge \text{Int}).\text{end}$, since no process can send a value that is both a Boolean and an integer: the intersection is empty.¹ Such descriptors constitute a pathological case, since no conversation can take place on channels conforming to them. Thus we will focus our attention on descriptors for which at least one dual exists, and that we dub *viable descriptors*. We write \mathcal{S} for the set of all viable descriptors and we write $\eta \bowtie \theta$ if η and θ are dual (duality is a symmetric relation). Then, we can define the interpretation of a descriptor as the set of its duals: $[\eta] = \{\theta \mid \eta \bowtie \theta\}$; extend it set-theoretically to sieves: $[\chi \wedge \chi'] = [\chi] \cap [\chi']$, $[\chi \vee \chi'] = [\chi] \cup [\chi']$, $[\neg \chi] = \mathcal{S} \setminus [\chi]$; and use it to semantically define the subsieving (and subsessioning) relation, denoted by “ \leq ”:

$$\chi \leq \chi' \stackrel{\text{def}}{\iff} [\chi] \subseteq [\chi'] \quad (6)$$

Duality plays a central role also in defining the semantics of types. We said that the semantics of a type constructor is the set of its values. Hence we have to define the values of the types $\text{begin}.\eta$. As suggested in Section 2.1, we can take as a value of a session type a public channel tagged by that type *or by a subtype*. Therefore to define values we need to determine when a session type is subtype of another, that is, when we can safely use a channel of some session type where a channel of a different (larger) type is expected. The key is to understand how a public channel is “used”. We make the assumption—matched by everyday practice—that there is a unique way to consume a public channel c of type $\text{begin}.\eta$, by requesting a conversation that conforms to the protocol described by η , expecting that at the other end of the communication channel there will be another process that follows some protocol θ that is dual of η . Thus, it is safe to replace c with a different channel d of smaller type $\text{begin}.\eta'$ only if every dual of η' is also a dual of η , namely when $\eta' \leq \eta$. For instance we have that $\text{?Int}.\text{end} \leq \text{?Real}.\text{end}$ since every descriptor that is dual of $\text{?Int}.\text{end}$ is also dual of $\text{?Real}.\text{end}$. Hence, $\text{begin}.\text{?Int}.\text{end} <: \text{begin}.\text{?Real}.\text{end}$ since if a process that uses a channel of type $\text{begin}.\text{?Real}.\text{end}$ is well typed, then the process obtained by replacing this channel for a different one of type $\text{begin}.\text{?Int}.\text{end}$ is well typed as well: it will receive an integer number in a place where it expects a real number.

Since we want our types to satisfy the strong disjunction property (5), then a public channel c must be tagged by types of the

¹This shows that our duality is semantically defined: $\text{?}(\text{Bool} \wedge \text{Int}).\text{end}$ is *not* dual of $!(\text{Bool} \wedge \text{Int}).\text{end}$ as a syntactic approach would suggest; neither descriptor does have a dual.

form $\text{begin}.\eta$ (and not, say, $\text{begin}.\eta \vee \text{begin}.\eta'$), which yields the following interpretation for session types:

$$\llbracket \text{begin}.\eta \rrbracket = \{c^{\text{begin}.\theta} \mid \theta \leq \eta\} \quad (7)$$

The next step is to formally define the duality relation.

2.2.2 Semantics of session descriptors

The formal semantics of a descriptor can be given by resorting to the labelled transition system (LTS) defined by the rules

$$\begin{array}{c} \text{(TR1)} \qquad \qquad \text{(TR2)} \qquad \qquad \text{(TR3)} \\ \hline \text{end} \xrightarrow{\checkmark} \text{end} \qquad \eta \oplus \eta' \longrightarrow \eta \qquad \eta + \eta'' \longrightarrow \eta' + \eta'' \\ \hline \text{(TR4)} \qquad \qquad \text{(TR5)} \qquad \qquad \text{(TR6)} \\ \hline \eta \xrightarrow{\mu} \eta' \qquad \eta \xrightarrow{\text{!v}} \eta'' \qquad \eta \xrightarrow{\text{!}\eta''} \eta''' \\ \hline \eta + \eta'' \xrightarrow{\mu} \eta' \qquad \eta + \eta' \longrightarrow \eta \qquad \eta + \eta' \longrightarrow \eta \\ \hline \text{(TR7)} \qquad \text{(TR8)} \qquad \text{(TR9)} \qquad \text{(TR10)} \\ \hline \text{v} \in \mathbf{t} \qquad \text{v} \in \mathbf{t} \qquad \eta \in \chi \qquad \eta \in \chi \\ \hline \text{?t}.\eta \xrightarrow{\text{?v}} \eta \qquad \text{!t}.\eta \xrightarrow{\text{!v}} \eta \qquad \text{?}\chi.\eta' \xrightarrow{\text{?}\eta} \eta' \qquad \text{!}\chi.\eta' \xrightarrow{\text{!}\eta} \eta' \end{array}$$

plus the symmetric of rules (TR2-TR6). In the rules μ ranges over actions of the form !v , or ?v , or $\text{!}\eta$, or $\text{?}\eta$, or \checkmark .

Rules (TR1-TR4) are straightforward: end emits a “tick” (TR1); an internal choice silently decides the behaviour it will successively follow (TR2); an external choice either performs an internal silent move (TR3) or it emits a signal μ that it offers as a possible choice to the interacting partner (TR4). Note that internal moves in one branch of an external choice do not preempt the behaviour of the other branch. This is typical of process languages with two distinct choice operators, such as CCS without τ 's [9].

The remaining rules are somewhat less common. Rules (TR7-TR8) state that the synchronization is performed on single values (strictly speaking, on singleton types) rather than on generic types. This is closer to what happens in practice, since $\text{!t}.\eta$ indicates that the descriptor is ready to emit some value of type \mathbf{t} (TR8), while $\text{?t}.\eta$ indicates that the descriptor is ready to accept any value of type \mathbf{t} (TR7). While this approach is reminiscent of the so-called *early semantics* in process algebras [26] (but note that here it is applied at type level rather than at process level), there is a technical reason to use values rather than types, which we explain after defining the subsetting relation.

Rules (TR9-TR10) follow the same idea as (TR7-TR8), and state that actions on descriptors emit a more precise information than what they declare. To understand this point we need to give some details. First note that a session descriptor η , despite the fact that it is usually called “session type” in the literature, is not a “real” type since it does not type any value. Session descriptors do not classify values but, rather, they keep track of the residual conversation that is allowed on a given session channel (whose “real” type is of the form $\text{begin}.\eta$). Therefore we cannot directly apply the same technique as for rules (TR7-TR8) since there does not exist any value for session descriptors. To mimic the behaviour of rules (TR7-TR8) we resort to the informal semantics we described in Section 2.2.1 where a type is interpreted as the set of its values and a descriptor—actually, a sieve—as the set of its duals: therefore, as an action on a type emits the same action on its values, so an action on a sieve emits the same action on its duals, where we use $\eta \in \llbracket \chi \rrbracket$ to denote that $\eta \in \llbracket \chi \rrbracket$.²

² Rules (TR7-TR10) hide a circularity since both values and duals are defined in terms of the duality relation we are defining. Theorem 2.6 in Section 2.2.3 shows that this circularity is only apparent.

Rules (TR5-TR6) state that outputs are irrevocable. This is a characteristic peculiar to our system and is reminiscent of Castellani and Hennessy’s treatment of external choices in the asynchronous CCS [7]. Roughly speaking, imagine a process offering two different outputs in an external choice. Then we can think of two possible implementations for such a choice. In one case the choice is an abstraction for a simple handshaking protocol that the communicating processes engage in order to decide which value is exchanged. This implementation does not fit very well a distributed scenario where processes are loosely coupled and communication latency may be important. In the second—and in our opinion closer to practice—case, the sender process autonomously decides which value to send. Rules (TR5-TR6) state that the decision is irrevocable in the sense that the sender cannot revoke its output and try with the other one. This behaviour is obtained by rules (TR5-TR6) by assimilating an external choice over output actions to an internal choice in which the process silently decides to send some particular value. In this respect the symmetry of input and output actions in rules (TR7-TR8)—but the same holds for (TR9-TR10) as well—may be misleading: we implicitly assumed that when a process waits for a value of type \mathbf{t} it is ready to accept *any* value of type \mathbf{t} (the choice of the particular value is left to the sender) while when a process sends a value of type \mathbf{t} , it internally decides a *particular* value of that type. We will break this symmetry in the formal notion of duality (Definition 2.5) to be defined next.

2.2.3 Duality

The discussion on the labelled transition system suggests that two dual descriptors can either agree on termination (so both emit \checkmark) or one of the two descriptors autonomously chooses to send an output that the other descriptor must be ready to receive. In order to formalize the notion of duality it is then handy to characterize outputs (when an output action *may* happen) and inputs (when an input action *must* happen). As usual we write \Longrightarrow for the reflexive and transitive closure of \longrightarrow ; we write $\xRightarrow{\mu}$ for $\Longrightarrow \xrightarrow{\mu} \Longrightarrow$; we write $\eta \xrightarrow{\mu}$ if there exists η' such that $\eta \xrightarrow{\mu} \eta'$, and similarly for $\xRightarrow{\mu}$; we write $\eta \not\xrightarrow{\mu}$ if there exists no η' such that $\eta \xrightarrow{\mu} \eta'$.

DEFINITION 2.2 (May and Must Actions). *We say that η may output μ , written $\eta \downarrow \mu$, if there exists η' such that $\eta \xRightarrow{\mu} \eta' \not\xrightarrow{\mu}$ and $\eta' \xrightarrow{\mu}$ and μ is either !v , or $\text{!}\eta$, or \checkmark .*

We say that η must input μ , written $\eta \Downarrow \mu$, if $\eta \xRightarrow{\mu} \eta' \not\xrightarrow{\mu}$ implies $\eta' \xrightarrow{\mu}$ and μ is either ?v , or $\text{?}\eta$, or \checkmark .

As usual we write $\eta \not\Downarrow \mu$ if not $\eta \Downarrow \mu$ and $\eta \not\downarrow \mu$ if not $\eta \downarrow \mu$.

Intuitively $\eta \downarrow \mu$ states that for a particular internal choice η will offer an output μ as an option, while $\eta \Downarrow \mu$ states that the input μ will be offered whatever internal choice η will do. For example $\text{!Int.end} \oplus \text{end} \downarrow \text{!3}$ and $\text{!Int.end} \oplus \text{end} \downarrow \checkmark$; on the other hand we have $\text{!Int.end} + \text{end} \not\downarrow \checkmark$, since $\text{!Int.end} + \text{end} \not\xrightarrow{\checkmark} \text{end}$. Similarly we have $\text{?Int.end} \oplus \text{?Real.end} \Downarrow \text{?3}$ because the action ?3 is always guaranteed independently of the internal choice, whereas $\text{?Int.end} \oplus \text{?Real.end} \not\Downarrow \text{?}\sqrt{2}$ because $\text{?Int.end} \oplus \text{?Real.end} \longrightarrow \text{?Int.end}$ and $\text{?Int.end} \not\Downarrow \text{?}\sqrt{2}$.

The previous definition induces two notions of convergence. Clearly convergence is a necessary condition for a session descriptor to have a dual.

DEFINITION 2.3 (May and Must Converge). *We say that η may converge, written $\eta \downarrow$, if for all η' such that $\eta \xRightarrow{\mu} \eta' \not\xrightarrow{\mu}$ we have $\eta' \downarrow \mu$ for some μ . We say that η must converge, written $\eta \Downarrow$, if $\eta \Downarrow \mu$ for some μ . As usual, we use $\eta \not\downarrow$ and $\eta \not\Downarrow$ to denote their respective negations.*

Note that the two contractivity conditions of Definition 2.1 rule out behaviours involving infinite sequences of consecutive internal

decisions. Therefore we will only consider strongly convergent processes, namely processes for which there does not exist an infinite sequence of \longrightarrow reductions.

The labelled transition system describes the *subjective evolution* of a session descriptor from the point of view of the process that uses a communication channel having that (residual) type. The last notion we need allows us to specify the evolution of a session descriptor from the dual point of view of the process at the other end of the communication channel. For example, we have $?Real.!Int.end + ?Int.!Bool.end \xrightarrow{?3} !Bool.end$ (the process receiving the integer value 3 knows that it has taken the right branch and now will send a Boolean value). However, the process sending the integer value 3 on the other end of the communication channel does not know whether the receiver has taken the left or the right branch, and both branches are actually possible. From the point of view of the sender, it is as if the receiver will behave according to the session descriptor $!Int.end \oplus !Bool.end$, which accounts for all of the possible states in which the receiver can be after the reception of 3. The *objective evolution* of a session descriptor after an action μ is defined next.

DEFINITION 2.4 (Successor). *Let $\eta \xrightarrow{\mu}$. The successor of η after μ , written $\eta\langle\mu\rangle$, is defined as: $\eta\langle\mu\rangle = \oplus \{\eta' \mid \eta \xrightarrow{\mu} \eta'\}$.*

For example, $(?Real.!Int.end + ?Int.!Bool.end)\langle?3\rangle = !Int.end \oplus !Bool.end$ but $(?Real.!Int.end + ?Int.!Bool.end)\langle?\sqrt{2}\rangle = !Int.end$. Note that $\eta\langle\mu\rangle$ is well defined because there is always a finite number of residuals η' such that $\eta \xrightarrow{\mu} \eta'$. This is a direct consequence of the contractivity conditions on session descriptors.

We now have all the ingredients for formally defining duality.

DEFINITION 2.5 (Duality). *Let the dual of a label μ , written $\bar{\mu}$, be defined by: (i) $\bar{\checkmark} = \checkmark$; (ii) $\bar{\dagger v} = \dagger v$; (iii) $\bar{\dagger \eta} = \dagger \eta$; where $\bar{!} = ?$ and $\bar{?} = !$. Then $\eta_1 \bowtie \eta_2$ is the largest symmetric relation between session descriptors such that one of the following condition holds:*

1. $\eta_1 \Downarrow \checkmark$ and $\eta_2 \Downarrow \checkmark$;
2. $\eta_1 \downarrow$ and $\eta_1 \downarrow \mu$ implies $\eta_2 \Downarrow \bar{\mu}$ and $\eta_1\langle\mu\rangle \bowtie \eta_2\langle\bar{\mu}\rangle$ for every μ .

The intuition behind the above definition is that a dual *must* accept every input that its partner *may* output, or they must both agree on termination. For example, we have $?Real.!Int.end + ?Int.!Bool.end \bowtie !Int.(?Int \vee Bool).end$, but $?Real.!Int.end + ?Int.!Bool.end \not\bowtie !Int.?Int.end$ because the descriptor on the right is not sure that its partner will answer with an integer. However $?Real.!Int.end + ?Int.!Bool.end \bowtie !(Real \setminus Int).?Int.end$. As another example, we have $?Int.end \oplus ?Real.end \bowtie !Int.end$ because $?Int.end \oplus ?Real.end \Downarrow ?v$ for every $v \in Int$, however $?Int.end \oplus ?Real.end \not\Downarrow !\sqrt{2}.end$ because $?Int.end \oplus ?Real.end \not\Downarrow \sqrt{2}$.

The reader may have observed that there is a circularity in the definitions of duality and of the labelled transition system. This is evident in rules (TR9-TR10) since the rules emit a dual of the sieve; that is, the relation $\eta \in \chi$ is defined in terms of $\llbracket \chi \rrbracket$ whose definition is given in terms of the duality relation. Less evident is the circularity of rules (TR7-TR8), which resides in the fact that these rules emit values of a given type; if this type has the form $begin.\eta$, then its values are all the channels of the form $c^{begin.\eta'}$ such that $\theta \bowtie \eta'$ implies $\theta \bowtie \eta$ for all θ (cf. equations (6) and (7)): so also the definition of the relation $v \in t$ depends on that of duality. The following theorem proves that this circularity is not one.

THEOREM 2.6 (Well-foundedness). *The definitions of $\eta \in \chi$, $v \in t$ and $\eta \bowtie \eta'$ are well founded.*

A corollary of this theorem is that the definitions of sub-sessioning $\eta \leq \eta'$ and sub-sieving $\chi \leq \chi'$ (the former being a special case of the latter) given by the equation (6) in Section 2.2.1 are well founded as well.

Using the definition of duality it is easy to see that $\llbracket \eta \oplus \eta' \rrbracket = \llbracket \eta \wedge \eta' \rrbracket$ since the duals of an internal choice must comply with both possible choices and thus be duals of both of them. Using this property it is easy to prove that sieves satisfy a disjunction property even stronger than the one for types, as the disjunction holds not only for single elements but for all the subsets of a union:

PROPOSITION 2.7. $\theta \leq \chi_1 \vee \chi_2 \iff \theta \leq \chi_1 \text{ or } \theta \leq \chi_2$.

This property is essential to prove decidability of \leq .

REMARK 2.8 (On the usefulness of sieves). *Since $\llbracket \eta \oplus \eta' \rrbracket = \llbracket \eta \wedge \eta' \rrbracket$ one may wonder whether we could not have spared one (or two) syntactic category(ies) by using descriptors (and/or, merging them with types) instead of sieves. The answer follows from the observation that the two operators above play rather different, but equally essential, roles: the internal choice is a behavioural operator, whereas sieve conjunction, as all sieves, is a pattern operator (whence the name “sieve”). We claim that any attempt to unify these two operators, by merging session descriptors and sieves together, disrupts the whole theory. The reason is that the notion of duality, which is intrinsically a behavioural one, does not match well with disjunction (which does not coincide with the external choice) and negation (defining the transition system of a negated session descriptor is puzzling at least). Overall the separation of behaviours and patterns is what allows us to provide semantic characterizations of all the notions in the formalism. The alternatives to sieves that we considered along the way all resulted in either uneasy syntactical restrictions or incompleteness results.*

Irrevocable outputs. By making external choices on output actions behave as internal ones, rules (TR5-TR6) state that outputs are irrevocable. This design choice was already explained in Section 2.2. In terms of duality, this choice corresponds to deciding whether, say, the external choices $!Int.end + ?Bool.end$ and $?Int.end + !Bool.end$ are to be considered as dual. In our setting the answer is negative as we consider that outputs may be asynchronously emitted even for external choices, therefore the two partners can get stuck if both decide to emit their outputs. This behaviour is a direct consequence of rules (TR5-TR6). As we discuss in the conclusion of this presentation, this is not the only reasonable answer. For instance, we could suppose that in a case such as the one above, the two partners perform some form of handshake to decide which one will perform the output; in that case rules (TR5-TR6) should be removed. We chose not to do so since the “irrevocable inputs” solution seems better fit a wide area network usage scenario.

