

Foundation of Session Types

Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, Elena Giachino, Luca Padovani

▶ **To cite this version:**

Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, Elena Giachino, Luca Padovani. Foundation of Session Types. 11th ACM SIGPLAN Conference on Principles and practice of declarative programming, 2009, Portugal. pp.219-230, 10.1145/1599410.1599437 . hal-00334435v2

HAL Id: hal-00334435
https://hal.science/hal-00334435v2

Submitted on 21 Jan 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Foundation of Session Types

Giuseppe Castagna[1] Mariangiola Dezani-Ciancaglini[2] Elena Giachino[1,2] Luca Padovani[3]

[1]PPS (CNRS) - Université Denis Diderot - Paris, France
[2]Dipartimento di Informatica - Università degli Studi di Torino - Torino, Italy
[3]Istituto 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 define the subsessioning relation. We give a coinductive characterization of subsessioning and describe algorithms to decide all the key relations defined in the article. We show that all monomorphic dyadic session types proposed in the literature are particular cases of our session types.

1. Introduction

Sessions are a common and widespread 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 given protocol which describes 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, booleans, 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 a private channel 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 behavior of sessions (i.e., their protocol) should permit the detection of communication mismatches and session deadlocks, ensuring successful termination of every session. Types are a good candidate for such a description, except that typical type systems for process algebrae 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* [21, 22] 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 connections, for delegation, 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 (see Section 3). 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 foundation of session types that aims at being as much language independent as possible. To achieve language independence, we design our types around the standard π-calculus: 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 [12], 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, all monomorphic, dyadic session types proposed in the literature are particular instances of the session types discussed here. Second, 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, the language independence is further increased by the fact that all our definitions are semantic-based, rather than syntax-oriented. Third, we enhance compositionality of branching constructs because the result of the combination of different branches is automatically computed at 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 finally we describe algorithms to decide all key relations defined in the article. Section 3 provides a more technical discussion about how our approach subsumes and improves existing session types proposals. Furthermore, it presents a π-calculus and an object-oriented calculus along with corresponding type systems guaranteeing a suitably defined progress property

in each case. Section 4 summarizes the contributions of our work, draws connections with some of the most closely related papers, and sketches future directions of research.

2. Session types

2.1 Type syntax

As said in the introduction we have two kinds of channels: public ones that are used to connect and establish a private channel for the conversation, and these private ones. At type level this distinction corresponds to two different syntactic categories. Public channels are associated with a *session type* of the form $\mathsf{begin}.\eta$. This type classifies channels ready to initiate a conversation on some private channel that will follow the description η. Thus, private channels are classified by *session descriptors*, ranged over by η. Session descriptors and types are defined by the grammar:

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

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 $?\mathsf{t}$ and $!\mathsf{t}$ 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 channel which already started a conversation and will continue it according to the behavior described by the session descriptor η. In particular, 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, $(!\mathsf{t}.\eta) + \mathsf{end}$ and $!\mathsf{t}.\eta + \mathsf{end}$ denote the same session descriptor, which is different from $!\mathsf{t}.(\eta + \mathsf{end})$. 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 $\mathsf{begin}.\eta$ which classify yet-to-be-used public channels whose conversation follows the descriptor η. 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:

$$?\mathsf{Real}.!\mathsf{Int}.\mathsf{end} + ?\mathsf{Int}.!\mathsf{Bool}.\mathsf{end} \tag{1}$$

The session descriptor above declares that if a process (that uses a channel with that behavior) 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 ($\mathsf{t} \setminus \mathsf{s}$ stands for $\mathsf{t}\wedge\neg\mathsf{s}$):

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

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

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

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

$$?\mathsf{Real}.?\mathsf{Int}.\mathsf{end} + ?\mathsf{Int}.?\mathsf{Bool}.\mathsf{end} \tag{4}$$

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: $!(\mathsf{Real} \setminus \mathsf{Int}).!\mathsf{Int}.\mathsf{end}$. 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 $?(\mathsf{Real}\setminus\mathsf{Int}).?\mathsf{Int}.\mathsf{end}+?\mathsf{Int}.?(\mathsf{Bool}\wedge\mathsf{Int}).\mathsf{end}$ which is equivalent to $?(\mathsf{Real} \setminus \mathsf{Int}).?\mathsf{Int}.\mathsf{end}$ 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 behavior is the solution of the equation) $\eta = \mathsf{end} + ?\mathsf{Int}.\eta$ 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 [17, 8] 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) $\mathsf{t} = \mathsf{t}\vee\mathsf{t}$ or $\eta = \eta\oplus\eta$; technically they say that the binary relation $\triangleright$ defined by $\mathsf{t}_1\vee\mathsf{t}_2\triangleright\mathsf{t}_i$, $\mathsf{t}_1\wedge\mathsf{t}_2\triangleright\mathsf{t}_i$, $\neg\mathsf{t}\triangleright\mathsf{t}$, $\chi_1\vee\chi_2\triangleright\chi_i$, $\chi_1\wedge\chi_2\triangleright\chi_i$, $\neg\chi\triangleright\chi$, $\eta_1+\eta_2\triangleright\eta_i$, $\eta_1\oplus\eta_2\triangleright\eta_i$ is Noetherian (that is, strongly normalizing), which gives an induction principle on terms that we will use in our proofs without any further explicit reference to the relation $\triangleright$. The third condition states that recursion cannot escape prefixes and thus it rules out terms such as $\eta = ?\eta.\mathsf{end}$; this restriction generalizes the typing technique used in all works on (recursive)

session types that forbids delegation of a channel over itself [22, 30] (strictly speaking we disallow types that in the cited works are not inhabited by any program) while, technically, it allows us to stratify the definition of the subtyping and subsessioning relations, stratification we use in the proof of Theorem 2.6.

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:

$$\vdash \mathsf{v} : \mathsf{t}_1 \vee \mathsf{t}_2 \quad \Longleftrightarrow \quad \vdash \mathsf{v} : \mathsf{t}_1 \text{ or } \vdash \mathsf{v} : \mathsf{t}_2 \tag{5}$$

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 as understood that values for a *session type* $\mathsf{begin}.\eta$ are channels explicitly associated with or tagged by that type (or, because of subtyping, by a $\mathsf{begin}.\eta'$ subtype of $\mathsf{begin}.\eta$: more about that later on).

We do not include in our session descriptors a construct for parallel composition (as opposed to [23, 5], for example). Since we assume an interleaving semantics of parallel composition, having two different choices is enough for faithfully describing possibly concurrent actions by means of well-known expansion laws (see [12] for an example).

The intuitive semantics of session descriptors we outlined above is formalized in the following section.

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 labeled 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 $[\![.]\!]$ such that $[\![\mathsf{t}]\!] = \{\mathsf{v} \mid \vdash \mathsf{v} : \mathsf{t}\}$ and that $[\![\mathsf{t} \wedge \mathsf{s}]\!] = [\![\mathsf{t}]\!] \cap [\![\mathsf{s}]\!]$, $[\![\mathsf{t} \vee \mathsf{s}]\!] = [\![\mathsf{t}]\!] \cup [\![\mathsf{s}]\!]$, and $[\![\neg \mathsf{t}]\!] = \mathscr{V} \setminus [\![\mathsf{t}]\!]$ (where $\mathscr{V}$ denotes the set of all values). The same interpretation can then be used to *define* the subtyping relation (denoted by "$<:$") as follows:

$$\mathsf{t} <: \mathsf{s} \quad \stackrel{\text{def}}{\Longleftrightarrow} \quad [\![\mathsf{t}]\!] \subseteq [\![\mathsf{s}]\!]$$

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 [17]: 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 $\mathsf{v} \in \mathsf{t}$ 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 behaviors 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 between two channels 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 $!\mathsf{Int}.?(\mathsf{Bool} \vee \mathsf{Int}).\mathsf{end}$ is dual to (1), too.

Note also that some session descriptors have no dual, for example $?(\mathsf{Bool} \wedge \mathsf{Int}).\mathsf{end}$, 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 $\eta \bowtie \theta$ 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: $[\![\chi \wedge \chi']\!] = [\![\chi]\!] \cap [\![\chi']\!]$, $[\![\chi \vee \chi']\!] = [\![\chi]\!] \cup [\![\chi']\!]$, $[\![\neg\chi]\!] = \mathscr{S} \setminus [\![\chi]\!]$ (where $\mathscr{S}$ denotes the set of all *viable* descriptors); and use it to semantically define the subsieving (and subsessioning) relation (denoted by "$\leq$"):

$$\chi \leq \chi' \quad \stackrel{\text{def}}{\Longleftrightarrow} \quad [\![\chi]\!] \subseteq [\![\chi']\!] \tag{6}$$

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 type constructor $\mathsf{begin}.\eta$. 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 session channel is "used". We make the assumption—matched by everyday practice—that there is unique way to consume a public channel c of type $\mathsf{begin}.\eta$, by invoking the service associated with c and starting a conversation that conforms to the protocol described by η. Thus it is safe to replace c with a different channel d of a smaller type $\mathsf{begin}.\eta'$ only if the conversation, which follows the protocol described by η and which was originally intended to occur with the service associated with c, works seamlessly with the service associated with d. This happens if at each step of the conversation the service associated with d is willing to receive at least all the messages accepted by c and never sends any message that c would not send. Roughly speaking, the service associated with d is "more tolerant" than the one associated with c. Of course, there are fewer services that, as d, support a η' conversation, since they must be able to satisfy more demanding clients. Therefore, passing from $\mathsf{begin}.\eta$ to $\mathsf{begin}.\eta'$ corresponds to restricting the set of possible services one can safely use, that is to say, reducing the sets of possible duals. So the intuition—that we will formalize in Section 2.3 by equation (11)—is that $\mathsf{begin}.\eta' <: \mathsf{begin}.\eta$

[1] This shows that our duality is semantically defined: $?(\mathsf{Bool} \wedge \mathsf{Int}).\mathsf{end}$ is *not* dual of $!(\mathsf{Bool} \wedge \mathsf{Int}).\mathsf{end}$ as a syntactic approach would suggest; both descriptors have no dual.

if and only if η' has fewer duals than η, that is by (6), $\eta' \leq \eta$. For instance we have that $?\mathsf{Int}.\mathsf{end} \leq ?\mathsf{Real}.\mathsf{end}$ since every descriptor that is dual of $?\mathsf{Int}.\mathsf{end}$ is also dual of $?\mathsf{Real}.\mathsf{end}$. Similarly $\mathsf{begin}.?\mathsf{Int}.\mathsf{end} <: \mathsf{begin}.?\mathsf{Real}.\mathsf{end}$ since if a process that uses a channel of type $\mathsf{begin}.?\mathsf{Real}.\mathsf{end}$ is well typed, then the process obtained by replacing this channel for a different one of type $\mathsf{begin}.?\mathsf{Int}.\mathsf{end}$ 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 form $\mathsf{begin}.\eta$ (and not, say, $\mathsf{begin}.\eta \vee \mathsf{begin}.\eta'$), which yields the following interpretation for session types: $[\![\mathsf{begin}.\eta]\!] = \{\mathsf{c}^{\mathsf{t}} \mid \mathsf{t} <: \mathsf{begin}.\eta\} = \{\mathsf{c}^{\mathsf{begin}.\eta'} \mid \eta' \leq \eta\}$, that is

$$[\![\mathsf{begin}.\eta]\!] = \{\mathsf{c}^{\mathsf{begin}.\eta'} \mid \forall \theta,\ \theta \bowtie \eta' \Rightarrow \theta \bowtie \eta\} \qquad (7)$$

The next step is to formally define the duality relation for which we have to characterize the observables of the session descriptors.

2.2.2 Semantics of session descriptors

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

$$\text{(TR1)}\ \frac{}{\mathsf{end} \xrightarrow{\checkmark} \mathsf{end}} \qquad \text{(TR2)}\ \frac{}{\eta \oplus \eta' \longrightarrow \eta} \qquad \text{(TR3)}\ \frac{\eta \longrightarrow \eta'}{\eta + \eta'' \longrightarrow \eta' + \eta''}$$

$$\text{(TR4)}\ \frac{\eta \xrightarrow{\mu} \eta'}{\eta + \eta'' \xrightarrow{\mu} \eta'} \qquad \text{(TR5)}\ \frac{\eta \xrightarrow{!\mathsf{v}} \eta''}{\eta + \eta' \longrightarrow \eta} \qquad \text{(TR6)}\ \frac{\eta \xrightarrow{!\eta''} \eta'''}{\eta + \eta' \longrightarrow \eta}$$

$$\text{(TR7)}\ \frac{\mathsf{v} \in \mathsf{t}}{?\mathsf{t}.\eta \xrightarrow{?\mathsf{v}} \eta} \qquad \text{(TR8)}\ \frac{\mathsf{v} \in \mathsf{t}}{!\mathsf{t}.\eta \xrightarrow{!\mathsf{v}} \eta} \qquad \text{(TR9)}\ \frac{\eta \in \chi}{?\chi.\eta' \xrightarrow{?\eta} \eta'} \qquad \text{(TR10)}\ \frac{\eta \in \chi}{!\chi.\eta' \xrightarrow{!\eta} \eta'}$$

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

Rules (TR1-TR4) are straightforward: end emits a "tick" (TR1); an internal choice silently decides the behavior 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 behavior of the other branch. This is typical of process languages with two distinct choice operators, such as CCS without τ's [12].

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 $!\mathsf{t}.\eta$ indicates that the descriptor is ready to emit some value of type t (TR8), while $?\mathsf{t}.\eta$ indicates that the descriptor is ready to accept any value of type t (TR7). While this approach is reminiscent of the so-called *early semantics* in process algebras [27] (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 subsessioning 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 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 $\mathsf{begin}.\eta$). 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 behavior 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 $\eta \in \chi$ to denote that $\eta \in [\![\chi]\!]$.[2]

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 [9]. 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 behavior 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 t it is ready to accept *any* value of type t (the choice of the particular value is left to the sender) while when a process sends a value of type 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 labeled 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 $\eta \not\longrightarrow$ if there exists no η' such that $\eta \longrightarrow \eta'$.

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

We say that η must input μ*, written* $\eta \Downarrow \mu$*, if* $\eta \Longrightarrow \eta' \not\longrightarrow$ *implies* $\eta' \xrightarrow{\mu}$ *and* μ *is either* $?\mathsf{v}$*, or* $?\eta$*, 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 $!\mathsf{Int}.\mathsf{end} \oplus \mathsf{end} \downarrow !3$ and $!\mathsf{Int}.\mathsf{end} \oplus \mathsf{end} \downarrow \checkmark$; on the other hand we have $!\mathsf{Int}.\mathsf{end} + \mathsf{end} \not\downarrow \checkmark$, since $!\mathsf{Int}.\mathsf{end} + \mathsf{end} \not\longrightarrow \mathsf{end}$. Similarly we have $?\mathsf{Int}.\mathsf{end} \oplus ?\mathsf{Real}.\mathsf{end} \Downarrow ?3$ because the action $?3$ is always guaranteed independently of the internal choice, whereas

[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.

$?\mathsf{Int.end} \oplus ?\mathsf{Real.end} \not\Downarrow ?\sqrt{2}$ because $?\mathsf{Int.end} \oplus ?\mathsf{Real.end} \longrightarrow ?\mathsf{Int.end}$ and $?\mathsf{Int.end} \not\Downarrow ?\sqrt{2}$.

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 $\eta \Longrightarrow \eta' \not\longrightarrow$ 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 behaviors 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 labeled 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 $?\mathsf{Real.!Int.end} + ?\mathsf{Int.!Bool.end} \xrightarrow{?3} \mathsf{!Bool.end}$ (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 $\mathsf{!Int.end} \oplus \mathsf{!Bool.end}$, 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 $\eta \stackrel{\mu}{\Longrightarrow}$. The* successor *of η* after *μ, written $\eta\langle\mu\rangle$, is defined as: $\eta\langle\mu\rangle = \oplus\{\eta' \mid \eta \stackrel{\mu}{\Longrightarrow} \eta'\}$.*

For example, $(?\mathsf{Real.!Int.end}+?\mathsf{Int.!Bool.end})\langle ?3\rangle = \mathsf{!Int.end}\oplus \mathsf{!Bool.end}$ but $(?\mathsf{Real.!Int.end}+?\mathsf{Int.!Bool.end})\langle ?\sqrt{2}\rangle = \mathsf{!Int.end}$. Note that $\eta\langle\mu\rangle$ is well defined because there is always a finite number of residuals η' such that $\eta \stackrel{\mu}{\Longrightarrow} \eta'$. 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 $\overline{\mu}$, be defined by: (i) $\overline{\checkmark} = \checkmark$; (ii) $\overline{\dagger \mathsf{v}} = \overline{\dagger}\mathsf{v}$; (iii) $\overline{\dagger \eta} = \overline{\dagger}\eta$; where $\overline{!} = ?$ and $\overline{?} = !$. 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 $\eta_2 \Downarrow \overline{\mu}$ and $\eta_1\langle\mu\rangle \bowtie \eta_2\langle\overline{\mu}\rangle$ 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 $?\mathsf{Real.!Int.end} + ?\mathsf{Int.!Bool.end} \bowtie \mathsf{!Int.?(Int} \vee \mathsf{Bool).end}$, but $?\mathsf{Real.!Int.end} + ?\mathsf{Int.!Bool.end} \not\bowtie \mathsf{!Int.?Int.end}$ because the descriptor on the right is not sure that its partner will answer with an integer. However $?\mathsf{Real.!Int.end} + ?\mathsf{Int.!Bool.end} \bowtie \mathsf{!(Real} \setminus \mathsf{Int).?Int.end}$. As another example, we have $?\mathsf{Int.end} \oplus ?\mathsf{Real.end} \bowtie \mathsf{!Int.end}$ because $?\mathsf{Int.end} \oplus ?\mathsf{Real.end} \Downarrow ?\mathsf{v}$ for every $\mathsf{v} \in \mathsf{Int}$, however $?\mathsf{Int.end} \oplus ?\mathsf{Real.end} \not\bowtie \mathsf{!}\sqrt{2}\mathsf{.end}$ because $?\mathsf{Int.end} \oplus ?\mathsf{Real.end} \not\Downarrow ?\sqrt{2}$.

The reader may have observed that there is a circularity in the definitions of duality and of the labeled 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 $[\![\chi]\!]$ 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 $\mathsf{begin}.\eta$, then its values are all the channels of the form $\mathsf{c}^{\mathsf{begin}.\eta'}$ such that $\theta \bowtie \eta'$ implies $\theta \bowtie \eta$ for all θ (*cf.* equation (7)): so also the definition of the relation $\mathsf{v} \in \mathsf{t}$ depends on that of duality. The following theorem proves that this circularity is not one.

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

PROOF. Thanks to condition 3 of Definition 2.1 recursion cannot enter descriptor prefixes (the condition that $\alpha.\eta$ is not a subtree of α). Therefore it is relatively easy to stratify the previous definitions. In particular let us define a weight as follows: a type has weight 0 if it does not contain session types; a descriptor has weight 0 if it contains just 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 descriptor is of weight $i+1$ if the prefixes occurring in it are of weight at most i (here is where the condition 3 ensures that this definition is well founded for all session types). Next we define a weight for each relation we introduced so far: each $\eta \stackrel{\mu}{\longrightarrow} \eta'$ 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 $\mathsf{v} \in \mathsf{t}$ to be of weight 0 if t is of weight 0; then notice that both $\eta \in \chi$ and $\mathsf{v} \in \mathsf{t}$ 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 in this section are well founded. □

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

The notion of duality is also a useful tool for better understanding the semantics of session descriptors. Let us revisit part of the LTS in the light of duality:

Output of values. Rules (TR7-TR8) in Section 2.2 state that a descriptor is ready to respectively input and output some value of type t. The use of single values in labels may appear at first look surprising, as one would expect to see labels pretty similar to the fired actions. If we stated, say, that $?\mathsf{t}.\eta$ emits $?\mathsf{t}$ and $!\mathsf{t}.\eta$ emits $!\mathsf{t}$ (or, to be more liberal, $?\mathsf{t}'$ and $!\mathsf{t}'$ with $\mathsf{t}' <: \mathsf{t}$), then we would obtain quite a different semantics. In particular, it would no longer be possible to prove the following equations (where "=" denotes the equality induced by the relation $\leq$):

$$?(\mathsf{t}_1 \vee \mathsf{t}_2).\eta = ?\mathsf{t}_1.\eta + ?\mathsf{t}_2.\eta \quad (8)$$

$$!(\mathsf{t}_1 \vee \mathsf{t}_2).\eta = !\mathsf{t}_1.\eta + !\mathsf{t}_2.\eta \quad (9)$$

$$!(\mathsf{t}_1 \vee \mathsf{t}_2).\eta = !\mathsf{t}_1.\eta \oplus !\mathsf{t}_2.\eta \quad (10)$$

In particular, the right hand-sided descriptors would no longer be smaller than the left hand-sided ones. Consider for example an instance of (9) where we take $\mathsf{t}_1 \equiv \mathsf{Int}$ and $\mathsf{t}_2 \equiv \mathsf{Bool}$ (here and henceforward we use "$\equiv$" to denote syntactic equality). It is clear that the two descriptors in the equation share the same set of duals (that is, they have the same semantics): the duals of both descriptors are descriptors that accept both an integer and a Boolean and then are dual of η. With the current definition of the LTS this holds true:

whatever signal the left hand descriptor emits will be matched by every dual of the right hand since, in both cases, these signals will be on values of type $\mathsf{Int} \vee \mathsf{Bool}$. Here is where the strong disjunction property on union types (5) is used: the left hand descriptor cannot emit a value that is neither an Int nor Bool. If the transition system had emitted types rather than values, then the duals of the right hand descriptor would not be able to match the signal $!(\mathsf{Int} \vee \mathsf{Bool})$ since each summand of the right hand descriptor could at most emit Int or Bool. We could have introduced some extra definition of sets of emitted signals and saturated these sets with unions for internal and external sums, but the current solution avoids all this clutter.

A similar reasoning holds for rules (TR9-TR10) because of the disjunction property we are going to prove next.

Internal choices, intersections, and disjoint unions. Using the definition of duality it is easy to see that $[\![\eta \oplus \eta']\!] = [\![\eta \wedge \eta']\!]$ 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. $\theta \leq \chi_1 \vee \chi_2 \iff \theta \leq \chi_1$ *or* $\theta \leq \chi_2$.

PROOF. 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. □

This property is essential to prove decidability of $\leq$.

A similar property holds for types of the form $\mathsf{begin}.\eta$:

PROPOSITION 2.8. $\mathsf{begin}.\eta <: \mathsf{begin}.\eta_1 \vee \mathsf{begin}.\eta_2 \iff \mathsf{begin}.\eta <: \mathsf{begin}.\eta_1$ *or* $\mathsf{begin}.\eta <: \mathsf{begin}.\eta_2$

PROOF. Recall that $[\![\mathsf{begin}.\eta]\!] = \{\mathsf{c}^{\mathsf{begin}.\eta'} \mid \mathsf{begin}.\eta' <: \mathsf{begin}.\eta\}$ (we cannot yet assume that $\mathsf{begin}.\eta' <: \mathsf{begin}.\eta$ implies $\eta' \leq \eta$). Take any channel of the form $\mathsf{c}^{\mathsf{begin}.\eta}$. By the strong disjunction property (equation (5)) $\mathsf{c}^{\mathsf{begin}.\eta} \in \mathsf{begin}.\eta_i$ for some i in $[1, 2]$. But by applying the definition of the semantics of session types to $\mathsf{begin}.\eta_i$, this implies that for that i, $\mathsf{begin}.\eta <: \mathsf{begin}.\eta_i$. □

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 $!\mathsf{Int}.\mathsf{end}+?\mathsf{Bool}.\mathsf{end}$ and $?\mathsf{Int}.\mathsf{end}+!\mathsf{Bool}.\mathsf{end}$ 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 behavior 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 above one, 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 subsessioning, 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 $\mathsf{begin}.\eta$, 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 defined in [17] 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 $\mathtt{t}_1 <: \mathtt{t}_2$ if and only if $\mathtt{t}_1 \wedge \neg \mathtt{t}_2 <: \emptyset$ (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 [7, 17] for details). In conclusion, in order to define our subtyping relation all we need is to decide when $\bigvee_{k \in K}(\bigwedge_{i \in I_k} \mathsf{begin}.\eta_i \wedge \bigwedge_{j \in J_k} \neg \mathsf{begin}.\eta_j) <: \emptyset$. 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 $\bigwedge_{i \in I} \mathsf{begin}.\eta_i <: \bigvee_{j \in J} \mathsf{begin}.\eta_j$.

