Università di Pisa
Dipartimento di Informatica

Technical Report: TR-08-05

# Types and deadlock free sessions for SCC

Roberto Bruni[1] Leonardo Gaetano Mezzina[2]

April 3, 2008

Address: Largo B. Pontecorvo 3, 56127 Pisa, Italy. Tel: +39 050 2212700 Fax: +39 050 2212726

# Types and deadlock free sessions for SCC*

Roberto Bruni[1] and Leonardo Gaetano Mezzina[2]

[1] Computer Science Department, University of Pisa, Italy
bruni@di.unipi.it
[2] IMT Lucca, Institute for Advanced Studies, Italy
leonardo.mezzina@imtlucca.it

**Abstract.** The notion of a session is fundamental in service oriented applications, as it serves to separate interactions between different instances of the same services, and to group together logical units of work. Recently, SCC has been proposed as a calculus centered around the concept of a dyadic session, where service interaction protocols and service orchestration can be conveniently expressed. In this paper we propose a generic type system to collect services' behaviors and then we fix a class of well typed processes that are guaranteed to be deadlock free. The type system is based on previous research on traditional mobile calculi, here conveniently extended and simplified thanks to the neat discipline imposed by the linguistic primitives of SCC.

## 1 Introduction

The success of service orientation is attracting the interest of both industry and academy. On the one hand, important standardization bodies and industrial consortia are developing the WS-* stack, targeting the engineering of web services technologies from a pragmatic perspective. The related documentation is often centered around typical usage patterns: it focuses on technical details of the implementation, but leaves several ambiguities on the overall methodology. On the other hand, several efforts are now posed on mathematical foundations, by developing formal languages and models tailored to services. The related documentation is aimed in first place to provide current standards with unambiguous semantics, but also to lay the basis for sound service orchestration, tackling the scenarios from a maybe more abstract perspective. Within this research thread, building on the experience maturated in the area of concurrency, many proposals of service oriented calculi have emerged [1, 4, 6–9, 13, 21, 22] (just to cite a few), where well-studied calculi have been enhanced with service-specific primitives.

The aim of this paper is to study a type system of one of such calculi. The ATM example in [16] describes the behavior of an automatic teller machine that can serve well to illustrate our main idea. The ATM offers three options to choose from: deposit, withdraw and balance. Once the user chooses one option the ATM establishes a new direct connection with the bank to account for the operation. Afterwards, the result is returned by the ATM to the user who can choose another option. At the type system level, even

* Research supported by the EU within the FET-GC II Integrated Project IST-2005-016004 SENSORIA and by the Italian FIRB Project TOCAI.IT.

if the connection with the bank is reiterated each time an user chooses an option, it is only necessary to check a single instance to guarantees safety of the communication. We show that fixing some communication patterns this type not only guarantees safety but a much stronger property such as deadlock freedom.

We study a session calculus derived from SCC [4] and its recent proposal CaSPiS [5] in which service invocation encompasses *one-way* and *request-response* [10–12, 3] but also allows for more sophisticated message exchanges, according to some fixed interaction protocol exposed by the service. The key feature is considering such messages as correlated, enclosed in special units of work, called *sessions*, and isolated from messages belonging to different invocations to the same service. Differently from other session languages [15, 16, 24] and also from languages based on correlation sets [22], here the programmers should not bother with the manipulation of sessions, as SCC chooses to instantiate sessions in an automatic way, upon service invocation. With respect to pi-calculus, in SCC the channel name expressing the communication media for exchanging messages is always implicit and determined by the context surrounding abstractions and concretions (i.e., the enclosing session/pipeline). Pipelines has been inspired by Cook and Misra's Orc [13], an elegant language for structured orchestration, where pipelines offer a basic mechanism for composing processes in a fashion that is more general and better suited w.r.t. concurrency than sequential composition.

Since sessions are automatically established and can be nested, the language is endowed with children-to-parent communications and with in-session communications. Only these two communication patterns not only are expressive enough but also typing a single instance of a session is suffice to guarantee deadlock freedom of recursive processes such as factorial service (see Example 2).

However, the resulting language is somehow too permissive to be dealt with using session types [15, 24, 16] directly. Session types permit to correctly typecheck sessions if a process has some constraints and, in particular, each input in a session must be matched by only an output. This condition is violated (and consequently subject reduction does not hold) if, for example, in the presence of an input we introduce parallel outputs (of different types of course). Our type system extends ordinary session types to work correctly with our language. Differently from [17], this permits for each variable to have a simple way for statically knowing its type in the case of the basic values or the service type in the case of service name mobility.

Thus, we investigate a subclass of SCC processes typical of the service oriented scenarios (e.g., service declarations are top level and replicated) for which we show the main theorem of this paper: we prove that these processes are deadlock free in the sense that they either diverge (by invoking new service instances and opening new sessions) or reach a normal form in which only service declarations remain; that is, every client terminates its computation unless someone diverges but the entire system cannot block on pending communications. The approach followed for the proof makes an extensive use of types to limit the number of possible cases, resulting less error prone.

A similar type system exists for SSCC (a variant of SCC based on named streams instead of pipelines) [21] that guarantees session safety. Streams introduce some sort of global buffers for extra-session communication and permit only a single type of value

for each stream. Likewise, the type system described in [2] resembles our type system but guarantees only session progress.

Concerning deadlock freedom we use definitions in [19, 20] as references for our definition which is slight different as it is adapted for the service oriented scenario.

*Outline of the paper.* Section 2 fixes the syntax and semantics of our SCC-like calculus. Section 3 presents the type system and the subject reduction result. Section 4 presents the class of initial processes and proves that every well typed and initial process is deadlock free. Section 5 summarizes the results and points out directions for further work. For referees' convenience, proofs omitted for space limitation are in Appendix.

## 2 Session Centered Calculus

### 2.1 Example

Before the formal presentation of our language we illustrate a simple example. Consider the following calculator service *calc*

$$C \;=\; calc.\Big(\ (\mathsf{sum}).(\mathtt{x},\mathtt{y}).\langle \mathtt{add}(\mathtt{x},\mathtt{y})\rangle \;+\; (\mathsf{inc}).(\mathtt{x}).\langle \mathtt{add}(\mathtt{x},1)\rangle\ \Big)$$

*calc* offers two operations, sum or inc, depending on the client choice. If the client after an invocation $\overline{calc}$ sends the label sum to the service then the service waits for a tuple of two integers $x$ and $y$, and after receiving them, sends their sum back to the client. Here add is a function only available on service side either directly implemented or built with other services (the reader might think of add to be replaced by an operation that requires either an access to local data or a computational power available only at the service side). Functions available to the service together with the base type are left unspecified, because we want to model the entire system with modularity; in the case of *calc* service, for example we want to hide the sum implementation. Assume the client

$$\begin{array}{l}\overline{calc}.\mathtt{if}\ (test)\ \mathtt{then}\ \langle \mathsf{inc}\rangle.\langle 1\rangle.(\mathtt{x}).\mathtt{return}\ \mathtt{x}\\ \qquad\qquad\quad \mathtt{else}\ \langle \mathsf{sum}\rangle.\langle 1,2\rangle.(\mathtt{y}).\mathtt{return}\ \mathtt{y}\end{array}$$

after invoking $\overline{calc}$ and depending on some internal condition *test*, it chooses between the two available options. The situation of the freshly established session $r$ after the client's choice of sum is the following:

$$(\nu\ r)\ \Big(\ r^{-} \triangleright \langle 1,2\rangle.(\mathtt{y}).\mathtt{return}\ \mathtt{y}\ \mid\ r^{+} \triangleright (\mathtt{x},\mathtt{y}).\langle \mathtt{add}(\mathtt{x},\mathtt{y})\rangle\ \mid\ C\ \Big)$$

where the client protocol is running on the left (the session with negative polarity) and the service protocol on the right (the session with positive polarity). Abstractions (e.g., $(x,y)$) and concretions (e.g., $\langle 1,2\rangle$) running on the opposite sides (service side $r^{+}$ and client side $r^{-}$) of the same session (bound name $r$) can exchange data, thus leading to

$$(\nu\ r)\ \Big(\ r^{-} \triangleright (\mathtt{y}).\mathtt{return}\ \mathtt{y}\ \mid\ r^{+} \triangleright \langle \mathtt{add}(1,2)\rangle\ \mid\ C\ \Big)$$

After another interaction the client side ($r^{-} \triangleright \mathtt{return}\ 3$) can return the result outside of $r$ (to the parent session, if any). We shall see in Section 3 that thanks to session types, the type system collects the usage in each session and compares it with the correspondent dual usage to guarantee sound session interactions.

| | | | |
|---|---|---|---|
| $P, Q, R$ | $::=$ | $\mathbf{0}$ | (nil) |
| | $\mid$ | $s.P$ | (service definition) |
| | $\mid$ | $\overline{v}.P$ | (invocation) |
| | $\mid$ | $\texttt{if}\ v = v_1\ \texttt{then}\ P\ \texttt{else}\ Q$ | (if-then-else) |
| | $\mid$ | $(\tilde{x}).P$ | (tuple input) |
| | $\mid$ | $\langle\tilde{v}\rangle.P$ | (value output) |
| | $\mid$ | $\Sigma_{i=1}^{n}(l_i).P_i$ | (label-guarded sum) |
| | $\mid$ | $\langle l\rangle.P$ | (label choice) |
| | $\mid$ | $\texttt{return}\ \tilde{v}.P$ | (return) |
| | $\mid$ | $r^p \triangleright P$ | (session) |
| | $\mid$ | $P > \tilde{x} > Q$ | (pipe) |
| | $\mid$ | $P \mid Q$ | (parallel) |
| | $\mid$ | $(\nu m)P$ | (restriction) |

| | | | |
|---|---|---|---|
| $v$ | $::=$ | $\mathsf{f}(\tilde{v})$ | (function call) |
| | $\mid$ | $x$ | (variable) |
| | $\mid$ | $m$ | (service/session) |
| | $\mid$ | $b$ | (basic data value) |

**Fig. 1.** Syntax of our service calculus.

### 2.2 Syntax

The set of processes is inductively defined by the grammar in Figure 1. We let $P, Q$ range over the processes, $s$ over service names, $r$ over session names, $m$ over both session and service names, $l$ over labels, $x$ over variables (over session names, services and data), and $v$ over values, which include an elsewhere specified set of basic data values and expressions (possibly with names, variables and functions). Operators are listed in decreasing order of precedence: $P > \tilde{x} > Q|R$ means $(P > \tilde{x} > Q)|R$ and $r \triangleright P > \tilde{x} > Q$ means $(r \triangleright P) > \tilde{x} > Q$.

As usual $[\tilde{\cdot}]$ identifies a tuple, $\mathbf{0}$ is the nil process, the trailing of $\mathbf{0}$ is often omitted, parallel composition is denoted by $P|Q$ and restriction by $(\nu m)P$. The construct $r^p \triangleright P$ indicates a generic session side with polarity $p$ (taking values in $\{+, -\}$). Nevertheless, differently from other languages that provide primitives for explicit session naming and creation, here all sessions could be built automatically, resulting in a more elegant and disciplined style of writing processes. In fact, a fresh session name $r$ and two polarized session ends $r^- \triangleright P$ and $r \triangleright^+ Q$ are generated (on client and service sides, respectively) upon each service invocation $\overline{s}.P$ of the service $s.Q$. Then, communications in $P$ and in $Q$ are uniquely directed toward the dual session side labeled with the same name $r$ but with opposite polarity. We use $p, q$ as meta variables to range over session polarities and $\overline{p}, \overline{q}$ are the opposite/dual polarities of $p$ and $q$., i.e.; $\overline{+} = -$ and $\overline{-} = +$.

Also labels can be exchanged, but they are just a way for expressing a choice on one side among a set of available options at the other side. The primitive `return` is used to output values to the parent session and the pipe $P > \tilde{x} > Q$ is a construct for on-side communications, i.e., for propagating values in the same side of a session. Pipe is inspired by Orc [13] to activate a fresh instance of $Q$ on any value produced by $P$. For example $\overline{calc}.\langle\mathsf{inc}\rangle.\langle 2\rangle.(x).\texttt{return}\ x > y > Q$ invokes *calc* and deliver the result in $y$ to $Q$. Since after a prefix of invocation a new session will be created, the pipe is the easiest way to continue the computation within the established session.

$$
\begin{array}{c}
P|\mathbf{0} \equiv P \qquad P|Q \equiv Q|P \qquad (P|Q)|R \equiv P|(Q|R) \\
(\nu m)\mathbf{0} \equiv \mathbf{0} \qquad (\nu m_1)(\nu m)P \equiv (\nu m)(\nu m_1)P \qquad ((\nu m)P)|Q \equiv (\nu m)(P|Q) \ \text{ if } m \notin \mathsf{fn}(Q) \\
(\nu m)P > \tilde{x} > Q \equiv (\nu m)(P > \tilde{x} > Q) \ \text{ if } m \notin \mathsf{fn}(Q) \qquad r^p \triangleright (\nu m)P \equiv (\nu m)(r^p \triangleright P) \ \text{ if } r \neq m \\
\mathbf{0} > \tilde{x} > P \equiv \mathbf{0} \qquad (P|Q) > \tilde{x} > R \equiv (P > \tilde{x} > R)|(Q > \tilde{x} > R) \\
(r^p \triangleright \mathbf{0}) > \tilde{x} > R \equiv r^p \triangleright \mathbf{0} \qquad r_1^p \triangleright (r_2^q \triangleright \mathbf{0}|Q) \equiv r_1^p \triangleright Q|r_2^q \triangleright \mathbf{0} \qquad (\nu r)(r^+ \triangleright \mathbf{0}|r^- \triangleright \mathbf{0}) \equiv \mathbf{0}
\end{array}
$$

**Fig. 2.** Structural congruence.

$$
\begin{array}{rrrl}
\lambda ::= & : s \Rightarrow r & \mid \quad s \Leftarrow r & \text{(service invocation/definition)} \\
\mid & \rightarrow \tilde{v} & \mid \quad \leftarrow \tilde{v} & \text{(value production/consumption)} \\
\mid & r^p :\rightarrow \tilde{v} & \mid r^p :\leftarrow \tilde{v} & \text{(production/consumption within } r\text{)} \\
\mid & \rightarrow l & \mid \quad \leftarrow l & \text{(choice selection/branching)} \\
\mid & r^p :\rightarrow l & \mid r^p :\leftarrow l & \text{(selection/branching within } r\text{)} \\
\mid & \tau & \mid \quad r\tau & \text{(silent steps)} \\
\mid & \uparrow \tilde{v} & \mid \quad (m)\lambda & \text{(value return/extrusion)} .
\end{array}
$$

**Fig. 3.** Labels of the transition system.

Processes are taken up to alpha-equivalence considering $(\tilde{x}).P$ and $P > \tilde{x} > Q$ as binders for variables $\tilde{x}$ and $(\nu m)P$ as the binder for session / service name $m$. Consequently, the notion of free names $\mathsf{fn}(P)$ and bound names $\mathsf{bn}(P)$ are defined as expected. It is worth noting that the operation of substitution $P[\tilde{v}/\tilde{x}]$, which replaces a tuple of variables with a tuple of values, considers that variables cannot appear in certain positions (i.e. that $x \triangleright P$ and $x.P$ are forbidden by the syntax). The effect of the substitution over the other processes is the standard capture avoiding substitution.