2.3 Subtyping

Now that we have defined the duality relation, and therefore sub-sessioning, we can also formally define the subtyping relation. The types defined in Section 2.1 include three type combinators (union, intersection, and negation), one type constructor $begin.\eta$, plus other basic types and type constructors (inherited from the host language) that we left unspecified (typically, $Real$, $Bool$, \times , ...). We define the subtyping relation semantically using the technique introduced in [13] and outlined in Section 2.2.1, according to which types are interpreted as the set of their values, type combinators are interpreted as the corresponding set-theoretic operations, and subtyping is interpreted as set containment. As a consequence, testing a subtyping relation is equivalent to testing whether a type is empty, since by simple set-theoretic transformations we have that $t_1 <: t_2$ if and only if $t_1 \wedge \neg t_2 <: \emptyset$ (where we use \emptyset to denote the

empty type, that is the type that has no value). Again by simple set-theoretic manipulations, every type can be rewritten in disjunctive normal form, that is a union of intersections of types. Furthermore, since type constructors are pairwise disjoint (there is no value that has both a session type and, say, a product type—or whatever type constructor is inherited from the host language), then these intersections are uniform since they intersect either a given type constructor, or its negation (see [5, 13] for details). In conclusion, in order to define our subtyping relation all we need is to decide when $\bigvee_{k \in K} (\bigwedge_{i \in I_k} \text{begin}.\eta_i \wedge \bigwedge_{j \in J_k} \neg \text{begin}.\eta_j) <: \emptyset$. Since a union of sets is empty if and only if every set in the union is empty, by applying the usual De Morgan laws we can reduce this problem to deciding the inclusion $\bigwedge_{i \in I} \text{begin}.\eta_i <: \bigvee_{j \in J} \text{begin}.\eta_j$.

As regards session channels, we notice that a value has type $(\text{begin}.\eta) \wedge (\text{begin}.\eta')$ if and only if it has type $\text{begin}.\eta \oplus \eta'$. Also note that $\text{begin}.\eta <: \text{begin}.\eta_1 \vee \text{begin}.\eta_2$ if and only if $\text{begin}.\eta <: \text{begin}.\eta_1$ or $\text{begin}.\eta <: \text{begin}.\eta_2$. Therefore the semantic subtyping relation for the types of Section 2.1 is completely defined by (the semantic subtyping framework of [13] and) the following equation

$$\bigwedge_{i \in I} \text{begin}.\eta_i <: \bigvee_{j \in J} \text{begin}.\eta_j \iff \exists j \in J : \bigoplus_{i \in I} \eta_i \leq \eta_j \quad (8)$$

Note that when in the equation above I and J are singletons it reduces to

$$\text{begin}.\eta_1 <: \text{begin}.\eta_2 \iff \eta_1 \leq \eta_2$$

that is the form discussed at the end of Section 2.2.1. A consequence of this last observation is that $\llbracket \text{begin}.\eta \rrbracket = \{c^{\text{begin}.\eta'} \mid \eta' \leq \eta\} = \{c^{\text{begin}.\eta'} \mid \text{begin}.\eta' <: \text{begin}.\eta\} = \{v \mid v \in \text{begin}.\eta\}$. Thus our initial interpretation of session types coincides with the interpretation of types as sets of their values: we have “closed the circle” in the sense of [13].

2.4 Coinductive characterizations

The subsessioning relation defined in terms of duality embeds the notion of safe substitutability because of its very definition, but it gives little insight on the properties enjoyed by \leq . This is a common problem of every semantically defined preorder relation based on *tests*, such as the well-known testing preorders [8] (the set of duals of a descriptor can be assimilated to the set of its successful tests). In order to gain some intuition over \leq and to obtain a useful tool that will help us studying its properties we will now provide an alternative coinductive characterization. Before doing so, we need to characterize the class of descriptors that admit at least one dual. Recall that η is *viable* if there exists η' such that $\eta \bowtie \eta'$. Any non-viable descriptor is the least element of \leq , which henceforward will be denoted by \perp .

DEFINITION 2.9 (Coinductive Viability). η^\bowtie is the largest predicate over descriptors such that either

1. $\eta \downarrow$ and $\eta \downarrow \mu$ implies $\eta\langle\mu\rangle^\bowtie$ for every μ , or
2. there exists μ such that $\eta \downarrow \mu$ and $\eta\langle\mu\rangle^\bowtie$.

The definition provides us with a correct and complete characterization of viable descriptors:

PROPOSITION 2.10. η^\bowtie if and only if η is viable.

We can now read the statement of Definition 2.9 in the light of the result of the above proposition: Definition 2.9 explains that a descriptor is viable if either (1) it emits an output action regardless of its internal state and every successor after every possible output action is viable too or (2) it guarantees at least one input action such that the corresponding successor is viable too.

DEFINITION 2.11 (Coinductive Subsession). $\eta \leq \eta'$ is the largest relation between session descriptors such that η^\bowtie implies η'^\bowtie and

1. $\eta' \not\downarrow$ and $\eta' \downarrow \mu$ imply $\eta \downarrow \mu$ with $\eta\langle\mu\rangle \leq \eta'\langle\mu\rangle$, and
2. $\eta \downarrow \mu$ and $\eta\langle\mu\rangle^\bowtie$ imply $\eta' \downarrow \mu$ with $\eta\langle\mu\rangle \leq \eta'\langle\mu\rangle$, and
3. $\eta \downarrow$ and $\eta' \downarrow$ imply $\eta \downarrow \checkmark$ and $\eta' \downarrow \checkmark$.

The definition states that any viable descriptor η may be a sub-session of η' only if η' is also viable. This is obvious since we want the duals of η to be duals of η' as well. Furthermore, condition (1) requires that any output action emitted by the larger descriptor must also be emitted by the smaller descriptor, and the respective continuations must be similarly related. This can be explained by noticing that a descriptor dual of η in principle will be able to properly handle only the outputs emitted by η ; thus in order to be also dual of η' it must also cope with η' outputs, which must thus be included in those of η , hence the condition. The requirement $\eta' \not\downarrow$ makes sure that η' really emits some output actions. Without this condition we would have $?Int.end \not\leq ?Int.end + end$ as the descriptor on the r.h.s. emits \checkmark which is not emitted by the l.h.s. However, it is trivial to see that $?Int.end \leq ?Int.end + end$. Condition (2) requires that any input action guaranteed by the smaller descriptor must also be guaranteed by the larger descriptor. Again this can be explained by noticing that a descriptor dual of η may rely on the capability of η of receiving a particular value/descriptor in order to continue the interaction without error. Hence, any guarantee provided by the smaller descriptor η must be present in the larger descriptor η' as well. The additional condition $\eta\langle\mu\rangle^\bowtie$ considers only guaranteed input actions that have a viable dual, for a guaranteed input action with a non-viable dual is practically useless. Without such condition we would have, for instance, that $?Int.!0.end + ?Bool.end \not\leq ?Bool.end$, because the descriptor on the l.h.s. guarantees the action $?3$ which is not guaranteed by the descriptor of the r.h.s. of $\not\leq$. It is clear however that in this case the subsessioning relation must hold since the l.h.s. and r.h.s. have the same set of duals. Finally, condition (3) captures the special case in which a descriptor emitting output actions ($\eta \downarrow$) is smaller than a descriptor guaranteeing input actions ($\eta' \downarrow$). This occurs only when η may internally decide to terminate ($\eta \downarrow \checkmark$) and η' guarantees termination ($\eta' \downarrow \checkmark$). In this case, every dual of η must be ready to terminate and to receive any output action emitted by η , hence it will also be dual of η' which guarantees termination but does not emit any output action.

We end this subsection by stating that the coinductive and the semantic definitions of subsessioning coincide, so from now on we will use \leq to denote both.

THEOREM 2.12. $\eta_1 \leq \eta_2 \iff \eta_1 \leq \eta_2$.

2.5 Algorithms

In order to use our type system we must be able to decide the relations we introduced in the previous sections, namely subsieving (and subsessioning), subtyping, and duality.

Subsieving. Let us start to show how to decide that a sieve is smaller than another. Since Boolean combinators have a set-theoretic interpretation we can apply exactly the same reasoning we did for types in Section 2.3. Namely, deciding $\chi \leq \chi'$ is equivalent to deciding $\chi \wedge \neg\chi' \leq \perp$. The l.h.s. can be rewritten in disjunctive normal form whose definition for sieves is (we convene that $\bigvee_{i \in \emptyset} \chi_i = \sum_{i \in \emptyset} \eta_i = \perp$):

DEFINITION 2.13 (Disjunctive normal form). A sieve is in disjunctive normal form if it is of the form $\bigvee_{i \in I} \bigwedge_{j \in J} \lambda_{ij}$, where λ_{ij} denote descriptor literals, that is either η or $\neg\eta$.

Next, we can check emptiness of each element of the union separately, reducing the problem to checking the following relation:

$\bigwedge_{i \in I} \eta_i \leq \bigvee_{j \in J} \eta_j$. Since this is equivalent to $\bigoplus_{i \in I} \eta_i \leq \bigvee_{j \in J} \eta_j$, we can apply the strong disjunction property (Proposition 2.7) we stated for descriptors and obtain

$$\bigwedge_{i \in I} \eta_i \leq \bigvee_{j \in J} \eta_j \iff \exists j \in J : \bigoplus_{i \in I} \eta_i \leq \eta_j$$

which is precisely the same problem that has to be solved in order to decide the subtyping relation (cf. equation (8)). In conclusion, in order to decide both subsieving and subtyping it suffices to decide subsessioning.

Subsessioning. To decide whether two descriptors are in subsessioning relation we define a normal form for descriptors and, more generally, sieves (the latter occurring in the prefixes of the former).

DEFINITION 2.14 (Strong normal form). A sieve χ in disjunctive normal form is in strong normal form if

1. if $\chi \equiv \bigvee_{i \in I} \bigwedge_{j \in J} \lambda_{ij}$, then for $i \in I, j \in J$, λ_{ij} is in strong normal form and $\bigwedge_{j \in J} \lambda_{ij} \neq \perp$ for all $i \in I$;
2. if $\chi \equiv \neg\eta$, then η is in strong normal form;
3. otherwise χ is either of the form $\bigoplus_{i \in I} !\psi_i.\eta_i\{\oplus \text{end}\}$ or $\sum_{i \in I} ?\psi_i.\eta_i\{+\text{end}\}$, where for all $i \in I$, $\psi_i \neq \emptyset$, ψ_i and η_i are in strong normal form and for all $i, j \in I$, $i \neq j$ implies $\psi_i \wedge \psi_j = \emptyset$, and end is possibly missing.

Transformation in strong normal form is effective:

THEOREM 2.15 (Normalization). For every sieve χ it is possible to effectively construct χ' in strong normal form such that $\chi = \chi'$.

Finally, to check that two descriptors are in relation we rewrite both of them in strong normal form, check that neither is \perp , and then apply the algorithm whose core rules are given below:

(END)	(PREFIX)	(MIX-CHOICES)
$\frac{}{\text{end} \leq \text{end}}$	$\frac{\eta \leq \eta'}{\alpha.\eta \leq \alpha.\eta'}$	$\frac{}{\bigoplus_{i \in I} \eta_i \oplus \text{end} \leq \sum_{j \in J} \eta'_j + \text{end}}$
(EXT-CHOICES)	(INT-CHOICES)	
$\frac{I \subseteq J \quad \eta_i \leq \eta'_i \ (\forall i \in I)}{\sum_{i \in I} \eta_i \leq \sum_{j \in J} \eta'_j}$	$\frac{J \subseteq I \quad \eta_j \leq \eta'_j \ (\forall j \in J)}{\bigoplus_{i \in I} \eta_i \leq \bigoplus_{j \in J} \eta'_j}$	

Rule (MIX-CHOICES) states that an internal choice is smaller than an external one if and only if they both have an end summand. Rule (EXT-CHOICES) states that it is safe to widen external choices whereas rule (INT-CHOICES) states that it is safe to narrow internal ones. Both rules are used in conjunction with (PREFIX), which states covariance over descriptor continuations. Note that rule (PREFIX) relates two descriptors only if they have the same prefix. Therefore before applying (EXT-CHOICES) and (INT-CHOICES) we have to transform the descriptors so that prefixes on the two sides that have a non-empty intersection are rewritten in several summands so as to find the same prefix on both sides. This can be done by repeatedly applying, among others, the following decomposition laws:

$$\begin{aligned} ?t.\eta + ?s.\eta' &= ?(t \setminus s).\eta + ?(s \setminus t).\eta' + ?(t \wedge s).(\eta \oplus \eta') \\ !t.\eta \oplus !s.\eta' &= !(t \setminus s).\eta \oplus !(s \setminus t).\eta' \oplus !(t \wedge s).(\eta \oplus \eta') \end{aligned}$$

the latter rule holding when none of the sets $t \setminus s$, $s \setminus t$, and $t \wedge s$ is empty. Similar rules can be derived for inputs and outputs of sieves, as opposed to types.

THEOREM 2.16 (Soundness and Completeness). The algorithm is sound and complete with respect to \leq and it terminates.

Duality. Duality can be reduced to subsessioning since $\eta \bowtie \eta'$ if and only if $\bar{\eta} \leq \eta'$, where we write $\bar{\eta}$ for the canonical dual of η , namely the least descriptor in the set-theoretic interpretation of η . Computing $\bar{\eta}$ is trivial once η is in strong normal form (see Theorem 2.15): it suffices to change every $?$ into $!$, every $+$ into \oplus and vice versa, and to coinductively apply the transformation to the continuations leaving end descriptors unchanged. Regularity ensures that the transformation terminates (by using memoization techniques) and showing that the obtained session descriptor is the canonical dual of η is a trivial exercise.

3. Typing the π -calculus with sessions

The most interesting observation on process typing is that with our framework sessions can be typed in the standard π -calculus with internal/external choices and bound/free outputs (with the single modification on the type-based selection of external choices we described in the introduction), without primitive operators tailored to session-based communications (in the spirit of Kobayashi's [24]). The intuition is that bound outputs, written $c!(x : \eta)$, are session initiations where c is the public channel of the session and x the session private channel of descriptor η ; free outputs are reserved for session communications/delegations, and inputs are either session communications or session connections according to whether they are meant to synchronize with free or bound outputs, respectively. For example, the following process models a node that handles communications described by a protocol η and delegates communications described by unknown protocols to a sibling node, in a token-ring fashion:

```

NODE(mypublicname, nextpublicname,  $\eta$ ,  $P$ ) =
  mypublicname?(x : ?T.end).           /* accept          */
    x?(y :  $\eta$ ).P                         /* catch&handle    */
  + x?(y :  $\neg\eta$ ).                       /* catch&delegate  */
    nextpublicname!(z : !T.end).        /* request         */
    z!y                                  /* throw           */

```

The node waits for a connection on its public channel `mypublicname` and, once the connection is made, catches on the established session channel x a delegated session y of an arbitrary descriptor (T is the top sieve). If the delegated session is of protocol η (this is checked by using an external choice), then the node handles the delegated session in the process P , otherwise it connects to a sibling node `nextpublicname` (via a bound output) and delegates y to it (via a free output). Both `mypublicname` and `nextpublicname` are public channels of type `begin.!T.end`. Under these hypotheses and provided that y is used in P according to η , the process above results typeable by the type system given right after the formal definition of the syntax and semantics of our π -calculus for session types (PIST) we present next.

3.1 Syntax and semantics of PIST

The main design criteria for our process calculus PIST are minimality and, to a lesser extent, similarity to π -calculus: we define the smallest calculus that allows us to use all the characteristics of our session types. Notably, we preserve the distinction of [20] between public channels used to connect processes and private channels on which the conversation takes place. This distinction is instead eliminated in [15, 14, 29] where only one kind of output is needed. The syntax of PIST is given in Table 1. It is a π -calculus with internal and external choices, free outputs on bound channels and bound outputs on free channels respectively denoted by `h!e` and `a!(x : η)`. Processes are the possibly infinite regular trees that are generated by the productions in Table 1 and that satisfy the contractivity condition requiring that on every infinite branch there are infinitely many applications of the prefixed process. Contractivity rules out

processes of the form, for instance, $P = P \oplus P$ and—as for types—it provides an induction principle based on the Noetherian relation $P_1 + P_2 \triangleright P_i$ and $P_1 \oplus P_2 \triangleright P_i$.

(prefixes)	π	::=	$u?(x : \psi) \mid h!e \mid a!(x : \eta)$
(processes)	P, Q	::=	$\mathbf{0} \mid \pi.P \mid P \oplus P \mid P + P$
(public channels)	a	::=	$x \mid c^{\text{begin}.\eta}$
(private channels)	h	::=	$x \mid \boxed{k}$
(channels)	u	::=	$a \mid h$
(expressions)	e	::=	$u \mid \dots$
(systems)	\mathbb{S}, \mathbb{T}	::=	$P \mid \mathbb{S} \parallel \mathbb{S}$

Table 1. Syntax of PiST processes and systems.

The calculus does not include restrictions or parallel composition. Parallel composition is present only at the upper level of *systems* where session conversations take place. We do not need explicit restriction, those implicitly defined in bound outputs are enough³ and are implemented by resorting to *internal channels*, denoted by k . These appear greyed in the syntax to stress that such channels occur only at runtime (they cannot be written by the programmer but they are generated at session connection). We assume that “ $\tilde{\cdot}$ ” is a bijective mapping from internal channels to internal channels with $k \neq \tilde{k}$ and that is an involution (i.e., $\tilde{\tilde{k}} = k$, a technique quite common in calculi for sessions). We say that k and \tilde{k} are *dual*. Dual channels represent the two end-points of a session. Besides internal channels, expressions include channel variables (ranged over by x), channel values (i.e., session public names, ranged over by c and tagged by their smallest type, which is of the form $\text{begin}.\eta$ to enforce the strong disjunction property (5)), and the expressions of the host language (represented in the productions by dots). We use v to range over both channel values and host language values and suppose given the relation $v \in \mathbf{t}$ that associates each host language value with its types and each channel value with the types greater than or equal to its tag.

The set of labels ℓ is defined by

$$\ell ::= \tau \mid n?(x : \psi) \mid k!m \mid c^{\dagger}(x : \eta)$$

where $m ::= v \mid k$ and $n ::= c^{\dagger} \mid k$.