As regards session channels, we notice that a value has type $(\mathsf{begin}.\eta) \wedge (\mathsf{begin}.\eta')$ if and only if it has type $\mathsf{begin}.(\eta \oplus \eta')$. Also note that $\mathsf{begin}.\eta <: \mathsf{begin}.\eta_1 \vee \mathsf{begin}.\eta_2$ if and only if $\mathsf{begin}.\eta <: \mathsf{begin}.\eta_1$ or $\mathsf{begin}.\eta <: \mathsf{begin}.\eta_2$ (Proposition 2.8). Therefore the semantic subtyping relation for the types of Section 2.1 is completely defined by (the semantic subtyping framework of [17] and) the following equation

$$\bigwedge_{i \in I} \mathsf{begin}.\eta_i <: \bigvee_{j \in J} \mathsf{begin}.\eta_j \iff \exists j \in J : \bigoplus_{i \in I} \eta_i \leq \eta_j \qquad (11)$$

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

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

that is the form discussed at the end of Section 2.2.1.

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 [11] (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 $\bot$.

DEFINITION 2.9 (Coinductive Viability). $\eta^{\bowtie}$ *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, as stated in the next proposition, which is proved in Appendix A.2.

PROPOSITION 2.10. $\eta^{\bowtie}$ *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). *$\eta \leqq \eta'$ is the largest relation between session descriptors such that $\eta^{\bowtie}$ implies $\eta'^{\bowtie}$ and*

1. *$\eta' \not\Downarrow$ and $\eta' \downarrow \mu$ imply $\eta \downarrow \mu$ with $\eta\langle\mu\rangle \leqq \eta'\langle\mu\rangle$, and*
2. *$\eta \Downarrow \mu$ and $\eta\langle\mu\rangle^{\bowtie}$ imply $\eta' \Downarrow \mu$ with $\eta\langle\mu\rangle \leqq \eta'\langle\mu\rangle$, 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 subsession 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 $?\mathsf{Int}.\mathsf{end} \not\leqq ?\mathsf{Int}.\mathsf{end} + \mathsf{end}$ 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 $?\mathsf{Int}.\mathsf{end} \leqq ?\mathsf{Int}.\mathsf{end} + \mathsf{end}$. 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 $?\mathsf{Int}.!\emptyset.\mathsf{end} + ?\mathsf{Bool}.\mathsf{end} \not\leqq ?\mathsf{Bool}.\mathsf{end}$, 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 $\not\leqq$. 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. The proof of this theorem is the content of A.3.

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

2.5 Properties of the subsession relation

Table 1 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 $\varnothing$ to denote either $\emptyset$ (the empty type) or $\bot$ (the least sieve) according to the context.

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 [11] 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 coincides 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 $\bot$ 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 $\mathsf{t} \wedge \mathsf{s} = \emptyset$, then both descriptors are $\bot$ (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 possi-

(E1)	$\eta + \eta$	$=$	η	
(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)	$?\mathsf{t}.\eta + ?\mathsf{s}.\eta$	$=$	$?(\mathsf{t} \vee \mathsf{s}).\eta$	
(E7)	$?\chi.\eta + ?\chi'.\eta$	$=$	$?(\chi \vee \chi').\eta$	
(E8)	$\eta + \bot$	$=$	η	
(I1)	$\eta \oplus \eta$	$=$	η	
(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)	$?\mathsf{t}.\eta \oplus ?\mathsf{s}.\eta'$	$=$	$?(\mathsf{t} \wedge \mathsf{s}).(\eta \oplus \eta')$	
(I7)	$?\chi.\eta \oplus ?\chi'.\eta'$	$=$	$?(\chi \wedge \chi').(\eta \oplus \eta')$	
(I8)	$\eta \oplus \bot$	$=$	$\bot$	
(B1)	$?\psi.\eta$	$=$	$\bot$	$(\psi = \varnothing)$
(B2)	$!\psi.\eta$	$=$	$\bot$	$(\psi = \varnothing)$
(B3)	$?\mathsf{t}.\eta \oplus ?\chi.\eta'$	$=$	$\bot$	
(B4)	$?\psi.\eta \oplus !\psi'.\eta'$	$=$	$\bot$	
(B5)	$?\psi.\eta \oplus \mathsf{end}$	$=$	$\bot$	
(O1)	$!\psi.\eta + \mathsf{end}$	$=$	$!\psi.\eta$	$(\psi \neq \varnothing)$
(O2)	$!\psi.\eta + ?\psi'.\eta'$	$=$	$!\psi.\eta$	$(\psi \neq \varnothing)$
(O3)	$!\psi.\eta + !\psi'.\eta'$	$=$	$!\psi.\eta \oplus !\psi'.\eta'$	$(\psi, \psi' \neq \varnothing)$
(O4)	$!\mathsf{t}.\eta \oplus !\mathsf{s}.\eta$	$=$	$!(\mathsf{t} \vee \mathsf{s}).\eta$	$(\mathsf{t}, \mathsf{s} \neq \emptyset)$
(O5)	$!\chi.\eta \oplus !\chi'.\eta$	$=$	$!(\chi \vee \chi').\eta$	$(\chi, \chi' \neq \bot)$
(S1)	$?\mathsf{t}.\eta$	$\leq$	$?(\mathsf{t} \vee \mathsf{s}).\eta$	
(S2)	$?\chi.\eta$	$\leq$	$?(\chi \vee \chi').\eta$	
(S3)	$!(\mathsf{t} \vee \mathsf{s}).\eta$	$\leq$	$!\mathsf{t}.\eta$	$(\mathsf{t} \neq \emptyset)$
(S4)	$!(\chi \vee \chi').\eta$	$\leq$	$!\chi.\eta$	$(\chi \neq \bot)$
(S5)	$\eta \oplus \eta'$	$\leq$	η	

Table 1. Selected equalities and inequalities.

bility is to send a descriptor that is dual of both θ and θ'. Rule (I8) states that $\bot$ 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 $\bot$. 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 a 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 $\mathsf{t} \neq \emptyset$ and $\chi \neq \bot$. 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 with two remarks. First of all, the rules of Table 1 allow us to derive the following decomposition laws:

$$?\mathsf{t}.\eta + ?\mathsf{s}.\eta' = ?(\mathsf{t} \setminus \mathsf{s}).\eta + ?(\mathsf{s} \setminus \mathsf{t}).\eta' + ?(\mathsf{t} \wedge \mathsf{s}).(\eta \oplus \eta')$$
$$!\mathsf{t}.\eta \oplus !\mathsf{s}.\eta' = !(\mathsf{t} \setminus \mathsf{s}).\eta \oplus !(\mathsf{s} \setminus \mathsf{t}).\eta' \oplus !(\mathsf{t} \wedge \mathsf{s}).(\eta \oplus \eta')$$

the latter rule holding when none of the sets $\mathsf{t} \setminus \mathsf{s}$, $\mathsf{s} \setminus \mathsf{t}$, and $\mathsf{t} \wedge \mathsf{s}$ is empty. Similar rules can be derived for inputs and outputs of sieves, as opposed to types. These rules play a fundamental role in all the algorithms that will follow because they allow us to rewrite external and internal sums so that every summand of the sum begins with a prefix that is disjoint from (emits labels that are not emitted by) the prefix of any other summand.

The second remark concerns 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 $?\mathsf{Int}.\mathsf{end} \leq ?\mathsf{Real}.\mathsf{end}$ however $?\mathsf{Int}.\mathsf{end} + ?\sqrt{2}.!3.\mathsf{end} \not\leq ?\mathsf{Real}.\mathsf{end} + ?\sqrt{2}.!3.\mathsf{end}$. The reason is that in widening $?\mathsf{Int}.\mathsf{end}$ to $?\mathsf{Real}.\mathsf{end}$ we create an interference with the term $?\sqrt{2}.!3.\mathsf{end}$ 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 (I6) we have $?(\mathsf{Int} \vee \sqrt{2}).\mathsf{end} \oplus ?\mathsf{Int}.!3.\mathsf{end} = ?\mathsf{Int}.(\mathsf{end} \oplus !3.\mathsf{end})$, but $(?(\mathsf{Int} \vee \sqrt{2}).\mathsf{end} \oplus ?\mathsf{Int}.!3.\mathsf{end}) + ?\sqrt{2}.!4.\mathsf{end} \neq ?\mathsf{Int}.(\mathsf{end} \oplus !3.\mathsf{end}) + ?\sqrt{2}.!4.\mathsf{end}$. Here the action $?\sqrt{2}$ is not guaranteed by $?(\mathsf{Int} \vee \sqrt{2}).\mathsf{end} \oplus ?\mathsf{Int}.!3.\mathsf{end}$ and (I6) tells us whose consequence is that in practice the capability of $?(\mathsf{Int} \vee \sqrt{2}).\mathsf{end}$ of receiving $\sqrt{2}$ is useless. However, removing this capability may also remove interferences in the context of an external choice, making (I6) 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 $\bot$ is totally inert. For instance we have $?\mathsf{Int}.\mathsf{end} \oplus !\mathsf{Int}.!\emptyset.\mathsf{end} = \bot$ and $\bot + ?\mathsf{Bool}.!3.\mathsf{end} = ?\mathsf{Bool}.!3.\mathsf{end}$, but $(?\mathsf{Int}.\mathsf{end} \oplus !\mathsf{Int}.!\emptyset.\mathsf{end}) + ?\mathsf{Bool}.!3.\mathsf{end} = (?\mathsf{Int}.\mathsf{end} + ?\mathsf{Bool}.!3.\mathsf{end}) \oplus (!\mathsf{Int}.!\emptyset.\mathsf{end} + ?\mathsf{Bool}.!3.\mathsf{end}) = \bot$. Rule (B5) suffers from a similar problem, which is slightly less severe because end denotes a terminated descriptor.

2.6 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 $\chi \wedge \neg\chi' \leq \bot$. The l.h.s. can be rewritten in disjunctive normal form whose definition for sieves is (we convene that $\bigvee_{i\in\emptyset} \chi_i = \sum_{i\in\emptyset} \eta_i = \bot$):

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 (11)). 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* $\bigwedge_{j\in J} \lambda_{ij} \neq \bot$ *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* $\bigoplus_{i\in I} !\psi_i.\eta_i\{ \oplus \mathsf{end}\}$ *or* $\sum_{i\in I} ?\psi_i.\eta_i\{ + \mathsf{end}\}$*, where for all* $i \in I$*,* $\psi_i \neq \varnothing$*,* ψ_i *and* η_i *are in strong normal form and for all* $i, j \in I$*,* $i \neq j$ *implies* $\psi_i \wedge \psi_j = \varnothing$*, and* end *is possibly missing.*

The following theorem proves that every sieve can be effectively transformed in strong normal form. Its proof, which is done by

simultaneous induction with that of Theorem 2.18 later on, is the content of Appendix A.1.

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 $\bot$, and then apply the algorithm whose core rules are given in Table 2.

Table 2.

$$\text{(END)}\quad \frac{}{\mathsf{end} \leq \mathsf{end}} \qquad \text{(PREFIX)}\quad \frac{\eta \leq \eta'}{\alpha.\eta \leq \alpha.\eta'} \qquad \text{(MIX-CHOICES)}\quad \frac{}{\bigoplus_{i\in I} \eta_i \oplus \mathsf{end} \leq \sum_{j\in J} \eta'_j + \mathsf{end}}$$

$$\text{(EXT-CHOICES)}\quad \frac{I \subseteq J \qquad \eta_i \leq \eta'_i \ {}^{(\forall i \in I)}}{\sum_{i\in I} \eta_i \leq \sum_{j\in J} \eta'_j} \qquad \text{(INT-CHOICES)}\quad \frac{J \subseteq I \qquad \eta_j \leq \eta'_j \ {}^{(\forall j \in J)}}{\bigoplus_{i\in I} \eta_i \leq \bigoplus_{j\in J} \eta'_j}$$

Table 2. Algorithmic subsessioning structural rules.

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 is done by the rules in Table 14 in Appendix A.1. These rules perform repeated applications of the decomposition laws described in Section 2.5.

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 $\mathscr{S}$. 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 2.16. *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 $\{\theta \mid \eta' \implies \theta \not\rightarrow\}$. Now choose any value v such that for all θ in this set $\theta \xrightarrow{!\mathsf{v}}$ implies that there exists $\mathsf{v}' \neq \mathsf{v}$ such that $\theta \xrightarrow{!\mathsf{v}'}$. 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 $\eta \equiv !\mathsf{v}.\mathsf{end}$ proves the result. □

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

LEMMA 2.17. *For all $\alpha_1, \ldots, \alpha_n, \eta, \eta'$, if $\eta \leq \eta'$, then $\alpha_1.\cdots\alpha_n.\eta \leq \alpha_1.\cdots\alpha_n.\eta'$.*

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. □

THEOREM 2.18 (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 $\overline{\eta} \leq \eta'$, where we write $\overline{\eta}$ for the *canonical dual* of η, namely the least descriptor in the set-theoretic interpretation of η. Computing $\overline{\eta}$ is trivial once η is in strong normal form (see Theorem 2.15): it suffices to change every ? into !, every $+$ into $\oplus$ and viceversa, 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

In the following section we first show how the session type theories discussed in various papers fit in our framework, and then we type a π-calculus and an object-oriented calculus in order to show the flexibility and espressivity of our approach.

3.1 Expressing existing session type theories

All monomorphic dyadic session type theories proposed in the literature are particular cases of the session descriptors discussed here.

The first version of session descriptors can be traced back to the seminal work of Honda, Vasconcelos, and Kubo [22] in the context of process algebras. In the same context [3] enhances sessions with correspondence assertions. Later on Vasconcelos, Gay, and Ravara have extended the approach to multithreaded functional languages [29]. In all these works session descriptors embed two n-ary operators for internal and external choice, which are strictly coupled with the communication of labels that indicate the selected branch in a choice. The session descriptors of [22, 3, 29] can be written in our model as follows:

$$\begin{array}{lcl} \eta & ::= & \mathsf{end} \mid \alpha.\eta \mid \bigoplus_{i\in I} !\ell_i.\eta_i \mid \Sigma_{i\in I} ?\ell_i.\eta_i \\ \alpha & ::= & !\mathsf{t} \mid ?\mathsf{t} \mid !\eta \mid ?\eta \end{array}$$

where we consider labels as singleton types. The same syntax can be used to describe the session descriptors developed for CORBA [28], Boxed Ambients [18], and the ones used to type the Conversation Calculus [5]. To draw a parallel with sequential languages, to pass from the systems above to our framework is like passing from a language with labels and records, in which the selection of the code to be executed is driven by the label received, to a language with dynamically resolved overloaded functions in which the code is selected according to the type of the received value.

In the context of object-oriented languages, session type theories assuring type safety and progress were investigated in MOOSE [14], AMOOSE [10], and MOOSE$_{<:}$ [13]. The syntax of all session descriptors discussed in the above papers, but for [13] where bounded polymorphism is considered, are special cases of our model, after identifying recursive types with their infinite unfolding. More specifically, the syntax of the session descriptors in [14] and [10] can be reduced to:

$$\begin{array}{lcl} \eta & ::= & \mathsf{end} \mid \alpha.\eta \mid !\mathsf{true}.\eta \oplus !\mathsf{false}.\eta \mid ?\mathsf{true}.\eta + ?\mathsf{false}.\eta \\ \alpha & ::= & !\mathsf{t} \mid ?\mathsf{t} \mid !\eta \mid ?\eta \end{array}$$

since these works deal with sessions where branching is controlled by means of boolean conditions.

A calculus that amalgamates the notion of session-based communication with the one of object-oriented programming is presented in [6, 2]. In these works sessions and methods are unified, channels are implicit, and delegation is realized by means of session calls. Branching is determined by the runtime type of the ob-

ject being communicated. Session descriptors with this form of dependencies can be written very naturally in our model as:

$$\eta \quad ::= \quad \mathsf{end} \;\mid\; \bigoplus_{i\in I} !\mathsf{C}_i.\eta_i \;\mid\; \Sigma_{i\in I} ?\mathsf{C}_i.\eta_i$$

where C_i are class names in [2] and generic class names in [6]. The cited works hardcode different selection policies for branching. We can easily reproduce them all by using Boolean combinators on classes.

3.2 π-calculus

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 [26]). The intuition is that bound outputs, written $\mathsf{c}!(x : \eta)$, 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, η, P) =

mypublicname$?(x : ?\top.\mathsf{end})$.	/* accept	*/
$x?(y : \eta).P$	/* catch&handle	*/
$+\, x?(y : \neg\eta)$.	/* catch&delegate	*/
nextpublicname$!(z : !\top.\mathsf{end})$.	/* 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 ($\top$ 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 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 $\mathsf{begin}.!\top.\mathsf{end}$. Under these hypotheses and provided that y is used in P according to η, the process above result typable by the type system given right after the formal definision of PIC syntax and semantics we present next.

3.2.1 Syntax and semantics

The main design criterion for our process calculus PIC is minimality and similarity to π-calculus: we define the smallest calculus that allows us to use all the characteristics of our session types. The syntax is given in Table 3. It is a π-calculus with internal and external choices, free outputs on bound channels and bound outputs on free channels respectively denoted by $\mathsf{h}!\mathsf{e}$ and $\mathsf{a}!(\mathsf{x} : \eta)$. Processes are the possibly infinite regular trees that are generated by the productions in Table 3 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, e.g., $\mathsf{P} = \mathsf{P} \oplus \mathsf{P}$ and—as for types—it provides an induction principle based on the Noetherian relation $\mathsf{P}_1 + \mathsf{P}_2 \triangleright \mathsf{P}_i$ and $\mathsf{P}_1 \oplus \mathsf{P}_2 \triangleright \mathsf{P}_i$.

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 enough[3] and are implemented resorting to *internal channels*, denoted by k. These appear greyed in the productions 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 "$\tilde{}$" is a bijective mapping from internal channels to internal channels with $\mathsf{k} \neq \tilde{\mathsf{k}}$ and that is an involution (i.e., $\tilde{\tilde{\mathsf{k}}} = \mathsf{k}$, a technique quite common in calculi for sessions). We say that k and $\tilde{\mathsf{k}}$ 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 (that is session public names, ranged over by c and tagged by their smallest type, which is of the form $\mathsf{begin}.\eta$ 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 $\mathsf{v} \in \mathsf{t}$ that associates each host language value with its types and each channel value with the types larger than or equal to its tag.

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

Table 3. Syntax of PIC processes and systems.

The set of labels ℓ is defined by

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

where

$$\mathsf{m} ::= \mathsf{v} \mid \mathsf{k} \qquad \mathsf{n} ::= \mathsf{c}^{\mathsf{t}} \mid \mathsf{k}$$

These are already quite informative. They tell us that synchronization can happen only on closed channels (and not on variables ranging over channels), 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 4. 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 system presented in Table 5, more in particular in the last two rules where synchronization takes place only if the objects of the outputs match the types/sieves of the input. Note also that a new session is started (rule CONNECTION) when a bound output is performed on a channel value, in that case a new pair k, $\tilde{\mathsf{k}}$ of internal channels is spawned. 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.

[3] We could have introduced at system level restrictions restrictions of the form $(\nu \mathsf{c} : \mathsf{begin}.\eta)$ 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.

$$\text{R-SEND}\ \frac{\mathsf{e}\downarrow\mathsf{m}}{\mathsf{k}!\mathsf{e}.\mathsf{P}\xrightarrow{\mathsf{k}!\mathsf{m}}\mathsf{P}} \qquad \text{R-RECEIVE}\ \frac{}{\mathsf{n}?(\mathsf{x}:\psi).\mathsf{P}\xrightarrow{\mathsf{n}?(\mathsf{x}:\psi)}\mathsf{P}} \qquad \text{R-CONNECT}\ \frac{}{\mathsf{c}^{\mathsf{t}}!(\mathsf{x}:\eta).\mathsf{P}\xrightarrow{\mathsf{c}^{\mathsf{t}}!(\mathsf{x}:\eta)}\mathsf{P}}$$

$$\text{R-INTCH}\ \mathsf{P}\oplus\mathsf{Q}\xrightarrow{\tau}\mathsf{P} \qquad \text{R-EXTINT}\ \frac{\mathsf{P}\xrightarrow{\tau}\mathsf{P}'}{\mathsf{P}+\mathsf{Q}\xrightarrow{\tau}\mathsf{P}'+\mathsf{Q}}$$

$$\text{R-EXTSEND}\ \frac{\mathsf{P}\xrightarrow{\mathsf{k}!\mathsf{m}}\mathsf{P}'}{\mathsf{P}+\mathsf{Q}\xrightarrow{\tau}\mathsf{P}} \qquad \text{R-EXTCH}\ \frac{\mathsf{P}\xrightarrow{\ell}\mathsf{P}' \qquad \ell\neq\tau}{\mathsf{P}+\mathsf{Q}\xrightarrow{\ell}\mathsf{P}'}$$

Table 4. PIC process reduction rules.

$$\text{LIFT}\ \frac{\mathsf{P}\xrightarrow{\ell}\mathsf{P}'}{\Sigma\vdash\mathsf{P}\xrightarrow{\ell}\Sigma\vdash\mathsf{P}'} \qquad \text{PAR}\ \frac{\Sigma\vdash\mathbb{S}\xrightarrow{\ell}\Sigma'\vdash\mathbb{S}'}{\Sigma\vdash\mathbb{S}\parallel\mathbb{T}\xrightarrow{\ell}\Sigma'\vdash\mathbb{S}'\parallel\mathbb{T}}$$

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

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

$$\text{DELEGATION}\ \frac{\Sigma\vdash\mathbb{S}\xrightarrow{\mathsf{k}!\mathsf{k}'}\Sigma\vdash\mathbb{S}' \quad \Sigma\vdash\mathbb{T}\xrightarrow{\tilde{\mathsf{k}}?(\mathsf{x}:\chi)}\Sigma\vdash\mathbb{T}' \quad \Sigma(\mathsf{k}')\leq\chi}{\Sigma,\mathsf{k}:\eta,\tilde{\mathsf{k}}:\eta'\vdash\mathbb{S}\parallel\mathbb{T}\xrightarrow{\tau}\Sigma,\mathsf{k}:\eta\langle!\Sigma(\tilde{\mathsf{k}}')\rangle,\tilde{\mathsf{k}}:\eta'\langle?\Sigma(\tilde{\mathsf{k}}')\rangle\vdash\mathbb{S}'\parallel\mathbb{T}'[\mathsf{k}'/\mathsf{x}]}$$

Table 5. PIC system reduction rules.

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

3.2.2 Typing of PIC

The original motivation for introducing session types [21, 22] 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 the progress property, i.e., that a started session cannot get stuck if the required connections are available, was first proposed in [15].

In the present calculus, as in [13], 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 the system $\mathsf{k}!3 \parallel \tilde{\mathsf{k}}?(\mathsf{x}:\mathsf{Bool})$ is stuck. We present in this section 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 $\vdash\mathbb{S}\stackrel{\tau}{\Longrightarrow}\Sigma\vdash\mathbb{S}'\stackrel{\tau}{\not\longrightarrow}$ implies that either $\mathbb{S}'$ does not contain internal channels or $\mathbb{S}'\xrightarrow{\mathsf{c}(\mathsf{z}:\eta)}$.*

Our process calculus 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 $\mathsf{c}^{\mathsf{t}}!(\mathsf{z}:\eta).\mathsf{z}!(3).\mathsf{z}?(\mathsf{x}:\mathsf{Real}).(\mathsf{z}!(\mathsf{x})\oplus\mathsf{z}!(\mathsf{true}))$ 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 $\eta=!\mathsf{Int}.?\mathsf{Real}.(!\mathsf{Real}.\mathsf{end}\oplus!\mathsf{Bool}.\mathsf{end})$ and t is (a subtype of) $\mathsf{begin}.\eta$. 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 *current session*, is spawned, then the process can act according to (combinations of) the following options:

1. establishing a new connection;
2. performing a communication on the current session;
3. ending 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);
4. delegating on the current session the innermost, not ended, enclosing session;[4] the process stops using the delegated session;
5. receiving a delegated session and using it in the continuation as the current session.