Each service definition is persistent (i.e., not consumed after an invocation). For this reason their protocols are not supposed to return any value to the parent in the type system given in Section 3.

### 2.3 Operational Semantics

We describe the semantics of our language by means of an LTS that exploits a structural congruence. The structural congruence $\equiv$ is the least relation defined by the equations in Figure 2. The structural congruence includes ordinary axioms about parallel and restriction, together with distributivity of parallel over pipes, and a few axioms for garbage collecting terminated session ends $r^p \triangleright \mathbf{0}$.

Our transition system exploits the labels $\lambda$ in Figure 3. We write $(m)\lambda$ to mean the label $\lambda$ where the name $m$ becomes bound (see Figure 4). We write $\leftrightarrow$ to mean either $\leftarrow$ or $\rightarrow$.

The semantics is given in the early style, which guesses the values and labels in the rules (In) and (Branch), respectively. Rule (Def) shows the replicated nature of the service and together with (Inv) creates two processes which are ready to communicate after that (Scom) creates a new shared common session. (SessionOut) accounts for the return of a value, which is converted in an output out of the current session when the session construct is traversed. (Session) marks with the name of the exchanging session

$$
\begin{aligned}
\mathsf{bn}((m)\lambda) &= \mathsf{bn}(\lambda) \cup m \\
\mathsf{bn}(\lambda) &= \emptyset \qquad\qquad \text{if } \lambda \neq (m)\lambda'
\end{aligned}
$$

**Fig. 4.** Bound names for labels.

(In)
$$(\tilde{x}).P \xrightarrow{\leftarrow \tilde{v}} P[\tilde{v}/\tilde{x}]$$

(Out)
$$\langle \tilde{v} \rangle.P \xrightarrow{\rightarrow \tilde{v}} P$$

(Session)
$$\frac{P \xrightarrow{\lambda} P' \quad \lambda \in \{\leftrightarrow \tilde{v}, \leftrightarrow l, \tau\}}{r^p \triangleright P \xrightarrow{r^p:\lambda} r^p \triangleright P'}$$

(Branch)
$$\Sigma_{i=0}^{n}(l_i).P_i \xrightarrow{\leftarrow l_i} P_i$$

(Choice)
$$\langle l \rangle.P \xrightarrow{\rightarrow l} P$$

(Select)
$$\frac{P \xrightarrow{r^p:\rightarrow l} P' \quad Q \xrightarrow{r^{\overline{p}}:\leftarrow l} Q'}{P|Q \xrightarrow{r\tau} (P'|Q')}$$

(Def)
$$\frac{r \notin \mathsf{fn}(s.P)}{s.P \xrightarrow{(r)s \Leftarrow r} r^+ \triangleright P|s.P}$$

(Inv)
$$\frac{r \notin \mathsf{fn}(\overline{s}.P)}{\overline{s}.P \xrightarrow{(r)s \Rightarrow r} r^- \triangleright P}$$

(SCom)
$$\frac{P \xrightarrow{(r)s \Rightarrow r} P' \quad Q \xrightarrow{(r)s \Leftarrow r} Q'}{(P|Q) \xrightarrow{\tau} (\nu r)(P'|Q')}$$

(Ret)
$$\texttt{return}\ \tilde{v}.P \xrightarrow{\uparrow \tilde{v}} P$$

(SessionOut)
$$\frac{P \xrightarrow{\uparrow \tilde{v}} P'}{r^p \triangleright P \xrightarrow{\rightarrow \tilde{v}} r^p \triangleright P'}$$

(NestInv)
$$\frac{P \xrightarrow{(r')s \Rightarrow r'} P' \quad r \neq r'}{r^p \triangleright P \xrightarrow{(r')s \Rightarrow r'} r^p \triangleright P'}$$

(Comm)
$$\frac{P \xrightarrow{r^p:\leftarrow \tilde{v}} P' \quad Q \xrightarrow{r:\overline{p} \rightarrow \tilde{v}} Q'}{P|Q \xrightarrow{r\tau} (P'|Q')}$$

(Open)
$$\frac{P \xrightarrow{\lambda} P' \quad \lambda \in \{\rightarrow \tilde{v}, r^p :\rightarrow \tilde{v}\} \wedge s \in \mathsf{n}(\tilde{v})}{(\nu s)P \xrightarrow{(s)\lambda} P'}$$

(Res)
$$\frac{P \xrightarrow{\lambda} P' \quad s \notin \mathsf{n}(\lambda)}{(\nu s)P \xrightarrow{\lambda} (\nu s)P'}$$

(SessRes)
$$\frac{P \xrightarrow{r\tau} P'}{(\nu r)P \xrightarrow{\tau} (\nu r)P'}$$

(Pipe)
$$\frac{P \xrightarrow{\rightarrow \tilde{v}} P'}{P > \tilde{x} > Q \xrightarrow{\tau} Q[\tilde{v}/\tilde{x}]|(P' > \tilde{x} > Q)}$$

(PipePass)
$$\frac{P \xrightarrow{\lambda} P' \quad \lambda \neq \rightarrow \tilde{w}}{P > \tilde{x} > Q \xrightarrow{\lambda} P' > \tilde{x} > Q}$$

(IfL)
$$\frac{P \xrightarrow{\lambda} P'}{\texttt{if}\ v = v\ \texttt{then}\ P\ \texttt{else}\ Q \xrightarrow{\lambda} P'}$$

(IfR)
$$\frac{v_1 \neq v \quad Q \xrightarrow{\lambda} Q'}{\texttt{if}\ v = v_1\ \texttt{then}\ P\ \texttt{else}\ Q \xrightarrow{\lambda} Q'}$$

(Par)
$$\frac{P \xrightarrow{\lambda} P' \quad \mathsf{bn}(\lambda) \cap \mathsf{fn}(Q) = \emptyset}{P|Q \xrightarrow{\lambda} P'|Q}$$

(Equiv)
$$\frac{P \xrightarrow{\lambda} P' \quad P \equiv Q \quad P' \equiv Q'}{Q \xrightarrow{\lambda} Q'}$$

**Fig. 5.** Operational semantics.

$$
\begin{array}{rcll}
T, U & ::= & \mathsf{end} & \text{(no action)} \\
 & | & ?(S_1, \ldots, S_n).T & \text{(input of a tuple)} \\
 & | & !(S_1, \ldots, S_n).T & \text{(output of a tuple)} \\
 & | & \&\{l_1 : T_1, \ldots, l_n : T_n\} & \text{(external choice)} \\
 & | & \oplus\{l_1 : T_1, \ldots, l_n : T_n\} & \text{(internal choice)} \\
 & & & \\
S & ::= & [T] & \text{(session)} \\
 & | & \mathcal{B} & \text{(basic data types)}
\end{array}
$$

**Fig. 6.** Syntax of types.

each operation in that session. (Comm) permits both communication of basic values and service names. Extrusion is handled by (Open) and (Par), but thanks to (Equiv) restricted names can be moved to the top before communication and a closure rule is not necessary. On the other hand, side condition of rule (Par) is useful for session floating since $r$ is bound in labels for the service invocation.

Rule (Pipe) creates a new concurrent copy of process $Q$ together with the residual $P' > \tilde{x} > Q$ in the case that $P$ outputs a value. Rule (PipePass) makes a move in $P$ if the action is not an output.

## 3 Typing

The set of session types, $U$,$T$,..., is defined by the grammar in Figure 6. Session types express sequences of typed tuples of input and output. Internal choice $\oplus$ records all the possible process choices at a certain point of a session, generated by means of an `if` instruction, like in the calculator example above. External choice & records the types of each offered options.

Sorts $S$ can be either $[T]$ which represents a session with session type $T$ or an elements of a given set of basic data types $\mathcal{B}$. By convention we point with $\tau_b$ the type of the basic value $b$ and $\tau_b \in \mathcal{B}$.We shall assume that $int \in \mathcal{B}$.

Our set of type judgments is in Figure 7. Type judgments for values take the form $\Gamma \vdash v : S$ where $S$ represents the sort of $v$. Type judgments for processes take the form $\Gamma \vdash P : U[T]$ where the type $U$ represents the outputs of $P$ to the parent session, while $T$ is the type of current session usage. Sometimes we write only $[T]$ as a shorthand for $\mathsf{end}[T]$. The *type environment* $\Gamma$ is a finite partial mapping from variables and services to sorts and function types. As usual the empty environment is annotated $\emptyset$. When $x \notin dom(\Gamma)$ (same holds for $m \notin dom(\Gamma)$) we write $\Gamma, x : S$ for the type environment obtained by extending $\Gamma$ with the binding of $x$ to $S$.

With respect to the two typing systems presented in [24] our type system is more like the second one, with balanced typing. This can be easily viewed observing the shape of the rule for service binder (Tnew).

The first four rules for values are standard and the signature of each used external function must be inserted in the environment as a functional type (rule (FuncV)) because they are not bound by the process.

$$\text{(Service)}\ \Gamma, s : S \vdash s : S \qquad \text{(Var)}\ \Gamma, x : S \vdash x : S \qquad \text{(BasV)}\ \frac{\tau_b \in \mathcal{B}}{\Gamma \vdash b : \tau_b} \qquad \text{(Func)}\ \frac{\Gamma \vdash v_1 : S_1 \ldots \Gamma \vdash v_n : S_n \quad \tau_b \in \mathcal{B}}{\Gamma, \mathsf{f} : S_1 \times \ldots \times S_n \to \tau_b \vdash \mathsf{f}(v_1, \ldots, v_n) : \tau_b}$$

$$\text{(Tzero)}\ \Gamma \vdash \mathbf{0} : \mathsf{end}[\mathsf{end}] \qquad \text{(Tdef)}\ \frac{\Gamma \vdash P : \mathsf{end}[T] \quad \Gamma \vdash s : [T]}{\Gamma \vdash s.P : \mathsf{end}[\mathsf{end}]} \qquad \text{(Tinv)}\ \frac{\Gamma \vdash P : U[T] \quad \Gamma \vdash v : [\overline{T}]}{\Gamma \vdash \overline{v}.P : \mathsf{end}[U]}$$

$$\text{(Tin)}\ \frac{\Gamma, \tilde{x} : \tilde{S} \vdash P : U[T]}{\Gamma \vdash (\tilde{x}).P : U[?(\tilde{S}).T]} \qquad \text{(Tout)}\ \frac{\Gamma \vdash P : U[T] \quad \Gamma \vdash \tilde{v} : \tilde{S}}{\Gamma \vdash \langle \tilde{v} \rangle.P : U[!(\tilde{S}).T]} \qquad \text{(Tret)}\ \frac{\Gamma \vdash P : U[T] \quad \Gamma \vdash \tilde{v} : \tilde{S}}{\Gamma \vdash \mathtt{return}\ \tilde{v}.P : !(\tilde{S}).U[T]}$$

$$\text{(Tbranch)}\ \frac{I \subseteq \{1, \ldots, n\} \quad \forall i \in I \quad \Gamma \vdash P_i : U[T_i]}{\Gamma \vdash \Sigma_{i=0}^{n}(l_i).P_i : U[\&\{l_i : T_i\}]_{i \in I}} \qquad \text{(TChoice)}\ \frac{l = l_i \in \{l_1, \ldots, l_n\} \quad \Gamma \vdash P : U[T_i]}{\Gamma \vdash \langle l \rangle.P : U[\oplus\{l_1 : T_1, \ldots, l_n : T_n\}]}$$

$$\text{(TparL)}\ \frac{\Gamma \vdash P : U[T] \quad \Gamma \vdash Q : U'[\mathsf{end}]}{\Gamma \vdash P|Q : U \circ U'[T]} \qquad \text{(TparR)}\ \frac{\Gamma \vdash P : U[\mathsf{end}] \quad \Gamma \vdash Q : U'[T]}{\Gamma \vdash P|Q : U \circ U'[T]}$$

$$\text{(Tpipe)}\ \frac{\Gamma \vdash P : U[T] \quad \Gamma, \tilde{x} : \tilde{S} \vdash Q : U'[T'] \quad \mathsf{pipe}(U[T], U'[T'], \tilde{S}) = U''[T'']}{\Gamma \vdash P > \tilde{x} > Q : U''[T'']}$$

$$\text{(Tses)}\ \frac{\Gamma \vdash P : U[T]}{\Gamma, r : [T] \vdash r^{+} \triangleright P : \mathsf{end}[U]} \qquad \text{(TsesI)}\ \frac{\Gamma \vdash P : U[T]}{\Gamma, r : [\overline{T}] \vdash r^{-} \triangleright P : \mathsf{end}[U]}$$

$$\text{(Tnew)}\ \frac{\Gamma, m : S \vdash P : U[T] \quad \mathsf{exists}(m, P)}{\Gamma \vdash (\nu m)P : U[T]} \qquad \text{(Tif)}\ \frac{\Gamma \vdash v_i : S_i \quad i = 1, 2 \quad \Gamma \vdash P : U[T] \quad \Gamma \vdash Q : U[T]}{\Gamma \vdash \mathtt{if}\ v_1 = v_2\ \mathtt{then}\ P\ \mathtt{else}\ Q : U[T]}$$