As customary, τ denotes internal silent moves, while the other labels are synchronization signals whose form is already quite informative. They tell us that synchronization can happen only on closed channels (ranged over by n : this excludes channel variables), that is, private channels for communication and public channels for session connection. Also, bound outputs can send only closed expressions (ranged over by m , that is values and private channels: the latter are not values though they operationally behave like them, since they are not associated to a type). Public channels appear tagged also in labels: though this tagging is not used it allows us to have more compact rules.

The labelled transition rules for processes are straightforward and they can be found in Table 2. They handle the irrevocability of outputs and compute expressions before synchronization. The semantics of systems is the standard π -calculus semantics apart the two points we evoked before, namely that outputs are irrevocable and that selection for external choices is based on the type of the arguments. This last point is showed by the synchronization rules for systems, presented in Table 3. Since the descriptors of internal channels evolve as long as the protocol advances, the label transition system uses *session environments*—i.e., maps from internal

³We could have introduced at system level restrictions of the form $(\nu c^{\text{begin}.\eta})$ so as to declare public session channels, limit their scope, and avoid explicit public channel tagging. We preferred to focus on a slightly simpler calculus.

R-CONNECT	R-RECEIVE	R-SEND
$\frac{}{c^{\dagger}(x : \eta).P \xrightarrow{c^{\dagger}(x:\eta)} P}$	$\frac{}{n?(x : \psi).P \xrightarrow{n?(x:\psi)} P}$	$\frac{e \downarrow m}{k!e.P \xrightarrow{k!m} P}$
R-INTCH	R-EXTINT	
$\frac{}{P \oplus Q \xrightarrow{\tau} P}$	$\frac{P \xrightarrow{\tau} P'}{P + Q \xrightarrow{\tau} P' + Q}$	
R-EXTSEND	R-EXTCH	
$\frac{P \xrightarrow{k!m} P'}{P + Q \xrightarrow{\tau} P}$	$\frac{P \xrightarrow{\ell} P' \quad \ell \neq \tau}{P + Q \xrightarrow{\ell} P'}$	

Table 2. PiST process reduction rules.

LIFT	PAR
$\frac{P \xrightarrow{\ell} P'}{\Sigma \vdash P \xrightarrow{\ell} \Sigma \vdash P'}$	$\frac{\Sigma \vdash \mathbb{S} \xrightarrow{\ell} \Sigma' \vdash \mathbb{S}'}{\Sigma \vdash \mathbb{S} \parallel \mathbb{T} \xrightarrow{\ell} \Sigma' \vdash \mathbb{S}' \parallel \mathbb{T}}$
CONNECTION	
$\frac{\Sigma \vdash \mathbb{S} \xrightarrow{c^{\dagger}(x:\eta)} \Sigma \vdash \mathbb{S}' \quad \Sigma \vdash \mathbb{T} \xrightarrow{c^{\dagger}(y:\eta')} \Sigma \vdash \mathbb{T}' \quad k \notin \text{dom}(\Sigma)}{\Sigma \vdash \mathbb{S} \parallel \mathbb{T} \xrightarrow{\tau} \Sigma, k : \eta, \tilde{k} : \eta' \vdash \mathbb{S}'[k/x] \parallel \mathbb{T}'[\tilde{k}/y]}$	
COMMUNICATION	
$\frac{\Sigma \vdash \mathbb{S} \xrightarrow{k!v} \Sigma \vdash \mathbb{S}' \quad \Sigma \vdash \mathbb{T} \xrightarrow{\tilde{k}?(x:t)} \Sigma \vdash \mathbb{T}' \quad v \in \mathbf{t}}{\Sigma, k : \eta, \tilde{k} : \eta' \vdash \mathbb{S} \parallel \mathbb{T} \xrightarrow{\tau} \Sigma, k : \eta(\! v), \tilde{k} : \eta'(\! v) \vdash \mathbb{S}' \parallel \mathbb{T}'[v/x]}$	
DELEGATION	
$\frac{\Sigma \vdash \mathbb{S} \xrightarrow{k!k'} \Sigma \vdash \mathbb{S}' \quad \Sigma \vdash \mathbb{T} \xrightarrow{\tilde{k}?(x:\chi)} \Sigma \vdash \mathbb{T}' \quad \Sigma(k') \leq \chi}{\Sigma, k : \eta, \tilde{k} : \eta' \vdash \mathbb{S} \parallel \mathbb{T} \xrightarrow{\tau} \Sigma, k : \eta(\! \Sigma(\tilde{k}')), \tilde{k} : \eta'(\! \Sigma(\tilde{k}')) \vdash \mathbb{S}' \parallel \mathbb{T}'[k'/x]}$	

Table 3. PiST system reduction rules.

channels to session descriptors, ranged over by Σ —to keep track of this evolution. The type-based dynamic branching is then implemented by the last two rules, according to which synchronization takes place only if the objects of the output match the type ($v \in \mathbf{t}$) or the sieve ($\Sigma(k') \leq \chi$) of the input. These two rules also record in the session environment that a synchronization step has been consumed and thus update the descriptors of the current session with the corresponding successor. Note also that a new session is started (rule CONNECTION) only when a bound output is performed on a channel value, in that case a new pair k, \tilde{k} of internal channels is spawned, and their descriptors are recorded in the session environment. Finally note that the dynamic checks in the last two rules are needed and used to drive the computation, since external choices are dynamically selected by using the type of the communicated value, or the descriptor of the delegated session.

We adopt the standard conventions of using $\xrightarrow{\tau}^*$ to denote $\xrightarrow{\tau}^*$ (i.e., the reflexive and transitive closure of $\xrightarrow{\tau}$) and $\xrightarrow{\ell}$ to denote $\xrightarrow{\tau} \xrightarrow{\ell} \xrightarrow{\tau}$.

3.2 Typing of PiST

The original motivation for introducing session types [18, 20] was to ensure that values sent and received in communication protocols were of appropriate types and that the two partners always agreed on how to continue the conversation. A type system ensuring also

T-AX $\frac{}{\Gamma, x : t \vdash x : t}$	T-VAL $\frac{v \in t}{\Gamma \vdash v : t}$	T-SUB $\frac{\Gamma \vdash e : t' \quad t' <: t}{\Gamma \vdash e : t}$	T-SYS $\frac{\Gamma \vdash P : \Delta}{\Gamma \Vdash P : \text{set}(\Delta)}$	T-PAR $\frac{\Gamma \Vdash \mathbb{S} : \Lambda_1 \quad \Gamma \Vdash \mathbb{T} : \Lambda_2}{\Gamma \Vdash \mathbb{S} \parallel \mathbb{T} : \Lambda_1 \cup \Lambda_2}$
T-ZERO $\frac{}{\Gamma \vdash \mathbf{0} : -}$	T-WEAK $\frac{\Gamma \vdash P : \Delta \quad h \notin \text{dom}(\Delta \cup \Gamma)}{\Gamma \vdash P : (h : \text{end} \cdot \Delta)}$		T-INT $\frac{\Gamma \vdash P : \Delta \quad \Gamma \vdash Q : \Delta}{\Gamma \vdash P \oplus Q : \Delta}$	
T-CONNECT-REQUEST $\frac{\Gamma \vdash P : (x : \eta \cdot \Delta) \quad \Gamma \vdash a : \text{begin}.\eta}{\Gamma \vdash a!(x : \eta).P : \Delta}$		T-CONNECT-ACCEPT $\frac{\Gamma \vdash P : (x : \eta \cdot \Delta) \quad \eta \bowtie \eta'}{\Gamma \vdash c^{\text{begin}.\eta'}?(x : \eta).P : \Delta}$		T-COMM $\frac{\Gamma \vdash_* P : \Delta}{\Gamma \vdash P : \Delta}$
T-RECEIVE $\frac{\Gamma, x : t \vdash P : (h : \eta \cdot \Delta)}{\Gamma \vdash_* h?(x : t).P : (h : ?t.\eta \cdot \Delta)}$	T-SEND $\frac{\Gamma \vdash e : t \quad \Gamma \vdash P : (h : \eta \cdot \Delta)}{\Gamma \vdash_* h!e.P : (h : !t.\eta \cdot \Delta)}$		T-RECEIVES $\frac{\Gamma \vdash P : (h : \eta \cdot x \cdot \eta') \quad \chi \leq \eta'}{\Gamma \vdash_* h?(x : \chi).P : (h : ?\chi.\eta)}$	
T-SENDS $\frac{\Gamma \vdash P : (h : \eta \cdot \Delta) \quad \eta' \leq \chi}{\Gamma \vdash_* h!h'.P : (h : !\chi.\eta \cdot (h' : \eta' \cdot \Delta))}$	T-INTCH $\frac{\Gamma \vdash_* P : (h : \eta_1 \cdot \Delta) \quad \Gamma \vdash_* Q : (h : \eta_2 \cdot \Delta)}{\Gamma \vdash_* P \oplus Q : (h : \eta_1 \oplus \eta_2 \cdot \Delta)}$		T-EXTCH $\frac{\Gamma \vdash_* P : (h : \eta_1 \cdot \Delta) \quad \Gamma \vdash_* Q : (h : \eta_2 \cdot \Delta)}{\Gamma \vdash_* P + Q : (h : \eta_1 + \eta_2 \cdot \Delta)}$	

Table 4. Type system for PiST.

the progress property, i.e., that a started session cannot get stuck if the required connections are available, was first proposed in [11].

In the present calculus, as in [10], the operational semantics itself ensures that there cannot be a type mismatch in communications, since all checks are performed at the moment of the synchronisation. So, for example, while the system $k!3 \parallel \tilde{k}?(x : \text{Bool})$ would be stuck, it cannot be generated by our processes since a CONNECTION rule can be executed only when the two session descriptors of the private channels are dual. In this section we present a type system which prevents also any deadlock due to the interleaving of two or more sessions.

More precisely we want to ensure that whenever a well-typed system is stuck (i.e., it cannot perform any internal reduction) it is because either all its processes have successfully terminated or at least one of them is on hold on a connection request. This means that whenever a session is started, if it does not perform any further connection, then either it eventually successfully terminates, or it continues to interact (recall that both process and session descriptors may be recursive). More formally:

DEFINITION 3.1 (Progress Property). *A system \mathbb{S} satisfies the progress property if $\vdash \mathbb{S} \xrightarrow{\tau} \Sigma \vdash \mathbb{S}' \xrightarrow{\tau} \text{implies that either } \mathbb{S}' \text{ does not contain internal channels or } \mathbb{S}' \xrightarrow{c^!(x:\eta)} \text{ or } \mathbb{S}' \xrightarrow{c^!(x:\eta)}$.*

Our process calculus, as the calculi in the literature of session types, is so close to the syntax of the session descriptors that it is not difficult to imagine how to map a given channel to its session type. For instance, consider the process $c^!(z : \eta).z!3.z?(x : \text{Real}).(z!x \oplus z!\text{true})$ which opens a connection on c in which it writes an integer, reads a real, and then decides whether to send back the received real or a Boolean value. It is clear that such a process is well typed when $\eta = !\text{Int}.\text{?Real}.(!\text{Real}.\text{end} \oplus !\text{Bool}.\text{end})$ and t is (a subtype of) $\text{begin}.\eta$. However, in order to ensure the progress property, the way in which a process uses *different* sessions must be quite limited. Once a connection is established, and a private channel, which we call the *current session*, is spawned, then the process can act according to (combinations of) the following options:

1. establish a new connection;
2. perform a communication (possibly paired with a branching) on the current session;

3. end the current session by stopping using the corresponding channel (there is no explicit end in processes, so the end of a session is reached when its channel is no longer used, for every possible continuation);
4. delegate on the current session the innermost, not ended, enclosing session;⁴ the process stops using the delegated session;
5. receive a delegated session and use it in the continuation as the current session.

Such restrictive behavior corresponds to using sessions as critical regions that forbid deadlocks on circular waits. Each critical region is associated with a particular internal channel: it is entered whenever this channel is received by delegation or started by a connect, it is closed when the channel is delegated or no longer used. Once a process has entered a critical region all it can do is to communicate on the channel associated with the region or to enter a new critical region. To see why these restrictions are necessary let us comment few examples of deadlock.

Let $t_1 = \text{begin}.\text{!Int}.\text{end}$, $\eta_1 = !\text{Int}.\text{end}$ and $\eta_2 = ?\text{Int}.\text{end}$. A first simple example of deadlock is given by

$$c^{t_1}!(z_1 : \eta_1).b^{t_1}?(z_2 : \eta_2).z_2?(x : \text{Int}).z_1!6 \parallel c^{t_1}!(z_3 : \eta_2).b^{t_1}!(z_4 : \eta_1).z_3?(x : \text{Int}).z_4!5 \quad (9)$$

After two executions of the CONNECTION rule, both processes starve waiting for values that are never sent. Note that the problem here is generated by the process that provides the service c (i.e., the second one) since it makes a communication on (the private channel of) c before having ended the session it requested on b .

Internal and external sums can produce deadlocks since they can make choices unavailable as in

$$c^{t_1}?(z_1 : \eta_2).b^{t_1}?(z_2 : \eta_2).z_1?(x : \text{Int}) + z_2?(x : \text{Int}) \parallel c^{t_1}!(z_3 : \eta_1).z_3!6 \parallel b^{t_1}!(z_4 : \eta_1).z_4!5 \quad (10)$$

and in the similar system obtained by replacing \oplus to $+$. The problem here is that the choice is performed on different (open) sessions: it should be either both on z_2 (we use the inner session) or both on z_1 (we closed the inner session and passed on the outer

⁴A special case is when the sieve is precisely end: in that case the process can delegate any non active channel.

one). Another example is

$$c^{t_1}?(z_1 : \eta_3).z_1?(x : \text{Int}) + b^{t_1}?(z_2 : \eta_2).z_2?(x : \text{Int}).z_1?(x : \text{Bool}) \\ \parallel c^{t_1}!(z_3 : \eta_1).z_3!6 \parallel b^{t_1}!(z_4 : \eta_1).z_4!5$$

where $\eta_3 = ?\text{Int}.\text{end} + ?\text{Bool}.\text{end}$: note that the connection on b forbids the communication on the private channel created by the connection on c .

Subtler examples of deadlock spring from session delegation, whereby a (sequential) process can receive the dual of a channel it already owns, making synchronization impossible. Consider

$$c^{t_1}?(z_1 : \eta_2).b^{t_2}!(z_2 : \eta_4).z_2!z_1 \parallel c^{t_1}!(z_3 : \eta_1). \\ b^{t_2}?(z_4 : \eta_5).z_4?(x : \eta_2).x?(y : \text{Int}).z_3!6 \quad (11)$$

where $t_2 = \text{begin}!(?\text{Int}.\text{end}).\text{end}$, $\eta_4 = !(\text{Int}.\text{end}).\text{end}$, $\eta_5 = ?(\text{Int}.\text{end}).\text{end}$. This phenomenon may also jeopardise subject reduction, as discussed in [30].

Such problems can be avoided by resorting to the strict usage discipline we described earlier which is enforced by the typing discipline defined in Table 4. The judgements for processes have two possible forms

$$\Gamma \vdash P : \Delta \quad \text{and} \quad \Gamma \vdash_* P : \Delta$$

where Γ is a *type environment* (a mapping from variables to types) and Δ is a *session stack*. The latter is a mapping from private channels to session descriptors to which identifiers of ended sessions can be freely added (rule T-WEAK) and is used to record the session descriptors of the channels used in P . It is organised as a stack (the leftmost element being the top) to keep track of the current session, that is the most recently created one (i.e., in our analogy, the one associated with the current critical region). The stack allows us to avoid the first example (9) of deadlock, by organising sessions as nested critical regions in which a channel cannot be used unless all nested sessions have been consumed (either because they ended or because they were delegated to some other process). Actions are allowed only if their subject is the current session channel, the one on the top of the stack (rules T-CONNECT-REQUEST, T-CONNECT-ACCEPT, T-SEND, T-RECEIVE, T-SENDS, and T-RECEIVES) and they are recorded in the conclusion. In addition, the rules for communication check that type constraints are satisfied while sieve constraints are checked by the delegation rules.

The rule T-CONNECT-ACCEPT hides several subtleties. If a process contains the action $c^{\text{begin}.\eta'}?(x : \eta).P$ it means that it provides the service c and it implements it by the process P . Note that in P the corresponding private channel implements a behavior dual of the one that tags c . Therefore the tag of a public channel declares the behavior that all clients of the service must follow. In other terms it describes the most demanding client this service is ready to serve. Also note that the typing rule imposes that the acceptance of a connect can only be written for an actual public channel and not for a variable of the corresponding type. This corresponds to the everyday practice that a service is associated to a particular URL instead of being dynamically bound to it (which does not prevent several processes to implement the same service) and enforces the assumption—we did in Section 2.2.1—that the only way to use a public service name is to call it. In other words, while the public names of services are first class and, as such, they can be passed around in communications, the only way to use them is to request a connection on them. Technically, this restriction allows us to avoid the use of polarities to type communications: because of the use of duality in the T-CONNECT-ACCEPT one should require contravariance for channels used for session acceptance, covariance for those used for session request and invariance for channels used in both cases. The solution we chose, besides being natural, corresponds to enforcing invariance when accepting connections.

To deal with the deadlocks induced by sums, illustrated by example (10), we use the \vdash_* judgements which ensure that the processes in the conclusion offer communications just on the channel that is on the top of the stack. Two processes can be composed by means of an internal choice only if either they are typed by exactly the same stack (rule T-INT) or if they differ for the typing of only one channel on which a communication is immediately available (rule T-INTCH). They can be composed by an external choice only in the latter case, that is, if they differ for the typing of only one channel on which a communication is immediately available (rule T-EXTCH).

The example of deadlock due to delegation, illustrated by example (11), is avoided by requiring that the only other internal channel that can occur in a process accepting a delegation is the channel on which the delegation took place (rule T-RECEIVE).

The typing discipline is lifted to systems by simply merging all channel assumptions, disregarding the order in which they appear (by means of the operator set), and obtaining in this way *session environments*, ranged over by Λ (these are the same as those in the dynamic semantics, but we preferred to use a different metavariable to avoid confusion).

Since evaluation consumes session descriptors and adds fresh initial channels with their descriptors, we need to introduce a partial order \preceq on session stacks and session environments, whose definition is as expected: if $\Lambda \preceq \Lambda'$, then Λ' is obtained from Λ by consuming descriptor actions and/or adding new descriptors of freshly initiated sessions. So that subject reduction can be formulated as follows.