Such restrictive behaviour 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 a couple of examples of deadlock.

Let $\mathsf{t}_1=\mathsf{begin}.!\mathsf{Int}.\mathsf{end}$, $\eta_1=!\mathsf{Int}.\mathsf{end}$ and $\eta_2=?\mathsf{Int}.\mathsf{end}$. A first simple example of deadlock is given by

$$\begin{array}{l}\mathsf{c}^{\mathsf{t}_1}!(\mathsf{z}_1:\eta_1).\mathsf{b}^{\mathsf{t}_1}?(\mathsf{z}_2:\eta_2).\mathsf{z}_2?(\mathsf{x}:\mathsf{Int}).\mathsf{z}_1!6\\ \quad\parallel\mathsf{c}^{\mathsf{t}_1}?(\mathsf{z}_3:\eta_2).\mathsf{b}^{\mathsf{t}_1}!(\mathsf{z}_4:\eta_1).\mathsf{z}_3?(\mathsf{x}:\mathsf{Int}).\mathsf{z}_4!5\end{array}$$

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 (that is the second one) since it make 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

$$\begin{array}{l}\mathsf{c}^{\mathsf{t}_1}?(\mathsf{z}_1:\eta_2).\mathsf{b}^{\mathsf{t}_1}?(\mathsf{z}_2:\eta_2).\mathsf{z}_1?(\mathsf{x}:\mathsf{Int})\ +\ \mathsf{z}_2?(\mathsf{x}:\mathsf{Int})\\ \quad\parallel\mathsf{c}^{\mathsf{t}_1}!(\mathsf{z}_3:\eta_1).\mathsf{z}_3!6\parallel\mathsf{b}^{\mathsf{t}_1}!(\mathsf{z}_4:\eta_1).\mathsf{z}_4!5\end{array}$$

and in the similar system obtained by replacing $\oplus$ to $+$. Another example is

$$\begin{array}{l}\mathsf{c}^{\mathsf{t}_1}?(\mathsf{z}_1:\eta_3).\mathsf{z}_1?(\mathsf{x}:\mathsf{Int})\ +\ \mathsf{b}^{\mathsf{t}_1}?(\mathsf{z}_2:\eta_2).\mathsf{z}_2?(\mathsf{x}:\mathsf{Int}).\mathsf{z}_1?(\mathsf{x}:\mathsf{Bool})\\ \quad\parallel\mathsf{c}^{\mathsf{t}_1}!(\mathsf{z}_3:\eta_1).\mathsf{z}_3!6\parallel\mathsf{b}^{\mathsf{t}_1}!(\mathsf{z}_4:\eta_1).\mathsf{z}_4!5\end{array}$$

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

More subtle examples of deadlock spring from session delegation, whereby a (sequential) process can receive the dual of a chan-

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

$$\text{T-AX}\ \frac{}{\Gamma, x : t \vdash x : t} \qquad \text{T-VAL}\ \frac{v \in t}{\Gamma \vdash v : t} \qquad \text{T-SUB}\ \frac{\Gamma \vdash e : t' \qquad t' <: t}{\Gamma \vdash e : t} \qquad \text{T-SYS}\ \frac{\Gamma \vdash P : \Delta}{\Gamma \Vdash P : \mathsf{set}(\Delta)} \qquad \text{T-PAR}\ \frac{\Gamma \Vdash \mathbb{S} : \Lambda_1 \qquad \Gamma \Vdash \mathbb{T} : \Lambda_2}{\Gamma \Vdash \mathbb{S} \,\|\, \mathbb{T} : \Lambda_1 \cup \Lambda_2}$$

$$\text{T-ZERO}\ \frac{}{\Gamma \vdash \mathbf{0} : -} \qquad \text{T-WEAK}\ \frac{\Gamma \vdash P : \Delta \qquad h \notin \mathsf{dom}(\Delta \cup \Gamma)}{\Gamma \vdash P : (h : \mathsf{end} \cdot \Delta)} \qquad \text{T-INT}\ \frac{\Gamma \vdash P : \Delta \qquad \Gamma \vdash Q : \Delta}{\Gamma \vdash P \oplus Q : \Delta}$$

$$\text{T-CONNECT-REQUEST}\ \frac{\Gamma \vdash P : (x : \eta \cdot \Delta) \qquad \Gamma \vdash a : \mathsf{begin}.\eta}{\Gamma \vdash a!(x : \eta).P : \Delta} \qquad \text{T-CONNECT-ACCEPT}\ \frac{\Gamma \vdash P : (x : \eta \cdot \Delta) \qquad \eta \bowtie \eta'}{\Gamma \vdash c^{\mathsf{begin}.\eta'}?(x : \eta).P : \Delta} \qquad \text{T-COMM}\ \frac{\Gamma \vdash_* P : \Delta}{\Gamma \vdash P : \Delta}$$

$$\text{T-RECEIVE}\ \frac{\Gamma, x : t \vdash P : (h : \eta \cdot \Delta)}{\Gamma \vdash_* h?(x : t).P : (h : ?t.\eta \cdot \Delta)} \qquad \text{T-SEND}\ \frac{\Gamma \vdash e : t \qquad \Gamma \vdash P : (h : \eta \cdot \Delta)}{\Gamma \vdash_* h!e.P : (h : !t.\eta \cdot \Delta)} \qquad \text{T-RECEIVES}\ \frac{\Gamma \vdash P : (h : \eta \cdot x : \eta') \qquad \chi \leq \eta'}{\Gamma \vdash_* h?(x : \chi).P : (h : ?\chi.\eta)}$$

$$\text{T-SENDS}\ \frac{\Gamma \vdash P : (h : \eta \cdot \Delta) \qquad \eta' \leq \chi}{\Gamma \vdash_* h!h'.P : (h : !\chi.\eta \cdot (h' : \eta' \cdot \Delta))} \qquad \text{T-INTCH}\ \frac{\Gamma \vdash_* P : (h : \eta_1 \cdot \Delta) \qquad \Gamma \vdash_* Q : (h : \eta_2 \cdot \Delta)}{\Gamma \vdash_* P \oplus Q : (h : \eta_1 \oplus \eta_2 \cdot \Delta)} \qquad \text{T-EXTCH}\ \frac{\Gamma \vdash_* P : (h : \eta_1 \cdot \Delta) \qquad \Gamma \vdash_* Q : (h : \eta_2 \cdot \Delta)}{\Gamma \vdash_* P + Q : (h : \eta_1 + \eta_2 \cdot \Delta)}$$

Table 6. Type system for PIC.

nel it already owns, making synchronization impossible. Consider

$$c^{t_1}?(z_1 : \eta_2).b^{t_2}!(z_2 : \eta_4).z_2!z_1 \,\|\, c^{t_1}!(z_3 : \eta_1).\\ b^{t_2}?(z_4 : \eta_5).z_4?(x : \eta_2).x?(y : \mathsf{Int}).z_3!6$$

where $t_2 \equiv \mathsf{begin}.!(?\mathsf{Int}.\mathsf{end}).\mathsf{end}$, $\eta_4 = !(?\mathsf{Int}.\mathsf{end}).\mathsf{end}$, $\eta_5 = ?(?\mathsf{Int}.\mathsf{end}).\mathsf{end}$. 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 6. The judgements for processes have the forms

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

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 rightmost 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 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 $c^{\mathsf{begin}.\eta'}?(x : \eta).P$ 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 behaviour dual of the one that tags c. Therefore the tag of a public channel declares the behaviour 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 allow 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, correspond to enforcing invariance when accepting connections.

To deal with the deadlocks induced by sums we use the $\vdash_*$ judgements which assure that the processes in the conclusion offer communications just on the channel which is on the top of the stack. Two processes can be composed by means of an internal choice only if either are typed by exactly the same stack (rule T-INT) or by means of an internal or external choice if they differ for the typing of only one channel on which a communication is immediately available (rules T-INTCH, T-EXTCH). The example of deadlock due to delegation is avoided by requiring that the only other internal channel which 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.

Since evaluation consumes session descriptors and adds fresh initial channels with their descriptors, we need to introduce a partial order $\preccurlyeq$ on session stacks and session environments 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* $\Delta \preccurlyeq \Delta'$.

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* $\Lambda \preccurlyeq \Lambda'$.

The definition of $\preccurlyeq$ together with the proofs of the above theorems are given in Appendix B.1.

Progress clearly fails for systems which contain free variables or internal channels that are not properly paired. For this reason our typing can only assure 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 process where 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 behaviours. For example consider the system

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

We get

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

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

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

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

An interesting consequence of the above lemma is *fidelity* of communications i.e., that once a session is started the communications are done in the expected order and exchange data of the expected types.

We state now the progress theorem whose proof is the content of Appendix B.2.

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.

3.3 An OO calculus

In this section we will show how we can naturally enhance the typing of [2] using the general session descriptors: in particular we will allow *session overloading*, i.e. the same session name can be declared with different session descriptors in a class hierarchy. At run time an appropriate session body will be chosen using the duality and subfiltering relations. As in the original paper we do not have cast and overriding, which are orthogonal to our approach. For simplicity we avoid to consider while commands.

3.3.1 Syntax of the OO calculus

The calculus of [2] is based on *Featherweight Java* [24] (abbreviated with FJ). We use the standard convention of denoting with $\overline{\xi}$ a sequence of elements $\xi_1, ..., \xi_n$.

Programs are defined from a collection of classes. The metavariables $\mathtt{C}$ and $\mathtt{D}$, possibly with subscripts, range over class names.

(class)	$\mathtt{L}$	::=	$\mathtt{class\ C} \lhd \mathtt{C}\,\{\,\overline{\mathtt{t\,f}};\ \overline{\mathtt{S}}\,\}$
(session)	$\mathtt{S}$	::=	$\mathtt{t}\,\eta\,\mathtt{s}\,\{\,\mathtt{e}\,\}$
(expression)	$\mathtt{e}$	::=	$\mathtt{x} \mid \mathtt{this} \mid \mathtt{o} \mid \mathtt{e;\,e} \mid \mathtt{e.f} := \mathtt{e} \mid \mathtt{e.f} \mid \mathtt{new\,C}(\overline{\mathtt{e}})$
		|	$\mathtt{e.s}\,\{\mathtt{e} : \eta\} \mid \mathtt{e} \bullet \mathtt{s}\,\{\,\mathtt{k} : \chi\}$
		|	$\mathtt{k.\,sendC}(\mathtt{e})\{\mathtt{t} \Rightarrow \mathtt{e} \,[]\, \mathtt{t} \Rightarrow \mathtt{e}\}$
		|	$\mathtt{k.\,recC}(\mathtt{x})\{\mathtt{t} \Rightarrow \mathtt{e} \,[]\, \mathtt{t} \Rightarrow \mathtt{e}\}$
(parallel threads)	P	::=	$\mathtt{e} \mid P \parallel P$

Table 7. OO: Syntax, where syntax occurring only at runtime appears shaded.

Each class has a name, a list of *fields* of the form $\overline{\mathtt{t\,f}}$, where $\mathtt{f}$ represents the field name and $\mathtt{t}$ its type, and a list of *sessions* of the form $\mathtt{t}\,\eta\,\mathtt{s}\,\{\,\mathtt{e}\,\}$, where $\mathtt{t}$ is the return type, η the session descriptor, $\mathtt{s}$ the session name, and $\mathtt{e}$ the session body. For the sake of conciseness the symbol $\lhd$ represents class extension, as in [24]. All classes are defined as extensions of the topmost class `Object`.

Expressions include variables, that are both standard term variables $\mathtt{x}$ and the special variables `this`. The variable `this` is considered implicitly bound in any session declaration.

In a *session request* $\mathtt{e.s}\,\{\mathtt{e}' : \eta\}$ we call the expression $\mathtt{e}'$ the *cobody* of the request (since it will be evaluated concurrently with the body of requested session). The session descriptor η represents the communications offered by the expression $\mathtt{e}'$.

In the *session delegation* expression, $\mathtt{e} \bullet \mathtt{s}\,\{\mathtt{k} : \chi\}$, the filter χ represents the communication requirements in the choice of the session body and the channel $\mathtt{k}$ is added by the operational semantics in order to keep track of the channel to pass to the delegated session.

Channels are implicit in the source language syntax. At runtime, a pair of dual communication channels are introduced at each new session request.

The body of a *communication expression* is a pair of alternatives $\{\mathtt{t}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{t}_2 \Rightarrow \mathtt{e}_2\}$, whose choice depends on the class of the object that is sent or received.

A *runtime expression* is either a user expression (i.e. an expression in Table 7 without shaded syntax) or an expression containing channels and/or object indentifiers. Furthermore, threads of runtime expressions can occur at runtime (see the operational semantics). Parallel threads are ranged over by P. Fully evaluated objects will be represented by object identifiers denoted by $\mathtt{o}$.

3.3.2 Auxiliary Functions

As in FJ, a class table *CT* is a mapping from class names to class declarations with domain $\mathscr{D}(CT)$. Then a program is a pair $(CT, \mathtt{e})$ of a class table (containing all the class definitions of the program) and an expression $\mathtt{e}$ (an expression belonging to the source language representing the program's main entry point). The class `Object` has no fields and its declaration does not appear in *CT*. As in FJ, from any *CT* we can read off the subtype relation between classes, as the transitive closure of $\lhd$ clause.

We assume a fixed *CT*. Thus, in the following, instead of writing $CT(\mathtt{C}) = \mathtt{class}\ \ldots$ we will simply write `class C` ... . The class table *CT* satisfies some usual sanity conditions as in FJ [24], but we allow the same session name to be declared with different session descriptors in a class hierarchy.

The *fields* lookup function is as in FJ. As for field type lookup we distinguish between the contexts where the field is used for reading ($ftype_r$) from those where it is for writing ($ftype_w$).

The *stype* lookup function returns a set of sets of session descriptors; in case it is invoked with a class name as argument, it will return a singleton containing a set of session descriptors. In

this way we take into account the overloading and the fact that in general the type of the receiver is a boolean combination of classes.

The *rtype* and the *sbody* lookup functions take as arguments also a session descriptor: they return the return type of a session and the body of a session, respectively.

All these functions are defined in Table 8, where we define

$$m(f,\overline{a},\mathtt{t}_1,\mathtt{t}_2) = \begin{cases} f(\overline{a},\mathtt{t}_1) & \text{if } \mathtt{t}_1 \leq \mathtt{t}_2, \\ f(\overline{a},\mathtt{t}_2) & \text{if } \mathtt{t}_2 \leq \mathtt{t}_1, \\ \bot & \text{otherwise.} \end{cases}$$

.

3.3.3 Operational Semantics

Session invocation involves the creation of concurrent and communicating threads. These threads communicate by means of a pair of dual communication channels.

Objects passed in asynchronous communications are stored in a *heap*. A heap h is a finite mapping with domain consisting of objects and channel names. Its syntax is given by:

$$h \ ::= \ [\,] \ \mid \ \mathtt{o} \mapsto (\mathtt{C}, \overline{\mathtt{f} : \mathtt{o}}) \ \mid \ \mathtt{k} \mapsto \overline{\mathtt{o}} \ \mid \ h :: h$$

where $::$ denotes heap concatenation.

The operational semantics associates to $\mathtt{k}$ and $\tilde{\mathtt{k}}$ two different queues of messages in the heap; when a thread, which uses the channel $\mathtt{k}$, wants to receive a message it will inspect the queue associated to $\mathtt{k}$, while, when it sends a message it will add it to the queue associated to $\tilde{\mathtt{k}}$.

During evaluation, any expression $\mathtt{new}\ \mathtt{C}(\overline{\mathtt{o}})$ will be replaced by a new object identifier $\mathtt{o}$. The heap will then maps the object identifier $\mathtt{o}$ to the pair $(\mathtt{C}, \overline{\mathtt{f} : \mathtt{o}})$ of its class name $\mathtt{C}$ and the list of its fields with corresponding objects $\overline{\mathtt{o}}$; this mapping is denoted by $\mathtt{o} \mapsto (\mathtt{C}, \overline{\mathtt{f} : \mathtt{o}})$.

The form $h[\mathtt{o} \mapsto h(\mathtt{o})[\mathtt{f} \mapsto \mathtt{o}']]$ denotes the update of the field $\mathtt{f}$ of the object $\mathtt{o}$ with the object $\mathtt{o}'$.

Channel names are mapped to queues of objects: $\mathtt{k} \mapsto \overline{\mathtt{o}}$. The heap produced by $h[\mathtt{k} \mapsto \overline{\mathtt{o}}]$ maps the channel $\mathtt{k}$ to the queue $\overline{\mathtt{o}}$. With some abuse of notation we write $\mathtt{o} :: \overline{\mathtt{o}}$ and $\overline{\mathtt{o}} :: \mathtt{o}$ to denote the queue whose first and last element is $\mathtt{o}$, respectively.

Heap membership for object identifiers and channels is checked using standard set notation, by identifying h with its domain, we can also write $\mathtt{o} \in h$, and $\mathtt{k} \in h$.

The values that can result from normal termination are parallel threads of fully evaluated objects.

In the reduction rules we make use of the special *channel addition* operation $\wr \ldots \int$ defined in Table 9. We denote by $\mathtt{e} \wr \mathtt{k} \int$ the source expression $\mathtt{e}$ in which all occurrences of receive, send, and delegation expressions which *are not* within the cobody of a session request are extended, so that they explicitly mention the channel $\mathtt{k}$ they will use (remember that channel names are not written by the programmer).

The reduction is a relation between pairs of threads and heaps:

$$P, h \longrightarrow P', h'$$

Reduction rules use evaluation contexts (based on runtime syntax) that capture the notion of the "next subexpression to be reduced":

$$\begin{array}{rcl} \mathscr{E} & ::= & [-] \mid \mathscr{E}; \mathtt{e} \mid \mathscr{E}.\mathtt{f} \mid \mathscr{E}.\mathtt{f} := \mathtt{e} \mid \mathtt{o}.\mathtt{f} := \mathscr{E} \mid \mathscr{E}.\mathtt{s}\,\{\mathtt{e} : \eta\} \mid \\ & & \mathscr{E} \bullet \mathtt{s}\,\{\mathtt{k} : \chi\} \mid \mathtt{k}.\mathtt{sendC}(\mathscr{E})\{\mathtt{t}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{t}_2 \Rightarrow \mathtt{e}_2\} \end{array}$$

The explicit mention of the evaluation context is needed in rule SESSREQ-R (Table 10), in which a new thread is generated in parallel with the evaluation context.

Reduction rules are in Table 10. Rule PAR-R models the execution of parallel threads. In this rule parallel composition is considered modulo structural equivalence. As usual, we define structural equivalence rules asserting that parallel composition is associative and commutative:

$$P \,||\, P_1 \equiv P_1 \,||\, P \quad P \,||\, (P_1 \,||\, P_2) \equiv (P \,||\, P_1) \,||\, P_2$$
$$P \equiv P' \ \Rightarrow \ P \,||\, P_1 \equiv P' \,||\, P_1$$

The successive four rules define the execution of standard object-oriented constructions.

Rule SESSREQ-R models the connection between the cobody $\mathtt{e}$ of a session request $\mathtt{o}.\mathtt{s}\,\{\mathtt{e} : \eta\}$ and the body $\mathtt{e}'$ of the session $\mathtt{s}$, in the class of the object $\mathtt{o}$, under the condition that $\mathtt{e}$ and $\mathtt{e}'$ have dual session descriptors. This connection is established through a pair of fresh channels $\mathtt{k}, \tilde{\mathtt{k}}$. For this purpose the expression $\mathtt{o}.\mathtt{s}\,\{\mathtt{e} : \eta\}$ reduces, in the same context, to its own cobody $\mathtt{e}\wr\mathtt{k}\int$ and in parallel, outside the context, it spawns the body $[\mathtt{o}/\mathtt{this}]\mathtt{e}'\wr \bowtie \mathtt{k}\int$ of the called session. The explicit substitution of $\mathtt{k}$ in $\mathtt{e}$ and of $\tilde{\mathtt{k}}$ in $\mathtt{e}'$ ensures that the communication is on the fresh dual channels $\mathtt{k}$ and $\tilde{\mathtt{k}}$. Thus, an object can serve *any number* of session requests.

Rule SESSDEL-R replaces the session delegation $\mathtt{o} \bullet \mathtt{s}\,\{\mathtt{k} : \chi\}$ by $[\mathtt{o}/\mathtt{this}]\mathtt{e} \wr \mathtt{k}\int$, where $\mathtt{e}$ is a body of the session $\mathtt{s}$, in the class of the object $\mathtt{o}$, such that the session descriptor of $\mathtt{e}$ is bigger or equal to the filter required by the delegation constructor. A part of the communication is delegated via the channel $\mathtt{k}$ to the object $\mathtt{o}$: this delegation is transparent for the thread using the dual channel $\tilde{\mathtt{k}}$. When the delegated job is over, the original thread can resume the communication via the channel $\mathtt{k}$.

The communication rule for $\mathtt{sendC}$, SENDCASE-R, puts the object $\mathtt{o}$, i.e. the result of evaluating the expression $\mathtt{e}$, in the queue associated to the dual channel $\tilde{\mathtt{k}}$ of the communication channel $\mathtt{k}$. The computation then proceeds with the expression $\mathtt{e}_i$, if $h(\mathtt{o}) = (\mathtt{C}, _)$ and $\mathtt{C} \leq \mathtt{t}_i$. Clearly if $\mathtt{C} \leq \mathtt{t}_1$ and $\mathtt{C} \leq \mathtt{t}_2$, the choice between $\mathtt{e}_1$ and $\mathtt{e}_2$ is non-deterministic, but we could choose some policy in order to have a deterministic choice, as done in [6], [2]. Dually the receive communication rule takes an object $\mathtt{o}$ from the queue associated to channel $\mathtt{k}$ and returns the expression $[\mathtt{o}/\mathtt{x}]\mathtt{e}_i$, if $h(\mathtt{o}) = (\mathtt{C}, _)$ and $\mathtt{C} \leq \mathtt{t}_i$.