**Fig. 7.** Typing rules.

The type of **0** in (Tzero) is $\mathsf{end}[\mathsf{end}]$ since no action is performed neither in the current session nor towards the parent session. Rule (Tdef) constrains the protocol of the service to be the same as the body process $P$ and rule (Tinv) checks that invoked service behaves in the dual manner with respect to the client. Here the dual of $T$, written $\overline{T}$ is inductively defi ned as:

$$\overline{?(\tilde{S}).T} = !(\tilde{S}).\overline{T} \qquad \overline{!(\tilde{S}).T'} = ?(\tilde{S}).\overline{T'} \qquad \overline{\mathsf{end}} = \mathsf{end}$$
$$\overline{\&\{l_1 : T_1, \ldots, l_n : T_n\}} = \oplus\{l_1 : \overline{T_1}, \ldots, l_n : \overline{T_n}\}$$
$$\overline{\oplus\{l_1 : T_1, \ldots, l_n : T_n\}} = \&\{l_1 : \overline{T_1}, \ldots, l_n : \overline{T_n}\}$$

Rules (Tin), (Tout) and (Tret) insert the usage type in the correct place. The type for the input variable $\tilde{x}$ in rule (Tin) is not specifi ed in the syntax but can be inferred with the help of the algorithm described in [23]. Rule (Tbranch) considers in the fi nal type only the effective branches used by all the dual sessions in the rule (Tchoice). In

fact, the `if` construct allows to choose between many branches at the same time and also different clients can invoke the same service making their own choices. The two rules for parallel composition (TparL) and (TparR) allow parallel composition of two processes only if at least one does not have any action in the current session, i.e. it has a type $U[\mathtt{end}]$. Note instead that both $P$ and $Q$ are allowed to produce values upwards, in which case the operation $U \circ U'$ is defined only if all the atomic actions in $U$ and $U'$ are of the same kind $!(\tilde{S})$, and in that case $U \circ U' = U.U' = U' \circ U$. This operation is sound because tail outputs of parallel values of the same type are not observable at the type system level. Rule (Tpipe) uses the function $\mathsf{pipe}(U[T], U'[T'], \tilde{S})$ defined as

$$\begin{array}{ll} \mathsf{pipe}(U[\mathtt{end}], U'[T'], \tilde{S}) & = U[\mathtt{end}] \\ \mathsf{pipe}(U[!(\tilde{S})], U'[T], \tilde{S}) & = U \circ U'[T] \\ \mathsf{pipe}(U[!(\tilde{S})^k], U'[\mathtt{end}], \tilde{S}) & = U \circ U'^k[\mathtt{end}] \quad k > 1 \end{array}$$

Intuitively $\mathsf{pipe}$ constrains $P > \tilde{}x > Q$ in the current session to allow a single output $P$ whenever the type of $Q$ is different from a single input or vice versa $Q$ to be a single input whenever $P$ is $!(\tilde{S})^k$ *a sequence of k-times* $!(\tilde{S})$. In this case the result is visible upward replying $k$ times $U'$. The first case of $\mathsf{pipe}$ is necessary to guarantee the subject reduction. (Tses) and (TsesI) are similar to service definition and invocation rules but $r$ is removed from the type environment which means that nesting of the same session name is forbidden.

Rule (Tnew) checks a corresponding service definition by means of $\mathsf{exists}(m, P)$. Function $\mathsf{exists}$ ensures that the process $P$ declares, in case $m$ is a service, the announced service and the inductive definition is as expected. Finally, rule (Tif) handles conditionals in the usual way.

Typing rules allow a deduction for processes like $r : !(int) \vdash r^+ \triangleright \langle 1 \rangle | r^+ \triangleright \langle 2 \rangle$ which do not preserve session linearity. We will exclude such processes by forbidding the direct usage of session constructs (see Definition 3).

*Example 1.* Let us take the calculator example. The type $!(int)[!(int, int).?(int)]$ expresses the following client usage: the output of two integers is followed by the reading of the result and an integer is returned outside the session (the first output out of the square brackets indicates a return action, that is an output out of the current session). Previous usage is compared with the session usage $?(int, int).!(int)$ to ensure that the invocation is sound. Below we report the typing proof for the client, where we let $\Gamma = calc : [\&\{\mathsf{sum} : ?(int, int).!(int), \mathsf{inc} : ?(int).!(int)\}]$, $P = \langle \mathsf{inc} \rangle.\langle 1 \rangle.(x).\mathtt{return}\ x$, $Q = \langle \mathsf{sum} \rangle.\langle 1, 2 \rangle.(x).\mathtt{return}\ x$ and $T = \oplus\{\mathsf{sum} : !(int, int).?(int), \mathsf{inc} : !(int).?(int)\}$:

$$\dfrac{\dfrac{\Gamma \vdash \langle 1 \rangle.(x).\mathtt{return}\ x : !(int)[!(int).?(int)]}{\Gamma \vdash \langle \mathsf{inc} \rangle.\langle 1 \rangle.(x).\mathtt{return}\ x : !(int)[T]} \qquad \dfrac{\Gamma \vdash \langle 1, 2 \rangle.(x).\mathtt{return}\ x : !(int)[!(int, int).?(int)]}{\Gamma \vdash \langle \mathsf{sum} \rangle.\langle 1, 2 \rangle.(x).\mathtt{return}\ x : !(int)[T]}}{\Gamma \vdash \mathtt{if}\ (test)\ \mathtt{then}\ P\ \mathtt{else}\ Q : !(int)[T]}$$

Moreover, we could safely replace the service definition with

$$calc.\Big(\mathtt{(sum).(x,y).\langle add(x,y)\rangle\ +\ (inc).(x).\langle add(x,1)\rangle\ +\ (sub).(x,y).\langle minus(x,y)\rangle}\Big)$$

(which extends the previous version of the service with additional behaviors) and still correctly type check the client. In fact, our type system can statically exclude the new branch $\mathsf{sub}$ when the client is typed.

*Example 2.* Let us consider the factorial service $fatt$, defined by:

$$fatt.(\mathtt{n}).\ \mathtt{if}\ (\mathtt{n}{=}{=}0)\ \mathtt{then}\ \langle 1\rangle \\ \qquad\qquad \mathtt{else}\ \overline{fatt}.\langle \mathtt{n}\text{-}1\rangle.(\mathtt{x}).\mathtt{return}\ \mathtt{x} > \mathtt{x} > \langle \mathtt{mul}(\mathtt{x},\mathtt{n})\rangle$$

Notice that in this case we are able to express the factorial thanks to service persistence, which guarantees a separation between each invocation. The entire program is well typed by type checking only a single session instance. As the Theorem 2 will show, this check suffices to ensure that $fatt$ is deadlock free. The typing proof is below, where we recall that $\mathtt{end}[!(int)] = [!(int)]$ and let $P = \overline{fatt}.\langle n-1\rangle.(x).\mathtt{return}\ x$, $Q' = \langle \mathtt{mul}(x,n)\rangle$, $Q = P > x > Q'$ and $\Gamma = fatt : [?(int).!(int)], n : int, \mathtt{mul} : int \times int \to int$

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash \langle n-1\rangle.(x).\mathtt{return}\ x : !(int)[!(int).?(int)]}{\Gamma \vdash \overline{fatt}.\langle n-1\rangle.(x).\mathtt{return}\ x : [!(int)]} \qquad \Gamma, x : int \vdash Q' : [!(int)]}{\Gamma \vdash P > x > Q' : [!(int)]} \qquad \emptyset \vdash \langle 1\rangle : [!(int)]}{\Gamma \vdash \mathtt{if}\ (n == 0)\ \mathtt{then}\ \langle 1\rangle\ \mathtt{else}\ Q : \mathtt{end}[!(int)]}}{fatt : [?(int).!(int)], \mathtt{mul} : \ldots \vdash (n).\mathtt{if}\ (n == 0)\ \mathtt{then}\ \langle 1\rangle\ \mathtt{else}\ Q : \mathtt{end}[?(int).!(int)]}}{fatt : [?(int).!(int)], \mathtt{mul} : \ldots \vdash fatt.(n).\mathtt{if}\ (n == 0)\ \mathtt{then}\ \langle 1\rangle\ \mathtt{else}\ Q : \mathtt{end}[\mathtt{end}]}$$

*Example 3.* Beyond the basic type, expressions may use the name of a service as parameter. Take a load balancing service that is called to discover, at each invocation, which service between $a$ and $b$ is more reliable for executing $P$.

$$(\nu\ a\ b)\ \Big(\ loadbalance.\mathtt{if}\ \mathtt{choose}(a,b) = 1\ \mathtt{then}\ \langle a\rangle\ \mathtt{else}\ \langle b\rangle\ \mid\ a.P\ \mid\ b.P\ \Big)$$

Here the function `choose` is a basic expression of type $[T] \times [T] \to int$ and uses the names of the two services as parameters. The client after receiving the name of the reliable service can substitute it for $x$ for all future invocations of the service.

$$\overline{loadbalance}.(x).\mathtt{return}\ x > x > \overline{x}.Q$$

It is a nice exercise to verify that the ensemble of the above processes is well-typed (assuming $P$ and $Q$ have types $[T]$ and $[\overline{T}]$ respectively).

The type system enjoys subject congruence and subject reduction. In their proof we need some auxiliary lemmas.

**Lemma 1 (Weakening).** *If $\Gamma \vdash P : U[T]$ and $m \notin \mathsf{fn}(P)$ then $\Gamma, m : S \vdash P : U[T]$.*

*Proof.* Straightforward induction on the derivation of the typing judgment, applying $\alpha$-conversion when needed. □

**Lemma 2 (Strengthening).** *If $\Gamma, m : S \vdash P : U[T]$ and $m \notin \mathsf{fn}(P)$ then $\Gamma \vdash P : U[T]$.*

*Proof.* Straightforward induction on the derivation of the typing judgment. □

**Proposition 1 (Subject Congruence).** *If $\Gamma \vdash P : U[T]$ and $P \equiv Q$ then $\Gamma \vdash Q : U[T]$*

**Lemma 3 (Substitution).** *Let $\Gamma, x : S \vdash P : U[T]$. If $\Gamma \vdash v : S$ then $\Gamma \vdash P[^{v}/_{x}] : U[T]$.*

**Theorem 1 (Subject reduction).** *Let $\subseteq$ denote the smallest partial order on types such that: $T \subseteq ?(\tilde{S}).T$, $T \subseteq !(\tilde{S}).T$, $T_i \subseteq \&\{l_1 : T_1, \ldots, l_n : T_n\}$ and $T_i \subseteq \oplus\{l_1 : T_1, \ldots, l_n : T_n\}$.*

- *If $\Gamma \vdash P : U[T]$ and $P \xrightarrow{\tau} Q$ then $\Gamma \vdash Q : U[T]$.*
- *If $\Gamma, r : S \vdash P : U[T]$ and $P \xrightarrow{r\tau} Q$ then $\Gamma, r : S' \vdash Q : U[T]$ where $S' \subseteq S$ and $[T'] \subseteq [T]$.*

In the proof of subject reduction we use the strengthening lemma to remove an assumption about a session from $\Gamma$ and the weakening lemma to insert a session with modified type. We use the strengthening lemma to replace $\Gamma'$ with $\Gamma$ every time that $\Gamma' \subseteq \Gamma$ holds, and the weakening lemma to replace $\Gamma'$ with $\Gamma$ every time that $\Gamma \subseteq \Gamma'$.

## 4 Deadlock freedom and normal form

In this section we show that a large subclass of well typed processes is deadlock free, in the sense that certain communications will eventually succeed (unless the process diverges).

Hereafter, $\xrightarrow{\omega}{}^*$ represents the reflexive and transitive closure of $\xrightarrow{\tau}$ and $\omega, \gamma \ldots$ is a (possibly empty) sequence of labels. We developed a notion of deadlock-freedom suitable for our processes. In particular, a process is deadlock-free if it cannot be blocked waiting a synchronization unless it reaches the *normal form*. Normal form means that all the possible communications are exhausted and only service definitions remain.

**Definition 1 (Normal form).** *A process $P$ is in normal form if $P \equiv (\nu s_1) \ldots (\nu s_n)(\Pi_1^n s_i.Q_i)$.*

**Definition 2 (Deadlock-free).** *A process $P$ is deadlock-free if for each $Q$ s.t. $P \xrightarrow{\omega}{}^* Q$ then either $Q \xrightarrow{\tau}$ or $Q$ reaches the normal form.*

Since deadlock-freedom is a strong property we necessarily need to restrict the set of possible processes. Thus we define a notion of initial process.

**Definition 3 (Initial process).** *A process $P$ is initial if it does not contain session constructs and all service definitions are at the top level.*

The main theorem states deadlock freedom of all initial processes $P$ such that $\emptyset \vdash P : \mathsf{end}[\mathsf{end}]$ (see Theorem 2).