THEOREM 3.2 (Subject Reduction for Processes). *If $\Gamma \vdash P : \Delta$ and $P \xrightarrow{\ell} P'$, then $\Gamma \vdash P' : \Delta'$, where $\Delta \preceq \Delta'$.*

THEOREM 3.3 (Subject Reduction for Systems). *If $\Gamma \Vdash \mathbb{S} : \Lambda$ and $\Sigma \vdash \mathbb{S} \xrightarrow{\ell} \Sigma' \vdash \mathbb{S}'$, then $\Gamma \Vdash \mathbb{S}' : \Lambda'$, where $\Lambda \preceq \Lambda'$.*

Progress clearly fails for systems that contain free variables or internal channels that are not properly paired. For this reason our typing can only ensure progress for initial systems, defined as follows:

DEFINITION 3.4 (Initial system). *A well-typed system is initial if it is the parallel composition of closed processes in which no internal channel occurs.*

The proof of progress depends on the remark that the session environments in the operational semantics and in the typing of systems respectively give the *objective* and *subjective* views of the internal channel behaviors. For example consider the system

$$\mathbb{S} = k!3.k?(x : 2 \vee 4) \parallel \tilde{k}?(y : 3).\tilde{k}!2 + \tilde{k}?(y' : 3).\tilde{k}!4.$$

which is formed by two processes that are carrying on a session over the internal channels k and \tilde{k} . We get

$$\{k : !3.?(2 \vee 4).\text{end}, \tilde{k} : ?3.!2.\text{end} + ?3.!4.\text{end}\} \vdash \mathbb{S} \xrightarrow{\tau} \Sigma' \vdash \mathbb{S}'$$

where $\Sigma' = \{k : ?(2 \vee 4).\text{end}, \tilde{k} : !2.\text{end} \oplus !4.\text{end}\}$ and $\mathbb{S}' = k?(x : 2 \vee 4) \parallel \tilde{k}!2$, while $\Vdash \mathbb{S}' : \{k : ?(2 \vee 4).\text{end}, \tilde{k} : !2.\text{end}\}$. The descriptor of \tilde{k} in Σ' is the internal choice between $!2.\text{end}$ and $!4.\text{end}$, since an observer does not know if the value 3 was received by the process $\tilde{k}?(y : 3).\tilde{k}!2$ or by the process $\tilde{k}?(y' : 3).\tilde{k}!4$. Instead the descriptor of \tilde{k} in the typing of \mathbb{S}' is $!2.\text{end}$, since the value 3 was received by the process $k?(y : 3).\tilde{k}!2$.

More precisely the session environments created in the operational semantics starting from an initial system assigns to internal channels descriptors equal to or smaller than the session environments used in typing. This is the content of the following lemma which is the cornerstone of the proof of progress.

LEMMA 3.5. *If \mathbb{S} is initial and $\vdash \mathbb{S} \xrightarrow{\tau} \Sigma \vdash \mathbb{S}'$, and $\Vdash \mathbb{S}' : \Lambda$, then $\Sigma(k) \leq \Lambda(k)$ for all k which occur in \mathbb{S}' .*

An interesting consequence of the lemma above is that our system satisfies the *fidelity* of communications [21], that is to say, that once a session is started communications happen in the expected order and they exchange data of the expected types.

We now state the progress theorem.

THEOREM 3.6 (Progress). *Every initial system satisfies the progress property.*

It is interesting to note that if we disregard the order in the session stack (i.e., we use session environments in all the rules) and we allow an arbitrary session stack in the premise of rule T-RECEIVE, then we get a type system which still enjoys subject reduction but no longer guarantees progress.

4. Related work and conclusions

We have defined a semantic theory of session types by subverting the usual session type presentations, where the subtyping (and subsession) relations are introduced first, and then shown to be sound. Here we have focused on duality as the main characterizing feature, and defined subtyping and subsessioning in terms of it. Our approach is akin to the testing approach to process semantics [8]: the notion of “passing a test” is embodied in our notion of duality, and subsessioning is the preorder induced by comparing the duals of two session types. In particular, two session types are equivalent if they have the same set of duals. Unlike the standard testing theories, our notion of duality is symmetric (in the spirit of the session types literature).

We see two main contributions of this work. First, we give semantic foundations to several concepts that can be found scattered in the literature of session types. Second, we provide Boolean combinations of session types. The price to pay is, of course, an increased complexity of the system and, to a lesser extent, of the syntax. For instance, the subtyping algorithm of [15] looks much simpler than the algorithm presented here. Nonetheless, our session types arise as a combination of orthogonal operators: those for communication and those for branching. This gives a straightforward solution to the problem of computing the meet of possibly recursive session types, which arises in those contexts where session types are *inferred* from processes as opposed to *checked* against processes [25]. Also, we shift from label-driven to type-driven branch selection. At first sight, this may seem a regression, insofar as it demands run-time type checking. This is not so in practice. First, when sessions are used as in any of the existing session types proposals, branching can be easily optimized by reducing run-time type checking to label/class matching. Second, general value-based dispatching can be implemented very efficiently anyway [12], with the exception of session type values which may require to check subsessioning. In any case, it is reasonable to assume that the matching overhead is negligible with respect to latency time of communications typical of service-oriented computing and that in a distributed, untrusted environment such as the Web, some kind of run-time checks will be needed anyhow.

From a technical viewpoint we introduce several novelties. We devise a new labelled transition system *for session descriptors* in which actions represent values rather than types, and we give a semantic characterization of the subsessioning relation in terms of a set-theoretic interpretation of session descriptors. The same interpretation is used to give semantics to a complete set of Boolean operators for session descriptors. With respect to concurrency theory we introduce an original treatment of output signals, by implementing a form of *partial asynchrony*. This treatment is similar to the one proposed by Castellani and Hennessy [7] for asynchronous

CCS, where outputs cannot be blocked even if they guard external choices (we dubbed this property “output irrevocability”). However, in our setting output signals are allowed to have a continuation. Thus the order of actions specified by a session type must be strictly followed (equivalence of session types modulo permutation of prefixes as in [19] is left for future work). Technically, this corresponds to observing inputs even in the presence of (partial) asynchrony.

The process language mixes and synthesizes several techniques that are scattered all over the literature. In particular it borrows the type-based dynamic selection of external choices and the technique of tagged channels from [4] and the use of polarized channels to ensure subject reduction from [15]; its typing discipline improves existing stack-based typing techniques [11], by sparing redundancies and resulting in compact and readable rules.

We already remarked that, unlike [22, 3], we did not include in our session descriptors a construct for parallel composition since in many cases they can be rendered by well-known expansion laws (see [9] for an example). Indeed while we think that internal and external choices are necessary to safely approximating the behaviour of a generic session (a general service must be able to offer some choices to a client and, according to the interaction with the client, make some internal choices that determine the prosecution of the session), we reckon that the introduction of parallel composition would limit the application of our theory to fewer programming languages. The reason is that the session interaction we consider here is a two-parties synchronization, therefore it can mostly be simulated by internal and external choices via some expansion laws. If session atomic synchronization involved more than two parties or offered the choice between several outputs, then this would no longer be true. By not introducing a parallel composition we let different type systems to use different expansion laws and thus type different kinds of parallel composition of processes (interleaving, restricted parallelism, asymmetric parallelism, and so on): if we added a parallel composition to our types we would limit the application of our theory only to calculi/languages in which the parallel composition of processes had the same semantics. In this respect we completely embrace the conclusions of [17].

There exist several related works both in general concurrency theory and more specifically in the rich literature on session types (see [30] and the references in there).

Session types are behavioural types and, as such, they can be put in the context of generic type systems [22], where types take the form of CCS terms describing the input/output flows of processes. Acciai and Boreale [1] use CCS terms as session types for the service calculus CASPIs [2]. In [22, 1] types are processes and, as such, they reduce according to a labelled transition system. This type reduction simulates the reduction of the processes they type. The subject reduction property states that if a process has some type T and it reduces to another process, then the reduced process has a type that can be obtained by reducing T . Our types simulate the process behaviour except that they do so after projecting the overall behaviour of a process on the various (session) channels it uses. So instead of having a CCS process that describes the communications of a process we have a set of pairs channel/projected communications that provides a partial, though somewhat more detailed view of the process’ communications. The view is partial because it does not capture the temporal dependencies between interactions occurring in different sessions, but it is more detailed than a plain CCS process because it describes interactions that have the same granularity (in terms of exchanged values/sessions) as those occurring in processes. In addition, we use the labelled transition system of session descriptors also to formally define their duality and then their semantics.

Subtyping relations for session types are studied in [28, 15, 10, 14, 16, 29]. In these works the definition of the subtyping relation is driven by the requirement that a channel of a smaller session type can be safely used wherever a channel of a greater session type is expected. This leads to the standard covariance and contra-variance of inputs and outputs, respectively [27]. Since in the cited works choices are guarded by labels, it turns out that external and internal choices have the same subtyping relation as record and variant types, respectively: an external choice is smaller than another external choice that offers more choices, whereas an internal choice is smaller than another internal choice that offers less choices. This is reflected in Section 2.5 by our (EXT-CHOICES) and (INT-CHOICES) rules, while the subtyping relation between external and internal choices (rule (MIX-CHOICES)) is a novelty of our approach. In all the mentioned works it holds the following relation between subtyping and duality, first stated and proved in [28]: $\eta \leq \eta'$ if and only if $\theta' \leq \theta$ for all $\theta \bowtie \eta$ and $\theta' \bowtie \eta'$. This relation implies the uniqueness of dual types modulo the equivalence relation induced by subtyping. Since we associate with each session descriptor a set of duals that is upward closed, this property cannot hold as stated in our setting. However it holds if one considers the canonical dual of a descriptor, that is, the smallest descriptor in a set of duals. In fact, it can be shown that every viable descriptor has a unique canonical dual and, remarkably, this is the one obtained by syntactically complementing normal forms (see Section 2.5). It is thus the working object of the algorithms and, though we gave a semantic characterization of it, it is more of syntactical nature and, as such, nicely fits the purely syntactical approach of the cited works.

As regards future research, the natural continuation of this work is to extend the semantic framework we propose to the features that we cannot account for yet. Specifically, we aim at studying polymorphism (to model session descriptors of [14]), exploring communication models other than output irrevocability, and extending our types to multi-party sessions. On the technical side, we aim at relaxing the contractivity conditions to be able to encode all the existing session types for dyadic interactions. Another restriction of the presented theory is that received public channels can only be used for performing request operations, whereas accept operations can only be performed on statically known, public channels. This restriction could be relaxed by designing two distinct session types, say $\text{begin}^?.\eta$ and $\text{begin}^!\eta$, classifying those public channels on which accept (respectively, request) operations can be performed. Then, their intersection would classify those public channels admitting both operations. Unfortunately, extending the framework with these two types causes a significant complication of the subtyping relation, and we were unable to produce a corresponding algorithm. Whether and how these two types can effectively co-exist is currently being investigated.

References

- [1] L. Acciai and M. Boreale. A type system for client progress in a service-oriented calculus. In *Concurrency, Graphs and Models*, volume 5065 of *LNCS*, pages 642–658. Springer, 2008.
- [2] M. Boreale, R. Bruni, R. De Nicola, and M. Loreti. Sessions and pipelines for structured service programming. In *FMOODS'08*, volume 5051 of *LNCS*, pages 19–38. Springer, 2008.
- [3] L. Caires and H. T. Vieira. Conversation types. In *ESOP'09*, *LNCS* 5502, pages 285–300. Springer, 2009.
- [4] G. Castagna, R. De Nicola, and D. Varacca. Semantic subtyping for the π -calculus. *Theor. Comput. Sci.*, 398(1-3):217–242, 2008.
- [5] G. Castagna and A. Frisch. A gentle introduction to semantic subtyping. In *PPDP '05*, pages 198–208. ACM Press (full version) and *ICALP '05*, *LNCS* n. 3580, pages 30–34. Springer (summary), 2005. Joint *ICALP-PPDP* keynote talk.
- [6] G. Castagna, N. Gesbert, and L. Padovani. A theory of contracts for web services. *ACM Transactions on Programming Languages and Systems*, 2009. Extended version of the article in *POPL '08*. To appear.
- [7] I. Castellani and M. Hennessy. Testing theories for asynchronous languages. In *FST&TCS '98*, volume 1350 of *LNCS*, pages 90–101. Springer, 1998.
- [8] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theor. Comput. Sci.*, 34:83–133, 1984.
- [9] R. De Nicola and M. Hennessy. CCS without τ 's. In *TAPSOFT/CAAP'87*, volume 249 of *LNCS*, pages 138–152. Springer, 1987.
- [10] M. Dezani-Ciancaglini, E. Giachino, S. Drossopoulou, and N. Yoshida. Bounded session types for object-oriented languages. In *FMCO'06*, volume 4709 of *LNCS*, pages 207–245. Springer, 2007.
- [11] M. Dezani-Ciancaglini, N. Yoshida, A. Ahern, and S. Drossopoulou. A distributed object oriented language with session types. In *TGC'05*, volume 3705 of *LNCS*, pages 299–318. Springer, 2005.
- [12] A. Frisch. Regular tree language recognition with static information. In *IFIP TCS'04*, pages 661–674. Kluwer, 2004.
- [13] A. Frisch, G. Castagna, and V. Benzaken. Semantic subtyping: dealing set-theoretically with function, union, intersection, and negation types. *Journal of the ACM*, 55(4):1–64, 2008.
- [14] S. Gay. Bounded polymorphism in session types. *MSCS*, 18(5):895–930, 2008.
- [15] S. Gay and M. Hole. Subtyping for session types in the pi-calculus. *Acta Informatica*, 42(2/3):191–225, 2005.
- [16] S. Gay and V. T. Vasconcelos. Linear type theory for asynchronous session types. Available online, 2008.
- [17] M. Hennessy and A. Ingólfssdóttir. A theory of communicating processes with value-passing. In *ICALP'90*, volume 443 of *LNCS*. Springer, 1990.
- [18] K. Honda. Types for dyadic interaction. In *CONCUR'93*, volume 715 of *LNCS*, pages 509–523. Springer, 1993.
- [19] K. Honda, D. Mostrous, and N. Yoshida. Global principal typing in partially commutative asynchronous sessions. In *ESOP'09*, *LNCS* 5502, pages 316–332. Springer, 2009.
- [20] K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In *ESOP'98*, volume 1381 of *LNCS*. Springer, 1998.
- [21] K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL'08*, pages 273–284. ACM, 2008.
- [22] A. Igarashi and N. Kobayashi. A generic type system for the pi-calculus. *Theor. Comput. Sci.*, 311(1-3):121–163, 2004.
- [23] N. Kobayashi. Type systems for concurrent programs. In *FMC'03*, *LNCS* 2757. Springer, 2003.
- [24] N. Kobayashi. Type systems for concurrent programs. Extended version of [23], Tohoku University, 2007.
- [25] L. G. Mezzina. How to infer finite session types in a calculus of services and sessions. In *COORDINATION'08*, volume 5052 of *LNCS*, pages 216–231. Springer, 2008.
- [26] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theor. Comput. Sci.*, 114(1):149–171, 1993.
- [27] B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Math. Struct. in Comp. Science*, 6(5):409–454, 1996.
- [28] A. Vallecillo, V. T. Vasconcelos, and A. Ravara. Typing the behavior of objects and components using session types. In *FOCLASA'02*, volume 68(3) of *ENTCS*, pages 439–456. Elsevier, 2002.
- [29] V. T. Vasconcelos. Fundamentals of session types. In *SFM'09*, volume 5569 of *LNCS*, pages 158–186. Springer, 2009.
- [30] N. Yoshida and V. T. Vasconcelos. Language primitives and type disciplines for structured communication-based programming revisited. In *SecRet'06*, volume 171(4) of *ENTCS*, pages 73–93. Elsevier, 2007.

A. Supplement to Section 2

A.1 Properties of the subsession relation

Table 5 shows some relevant rules regarding \leq . Aside from providing further insight on the properties of \leq , these rules are also used in the following for proving the existence of the normal forms for session descriptors and the correctness of the algorithms. In the table we write \emptyset to denote either \emptyset (the empty type) or \perp (the least sieve) according to the context.

(E1)	$\eta + \eta = \eta$	
(E2)	$\eta + \eta' = \eta' + \eta$	
(E3)	$\eta + (\eta' + \eta'') = (\eta + \eta') + \eta''$	
(E4)	$\eta + (\eta' \oplus \eta'') = (\eta + \eta') \oplus (\eta + \eta'')$	
(E5)	$\alpha.\eta + \alpha.\eta' = \alpha.(\eta \oplus \eta')$	
(E6)	$?t.\eta + ?s.\eta = ?(t \vee s).\eta$	
(E7)	$?x.\eta + ?x'.\eta = ?(\chi \vee \chi').\eta$	
(E8)	$\eta + \perp = \eta$	
(I1)	$\eta \oplus \eta = \eta$	
(I2)	$\eta \oplus \eta' = \eta' \oplus \eta$	
(I3)	$\eta \oplus (\eta' \oplus \eta'') = (\eta \oplus \eta') \oplus \eta''$	
(I4)	$\eta \oplus (\eta' + \eta'') = (\eta \oplus \eta') + (\eta \oplus \eta'')$	
(I5)	$\alpha.\eta \oplus \alpha.\eta' = \alpha.(\eta \oplus \eta')$	
(I6)	$?t.\eta \oplus ?s.\eta' = ?(t \wedge s).(\eta \oplus \eta')$	
(I7)	$?x.\eta \oplus ?x'.\eta' = ?(\chi \wedge \chi').(\eta \oplus \eta')$	
(I8)	$\eta \oplus \perp = \perp$	
(B1)	$? \psi.\eta = \perp$	$(\psi = \emptyset)$
(B2)	$! \psi.\eta = \perp$	$(\psi = \emptyset)$
(B3)	$?t.\eta \oplus ?x.\eta' = \perp$	
(B4)	$? \psi.\eta \oplus ! \psi'.\eta' = \perp$	
(B5)	$? \psi.\eta \oplus \text{end} = \perp$	
(O1)	$! \psi.\eta + \text{end} = ! \psi.\eta$	$(\psi \neq \emptyset)$
(O2)	$! \psi.\eta + ? \psi'.\eta' = ! \psi.\eta$	$(\psi \neq \emptyset)$
(O3)	$! \psi.\eta + ! \psi'.\eta' = ! \psi.\eta \oplus ! \psi'.\eta'$	$(\psi, \psi' \neq \emptyset)$
(O4)	$!t.\eta \oplus !s.\eta = !(t \vee s).\eta$	$(t, s \neq \emptyset)$
(O5)	$!x.\eta \oplus !x'.\eta = !(x \vee x').\eta$	$(x, x' \neq \perp)$
(S1)	$?t.\eta \leq ?(t \vee s).\eta$	
(S2)	$?x.\eta \leq ?(\chi \vee \chi').\eta$	
(S3)	$!(t \vee s).\eta \leq !t.\eta$	$(t \neq \emptyset)$
(S4)	$!(x \vee x').\eta \leq !x.\eta$	$(x \neq \perp)$
(S5)	$\eta \oplus \eta' \leq \eta$	

Table 5. Selected equalities and inequalities.