3.3.4 Typing

The presence in our calculus of the sequencing operator and of other operators which implicitly put in sequence the evaluation of expressions makes useful the definition of a concatenation operator ($\circ$) between session descriptors and the addition of the neutral element with respect to concatenation (ε). Instead we do not need the explicit session terminator $\mathtt{end}$, so to sum up we have the

$$\mathtt{e}\wr\mathtt{k}\int = \begin{cases} \mathtt{e}_1 \wr \mathtt{k} \int ; \mathtt{e}_2 \wr \mathtt{k}\int & \text{if } \mathtt{e} = \mathtt{e}_1;\mathtt{e}_2, \\ \mathtt{e}_1 \wr \mathtt{k} \int .\mathtt{f} & \text{if } \mathtt{e} = \mathtt{e}_1.\mathtt{f}, \\ \mathtt{e}_1 \wr \mathtt{k} \int .\mathtt{f} := \mathtt{e}_2 \wr \mathtt{k}\int & \text{if } \mathtt{e} = \mathtt{e}_1.\mathtt{f} := \mathtt{e}_2, \\ \mathtt{e}_1 \wr \mathtt{k} \int .\mathtt{s}\,\{\mathtt{e}_2 : \eta\} & \text{if } \mathtt{e} = \mathtt{e}_1.\mathtt{s}\,\{\mathtt{e}_2 : \eta\}, \\ \mathtt{e}_1 \wr \mathtt{k} \int \bullet \mathtt{s}\,\{\mathtt{k} : \chi\} & \text{if } \mathtt{e} = \mathtt{e}_1 \bullet \mathtt{s}\,\{\chi\}, \\ \mathtt{k}.\mathtt{sendC}(\mathtt{e}_0 \wr \mathtt{k}\int)\{\overline{\mathtt{C} \Rightarrow \mathtt{e} \wr \mathtt{k}\int}\} & \text{if } \mathtt{e} = \mathtt{sendC}(\mathtt{e}_0)\{\overline{\mathtt{C} \Rightarrow \mathtt{e}}\}, \\ \mathtt{k}.\mathtt{recC}(\mathtt{x})\{\overline{\mathtt{C} \Rightarrow \mathtt{e} \wr \mathtt{k}\int}\} & \text{if } \mathtt{e} = \mathtt{recC}(\mathtt{x})\{\overline{\mathtt{C} \Rightarrow \mathtt{e}}\}, \\ \mathtt{e} & \text{otherwise.} \end{cases}$$

Table 9. OO: Channel Addition

$$\frac{\{\mathtt{this} : \mathtt{C}\} \vdash \mathtt{e} : \mathtt{t} \,\S\, \eta}{\mathtt{t}\,\eta\,\mathtt{s}\,\{\,\mathtt{e}\,\}\ \mathtt{ok\ in\ C}}\ \textsc{Sess-WF} \qquad \frac{\mathtt{D\ ok} \qquad \overline{\mathtt{S}}\ \mathtt{ok\ in\ C}}{\mathtt{class\ C} \lhd \mathtt{D}\,\{\,\overline{\mathtt{t}\,\mathtt{f}};\ \overline{\mathtt{S}}\,\}\ \mathtt{ok}}\ \textsc{Class-WF}$$

Table 12. OO: Well-formed Class Tables.

$$\mathit{fields}(\texttt{Object}) = \bullet \qquad \frac{\mathit{fields}(\texttt{D}) = \overline{\texttt{t}'\,\texttt{f}'} \qquad \texttt{class C} \lhd \texttt{D}\,\{\,\overline{\texttt{t}\,\texttt{f}};\ \overline{\texttt{S}}\,\}}{\mathit{fields}(\texttt{C}) = \overline{\texttt{t}\,\texttt{f}}, \overline{\texttt{t}'\,\texttt{f}'}}$$

$$\frac{\mathit{fields}(\texttt{C}) = \overline{\texttt{t}\,\texttt{f}}}{\mathit{ftype}_w(\texttt{f}_i, \texttt{C}) = \mathit{ftype}_r(\texttt{f}_i, \texttt{C}) = \texttt{t}_i}$$

$$\mathit{ftype}_w(\texttt{f}, \texttt{t}_1 \wedge \texttt{t}_2) = m(\mathit{ftype}_w, \texttt{f}, \texttt{t}_1, \texttt{t}_2) \qquad \mathit{ftype}_r(\texttt{f}, \texttt{t}_1 \wedge \texttt{t}_2) = m(\mathit{ftype}_r, \texttt{f}, \texttt{t}_1, \texttt{t}_2)$$

$$\mathit{ftype}_w(\texttt{f}, \texttt{t}_1 \vee \texttt{t}_2) = \begin{cases} \mathit{ftype}_w(\texttt{f}, \texttt{t}_1) & \text{if } \mathit{ftype}_w(\texttt{f}, \texttt{t}_1) \leq \mathit{ftype}_w(\texttt{f}, \texttt{t}_2), \\ \mathit{ftype}_w(\texttt{f}, \texttt{t}_2) & \text{if } \mathit{ftype}_w(\texttt{f}, \texttt{t}_2) \leq \mathit{ftype}_w(\texttt{f}, \texttt{t}_1), \\ \bot & \text{otherwise.} \end{cases}$$

$$\mathit{ftype}_r(\texttt{f}, \texttt{t}_1 \vee \texttt{t}_2) = \mathit{ftype}_r(\texttt{f}, \texttt{t}_1) \vee \mathit{ftype}_r(\texttt{f}, \texttt{t}_2) \qquad \mathit{ftype}_w(\texttt{f}, \neg\texttt{t}) = \mathit{ftype}_r(\texttt{f}, \neg\texttt{t}) = \bot$$

$$\frac{\texttt{class C} \lhd \texttt{D}\,\{\,\overline{\texttt{t}\,\texttt{f}};\ \overline{\texttt{S}}\,\} \qquad \overline{\texttt{t}\,\eta\,\texttt{s}\,\{\,\texttt{e}\,\}} \in \overline{\texttt{S}} \qquad \mathit{stype}(\texttt{s}, \texttt{D}) = \{\{\overline{\eta'}\}\}}{\mathit{stype}(\texttt{s}, \texttt{C}) = \{\{\overline{\eta}, \overline{\eta'}\}\}} \qquad \frac{\texttt{class C} \lhd \texttt{D}\,\{\,\overline{\texttt{t}\,\texttt{f}};\ \overline{\texttt{S}}\,\} \qquad \texttt{s} \notin \overline{\texttt{S}}}{\mathit{stype}(\texttt{s}, \texttt{C}) = \mathit{stype}(\texttt{s}, \texttt{D})}$$

$$\mathit{stype}(\texttt{s}, \texttt{t}_1 \wedge \texttt{t}_2) = m(\mathit{stype}, \texttt{s}, \texttt{t}_1, \texttt{t}_2) \qquad \mathit{stype}(\texttt{s}, \texttt{t}_1 \vee \texttt{t}_2) = \mathit{stype}(\texttt{s}, \texttt{t}_1) \cup \mathit{stype}(\texttt{s}, \texttt{t}_2) \qquad \mathit{stype}(\texttt{s}, \neg\texttt{t}) = \bot$$

$$\frac{\texttt{class C} \lhd \texttt{D}\,\{\,\overline{\texttt{t}\,\texttt{f}};\ \overline{\texttt{S}}\,\} \qquad \texttt{t}\,\eta\,\texttt{s}\,\{\,\texttt{e}\,\} \in \overline{\texttt{S}}}{\mathit{rtype}(\texttt{s}, \eta, \texttt{C}) = \texttt{t}} \qquad \frac{\texttt{class C} \lhd \texttt{D}\,\{\,\overline{\texttt{t}\,\texttt{f}};\ \overline{\texttt{S}}\,\} \qquad \texttt{t}\,\eta\,\texttt{s}\,\{\,\texttt{e}\,\} \notin \overline{\texttt{S}}}{\mathit{rtype}(\texttt{s}, \eta, \texttt{C}) = \mathit{rtype}(\texttt{s}, \eta, \texttt{D})}$$

$$\mathit{rtype}(\texttt{s}, \eta, \texttt{t}_1 \wedge \texttt{t}_2) = m(\mathit{rtype}, \texttt{s}, \eta, \texttt{t}_1, \texttt{t}_2) \qquad \mathit{rtype}(\texttt{s}, \eta, \texttt{t}_1 \vee \texttt{t}_2) = \mathit{rtype}(\texttt{s}, \eta, \texttt{t}_1) \vee \mathit{rtype}(\texttt{s}, \eta, \texttt{t}_2) \qquad \mathit{rtype}(\texttt{s}, \eta, \neg\texttt{t}) = \bot$$

$$\frac{\texttt{class C} \lhd \texttt{D}\,\{\,\overline{\texttt{t}\,\texttt{f}};\ \overline{\texttt{S}}\,\} \qquad \texttt{t}\,\eta\,\texttt{s}\,\{\,\texttt{e}\,\} \in \overline{\texttt{S}}}{\mathit{sbody}(\texttt{s}, \eta, \texttt{C}) = \texttt{e}} \qquad \frac{\texttt{class C} \lhd \texttt{D}\,\{\,\overline{\texttt{t}\,\texttt{f}};\ \overline{\texttt{S}}\,\} \qquad \texttt{s} \notin \overline{\texttt{S}}}{\mathit{sbody}(\texttt{s}, \eta, \texttt{C}) = \mathit{sbody}(\texttt{s}, \eta, \texttt{D})}$$

Table 8. OO: Lookup Functions.

$$\textsc{Par-R}\ \frac{\texttt{e}, h \longrightarrow P, h'}{\texttt{e} \mid\mid P_1, h \longrightarrow P \mid\mid P_1, h'} \qquad \textsc{Seq-R}\ \mathscr{E}[\texttt{o}; \texttt{e}], h \longrightarrow \mathscr{E}[\texttt{e}], h \qquad \textsc{Fld-R}\ \frac{h(\texttt{o}) = (\texttt{C}, \overline{\texttt{f} : \texttt{o}})}{\mathscr{E}[\texttt{o.f}_i], h \longrightarrow \mathscr{E}[\texttt{o}_i], h}$$

$$\textsc{NewC-R}\ \frac{\mathit{fields}(\texttt{C}) = \overline{\texttt{t}\,\texttt{f}} \qquad \texttt{o} \notin h}{\mathscr{E}[\texttt{new C}(\overline{\texttt{o}})], h \longrightarrow \mathscr{E}[\texttt{o}], h :: [\texttt{o} \mapsto (\texttt{C}, \overline{\texttt{f} : \texttt{o}})]} \qquad \textsc{FldAss-R}\ \mathscr{E}[\texttt{o.f} := \texttt{o}'], h \longrightarrow \mathscr{E}[\texttt{o}'], h[\texttt{o} \mapsto h(\texttt{o})[\texttt{f} \mapsto \texttt{o}']]$$

$$\textsc{SessReq-R}\ \frac{h(\texttt{o}) = (\texttt{C}, _) \qquad \eta \bowtie \eta' \qquad \mathit{sbody}(\texttt{s}, \eta', \texttt{C}) = \texttt{e}' \qquad \texttt{k}, \tilde{\texttt{k}} \notin h}{\mathscr{E}[\texttt{o.s}\,\{\texttt{e} : \eta\}], h \longrightarrow \mathscr{E}[\texttt{e}\,\wr\,\texttt{k}\rceil] \mid\mid [\%\texttt{this}]\texttt{e}'\,\wr\,\tilde{\texttt{k}}\rceil, h[\texttt{k}, \tilde{\texttt{k}} \mapsto ()]} \qquad \textsc{SessDel-R}\ \frac{h(\texttt{o}) = (\texttt{C}, _) \qquad \chi \leq \eta \qquad \mathit{sbody}(\texttt{s}, \eta, \texttt{C}) = \texttt{e}}{\mathscr{E}[\texttt{o} \bullet \texttt{s}\,\{\texttt{k} : \chi\}], h \longrightarrow \mathscr{E}[[\%\texttt{this}]\texttt{e}\,\wr\,\texttt{k}\rceil], h}$$

$$\textsc{SendCase-R}\ \frac{h(\tilde{\texttt{k}}) = \overline{\texttt{o}} \qquad h(\texttt{o}) = (\texttt{C}, _) \qquad \texttt{C} \leq \texttt{t}_i}{\mathscr{E}[\texttt{k.sendC(o)}\{\texttt{t}_1 \Rightarrow \texttt{e}_1 \,[]\, \texttt{t}_2 \Rightarrow \texttt{e}_2\}], h \longrightarrow \mathscr{E}[\texttt{e}_i], h[\tilde{\texttt{k}} \mapsto \overline{\texttt{o}} :: \texttt{o}]}$$

$$\textsc{ReceiveCase-R}\ \frac{h(\texttt{k}) = \texttt{o} :: \overline{\texttt{o}} \qquad h(\texttt{o}) = (\texttt{C}, _) \qquad \texttt{C} \leq \texttt{t}_i}{\mathscr{E}[\texttt{k.recC(x)}\{\texttt{t}_1 \Rightarrow \texttt{e}_1 \,[]\, \texttt{t}_2 \Rightarrow \texttt{e}_2\}], h \longrightarrow \mathscr{E}[[\%\texttt{x}]\texttt{e}_i], h[\texttt{k} \mapsto \overline{\texttt{o}}]}$$

Table 10. OO: Reduction Rules.

following syntax for session descriptors:

$$\eta \quad ::= \quad \varepsilon \ \mid\ !\texttt{t}.\eta \oplus !\texttt{t}.\eta \ \mid\ ?\texttt{t}.\eta + ?\texttt{t}.\eta$$

and we define:

$$\eta \circ \eta' = \begin{cases} \eta' & \text{if } \eta = \varepsilon, \\ \alpha.(\eta'' \circ \eta') & \text{if } \eta = \alpha.\eta'' \end{cases}$$

where $\alpha \in \{!\texttt{t}, ?\texttt{t}\}$. The descriptor ε is dual of itself.

We first define typing for user expressions, in which communication channels are implicit. For technical reasons it is useful to consider also expressions with occurrences of object identifiers, which are not directly expressible in user syntax. We call these expressions *channel free expressions*. The term environments therefore will contain also type assignments to object identifiers. This permits a simpler formulation of the runtime typing rules.

The typing judgement has the shape

$$\Gamma \vdash \texttt{e} : \texttt{t} \fatsemi \eta$$

where Γ is a term environment, which maps `this`, variables and objects to types, and η represents the session descriptor of the (implicit) active channel.

Typing rules for channel free expressions are in Table 11. For the sake of simplicity in rule NewC-T we require that the initialisation of an object does not involve communications. Notice that in rule Seq-T we use concatenation to represent that first the communications in $\texttt{e}_1$ and then those in $\texttt{e}_2$ are performed.

$$\textsc{Axiom-T}\quad \Gamma \vdash \mathtt{z} : \Gamma(\mathtt{z}) \fatsemi \varepsilon$$

$$\textsc{Sub-T}\quad \frac{\Gamma \vdash \mathtt{e} : \mathtt{t} \fatsemi \eta \qquad \mathtt{t} \leq \mathtt{t}'}{\Gamma \vdash \mathtt{e} : \mathtt{t}' \fatsemi \eta}$$

$$\textsc{NewC-T}\quad \frac{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{t}\,\mathtt{f}} \qquad \Gamma \vdash \mathtt{e}_i : \mathtt{t}_i \fatsemi \varepsilon}{\Gamma \vdash \mathtt{new\,C}(\overline{\mathtt{e}}) : \mathtt{C} \fatsemi \varepsilon}$$

$$\textsc{Fld-T}\quad \frac{\Gamma \vdash \mathtt{e} : \mathtt{t} \fatsemi \eta}{\Gamma \vdash \mathtt{e.f} : \mathit{ftype}_r(\mathtt{f},\mathtt{t}) \fatsemi \eta}$$

$$\textsc{Seq-T}\quad \frac{\Gamma \vdash \mathtt{e} : \mathtt{t} \fatsemi \eta \qquad \Gamma \vdash \mathtt{e}' : \mathtt{t}' \fatsemi \eta'}{\Gamma \vdash \mathtt{e;e}' : \mathtt{t}' \fatsemi \eta \circ \eta'}$$

$$\textsc{FldAss-T}\quad \frac{\Gamma \vdash \mathtt{e} : \mathtt{t} \fatsemi \eta \qquad \Gamma \vdash \mathtt{e}' : \mathit{ftype}_w(\mathtt{f},\mathtt{t}) \fatsemi \eta'}{\Gamma \vdash \mathtt{e.f} := \mathtt{e}' : \mathit{ftype}_w(\mathtt{f},\mathtt{t}) \fatsemi \eta \circ \eta'}$$

$$\textsc{SessReq-T}\quad \frac{\Gamma \vdash \mathtt{e} : \mathtt{t} \fatsemi \eta \qquad \Gamma \vdash \mathtt{e}' : \mathtt{t}' \fatsemi \eta' \qquad \forall \{\overline{\eta}\} \in \mathit{stype}(\mathtt{s},\mathtt{t}) \exists \eta'' \in \overline{\eta}.\eta' \bowtie \eta''}{\Gamma \vdash \mathtt{e.s}\,\{\mathtt{e}' : \eta'\} : \mathtt{t}' \fatsemi \eta}$$

$$\textsc{SessDel-T}\quad \frac{\Gamma \vdash \mathtt{e} : \mathtt{t} \fatsemi \eta \qquad \eta' \leq \chi \qquad \forall \{\overline{\eta}\} \in \mathit{stype}(\mathtt{s},\mathtt{t}) \exists \eta'' \in \overline{\eta}.\chi \leq \eta'' \qquad \{\overline{\eta}\} \in \mathit{stype}(\mathtt{s},\mathtt{t}) \,\&\, \eta'' \in \overline{\eta} \,\&\, \chi \leq \eta'' \implies \mathit{rtype}(\mathtt{s},\eta'',\mathtt{t}) = \mathtt{t}'}{\Gamma \vdash \mathtt{e} \bullet \mathtt{s}\,\{\chi\} : \mathtt{t}' \fatsemi \eta \circ \eta'}$$

$$\textsc{SendC-T}\quad \frac{\Gamma \vdash \mathtt{e} : \mathtt{t}_1 \vee \mathtt{t}_2 \fatsemi \eta \qquad \Gamma \vdash \mathtt{e}_i : \mathtt{t} \fatsemi \eta_i}{\Gamma \vdash \mathtt{sendC(e)}\{\mathtt{t}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{t}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{t} \fatsemi \eta \circ (!\mathtt{t}_1.\eta_1 \oplus !\mathtt{t}_2.\eta_2)}$$

$$\textsc{ReceiveC-T}\quad \frac{\Gamma, \mathtt{x} : \mathtt{t}_i \vdash \mathtt{e}_i : \mathtt{t} \fatsemi \eta_i}{\Gamma \vdash \mathtt{recC(x)}\{\mathtt{t}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{t}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{t} \fatsemi ?\mathtt{t}_1.\eta_1 + ?\mathtt{t}_2.\eta_2}$$

Table 11. OO: Typing Rules for Channel Free Expressions.

$$\textsc{Axiom-RT}\quad \Gamma \models_{\mathtt{r}} \mathtt{z} : \Gamma(\mathtt{z}) \fatsemi \emptyset$$

$$\textsc{Sub-RT}\quad \frac{\Gamma \models_{\mathtt{r}} \mathtt{e} : \mathtt{t} \fatsemi \Theta \qquad \mathtt{t} \leq \mathtt{t}'}{\Gamma \models_{\mathtt{r}} \mathtt{e} : \mathtt{t}' \fatsemi \Theta}$$

$$\textsc{NewC-RT}\quad \frac{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{t}\,\mathtt{f}} \qquad \Gamma \models_{\mathtt{r}} \mathtt{e}_i : \mathtt{t}_i \fatsemi \emptyset}{\Gamma \models_{\mathtt{r}} \mathtt{new\,C}(\overline{\mathtt{e}}) : \mathtt{C} \fatsemi \emptyset}$$

$$\textsc{Fld-RT}\quad \frac{\Gamma \models_{\mathtt{r}} \mathtt{e} : \mathtt{t} \fatsemi \Theta}{\Gamma \models_{\mathtt{r}} \mathtt{e.f} : \mathit{ftype}_r(\mathtt{f},\mathtt{t}) \fatsemi \Theta}$$

$$\textsc{Seq-RT}\quad \frac{\Gamma \models_{\mathtt{r}} \mathtt{e} : \mathtt{t} \fatsemi \Theta \qquad \Gamma \models_{\mathtt{r}} \mathtt{e}' : \mathtt{t}' \fatsemi \Theta'}{\Gamma \models_{\mathtt{r}} \mathtt{e;e}' : \mathtt{t}' \fatsemi \Theta \circ \Theta'}$$

$$\textsc{FldAss-RT}\quad \frac{\Gamma \models_{\mathtt{r}} \mathtt{e} : \mathtt{t} \fatsemi \Theta \qquad \Gamma \models_{\mathtt{r}} \mathtt{e}' : \mathit{ftype}_w(\mathtt{f},\mathtt{t}) \fatsemi \Theta'}{\Gamma \models_{\mathtt{r}} \mathtt{e.f} := \mathtt{e}' : \mathit{ftype}_w(\mathtt{f},\mathtt{t}) \fatsemi \Theta \circ \Theta'}$$

$$\textsc{SessReq-RT}\quad \frac{\Gamma \models_{\mathtt{r}} \mathtt{e} : \mathtt{t} \fatsemi \Theta \qquad \Gamma \vdash \mathtt{e}' : \mathtt{t}' \fatsemi \eta \qquad \forall \{\overline{\eta}\} \in \mathit{stype}(\mathtt{s},\mathtt{t}) \exists \eta' \in \overline{\eta}.\eta \bowtie \eta'}{\Gamma \models_{\mathtt{r}} \mathtt{e.s}\,\{\mathtt{e}' : \eta\} : \mathtt{t}' \fatsemi \Theta}$$

$$\textsc{SessDel-RT}\quad \frac{\Gamma \models_{\mathtt{r}} \mathtt{e} : \mathtt{t} \fatsemi \Theta \qquad \eta \leq \chi \qquad \forall \{\overline{\eta}\} \in \mathit{stype}(\mathtt{s},\mathtt{t}) \exists \eta' \in \overline{\eta}.\chi \leq \eta' \qquad \{\overline{\eta}\} \in \mathit{stype}(\mathtt{s},\mathtt{t}) \,\&\, \eta' \in \overline{\eta} \,\&\, \chi \leq \eta' \implies \mathit{rtype}(\mathtt{s},\eta',\mathtt{t}) = \mathtt{t}'}{\Gamma \models_{\mathtt{r}} \mathtt{e} \bullet \mathtt{s}\,\{\mathtt{k} : \chi\} : \mathtt{t}' \fatsemi \Theta \circ \{\mathtt{k} : \eta\}}$$

$$\textsc{SendC-RT}\quad \frac{\Gamma \models_{\mathtt{r}} \mathtt{e} : \mathtt{t}_1 \vee \mathtt{t}_2 \fatsemi \Theta \qquad \Gamma \models_{\mathtt{r}} \mathtt{e}_i : \mathtt{t} \fatsemi \{\mathtt{k} : \eta_i\}}{\Gamma \models_{\mathtt{r}} \mathtt{k.sendC(e)}\{\mathtt{t}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{t}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{t} \fatsemi \Theta \circ \{\mathtt{k} : !\mathtt{t}_1.\eta_1 \oplus !\mathtt{t}_2.\eta_2\}}$$

$$\textsc{ReceiveC-RT}\quad \frac{\Gamma, \mathtt{x} : \mathtt{t}_i \models_{\mathtt{r}} \mathtt{e}_i : \mathtt{t} \fatsemi \{\mathtt{k} : \eta_i\}}{\Gamma \models_{\mathtt{r}} \mathtt{k.recC(x)}\{\mathtt{t}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{t}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{t} \fatsemi \{\mathtt{k} : ?\mathtt{t}_1.\eta_1 + ?\mathtt{t}_2.\eta_2\}}$$

Table 13. OO: Typing Rules for Runtime Expressions.

The rule for session request SESSREQ-T prescribes that in all sets of session descriptors which build $\mathit{stype}(\mathtt{s},\mathtt{t})$ there is a session descriptor which is dual of the session descriptor of the the cobody $\mathtt{e}'$. This assures that there is always a possible choice for the body of the session $\mathtt{s}$ which will communicate properly with the cobody. In typing session delegation (rule SESSDEL-T) we take into account that the whole expression will be replaced by one of the session bodies defined in the class of the expression to which the session is delegated (cf. the reduction rule SESSDEL-R, Table 10). The condition

$$\chi \leq \eta' \text{ implies } \mathit{rtype}(\mathtt{s},\eta',\mathtt{t}) = \mathtt{t}'$$

assures that all bodies of $\mathtt{s}$ which can be chosen have the same return type.

Table 12 defines well-formed class tables. Rule SESS-WF type checks the session bodies with respect to the current class $\mathtt{C}$ taking as term environment the association between $\mathtt{this}$ and $\mathtt{C}$.

During evaluation of well-typed programs, channel names are made explicit in send and receive expressions, as well as in session delegation. Thus, in order to show how well-typedness is preserved under evaluation, we need to define new typing rules for runtime expressions. Furthermore, in typing runtime expressions, we must take into account the session types of more than one channel: runtime expressions contain explicit channel names (used for communication) thus session types must be associated with channel names in an appropriate way. Then judgements have the form

$$\Gamma \models_{\mathtt{r}} \mathtt{e} : \mathtt{t} \fatsemi \Theta$$

where Θ denotes a *session environment* which maps channels to session descriptors.