Before proving the main theorem we need some auxiliary definitions and results. Next lemma shows that only "safe" programs are produced starting from an initial process, that is situation like $r^p \triangleright \langle 1 \rangle | r^p \triangleright \langle 1 \rangle$ cannot arise at runtime. In fact, the process is an example for which session linearity is not respected since $r^p$ is used twice.

**Lemma 4 (Linear sessions).** *Let $P$ an initial process. If $P \xrightarrow{\omega}{}^* Q$, then for each session $r$ in $Q$, exactly two session constructs for $r$ appear in $Q$ with opposite polarities and there is a restriction $(\nu r)$ binding them.*

*Proof.* The proofs of the lemma is by induction on the length of the computation. □

Proving deadlock freedom involves the possibility of exhibiting a suitable $\tau$ reduction after an arbitrary number of evaluation steps. However, to characterize the next computational reduction in a constructive way, we need to argue about a specific session. To make it observable the session in which a synchronization is taking place we need a mild modification to the rule (SCom) of the transition system. The basic idea is to predicate over $P_{\text{ses}}$ instead of $P$, for $P \equiv (\nu\tilde{\ }r)P_{\text{ses}}$, i.e. where $P_{\text{ses}}$ has no binder on session names. Similarly, when $P \xrightarrow{\tau} Q$ we want to reason on the underlying reduction from $P_{\text{ses}}$ to $Q_{\text{ses}}$. The revised rule, called (Scom'), behaves like the original but does not restrict the session with a binder and uses the label $r\iota$ to identify the current session.

$$\text{(SCom')}\quad \frac{P \xrightarrow{(r)s\Rightarrow r} P' \quad Q \xrightarrow{(r)s\Leftarrow r} Q'}{(P|Q) \xrightarrow{r\iota} (P'|Q')}$$

The modification of the rule is sound since we are considering only initial processes or processes that represent states in the evaluation of an initial process. In fact, by Lemma 4 any initial process will produce for each new session a corresponding binder to restrict the session. For this reason the LTS with rule (Scom') has essentially the same behavior of the previous LTS with rule (Scom); that is we can rename both labels $r\iota$ and $r\tau$ (generated by communications) in $\tau$. Accordingly, in the new system $\omega, \gamma$ are labels $\tau$,$r\tau$ and $r\iota$. Another subtle aspect is that now session $r$ is bound in the label $r\tau$ and $r\iota$, because we need $r$ to be fresh w.r.t. all the other pre-existing session names, but we omit parentheses in favor of a lighter syntax. On the other hand, the type environment $\Gamma$ for closed processes can contain assumptions about session names.

We also need to reason inductively on the way sessions are nested and introduce the notion of a *context* as a convenient notation.

**Definition 4 (Contexts).** *The set of contexts is inductively defined by:*

$$\begin{array}{lcl} \mathbb{C} & ::= & [\![\cdot]\!] \mid \mathbb{C}|P \mid r^p \triangleright \mathbb{C} \mid (\nu s)\mathbb{C} \mid \mathbb{C} > \tilde{\ }x > P \\ \mathbb{C}_{r^p} & ::= & r^p \triangleright [\![\cdot]\!]|P \end{array}$$

As usual $\mathbb{C}[\![P]\!]$ and $\mathbb{C}_{r^p}[\![P]\!]$ are the processes obtained by replacing the hole in $\mathbb{C}$ and $\mathbb{C}_{r^p}$ with $P$ respectively.

**Definition 5 (Session nesting relation).** *Let $r_1 \prec_P r_2$ iff $P \equiv \mathbb{C}[\![\mathbb{C}_{r_1^p}[\![r_2^q \triangleright P]\!]]\!]$ and let $<_P$ the transitive closure of $\prec_P$.*

Relation $<_P$ for initial processes is also acyclic ($<_P$ is irreflexive), it is preserved by $\tau$ reductions and it holds that if $P \xrightarrow{r\iota} Q$ then $<_Q = <_P \cup \{(x, r) \mid \exists r_1 . r_1 \prec_P r \wedge x <_P r_1\}$.

Next proposition is a sort of progress property valid for the outermost (in terms of $\prec$-relation) active sessions. In fact, if one of such sessions has a pending action enabled then it is either guaranteed that after a finite number of steps a suitable synchronization is accomplished or the opening of a new nested session is performed (by means of a service invocation).

**Proposition 2.** *Let $\Gamma \vdash P : \mathsf{end}[\mathsf{end}]$ be an initial process typed over $\Gamma$ that contains only assumptions about sessions. Whenever $P \xrightarrow{\omega}{}^{*} Q$ and $r$ is a session name in $Q$ such that $\nexists r_2 <_Q r$ with $Q \xrightarrow{r_2^q:\lambda}$ or $Q \equiv \mathbb{C}[\![\mathbb{C}_{r_2^p}[\![P'' > \tilde{x} > Q']\!]]\!]$ and $\mathbb{C}[\![\mathbb{C}_{r_2^p}[\![P'']\!]]\!] \xrightarrow{r_2^q:\lambda}$ all of the following hold:*

*(I)* *if $Q \xrightarrow{r^p:\leftarrow\tilde{v}}$ then $Q \xrightarrow{\gamma}{}^{*} \xrightarrow{r^{\overline{p}}:\rightarrow\tilde{v}} \vee\, Q \xrightarrow{\gamma}{}^{*} \xrightarrow{r_1\iota} Q_1$ and $r <_{Q_1} r_1$*

*(II)* *if $Q \xrightarrow{r^p:\rightarrow\tilde{v}}$ then $Q \xrightarrow{\gamma}{}^{*} \xrightarrow{r^{\overline{p}}:\leftarrow\tilde{v}} \vee\, Q \xrightarrow{\gamma}{}^{*} \xrightarrow{r_1\iota} Q_1$ and $r <_{Q_1} r_1$*

*(III)* *if $Q \xrightarrow{r^p:\leftarrow l}$ then $Q \xrightarrow{\gamma}{}^{*} \xrightarrow{r^{\overline{p}}:\rightarrow l} \vee\, Q \xrightarrow{\gamma}{}^{*} \xrightarrow{r_1\iota} Q_1$ and $r <_{Q_1} r_1$*

*(IV)* *if $Q \xrightarrow{r^p:\rightarrow l}$ then $Q \xrightarrow{\gamma}{}^{*} \xrightarrow{r^{\overline{p}}:\leftarrow l} \vee\, Q \xrightarrow{\gamma}{}^{*} \xrightarrow{r_1\iota} Q_1$ and $r <_{Q_1} r_1$*

*(V)* *if $Q \equiv \mathbb{C}[\![\mathbb{C}_{r^p}[\![P' > \tilde{x} > Q]\!]]\!]$ and $\Gamma_1 \vdash P' : U[!(\tilde{S}).T]$ then*
*$\mathbb{C}[\![\mathbb{C}_{r^p}[\![P']\!]]\!] \xrightarrow{\gamma}{}^{*} \xrightarrow{r^p:\rightarrow\tilde{v}} \vee\, \mathbb{C}[\![\mathbb{C}_{r^p}[\![P']\!]]\!] \xrightarrow{\gamma}{}^{*} \xrightarrow{r_1\iota} Q_1$ and $r <_{Q_1} r_1$*

*Proof.* The proof is by induction on the length $llns(r, Q)$ of the longest nesting sequence induced by $<_Q$ and starting with $r$; that is the longest sequence of the form $r <_Q r_1 <_Q r_2 \ldots r_{n-1} <_Q r_n$, and then on the structure of the processes. More precisely, the well-founded order we consider for the induction is defined on pairs $(r^p, Q)$ by letting $(r_1^{p_1}, Q_1) < (r_2^{p_2}, Q_2)$ be the least transitive relation satisfying:

- $(r_1^{p_1}, Q_1) < (r_2^{p_2}, Q_2)$ if $llns(r_1, Q_1) < llns(r_2, Q_2)$
- $(r^p, \mathbb{C}[\![\mathbb{C}_{r^p}[\![Q_1]\!]]\!]) < (r^p, \mathbb{C}[\![\mathbb{C}_{r^p}[\![Q_2]\!]]\!])$ if $llns(r, Q_1) = llns(r, Q_2)$ and $Q_1$ is a subterm of $Q_2[\tilde{v}/\tilde{x}]$ for suitable $\tilde{v}$.

We sketch the first case (see the Appendix for the other cases). Actually proving *(I)* means proving two different statements, depending on the essence of $p$.

(I).a if $Q \xrightarrow{r^-:\leftarrow\tilde{v}}$ and $\Gamma, r : [!(\tilde{S}).T] \vdash Q : \mathsf{end}[\mathsf{end}]$ then $Q \xrightarrow{\gamma}{}^{*} \xrightarrow{r^+:\rightarrow\tilde{v}} \vee\, Q \xrightarrow{\gamma}{}^{*} \xrightarrow{r_1\iota} Q_1$ and $r <_{Q_1} r_1$

(I).b if $Q \xrightarrow{r^+:\leftarrow\tilde{v}}$ and $\Gamma, r : [?(\tilde{S}).T] \vdash Q : \mathsf{end}[\mathsf{end}]$ then $Q \xrightarrow{\gamma}{}^{*} \xrightarrow{r^-:\rightarrow\tilde{v}} \vee\, Q_1 \xrightarrow{\gamma}{}^{*} \xrightarrow{r_1\iota} Q_1$ and $r <_{Q_1} r_1$

We can read previous statements as "a session side must respect, after a certain number of steps $\gamma$, the obligation imposed by its type unless it postpones the obligation with a new service call". The fact that the type of $r$ reflects the enabled action is a direct consequence of the subject reduction.
Case *(I).a* If $r : [!(\tilde{S}).T]$ it means that $\Gamma_1 \vdash Q' : U[!(\tilde{S}).T]$, where $Q \equiv \mathbb{C}[\![r^+ \triangleright Q']\!]$. The entire proof is completely type-driven, the key idea is that we consider only rules able to yield $U[!(\tilde{S}).T]$ in the conclusion. For ease of readability we also use the variable $W$ to range over processes.
**Base cases**: The base cases in Figure 8(a) are those prefixes compatible with an $r^+$ output action. In these cases $\gamma$ is the empty sequence.
**(Some) Inductive cases**: The thesis follows by inductive hypothesis on $\mathbb{C}[\![r^+ \triangleright W]\!]$ for (TparL) and $\mathbb{C}[\![r^+ \triangleright R]\!]$ for (TparR) (see Figure 8(b)).

For (Tpipe), figure 8(c), we apply the inductive hypothesis case *(V)* on $\mathbb{C}[\![r^+ \triangleright W]\!]$. If $\mathbb{C}[\![r^+ \triangleright W]\!] \xrightarrow{\gamma'}{}^{*} \xrightarrow{r^+\rightarrow\tilde{v}} \mathbb{C}[\![r^+ \triangleright W']\!]$ then $\mathbb{C}[\![r^+ \triangleright (W > \tilde{x} > R)]\!] \xrightarrow{\gamma'}{}^{*} \xrightarrow{\tau}$

$$\text{(TOUT)}\ \frac{\Gamma', \tilde{v} : \tilde{S} \vdash W : U[T]}{\Gamma \vdash \langle \tilde{v} \rangle.W : U[!(\tilde{S}).T]} \qquad \text{(TSESI)}\ \frac{\Gamma', \tilde{v} : \tilde{S} \vdash \mathtt{return}\ \tilde{v}.W : !(\tilde{S}).T[T']}{\Gamma', r_1 : [\overline{T'}], \tilde{v} : \tilde{S} \vdash r_1^- \triangleright \mathtt{return}\ \tilde{v}.W : [!(\tilde{S}).T]} \qquad \text{(TINV)}\ \frac{\Gamma', s : [\overline{T'}] \vdash W : !(\tilde{S}).T[T']}{\Gamma', s : [\overline{T'}] \vdash \overline{s}.W : [!(\tilde{S}).T]}$$

(a) Base cases

$$\text{(TPARL)}\ \frac{\Gamma \vdash W : U_W[!(\tilde{S}).T] \quad \Gamma \vdash R : U_R[\mathsf{end}]}{\Gamma \vdash W|R : U_W \circ U_R[!(\tilde{S}).T]} \qquad \text{(TPARR)}\ \frac{\Gamma \vdash W : U_W[\mathsf{end}] \quad \Gamma \vdash R : U_R[!(\tilde{S}).T]}{\Gamma \vdash W|R : U_W \circ U_R[!(\tilde{S}).T]}$$

(b) Inductive cases: parallel

$$\text{(TPIPE)}\ \frac{\Gamma \vdash W : U_W[!(\tilde{S}')] \quad \Gamma, \tilde{x} : \tilde{S}' \vdash R : U_R[!(\tilde{S}).T]}{\Gamma \vdash W > \tilde{x} > R : U_W \circ U_R[!(\tilde{S}).T]} \qquad \text{(TSESI)}\ \frac{\Gamma' \vdash (\tilde{x}).W : !(\tilde{S}).T[T']}{\Gamma', r_1 : [\overline{T'}] \vdash r_1^- \triangleright (\tilde{x}).W : \mathsf{end}[!(\tilde{S}).T]}$$

(c) Inductive cases: pipe

(d) Inductive cases: nested session

**Fig. 8.** Deduction rules

$\mathbb{C}[\![r^+ \triangleright (W' > \tilde{x} > R)|R[\tilde{v}/\tilde{x}]]\!]$, and then the thesis follows by inductive hypothesis on $\mathbb{C}[\![r^+ \triangleright R[\tilde{v}/\tilde{x}]]\!]$ if instead $\mathbb{C}[\![r^+ \triangleright W]\!] \xrightarrow{\gamma}{}^* \xrightarrow{r_2\iota}$ then $\mathbb{C}[\![r^+ \triangleright (W > \tilde{x} > R)]\!] \xrightarrow{\gamma}{}^* \xrightarrow{r_2\iota} Q_1$ and $r <_{Q_1} r_2$.