Rules (E1–E8) state the fundamental properties of the external choice operator. Rules (E1–E4) are trivial being the usual idempotency, commutativity, associativity and distributivity laws of external choices. Rule (E5) shows that an external choice may actually hide an internal choice if it combines descriptors having a common prefix. This is a well-known axiom in the testing theories [8] and it also shows that the external choice does not coincide with the set-theoretic union operator (the internal choice, on the other hand, does coincide with the set-theoretic intersection). Rule (E6) shows the interaction between input actions (over types) and the external choice operator: the value received from the channel is chosen externally, it cannot be negotiated by the receiver. Rule (E7) is similar to rule (E6), except that it deals with sieves. Rule (E8) states that \perp is the neutral element of the external choice.

Rules (I1–I9) state the fundamental properties of the internal choice operator. Rules (I1–I4) are similar to rules (E1–E4). Rule (I5) is the distributivity law of the prefix operator over the

internal choice (the same law does not hold for the external choice operator). Rule (I6) shows the interaction between input actions over types and the internal choice operator. A dual of the descriptor on the l.h.s. of $=$ does not know whether the descriptor is ready to receive a value of type t or of type s . Thus, the only possibility is to send a value that has both types. As a consequence, if t and s are disjoint types, namely if $t \wedge s = \emptyset$, then both descriptors are \perp (see rule (B1) below). Rule (I7) is similar to rule (I6), except that it deals with sieves. A dual of the descriptor on the l.h.s. of $=$ does not know whether the descriptor is ready to receive a descriptor θ such that θ is dual of χ or such that θ is dual of χ' . Thus, the only possibility is to send a descriptor that is dual of both θ and θ' . Rule (I8) states that \perp is the absorbing element of the internal choice.

Rules (B1–B5) characterize non-viable descriptors, namely those descriptors that have no dual. Rules (B1–B2) deal with communications of values from empty types and delegations of sessions with non-viable descriptors. Since these descriptors are completely inert (they do not emit any visible action), they are comparable to the canonical non-viable descriptor \perp . Rule (B3) states the disjunction between values and descriptors: no value is a descriptor, and no descriptor is a value. Rule (B4) states the directionality of our communication model. In order to be viable, a descriptor cannot simultaneously allow both input and output actions. The only exception to this rule is when the output actions are offered in an external choice, see rules (O1–O3) below. Rule (B5) is similar to rule (B4), except that it deals with end and input actions.

Rules (O1–O5) characterize the peculiar properties of output actions. In every rule the side condition ensures that the output action is not inert (see rule (B2) above). Rules (O1–O2) state that an output action composed in external choice with an end or an input action preempts the alternative action. Rule (O3) states that external and internal choices of output actions are indistinguishable, since these actions are irrevocable. Rule (O4) shows the interaction between output actions over types and the internal choice operator: the value sent over the channel is decided internally by the sender, it will not be negotiated with the receiver. Rule (O5) is similar to rule (O4), except that it deals with sieves.

Rules (S1–S2) show the standard covariant property of inputs: the duals of a session that is capable of receiving values of type t will also be duals of a session that is capable of receiving more values. Rule (S2) is similar to rule (S1) except that it deals with input of descriptors and it can be explained in the same way using the intuition that sieves stand for the set of their duals. Rules (S3–S4) complement rules (S1–S2) with dual properties for output actions, where we have contravariance. Note that in both cases we need one extra hypothesis, namely that $t \neq \emptyset$ and $x \neq \perp$. This guarantees that the larger descriptor will actually output some value/descriptor whenever the smaller one does so.

Finally, rule (S5) states that the duals of some session are also duals of a more deterministic session. In the testing theories for processes this law characterizes the deadlock sensitive *must* preorder.

We conclude this section discussing the interaction of \leq with the operators of session descriptors. It is easy to see that \leq is preserved by the prefix and the internal choice operators. In the latter case, this follows from the fact that \oplus coincides with the intersection operator in the set-theoretic interpretation of session descriptors. However, as we have already seen while discussing rule (E5), $+$ does not correspond to a Boolean operation and this ultimately makes $+$ quite subtle, as \leq is not respected by $+$ in general. For example, by rule (S1) we have $? \text{Int.end} \leq ? \text{Real.end}$ however $? \text{Int.end} + ? \sqrt{2}.!3.\text{end} \not\leq ? \text{Real.end} + ? \sqrt{2}.!3.\text{end}$. The reason is that in widening $? \text{Int.end}$ to $? \text{Real.end}$ we create an interference with the term $? \sqrt{2}.!3.\text{end}$ because of the guaranteed action $? \sqrt{2}$. Such interferences are not avoided even when we

operate with $=$ (as opposed to \leq). For instance, according to rule (16) we have $?(Int \vee \sqrt{2}).end \oplus ?Int.!3.end = ?Int.(end \oplus !3.end)$, but $?(Int \vee \sqrt{2}).end \oplus ?Int.!3.end + ?\sqrt{2}!.4.end \neq ?Int.(end \oplus !3.end) + ?\sqrt{2}!.4.end$. Here the action $?\sqrt{2}$ is not guaranteed by $?(Int \vee \sqrt{2}).end \oplus ?Int.!3.end$ and (16) tells us that in practice the capability of $?(Int \vee \sqrt{2}).end$ of receiving $\sqrt{2}$ is useless. However, removing this capability may also remove interferences in the context of an external choice, making (16) unsafe in general.

Finally, rule (B4) must be used with care within an external choice because its output capability makes the descriptor on the l.h.s. of $=$ to be observable (it may autonomously emit an action), whereas \perp is totally inert. For instance we have $?Int.end \oplus !Int.!0.end = \perp$ and $\perp + ?Bool.!3.end = ?Bool.!3.end$, but $(?Int.end \oplus !Int.!0.end) + ?Bool.!3.end = (?Int.end + ?Bool.!3.end) \oplus (!Int.!0.end + ?Bool.!3.end) = \perp$. Rule (B5) suffers from a similar problem, which is slightly less severe because end denotes a terminated descriptor.

A.2 Proofs

We will first prove Theorem 2.6 and Proposition 2.7. Next we prove Theorems 2.15 and 2.16 by simultaneous induction and then Proposition 2.10 and Theorem 2.12 since the latest two proofs use the strong normal forms of sieves. Notice that this does not introduce circularity since the first proof is independent from Proposition 2.10 and Theorem 2.12.

We start by defining the weight of types and sieves.

DEFINITION A.1 (Weight). *Given a term ψ we denote by $w(\psi)$ its weight defined as follows: a type has weight 0 if no session type occurs in it; a sieve has weight 0 if no prefix occurs in it (i.e., it is a boolean combination of possibly empty sums of end); a type has weight $i + 1$ if the session types occurring in it are on descriptors of weight at most i ; a sieve is of weight $i + 1$ if the prefixes occurring in it are of weight at most i .*

Observe that the definition above is well founded. First notice that there are only two places where the weight is increased: when a type t has a subterm of the form $begin.\eta$ and when a descriptor (more generally, a sieve) η has a subterm of the form $\alpha.\eta'$. Second notice that in both cases regularity ensures that there finitely many distinct such subterms, therefore taking the max of their weights is meaningful. Finally, and most importantly, notice that thanks to condition 3 of Definition 2.1 it is also meaningful to speak of the weight of these subterms, that is, the definition above is well founded. More precisely, the definition of the weight of η is given in terms of the weight of (the sieves forming) any prefix α occurring in η and by condition 3 we know that η cannot occur in α . Similarly, the weight of a type t is given in terms of any descriptor η occurring in it, and if a type occurs in a descriptor, it occurs in a prefix position; thus once more condition 3 ensures that t cannot occur in these η 's.

A.2.1 Proof of Theorem 2.6

Starting from Definition A.1 we define a weight for each relation we introduced in Section 2.2: each $\eta \xrightarrow{\mu} \eta'$ and $\eta \longrightarrow \eta'$ has weight 0 if it uses only axioms (i.e., (TR1) and (TR2)) and has weight $i + 1$ if it is proved by using relations of weight at most i ; $\eta \langle \mu \rangle$ has weight i if it is defined by reductions of weight at most i (we consider the successor as a binary relation); may/must actions/convergences relations and the duality relation all have weight i if they are proved by using relations of weight at most i . Finally, let us first define $v \in t$ to be of weight 0 if t is of weight 0; then notice that both $\eta \in \chi$ and $v \in t$ for t of weight at least

1 (cf. equation (7)) are defined in terms of duality: we then assign to each of them the greatest weight of the duality relations used in their definition.

Using this weight it is easy to check that the definitions of Section 2.2 are well founded.

A.2.2 Proof of Proposition 2.7

Suppose that (\Rightarrow) does not hold (the converse is trivial). Then there exists a descriptor η_1 dual of θ , such that $\eta_1 \in \chi_1$ and $\eta_1 \notin \chi_2$, and a descriptor η_2 dual of θ , such that $\eta_2 \notin \chi_1$ and $\eta_2 \in \chi_2$. Now consider $\eta_1 \oplus \eta_2$: since the semantics of an internal choice is the intersection of the duals of the choices, and θ is dual of both η_1 and η_2 , then θ is dual of $\eta_1 \oplus \eta_2$. But for the same reason we deduce that $\eta_1 \oplus \eta_2 \notin \chi_1$ and $\eta_1 \oplus \eta_2 \notin \chi_2$, and thus $\eta_1 \oplus \eta_2 \notin \chi_1 \vee \chi_2$ by definition, yielding a contradiction.

A.2.3 Proof of Theorems 2.15 and 2.16

The soundness and completeness of the algorithm need two preliminary results. The first one states that no finite union of session descriptors covers the whole \mathcal{S} . Namely, it is always possible to find another η having at least one dual descriptor that is not dual of any of the descriptors in the finite union.

LEMMA A.2. *For every viable sieve of the form $\bigvee_{i \in I} \eta_i$, there exists a descriptor η such that $\bigvee_{i \in I} \eta_i \vee \eta \not\leq \bigvee_{i \in I} \eta_i$.*

PROOF. By induction on the cardinality of I . We just consider the case for $|I| = 1$, that is $\bigvee_{i \in I} \eta_i \equiv \eta'$, the result follows by straightforward induction. Consider the set $\{\theta \mid \eta' \Longrightarrow \theta \dashv\rightarrow\}$. Now choose any value v such that for all θ in this set $\theta \xrightarrow{!v}$ implies that there exists $v' \neq v$ such that $\theta \xrightarrow{!v'}$. Note that such a v always exists because no descriptor can emit infinitely many singleton types (indeed if from a stable form a session type can emit just one output value, then this means that the output is on the singleton containing that value). Then setting $\eta \equiv !v.end$ proves the result. \square

The second auxiliary result simply states the correctness of rule (PREFIX) generalized to a finite number of prefixes.

LEMMA A.3. *For all $\alpha_1, \dots, \alpha_n, \eta, \eta'$, if $\eta \leq \eta'$, then $\alpha_1 \cdot \dots \cdot \alpha_n.\eta \leq \alpha_1 \cdot \dots \cdot \alpha_n.\eta'$.*

PROOF. By examination of the coinductive characterization of \leq (Definition 2.11) it is easy to check that the result holds for $n = 1$. The whole result follows from a straightforward induction on n . \square

We prove Theorems 2.15 and 2.16 by simultaneous induction. We do the proof by a double induction on the Noetherian measure given after Definition 2.1 and on the weight defined in Definition A.1, lexicographically ordered. Then we prove:

1. Each sieve χ can be effectively transformed into an equivalent strong normal form whose weight is smaller than or equal to the weight of χ .
2. $\chi \not\leq \chi'$ is decidable.

First of all notice that by classical set theoretic transformations it is possible to put every sieve in disjunctive normal form. This can be effectively done by the regularity of the trees. So let us assume that all the sieves we use in this proof are in disjunctive normal form.

Base case. The base case for measure and weight 0 is when both χ and χ' are possibly empty sums of end 's. The normal form of all such sieves is end (which can be obtained by rules (E1) and (I1) of Table 5) and $\chi \not\leq \chi'$ is easily decidable.

$$\begin{array}{c}
\text{[R-EX-SPLIT]} \\
\frac{\sum_{i \in I} ?\psi_i.\eta_i \{ + \text{end} \} \leq \sum_{j \in J \setminus \{k\}} ?\psi'_j.\eta'_j + ?(\psi'_k \setminus \psi_h).\eta'_k + ?\psi_h.\eta'_k \{ + \text{end} \}}{\sum_{i \in I} ?\psi_i.\eta_i \{ + \text{end} \} \leq \sum_{j \in J} ?\psi'_j.\eta'_j \{ + \text{end} \}} \left(\begin{array}{l} \psi_h \sqsubseteq \psi'_k \\ \psi_h \neq \psi'_k \end{array} \right) \\
\\
\text{[L-EX-SPLIT]} \\
\frac{\sum_{i \in I \setminus \{h\}} ?\psi_i.\eta_i + ?(\psi_h \setminus \psi'_k).\eta_h + ?\psi'_k.\eta_h \{ + \text{end} \} \leq \sum_{j \in J} ?\psi'_j.\eta'_j \{ + \text{end} \}}{\sum_{i \in I} ?\psi_i.\eta_i \{ + \text{end} \} \leq \sum_{j \in J} ?\psi'_j.\eta'_j \{ + \text{end} \}} \left(\begin{array}{l} \psi'_k \sqsubseteq \psi_h \\ \psi_h \neq \psi'_k \end{array} \right) \\
\\
\text{[LR-EX-SPLIT]} \\
\frac{\sum_{i \in I \setminus \{h\}} ?\psi_i.\eta_i + ?(\psi_h \setminus \psi'_k).\eta_h + ?\psi'_k.\eta_h \{ + \text{end} \} \leq \sum_{j \in J \setminus \{k\}} ?\psi'_j.\eta'_j + ?(\psi'_k \setminus \psi_h).\eta'_k + ?\psi_h.\eta'_k \{ + \text{end} \}}{\sum_{i \in I} ?\psi_i.\eta_i \{ + \text{end} \} \leq \sum_{j \in J} ?\psi'_j.\eta'_j \{ + \text{end} \}} \left(\begin{array}{l} \psi'_k \wedge \psi_h \neq \emptyset \\ \psi'_k \wedge \psi_h \neq \psi'_k \\ \psi'_k \wedge \psi_h \neq \psi_h \end{array} \right) \\
\\
\text{[R-IN-SPLIT]} \\
\frac{\bigoplus_{i \in I} !\psi_i.\eta_i \{ \oplus \text{end} \} \leq \bigoplus_{j \in J \setminus \{k\}} !\psi'_j.\eta'_j \oplus !(\psi'_k \setminus \psi_h).\eta'_k \oplus !\psi_h.\eta'_k \{ \oplus \text{end} \}}{\bigoplus_{i \in I} !\psi_i.\eta_i \{ \oplus \text{end} \} \leq \bigoplus_{j \in J} !\psi'_j.\eta'_j \{ \oplus \text{end} \}} \left(\begin{array}{l} \psi_h \sqsubseteq \psi'_k \\ \psi_h \neq \psi'_k \end{array} \right) \\
\\
\text{[L-IN-SPLIT]} \\
\frac{\bigoplus_{i \in I \setminus \{h\}} !\psi_i.\eta_i \oplus !(\psi_h \setminus \psi'_k).\eta_h \oplus !\psi'_k.\eta_h \{ \oplus \text{end} \} \leq \bigoplus_{j \in J} !\psi'_j.\eta'_j \{ \oplus \text{end} \}}{\bigoplus_{i \in I} !\psi_i.\eta_i \{ \oplus \text{end} \} \leq \bigoplus_{j \in J} !\psi'_j.\eta'_j \{ \oplus \text{end} \}} \left(\begin{array}{l} \psi'_k \sqsubseteq \psi_h \\ \psi_h \neq \psi'_k \end{array} \right) \\
\\
\text{[LR-IN-SPLIT]} \\
\frac{\bigoplus_{i \in I \setminus \{h\}} !\psi_i.\eta_i \oplus !(\psi_h \setminus \psi'_k).\eta_h \oplus !\psi'_k.\eta_h \{ \oplus \text{end} \} \leq \bigoplus_{j \in J \setminus \{k\}} !\psi'_j.\eta'_j \oplus !(\psi'_k \setminus \psi_h).\eta'_k \oplus !\psi_h.\eta'_k \{ \oplus \text{end} \}}{\bigoplus_{i \in I} !\psi_i.\eta_i \{ \oplus \text{end} \} \leq \bigoplus_{j \in J} !\psi'_j.\eta'_j \{ \oplus \text{end} \}} \left(\begin{array}{l} \psi'_k \wedge \psi_h \neq \emptyset \\ \psi'_k \wedge \psi_h \neq \psi'_k \\ \psi'_k \wedge \psi_h \neq \psi_h \end{array} \right)
\end{array}$$

Table 6. Algorithmic subsessioning simplification rules. In these rules we use \sqsubseteq to denote either \leq or $<$., curly braces to denote optional end summands, and suppose that all operators are uniformly applied either on types or on sieves.

Inductive case. Let us now study the inductive case, and therefore suppose the two properties hold for sieves of either strictly smaller measure or same measure and strictly smaller weight. For the sake of the presentation we prove the two points one after the other, although we should do the proof of the two properties—in the order in which we stated them—for each case.

Thus let us start proving the **point 1** by performing a case analysis on the form of χ .

If $\chi \equiv \bigvee_{i \in I} \bigwedge_{j \in J} \lambda_{ij}$ where $|I| > 1$, then we have to discard all the intersections that are bottom, that is all $\bigwedge_{j \in J} \lambda_{ij} = \perp$. Whether each of these intersections is equivalent to bottom can be effectively decided by induction hypothesis thanks to the point 2 of the theorem. The normal form is then obtained by coinductively applying the transformation on all remaining literals, which is possible thanks to the induction hypothesis.