A session environment maps only a finite set of channels to session descriptors different from ε, and all the remaining to ε. We can then represent one session environment with an infinite number of finite sets which give all the meaningful associations and some of the others. For example $\{\mathsf{k} : \eta\}$ and $\{\mathsf{k} : \eta, \mathsf{k}' : \varepsilon\}$ represent the same environment. This choice avoids an explicit weakening rule for session environments. Table 13 gives the typing rules for runtime expressions, which differ from those for channel free expressions for having session environments instead of a unique session descriptor. For this reason we extend the *concatenation* of session descriptors to *session environments* as follows:

$$\Theta \circ \Theta'(\mathsf{k}) = \Theta(\mathsf{k}) \circ \Theta'(\mathsf{k}).$$

Notice that in rule SESSREQ-RT we are making use of the judgement $\Gamma \vdash \mathsf{e}' : \mathtt{t} \fatsemi \eta'$, where the expression e' does not contain channels, but it can contain object identifiers. This justifies our choice of considering channel free expressions instead of user expressions in the typing rules of previous subsection. Notice also that the session environments of the branches in the communication expressions only contain the current channel as subject, since these expressions will never be reduced before the selection has been done.

Our type system enjoys subject reduction and assures progress ($\longrightarrow^*$ is the reflexive and transitive closure or $\longrightarrow$):

THEOREM 3.7 (Subject Reduction and Progress). *If* $\emptyset \vdash \mathsf{e}_0 : \mathtt{t}_0 \fatsemi \varepsilon$ *and* $\mathsf{e}_0, [\,] \longrightarrow^* P, h$*, where* $P \equiv \mathsf{e}_1 \mid\mid \ldots \mid\mid \mathsf{e}_n$*, then:*

1. *for each* e_i *we get* $\Gamma \vdash_{\mathtt{r}} \mathsf{e}_i : \mathtt{t}_i \fatsemi \Theta_i$ *for some* $\Gamma, \mathtt{t}_i, \Theta_i$ $(1 \leq i \leq n)$*, and there is* j $(1 \leq j \leq n)$ *such that* $\mathtt{t}_j = \mathtt{t}_0$*, and;*
2. *either* $P, h \longrightarrow P', h'$ *for some* P', h'*, or for all* i $(1 \leq i \leq n)$ e_i *is an object identifier.*

The proof of this theorem is the content of Appendix C.

Notice that evaluation of closed expressions stops returning a parallel of object identifiers and that an object identifier can be typed, using the type system for channel free expressions, from the context Γ which defines the type assumption for that identifier.

The runtime errors which our type system prevents are:

1. the selection of a field and the request of a session which do not belong to the class of the current object;
2. the creation of a pair of dual channels whose communication sequences do not perfectly match.

4. Conclusions and related work

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.

We claim that our theory of session types is minimal—insofar as the addition of any further constructor such as parallel composition or labeled synchronization would restrict the programming language to which the framework could be applied—and yet complete. The key ingredients of our theory are communication primitives, behavioral composition operators for describing branching points, and boolean composition operators for types and sieves. All the existing proposal of session types of comparable expressiveness can be obtained by using suitable combinations of these ingredients and our session types can be used for typing generic π-calculus processes without any dedicated primitive for session management.

The relation of our work with some others defining theories of session types is already explored in Section 3. Subtyping relations for session types are studied in [20, 19]. In these works the definition of the subtyping relation is driven by the observation that one can safely replace a session by another that externally offers more choices and internally can make less choices. 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. Here we work with external choices that are driven by the messages being exchanged rather than by labels: when a session offers an output it will be able to synchronize only with the branches of a choice that accept that output. This policy was first considered in [13] for an object-oriented calculus. The resulting subtyping relation generalizes the safe substitutability principle of the existing settings by permitting (a combination of) branches of a choice to subsume another set of branches. Furthermore, while in the cited works the subtyping relation is defined coinductively and axiomatically, we characterize the relation semantically, and this captures directly the desired safety property.

Our approach to session type semantics is akin to the testing approach to process semantics [11]: 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).

Moving from label-driven to type-driven branch selection 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 [16], 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.

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 [9] for asynchronous CCS, where outputs cannot be blocked even if they guard external choices (we called 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 consecutive outputs is left for future work). Technically, this corresponds to observing inputs even in the presence of (partial) asynchrony.

From a technical viewpoint in this work we introduce several novelties. We devise a new labeled transition system *for session descriptors* in which actions represent values rather than types, 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. A labeled transition system for session types is also proposed in [1], where a type system ensuring progress for the CaSPiS calculus [4] is designed.

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] L. Bettini, S. Capecchi, M. Dezani-Ciancaglini, E. Giachino, and B. Venneri. Session and Union Types for Object Oriented Programming. In *Concurrency, Graphs and Models*, volume 5065 of *LNCS*, pages 659–680. Springer, 2008.

[3] E. Bonelli, A. Compagnoni, and E. Gunter. Correspondence Asser-

tions for Process Synchronization in Concurrent Communications. *J. Funct. Progr.*, 15(2):219–248, 2005.

[4] M. Boreale, R. Bruni, R. D. Nicola, and M. Loreti. Sessions and pipelines for structured service programming. In *FMOODS'08*, volume 5051 of *LNCS*, pages 19–38. Springer, 2008.

[5] L. Caires and H. T. Vieira. Conversation Types. In *ESOP'09*, 2009. To appear.

[6] S. Capecchi, M. Coppo, M. Dezani-Ciancaglini, S. Drossopoulou, and E. Giachino. Amalgamating Sessions and Methods in Object Oriented Languages with Generics. *Theoretical Computer Science*, 2008. to appear.

[7] 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.

[8] G. Castagna, N. Gesbert, and L. Padovani. A theory of contracts for web services. Extended version of the article included in *POPL '08*, submitted, available on authors' web pages, 2008.

[9] I. Castellani and M. Hennessy. Testing theories for asynchronous languages. In *FST&TCS '98*, volume 1350 of *LNCS*, pages 90–101. Springer, 1998.

[10] M. Coppo, M. Dezani-Ciancaglini, and N. Yoshida. Asynchronous Session Types and Progress for Object-Oriented Languages. In M. Bonsangue and E. B. Johnsen, editors, *FMOODS'07*, volume 4468 of *LNCS*, pages 1–31. Springer-Verlag, 2007.

[11] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theor. Comput. Sci*, 34:83–133, 1984.

[12] R. De Nicola and M. Hennessy. CCS without τ's. In *TAPSOFT/-CAAP'87*, volume 249 of *LNCS*, pages 138–152. Springer, 1987.

[13] 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.

[14] M. Dezani-Ciancaglini, D. Mostrous, N. Yoshida, and S. Drossopoulou. Session Types for Object-Oriented Languages. In D. Thomas, editor, *ECOOP'06*, volume 4067 of *LNCS*, pages 328–352. Springer-Verlag, 2006.

[15] 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.

[16] A. Frisch. Regular tree language recognition with static information. In *IFIP TCS*, pages 661–674. Kluwer, 2004.

[17] 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.

[18] P. Garralda, A. Compagnoni, and M. Dezani-Ciancaglini. BASS: Boxed Ambients with Safe Sessions. In *PPDP'06*, pages 61–72. ACM Press, 2006.

[19] S. Gay. Bounded polymorphism in session types. *MSCS*, 2008. To appear.

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

[21] K. Honda. Types for dyadic interaction. In *CONCUR'93*, volume 715 of *LNCS*, pages 509–523. Springer, 1993.

[22] 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.

[23] A. Igarashi and N. Kobayashi. A generic type system for the pi-calculus. *Theor. Comput. Sci*, 311(1-3):121–163, 2004.

[24] A. Igarashi, B. C. Pierce, and P. Wadler. Featherweight Java: a Minimal Core Calculus for Java and GJ. *ACM TOPLAS*, 23(3):396–450, 2001.

[25] N. Kobayashi. Type systems for concurrent programs. In *FMC'03*, LNCS 2757. Springer, 2003.

[26] N. Kobayashi. Type systems for concurrent programs. Extended version of [25], Tohoku University, 2007.

[27] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theor. Comput. Sci.*, 114(1):149–171, 1993.

[28] A. Vallecillo, V. T. Vasconcelos, and A. Ravara. Typing the Behavior of Objects and Components using Session Types. In A. Brogi and J.-M. Jacquet, editors, *FOCLASA'02*, volume 68(3) of *ENTCS*, pages 439–456. Elsevier, 2002.

[29] V. T. Vasconcelos, S. Gay, and A. Ravara. Typechecking a Multithreaded Functional Language with Session Types. *Theorical Computer Science*, 368(1-2):64–87, 2006.

[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. Proofs of Section 2

We will first prove Theorems 2.15 and 2.18 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.

A.1 Proof of Theorems 2.15 and 2.18

We prove Theorems 2.15 and 2.18 by simultaneous induction. More precisely we consider the weight defined in the proof of Theorem 2.6. 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 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 1) and $\chi \not\leq \chi'$ is easily decidable.

Inductive case. Let us now study the inductive case, and therefore suppose the two properties hold for sieves of 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 this order—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 $\bigwedge_{j\in J} \lambda_{ij} = \bot$. 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.

If $\chi \equiv \bigwedge_{j\in J} \lambda_j$ where at least one literal is not negated. The normal form is either $\bot$, 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 $\bot$. Since we cannot directly use the induction hypothesis (as we should apply it to the whole sieve),

[R-EX-SPLIT]
$$\frac{\sum_{i\in I} ?\psi_i.\eta_i\{+\mathsf{end}\} \quad\leq\quad \sum_{j\in J\setminus\{k\}} ?\psi'_j.\eta'_j + ?(\psi'_k \setminus \psi_h).\eta'_k + ?\psi_h.\eta'_k\{+\mathsf{end}\}}{\sum_{i\in I} ?\psi_i.\eta_i\{+\mathsf{end}\} \quad\leq\quad \sum_{j\in J} ?\psi'_j.\eta'_j\{+\mathsf{end}\}} \left(\begin{array}{l}\psi_h \sqsubseteq \psi'_k\\ \psi_h \neq \psi'_k\end{array}\right)$$

[L-EX-SPLIT]
$$\frac{\sum_{i\in I\setminus\{h\}} ?\psi_i.\eta_i + ?(\psi_h \setminus \psi'_k).\eta_h + ?\psi'_k.\eta_h\{+\mathsf{end}\} \quad\leq\quad \sum_{j\in J} ?\psi'_j.\eta'_j\{+\mathsf{end}\}}{\sum_{i\in I} ?\psi_i.\eta_i\{+\mathsf{end}\} \quad\leq\quad \sum_{j\in J} ?\psi'_j.\eta'_j\{+\mathsf{end}\}} \left(\begin{array}{l}\psi'_k \sqsubseteq \psi_h\\ \psi_h \neq \psi'_k\end{array}\right)$$

[LR-EX-SPLIT]
$$\frac{\sum_{i\in I\setminus\{h\}} ?\psi_i.\eta_i + ?(\psi_h \setminus \psi'_k).\eta_h + ?\psi'_k.\eta_h\{+\mathsf{end}\} \quad\leq\quad \sum_{j\in J\setminus\{k\}} ?\psi'_j.\eta'_j + ?(\psi'_k \setminus \psi_h).\eta'_k + ?\psi_h.\eta'_k\{+\mathsf{end}\}}{\sum_{i\in I} ?\psi_i.\eta_i\{+\mathsf{end}\} \quad\leq\quad \sum_{j\in J} ?\psi'_j.\eta'_j\{+\mathsf{end}\}} \left(\begin{array}{l}\psi'_k \wedge \psi_h \neq \varnothing\\ \psi'_k \wedge \psi_h \neq \psi'_k\\ \psi'_k \wedge \psi_h \neq \psi_h\end{array}\right)$$

[R-IN-SPLIT]
$$\frac{\bigoplus_{i\in I} !\psi_i.\eta_i\{\oplus\mathsf{end}\} \quad\leq\quad \bigoplus_{j\in J\setminus\{k\}} !\psi'_j.\eta'_j \oplus !(\psi'_k \setminus \psi_h).\eta'_k \oplus !\psi_h.\eta'_k\{\oplus\mathsf{end}\}}{\bigoplus_{i\in I} !\psi_i.\eta_i\{\oplus\mathsf{end}\} \quad\leq\quad \bigoplus_{j\in J} !\psi'_j.\eta'_j\{\oplus\mathsf{end}\}} \left(\begin{array}{l}\psi_h \sqsubseteq \psi'_k\\ \psi_h \neq \psi'_k\end{array}\right)$$

[L-IN-SPLIT]
$$\frac{\bigoplus_{i\in I\setminus\{h\}} !\psi_i.\eta_i \oplus !(\psi_h \setminus \psi'_k).\eta_h \oplus !\psi'_k.\eta_h\{\oplus\mathsf{end}\} \quad\leq\quad \bigoplus_{j\in J} !\psi'_j.\eta'_j\{\oplus\mathsf{end}\}}{\bigoplus_{i\in I} !\psi_i.\eta_i\{\oplus\mathsf{end}\} \quad\leq\quad \bigoplus_{j\in J} !\psi'_j.\eta'_j\{\oplus\mathsf{end}\}} \left(\begin{array}{l}\psi'_k \sqsubseteq \psi_h\\ \psi_h \neq \psi'_k\end{array}\right)$$

[LR-IN-SPLIT]
$$\frac{\bigoplus_{i\in I\setminus\{h\}} !\psi_i.\eta_i \oplus !(\psi_h \setminus \psi'_k).\eta_h \oplus !\psi'_k.\eta_h\{\oplus\mathsf{end}\} \quad\leq\quad \bigoplus_{j\in J\setminus\{k\}} !\psi'_j.\eta'_j \oplus !(\psi'_k \setminus \psi_h).\eta'_k \oplus !\psi_h.\eta'_k\{\oplus\mathsf{end}\}}{\bigoplus_{i\in I} !\psi_i.\eta_i\{\oplus\mathsf{end}\} \quad\leq\quad \bigoplus_{j\in J} !\psi'_j.\eta'_j\{\oplus\mathsf{end}\}} \left(\begin{array}{l}\psi'_k \wedge \psi_h \neq \varnothing\\ \psi'_k \wedge \psi_h \neq \psi'_k\\ \psi'_k \wedge \psi_h \neq \psi_h\end{array}\right)$$

Table 14. 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.

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 $\chi \not\leq \bot$ 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 $\chi \not\leq \bot$, that is whether $\neg \bigvee_{j\in J} \eta_j \not\leq \bot$. This is always true for Lemma 2.16, 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 $\varnothing$ (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 $\psi = \varnothing$ 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. I.e. we obtain a descriptor of the shape:
   $$\bigoplus_{J\in K} (\sum_{j\in J} ?\psi_j.\eta_j\{+\mathsf{end}\}) \oplus \bigoplus_{h\in H} !\psi_h.\eta_h\{\oplus\mathsf{end}\} \qquad (12)$$
4. we have the following cases:
   (a) $K = \emptyset$.
   (b) $H = \emptyset$. Then (12) is equivalent to:
   $$\sum_{j_1\in J_1} \cdots \sum_{j_k\in J_k} ?(\psi_{j_1} \wedge \cdots \wedge \psi_{j_k}).(\eta_{j_1} \oplus \cdots \oplus \eta_{j_k})\{+\mathsf{end}\} \qquad (13)$$

where $K = \{J_1, \ldots, J_k\}$ and $\psi_{j_1} \wedge \cdots \wedge \psi_{j_k} \neq \varnothing$ 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

$$\eta = \bigoplus_{h \in H} !\psi_h.\eta_h \oplus \mathsf{end} \tag{14}$$

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

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 $\psi \wedge \psi' = \varnothing$ by applying rules (O4), (O5), and (I5).
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 $\psi \wedge \psi' = \varnothing$ 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 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 $\bot$.

Let us pass to the proof of **point 2**, that is show that it is possible to decide whether $\chi \not\leq \chi'$ 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 $\bigvee_{i \in I} \bigwedge_{j \in J} \lambda_{ij} \not\leq \bigvee_{h \in H} \bigwedge_{k \in K} \lambda'_{hk}$ 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 $\bigwedge_{j \in J} \lambda_{ij} \not\leq \bigvee_{h \in H} \bigwedge_{k \in K} \lambda'_{hk}$, which can be checked by induction hypothesis.

Case $\bigwedge_{j \in J} \lambda_j \not\leq \bigvee_{h \in H} \bigwedge_{k \in K} \lambda'_{hk}$ such that $|H| > 1$. By applying classical set-theoretic distribution laws, this problem can be reduced to $\bigwedge_{j \in J} \lambda_j \not\leq \bigwedge_{h \in H} \bigvee_{k \in K} \lambda'_{hk}$. We can now split the intersection on the right and reduce it to check whether there exists $h \in H$ such that $\bigwedge_{j \in J} \lambda_j \not\leq \bigvee_{k \in K} \lambda'_{hk}$. The result follows by induction hypothesis.

Case $\bigwedge_{j \in J} \lambda_j \not\leq \bigvee_{k \in K} \lambda'_k$. Let us highlight negative and positive literals:

$$\bigwedge_{p \in P_J} \theta_p \wedge \bigwedge_{n \in N_J} \neg\eta_n \not\leq \bigwedge_{p \in P_K} \theta_p \wedge \bigwedge_{n \in N_K} \neg\eta_n.$$

By simple set-theoretic manipulations this is equivalent to check whether $\bigwedge_{p \in P_J \cup P_K} \theta_p \not\leq \bigvee_{n \in N_J \cup N_K} \eta_n$. Since intersection is equivalent to internal choice this is reduced to checking whether $\bigoplus_{p \in P_J \cup P_K} \theta_p \not\leq \bigvee_{n \in N_J \cup N_K} \eta_n$ which by strong disjunction is equivalent to prove that $\bigoplus_{p \in P_J \cup P_K} \theta_p \not\leq \eta_n$ holds for all $n \in N_J \cup N_K$. The result follows by induction hypothesis.

Case $\eta \not\leq \eta'$. This is the last remaining case and also the most difficult one. We can feed η and η' to the algorithmic rules of Table 2. 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 soundness, let us prove the point by induction on the Noetherian measure defined in Section 2.1. More precisely for each rule of Tables 2 and 14 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 $!\psi.\eta''$, 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 may increase. We are in the case where we have deduced $\alpha.\theta \leq \alpha.\theta'$ 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 2.17.

Completeness. For completeness let us prove this point by induction on the Noetherian measure defined in Section 2.1. Suppose that the result holds for descriptors of smaller measure 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 since the types are 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 $j \in J$, $\eta_i \not\leqq \eta_j$. 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 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 $\overline{\eta} \bowtie \eta$. Regularity of $\overline{\eta}$ is a direct consequence of the regularity of η.

Assume $\eta \equiv \sum_{i\in I} ?\psi_i.\eta_i\{+\mathsf{end}\}$, where the end subterm may be missing. Then $\eta^{\bowtie}$ 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 $\overline{\eta} = \mathsf{end}$. 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 $\overline{\eta} = !\psi_k.\overline{\eta_k}$.

Assume $\eta \equiv \bigoplus_{i\in I} !\psi_i.\eta_i\{\oplus\mathsf{end}\}$, where the end subterm may be missing. Then $\eta^{\bowtie}$ 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 $\eta_i^{\bowtie}$ since $\eta\langle\mu\rangle = \eta_i$. We conclude by taking $\overline{\eta} = \sum_{i\in I} ?\psi_i.\overline{\eta_i} + \mathsf{end}$.

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

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

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

is a coinductive viability. Let $\eta \in \mathscr{R}$. 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 $\mathsf{end} \in \mathscr{R}$ by definition of $\mathscr{R}$).

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 $\mathscr{R}$ we have $\eta\langle\mu\rangle \in \mathscr{R}$, hence condition (1) of Definition 2.9 is satisfied.

A.3 Proof of Theorem 2.12

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

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

is a duality relation, since $(\eta, \eta_2) \in \mathscr{C}$ by definition of $\mathscr{C}$. Let $(\eta', \eta_2') \in \mathscr{C}$. Then there exists η_1' such that $\eta' \bowtie \eta_1'$ and $\eta_1' \leqq \eta_2'$. 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 $\eta_1' \leqq \eta_2'$ 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 $\eta_1' \Downarrow \overline{\mu}$ and $\eta'\langle\mu\rangle \bowtie \eta_1'\langle\mu\rangle$. From $\eta_1' \leqq \eta_2'$ we derive $\eta_2' \Downarrow \mu$ and $\eta_1'\langle\mu\rangle \leqq \eta_2'\langle\mu\rangle$ hence $(\eta'\langle\mu\rangle, \eta_2'\langle\overline{\mu}\rangle) \in \mathscr{C}$ by definition of $\mathscr{C}$ and we conclude by condition (2) of Definition 2.5.

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

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

is a coinductive subsession. Let $(\eta, \eta') \in \mathscr{R}$ and assume $\eta^{\bowtie}$ 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 $\mathscr{R}$ we deduce that $\theta \bowtie \eta'$, hence $\eta'^{\bowtie}$ again by Proposition 2.10. We reason by cases on the structure of η and η'.

Assume $\eta \equiv \sum_{i\in I} ?\psi_i.\eta_i\{+\mathsf{end}\}$ and $\eta' \equiv \sum_{j\in J} ?\psi_j.\eta_j\{+\mathsf{end}\}$. Condition (1) of Definition 2.11 is trivially satisfied since $\eta' \not\Downarrow$. 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 $\mathsf{end} \bowtie \eta$ and $\mathsf{end} \not\bowtie \eta'$ which is absurd by definition of $\mathscr{R}$. If $\mu \neq \checkmark$, then there exists $i \in I$ such that $?\psi_i.\eta_i \Downarrow \mu$ and $\eta_i^{\bowtie}$. 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 $\eta_i^{\bowtie}$ 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 $(\eta_i, \eta_j) \in \mathscr{R}$ 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 $\eta \not\Downarrow$.

Assume $\eta \equiv \bigoplus_{i\in I} !\psi_i.\eta_i\{\oplus\mathsf{end}\}$ and $\eta' \equiv \bigoplus_{j\in J} !\psi_j.\eta_j\{\oplus\mathsf{end}\}$ and $\eta' \not\Downarrow$. 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 $\eta^{\bowtie}$). Let $\theta \equiv \sum_{i\in I} ?\psi_i.\theta_i\{+\mathsf{end}\}$ 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 $(\eta_i, \eta_j) \in \mathscr{R}$ 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 $\eta' \not\Downarrow$.

Assume $\eta \equiv \bigoplus_{i\in I} !\psi_i.\eta_i\{\oplus\mathsf{end}\}$ and $\eta' \equiv \sum_{j\in J} ?\psi_j.\eta_j\{+\mathsf{end}\}$ and $\eta \not\Downarrow$. Condition (1) and (2) of Definition 2.11 are trivially satisfied since $\eta \not\Downarrow$ and $\eta' \not\Downarrow$. 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 $\eta^{\bowtie}$). Let $\theta \equiv \sum_{i\in I} ?\psi_i.\theta_i\{+\mathsf{end}\}$ 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 3.2

B.1 Proof of Subject Reduction

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

The *core* of a session stack ($\mathsf{core}(\Delta)$) 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).

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

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

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

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

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

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

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

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

PROPOSITION B.5. *1. If $\Delta \preccurlyeq \Delta'$, then $\mathsf{set}(\Delta) \preccurlyeq \mathsf{set}(\Delta')$.*
2. If $\Lambda_1 \preccurlyeq \Lambda_1'$ and $\Lambda_2 \preccurlyeq \Lambda_2'$, then $\Lambda_1 \cup \Lambda_2 \preccurlyeq \Lambda_1' \cup \Lambda_2'$.