In the presence of a nested session $r_1$ with $r \prec_Q r_1$ we have various cases all similar. For example, if $Q \xrightarrow{r_1^- : \leftarrow \tilde{v}}$, figure 8(d), then by inductive hypothesis we have two cases. $Q \xrightarrow{\gamma'}{}^* \xrightarrow{r_1^- : \rightarrow \tilde{v}}$ and then the thesis follows adding $r_1\tau$ at the begin of the resulting sequence generated by another application of the inductive hypothesis on $\mathbb{C}[\![r^+ \triangleright (r_1^- \triangleright W)]\!]$. Or $Q \xrightarrow{\gamma'}{}^* \xrightarrow{r_2\iota} Q_1$ with $r_1 <_{Q_1} r_2$ and then the thesis follows since $r \prec_Q r_1$ and $r_1 <_{Q_1} r_2$. Case *(I).b* is similar but we might also have $Q \equiv \mathbb{C}[\![r^- \triangleright \mathtt{return}\ \tilde{v}.W]\!]$. As $Q$ has type $\mathsf{end}[\mathsf{end}]$ there exists a parent session $r_1$ for which the return become an enabled output. Since $r_1 \prec_Q r$ this case contradicts the assumptions. □

We are now ready to prove the main result.

**Theorem 2 (Deadlock freedom).** *Let $P$ an initial process s.t. $\emptyset \vdash P : \mathsf{end}[\mathsf{end}]$ then for each $Q$ s.t. $P \xrightarrow{\omega}{}^* Q$ then either $Q \xrightarrow{\tau}$ or $Q$ is in normal form.*

*Proof.* By contradiction, if Proposition 2 holds then it cannot be the case $Q$ blocked on a pending action in the middle of a session. In fact, it is always possible to accomplish the synchronization choosing the right session that fits the proposition hypothesis. Such a session must be one of the outermost sessions with an enabled action. Furthermore, since the process is closed the type system (rule (TNew)) ensures that every service call

is successful. Otherwise, if only service declarations remain the process is already in normal form. □

*Remark 1.* The result can be extended to processes $P$ that can output some values. In fact, if $\Gamma \vdash P : \mathtt{end}[!(\tilde{S})^k]$ then we can take any suitable $Q$ (designed to work on the resulting values) such that $\Gamma \vdash P > \tilde{}x > Q : \mathtt{end}[\mathtt{end}]$, thus fitting the requirements of Proposition 2. (The simplest case is $Q \equiv \mathbf{0}$.)

*Example 4.* The process $P = s.(x).\overline{s}.\langle x\rangle.(y).\mathtt{return}\ y \mid \overline{s}.\langle 5\rangle.(y)$ is well typed in the environment $s : ?(int).!(int)$ and hence is deadlock free. Notice that the input of $y$ never succeeds but the process is deadlock free since it keeps invoking new instances of $s$ (nested within the first established top session).

As a final observation, consider that even if simple, our framework can also correctly typecheck non-tail recursive processes. Take for example the initial process $s.\overline{s}.\mathtt{return}\ 1.(x)|\overline{s}.(y)$. It is well formed and welltyped in the type environment $s : !(int)$ then we can conclude the process is deadlock free. An equivalent $\pi$-calculus process is $*s(r).(\nu r')(\overline{s}(r').\overline{r}(1).r'(x))|(\nu r)\overline{s}(r).r(y)$ and in Kobayashi's type system [19, 18] the action $\overline{r}(1)$ is not ensured to succeed. In fact, $r$ and $r'$ have the same type and hence the capability level of $r$ equal to the obligation level of $r'$ but rule **A'**[19] is not applicable since $r$ is created less recently than $r'$. Proposition 2 instead ensures that $\mathtt{return}\ 1$ is eventually performed.

## 5 Conclusion

We have studied a service language with sessions and pipelines. Differently from [15, 16, 24] our language build sessions automatically on each service invocation. For this language we have defined a simple type system. To some extent, it is similar to the simply typed $\pi$-calculus because it only tracks the exchanged values in each session. In fact, since sessions are developed as low level runtime primitives, the type system does not need to check session linearity, thus it only worries about session duality. Checking duality means simply tracking active session usages; i.e., the current session, and the parent session.

Subsequently, we restrict on the class of initial processes for which well typing imply a suitable notion of deadlock freedom. Together with the type interference algorithm, reported in [23], we have a simple tool to check deadlock freedom of our processes. Among the main novelties of our approach we emphasize the form of the types for sessions with returns ($U[T]$) and the typing rules for pipelines together with the particular form of induction due to sessions structure.

The full version of this work will address the enhancement of the type system with a notion of subtyping so that different usages of the same service can be typed consistently. Moreover, recursion and regular $\mu$-types [14] will be accounted for in the type system, even if Proposition 2 will no longer hold in the present form. To see this, think of a process making the same unbounded number of inputs and returns: if we type the process as $\mu\alpha.!(\tilde{S}).\alpha[\mu\alpha.?(\tilde{S}).\alpha]$ then we just approximate its real essence, because the number of inputs and returns must be the same. Furthermore, we want to relax some

requirements on parallel usages, admit session passing and extend the type system to multiparty session languages.

## References


1. L. Acciai and M. Boreale. Xpi: a typed process calculus for xml messaging. In *Proceedings of FMOODS'05*, volume 3535 of *Lect. Notes in Comput. Sci.*, pages 47–66. Springer Verlag, 2005.
2. L. Acciai and M. Boreale. A type system for client progress in a service-oriented calculus. 2008. Lect. Notes in Comput. Sci. to apper.
3. D. Booth and C. Liu. *Web Services Description Language (*WSDL*) Version 2.0 Part 0: Primer*, 2006. `http://www.w3.org/TR/2006/CR-wsdl20-primer-20060327`.
4. M. Boreale, R. Bruni, L. Caires, R. De Nicola, I. Lanese, M. Loreti, F. Martins, U. Montanari, A. Ravara, D. Sangiorgi, V. Vasconcelos, and G. Zavattaro. SCC: A service centered calculus. In M. Bravetti, M. Núñez, and G. Zavattaro, editors, *Proceedings of WS-FM'06*, volume 4184 of *Lect. Notes in Comput. Sci.*, pages 38–57. Springer Verlag, 2006.
5. M. Boreale, R. Bruni, R. De Nicola, and M. Loreti. Sessions and pipelines for structured service programming. In *Proceedings of FMOODS'08*, Lect. Notes in Comput. Sci. Springer Verlag, 2008. To appear.
6. A. L. Brown, C. Laneve, and L. G. Meredith. Piduce: A process calculus with native XML datatypes. In *Proceedings of EPEW'05/WS-FM'05*, volume 3670 of *Lect. Notes in Comput. Sci.*, pages 18–34. Springer Verlag, 2005.
7. M. G. Buscemi and U. Montanari. Cc-pi: A constraint-based language for specifying service level agreements. In *Proceedings of ESOP'07*, volume 4421 of *Lect. Notes in Comput. Sci.* Springer Verlag, 2007.
8. N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro. Sock: a calculus for service oriented computing. In A. Dan and W. Lamersdorf, editors, *Proceedings of ICSOC'06*, volume 4294 of *Lect. Notes in Comput. Sci.*, pages 327–338. Springer Verlag, 2006.
9. M. Carbone, K. Honda, and N. Yoshida. Structured communication-centred programming for web services. In *Proceedings of ESOP'07*, volume 4421 of *Lect. Notes in Comput. Sci.*, pages 2–17. Springer Verlag, 2007.
10. R. Chinnici, H. Haas, A. Lewis, J.-J. Moreau, et al. *Web Services Description Language (*WSDL*) Version 2.0 Part 2: Adjuncts*, 2006. `http://www.w3.org/TR/2006/CR-wsdl20-adjuncts-20060327`.
11. R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana. *Web Services Description Language (*WSDL*) Version 2.0 Part 1: Core Language*, 2006. `http://www.w3.org/TR/2006/CR-wsdl20-20060327`.
12. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. *Web Services Description Language (*WSDL*) 1.1*, 2001. `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`.
13. W. Cook, D. Kitchin, and J. Misra. A language for task orchestration and its semantic properties. In C. Baier and H. Hermanns, editors, *Proceedings of CONCUR'06*, volume 4137 of *Lect. Notes in Comput. Sci.*, pages 477–491. Springer Verlag, 2006.
14. V. Gapeyev, M. Levin, and B. Pierce. Recursive subtyping revealed. *J. Funct. Program.*, 12(6):511–548, 2002.
15. S. Gay and M. Hole. Subtyping for session types in the pi calculus. *Acta Inform.*, 42(2):191–225, 2005.
16. K. Honda, V. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In C. Hankin, editor, *Proceedings of ESOP'98*, volume 1381 of *Lect. Notes in Comput. Sci.*, pages 122–138. Springer Verlag, 1998.

17. A. Igarashi and N. Kobayashi. A generic type system for the pi-calculus. *ACM SIGPLAN Notices*, 36(3):128–141, 2001.
18. N. Kobayashi. Typical: Type-based static analyzer for the pi-calculus. Tool available at `http://www.kb.ecei.tohoku.ac.jp/~koba/typical/`.
19. N. Kobayashi. New type system for deadlock-free processes. In C. Baier and H. Hermanns, editors, *Proceedings of CONCUR'06*, volume 4137 of *Lect. Notes in Comput. Sci.*, pages 233–247. Springer Verlag, 2006.
20. N. Kobayashi and D. Sangiorgi. A hybrid type system for lock-freedom of mobile processes. `http://www.cs.unibo.it/~sangio/DOC_public/hybrid.ps.gz`.
21. I. Lanese, V. Vasconcelos, F. Martins, and A. Ravara. Disciplining orchestration and conversation in service-oriented computing. In *Proceedings of SEFM'07*. IEEE Computer Society Press, 2007. To appear.
22. A. Lapadula, R. Pugliese, and F. Tiezzi. A calculus for orchestration of web services. In *Proceedings of ESOP'07*, volume 4421 of *Lect. Notes in Comput. Sci.*, pages 33–47. Springer Verlag, 2007.
23. L. G. Mezzina. How to infer finite session types in a calculus of services and sessions. In *Proceedings of COORDINATION'08*, Lect. Notes in Comput. Sci. Springer Verlag, 2008. To appear.
24. N. Yoshida and V. Vasconcelos. Language primitives and type discipline for structured communication-based programming revisited: Two systems for higher-order session communication. *Electron. Notes Theor. Comput. Sci.*, 171(4):73–93, 2007.

## Appendix

***Definition (Operational semantics).*** *The following is an alternative formulation of the operational semantics by means of evaluation contexts.*

$$
\begin{array}{lll}
(Inv) & \mathbb{D}[\![\overline{s}.P, s.Q]\!] \;\rightarrow\; (\nu r)\mathbb{D}[\![r^- \triangleright P, r^+ \triangleright Q \mid s.Q]\!] & r \notin \mathsf{fn}(\mathbb{D}[\![\overline{s}.P, s.Q]\!]) \\
(Com) & \mathbb{D}_r[\![(\tilde{x}).P, \langle \tilde{v} \rangle.Q]\!] \;\rightarrow\; \mathbb{D}_r[\![P[\tilde{v}/\tilde{x}], Q]\!] & \\
(Lcom) & \mathbb{D}_r[\![\Sigma_{i=1}^{n}(l_i).P_i, \langle l_k \rangle.Q]\!] \;\rightarrow\; \mathbb{D}_r[\![P_k, Q]\!] & (1 \leq k \leq n) \\
(Ret) & \mathbb{D}_{r_1}[\![(\tilde{x}).P, \mathbb{C}_{r^p}[\![\mathtt{return}\ \tilde{v}.Q]\!]]\!] \;\rightarrow\; \mathbb{D}_{r_1}[\![P[\tilde{v}/\tilde{x}], \mathbb{C}_{r^p}[\![Q]\!]]\!] & \\
(Pipe) & \mathbb{C}[\![\langle \tilde{v} \rangle.P > \tilde{x} > Q]\!] \;\rightarrow\; \mathbb{C}[\![P > \tilde{x} > Q \mid Q[\tilde{v}/\tilde{x}]]\!] & \\
(PipeRet) & \mathbb{C}[\![\mathbb{C}_{r^p}[\![\mathtt{return}\ \tilde{v}.P]\!] > \tilde{x} > Q]\!] \;\rightarrow\; \mathbb{C}[\![\mathbb{C}_{r^p}[\![P]\!] > \tilde{x} > Q \mid Q[\tilde{v}/\tilde{x}]]\!] & \\
(IfT) & \mathbb{C}[\![\mathtt{if}\ v = v_1\ \mathtt{then}\ P\ \mathtt{else}\ Q]\!] \rightarrow \mathbb{C}[\![P]\!] & (v = v_1) \downarrow \mathtt{true} \\
(IfF) & \mathbb{C}[\![\mathtt{if}\ v = v_1\ \mathtt{then}\ P\ \mathtt{else}\ Q]\!] \rightarrow \mathbb{C}[\![Q]\!] & (v = v_1) \downarrow \mathtt{false} \\
(Scop) & P \;\rightarrow\; P' \;\Rightarrow\; (\nu m)P \rightarrow (\nu m)P' & \\
(Str) & P \equiv P' \;\wedge\; P' \rightarrow Q' \;\wedge\; Q' \equiv Q \Rightarrow\; P \rightarrow Q &
\end{array}
$$