Let $\chi \equiv \bigwedge_{j \in J} \lambda_j$ where at least one literal is not negated. The normal form is either \perp , or it is obtained by a coinductive application of the transformation. The latter is always possible thanks to the induction hypothesis. Thus all it remains to prove is that we can decide whether χ is \perp . Since we cannot directly use

the induction hypothesis (as we should apply it to the whole sieve), let us separate negated literals from positive ones. That is, define $J = P \cup N$, such that $\chi \equiv \bigwedge_{p \in P} \eta_p \wedge \bigwedge_{n \in N} \neg \eta_n$. Then $\chi \not\leq \perp$ if and only if $\bigoplus_{p \in P} \eta_p \not\leq \bigvee_{n \in N} \eta_n$, if and only if—by the strong disjunction property for descriptors— $\bigoplus_{p \in P} \eta_p \not\leq \eta_n$ holds for all $n \in N$. This can be decided by induction hypothesis using the point 2 of the theorem.

If $\chi \equiv \bigwedge_{j \in J} \neg \eta_j$. As above, let us first show that it is possible to decide that $\chi \not\leq \perp$, that is whether $\neg \bigvee_{j \in J} \eta_j \not\leq \perp$. This is always true for Lemma A.2, therefore we can coinductively apply the transformation by induction hypothesis.

All the coinductive transformations above terminate by the regularity of our sieves. Furthermore it is easy to see that they do not increase the weight of the sieves.

If $\chi \equiv \eta$, and the descriptor is prefixed, then we check that its prefix is not \emptyset (which can be done by induction hypothesis) and possibly coinductively apply the transformation on its continuation.

Otherwise we will do the following transformations:

1. get rid of every subterm of the form $?\psi.\eta$ and $!\psi.\eta$ such that $\psi = \emptyset$ by means of rules (B1), (B2), (E8), and (I8);

2. get internal choices of external choices of prefixed descriptors and end using (E4);
3. get internal choices of:
 - external choices of input descriptors and possibly of end
 - output descriptors
 - possibly end

by applying the rules (O1), (O2), and (O3) inside the external choices. That is, we obtain a descriptor of the shape:

$$\bigoplus_{J \in K} \left(\sum_{j \in J} ?\psi_j \cdot \eta_j \{ + \text{end} \} \right) \oplus \bigoplus_{h \in H} !\psi_h \cdot \eta_h \{ \oplus \text{end} \} \quad (12)$$

4. we have the following cases:

- (a) $K = \emptyset$.
- (b) $H = \emptyset$. Then (12) is equivalent to:

$$\sum_{j_1 \in J_1} \cdots \sum_{j_k \in J_k} ?(\psi_{j_1} \wedge \cdots \wedge \psi_{j_k}) \cdot (\eta_{j_1} \oplus \cdots \oplus \eta_{j_k}) \{ + \text{end} \} \quad (13)$$

where $K = \{J_1, \dots, J_k\}$ and $\psi_{j_1} \wedge \cdots \wedge \psi_{j_k} \neq \emptyset$ and end is present only if it occurs in all the external choices of (12).

- (c) $K, H \neq \emptyset$ and end is present in all external choices. Then (12) is equivalent to

$$\eta = \bigoplus_{h \in H} !\psi_h \cdot \eta_h \oplus \text{end} \quad (14)$$

- (d) $K, H \neq \emptyset$, at least one external choice has no end subterm. Then (12) is equivalent to \perp .

5. in cases (4a) and (4c) we can obtain an internal choice of outputs and possibly end such that if $!\psi$ and $!\psi'$ are two top level prefixes we have $\psi \wedge \psi' = \emptyset$ by applying rules (O4), (O5), and (15).
6. in case (4b) we obtain an external choice of inputs and possibly end such that if $?\psi$ and $?\psi'$ are two top level prefixes we have $\psi \wedge \psi' = \emptyset$ by applying rules (E5), (E6), and (E7).
7. we convert any χ occurring in a top level prefix in strong normal form (this can be effectively done by the induction hypothesis).
8. we coinductively apply the algorithm to every continuation of every top level guarded descriptor of the (internal or external) choice.

Note that the coinductive application of the algorithm terminates by the regularity of our descriptors. All it remains to prove is that the passage from step (3.) to step (4.) is sound, that is it yields an equivalent descriptor.

For case (4a) this is trivial.

For case (4b) we have that equation (12) becomes an internal choice of external choices of inputs. Let us examine the set of duals of (12). Since we only have inputs then we have to check which inputs are guaranteed. These are exactly all the inputs that are guaranteed by *all* the summands of the internal choice, that is those that are emitted by all prefixes, and thus by their intersection. The continuation of this intersection is then obtained by the definition of successor yielding the descriptor (13).

In case (4c) by the irrevocability of outputs we have to check which outputs are offered in order to characterise the set of duals of (12). It is easy to see that (12) and (14) offer the same outputs and they both offer end.

In case (4d) (12) is not viable since it does not converge and therefore it is equivalent to \perp .

Let us pass to the proof of **point 2**, that is show that it is possible to decide whether $\chi \preceq \chi'$ holds. Thanks to point 1 we can suppose that both χ and χ' are in strong normal form.

We proceed by case analysis on the form of χ and χ' by starting with the simplest cases first.

Case $\bigvee_{i \in I} \bigwedge_{j \in J} \lambda_{ij} \preceq \bigvee_{h \in H} \bigwedge_{k \in K} \lambda'_{hk}$ such that $|I| > 1$. We can split the union on the left, and reduce this problem to check whether there exists $i \in I$ such that $\bigwedge_{j \in J} \lambda_{ij} \preceq \bigvee_{h \in H} \bigwedge_{k \in K} \lambda'_{hk}$, which can be checked by induction hypothesis.

Case $\bigwedge_{j \in J} \lambda_j \preceq \bigvee_{h \in H} \bigwedge_{k \in K} \lambda'_{hk}$ such that $|H| > 1$. By applying classical set-theoretic distribution laws, this problem can be reduced to $\bigwedge_{j \in J} \lambda_j \preceq \bigwedge_{h \in H} \bigvee_{k \in K} \lambda'_{hk}$. We can now split the intersection on the right and reduce it to check whether there exists $h \in H$ such that $\bigwedge_{j \in J} \lambda_j \preceq \bigvee_{k \in K} \lambda'_{hk}$. The result follows by induction hypothesis.

Case $\bigwedge_{j \in J} \lambda_j \preceq \bigvee_{k \in K} \lambda'_k$. Let us highlight negative and positive literals:

$\bigwedge_{p \in P_J} \theta_p \wedge \bigwedge_{n \in N_J} \neg \eta_n \preceq \bigwedge_{p \in P_K} \theta_p \wedge \bigwedge_{n \in N_K} \neg \eta_n$.
By simple set-theoretic manipulations this is equivalent to check whether $\bigwedge_{p \in P_J \cup P_K} \theta_p \preceq \bigvee_{n \in N_J \cup N_K} \eta_n$. Since intersection is equivalent to internal choice this is reduced to checking whether $\bigoplus_{p \in P_J \cup P_K} \theta_p \preceq \bigvee_{n \in N_J \cup N_K} \eta_n$ which by strong disjunction is equivalent to prove that $\bigoplus_{p \in P_J \cup P_K} \theta_p \preceq \eta_n$ holds for all $n \in N_J \cup N_K$. The result follows by induction hypothesis.

Case $\eta \preceq \eta'$. This is the last remaining case and also the most difficult one. We can feed η and η' to the algorithmic rules listed after Theorem 2.15 and in Table 6. So to prove this case we have to prove that these rules are sound and complete with respect to the semantic definition of subtyping.

Soundness. For each rule listed after Theorem 2.15 and in Table 6 we must prove that the set of duals of the lhs of its conclusion is included in the set of duals of the rhs. Let us proceed by a case analysis on the last rule applied in the deduction of $\eta \leq \eta'$.

Case [END] and [MIX-CHOICES]. The result follows by a direct application of the definition of duality.

Case [EXT-CHOICES]. By induction hypothesis for all $i \in I$ the set of duals of η_i is contained in the set of duals of η'_i . We have two subcases. (i) Case $|I| > 1$: since both descriptors are in strong normal form, then they can only emit input signals or a tick; thus for these choices we can apply only the rules TR3 and TR4; this implies that the signals emitted by each descriptor in the conclusion are exactly the same as those of their subcomponents. The result follows by the definition of duality. (ii) Case $|I| = 1$: then there are three subcases. Either the lhs of the conclusion emits an input, and then we proceed as in the case before; or it is an end, but then one of the summands of the rhs is also end, and the result follows from the definition of duality as both must ensure \checkmark ; or it is of the form $!\psi \cdot \eta''$, but then also $|J| = 1$ and therefore this rule does not apply ([PREFIX] should be used instead).

Case [INT-CHOICES]. Similar to the previous case.

Case [*-SPLIT] Notice that by rules I5-I7 and E5-E7 the corresponding session descriptors at the premise and at the conclusion of each rule have exactly the same set of duals, whence the result.

Case [PREFIX]. This is the hard case since we cannot use the induction hypothesis as the Noetherian measure and the weight may not decrease. We are in the case where we have deduced $\alpha \cdot \theta \leq \alpha \cdot \theta'$ from $\theta \leq \theta'$. So let us consider the deduction for $\theta \leq \theta'$ and explore it upwards from the root. By the regularity of our descriptors we have just two possible cases: either we traverse a *finite* (and possibly null) number of applications of the PREFIX

rule and arrive to the application of a different rule, or we traverse an again *finite* (and possibly null) number of applications of the PREFIX rule and arrive to the judgment $\theta \leq \theta'$. The latter case is straightforward because it means that $\theta \equiv \theta'$, therefore the result holds for reflexivity. In the former case instead we perform a case analysis on the rule we have reached, apply the same reasoning as above for the corresponding case and deduce the result by Lemma A.3.

Completeness. Suppose that the result holds for descriptors of either smaller measure or same measure and smaller weight and let us prove for the general case. Imagine by contradiction that the result does not hold for the general case. Then there exist η and η' such that all the duals of η are also duals of η' but for which the algorithm answers no. Therefore there exists at least one rule that fails. Since we did not put any constraint on η and η' , then we can consider without loss of generality that it is the last one and that all the preceding applications of the rules hold. Let us then perform a case analysis on this rule:

Case [END]. This is the base case and vacuously holds since it cannot fail.

Case [PREFIX]. This is another base case and vacuously holds since it cannot fail (if it fails it is because the premise failed, but this contradicts the fact that the PREFIX rule is the last one to have failed).

Case [MIX-CHOICES]. The last base case. It may have failed because one (or both) of the two end's is absent, but this contradicts our hypothesis: if it is the rhs end that is missing then end is a dual of the first but not of the second; if it is the lhs end that is missing then I is not empty (otherwise the rule would not fail), but then, being the types in strong normal form, a dual of η_i cannot be dual of the lhs, since both of them ensure an input. It may have also failed because the internal choice is on the right and the external one is on the left (strictly speaking this is not a failure of this rule but it is the only case in which no rule applies), but then it is easy to build a dual for the lhs which is not dual for the rhs.

Case [EXT-CHOICES]. If this failed it is because there exists $i \in I$ such that for all $j \in J$, $\eta_i \not\leq \eta_j$. By induction hypothesis since the algorithm is complete, then there exists θ that is dual of η_i but it is not dual of any η_j . By definition θ is dual of $\sum_{i \in I} \eta_i$ but not of $\sum_{j \in J} \eta_j$, contradiction.

Case [INT-CHOICE]. Similar to the previous case.

Case [*-SPLIT]. These rules never fail since they can always be applied.

A.2.4 Proof of Proposition 2.10

(\Rightarrow) By Theorem 2.15 we may assume that η is in strong normal form. We define a function $\bar{\cdot}$ such that $\bar{\eta} \bowtie \eta$. Regularity of $\bar{\eta}$ is a direct consequence of the regularity of η .

Assume $\eta \equiv \sum_{i \in I} ?\psi_i.\eta_i\{+\text{end}\}$, where the end subterm may be missing. Then η^\bowtie must be justified by condition (1) of Definition 2.9, namely there exists μ such that $\eta \Downarrow \mu$ and $\eta\langle\mu\rangle^\bowtie$. If $\mu = \checkmark$, then we conclude immediately by taking $\bar{\eta} = \text{end}$. If $\mu \neq \checkmark$, then there exists $k \in I$ such that $?\psi_k.\eta_k \Downarrow \mu$ and $\eta\langle\mu\rangle^\bowtie$. Because η is in disjoint normal form we have $?\psi_i.\eta_i \Downarrow \mu'$ implies $\eta\langle\mu'\rangle = \eta_i$ for every μ' . Hence we conclude by taking $\bar{\eta} = !\psi_k.\bar{\eta}_k$.

Assume $\eta \equiv \bigoplus_{i \in I} !\psi_i.\eta_i\{\oplus\text{end}\}$, where the end subterm may be missing. Then η^\bowtie must be justified by condition (2) of Definition 2.9, namely for every $i \in I$ we have $!\psi_i.\eta_i \Downarrow \mu$ for some μ and η_i^\bowtie since $\eta\langle\mu\rangle = \eta_i$. We conclude by taking $\bar{\eta} = \sum_{i \in I} ?\psi_i.\bar{\eta}_i + \text{end}$.

The proof that $\bar{\eta} \bowtie \eta$ is trivial.

(\Leftarrow) It is sufficient to show that the relation

$$\mathcal{R} = \{\eta \mid \exists \eta' : \eta \bowtie \eta'\}$$

is a coinductive viability. Let $\eta \in \mathcal{R}$. Then there exists η' such that $\eta \bowtie \eta'$. We reason by cases on the justification of $\eta \bowtie \eta'$ according to Definition 2.5.

Assume $\eta \bowtie \eta'$ is justified by condition (1) of Definition 2.5. Then $\eta \Downarrow \checkmark$ and $\eta' \Downarrow \checkmark$. Hence condition (2) of Definition 2.9 is satisfied (note that $\text{end} \in \mathcal{R}$ by definition of \mathcal{R}).

Assume $\eta \bowtie \eta'$ is justified by condition (2) of Definition 2.5. Then $\eta \Downarrow$ and $\eta \Downarrow \mu$ implies $\eta' \Downarrow \mu$ and $\eta\langle\mu\rangle \bowtie \eta'\langle\mu\rangle$. By definition of \mathcal{R} we have $\eta\langle\mu\rangle \in \mathcal{R}$, hence condition (1) of Definition 2.9 is satisfied.

A.2.5 Proof of Theorem 2.12

(\Rightarrow) Assume $\eta_1 \leq \eta_2$ and η_1^\bowtie . By Proposition 2.10 we have that η_1 is viable. Let $\eta \bowtie \eta_1$. It is sufficient to show that

$$\mathcal{C} = \{(\eta', \eta_2) \mid \exists \eta_1' : \eta' \bowtie \eta_1' \wedge \eta_1' \leq \eta_2'\}$$

is a duality relation, since $(\eta, \eta_2) \in \mathcal{C}$ by definition of \mathcal{C} . Let $(\eta', \eta_2') \in \mathcal{C}$. Then there exists η_1' such that $\eta' \bowtie \eta_1'$ and $\eta_1' \leq \eta_2'$. We reason by cases on the justification of $\eta' \bowtie \eta_1'$ for showing that η' and η_2' satisfy at least one of the conditions of Definition 2.5. Assume that $\eta' \bowtie \eta_1'$ is justified by condition (1) of Definition 2.5. Then $\eta' \Downarrow \checkmark$ and $\eta_1' \Downarrow \checkmark$. From $\eta_1' \leq \eta_2'$ we derive $\eta_2' \Downarrow \checkmark$ hence we conclude by condition (1) of Definition 2.5. Assume that $\eta' \bowtie \eta_1'$ is justified by condition (2) of Definition 2.5. Then $\eta' \Downarrow$ and $\eta' \Downarrow \mu$ implies $\eta_1' \Downarrow \mu$ and $\eta'\langle\mu\rangle \bowtie \eta_1'\langle\mu\rangle$. From $\eta_1' \leq \eta_2'$ we derive $\eta_2' \Downarrow \mu$ and $\eta_1'\langle\mu\rangle \leq \eta_2'\langle\mu\rangle$ hence $(\eta'\langle\mu\rangle, \eta_2'\langle\mu\rangle) \in \mathcal{C}$ by definition of \mathcal{C} and we conclude by condition (2) of Definition 2.5.

(\Leftarrow) It is sufficient to show that the relation

$$\mathcal{R} = \{(\eta, \eta') \mid \forall \theta : \theta \bowtie \eta \Rightarrow \theta \bowtie \eta'\}$$

is a coinductive subsession. Let $(\eta, \eta') \in \mathcal{R}$ and assume η^\bowtie for otherwise there is nothing to prove. By Theorem 2.15 we may assume that both η and η' are in strong normal form. By Proposition 2.10 there exists θ such that $\theta \bowtie \eta$. By definition of \mathcal{R} we deduce that $\theta \bowtie \eta'$, hence η'^\bowtie again by Proposition 2.10. We reason by cases on the structure of η and η' .

Assume $\eta \equiv \sum_{i \in I} ?\psi_i.\eta_i\{+\text{end}\}$ and $\eta' \equiv \sum_{j \in J} ?\psi_j.\eta_j\{+\text{end}\}$. Condition (1) of Definition 2.11 is trivially satisfied since $\eta' \Downarrow$. As regards condition (2) of Definition 2.11, assume $\eta \Downarrow \mu$ and $\eta\langle\mu\rangle^\bowtie$. If $\mu = \checkmark$, then $\eta' \Downarrow \checkmark$ for otherwise $\text{end} \bowtie \eta'$ and $\text{end} \not\bowtie \eta'$ which is absurd by definition of \mathcal{R} . If $\mu \neq \checkmark$, then there exists $i \in I$ such that $?\psi_i.\eta_i \Downarrow \mu$ and η_i^\bowtie . Suppose by contradiction that $\eta' \not\Downarrow \mu$ and consider $\theta \equiv !\psi_i.\theta_i$, where θ_i is an arbitrary dual of η_i (it exists from the hypothesis η_i^\bowtie and by Proposition 2.10). Then $\theta \bowtie \eta$ but $\theta \not\bowtie \eta'$ which is absurd. Hence there exists $j \in J$ such that $?\psi_j.\eta_j \Downarrow \mu$. By definition of duality we deduce $\theta_i \bowtie \eta_i$ and $\theta_i \bowtie \eta_j$, hence $(\eta_i, \eta_j) \in \mathcal{R}$ since θ_i is arbitrary. We conclude by observing that $\eta\langle\mu\rangle = \eta_i$ and $\eta'\langle\mu\rangle = \eta_j$. Condition (3) of Definition 2.11 is trivially satisfied since $\eta \Downarrow$.