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 $\mathsf{core}(\Delta) = -$.*
2. If $\Gamma \vdash \mathsf{a}!(\mathsf{x} : \eta).\mathsf{P} : \Delta$, then $\Gamma \vdash \mathsf{P} : (\mathsf{x} : \eta \cdot \Delta)$ and $\Gamma \vdash \mathsf{a} : \mathsf{begin}.\eta$.
3. If $\Gamma \vdash \mathsf{c}^{\mathsf{begin}.\eta'}?(\mathsf{x} : \eta).\mathsf{P} : \Delta$, then $\Gamma \vdash \mathsf{P} : (\mathsf{x} : \eta \cdot \Delta)$ and $\eta \bowtie \eta'$.
4. If $\Gamma \vdash \mathsf{h}?(\mathsf{x} : \mathsf{t})\mathsf{P} : \Delta$, then $\mathsf{core}(\Delta) = (\mathsf{h} : ?\mathsf{t}.\eta \cdot \Delta')$ and $\Gamma, \mathsf{x} : \mathsf{t} \vdash \mathsf{P} : (\mathsf{h} : \eta \cdot \Delta')$.
5. If $\Gamma \vdash \mathsf{h}!\mathsf{e}.\mathsf{P} : \Delta$, then $\mathsf{core}(\Delta) = (\mathsf{h} : !\mathsf{t}.\eta \cdot \Delta')$ and $\Gamma \vdash \mathsf{e} : \mathsf{t}$ and $\Gamma \vdash \mathsf{P} : (\mathsf{h} : \eta \cdot \Delta')$.
6. If $\Gamma \vdash \mathsf{h}?(\mathsf{x} : \chi).\mathsf{P} : \Delta$, then $\mathsf{core}(\Delta) = (\mathsf{h} : ?\chi.\eta)$ and $\Gamma \vdash \mathsf{P} : (\mathsf{h} : \eta \cdot \mathsf{x} : \eta')$ and $\chi \leq \eta'$.
7. If $\Gamma \vdash \mathsf{h}!\mathsf{h}'\mathsf{P} : \Delta$, then $\mathsf{core}(\Delta) = (\mathsf{h} : !\chi.\eta \cdot (\mathsf{h}' : \eta' \cdot \Delta'))$ and $\Gamma \vdash \mathsf{P} : (\mathsf{h} : \eta \cdot \Delta')$ and $\eta' \leq \chi$.
8. If $\Gamma \vdash \mathsf{P} + \mathsf{Q} : \Delta$, then $\mathsf{core}(\Delta) = (\mathsf{h} : \eta_1 + \eta_2 \cdot \Delta')$ and $\Gamma \vdash \mathsf{P} : (\mathsf{h} : \eta_1 \cdot \Delta')$ and $\Gamma \vdash \mathsf{Q} : (\mathsf{h} : \eta_2 \cdot \Delta')$.
9. If $\Gamma \vdash \mathsf{P} \oplus \mathsf{Q} : \Delta$, then either $\mathsf{core}(\Delta) = (\mathsf{h} : \eta_1 \oplus \eta_2 \cdot \Delta')$ and $\Gamma \vdash \mathsf{P} : (\mathsf{h} : \eta_1 \cdot \Delta')$ and $\Gamma \vdash \mathsf{Q} : \Delta', \mathsf{h} : \eta_2$ or $\Gamma \vdash \mathsf{P} : \Delta$, and $\Gamma \vdash \mathsf{Q} : \Delta$.

LEMMA B.7 (Generation Lemma for Systems). *1. If $\Gamma \Vdash \mathsf{P} : \Lambda$, then there exists Δ such that $\Lambda = \mathsf{set}(\Delta)$ and $\Gamma \vdash \mathsf{P} : \Delta$.*
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 $\Gamma, \mathsf{x} : \mathsf{t} \Vdash \mathbb{S} : \Lambda$ and $\vdash \mathsf{v} : \mathsf{t}$, then $\Gamma \Vdash \mathbb{S}[\mathsf{v}/\mathsf{x}] : \Lambda$.*
2. If $\Gamma \Vdash \mathbb{S} : \Lambda, \mathsf{x} : \eta$, then $\Gamma \Vdash \mathbb{S}[\mathsf{k}/\mathsf{x}] : \Lambda, \mathsf{k} : \eta$.

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 $\Gamma \vdash \mathsf{P} : \Delta$, then*

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

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

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

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

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

From Lemma B.6(8), we have that $\mathsf{core}(\Delta) = (\mathsf{k} : \eta_1 + \eta_2 \cdot \Delta')$ and $\Gamma \vdash \mathsf{P} : (\mathsf{k} : \eta_1 \cdot \Delta')$, and $\Gamma \vdash \mathsf{Q} : (\mathsf{k} : \eta_2 \cdot \Delta')$. By induction hypothesis on $\mathsf{P} \xrightarrow{\ell} \mathsf{P}'$ we get $\Gamma' \vdash \mathsf{P}' : \Delta''$, where $(\mathsf{k} : \eta_1 \cdot \Delta') \preccurlyeq \Delta''$. By Definition B.3(5), we have that $(\mathsf{k} : \eta_1 + \eta_2 \cdot \Delta') \preccurlyeq (\mathsf{k} : \eta_1 \cdot \Delta')$ and by transitivity $\Delta \preccurlyeq \Delta''$.

Case R-EXTINT. We have that

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

From Lemma B.6(8), we have that $\mathsf{core}(\Delta) = (\mathsf{k} : \eta_1 + \eta_2 \cdot \Delta')$ and $\Gamma \vdash \mathsf{P} : (\mathsf{k} : \eta_1 \cdot \Delta')$, and $\Gamma \vdash \mathsf{Q} : (\mathsf{k} : \eta_2 \cdot \Delta')$. By induction hypothesis on $\mathsf{P} \xrightarrow{\tau} \mathsf{P}'$ we get $\Gamma \vdash \mathsf{P}' : \Delta''$ where $(\mathsf{k} : \eta_1 \cdot \Delta') \preccurlyeq \Delta''$. 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 $\Delta'' = (\mathsf{k} : \eta_1' \cdot \Delta')$, for some η_1' such that $\eta_1 \longrightarrow \eta_1'$. By typing rule T-EXTCH we get $\Gamma \vdash \mathsf{P}' + \mathsf{Q} : (\mathsf{k} : \eta_1' + \eta_2 \cdot \Delta')$ and by the descriptor transition rule (TR3) we get $\eta_1 + \eta_2 \longrightarrow \eta_1' + \eta_2$. We conclude since $\Delta \preccurlyeq (\mathsf{k} : \eta_1' + \eta_2 \cdot \Delta')$ by Definition B.3(2).

Case R-EXTSEND. We have that

$$\frac{\mathsf{P} \xrightarrow{\mathsf{k}!\mathsf{v}}}{\mathsf{P} + \mathsf{Q} \xrightarrow{\tau} \mathsf{P}} \quad \text{and} \quad \Gamma \vdash \mathsf{P} + \mathsf{Q} : \Delta.$$

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

We have that

$$\frac{\mathsf{P} \xrightarrow{\mathsf{k}!\mathsf{k}_1}}{\mathsf{P} + \mathsf{Q} \xrightarrow{\tau} \mathsf{P}} \quad \text{and} \quad \Gamma \vdash \mathsf{P} + \mathsf{Q} : \Delta.$$

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

Case R-INTCH. We have that

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

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

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

□

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 $\mathbb{S} = \mathbb{T} \parallel \mathsf{P} \parallel \mathbb{T}'$ and $\mathbb{S}' = \mathbb{T} \parallel \mathsf{P}' \parallel \mathbb{T}'$ and $\mathsf{P} \xrightarrow{\ell} \mathsf{P}'$, where one or both $\mathbb{T}$, $\mathbb{T}'$ can be missing.*
2. *If $\Sigma \vdash \mathbb{S} \parallel \mathsf{P} \parallel \mathbb{T} \xrightarrow{\ell} \Sigma' \vdash \mathbb{S} \parallel \mathsf{P}' \parallel \mathbb{T}$ and $\Gamma \Vdash \mathbb{S} \parallel \mathsf{P} \parallel \mathbb{T} : \Lambda$, then there are Λ', Δ and Δ' such that $\Lambda = \Lambda' \cup \mathsf{set}(\Delta)$, $\Gamma \Vdash \mathbb{S} \parallel \mathsf{P} \parallel \mathbb{T} : \Lambda' \cup \mathsf{set}(\Delta')$, 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 $\Lambda \preccurlyeq \Lambda'$.*

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

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

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 $\Gamma \Vdash \mathbb{S}' : \Lambda_1, \mathsf{x} : \eta$, and $\Gamma \Vdash \mathbb{T}' : \Lambda_2, \mathsf{x} : \eta'$. From Lemma B.8(2), we have that $\Gamma \Vdash \mathbb{S}'[\mathsf{k}/\mathsf{x}] : \Lambda_1, \mathsf{k} : \eta$ and $\Gamma \Vdash \mathbb{T}'[\tilde{\mathsf{k}}/\mathsf{x}] : \Lambda_2, \tilde{\mathsf{k}} : \eta'$. Applying typing rule T-PAR we get $\Gamma \Vdash \mathbb{S}'[\mathsf{k}/\mathsf{x}] \parallel \mathbb{T}'[\tilde{\mathsf{k}}/\mathsf{x}] : \Lambda'$, where $\Lambda' = \Lambda_1 \cup \Lambda_2 \cup \{\mathsf{k} : \eta, \tilde{\mathsf{k}} : \eta'\}$. We conclude since $\Lambda \preccurlyeq \Lambda'$ by Definition B.4(1).

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

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

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(3) and (4) we have that $\Gamma \Vdash \mathbb{S}' : \Lambda_1'$, and $\Gamma, \mathsf{x} : \mathsf{t} \Vdash \mathbb{T}' : \Lambda_2'$, where $\Lambda_1 \preccurlyeq \Lambda_1'$ and $\Lambda_2 \preccurlyeq \Lambda_2'$. From Lemma B.8(1), we have that $\Gamma \Vdash \mathbb{T}'[\mathsf{v}/\mathsf{x}] : \Lambda_2'$. Applying typing rule T-PAR we get $\Gamma \Vdash \mathbb{S}' \parallel \mathbb{T}'[\mathsf{v}/\mathsf{x}] : \Lambda_1' \cup \Lambda_2'$. We conclude since $\Lambda \preccurlyeq \Lambda_1' \cup \Lambda_2'$ by Proposition B.5(2).

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

$$\frac{\Sigma \vdash \mathbb{S} \xrightarrow{\mathsf{k}!\mathsf{k}_1} \Sigma \vdash \mathbb{S}' \qquad \Sigma \vdash \mathbb{T} \xrightarrow{\tilde{\mathsf{k}}?(\mathsf{x}:\chi)} \Sigma \vdash \mathbb{T}' \qquad \Sigma(\mathsf{k}_1) \leq \chi}{\Sigma, \mathsf{k} : \eta, \tilde{\mathsf{k}} : \eta' \vdash \mathbb{S} \parallel \mathbb{T} \xrightarrow{\tau} \Sigma' \vdash \mathbb{S}' \parallel \mathbb{T}'[\mathsf{k}_1/\mathsf{x}]}$$

where $\Sigma' = \Sigma, \mathsf{k} : \eta\langle!\Sigma(\tilde{\mathsf{k}}_1)\rangle, \tilde{\mathsf{k}} : \eta'\langle?\Sigma(\tilde{\mathsf{k}}_1)\rangle$.
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 6) and (5) we have that $\Gamma \Vdash \mathbb{S}' : \Lambda_1'$, and $\Gamma \Vdash \mathbb{T}' : \Lambda_2', \mathsf{x} : \eta_0$, where $\Lambda_1 \preccurlyeq \Lambda_1'$ and $\Lambda_2 \preccurlyeq \Lambda_2'$ and $\chi \leq \eta_0$. From Lemma B.8(2), we have that $\Gamma \Vdash \mathbb{T}'[\mathsf{k}_1/\mathsf{x}] : \Lambda_2', \mathsf{k}_1 : \eta_0$. Applying typing rule T-PAR we get $\Gamma \Vdash \mathbb{S}' \parallel \mathbb{T}'[\mathsf{k}_1/\mathsf{x}] : \Lambda'$, where $\Lambda' = \Lambda_1' \cup \Lambda_2' \cup \{\mathsf{k}_1 : \eta_0\}$. We conclude since $\Lambda \preccurlyeq \Lambda'$ 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 $\mathsf{k} : \eta$ via the session environment Σ (notation $\ell \ltimes_\Sigma \mathsf{k} : \eta$) is the smallest relation such that $\Sigma(\mathsf{k}) \leq \eta$ and:*

$$\mathsf{v} \in \mathsf{t} \text{ implies } \mathsf{k}!\mathsf{v} \ltimes_\Sigma \mathsf{k} : !\mathsf{t}.\eta \qquad \mathsf{k}?(\mathsf{x} : \psi) \ltimes_\Sigma \mathsf{k} : ?\psi.\eta$$
$$\Sigma(\mathsf{k}') \leq \chi \text{ implies } \mathsf{k}!\mathsf{k}' \ltimes_\Sigma \mathsf{k} : !\chi.\eta'$$
$$\ell \ltimes_\Sigma \mathsf{k} : \eta \text{ implies } \ell \ltimes_\Sigma \mathsf{k} : \eta \oplus \eta'$$
$$\ell \ltimes_\Sigma \mathsf{k} : \eta \text{ implies } \ell \ltimes_\Sigma \mathsf{k} : \eta + \eta'.$$

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

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

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

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-EXT, T-INT. T-CONNECT, □

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* $\mathsf{P} \stackrel{\ell}{\Longrightarrow}$ *and* $\ell \ltimes_\Sigma \mathsf{k} : \eta$, $\mathsf{Q} \stackrel{\ell'}{\Longrightarrow}$ *and* $\ell' \ltimes_\Sigma \tilde{\mathsf{k}} : \theta$, *and* $\eta \bowtie \theta$. *Then* $\Sigma \vdash \mathsf{P} \,\|\, \mathsf{Q} \stackrel{\tau}{\longrightarrow}$.

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

1. $\mathsf{P} \stackrel{\mathsf{k}!\mathsf{v}}{\Longrightarrow}$ and $\mathsf{Q} \stackrel{\tilde{\mathsf{k}}?(\mathsf{x}:\mathsf{t})}{\Longrightarrow}$ and $\mathsf{v} \in \mathsf{t}$.
   In this case $\mathsf{P} \stackrel{\tau}{\Longrightarrow} \mathsf{k}!(\mathsf{e}).\mathsf{P}'$, and $\mathsf{Q} \stackrel{\tau}{\Longrightarrow} \tilde{\mathsf{k}}?(\mathsf{x} : \eta).\mathsf{Q}'\,\{+\,\mathsf{Q}''\}$ and we conclude by the reduction rules LIFT and COMMUNICATION.
2. $\mathsf{P} \stackrel{\mathsf{k}!(\!|\mathsf{k}_1|\!)}{\Longrightarrow}$ and $\mathsf{Q} \stackrel{\tilde{\mathsf{k}}?(\!|\mathsf{z}:\chi|\!)}{\Longrightarrow}$ and $\Sigma(\mathsf{k}_1) \leq \chi$.
   In this case $\mathsf{P} \stackrel{\tau}{\Longrightarrow} \mathsf{k}!(\!|\mathsf{k}_1|\!).\mathsf{P}'$, and $\mathsf{Q} \stackrel{\tau}{\Longrightarrow} \tilde{\mathsf{k}}?(\!|\mathsf{z} : \chi|\!).\mathsf{Q}'\,\{+\,\mathsf{Q}''\}$ and we conclude by the reduction rules LIFT and DELEGATION.

□

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* $\mathsf{k} : \eta$ *it contains also* $\tilde{\mathsf{k}} : \eta' \in \Lambda$ *with* $\eta \bowtie \eta'$.

LEMMA B.15. *If* $\mathbb{S}$ *is initial and* $\vdash \mathbb{S} \stackrel{\tau}{\Longrightarrow} \Sigma \vdash \mathsf{P}_1 \,\|\, \ldots \,\|\, \mathsf{P}_n$, *then* $\vdash \mathsf{P}_i : \Delta_i$ *for* $1 \leq i \leq n$ *imply:*

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

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

If $\mathbb{S}$ is an initial system, then it is a parallel composition of sums of $![\mathsf{z}]$. 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 rules pushes in both session environments the assumptions $\mathsf{k} : \eta$ and $\tilde{\mathsf{k}} : \eta'$, 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. □

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 $\mathsf{core}(\Lambda)$.

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

PROOF. By induction and by cases on $\stackrel{\tau}{\Longrightarrow}$. The more interesting case is that of rule DELEGATION with $\Vdash \mathbb{S} \,\|\, \mathbb{T} : \Lambda$ and:

$$\frac{\Sigma \vdash \mathbb{S} \stackrel{\mathsf{k}!(\!|\mathsf{k}_1|\!)}{\longrightarrow} \Sigma \vdash \mathbb{S}' \qquad \Sigma \vdash \mathbb{T} \stackrel{\tilde{\mathsf{k}}?(\!|\mathsf{z}:\chi|\!)}{\longrightarrow} \Sigma \vdash \mathbb{T}' \qquad \Sigma(\mathsf{k}_1) \leq \chi}{\Sigma, \mathsf{k} : \eta, \tilde{\mathsf{k}} : \eta' \vdash \mathbb{S} \,\|\, \mathbb{T} \stackrel{\tau}{\longrightarrow} \Sigma' \vdash \mathbb{S}' \,\|\, \mathbb{T}'[\mathsf{k}_1/\mathsf{z}]}$$

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

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* $\vdash \mathbb{S} \stackrel{\tau}{\Longrightarrow} \Sigma \vdash \mathsf{P}_1 \,\|\, \ldots \,\|\, \mathsf{P}_n$, *and the fresh internal channels take successive numbers according to the order of creation, then* $\Gamma_i \vdash \mathsf{P}_i : \Delta_i$ *implies that the indexes of internal channels in* Δ_i *are decreasing for* $1 \leq i \leq n$.

PROOF. By induction on $\stackrel{\tau}{\Longrightarrow}$ 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. □

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

PROOF. Let $\mathbb{S}$ be initial and $\vdash \mathbb{S} \stackrel{\tau}{\Longrightarrow} \Sigma \vdash \mathsf{P}_1 \,\|\, \ldots \,\|\, \mathsf{P}_n$, where $\vdash \mathsf{P}_i : \Delta_i$. It is easy to verify that $\Vdash \mathsf{P}_1 \,\|\, \ldots \,\|\, \mathsf{P}_n : \Lambda$, where $\Lambda = \bigcup_{1\leq i\leq n} \mathsf{set}(\mathsf{core}(\Delta_i))$. 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 $\mathsf{k}_j \in \mathsf{dom}(\mathsf{core}(\Delta_l))$ and $\tilde{\mathsf{k}}_j \in \mathsf{dom}(\mathsf{core}(\Delta_{l'}))$. By Lemma B.16, k_j and $\tilde{\mathsf{k}}_j$ 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. $\mathsf{P}_l \stackrel{\gamma}{\Longrightarrow}$ and $\vdash \mathsf{P}_{l'} \stackrel{\gamma'}{\Longrightarrow}$;
2. $\mathsf{P}_l \stackrel{\gamma}{\Longrightarrow}$ and $\mathsf{P}_{l'} \stackrel{\ell}{\Longrightarrow}$ and $\ell \ltimes_\Sigma \tilde{\mathsf{k}}_j : \Delta_{l'}(\tilde{\mathsf{k}}_j)$;
3. $\mathsf{P}_l \stackrel{\ell}{\Longrightarrow}$ and $\ell \ltimes_\Sigma \mathsf{k}_j : \Delta_l(\mathsf{k}_j)$ and $\mathsf{P}_{l'} \stackrel{\gamma}{\Longrightarrow}$;
4. $\mathsf{P}_l \stackrel{\ell}{\Longrightarrow}$ and $\ell \ltimes_\Sigma \mathsf{k}_j : \Delta_l(\mathsf{k}_j)$ and $\mathsf{P}_{l'} \stackrel{\ell'}{\Longrightarrow}$ and $\ell' \ltimes_\Sigma \tilde{\mathsf{k}}_j : \Delta_{l'}(\tilde{\mathsf{k}}_j)$.

In the last case the coherence of Λ (assured by Lemma B.15(3)) implies the duality between $\Delta_l(\mathsf{k}_j)$ and $\Delta_{l'}(\tilde{\mathsf{k}}_j)$. Therefore $\Sigma \vdash \mathsf{P}_l \,\|\, \mathsf{P}_{l'} \stackrel{\tau}{\longrightarrow}$ by Lemma B.13. □

C. Proofs of Section 3.3

C.1 Proof of Subject Reduction

In order to state and prove the Subject Reduction Theorem we have first to introduce some definitions and propositions.

The first definition formalises the evolution of session types and session environments.

DEFINITION C.1. *1. A session type* η' *is at a* later stage *than another session type* η, $\eta \sqsubseteq \eta'$, *if that is deducible from the following rules.*

$$\text{LATER-0}\ \frac{}{\eta \sqsubseteq \varepsilon} \qquad \text{LATER-1}\ \frac{}{\eta \sqsubseteq \eta} \qquad \text{LATER-2}\ \frac{\eta \sqsubseteq \eta'' \quad \eta'' \sqsubseteq \eta'}{\eta \sqsubseteq \eta'}$$

$$\text{LATER-3}\ \frac{\eta \sqsubseteq \eta'}{\eta.\eta'' \sqsubseteq \eta'.\eta''} \qquad \text{LATER-4}\ \frac{}{!\mathsf{t}.\eta \oplus !\mathsf{t}.\eta \sqsubseteq \eta_i} \qquad \text{LATER-5}\ \frac{}{?\mathsf{t}.\eta + ?\mathsf{t}.\eta \sqsubseteq \eta_i}$$

2. *A session environment* Θ' *is at a* later stage *than another session environment* Θ, $\Theta \sqsubseteq \Theta'$, *if*
   (a) $\mathsf{dom}(\Theta) \subseteq \mathsf{dom}(\Theta')$, *and*

(b) $\forall \mathtt{k} \in \mathsf{dom}(\Theta) : \Theta(\mathtt{k}) \sqsubseteq \Theta'(\mathtt{k})$.

The second definition gives standard conditions on well-formation of heaps and agreement between heaps and term environments.

DEFINITION C.2 (Well-Formed Heap and Agreement). *A term environment Γ and a heap h* agree*, written wf$(\Gamma; h)$, if h is well-formed and the classes of objects in h are the classes associated to them by Γ. Thus wf$(\Gamma; h)$ if*

1. $h(\mathtt{o}) = (\mathtt{C}, \overline{\mathtt{f} : \mathtt{o}})$, *ftype*$_r(\mathtt{C}, \mathtt{f}_i) = \mathtt{t} \Rightarrow h(\mathtt{o})(\mathtt{f}_i) = (\mathtt{C}', _)$, $\mathtt{C}' <: \mathtt{t}$,
2. $\forall \mathtt{o} \in \mathsf{dom}(\Gamma)$, $h(\mathtt{o}) = (\Gamma(\mathtt{o}), -)$.

In point 1 of the above definition, recall that *ftype*$_r(\mathtt{C}, \mathtt{f}_i)$=*ftype*$_w(\mathtt{C}, \mathtt{f}_i)$ being $\mathtt{C}$ a class.

We type only single expressions, but they can result in parallel threads. Since we do not have a typing for parallel threads we require each single expression to be well-typed. Moreover we want to get our property in the most general form, allowing the property to hold for all well-typed expressions, which sometimes can be generated by initial expressions only in parallel with other expressions. For example, no initial expression can reduce to the expression

$$\mathtt{e} = \mathtt{o}.\eta\,\{\mathtt{sendC(5)}\{\mathtt{e}_1\}\}; \mathtt{k}.\mathtt{sendC(3)}\{\mathtt{e}_2 \wr \mathtt{k}\int\},$$

but

$$\mathtt{e}_0 = \mathtt{o}'.\eta'\,\{\mathtt{o}.\eta\,\{\mathtt{sendC(5)}\{\mathtt{e}_1\}\}; \mathtt{sendC(3)}\{\mathtt{e}_2\}\}$$

reduces to $\mathtt{e} \mid\mid \tilde{\mathtt{k}}.\mathtt{recC}(\mathtt{x})\{[\mathtt{o}/\mathtt{this}]\mathtt{e}' \wr \tilde{\mathtt{k}}\int\}$ if $\mathtt{recC}(\mathtt{x})\{\mathtt{e}'\}$ is the body of session η in the class of the object $\mathtt{o}$.

Notice also that receive expressions can never get objects of wrong types. For example the execution of $\mathtt{k}.\mathtt{recC}(\mathtt{x})\{\mathtt{Bool} \Rightarrow \neg\mathtt{x}\}$ if $h(\mathtt{k}) = 3$ is simply stopped, i.e. it does not produce a runtime error. For this reason, in contrast to the calculus of [10], we do not need to require agreement between the objects in the queues associated to channels by the heap and the session types of the same channels in the session environment. The example above also suggests that this agreement will be a key ingredient of the progress proof.

The typing rules for runtime expressions differ from the ones for user expressions only in assigning the session type to explicit channels, not in the union type. The relations between the two systems are clarified by the following proposition that will be essential in considering the progress property.