*where*

$$
\begin{array}{lll}
\mathbb{C} & ::= & [\![\cdot]\!] \mid \mathbb{C}|P \mid r^p \triangleright \mathbb{C} \mid \mathbb{C} > \tilde{x} > P \\
\mathbb{C}_{r^p} & ::= & r^p \triangleright [\![\cdot]\!]|P \\
\mathbb{D} & ::= & \mathbb{C}[\![\mathbb{C}' \mid \mathbb{C}'']\!] \\
\mathbb{D}_r & ::= & \mathbb{D}[\![\mathbb{C}'_{r^p}, \mathbb{C}''_{\overline{r^p}}]\!] \qquad r \notin \mathsf{fn}(\mathbb{D})
\end{array}
$$

***Definition ($T^k$).***

$$
\begin{array}{ll}
(T.\mathsf{end})^k & = T^k \\
(!(\tilde{S}))^k & = !(\tilde{S}).!(\tilde{S})^{k-1} \\
(!(\tilde{S}))^1 & = !(\tilde{S}).\mathsf{end}
\end{array}
$$

***Definition ($\circ$).***

$$
T \circ T' = \begin{cases}
\mathsf{end} \circ \mathsf{end} & = \mathsf{end} \\
!(\tilde{S}) \circ \mathsf{end} & = !(\tilde{S}) \\
\mathsf{end} \circ !(\tilde{S}) & = !(\tilde{S}) \\
!(\tilde{S}) \circ !(\tilde{S}) & = !(\tilde{S}).!(\tilde{S}) \\
!(\tilde{S}).T \circ \mathsf{end} & = !(\tilde{S}) \circ T \\
\mathsf{end} \circ !(\tilde{S}).T & = !(\tilde{S}) \circ T \\
!(\tilde{S}) \circ !(\tilde{S}).T & = !(\tilde{S}).!(\tilde{S}) \circ T \\
!(\tilde{S}).T \circ T' & = !(\tilde{S}) \circ (T.T') \\
\textit{undefined otherwise} &
\end{cases}
$$

***Proposition (Subject Congruence).*** *If $\Gamma \vdash P : U[T]$ and $P \equiv Q$ then $\Gamma \vdash Q : U[T]$*

*Proof.* By induction on the derivation of $P \equiv Q$, with a case analysis on the last rule used. The inductive cases are the congruence rules, and are straightforward. Of the other cases we show the two most significant.
Case for $((\nu m)P)|Q \equiv (\nu m)(P|Q)$ if $m \notin \mathsf{fn}(Q)$, we have:

$$
* \quad \dfrac{\dfrac{\Gamma, m : S \vdash P : U[T]}{\Gamma \vdash (\nu m)P : U[T]}\,(\text{Tnew}) \qquad \Gamma \vdash Q : U'[\mathsf{end}]}{\Gamma \vdash ((\nu m)P)|Q : U \circ U'[T]}\,(\text{TparL})
$$

which can be rearranged to give:

$$** \quad \dfrac{\dfrac{\Gamma, m : S \vdash P : U[T] \qquad \Gamma, m : S \vdash Q : U'[\mathsf{end}]}{\Gamma, m : S \vdash (P|Q) : U \circ U'[T]}\,(\text{TparL})}{\Gamma \vdash (\nu m)(P|Q) : U \circ U'[T]}\,(\text{Tnew})$$

The two proofs * and ** can be read from * to ** to have the only if side in which we use the weakening lemma to type $Q$ with the assumption $m$ and from ** to * for the if side in which we use the strengthening lemma to type $Q$ without the assumption $m$. The case in which (TparR) is used instead is similar.
Case for $r_1^p \triangleright (r_2^q \triangleright \mathbf{0}|Q) \equiv r_1^p \triangleright Q|r_2^q \triangleright \mathbf{0}$, we have:

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash \mathbf{0} : \mathsf{end}[\mathsf{end}]}{\Gamma, r_2 : [\mathsf{end}] \vdash r_2^+ \triangleright \mathbf{0} : \mathsf{end}[\mathsf{end}]}\,(\text{Tses}) \qquad \Gamma, r_2 \vdash Q : U[T]}{\Gamma, r_2 \vdash (r_2^+ \triangleright \mathbf{0}|Q) : U[T]}\,(\text{TparR})}{\Gamma, r_1 : [T], r_2 : [\mathsf{end}] \vdash r_1^+ \triangleright (r_2^+ \triangleright \mathbf{0}|Q) : \mathsf{end}[U]}\,(\text{Tses})$$

which can be rearranged to give:

$$\dfrac{\dfrac{\Gamma, r_2 : [\mathsf{end}] \vdash Q : U[T]}{\Gamma, r_1 : [T], r_2 : [\mathsf{end}] \vdash r_1^+ \triangleright Q : \mathsf{end}[U]}\,(\text{Tses}) \qquad \dfrac{\Gamma, r_1 : [T] \vdash \mathbf{0} : \mathsf{end}[\mathsf{end}]}{\Gamma, r_1 : [T], r_2 : [\mathsf{end}] \vdash r_2^+ \triangleright \mathbf{0} : \mathsf{end}[\mathsf{end}]}\,(\text{Tses})}{\Gamma, r_1 : [T], r_2 : [\mathsf{end}] \vdash r_1^+ \triangleright Q|r_2^+ \triangleright \mathbf{0} : \mathsf{end}[U]}\,(\text{TparL})$$

Since the two proofs use the same sets of assumptions we can conclude the hypothesis.
The mirror cases with different polarities for $r_1$ and $r_2$ are similar.
Case for $(P|Q) > \tilde{}x > R \equiv (P > \tilde{}x > R)|(Q > \tilde{}x > R)$:

$$\dfrac{\dfrac{\Gamma \vdash P : U[!(\tilde{S})] \qquad \Gamma \vdash Q : U'[\mathsf{end}]}{\Gamma \vdash (P|Q) : U \circ U'[!(\tilde{S})]}\,(\text{TparR}) \qquad \Gamma, \tilde{}x : \tilde{S} \vdash R : U''[T]}{\Gamma \vdash (P|Q) > \tilde{}x > R : U \circ U' \circ U''[T]}\,(\text{Tpipe})$$

which can be arranged to give:

$$\dfrac{\dfrac{\Gamma \vdash P : U[!(\tilde{S})] \qquad \Gamma, \tilde{}x : \tilde{S} \vdash R : U''[T]}{\Gamma \vdash (P > \tilde{}x > R) : U \circ U'[T]}\,(\text{Tpipe}) \qquad \dfrac{\Gamma \vdash Q : U'[\mathsf{end}] \qquad \Gamma, \tilde{}x : \tilde{S} \vdash R : U''[T]}{\Gamma \vdash Q > \tilde{}x > R : U'[\mathsf{end}]}\,(\text{Tpipe})}{\Gamma \vdash (P > \tilde{}x > R)|(Q > \tilde{}x > R) : U \circ U' \circ U''[T]}\,(\text{TparR})$$

The mirror case which uses (TParL) is similar. There are also two other cases depending of the pipe definition in which the process $R$ can make only one input.

$$\dfrac{\dfrac{\Gamma \vdash P : U[(!(\tilde{S}))^k] \qquad \Gamma \vdash Q : U'[\mathsf{end}]}{\Gamma \vdash (P|Q) : U \circ U'[(!(\tilde{S}))^k]}\,(\text{TparR}) \qquad \Gamma, \tilde{}x : \tilde{S} \vdash R : U''[\mathsf{end}]}{\Gamma \vdash (P|Q) > \tilde{}x > R : U \circ U' \circ (U'')^k[\mathsf{end}]}\,(\text{Tpipe})$$

which can be arranged to give:

$$\dfrac{\dfrac{\Gamma \vdash P : U[(!(\tilde{S}))^k] \qquad \Gamma, \tilde{x} : \tilde{S} \vdash R : U''[\mathsf{end}]}{\Gamma \vdash (P > \tilde{x} > R) : U \circ (U'')^k[\mathsf{end}]}\,(\text{Tpipe}) \qquad \dfrac{\Gamma \vdash Q : U'[\mathsf{end}] \qquad \Gamma, \tilde{x} : \tilde{S} \vdash R : U''[\mathsf{end}]}{\Gamma \vdash Q > \tilde{x} > R : U'[\mathsf{end}]}\,(\text{Tpipe})}{\Gamma \vdash (P > \tilde{x} > R) | (Q > \tilde{x} > R) : U \circ U' \circ (U'')^k[\mathsf{end}]}\,(\text{TparR})$$

□

***Lemma (Substitution lemma).*** *Let $\Gamma, x : S \vdash P : U[T]$. If $\Gamma \vdash w : S$ then $\Gamma \vdash P[^w/_x] : U[T]$.*

*Proof.* By induction on the typing judgment.
Without losing the essence of the argument, we consider only 1-arity.
Case for (Tzero): straightforward.
Cases for (Tin) and (Tnew) : follow directly from the induction hypothesis and the fact that variable under the binder cannot be chosen neither as a object nor as a subject of the substitution.
Case for (Tout): the only interesting case is when $x = v$, we have:

$$\dfrac{\overset{(\text{Tout})}{\Gamma, v : S \vdash P : U[T] \quad \Gamma, v : S \vdash v : S}}{\Gamma, v : S \vdash \langle v \rangle.P : U[!(S).T]}$$

by assumption $\Gamma \vdash w : S$ and induction hypothesis $\Gamma \vdash P[^w/_v] : U[T]$ so (Tout) gives $\Gamma \vdash \langle w \rangle.P[^w/_v] : U[!(S).T]$ which is the required judgment.
Case for (Tret): the only interesting case is when $x = v$, we have:

$$\dfrac{\overset{(\text{Tret})}{\Gamma, v : S \vdash P : U[T] \quad \Gamma, v : S \vdash v : S}}{\Gamma', v : S \vdash \mathtt{return}\, v.P : !(S).U[T]}$$

by assumption $\Gamma \vdash w : S$ and induction hypothesis $\Gamma \vdash P[^w/_v] : U[T]$ so (Tret) gives $\Gamma \vdash \mathtt{return}\, w.P[^w/_v] : !(S).U[T]$ which is the required judgment.
Case for (Tinv) the only interesting case is when $x = v$, we have:

$$\dfrac{\overset{(\text{Tinv})}{\Gamma', v : [\overline{T}], w : [\overline{T}] \vdash P : U[T]}}{\Gamma', v : [\overline{T}], w : [\overline{T}] \vdash \overline{v}.P : \mathsf{end}[U]}$$

by induction hypothesis $\Gamma', w : [\overline{T}] \vdash P[^w/_v] : U[T]$ so (Tinv) gives $\Gamma', w : [\overline{T}] \vdash \overline{w}.P[^w/_v] : \mathsf{end}[U]$ which is the required judgment.
Cases for (Tchoice), (Tbranch), (Tses), (TsesI) and (Tdef) follow directly from the induction hypothesis.
Cases for (TparL), (TparR), (Tif) and (Tpipe) follow directly from the distributivity of substitution with respect to |, `if` and > and the fact that pipe and ∘ work only on the resulting type. □

***Theorem (Subject reduction).*** *If $\Gamma \vdash R : U[T]$ and $R \xrightarrow{\lambda} R'$ then $\exists \Gamma', T', U'$ s.t. $\Gamma' \vdash Q : U'[T']$ provided that whenever $R \xrightarrow{r^p : \leftarrow \tilde{v}} R'$ also $R \xrightarrow{r^{\overline{p}} : \rightarrow \tilde{v}} R''$ and whenever $R \xrightarrow{r^p : \leftarrow l} R'$ also $R \xrightarrow{r^{\overline{p}} : \rightarrow l} R''$.*

*Proof.* We prove a more general statement, assumptions are necessary due to the early style operational semantics, in particular for the shape of (In) and (Branch) rules. The proof is by induction on the derivation with a case analysis on the last used rule.

Case for (In): We have $\Gamma \vdash (\tilde{x}).P : U[?\tilde{S}).T]$ where $(\tilde{x}).P \xrightarrow{\leftarrow \tilde{v}} P[\tilde{v}/\tilde{x}]$. We also know that $\Gamma', \tilde{x} : \tilde{S} \vdash P : U[T]$ where $\Gamma'$ depends of the shape of $\tilde{v} : \tilde{S}$ and $\Gamma \subseteq \Gamma'$. It's worth noticing the fact that the chosen $\tilde{v}$ has the same type of $\tilde{x}$ follow from the hypotheses. By the application of the substitution lemma we have $\Gamma' \vdash P[\tilde{v}/\tilde{x}] : U[T]$ which is the desiderated judgment.

Case for (Branch) We have $\Gamma \vdash \Sigma_{i=1}^{n}(l_i).P_i : U[\&\{l_i : T_i\}]_{i \in I}$ and $I \subseteq \{1, \ldots, n\}$. In particular, the last point follow from the hypotheses. The desiderated judgment follows because $\Gamma \vdash P_i : U[T_i]$ and $i \in I$ .

Case for (Choice): We have $\Gamma \vdash \langle l \rangle.P : U[\oplus\{l_1 : T_1 \ldots l_n : T_n\}]$ and $l = l_i$. The desiderated judgment follows because $\Gamma \vdash P : U[T_i]$.

Cases for (Select): We have $\Gamma \vdash (P|Q) : U_P \circ U_Q[T]$. By inductive hypothesis we know that either $\Gamma \vdash P' : U_P[T]$ and $\Gamma \vdash Q' : U_Q[\mathsf{end}]$ or $\Gamma \vdash P' : U_P[\mathsf{end}]$ and $\Gamma \vdash Q' : U_Q[T]$. Applying (TparL)/(TparR) we obtain $\Gamma \vdash (P'|Q') : U_P \circ U_Q[T]$ which is the desiderated judgment.

Case for (Out): We have $\Gamma', \tilde{v} : \tilde{S} \vdash \langle \tilde{v} \rangle.P : U[!\tilde{S}).T]$ and $\langle \tilde{v} \rangle.P \xrightarrow{\rightarrow \tilde{v}} P$. We also know that either $\Gamma', \tilde{v} : \tilde{S} \vdash P : U[T]$ or $\Gamma' \vdash P : U[T]$ (depending if $\tilde{v}$ is still used by $P$) which is the desiderated judgment.