Assume $\eta \equiv \bigoplus_{i \in I} !\psi_i.\eta_i\{\oplus\text{end}\}$ and $\eta' \equiv \bigoplus_{j \in J} !\psi_j.\eta_j\{\oplus\text{end}\}$ and $\eta' \Downarrow$. As regards condition (1) of Definition 2.11, for every $i \in I$ let θ_i be an arbitrary session descriptor such that $\theta_i \bowtie \eta_i$ (these descriptors exist because η_i^\bowtie). Let $\theta \equiv \sum_{i \in I} ?\psi_i.\theta_i\{+\text{end}\}$ where the end subterm is present only if it is present also in η . Assume $\eta' \Downarrow \mu$. Then there exists $j \in J$ such that $!\psi_j.\eta_j \Downarrow \mu$. Suppose contradiction that $\eta \not\Downarrow \mu$. Then $\theta \bowtie \eta$ but $\theta \not\bowtie \eta'$, which is absurd. Hence there exists $i \in I$ such that $!\psi_i.\eta_i \Downarrow \mu$. We derive $(\eta_i, \eta_j) \in \mathcal{R}$ since θ_i is arbitrary. We conclude by observing that $\eta\langle\mu\rangle = \eta_i$ and $\eta'\langle\mu\rangle = \eta_j$. Condition (3) of Definition 2.11 is trivially satisfied since $\eta' \Downarrow$.

Assume $\eta \equiv \bigoplus_{i \in I} !\psi_i.\eta_i\{\oplus\text{end}\}$ and $\eta' \equiv \sum_{j \in J} ?\psi_j.\eta_j\{+\text{end}\}$ and $\eta \Downarrow$. Condition (1) and (2) of Definition 2.11 are trivially satisfied since $\eta \Downarrow$ and $\eta' \Downarrow$. As regards condition (3) of Definition 2.11, for every $i \in I$ let θ_i be an arbitrary session descriptor such that $\theta_i \bowtie \eta_i$ (these descriptors exist because η_i^\bowtie). Let

$\theta \equiv \sum_{i \in I} ?\psi_i.\theta_i\{\text{end}\}$ where the end subterm is present only if it is present also in η . Assume by contradiction $\eta \not\Downarrow \checkmark$ or $\eta' \not\Downarrow \checkmark$. Then $\theta \bowtie \eta$ but $\theta \not\bowtie \eta'$, which is absurd. We conclude $\eta \Downarrow \checkmark$ and $\eta' \Downarrow \checkmark$.

B. Proofs of Section 3

B.1 Proof of Subject Reduction

LEMMA B.1 (Strengthening). *If $\Gamma \vdash P : (h : \text{end} \cdot \Delta)$, then $\Gamma \vdash P : \Delta$.*

The *core* of a session stack ($\text{core}(\Delta)$) is the stack obtained by removing all ended channels. This is sound by Lemma B.1. The core of a session environment is defined similarly.

DEFINITION B.2 (Core).

$$\text{core}(\Delta) = \begin{cases} (h : \eta \cdot \text{core}(\Delta')) & \text{if } \Delta = (h : \eta \cdot \Delta') \text{ and } \eta \neq \text{end} \\ \text{core}(\Delta') & \text{if } \Delta = (h : \text{end} \cdot \Delta') \end{cases}$$

$$\text{core}(\Lambda) = \{h : \eta \mid h : \eta \in \Lambda \text{ and } \eta \neq \text{end}\}$$

The partial order relation \preceq between session stacks and session environments takes into account their evolution due to process and system reductions.

DEFINITION B.3. $\Delta \preceq \Delta'$ is the smallest partial order relation such that:

1. $\text{core}(\Delta') = (h : \eta \cdot \Delta'')$, and $\text{core}(\Delta) = \Delta''$ or
2. $\text{core}(\Delta') = (h : \eta' \cdot \Delta'')$, $\text{core}(\Delta) = (h : \eta \cdot \Delta'')$, and either $\eta \xrightarrow{\mu} \eta'$ or $\eta \longrightarrow \eta'$, or
3. $\text{core}(\Delta') = (h : \eta \cdot \Delta'')$, $\text{core}(\Delta) = (h : !\chi.\eta \cdot (h' : \eta' \cdot \Delta''))$ and $\eta' \leq \chi$, or
4. $\text{core}(\Delta') = (h : \eta \cdot x : \eta')$, $\text{core}(\Delta) = (h : ?\chi.\eta)$ and $\chi \leq \eta'$.
5. $\text{core}(\Delta') = (h : \eta_1 \cdot \Delta'')$ and $\text{core}(\Delta) = (h : \eta_1 + \eta_2 \cdot \Delta'')$.

DEFINITION B.4. $\Lambda \preceq \Lambda'$ is the smallest partial order relation such that:

1. $\text{core}(\Lambda') = \text{core}(\Lambda) \cup \{h : \eta\}$, or
2. $\text{core}(\Lambda') = \Lambda'' \cup \{h : \eta'\}$, $\text{core}(\Lambda) = \Lambda'' \cup \{h : \eta\}$, and either $\eta \xrightarrow{\mu} \eta'$ or $\eta \longrightarrow \eta'$, or
3. $\text{core}(\Lambda') = \Lambda'' \cup \{h : \eta\}$, $\text{core}(\Lambda) = \Lambda'' \cup \{h' : \eta', h : !\chi.\eta\}$ and $\eta' \leq \chi$, or
4. $\text{core}(\Lambda') = x : \eta' \cup \{h : \eta\}$, $\text{core}(\Lambda) = \{h : ?\chi.\eta\}$ and $\chi \leq \eta'$.

It is easy to verify that \preceq agrees with the mapping set and with union of session environments.

PROPOSITION B.5. 1. *If $\Delta \preceq \Delta'$, then $\text{set}(\Delta) \preceq \text{set}(\Delta')$.*

2. *If $\Lambda_1 \preceq \Lambda'_1$ and $\Lambda_2 \preceq \Lambda'_2$, then $\Lambda_1 \cup \Lambda_2 \preceq \Lambda'_1 \cup \Lambda'_2$.*

As usual generation and substitution lemmas are the key of our subject reduction proof.

LEMMA B.6 (Generation Lemma for Processes). 1. *If $\Gamma \vdash \mathbf{0} : \Delta$, then $\text{core}(\Delta) = -$.*

2. *If $\Gamma \vdash \text{a}!(x : \eta).P : \Delta$, then $\Gamma \vdash P : (x : \eta \cdot \Delta)$ and $\Gamma \vdash \text{a} : \text{begin}.\eta$.*
3. *If $\Gamma \vdash \text{c}^{\text{begin}.\eta'}?(x : \eta).P : \Delta$, then $\Gamma \vdash P : (x : \eta \cdot \Delta)$ and $\eta \bowtie \eta'$.*
4. *If $\Gamma \vdash \text{h}?(x : t).P : \Delta$, then $\text{core}(\Delta) = (h : ?t.\eta \cdot \Delta')$ and $\Gamma, x : t \vdash P : (h : \eta \cdot \Delta')$.*
5. *If $\Gamma \vdash \text{h}!e.P : \Delta$, then $\text{core}(\Delta) = (h : !t.\eta \cdot \Delta')$ and $\Gamma \vdash e : t$ and $\Gamma \vdash P : (h : \eta \cdot \Delta')$.*

6. *If $\Gamma \vdash \text{h}?(x : \chi).P : \Delta$, then $\text{core}(\Delta) = (h : ?\chi.\eta)$ and $\Gamma \vdash P : (h : \eta \cdot x : \eta')$ and $\chi \leq \eta'$.*
7. *If $\Gamma \vdash \text{h}!h'.P : \Delta$, then $\text{core}(\Delta) = (h : !\chi.\eta \cdot (h' : \eta' \cdot \Delta'))$ and $\Gamma \vdash P : (h : \eta \cdot \Delta')$ and $\eta' \leq \chi$.*
8. *If $\Gamma \vdash P + Q : \Delta$, then $\text{core}(\Delta) = (h : \eta_1 + \eta_2 \cdot \Delta')$ and $\Gamma \vdash P : (h : \eta_1 \cdot \Delta')$ and $\Gamma \vdash Q : (h : \eta_2 \cdot \Delta')$.*
9. *If $\Gamma \vdash P \oplus Q : \Delta$, then either $\text{core}(\Delta) = (h : \eta_1 \oplus \eta_2 \cdot \Delta')$ and $\Gamma \vdash P : (h : \eta_1 \cdot \Delta')$ and $\Gamma \vdash Q : \Delta', h : \eta_2$ or $\Gamma \vdash P : \Delta$, and $\Gamma \vdash Q : \Delta$.*

LEMMA B.7 (Generation Lemma for Systems). 1. *If $\Gamma \Vdash P : \Lambda$, then there exists Δ such that $\Lambda = \text{set}(\Delta)$ and $\Gamma \vdash P : \Delta$.*

2. *If $\Gamma \Vdash \mathbb{S} \parallel \mathbb{T} : \Lambda$, then there exist Λ_1 and Λ_2 such that $\Lambda = \Lambda_1 \cup \Lambda_2$ and $\Gamma \Vdash \mathbb{S} : \Lambda_1$ and $\Gamma \Vdash \mathbb{T} : \Lambda_2$.*

LEMMA B.8 (Substitution). 1. *If $\Gamma, x : t \Vdash \mathbb{S} : \Lambda$ and $\vdash v : t$, then $\Gamma \Vdash \mathbb{S}[v/x] : \Lambda$.*

2. *If $\Gamma \Vdash \mathbb{S} : \Lambda, x : \eta$, then $\Gamma \Vdash \mathbb{S}[k/x] : \Lambda, k : \eta$.*

The following lemma relates the one step reductions of processes with labels different from τ with the changes of the session stacks.

LEMMA B.9. *Let $\Gamma \vdash P : \Delta$, then*

1. *If $P \xrightarrow{\text{c}^!(x:\eta)} P'$, then $\Gamma \vdash P' : (x : \eta \cdot \Delta)$.*
2. *If $P \xrightarrow{\text{c}^?(x:\eta)} P'$, then $\Gamma \vdash P' : (x : \eta \cdot \Delta)$ and $t = \text{begin}.\eta'$ and $\eta \bowtie \eta'$.*
3. *If $P \xrightarrow{\text{k}!v} P'$, then $\Gamma \vdash P' : (k : \eta \cdot \Delta')$ and $\text{core}(\Delta) = (k : !t.\eta \{+\theta\} \cdot \Delta')$.*
4. *If $P \xrightarrow{\text{k}?(x:t)} P'$, then $\Gamma, x : t \vdash P' : (k : \eta \cdot \Delta')$ and $\text{core}(\Delta) = ((k : ?t.\eta \{+\theta\}) \cdot \Delta')$.*
5. *If $P \xrightarrow{\text{k}!k'} P'$, then $\Gamma \vdash P' : (k : \eta \cdot \Delta')$ and $\text{core}(\Delta) = (k : !\chi.\eta \{+\theta\} \cdot (k' : \eta' \cdot \Delta'))$ and $\eta' \leq \chi$.*
6. *If $P \xrightarrow{\text{k}?(x:\chi)} P'$, then $\Gamma \vdash P' : (k : \eta \cdot x : \eta')$ and $\text{core}(\Delta) = (k : ?\chi.\eta \{+\theta\})$ and $\chi \leq \eta'$.*

PROOF. By cases on $\xrightarrow{\ell}$ using Lemma B.6. \square

THEOREM 3.2 (Subject Reduction for Processes) *If $\Gamma \vdash P : \Delta$ and $P \xrightarrow{\ell} P'$, then $\Gamma' \vdash P' : \Delta'$, where $\Delta \preceq \Delta'$.*

PROOF. The proof is by induction and by cases on $\xrightarrow{\ell}$. For external and internal choices we only consider the first cases of Lemma B.6(8) and (9), since for the second cases the proof is similar and simpler.

Case R-CONNECT. Easy from Lemma B.9(case 1) and Definition B.3(case 1).

Case R-SEND. Easy from Lemma B.9(cases 3 and 5), and Definition B.3(cases 1 and 2).

Case R-RECEIVE. Easy from Lemma B.9(cases 2, 4 and 6) and Definition B.3(cases 1 and 2).

Case R-EXTCH. We have that

$$\frac{P \xrightarrow{\ell} P' \quad \ell \neq \tau}{P + Q \xrightarrow{\ell} P'} \quad \text{and} \quad \Gamma \vdash P + Q : \Delta.$$

From Lemma B.6(8), we have that $\text{core}(\Delta) = (k : \eta_1 + \eta_2 \cdot \Delta')$ and $\Gamma \vdash P : (k : \eta_1 \cdot \Delta')$, and $\Gamma \vdash Q : (k : \eta_2 \cdot \Delta')$.

By induction hypothesis on $P \xrightarrow{\ell} P'$ we get $\Gamma' \vdash P' : \Delta''$, where $(k : \eta_1 \cdot \Delta') \preceq \Delta''$. By Definition B.3(5), we have that $(k : \eta_1 + \eta_2 \cdot \Delta') \preceq (k : \eta_1 \cdot \Delta')$ and by transitivity $\Delta \preceq \Delta''$.

Case R-EXTINT. We have that

$$\frac{P \xrightarrow{\tau} P'}{P + Q \xrightarrow{\tau} P' + Q} \quad \text{and} \quad \Gamma \vdash P + Q : \Delta.$$

From Lemma B.6(8), we have that $\text{core}(\Delta) = (k : \eta_1 + \eta_2 \cdot \Delta')$ and $\Gamma \vdash P : (k : \eta_1 \cdot \Delta')$, and $\Gamma \vdash Q : (k : \eta_2 \cdot \Delta')$. By induction hypothesis on $P \xrightarrow{\tau} P'$ we get $\Gamma \vdash P' : \Delta''$ where $(k : \eta_1 \cdot \Delta') \preceq \Delta''$. But since P becomes P' by a τ action, then we know by rules R-EXTINT, R-EXTSEND and R-INTCH that P is either an internal or an external choice, then by Lemmas B.6(9) and B.6(8) we get $\Delta'' = (k : \eta'_1 \cdot \Delta')$, for some η'_1 such that $\eta_1 \rightarrow \eta'_1$. By typing rule T-EXTCH we get $\Gamma \vdash P' + Q : (k : \eta'_1 + \eta_2 \cdot \Delta')$ and by the descriptor transition rule (TR3) we get $\eta_1 + \eta_2 \rightarrow \eta'_1 + \eta_2$. We conclude since $\Delta \preceq (k : \eta'_1 + \eta_2 \cdot \Delta')$ by Definition B.3(2).

Case R-EXTSEND. We have that

$$\frac{P \xrightarrow{klv}}{P + Q \xrightarrow{\tau} P} \quad \text{and} \quad \Gamma \vdash P + Q : \Delta.$$

From Lemma B.6(8), we have that $\text{core}(\Delta) = (k : \eta_1 + \eta_2 \cdot \Delta')$ and $\Gamma \vdash P : (k : \eta_1 \cdot \Delta')$, and $\Gamma \vdash Q : (k : \eta_2 \cdot \Delta')$. By Definition B.3(2), we have that $\Delta \preceq (k : \eta_1 \cdot \Delta')$.

We have that

$$\frac{P \xrightarrow{klk_1}}{P + Q \xrightarrow{\tau} P} \quad \text{and} \quad \Gamma \vdash P + Q : \Delta.$$

From Lemma B.6(8), we have that $\text{core}(\Delta) = (k : \eta_1 + \eta_2 \cdot \Delta')$ and $\Gamma \vdash P : (k : \eta_1 \cdot \Delta')$, and $\Gamma \vdash Q : (k : \eta_2 \cdot \Delta')$. By Definition B.3(2), we have that $\Delta \preceq (k : \eta_1 \cdot \Delta')$.

Case R-INTCH. We have that

$$P \oplus Q \xrightarrow{\tau} P \quad \text{and} \quad \Gamma \vdash P \oplus Q : \Delta.$$

From Lemma B.6(9), we have that

- either $\text{core}(\Delta) = (k : \eta_1 \oplus \eta_2 \cdot \Delta')$ and $\Gamma \vdash P : (k : \eta_1 \cdot \Delta')$, and $\Gamma \vdash Q : (k : \eta_2 \cdot \Delta')$. By Definition B.3(2), we have that $\Delta \preceq (k : \eta_1 \cdot \Delta')$.
- or $\Gamma \vdash P : \Delta$ and $\Gamma \vdash Q : \Delta$ and the result follows immediately by reflexivity of \preceq .

□

We can lift the relations between reductions of processes and changes of session stacks shown in Lemma B.9 to relations between reductions of systems and changes of session environments. More precisely we can easily prove the following lemma:

LEMMA B.10. *Let $\ell \neq \tau$.*

1. *If $\Sigma \vdash \mathbb{S} \xrightarrow{\ell} \Sigma' \vdash \mathbb{S}'$, then there are \mathbb{T}, \mathbb{T}' and P such that $\mathbb{S} = \mathbb{T} \parallel P \parallel \mathbb{T}'$ and $\mathbb{S}' = \mathbb{T} \parallel P' \parallel \mathbb{T}'$ and $P \xrightarrow{\ell} P'$, where one or both \mathbb{T}, \mathbb{T}' can be missing.*
2. *If $\Sigma \vdash \mathbb{S} \parallel P \parallel \mathbb{T} \xrightarrow{\ell} \Sigma' \vdash \mathbb{S}' \parallel P' \parallel \mathbb{T}$ and $\Gamma \Vdash \mathbb{S} \parallel P \parallel \mathbb{T} : \Lambda$, then there are Λ', Δ and Δ' such that $\Lambda = \Lambda' \cup \text{set}(\Delta)$, $\Gamma \Vdash \mathbb{S} \parallel P \parallel \mathbb{T} : \Lambda' \cup \text{set}(\Delta')$, and Δ' depends on Δ and ℓ as in Lemma B.9.*

THEOREM 3.3 (Subject Reduction for Systems) *If $\Gamma \Vdash \mathbb{S} : \Lambda$ and $\Sigma \vdash \mathbb{S} \xrightarrow{\ell} \Sigma' \vdash \mathbb{S}'$, then $\Gamma \Vdash \mathbb{S}' : \Lambda'$, where $\Lambda \preceq \Lambda'$.*

PROOF. The proof is by induction and by cases on $\xrightarrow{\ell}$.