PROPOSITION C.3. $\Gamma \vdash \mathtt{e} : \mathtt{t} \fatsemi \eta$ *implies* $\Gamma \models_{\mathtt{r}} \mathtt{e} \wr \mathtt{k}\int : \mathtt{t} \fatsemi \{\mathtt{k} : \eta\}$.

Let us notice that $\Gamma \models_{\mathtt{r}} \mathtt{e} : \mathtt{t} \fatsemi \emptyset$ is equivalent to $\Gamma \models_{\mathtt{r}} \mathtt{e} : \mathtt{t} \fatsemi \{\mathtt{k} : \varepsilon\}$ by our convention on session environments.

We now introduce some lemmas used in the proof of the Theorem C.1; their proofs are trivial and then are omitted.

Lemma C.4 states that every expression, different from an object, can be written as a composition of an evaluation context and a redex, where the set of redexes is defined by:

$$\begin{array}{c} \mathtt{new}\,\mathtt{C}(\overline{\mathtt{o}}) \mid \mathtt{o.f} \mid \mathtt{o.f} := \mathtt{o} \mid \mathtt{o.s}\,\{\mathtt{e} : \eta\} \mid \mathtt{o} \bullet \mathtt{s}\,\{\mathtt{k} : \chi\} \\ \mathtt{k.sendC}(\mathtt{o})\{\mathtt{C} \Rightarrow \mathtt{e} \,[]\, \mathtt{C} \Rightarrow \mathtt{e}\} \mid \mathtt{k.recC}(\mathtt{x})\{\mathtt{C} \Rightarrow \mathtt{e} \,[]\, \mathtt{C} \Rightarrow \mathtt{e}\} \end{array}$$

The fact that the evaluation context is unique expresses the determinism of the evaluation strategy.

LEMMA C.4. *Let* $\mathtt{e}$ *be a runtime expression such that:*

1. $\mathtt{e}$ *is not an object,*
2. $\Gamma \models_{\mathtt{r}} \mathtt{e} : \mathtt{t} \fatsemi \Theta$ *for some* Γ, Θ *and* $\mathtt{t}$,

then there exists a unique evaluation context $\mathscr{E}$ such that $\mathtt{e} = \mathscr{E}[\mathtt{r}]$ *for some redex* $\mathtt{r}$.

The following lemma states the obvious property that, in any type derivation ending by rule SUB-RT, there is a subderivation giving a subtype to the same expression such that its final rule is different from SUB-RT.

LEMMA C.5. *In any derivation of* $\Gamma \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{t} \fatsemi \Theta$ *there is a subderivation of* $\Gamma \models_{\mathtt{r}} \mathtt{e} : \mathtt{t}' \fatsemi \Theta$, *with* $\mathtt{t} <: \mathtt{t}'$, *where the last applied rule is different from* SUB-RT.

Lemma C.6 states that the typing of $\mathscr{E}[\mathtt{e}]$ can be broken down into the typing of $\mathtt{e}$, and the typing of $\mathscr{E}[\mathtt{x}]$. Accordingly the environment Θ , that is used to type $\mathscr{E}[\mathtt{e}]$, can be broken down into two environments, $\Theta = \Theta_1 \circ \Theta_2$, where Θ_1 is used to type $\mathtt{e}$, and Θ_2 is used to type $\mathscr{E}[\mathtt{x}]$.

LEMMA C.6 (Subderivations). *If* $\Gamma \models_{\mathtt{r}} \mathscr{E}[\mathtt{e}] : \mathtt{t} \fatsemi \Theta$, *then there exist* Θ_1, Θ_2, $\mathtt{t}'$, *such that for all* $\mathtt{x}$ *fresh in* $\mathscr{E}$, Γ, *we get* $\Theta = \Theta_1 \circ \Theta_2$, *and* $\Gamma \models_{\mathtt{r}} \mathtt{e} : \mathtt{t}' \fatsemi \Theta_1$, *and* $\Gamma(\mathtt{x} : \mathtt{t}') \models_{\mathtt{r}} \mathscr{E}[\mathtt{x}] : \mathtt{t} \fatsemi \Theta_2$.

Lemma C.7 allows the combination of the typing of $\mathscr{E}[\mathtt{x}]$ and the typing of $\mathtt{e}$, provided that the type of $\mathtt{e}$ is the same as that of $\mathtt{x}$ in the first typing.

LEMMA C.7 (Context Substitution). *If* $\Gamma \models_{\mathtt{r}} \mathtt{e} : \mathtt{t} \fatsemi \Theta_1$, *and* $\Gamma(\mathtt{x} : \mathtt{t}) \models_{\mathtt{r}} \mathscr{E}[\mathtt{x}] : \mathtt{t}' \fatsemi \Theta_2$, *then* $\Gamma \models_{\mathtt{r}} \mathscr{E}[\mathtt{e}] : \mathtt{t}' \fatsemi \Theta_1 \circ \Theta_2$.

The following lemma expresses the weakening property for term environments.

LEMMA C.8 (Weakening). *Let* $\Gamma \models_{\mathtt{r}} \mathtt{e} : \mathtt{t} \fatsemi \Theta$.

1. *If* $\mathtt{x} \notin \mathsf{dom}(\Gamma)$ *then* $\Gamma(\mathtt{x} : \mathtt{t}') \models_{\mathtt{r}} \mathtt{e} : \mathtt{t} \fatsemi \Theta$.
2. *If* $\mathtt{o} \notin \mathsf{dom}(\Gamma)$ *then* $\Gamma(\mathtt{o} : \mathtt{C}) \models_{\mathtt{r}} \mathtt{e} : \mathtt{t} \fatsemi \Theta$.

As usual, for proving Subject Reduction, it is handy to show preservation of typing under term substitution.

LEMMA C.9 (Term Substitution). *If* $\Gamma(\mathtt{z} : \mathtt{C}) \vdash_{\mathtt{r}} \mathtt{e} : \mathtt{t} \fatsemi \Theta$ *and* $\Gamma \models_{\mathtt{r}} \mathtt{o} : \mathtt{C} \fatsemi \emptyset$, *then* $\Gamma \models_{\mathtt{r}} [\mathtt{o}/\mathtt{z}]\mathtt{e} : \mathtt{t} \fatsemi \Theta$.

The last lemma expresses a standard property about the typing of session bodies.

LEMMA C.10 (Typing of Session Bodies). *If* *stype*$(\mathtt{s}, \mathtt{C}) = \eta$, *and* *rtype*$(\mathtt{s}, \eta, \mathtt{C}) = \mathtt{t}$ *and* *sbody*$(\mathtt{s}, \eta, \mathtt{C}) = \mathtt{e}$, *then* $\{\mathtt{this} : \mathtt{C}\} \vdash \mathtt{e} : \mathtt{t} \fatsemi \eta$.

THEOREM (Subject Reduction). *If wf$(\Gamma; h)$ and* $\Gamma \models_{\mathtt{r}} \mathtt{e} : \mathtt{t} \fatsemi \Theta$ *then*

1. $\mathtt{e}, h \longrightarrow \mathtt{e}', h'$ *implies that there exist* Θ', Γ' *such that* $\Gamma \subseteq \Gamma'$ *and* $\Theta \sqsubseteq \Theta'$, *and wf$(\Gamma'; h')$, and* $\Gamma' \models_{\mathtt{r}} \mathtt{e}' : \mathtt{t} \fatsemi \Theta'$.
2. $\mathtt{e}, h \longrightarrow \mathtt{e}_1 \mid\mid \mathtt{e}_2, h'$ *implies that* $h' = h[\mathtt{k}, \tilde{\mathtt{k}} \mapsto ()]$ *for some fresh* $\mathtt{k}$, *and wf$(\Gamma; h')$, and that there exist* $\mathtt{t}', \eta, \eta'$ *such that* $\Gamma \models_{\mathtt{r}} \mathtt{e}_1 : \mathtt{t} \fatsemi \Theta, \{\mathtt{k} : \eta\}$, *and* $\Gamma \models_{\mathtt{r}} \mathtt{e}_2 : \mathtt{t}' \fatsemi \{\tilde{\mathtt{k}} : \eta'\}$, *and* $\eta \bowtie \eta'$.

By induction on the definition of $\longrightarrow$. We proceed by case analysis.

By Lemma C.5 we consider typing derivations of $\Gamma \models_{\mathtt{r}} \mathtt{e} : \mathtt{t} \fatsemi \Theta$ where the last applied rule is different from the rule SUBRT.

Let the last applied rule be SESSREQ-R:

$$\frac{h(\mathtt{o}) = (\mathtt{C}, _) \qquad \mathit{sbody}(\mathtt{s}, \eta', \mathtt{C}) = \mathtt{e}' \qquad \mathtt{k}, \tilde{\mathtt{k}} \notin h \qquad \eta \bowtie \eta'}{\mathscr{E}[\mathtt{o.s}\,\{\mathtt{e} : \eta\}], h \longrightarrow \mathscr{E}[\mathtt{e} \wr \mathtt{k}\int] \mid\mid [\mathtt{o}/\mathtt{this}]\mathtt{e}' \wr \tilde{\mathtt{k}}\int, h[\mathtt{k}, \tilde{\mathtt{k}} \mapsto ()]}$$

By AXIOM-RT, and by *wf$(\Gamma; h)$* and $h(\mathtt{o}) = (\mathtt{C}, _)$, we get that $\emptyset \models_{\mathtt{r}} \mathtt{o} : \mathtt{C} \fatsemi \emptyset$.

Since, by hypothesis $\Gamma \models_{\mathtt{r}} \mathscr{E}[\mathtt{o.s}\,\{\mathtt{e} : \eta\}] : \mathtt{t} \fatsemi \Theta$, by Lemma C.6, we have that $\Gamma \models_{\mathtt{r}} \mathtt{o.s}\,\{\mathtt{e} : \eta\} : \mathtt{t}' \fatsemi \Theta_1$ and $\Theta = \Theta_1 \circ \Theta_2$. From rule SESSREQ-RT we have that $\Theta_1 = \emptyset$, $\Theta_2 = \Theta$, $\Gamma \vdash \mathtt{e} : \mathtt{t}' \fatsemi \eta'$.

By Proposition C.3, $\Gamma \models_{\mathtt{r}} \mathtt{e} \wr \mathtt{k}\int : \mathtt{t}' \fatsemi \{\mathtt{k} : \eta'\}$.

By Lemma C.7, we have that $\Gamma \models_{\mathtt{r}} \mathscr{E}[\mathtt{e} \wr \mathtt{k}\int] : \mathtt{t} \fatsemi \{\mathtt{k} : \eta'\} \circ \Theta$.

Let $rtype(\mathtt{s}, \eta', \mathtt{C}) = \mathtt{t}_0$, then $\mathtt{this} : \mathtt{C} \vdash \mathtt{e}' : \mathtt{t}_0 \S \eta$ by Lemma C.10, and $\mathtt{this} : \mathtt{C} \vdash_{\mathtt{r}} \mathtt{e}' \wr \tilde{\mathtt{k}} \int : \mathtt{t}_0 \S \{\tilde{\mathtt{k}} : \eta\}$, by Proposition C.3.

Therefore, by Lemmas C.8 and C.9, we have that $\Gamma \vDash_{\mathtt{r}} [\mathtt{o}/\mathtt{this}]\mathtt{e}' \wr \tilde{\mathtt{k}} \int : \mathtt{t}_0 \S \{\tilde{\mathtt{k}} : \eta\}$.

Notice that the new heap $h[\mathtt{k}, \tilde{\mathtt{k}} \mapsto ()]$ still agrees with Γ since the only changes are about channels.

Let the last applied rule be SESSDEL-R:

$$\frac{h(\mathtt{o}) = (\mathtt{C}, _) \qquad sbody(\mathtt{s}, \eta, \mathtt{C}) = \mathtt{e} \qquad \chi \leq \eta}{\mathscr{E}[\mathtt{o} \bullet \mathtt{s}\,\{\mathtt{k} : \chi\}], h \longrightarrow \mathscr{E}[[\mathtt{o}/\mathtt{this}]\mathtt{e} \wr \mathtt{k} \int], h}$$

By AXIOM-RT, and by $wf(\Gamma; h)$ and $h(\mathtt{o}) = (\mathtt{C}, _)$, we get that $\emptyset \vDash_{\mathtt{r}} \mathtt{o} : \mathtt{C} \S \emptyset$.

Since, by hypothesis $\Gamma \vDash_{\mathtt{r}} \mathscr{E}[\mathtt{o} \bullet \mathtt{s}\,\{\mathtt{k} : \chi\}] : \mathtt{t}' \S \Theta$, by Lemma C.6, we have that $\Gamma \vDash_{\mathtt{r}} \mathtt{o} \bullet \mathtt{s}\,\{\mathtt{k} : \chi\} : \mathtt{t}' \S \Theta_1$ and $\Theta = \Theta_1 \circ \Theta_2$. From rule SESSDEL-RT we have that $\Theta_1 = \{\mathtt{k} : \eta\}$, $stype(\mathtt{s}, \mathtt{C}) = \eta$ and $rtype(\mathtt{s}, \eta, \mathtt{C}) = \mathtt{t}'$.

By Lemma C.10, $\mathtt{this} : \mathtt{C} \vdash \mathtt{e} : \mathtt{t}' \S \eta$, and $\mathtt{this} : \mathtt{C} \vDash_{\mathtt{r}} \mathtt{e} \wr \mathtt{k} \int : \mathtt{t}' \S \{\mathtt{k} : \eta\}$, by Proposition C.3. Therefore, by Lemmas C.8 and C.9, we have that $\Gamma \vDash_{\mathtt{r}} [\mathtt{o}/\mathtt{this}]\mathtt{e} \wr \mathtt{k} \int : \mathtt{t}' \S \{\mathtt{k} : \eta\}$. By Lemma C.7, we have that $\Gamma \vDash_{\mathtt{r}} \mathscr{E}[[\mathtt{o}/\mathtt{this}]\mathtt{e} \wr \mathtt{k} \int] : \mathtt{t} \S \Theta$.

Let the last applied rule be SENDCASE-R:

$$\frac{h(\tilde{\mathtt{k}}) = \overline{\mathtt{o}} \quad h(\mathtt{o}) = (\mathtt{C}, _) \quad \mathtt{C} \leq \mathtt{t}_i}{\mathscr{E}[\mathtt{k}.\mathtt{sendC}(\mathtt{o})\{\mathtt{t}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{t}_2 \Rightarrow \mathtt{e}_2\}], h \longrightarrow \mathscr{E}[\mathtt{e}_i], h[\tilde{\mathtt{k}} \mapsto \overline{\mathtt{o}} :: \mathtt{o}]}$$

By AXIOM-RT, and by $wf(\Gamma; h)$ and $h(\mathtt{o}) = (\mathtt{C}, _)$, we get that $\emptyset \vDash_{\mathtt{r}} \mathtt{o} : \mathtt{C} \S \emptyset$.

Since, by hypothesis $\Gamma \vDash_{\mathtt{r}} \mathscr{E}[\mathtt{k}.\mathtt{sendC}(\mathtt{o})\{\mathtt{t}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{t}_2 \Rightarrow \mathtt{e}_2\}] : \mathtt{t} \S \Theta$, by Lemma C.6, we have that $\Gamma \vDash_{\mathtt{r}} \mathtt{k}.\mathtt{sendC}(\mathtt{o})\{\mathtt{t}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{t}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{t}' \S \Theta_1$ and $\Theta = \Theta_1 \circ \Theta_2$. From rule SENDC-RT we have that $\Theta_1 = \{\mathtt{k} : !\mathtt{t}_1.\eta_1 \oplus !\mathtt{t}_2.\eta_2\}$, $\Gamma \vDash_{\mathtt{r}} \mathtt{e}_i : \mathtt{t}' \S \{\mathtt{k} : \eta_i\}$.

By Lemma C.7, we have that $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[\mathtt{e}_i] : \mathtt{t} \S \Theta'$, where $\Theta' = \{\mathtt{k} : \eta_i\} \circ \Theta_2$.

From Definition C.1, LATER-3 and LATER-4, we have that $\Theta \sqsubseteq \Theta'$.

Notice that the new heap $h[\tilde{\mathtt{k}} \mapsto \overline{\mathtt{o}} :: \mathtt{o}]$ still agrees with Γ since the only changes are about channels.

Let the last applied rule be RECEIVECASE-R:

$$\frac{h(\mathtt{k}) = \mathtt{o} :: \overline{\mathtt{o}} \quad h(\mathtt{o}) = (\mathtt{C}, _) \quad \mathtt{C} \leq \mathtt{t}_i}{\mathscr{E}[\mathtt{k}.\mathtt{recC}(\mathtt{x})\{\mathtt{t}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{t}_2 \Rightarrow \mathtt{e}_2\}], h \longrightarrow \mathscr{E}[[\mathtt{o}/\mathtt{x}]\mathtt{e}_i], h[\mathtt{k} \mapsto \overline{\mathtt{o}}]}$$

By AXIOM-RT, and by $wf(\Gamma; h)$ and $h(\mathtt{o}) = (\mathtt{C}, _)$, we get that $\emptyset \vDash_{\mathtt{r}} \mathtt{o} : \mathtt{C} \S \emptyset$. Applying rule SUB-RT, we get $\emptyset \vDash_{\mathtt{r}} \mathtt{o} : \mathtt{C}_i \S \emptyset$.

Since, by hypothesis $\Gamma \vDash_{\mathtt{r}} \mathscr{E}[\mathtt{k}.\mathtt{recC}(\mathtt{x})\{\mathtt{t}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{t}_2 \Rightarrow \mathtt{e}_2\}] : \mathtt{t} \S \Theta$, by Lemma C.6, we have that $\Gamma \vDash_{\mathtt{r}} \mathtt{k}.\mathtt{recC}(\mathtt{x})\{\mathtt{t}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{t}_2 \Rightarrow \mathtt{e}_2\} : \mathtt{t}' \S \Theta_1$ and $\Theta = \Theta_1 \circ \Theta_2$. From rule RECEIVEC-RT we have that $\Theta_1 = \{\mathtt{k} : ?\mathtt{t}_1.\eta_1 + ?\mathtt{t}_2.\eta_2\}$, $\Gamma, \mathtt{x} : \mathtt{t}_i \vDash_{\mathtt{r}} \mathtt{e}_i : \mathtt{t}' \S \{\mathtt{k} : \eta_i\}$.

By Lemma C.9, we have that $\Gamma \vDash_{\mathtt{r}} [\mathtt{o}/\mathtt{x}]\mathtt{e}_i : \mathtt{t}' \S \{\mathtt{k} : \eta_i\}$.

By Lemma C.7, we have that $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[[\mathtt{o}/\mathtt{x}]\mathtt{e}_i] : \mathtt{t} \S \Theta'$, where $\Theta' = \{\mathtt{k} : \eta_i\} \circ \Theta_2$.

From Definition C.1, LATER-3 and LATER-4, we have that $\Theta \sqsubseteq \Theta'$.

Notice that the new heap $h[\mathtt{k} \mapsto \overline{\mathtt{o}}]$ still agrees with Γ since the only changes are about channels.

The remaining cases easily follow from the induction hypothesis.

C.2 Proof of Progress

The proof of progress requires to study global properties of type preservation during the reduction of parallel threads. Namely we need to take into account the objects in the queues associated to channels and their relations with the session types of the channels themselves. In the following definition we extend the notion of duality between session types taking into account also the objects already sent by a thread, and waiting to be read by the thread which has the dual channel.

DEFINITION C.11. *Let h be a heap, $\overline{\mathtt{o}}$ be a queue of objects in h and η, η' two session types. The relation $\eta \ltimes_h^{\overline{\mathtt{o}}} \eta'$ is defined by:*

1. $\eta \ltimes_h^{()} \eta'$ *if* $\eta \bowtie \eta'$,
2. $\eta_i \circ \eta' \ltimes_h^{\overline{\mathtt{o}}::\mathtt{o}_i} \eta''$ *if* $(!\mathtt{t}_1.\eta_1 \oplus !\mathtt{t}_2.\eta_2) \circ \eta' \ltimes_h^{\overline{\mathtt{o}}} \eta''$ *and* $h(\mathtt{o}_i) = (\mathtt{C}, _)$ *and* $\mathtt{C} \leq \mathtt{t}_i$.

Intuitively, the definition above describes an agreement between the (session) type η of a channel $\mathtt{k}$ and the type η' of $\tilde{\mathtt{k}}$ after the objects $\overline{\mathtt{o}}$ have been memorized in the queue associated with $\tilde{\mathtt{k}}$ in h (recall that communication is asynchronous and that only one between the queues $h(\mathtt{k})$ and $h(\tilde{\mathtt{k}})$ can be nonempty). Thus, when the queue is empty (case (1) of the definition), η' and η agree if they are dual. When the queue is $\overline{\mathtt{o}} :: \mathtt{o}_i$ (case (2)), if the type η'' agrees with $(!\mathtt{t}_1.\eta_1 \oplus !\mathtt{t}_2.\eta_2) \circ \eta'$ after the objects $\overline{\mathtt{o}}$ have been put in the queue, then it also agrees with the type $\eta_i \circ \eta'$, where η_i is the session type of the branch obtained after putting in the queue the object $\mathtt{o}_i$. For instance, η'' agrees with $\eta_1 \circ \eta'$ via the queue $\mathtt{"a"::true::3}$ (notation $\eta_1 \circ \eta' \ltimes_h^{\mathtt{"a"::true::3}} \eta''$) if it agrees with $(!\mathtt{Int}.\eta_1 \oplus !\mathtt{Object}.\eta_2) \circ \eta'$ via the queue $\mathtt{"a"::true}$: indeed after branch selection (the sent value 3 is an $\mathtt{Int}$) the continuation of $(!\mathtt{Int}.\eta_1 \oplus !\mathtt{Object}.\eta_2) \circ \eta'$ is $\eta_1 \circ \eta'$.

The main lemma concerning the above relation says that if the type η of a channel $\mathtt{k}$ agrees with the type $(?\mathtt{t}_1.\eta_1 + ?\mathtt{t}_2.\eta_2) \circ \eta'$ of $\tilde{\mathtt{k}}$ when h maps $\tilde{\mathtt{k}}$ to the queue $\mathtt{o}_i :: \overline{\mathtt{o}}$, and $\mathtt{C} \leq \mathtt{t}_i$, where $\mathtt{C}$ is the class of $\mathtt{o}_i$ in h, then η agrees with $\eta_i \circ \eta'$ when h maps $\tilde{\mathtt{k}}$ to the queue $\overline{\mathtt{o}}$.

LEMMA C.12. *Let $\eta \ltimes_h^{\mathtt{o}_i::\overline{\mathtt{o}}} (?\mathtt{t}_1.\eta_1 + ?\mathtt{t}_2.\eta_2) \circ \eta'$, and $h(\mathtt{o}_i) = (\mathtt{C}, _)$, and $\mathtt{C} \leq \mathtt{t}_i$, then $\eta \ltimes_h^{\overline{\mathtt{o}}} \eta_i \circ \eta'$.*

PROOF. By induction on the length of $\overline{\mathtt{o}}$. The base case $\overline{\mathtt{o}} = ()$ is straightforward from Definition C.11.
For the induction case assume $\overline{\mathtt{o}} = \overline{\mathtt{o}'} :: \mathtt{o}^+$: thus the hypothesis becomes $\eta \ltimes_h^{\mathtt{o}_i::\overline{\mathtt{o}'}::\mathtt{o}^+} (?\mathtt{t}_1.\eta_1 + ?\mathtt{t}_2.\eta_2) \circ \eta'$. This relation can only have been obtained by case (2) of Definition C.11. So we have $\eta = \eta_j^+ \circ \eta^*$ for some η_j^+ $(1 \leq j \leq n)$ and η^* and $h(\mathtt{o}^+) = (\mathtt{C}^+, _)$ and $\mathtt{C}^+ \leq \mathtt{t}_j^+$ and $(!\mathtt{t}^+{}_1.\eta^+{}_1 \oplus !\mathtt{t}^+{}_2.\eta^+{}_2) \circ \eta^* \ltimes_h^{\mathtt{o}_i::\overline{\mathtt{o}'}} (?\mathtt{t}_1.\eta_1 + ?\mathtt{t}_2.\eta_2) \circ \eta'$. By induction hypothesis we have $\mathtt{C} \leq \mathtt{t}_i$ and $(!\mathtt{t}^+{}_1.\eta^+{}_1 \oplus !\mathtt{t}^+{}_2.\eta^+{}_2) \circ \eta^* \ltimes_h^{\overline{\mathtt{o}'}} \eta_i \circ \eta'$. Applying again Definition C.11(2) we get the result. □ □

A session environment and a heap are well-formed if:

- the same set of channels occurs in the session environment and in the heap,
- at least one between the queue of a channel and the one of its dual is empty,
- when the queue of a channel $\mathtt{k}$ is empty, then the queue of $\tilde{\mathtt{k}}$ relates the session type of $\mathtt{k}$ with the session type of $\tilde{\mathtt{k}}$.