Case for (Ret): We have $\Gamma', \tilde{v} : \tilde{S} \vdash \mathtt{return}\ \tilde{v}.P : !\tilde{S}).U[T]$ and $\mathtt{return}\ \tilde{v}.P \xrightarrow{\uparrow \tilde{v}} P$. We also know that $\Gamma', \tilde{v} : \tilde{S} \vdash P : U[T]$ or $\Gamma' \vdash P : U[T]$ (depending if $\tilde{v}$ is still used by $P$) which is the desiderated judgment.

Case for (Def): We have $\Gamma', s : [T] \vdash s.P : \mathsf{end}[\mathsf{end}]$ and $s.P \xrightarrow{s \Rightarrow r} r^+ \triangleright P|s.P$. We also know that $\Gamma', s : [T] \vdash P : \mathsf{end}[T]$ and that the session $r$ has type $[T]$ which is the same of $s$. Thanks to (TSes) we can conclude that $\Gamma', s : [T], r : [T] \vdash r^+ \triangleright P : \mathsf{end}[\mathsf{end}]$ and finally with (TparL) $\Gamma', s : [T], r : [T] \vdash r \triangleright P|s.P : \mathsf{end}[\mathsf{end}]$ which is the desiderated judgment.

Case for (Inv): We have $\Gamma', s : [\overline{T}] \vdash s.P : \mathsf{end}[U]$ and $\overline{s}.P \xrightarrow{s \Leftarrow r} r^- \triangleright P$. We also know that $\Gamma', s : [\overline{T}] \vdash P : U[T]$ and that the session $r$ has a type $[T]$. Thanks to (TSesI) we can conclude that $\Gamma', s : [\overline{T}], r : [\overline{T}] \vdash r^- \triangleright P : \mathsf{end}[U]$ which is the desiderated judgment.

Case for (SCom): We have $\Gamma', s : S \vdash (P|Q) : U_P \circ U_Q[T]$. By inductive hypothesis we also know $\Gamma', s : S, r : \overline{S} \vdash P' : U_Q[T]$ and $\Gamma', s : S\, r : S \vdash Q' : U_Q[\mathsf{end}]$ and by applying (TParR) and (Tnew) we obtain $\Gamma' \vdash (\nu r)(P'|Q') : U_P \circ U_Q[T]$. It is worth noticing that if $\overline{s}$ is a top level invocation and it returns some values then $T$ is exactly the type of these values.

Case for (SessionOut): We have $\Gamma', r : [?(\tilde{S}).\overline{T}] \vdash r^- \triangleright P : U[!(\tilde{S}).T]$. By inductive hypothesis we know that $\Gamma'' \vdash P' : T[T']$ where $\Gamma'' \subseteq \Gamma'$. Thanks to (TsesI) we can conclude $\Gamma'', r : [\overline{T}] \vdash r^- \triangleright P' : [T]$ which is the desiderated judgment. In the other case we have, $\Gamma', r : [!(\tilde{S}).T] \vdash r^+ \triangleright P : U[!(\tilde{S}).T]$ and it is similar.

Case for (Session): We have $\Gamma', r : [T] \vdash r^+ \triangleright P : \mathsf{end}[U]$ and $T$ depending on $\lambda$. By inductive hypothesis we know $\Gamma'' \vdash P' : T'[U']$ where either $\Gamma'' \subseteq \Gamma'$ (for an output) or $\Gamma' \subseteq \Gamma''$ (for an input) or $\Gamma' = \Gamma''$ (for labels and $\tau$) and thanks to (Tses) we can

conclude $\Gamma'', r : [T'] \vdash r \triangleright^{+} P' : \mathsf{end}[U]$ which is the desiderated judgment.
Case for (NestInv). We have $\Gamma', s : [T] \vdash r^{+} \triangleright P : \mathsf{end}[U]$. By inductive hypothesis we know $\Gamma', s : [T], r' : [\overline{T}] \vdash P' : U[\overline{T}]$. Thanks to (Tses) we can conclude $\Gamma', s : [T], r' : [\overline{T}] \vdash r^{+} \triangleright P' : \mathsf{end}[U]$. The mirror case with $r^{-} \triangleright P$ is similar.
Case for (Comm): We have either $\Gamma', r : [!(\tilde{S}).T] \vdash (P|Q) : U_P \circ U_Q[T']$ or $\Gamma', r : [!(\tilde{S}).T] \vdash (P|Q) : U_P \circ U_Q[T']$ depending on the polarity of $p$. By inductive hypothesis we know $\Gamma'' \vdash P' : U_P[T']$ where $\Gamma \subseteq \Gamma''$ and $\Gamma''' \vdash Q' : U_Q[\mathsf{end}]$ where $\Gamma''' \subseteq \Gamma$ with $\overline{T''} = T$. Applying (TparL) we obtain $\Gamma', r : [T] \vdash (P'|Q') : U_P \circ U_Q[T']$ which is the desiderated judgment. The other cases one where $P$ is located on the client side and one where $Q$ performs actions are similar.
Case for (Res): We have $\Gamma \vdash P : U[T]$ and by inductive hypothesis $\Gamma, s : S \vdash P' : U'[T']$. Applying (Tnew) we obtain $\Gamma \vdash P' : U'[T']$ which is the desiderated judgment.
Cases for (SessRes) and (Open) similar to the previous case.
Case for (Par): We have $\Gamma \vdash P|Q : U_P \circ U_Q[T]$ and by inductive hypothesis $\Gamma' \vdash P' : U'_P[T']$ where $\Gamma'$,$U'_P$ and $T'$ depend of the shape of the label $\lambda$. Applying (TParL) we have $\Gamma \vdash P'|Q : U'_P \circ U_Q[T']$. The other case with $P : U_P[\mathsf{end}]$ is similar.
Case for (Pipe): We have $\Gamma \vdash P > \tilde{x} > Q : U[T]$ and by inductive hypothesis we also have $\Gamma' \vdash P' : U_P[\mathsf{end}]$ where $\Gamma' \subseteq \Gamma$. Substitution lemma and inductive hypothesis on $Q$ give $\Gamma'' \vdash Q[\tilde{v}/\tilde{x}] : U_Q[T]$ where $\Gamma'' \subseteq \Gamma$ with $U = U_P \circ U_Q$. The desiderated judgment follows by applying (Tpipe) and (TParR) which give $\Gamma \vdash P' > \tilde{x} > Q|Q[\tilde{v}/\tilde{x}] : U[T]$. In the other case we have $\Gamma \vdash P > \tilde{x} > Q : U[\mathsf{end}]$. By inductive hypothesis $\Gamma' \vdash P' : U_P[T]$ where $\Gamma' \subseteq \Gamma$ and $\Gamma'' \vdash Q : U_Q[\mathsf{end}]$ where $\Gamma'' \subseteq \Gamma$. By applying (TPipe) we have $\Gamma \vdash P' > \tilde{x} > Q : U[\mathsf{end}]$ and with (TParR) and the substitution lemma finally we can conclude $\Gamma \vdash (P' > \tilde{x} > Q)|Q[\tilde{v}/\tilde{x}] : U[\mathsf{end}]$
Case for (PipePass) we have $\Gamma \vdash P > \tilde{x} > Q : U[T]$ and by inductive hypothesis we also have $\Gamma' \vdash P' : U'[T']$ and by applying (TPipe) $\Gamma'' \vdash P' > \tilde{x} > Q : U'[T]$ where $\Gamma \subseteq \Gamma''$.
Cases for (IfL) and (IfR): Follow directly from the inductive hypothesis.
Case for (Equiv): Follow directly by inductive hypothesis and the subject congruence lemma. □

***Proposition.*** *Let $\Gamma \vdash P : \mathsf{end}[\mathsf{end}]$ be an initial process typed over $\Gamma$ that contains only assumptions about sessions. Whenever $P \xrightarrow{\omega}{}^{*} Q$ and $r$ is a session name in $Q$ such that $\nexists r_2 <_Q r$ with $Q \xrightarrow{r_2^q:\lambda}$ or $Q \equiv \mathbb{C}[\![\mathbb{C}_{r_2^p}[\![P'' > \tilde{x} > Q']\!]\!]$ and $\mathbb{C}[\![\mathbb{C}_{r_2^p}[\![P'']\!]\!] \xrightarrow{r_2^q:\lambda}$ all of the following hold:*

*(I) if $Q \xrightarrow{r^p:\leftarrow\tilde{v}}$ then $Q \xrightarrow{\gamma}{}^{*} \xrightarrow{r^{\overline{p}}:\rightarrow\tilde{v}} \vee Q \xrightarrow{\gamma}{}^{*} \xrightarrow{r_1\iota} Q_1$ and $r <_{Q_1} r_1$*
*(II) if $Q \xrightarrow{r^p:\rightarrow\tilde{v}}$ then $Q \xrightarrow{\gamma}{}^{*} \xrightarrow{r^{\overline{p}}:\leftarrow\tilde{v}} \vee Q \xrightarrow{\gamma}{}^{*} \xrightarrow{r_1\iota} Q_1$ and $r <_{Q_1} r_1$*
*(III) if $Q \xrightarrow{r^p:\leftarrow l}$ then $Q \xrightarrow{\gamma}{}^{*} \xrightarrow{r^{\overline{p}}:\rightarrow l} \vee Q \xrightarrow{\gamma}{}^{*} \xrightarrow{r_1\iota} Q_1$ and $r <_{Q_1} r_1$*
*(IV) if $Q \xrightarrow{r^p:\rightarrow l}$ then $Q \xrightarrow{\gamma}{}^{*} \xrightarrow{r^{\overline{p}}:\leftarrow l} \vee Q \xrightarrow{\gamma}{}^{*} \xrightarrow{r_1\iota} Q_1$ and $r <_{Q_1} r_1$*
*(V) if $Q \equiv \mathbb{C}[\![\mathbb{C}_{r^p}[\![P' > \tilde{x} > Q]\!]\!]$ and $\Gamma_1 \vdash P' : U[!(\tilde{S}).T]$ then*
*$\mathbb{C}[\![\mathbb{C}_{r^p}[\![P']\!]\!] \xrightarrow{\gamma}{}^{*} \xrightarrow{r^p:\rightarrow\tilde{v}} \vee \mathbb{C}[\![\mathbb{C}_{r^p}[\![P']\!]\!] \xrightarrow{\gamma}{}^{*} \xrightarrow{r_1\iota} Q_1$ and $r <_{Q_1} r_1$*

*Proof.* The proof is by induction on the length $llns(r, Q)$ of the longest nesting sequence induced by $<_Q$ and starting with $r$; that is the longest sequence of the form

$r \prec_Q r_1 \prec_Q r_2 \ldots r_{n-1} \prec_Q r_n$, and then on the structure of the processes. More precisely, the well-founded order we consider for the induction is defined on pairs $(r^p, Q)$ by letting $(r_1^{p_1}, Q_1) < (r_2^{p_2}, Q_2)$ be the least transitive relation satisfying:

- $(r_1^{p_1}, Q_1) < (r_2^{p_2}, Q_2)$ if $llns(r_1, Q_1) < llns(r_2, Q_2)$
- $(r^p, \mathbb{C}[\![\mathbb{C}_{r^p}[\![Q_1]\!]]\!]) < (r^p, \mathbb{C}[\![\mathbb{C}_{r^p}[\![Q_2]\!]]\!])$ if $llns(r, Q_1) = llns(r, Q_2)$ and $Q_1$ is a subterm of $Q_2[\tilde{v}/\tilde{x}]$ for suitable $\tilde{v}$.

We show the first case the others are simpler.

Actually proving *(I)* means proving two different statements.

(I).a if $\Gamma, r : [!(\tilde{S}).T] \vdash Q : \mathsf{end}[\mathsf{end}]$ and $Q \xrightarrow{r^-:\leftarrow\tilde{v}}$ then $Q \xrightarrow{\gamma}{}^* \xrightarrow{r^+:\rightarrow\tilde{v}} \vee\, Q \xrightarrow{\gamma}{}^* \xrightarrow{r_1\iota} Q_1$ and $r <_{Q_1} r_1$

(I).b if $\Gamma, r : [?(\tilde{S}).T] \vdash Q : \mathsf{end}[\mathsf{end}]$ and $Q \xrightarrow{r^+:\leftarrow\tilde{v}}$ then $Q \xrightarrow{\gamma}{}^* \xrightarrow{r^-:\rightarrow\tilde{v}} \vee\, Q_1 \xrightarrow{\gamma}{}^* \xrightarrow{r_1\iota} Q_1$ and $r <_{Q_1} r_1$

We can read previous statements as "a session side must respect, after a certain number of steps $\gamma$, the obligation imposed by its type unless it postpones the obligation with a new service call". The fact that the type of $r$ reflects the type of the enabled action is straightforward.

Case *(I).a* if $\Gamma(r) = [!(\tilde{S}).T]$ it means that:

$$\text{(TSES)}\quad \frac{\Gamma \vdash Q' : U[!(\tilde{S}).T]}{\Gamma, r : [!(\tilde{S}).T] \vdash r^+ \triangleright Q' : \mathsf{end}[U]}$$

where $Q \equiv \mathbb{C}[\![r^+ \triangleright Q']\!]$. The entire proof is completely types driven, the key idea is that we consider only rules able to yields $U[!(\tilde{S}).T]$ in the conclusion. For ease of readability we also use the variable $W$ to range over processes.