Case LIFT. The result follows from Theorem 3.2, Lemma B.7(1) and Proposition B.5(1).

Case CONNECTION. We have that $\Gamma \Vdash \mathbb{S} \parallel \mathbb{T} : \Lambda$ and

$$\frac{\Sigma \vdash \mathbb{S} \xrightarrow{c^{\dagger!(x:\eta)}} \Sigma \vdash \mathbb{S}' \quad \Sigma \vdash \mathbb{T} \xrightarrow{c^{\dagger?(x:\eta')}} \Sigma \vdash \mathbb{T}' \quad k \notin \text{dom}(\Sigma)}{\Sigma \vdash \mathbb{S} \parallel \mathbb{T} \xrightarrow{\tau} \Sigma, k : \eta, \tilde{k} : \eta' \vdash \mathbb{S}'[k/x] \parallel \mathbb{T}'[\tilde{k}/x]}$$

From Lemma B.7(2), we get that there exist Λ_1 and Λ_2 such that $\Lambda = \Lambda_1 \cup \Lambda_2$ and $\Gamma \Vdash \mathbb{S} : \Lambda_1$ and $\Gamma \Vdash \mathbb{T} : \Lambda_2$. By Lemmas B.10 and B.9(cases 1 and 2) we have that $\Gamma \Vdash \mathbb{S}' : \Lambda_1, x : \eta$, and $\Gamma \Vdash \mathbb{T}' : \Lambda_2, x : \eta'$. From Lemma B.8(2), we have that $\Gamma \Vdash \mathbb{S}'[k/x] : \Lambda_1, k : \eta$ and $\Gamma \Vdash \mathbb{T}'[\tilde{k}/x] : \Lambda_2, \tilde{k} : \eta'$. Applying typing rule T-PAR we get $\Gamma \Vdash \mathbb{S}'[k/x] \parallel \mathbb{T}'[\tilde{k}/x] : \Lambda'$, where $\Lambda' = \Lambda_1 \cup \Lambda_2 \cup \{k : \eta, \tilde{k} : \eta'\}$. We conclude since $\Lambda \preceq \Lambda'$ by Definition B.4(1).

Case COMMUNICATION. We have that $\Gamma \Vdash \mathbb{S} \parallel \mathbb{T} : \Lambda$ and

$$\frac{\Sigma \vdash \mathbb{S} \xrightarrow{klv} \Sigma \vdash \mathbb{S}' \quad \Sigma \vdash \mathbb{T} \xrightarrow{\tilde{k}?(x:t)} \Sigma \vdash \mathbb{T}' \quad v \in t}{\Sigma, k : \eta, \tilde{k} : \eta' \vdash \mathbb{S} \parallel \mathbb{T} \xrightarrow{\tau} \Sigma, k : \eta(!v), \tilde{k} : \eta'(?v) \vdash \mathbb{S}' \parallel \mathbb{T}'[v/x]}$$

From Lemma B.7(2), we get that there exist Λ_1 and Λ_2 such that $\Lambda = \Lambda_1 \cup \Lambda_2$ and $\Gamma \Vdash \mathbb{S} : \Lambda_1$ and $\Gamma \Vdash \mathbb{T} : \Lambda_2$. By induction hypothesis on \mathbb{S} and \mathbb{T} and Lemmas B.10 and B.9 (cases 3 and 4) we have that $\Gamma \Vdash \mathbb{S}' : \Lambda'_1$, and $\Gamma, x : t \Vdash \mathbb{T}' : \Lambda'_2$, where $\Lambda_1 \preceq \Lambda'_1$ and $\Lambda_2 \preceq \Lambda'_2$. From Lemma B.8(1), we have that $\Gamma \Vdash \mathbb{T}'[v/x] : \Lambda'_2$. Applying typing rule T-PAR we get $\Gamma \Vdash \mathbb{S}' \parallel \mathbb{T}'[v/x] : \Lambda'_1 \cup \Lambda'_2$. We conclude since $\Lambda \preceq \Lambda'_1 \cup \Lambda'_2$ by Proposition B.5(2).

Case DELEGATION. We have that $\Gamma \Vdash \mathbb{S} \parallel \mathbb{T} : \Lambda$ and

$$\frac{\Sigma \vdash \mathbb{S} \xrightarrow{klk_1} \Sigma \vdash \mathbb{S}' \quad \Sigma \vdash \mathbb{T} \xrightarrow{\tilde{k}?(x:\chi)} \Sigma \vdash \mathbb{T}' \quad \Sigma(k_1) \leq \chi}{\Sigma, k : \eta, \tilde{k} : \eta' \vdash \mathbb{S} \parallel \mathbb{T} \xrightarrow{\tau} \Sigma' \vdash \mathbb{S}' \parallel \mathbb{T}'[k_1/x]}$$

where $\Sigma' = \Sigma, k : \eta(!\Sigma(\tilde{k}_1)), \tilde{k} : \eta'(? \Sigma(\tilde{k}_1))$.

From Lemma B.7(2), we get that there exist Λ_1 and Λ_2 such that $\Lambda = \Lambda_1 \cup \Lambda_2$ and $\Gamma \Vdash \mathbb{S} : \Lambda_1$ and $\Gamma \Vdash \mathbb{T} : \Lambda_2$. By induction hypothesis on \mathbb{S} and \mathbb{T} and from Lemmas B.10 and B.9 (cases 5 and 6) we have that $\Gamma \Vdash \mathbb{S}' : \Lambda'_1$, and $\Gamma \Vdash \mathbb{T}' : \Lambda'_2, x : \eta_0$, where $\Lambda_1 \preceq \Lambda'_1$ and $\Lambda_2 \preceq \Lambda'_2$ and $\chi \leq \eta_0$. From Lemma B.8(2), we have that $\Gamma \Vdash \mathbb{T}'[k_1/x] : \Lambda'_2, k_1 : \eta_0$. Applying typing rule T-PAR we get $\Gamma \Vdash \mathbb{S}' \parallel \mathbb{T}'[k_1/x] : \Lambda'$, where $\Lambda' = \Lambda'_1 \cup \Lambda'_2 \cup \{k_1 : \eta_0\}$. We conclude since $\Lambda \preceq \Lambda'$ by Definition B.4(1) and Proposition B.5(2).

Case PAR. By straightforward induction. □

B.2 Proof of Progress

The key of our progress proof is the natural correspondence between labels of the LTS for processes and typing assumptions on internal channels in a fixed session environment.

DEFINITION B.11 (Agreement). *The agreement between the label ℓ and the assumption $k : \eta$ via the session environment Σ (notation $\ell \times_{\Sigma} k : \eta$) is the smallest relation such that $\Sigma(k) \leq \eta$ and:*

$$\begin{aligned} v \in t \text{ implies } k!v \times_{\Sigma} k : !t.\eta \quad k?(x:\psi) \times_{\Sigma} k : ?\psi.\eta \\ \Sigma(k') \leq \chi \text{ implies } k!k' \times_{\Sigma} k : !\chi.\eta' \\ \ell \times_{\Sigma} k : \eta \text{ implies } \ell \times_{\Sigma} k : \eta \oplus \eta' \\ \ell \times_{\Sigma} k : \eta \text{ implies } \ell \times_{\Sigma} k : \eta + \eta'. \end{aligned}$$

We use γ to range over finite sequences of labels of the shape $c^{\dagger?(x:\eta)}$ and $c^{\dagger!(x:\eta)}$.

LEMMA B.12. 1. If $\Gamma \vdash_* P : \Delta$ and $\text{core}(\Delta) = (k : \eta \cdot \Delta)$, then $P \xrightarrow{\ell}$ and $\ell \times_{\Sigma} k : \eta$ for all Σ such that $\Sigma(k) \leq \eta$ and $\ell = k!(k')$ implies $\Sigma(k') \leq \Delta(k')$.

2. If $\Gamma \vdash P : \Delta$ and $\text{core}(\Delta) = (k : \eta \cdot \Delta)$, then either $P \xrightarrow{\gamma}$ for some γ or $P \xrightarrow{\ell}$ and $\ell \times_{\Sigma} k : \eta$ for all Σ such that $\Sigma(k) \leq \eta$ and $\ell = k!(k')$ implies $\Sigma(k') \leq \Delta(k')$.

PROOF. (1) The last applied rule in a derivation for P can only be one of the rules T-SEND, T-RECEIVE, T-SENDS, T-RECEIVES, T-EXTCH, T-INTCH. In the first four cases P must be a communication process on channel k . In the last two cases the result follows by induction.

(2) From (1), taking into account that the last applied rule can also be T-CONNECT-REQUEST, T-CONNECT-ACCEPT, T-COMM, T-INT, T-WEAK. \square

The agreement between labels and typing assumptions is exploited in the following lemma: it assure that the parallel of processes offering labels which agree with dual assumptions always reduce in the current session environment.

LEMMA B.13. Let P and Q be such that $P \xrightarrow{\ell}$ and $\ell \times_{\Sigma} k : \eta$, $Q \xrightarrow{\ell'}$ and $\ell' \times_{\Sigma} \tilde{k} : \theta$, and $\eta \bowtie \theta$. Then $\Sigma \vdash P \parallel Q \xrightarrow{\tau}$.

PROOF. Because of Definition B.11 and the duality between η and θ we have only to consider the following cases:

1. $P \xrightarrow{k!v}$ and $Q \xrightarrow{\tilde{k}?(x:t)}$ and $v \in t$.
In this case $P \xrightarrow{\tau} k!(e).P'$, and $Q \xrightarrow{\tau} \tilde{k}?(x : \eta).Q' \{+ Q''\}$ and we conclude by the reduction rules LIFT and COMMUNICATION.
2. $P \xrightarrow{k!(k_1)}$ and $Q \xrightarrow{\tilde{k}?(z:\chi)}$ and $\Sigma(k_1) \leq \chi$.
In this case $P \xrightarrow{\tau} k!(k_1).P'$, and $Q \xrightarrow{\tau} \tilde{k}?(z : \chi).Q' \{+ Q''\}$ and we conclude by the reduction rules LIFT and DELEGATION. \square

We can show that starting from an initial system we only get coherent session environments, i.e., session environments in which dual internal channel are mapped to dual session descriptors.

DEFINITION B.14 (Coherent session environment). A session environment is coherent if whenever it contains $k : \eta$ it contains also $\tilde{k} : \eta'$ with $\eta \bowtie \eta'$.

LEMMA B.15. If \mathbb{S} is initial and $\vdash \mathbb{S} \xrightarrow{\tau} \Sigma \vdash P_1 \parallel \dots \parallel P_n$, then $\vdash P_i : \Delta_i$ for $1 \leq i \leq n$ imply:

1. $\text{dom}(\text{core}(\Delta_i))$ only contain internal channels;
2. no $\text{dom}(\text{core}(\Delta_i))$ contains an internal channel and its dual;
3. $\bigcup_{1 \leq i \leq n} \text{set}(\text{core}(\Delta_i))$ and Σ are coherent.

PROOF. By induction on $\xrightarrow{\tau}$ using Lemmas B.9 and B.10.

If \mathbb{S} is an initial system, then it is a parallel composition of sums of processes of the form $c^! (x : \eta).P$ or $c^? (x : \eta).P$. Then \mathbb{S} satisfies banally the three conditions above.

Let \mathbb{S}' be the system obtained from \mathbb{S} after a finite sequence of reductions, in which the three conditions above hold, and \mathbb{S}' can still perform a τ action. If we can apply rule CONNECTION we get a system which still satisfies the conditions because this rule pushes in both session environments the assumptions $k : \eta$ and $\tilde{k} : \eta'$, for some fresh k and with $\eta \bowtie \eta'$. If we can apply rules COMMUNICATION or DELEGATION we get a system which still satisfies the conditions because the successors of dual sessions are still dual sessions by definition of duality. \square

We restate here Lemma 3.5 by taking advantage of the definition of core . Indeed $\vdash \mathbb{S} : \Lambda$ with \mathbb{S} closed implies that the set of internal channels which occur in \mathbb{S} is the domain of $\text{core}(\Lambda)$.

LEMMA 3.5 If \mathbb{S} is initial and $\vdash \mathbb{S} \xrightarrow{\tau} \Sigma \vdash \mathbb{S}'$, and $\vdash \mathbb{S}' : \Lambda$, then $\Sigma(k) \leq \Lambda(k)$ for all $k \in \text{dom}(\text{core}(\Lambda))$.

PROOF. By induction and by cases on $\xrightarrow{\tau}$. The more interesting case is that of rule DELEGATION with $\vdash \mathbb{S} \parallel \mathbb{T} : \Lambda$ and:

$$\frac{\Sigma \vdash \mathbb{S} \xrightarrow{k!(k_1)} \Sigma \vdash \mathbb{S}' \quad \Sigma \vdash \mathbb{T} \xrightarrow{\tilde{k}?(z:\chi)} \Sigma \vdash \mathbb{T}' \quad \Sigma(k_1) \leq \chi}{\Sigma, k : \eta, \tilde{k} : \eta' \vdash \mathbb{S} \parallel \mathbb{T} \xrightarrow{\tau} \Sigma \vdash \mathbb{S}' \parallel \mathbb{T}'[k_1/z]}$$

where $\Sigma' = \Sigma, k : \eta!(\Sigma(\tilde{k}_1)), \tilde{k} : \eta'(? \Sigma(\tilde{k}_1))$. Note that $\Sigma'(k) = \Sigma(k)!(\Sigma(\tilde{k}_1))$, $\Sigma'(\tilde{k}) = \Sigma(k) (? \Sigma(\tilde{k}_1))$ and $\Sigma'(k_1) = \Sigma(k_1)$. Let $\vdash \mathbb{S}' \parallel \mathbb{T}'[k_1/z] : \Lambda'$. By Lemmas B.10 and B.9 (cases 5 and 6) we get $\Lambda(k) = !\chi'.\Lambda'(k) \{+\theta\}$, $\Lambda(\tilde{k}) = ?\chi.\Lambda'(\tilde{k}) \{+\theta\}$, $\chi \leq \Lambda'(k_1)$ and $\Lambda(k_1) \leq \chi'$. By induction $\Sigma(k) \leq \Lambda(k)$, $\Sigma(k_1) \leq \Lambda(k_1)$, and $\Sigma(k_1) \leq \chi \leq \Lambda'(k_1)$. From $\Sigma(k_1) \leq \Lambda(k_1)$ and $\Lambda(k_1) \leq \chi'$ and the coherence of Σ we derive that $\Sigma(\tilde{k}_1)$ is dual of χ' and therefore $\Lambda(k)!(\Sigma(\tilde{k}_1)) \leq \Lambda'(k)$. From $\Sigma(k) \leq \Lambda(k)$ we have $\Sigma(k)!(\Sigma(\tilde{k}_1)) \leq \Lambda(k)!(\Sigma(\tilde{k}_1))$ and so we conclude $\Sigma'(k) \leq \Lambda'(k)$. Similarly from $\Sigma(k_1) \leq \chi$ we derive that $\Sigma(\tilde{k}_1)$ is dual of χ and therefore $\Lambda(\tilde{k})(? \Sigma(\tilde{k}_1)) \leq \Lambda'(\tilde{k})$. From $\Sigma(k) \leq \Lambda(\tilde{k})$ we have $\Sigma(\tilde{k})(? \Sigma(\tilde{k}_1)) \leq \Lambda(\tilde{k})(? \Sigma(\tilde{k}_1))$ and so we conclude $\Sigma'(\tilde{k}) \leq \Lambda'(\tilde{k})$. \square

The last technical tool we use is to index the internal channels with increasing indexes according to their order of creation.

LEMMA B.16. If \mathbb{S} is initial and $\vdash \mathbb{S} \xrightarrow{\tau} \Sigma \vdash P_1 \parallel \dots \parallel P_n$, and the fresh internal channels take successive numbers according to the order of creation, then $\Gamma_i \vdash P_i : \Delta_i$ implies that the indexes of internal channels in Δ_i are decreasing for $1 \leq i \leq n$.

PROOF. By induction on $\xrightarrow{\tau}$ using Lemma B.9 and B.10. Notice that the only rule which adds channels to a possibly non empty stack is CONNECTION and this rule adds the internal channels with the maximum index. \square

THEOREM 3.6 (Progress) Every initial system satisfies the progress property:

PROOF. Let \mathbb{S} be initial and $\vdash \mathbb{S} \xrightarrow{\tau} \Sigma \vdash P_1 \parallel \dots \parallel P_n$, where $\vdash P_i : \Delta_i$. It is easy to verify that $\vdash P_1 \parallel \dots \parallel P_n : \Lambda$, where $\Lambda = \bigcup_{1 \leq i \leq n} \text{set}(\text{core}(\Delta_i))$. Assume the fresh internal channels take successive numbers according to the order of creation and j be the maximal index of the internal channel which occur in Λ . By Lemma B.15(2) there are l, l' such that $k_j \in \text{dom}(\text{core}(\Delta_l))$ and $\tilde{k}_j \in \text{dom}(\text{core}(\Delta_{l'}))$. By Lemma B.16, k_j and \tilde{k}_j must be the top of Δ_l and $\Delta_{l'}$, respectively. Note that by Lemma 3.5 Σ satisfies the conditions of Lemma B.12 for Δ_l and $\Delta_{l'}$. Therefore at least one of the following alternatives holds:

1. $P_l \xrightarrow{\gamma}$ and $\vdash P_{l'} \xrightarrow{\gamma'}$;
2. $P_l \xrightarrow{\gamma}$ and $P_{l'} \xrightarrow{\ell}$ and $\ell \times_{\Sigma} \tilde{k}_j : \Delta_{l'}(\tilde{k}_j)$;
3. $P_l \xrightarrow{\ell}$ and $\ell \times_{\Sigma} k_j : \Delta_l(k_j)$ and $P_{l'} \xrightarrow{\gamma'}$;
4. $P_l \xrightarrow{\ell}$ and $\ell \times_{\Sigma} k_j : \Delta_l(k_j)$ and $P_{l'} \xrightarrow{\ell'}$ and $\ell' \times_{\Sigma} \tilde{k}_j : \Delta_{l'}(\tilde{k}_j)$.

In the last case the coherence of Λ (assured by Lemma B.15(3)) implies the duality between $\Delta_l(k_j)$ and $\Delta_{l'}(\tilde{k}_j)$. Therefore $\Sigma \vdash P_l \parallel P_{l'} \xrightarrow{\tau}$ by Lemma B.13. \square