A standard environment, a session environment and a heap are well-formed if both the heap with the standard environment and the heap with the session environment are well-formed.

DEFINITION C.13. *1. The predicate $wf(\Theta; h)$ is defined by:*

$$wf(\Theta; h) \quad \text{if} \quad \begin{cases} \mathtt{k} \in \mathsf{dom}(\Theta) \;\Leftrightarrow\; \mathtt{k} \in \mathsf{dom}(h), \\ \forall \mathtt{k} \in \mathsf{dom}(\Theta) : h(\mathtt{k}) \neq () \;\Rightarrow\; h(\tilde{\mathtt{k}}) = (), \\ \forall \mathtt{k} \in \mathsf{dom}(\Theta) : h(\mathtt{k}) = () \;\Rightarrow\; \Theta(\mathtt{k}) \ltimes_h^{h(\tilde{\mathtt{k}})} \Theta(\tilde{\mathtt{k}}). \end{cases}$$

2. $wf(\Gamma; \Theta; h)$ if $wf(\Gamma; h)$ and $wf(\Theta; h)$.

It is easy to verify that $wf(\Theta; h)$ and $\mathtt{k} \in \mathsf{dom}(h)$ imply $\tilde{\mathtt{k}} \in \mathsf{dom}(h)$.

The following key lemma states that, by reducing parallel threads in a heap which is well-formed with respect to the environments used to type the threads, we get parallel threads and heaps with the same well-formedness property.

LEMMA C.14. *Let* $\Gamma \vdash_{\mathtt{r}} \Theta_i : \mathtt{e}_i \triangleright \mathtt{C}_i$, $(1 \leq i \leq n)$, *and assume* $wf(\Gamma; \Theta; h)$ *where* $\Theta = \bigcup_{1 \leq i \leq n} \Theta_i$. *Then if*

$$\mathtt{e}_1 \mid\mid \ldots \mid\mid \mathtt{e}_n, h \longrightarrow \mathtt{e}'_1 \mid\mid \ldots \mid\mid \mathtt{e}'_{n'}, h$$

there are Γ' *and* Θ'_i *such that* $\Gamma' \vdash_{\mathtt{r}} \Theta'_i : \mathtt{e}'_i \triangleright \mathtt{C}_i$ $(1 \leq i \leq n')$, *and* $\Gamma \subseteq \Gamma'$, $\Theta \sqsubseteq \Theta'$ *and* $wf(\Gamma'; \Theta'; h')$, *where* $\Theta' = \bigcup_{1 \leq i \leq n'} \Theta'_i$.

PROOF. We have that, for some i $(1 \leq i \leq n)$, either $\mathtt{e}_i, h \longrightarrow \mathtt{e}'_i \mid\mid \mathtt{e}''_i, h'$ by an application of rule SESSREQ-R or $\mathtt{e}_i, h \longrightarrow \mathtt{e}'_i, h'$ by the application of any one of the other reduction rules. In the former case the proof follows immediately by Theorem C.1(2) and Definition C.13.
So let $\mathtt{e}_i, h \longrightarrow \mathtt{e}'_i, h'$. If this reduction has not been obtained by a communication rule the proof is trivial by Theorem C.1(1). The interesting cases are when the reduction $\mathtt{e}_i, h \longrightarrow \mathtt{e}'_i, h'$ is obtained by a communication rule.

RECEIVECASE-R. Assume $\mathtt{e}_i = \mathscr{E}[\mathtt{k}.\mathtt{recC}(\mathtt{x})\{\mathtt{t}_1 \Rightarrow \mathtt{e}''_1 \,[]\, \mathtt{t}_2 \Rightarrow \mathtt{e}''_2\}]$. We have that

$$\mathscr{E}[\mathtt{k}.\mathtt{recC}(\mathtt{x})\{\mathtt{t}_1 \Rightarrow \mathtt{e}''_1 \,[]\, \mathtt{t}_2 \Rightarrow \mathtt{e}''_2\}],\ h \longrightarrow \mathscr{E}[[\mathtt{o}/\mathtt{x}]\mathtt{e}''_j],\ h',$$

where $h(\mathtt{k}) = \mathtt{o} :: \overline{\mathtt{o}}$, and $h(\mathtt{o}) = (\mathtt{D}, _)$, and $h' = h[\mathtt{k} \mapsto \overline{\mathtt{o}}]$ and $\mathtt{D} \leq \mathtt{t}_j$.
Since $\Gamma \vdash_{\mathtt{r}} \Theta_i : \mathtt{e}_i \triangleright \mathtt{t}'_i$, by rule RECEIVEC-RT and Lemma C.6 we must have for some Θ''_i, η_1, η_2, η', $\mathtt{t}_0$, and $\mathtt{y}$ fresh in $\mathscr{E}$:

- $\Theta_i = \Theta''_i, \mathtt{k} : (?\mathtt{t}_1.\eta_1 \,+\, ?\mathtt{t}_2.\eta_2) \circ \eta'$,
- $\Gamma \vdash_{\mathtt{r}} \mathtt{t}_0 : \mathtt{k}.\mathtt{recC}(\mathtt{x})\{\mathtt{t}_1 \Rightarrow \mathtt{e}''_1 \,[]\, \mathtt{t}_2 \Rightarrow \mathtt{e}''_2\} \triangleright \{\mathtt{k} : ?\mathtt{t}_1.\eta_1 \,+\, ?\mathtt{t}_2.\eta_2\}$,
- $\Gamma, \mathtt{y} : \mathtt{t}_0 \vdash_{\mathtt{r}} \mathscr{E}[\mathtt{y}] : \mathtt{t}_i \triangleright \Theta''_i, \mathtt{k} : \eta'$.

By the assumptions and Definitions C.13 we get that $\Theta(\tilde{\mathtt{k}}) \ltimes_h^{\mathtt{o}::\overline{\mathtt{o}}} (?\mathtt{t}_1.\eta_1 \,+\, ?\mathtt{t}_2.\eta_2) \circ \eta'$. Then by Lemma C.12 we have that $\Theta(\tilde{\mathtt{k}}) \ltimes_h^{\overline{\mathtt{o}}} \eta_j \circ \eta'$.
By rule RECEIVEC-TR we can derive $\Gamma, \mathtt{x} : \mathtt{t}_j \vdash_{\mathtt{r}} \mathtt{e}''_j : \mathtt{t}_0 \triangleright \mathtt{k} : \eta_j$. From $wf(\Gamma; h)$ and $h(\mathtt{o}) = (\mathtt{D}, _)$ and $\mathtt{D} \leq \mathtt{t}_j$ we get $\Gamma \vdash_{\mathtt{r}} \mathtt{o} : \mathtt{t}_j \triangleright \emptyset$ by applying AXIOM-RT and possibly SUB-RT. Then from Lemma C.9 we have that $\Gamma \vdash_{\mathtt{r}} [\mathtt{o}/\mathtt{x}]\mathtt{e}''_j : \mathtt{t}_0 \triangleright \mathtt{k} : \eta_j$, which implies $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[[\mathtt{o}/\mathtt{x}]\mathtt{e}''_j] : \mathtt{t}'_i \triangleright \Theta''_i, \mathtt{k} : \eta_j \circ \eta'$ by Lemma C.7. Lastly we get $\Gamma' = \Gamma$, and $\Theta'_i = \Theta''_i, \mathtt{k} : \eta_j \circ \eta'$ and $h'(\mathtt{k}) = \overline{\mathtt{o}\eta'}$, which assure $\Theta \sqsubseteq \Theta'$ and $wf(\Gamma'; \Theta'; h')$.

SENDCR. Assume $\mathtt{e}_i = \mathscr{E}[\mathtt{k}.\mathtt{sendC}(\mathtt{o})\{\mathtt{t}_1 \Rightarrow \mathtt{e}''_1 \,[]\, \mathtt{t}_2 \Rightarrow \mathtt{e}''_2\}]$. We have that

$$\mathscr{E}[\mathtt{k}.\mathtt{sendC}(\mathtt{o})\{\mathtt{t}_1 \Rightarrow \mathtt{e}''_1 \,[]\, \mathtt{t}_2 \Rightarrow \mathtt{e}''_2\}],\ h \longrightarrow \mathscr{E}[\mathtt{e}''_j],\ h'$$

where $h(\tilde{\mathtt{k}}) = \overline{\mathtt{o}}$, and $h(\mathtt{o}) = (\mathtt{D}, _)$, and $h' = h[\tilde{\mathtt{k}} \mapsto \overline{\mathtt{o}} :: \mathtt{o}]$, and $\mathtt{D} \leq \mathtt{t}_j$.
Since $\Gamma \vdash_{\mathtt{r}} \Theta_i : \mathtt{e}_i \triangleright \mathtt{t}'_i$, by rule SENDC-RT (observing that an object is typed with the empty session environment) and Lemma C.6 we must have for some Θ''_i, η_1, η_2, η', $\mathtt{t}_0$, and $\mathtt{y}$ fresh in $\mathscr{E}$:

- $\Theta_i = \Theta''_i, \mathtt{k} : (!\mathtt{t}_1.\eta_1 \,\oplus\, !\mathtt{t}_2.\eta_2) \circ \eta'$,
- $\Gamma \vdash_{\mathtt{r}} \mathtt{k}.\mathtt{sendC}(\mathtt{o})\{\mathtt{t}_1 \Rightarrow \mathtt{e}''_1 \,[]\, \mathtt{t}_2 \Rightarrow \mathtt{e}''_2\} : \mathtt{C}_0 \triangleright \{\mathtt{k} : !\mathtt{t}_1.\eta_1 \,\oplus\, !\mathtt{t}_2.\eta_2\}$,
- $\Gamma, \mathtt{y} :_{\mathtt{o}} \vdash_{\mathtt{r}} \mathtt{t}'_{\mathtt{i}} : \mathscr{E}[\mathtt{y}] \triangleright \Theta''_{\mathtt{i}}, \mathtt{k} : \eta'$.

By the assumptions and Definition C.13 we have that $(!\mathtt{t}_1.\eta_1 \,\oplus\, !\mathtt{t}_2.\eta_2) \circ \eta' \ltimes_h^{\overline{\mathtt{o}}} \Theta(\tilde{\mathtt{k}})$, which implies $\eta_j \circ \eta' \ltimes_h^{\overline{\mathtt{o}}::\mathtt{o}} \Theta(\tilde{\mathtt{k}})$ by Definition C.11.
By rule SENDC-RT we can derive $\Gamma \vdash_{\mathtt{r}} \mathtt{e}''_j : \mathtt{t}_0 \triangleright \mathtt{k} : \eta_j$, which implies $\Gamma \vdash_{\mathtt{r}} \mathscr{E}[\mathtt{e}''_j] : \mathtt{t}'_i \triangleright \Theta''_i, \mathtt{k} : \eta_j \circ \eta'$ by Lemma C.7. Lastly we get $\Gamma' = \Gamma$, and $\Theta'_i = \Theta''_i, \mathtt{k} : \eta_j \circ \eta'$ and $h'(\tilde{\mathtt{k}}) = \overline{\mathtt{o}} :: \mathtt{o}$, which assure $\Theta \sqsubseteq \Theta'$ and $wf(\Gamma'; \Theta'; h')$. □

It is handy to take into account the order on which subexpressions of the same expression are reduced. To this aim we introduce the "follows" relation between expressions. We denote by $\mathscr{C}$ an arbitrary context, while $\mathscr{E}$ is an evaluation context.

DEFINITION C.15. *Let* $\mathtt{e}$ *be an expression and* $\mathtt{e}_1, \mathtt{e}_2$ *be two subexpressions of* $\mathtt{e}$. *We say that* $\mathtt{e}_2$ follows $\mathtt{e}_1$ *in* $\mathtt{e}$ *if, for some arbitrary context* $\mathscr{C}$ *and evaluation context* $\mathscr{E}$, *we have that* $\mathscr{E}[\mathtt{e}_1] = \mathscr{C}[\mathtt{e}_2]$ *is a subexpression of* $\mathtt{e}$.

It is easy to check that, if $\mathtt{e}_1$ and $\mathtt{e}_2$ are as in previous definition, then $\mathtt{e}_1$ needs to be reduced before $\mathtt{e}_2$, since $\mathtt{e}_1$ occurs in the hole of an evaluation context $\mathscr{E}$, while $\mathtt{e}_2$ occurs in the hole of an arbitrary context $\mathscr{C}$, and $\mathscr{E}[\mathtt{e}_1]$, $\mathscr{C}[\mathtt{e}_2]$ are two different decompositions of the same expression.

We convene that all fresh channels created reducing parallel threads take successive indexes according to the order of creation, i.e. they are named $\mathtt{k}_0$, $\mathtt{k}_1, \ldots$. This means that if $P, h \longrightarrow^* Q, h' \longrightarrow^* R, h''$ and $\mathtt{k}_i$ is a channel created in the reduction $P, h \longrightarrow^* Q, h'$, and $\mathtt{k}_j$ is a channel created in the reduction $Q, h' \longrightarrow^* R, h''$, then $i < j$.

The *subject* of a communication expression is the channel specified in its syntax on which the communication takes place. The *index of a communication expression* is the index of its subject.

The following crucial lemma states that a channel and its dual cannot occur in the same thread. Moreover it states that the order on communication expression indexes agrees with the "follows" relation between expressions.

LEMMA C.16. *Let* $\emptyset \vdash \mathtt{e} : \mathtt{t} \triangleright \varepsilon$ *and* $\mathtt{e}, [\,] \longrightarrow^* \mathtt{e}_1 \mid\mid \ldots \mid\mid \mathtt{e}_n, h$. *Then:*

1. *no expression* $\mathtt{e}_i$ *can contain occurrences of both* $\mathtt{k}$ *and* $\tilde{\mathtt{k}}$ *for some channel* $\mathtt{k}$,
2. *if* $\mathtt{e}'$, $\mathtt{e}''$ *are communication subexpressions of* $\mathtt{e}_i$ *and* $\mathtt{e}''$ *follows* $\mathtt{e}'$, *then the index of* $\mathtt{e}'$ *is greater than or equal to the index of* $\mathtt{e}''$.

PROOF.

1. Straightforward, noting that the channels $\mathtt{k}$ and $\tilde{\mathtt{k}}$ are introduced by the rule SESSREQ-R in two different parallel threads.
2. $\emptyset \vdash \mathtt{e} : \mathtt{t} \triangleright \varepsilon$ implies that no channel occurs in $\mathtt{e}$ and so the property trivially holds. We now prove that the reduction preserves the property, namely if all the channels in the subexpressions of an expression are indexed in a not increasing order, starting from the redex to all the following expressions, in the sense of Definition C.15, then after one step of reduction we get expressions that have the same property. The proof is by case analysis on the definition of $\longrightarrow$.

   Case SESSREQ-R. We have that

$$\frac{h(\mathtt{o}) = (\mathtt{C}, _) \qquad \mathit{sbody}(\mathtt{s}, \eta', \mathtt{C}) = \mathtt{e}' \qquad \mathtt{k}, \tilde{\mathtt{k}} \notin h \qquad \eta \bowtie \eta'}{\mathscr{E}[\mathtt{o}.\mathtt{s}\,\{\mathtt{e} : \eta\}], h \longrightarrow \mathscr{E}[\mathtt{e} \wr \mathtt{k}\rceil] \mid\mid [\mathtt{o}/\mathtt{this}]\mathtt{e}' \wr \tilde{\mathtt{k}}\rceil, h[\mathtt{k}, \tilde{\mathtt{k}} \mapsto ()]}$$

   Let $\mathscr{E}[\mathtt{o}.\mathtt{s}\,\{\mathtt{e} : \eta\}]$ be an expression in which the desired property holds. After one step of reduction, in the expression $\mathtt{e} \wr \mathtt{k}\rceil$ the new channel $\mathtt{k}$ is the one with the highest index and no other channel occurs in it. Moreover all communication expressions occurring in $\mathtt{e} \wr \mathtt{k}\rceil$ precede all communication expressions in $\mathscr{E}$. Lastly note that by induction hypothesis the desired property holds for all communication subexpressions occurring in $\mathscr{E}$.

In parallel we have the expression $[\mathtt{o}/\mathtt{this}]\mathtt{e}' \wr \tilde{\mathtt{k}} \int$, where $\mathtt{e}'$ is a session body, so the only channel in this expression is $\tilde{\mathtt{k}}$. Then this reduction rule preserves the property.

Case SESSDEL-R**.** We have that

$$\frac{h(\mathtt{o}) = (\mathtt{C}, _) \qquad \mathit{sbody}(\mathtt{s}, \eta, \mathtt{C}) = \mathtt{e} \qquad \chi \leq \eta}{\mathscr{E}[\mathtt{o} \bullet \mathtt{s}\,\{\mathtt{k} : \chi\}], h \longrightarrow \mathscr{E}[[\mathtt{o}/\mathtt{this}]\mathtt{e} \wr \mathtt{k} \int], h}$$

Let $\mathscr{E}[\mathtt{o} \bullet \mathtt{s}\,\{\mathtt{k} : \chi\}]$ be an expression in which the desired property holds. Since $\mathtt{o} \bullet \mathtt{s}\,\{\mathtt{k} : \chi\}$ is the redex, then $\mathtt{k}$ is the channel with the highest index. After one step of reduction, $[\mathtt{o}/\mathtt{this}]\mathtt{e} \wr \mathtt{k} \int$ is the first expression to be reduced next, and $\mathtt{k}$ is still the only channel which occurs in it.

Case SENDCASE-R**.** We have that

$$\frac{h(\tilde{\mathtt{k}}) = \overline{\mathtt{o}} \qquad h(\mathtt{o}) = (\mathtt{C}, _) \qquad \mathtt{C} \leq \mathtt{t}_i}{\mathscr{E}[\mathtt{k}.\mathtt{sendC}(\mathtt{o})\{\mathtt{t}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{t}_2 \Rightarrow \mathtt{e}_2\}], h \longrightarrow \mathscr{E}[\mathtt{e}_i], h[\tilde{\mathtt{k}} \mapsto \overline{\mathtt{o}} :: \mathtt{o}]}$$

If the expression $\mathscr{E}[\mathtt{k}.\mathtt{sendC}(\mathtt{o})\{\mathtt{t}_1 \Rightarrow \mathtt{e}_1 \,[]\, \mathtt{t}_2 \Rightarrow \mathtt{e}_2\}]$ is an expression in which the desired property holds, then $\mathtt{k}$ is the channel with the highest index. The channel $\mathtt{k}$ is the only channel which occurs in the expressions $\mathtt{e}_1, \mathtt{e}_2$. Then, after one step of reduction the expression $\mathtt{e}_i$ can contain only the channel $\mathtt{k}$, that is the one with the highest index, or it can contain no channel, but the context is unchanged, then the property still holds.

Case RECEIVECASE-R**.** The proof is similar to the previous one.

In all the remaining case no channel is introduced or modified, then the property is trivially preserved. □

□

THEOREM (Progress). *If* $\emptyset \vdash \mathtt{e}_0 : \mathtt{t}_0 \,\S\, \varepsilon$ *and* $\mathtt{e}_0, [\,] \longrightarrow^* \mathtt{e}_1 \,||\, \ldots \,||\, \mathtt{e}_n, h$*, then for each* $\mathtt{e}_i$ $(1 \leq i \leq n)$ *one of the following conditions holds:*

- $\mathtt{e}_i, h \longrightarrow P, h'$ *for some* P, h'*,*
- $\mathtt{e}_i = \mathtt{o}$ *for some* $\mathtt{o}$*.*

PROOF. Toward a contradiction we assume that $\mathtt{e} \longrightarrow^* \mathtt{e}_1 \,|...|\, \mathtt{e}_n, h$ which is irreducible and is not a parallel composition of objects. By Lemma C.14 we have that there are Γ, $\Theta_1, ..., \Theta_n$ and $\mathtt{t}_1, ..., \mathtt{t}_n$ such that $wf(\Gamma; \Theta; h)$, where $\Theta = \bigcup_{1 \leq i \leq n} \Theta_i$ and $\Gamma \vDash_{\mathtt{r}} \Theta_i : \mathtt{e}_i \,\S\, \mathtt{t}_i$ for all $1 \leq i \leq n$. We can assume without loss of generality that, for $1 \leq m \leq n$, the expressions $\mathtt{e}_1, ..., \mathtt{e}_m$ are not objects. By Lemma C.14 we get $wf(\Theta; h)$. Then the evaluation of the expressions $\mathtt{e}_1, ..., \mathtt{e}_m$ can only be stopped by a receiving expression waiting for data in the associated channel. So for all $1 \leq l \leq m$ we must have $\mathtt{e}_l = E[\mathtt{e}'_l]$, where $\mathtt{e}'_l$ is a case receiving communication expression.

Let j be the highest among the indexes of the channels occurring in $\mathtt{e}_1 \,|...|\, \mathtt{e}_n$. If both $\mathtt{k}_j$ and $\tilde{\mathtt{k}_j}$ occur in $\mathtt{e}_1 \,|...|\, \mathtt{e}_n$, then by Lemma C.16(1) they occur in two different expressions, let them be $\mathtt{e}_p$ and $\mathtt{e}_q$ with $1 \leq p \neq q \leq m$. By Lemma C.16(2) the subjects of the two expressions $\mathtt{e}'_p$ and $\mathtt{e}'_q$ are the channels $\mathtt{k}_j$ and $\tilde{\mathtt{k}_j}$. Moreover we must have that $\Theta_p(\mathtt{k}_j)$, $\Theta_q(\tilde{\mathtt{k}_j})$ are of the forms $(?\mathtt{t}'_1.\eta_1 + ?\mathtt{t}'_2.\eta_2) \circ \eta$, $(?\mathtt{t}''_1.\eta'_1 + ?\mathtt{t}''_2.\eta'_2) \circ \eta'$, since $\mathtt{e}'_p$ and $\mathtt{e}'_q$ are case receiving expressions.

If $h(\mathtt{k}_j)$ is not empty, then let $h(\mathtt{k}_j) = \mathtt{o} :: \overline{\mathtt{o}'}$, and by Lemma C.12 $h(\mathtt{o}) = (\mathtt{C}, -)$ and $\mathtt{C} \leq \mathtt{t}'_i$ is defined, so $\mathtt{e}'_p$ can perform a RECEIVECASE-R step against the hypothesis. Similarly if $h(\tilde{\mathtt{k}_j})$ is not empty.

Otherwise, if both $h(\mathtt{k}_j)$ and $h(\tilde{\mathtt{k}_j})$ are empty, then by Lemma C.14 we get $wf(\Theta; h)$, which implies $\Theta_q(\tilde{\mathtt{k}_j}) \bowtie \Theta_p(\mathtt{k}_j)$. But this is impossible since $\Theta_p(\mathtt{k}_j)$ and $\Theta_q(\tilde{\mathtt{k}_j})$ are of the forms $(?\mathtt{t}'_1.\eta_1 + ?\mathtt{t}'_2.\eta_2) \circ \eta$, $(?\mathtt{t}''_1.\eta'_1 + ?\mathtt{t}''_2.\eta'_2) \circ \eta'$.

If only $\mathtt{k}_j$ occurs in $\mathtt{e}_1 \,|...|\, \mathtt{e}_n$, then we must have $\Theta(\tilde{\mathtt{k}_j}) = \epsilon$ and from $wf(\Theta; h)$ get that $h(\mathtt{k}_j)$ is not empty, and so we can argue as before. □ □