**Base cases**: The bases cases are those prefixes compatible with an $r^+$ output action.

$$\text{(TOUT)}\quad \frac{\Gamma', \tilde{v} : \tilde{S} \vdash W : U[T]}{\Gamma \vdash \langle \tilde{v} \rangle.W : U[!(\tilde{S}).T]} \qquad \text{(TSESI)}\quad \frac{\Gamma', \tilde{v} : \tilde{S} \vdash \mathtt{return}\ \tilde{v}.W : !(\tilde{S}).T[T']}{\Gamma', r_1 : [\overline{T'}], \tilde{v} : \tilde{S} \vdash r_1^- \triangleright \mathtt{return}\ \tilde{v}.W : \mathsf{end}[!(\tilde{S}).T]}$$

$$\text{(TINV)}\quad \frac{\Gamma', s : [\overline{T'}] \vdash W : !(\tilde{S}).T[T']}{\Gamma', s : [\overline{T'}] \vdash \overline{s}.W : \mathsf{end}[!(\tilde{S}).T]}$$

in these cases $\gamma$ is the empty sequence and $r_1 \neq r$ since session nesting is acyclic.

**Inductive cases**:

$$\text{(TPARL)}\quad \frac{\Gamma \vdash W : U_W[!(\tilde{S}).T] \quad \Gamma \vdash R : U_R[\mathsf{end}]}{\Gamma \vdash W|R : U_W \circ U_R[!(\tilde{S}).T]} \qquad \text{(TPARR)}\quad \frac{\Gamma \vdash W : U_W[\mathsf{end}] \quad \Gamma \vdash R : U_R[!(\tilde{S}).T]}{\Gamma \vdash W|R : U_W \circ U_R[!(\tilde{S}).T]}$$

The thesis follows by induction hypothesis on $\mathbb{C}[\![r^+ \triangleright W]\!]$ for (TparL) and $\mathbb{C}[\![r^+ \triangleright R]\!]$ for (TparR).

$$\frac{\Gamma \vdash v_i : S_i \quad i = 1,2 \quad \Gamma \vdash W : U[!(\tilde{S}).T] \quad \Gamma \vdash R : U[!(\tilde{S}).T]}{\Gamma \vdash \texttt{if}\ v_1 = v_2\ \texttt{then}\ W\ \texttt{else}\ R : U[!(\tilde{S}).T]}\ (\textsc{Tif})$$

The thesis follows by induction hypothesis either on $\mathbb{C}[\![r^+ \triangleright W]\!]$ or $\mathbb{C}[\![r^+ \triangleright R]\!]$ depending on the evaluation of the if guard.

$$\frac{\Gamma \vdash W : U_W[!(\tilde{S}')] \quad \Gamma, \tilde{x} : \tilde{S}' \vdash R : U_R[!(\tilde{S}).T]}{\Gamma \vdash W > \tilde{x} > R : U_W \circ U_R[!(\tilde{S}).T]}\ (\textsc{Tpipe})$$

We apply the induction hypothesis case *(V)* on $\mathbb{C}[\![r^+ \triangleright W]\!]$. If $\mathbb{C}[\![r^+ \triangleright W]\!] \xrightarrow{\gamma'}{}^* \xrightarrow{r^+ \to \tilde{v}} \mathbb{C}[\![r^+ \triangleright W']\!]$ then $\mathbb{C}[\![r^+ \triangleright (W > \tilde{x} > R)]\!] \xrightarrow{\gamma'}{}^* \xrightarrow{\tau} \mathbb{C}[\![r^+ \triangleright (W' > \tilde{x} > R) | R[\tilde{v}/\tilde{x}]]\!]$, and then the thesis follows by induction hypothesis on $\mathbb{C}[\![r^+ \triangleright R[\tilde{v}/\tilde{x}]]\!]$ if instead $\mathbb{C}[\![r^+ \triangleright W]\!] \xrightarrow{\gamma}{}^* \xrightarrow{r_2 \iota}$ then $\mathbb{C}[\![r^+ \triangleright (W > \tilde{x} > R)]\!] \xrightarrow{\gamma}{}^* \xrightarrow{r_2 \iota} Q_1$ and $r <_{Q_1} r_2$.

$$\frac{\Gamma, s : S \vdash W : U[!(\tilde{S}).T]}{\Gamma \vdash (\nu s)W : U[!(\tilde{S}).T]}\ (\textsc{Tnew})$$

Follows directly by inductive hypothesis on $\mathbb{C}[\![r^+ \triangleright W]\!]$.

$$\frac{\Gamma' \vdash (\tilde{x}).W : !(\tilde{S}).T[T']}{\Gamma', r_1 : [\overline{T'}] \vdash r_1^- \triangleright (\tilde{x}).W : \mathsf{end}[!(\tilde{S}).T]}\ (\textsc{TsesI})$$

Since $Q \xrightarrow{r_1^- : \leftarrow \tilde{v}}$ and $r \prec_Q r_1$ then by induction hypothesis we have two cases. $Q \xrightarrow{\gamma'}{}^* \xrightarrow{r_1^- : \to \tilde{v}}$ and then the thesis follows adding $r_1 \tau$ at the begin of the resulting sequence generated by another application of the induction hypothesis on $\mathbb{C}[\![r^+ \triangleright (r_1^- \triangleright W)]\!]$. Or $Q \xrightarrow{\gamma'}{}^* \xrightarrow{r_2 \iota} Q_1$ with $r_1 <_{Q_1} r_2$ and then the thesis follows since $r \prec_Q r_1$ and $r_1 <_{Q_1} r_2$.

$$\frac{\Gamma' \vdash \langle \tilde{v} \rangle.W : !(\tilde{S}).T[T']}{\Gamma', r_1 : [\overline{T'}] \vdash r_1^- \triangleright \langle \tilde{v} \rangle.W : \mathsf{end}[!(\tilde{S}).T]}\ (\textsc{TsesI}) \qquad \frac{\Gamma' \vdash \langle l_i \rangle.W : !(\tilde{S}).T[T']}{\Gamma', r_1 : [\overline{T'}] \vdash r_1^- \triangleright \langle l_i \rangle.W : \mathsf{end}[!(\tilde{S}).T]}\ (\textsc{TsesI})$$

$$\frac{\Gamma' \vdash \sum_{i=1}^{n} (l_i).W_i : !(\tilde{S}).T[T']}{\Gamma, r_1 : [\overline{T'}] \vdash r_1^- \triangleright \langle l_i \rangle.W : \mathsf{end}[!(\tilde{S}).T]}\ (\textsc{TsesI}) \qquad \frac{\Gamma' \vdash (\nu s)W : !(\tilde{S}).T[T']}{\Gamma', r_1 : [\overline{T'}] \vdash r_1^- \triangleright (\nu s)W : \mathsf{end}[!(\tilde{S}).T]}\ (\textsc{TsesI})$$

They are similar to the previous case.

$$\frac{\Gamma' \vdash W : U_W[T'] \quad \Gamma' \vdash R : !(\tilde{S}).U_R[\mathsf{end}]}{\Gamma', r_1 : [\overline{T'}] \vdash r_1^- \triangleright (W|R) : \mathsf{end}[!(\tilde{S}).U_W \circ U_R]}\ (\textsc{Tsesi}) \qquad \frac{\Gamma' \vdash W : !(\tilde{S}).U_W[T'] \quad \Gamma' \vdash R : U_R[\mathsf{end}]}{\Gamma', r_1 : [\overline{T'}] \vdash r_1^- \triangleright (W|R) : \mathsf{end}[!(\tilde{S}).U_W \circ U_R]}\ (\textsc{Tsesi})$$

$$\frac{\Gamma' \vdash W : U_W[\mathsf{end}] \quad \Gamma' \vdash R : !(\tilde{S}).U_R[T']}{\Gamma', r_1 : [\overline{T'}] \vdash r_1^- \triangleright (W|R) : \mathsf{end}[!(\tilde{S}).U_W \circ U_R]}\ (\textsc{Tsesi}) \qquad \frac{\Gamma' \vdash W : !(\tilde{S}).U_W[\mathsf{end}] \quad \Gamma' \vdash R : U_R[T']}{\Gamma', r_1 : [\overline{T'}] \vdash r_1^- \triangleright (W|R) : \mathsf{end}[!(\tilde{S}).U_W \circ U_R]}\ (\textsc{Tsesi})$$

Thesis follows directly on induction on the process producing the output.

$$\frac{\dfrac{\Gamma' \vdash W : !(\tilde{S}).U_W[!(\tilde{S}')] \qquad \Gamma', \tilde{x} : \tilde{S}' \vdash R : U_R[T']}{\Gamma' \vdash W > \tilde{x} > R : !(\tilde{S}).U_W \circ U_R[T']}}{\Gamma', r_1 : [\overline{T'}] \vdash r_1^- \triangleright (W > \tilde{x} > R) : \mathsf{end}[!(\tilde{S}).U_W \circ U_R]}$$

$$\frac{\dfrac{\Gamma' \vdash W : U_W[!(\tilde{S}')] \qquad \Gamma', \tilde{x} : \tilde{S}' \vdash R : !(\tilde{S}).U_R[T']}{\Gamma' \vdash W > \tilde{x} > R : !(\tilde{S}).U_W \circ U_R[T']}}{\Gamma', r_1 : [\overline{T'}] \vdash r_1^- \triangleright (W > \tilde{x} > R) : \mathsf{end}[!(\tilde{S}).U_W \circ U_R]}$$

We apply the induction hypothesis case *(V)* on $\mathbb{C}[\![r^+ \triangleright (r_1^- \triangleright W)]\!]$. If $\mathbb{C}[\![r^+ \triangleright (r_1^- \triangleright W)]\!] \xrightarrow{\gamma'}{}^* \xrightarrow{r_1^- \to \tilde{v}} \mathbb{C}[\![r^+ \triangleright (r_1^- \triangleright W')]\!]$ then $\mathbb{C}[\![r^+ \triangleright (r_1^- \triangleright (W > \tilde{x} > R))]\!] \xrightarrow{\gamma'}{}^* \xrightarrow{\tau} \mathbb{C}[\![r^+ \triangleright (r_1^- \triangleright (W' > \tilde{x} > R)|R[\tilde{v}/\tilde{x}]))]\!]$, and then the thesis follows by induction hypothesis either on $\mathbb{C}[\![r^+ \triangleright (r_1^- \triangleright W')]\!]$ for the first case or on $\mathbb{C}[\![r^+ \triangleright (r_1^- \triangleright R[\tilde{v}/\tilde{x}]))]\!]$ for the second case. If instead $\mathbb{C}[\![r^+ \triangleright (r_1^- \triangleright W)]\!] \xrightarrow{\gamma}{}^* \xrightarrow{r_1 \iota}$ then $\mathbb{C}[\![r^+ \triangleright (r_1^- \triangleright (W > \tilde{x} > R))]\!] \xrightarrow{\gamma}{}^* \xrightarrow{r_2 \iota} Q_1$ and $r <_{Q_1} r_2$.

$$\frac{\dfrac{\Gamma' \vdash W : !(\tilde{S}).U_W[!(\tilde{S}')^k] \qquad \Gamma', \tilde{x} : \tilde{S}' \vdash R : U_R[\mathsf{end}]}{\Gamma' \vdash W > \tilde{x} > R : !(\tilde{S}).U_W \circ U_R^k[\mathsf{end}]}}{\Gamma', r_1 : [\mathsf{end}] \vdash r_1^- \triangleright (W > \tilde{x} > R) : \mathsf{end}[!(\tilde{S}).U_W \circ U_R^k]}$$

$$\frac{\dfrac{\Gamma' \vdash W : U_W[!(\tilde{S}')^k] \qquad \Gamma', \tilde{x} : \tilde{S}' \vdash R : !(\tilde{S}).U_R[\mathsf{end}]}{\Gamma' \vdash W > \tilde{x} > R : !(\tilde{S})^k.U_W \circ U_R^k[\mathsf{end}]}}{\Gamma', r_1 : [\mathsf{end}] \vdash r_1^- \triangleright (W > \tilde{x} > R) : \mathsf{end}[!(\tilde{S})^k.U_W \circ U_R^k]}$$

We apply the induction hypothesis case *(V)*. If $\mathbb{C}[\![r^+ \triangleright (r_1^- \triangleright W)]\!] \xrightarrow{\gamma'}{}^* \xrightarrow{\to \tilde{v}}$ then $\mathbb{C}[\![r^+ \triangleright (r_1^- \triangleright (W > \tilde{x} > R))]\!] \xrightarrow{\gamma'}{}^* \xrightarrow{\tau} \mathbb{C}[\![r^+ \triangleright (r_1^- \triangleright (W > \tilde{x} > R))|R[\tilde{v}/\tilde{x}]))]\!]$, and then the thesis follows by induction hypothesis either on $\mathbb{C}[\![r^+ \triangleright (r_1^- \triangleright W)]\!]$ for the first case or on $\mathbb{C}[\![r^+ \triangleright (r_1^- \triangleright R[\tilde{v}/\tilde{x}]))]\!]$ for the second case. If instead $\mathbb{C}[\![r^+ \triangleright (r_1^- \triangleright W)]\!] \xrightarrow{\gamma}{}^* \xrightarrow{r_1 \iota}$ then $\mathbb{C}[\![r^+ \triangleright (r_1^- \triangleright (W > \tilde{x} > R))]\!] \xrightarrow{\gamma}{}^* \xrightarrow{r_2 \iota} Q_1$ and $r <_{Q_1} r_2$.

$$\frac{\Gamma \vdash W : T'[!(\tilde{S}).T]}{\Gamma \vdash \mathtt{return}\ \tilde{v}.W : !(\tilde{S}').T'[!(\tilde{S}).T]}\ (\textsc{Tret})$$

Case *(I).b* is similar but we might also have $Q \equiv \mathbb{C}[\![r^- \triangleright \texttt{return}\ \tilde{v}.W]\!]$. As $Q$ has type $\mathsf{end}[\mathsf{end}]$ there exists a parent session $r_1$ for which the return become an enabled output. Since $r_1 \prec_Q r$ this case contradicts the assumptions.

$\square$