Under consideration for publication in Math. Struct. in Comp. Science

Typed observational equivalence for sessions

Marco Giunti

INRIA Saclay and LIX, Ecole Polytechnique, France

Received 13 December 2011

We propose a behavioural theory to contrast processes described by session type abstractions. We introduce a notion of typed observational equivalence for a polyadic pi calculus with matching where the discerning capability of the observer is regulated by the type checker. In particular, type checking forces contexts to not interfere with a session shared by two participants. Behaviourally equivalent pi calculus processes exhibit the same observables in all type checked contexts. To avoid universal quantification, we rely on a proof technique based on bisimulation over typed labelled semantics. By establishing the soundness and the completeness of bisimulation semantics with respect to observational equivalence, we provide a framework to reason about the behaviour of service-oriented protocols.

1. Introduction

Service oriented computing is emerging as one of the prominent paradigms for distributed computations. A novelty introduced by the paradigm is the concept of session, which describes a serie of interactions among web services and clients having a precise structure. While the session abstraction shares some similarity with other approaches for distributed computing, as for instance communication protocols, it also introduces advanced features that need a specific account; one of such capabilities consists in *delegating* a session to other threads in order to distribute the load. A protocol based on web services can therefore be complex and error-prone; to help software engineers and programmers in avoiding mistakes, a static analysis based on strong static typing can be effective. Starting from the seminal works of (Takeuchi, Honda and Kubo 1994; Honda, Vasconcelos and Kubo 1998), which introduced *session types* for a dialect of pi calculus, several approaches for a typed analysis of sessions have been developed; see (Dezani-Ciancaglini and de'Liguoro 2009) for a recent overview.

In this paper, we focus on session type abstractions for a polyadic pi calculus with matching (cf.(Hennessy 2007)), and on reasoning techniques to contrast processes described by such abstractions. Our aim is to provide for a method to complement the typed analysis of processes with a behavioural analysis based on co-inductive techniques. This can be of help in detecting behavioural errors of web service specifications which are not detected by the typing system. To illustrate, consider the pi calculus process R

below willing to describe a service protocol for the exchange of a booking reference:

$$P \stackrel{\text{def}}{=} \overline{s}\langle user\rangle.s(x).\mathsf{if}\, x = ref\, \mathsf{then}\, P_1\, \mathsf{else}\, P_2$$
$$Q \stackrel{\text{def}}{=} s(y).\mathsf{if}\, y = mg\, \mathsf{then}\, \overline{s}\langle mgref\rangle$$
$$R \stackrel{\text{def}}{=} P \mid Q$$

The channel s does represent a session established among the two threads P and Q. The thread P sends on the session channel the identifier of an user in order to receive a confirmation of his booking reference. If the received reference corresponds with the expected one, the thread continues as P_1, otherwise it continues as P_2. The thread Q exhibits a dual behavior, that is it waits on the session channel for the acronym of an user and it sends on the same channel his booking reference; afterwards it terminates.

The behavior of the session in P is abstracted by the *end point* type !User.?RefId.end which depicts one end point of the session, the output part. Here the types $!S_1.S_2$ and $?S_1.S_2$ denote respectively send and receive a variable of type S_1 and afterwards continue as type S_2, and the type end indicates the end of the session, that is no further use in input or output of the session channel. The behavior of Q is depicted by the dual end point type ?User.!RefId.end, which illustrates the input part of the session. The whole session can be abstracted by means of a type constructor representing the concurrent behavior of the two end points (Giunti and Vasconcelos 2010):

$$T \stackrel{\text{def}}{=} (!\text{User}.?\text{RefId}.\mathsf{end}, ?\text{User}.!\text{RefId}.\mathsf{end})\ .$$

The type above describes a *dyadic* session, that is a session involving two participants. For the session protocol to be sound, we must therefore enforce that (at most) two participants can join the session. This can be imposed by means of the type judgement below, which enforces the session s to be shared at most by the two participants P and Q, thus establishing an *affine* type discipline for session channels: the type checker rules out processes that try to interfere with the session.

$$\Gamma, s\colon T \vdash R \tag{1}$$

However, the same assumption assigning to the session s the type T can be used to type check processes that exhibit behavioural errors. Consider the process P' below, which is obtained by exchanging in P the continuation P_1 with Q.

$$P' \stackrel{\text{def}}{=} \overline{s}\langle user\rangle.s(x).\mathsf{if}\, x = ref\, \mathsf{then}\, Q\, \mathsf{else}\, P_2$$

By using the same typing system of (1) we can infer the judgement below that permits to accept P'. Our claim is that is that process P' is *deadlocked*, since we believe the type system to forbid to processes put in parallel with P' to interact over the session s, and in turn to unblock the output over s of P'.

$$\Gamma', s\colon T \vdash P' \tag{2}$$

Intuitively, the problem that we notice in P' is that the two participants of the session are sequential, rather than concurrent. This could happen due to an ingenuity of the programmer, or because of mobility of sessions: that is, one or both ends of the session s

have been delegated to P'. Conceptually, we note that the type construct (S_1, S_2) describes the concurrent behaviour of end points S_1 and S_2 that, similarly to prefixes of processes, can be rejoned in a single, sequential thread.

To formally reason on the behaviour of such processes we shall have a notion of equivalence which takes into account the typed knowledge of sessions of the observer. More in detail, we want to be able to express that process P' is deadlocked by establishing the semantic equivalence of P' and the inert process $\mathbf{0}$ in all admissible contexts:

$$P' \stackrel{?}{\cong} \mathbf{0}$$

Clearly, this equation does not hold for untyped contexts, which can detect the process on the left by trying to read from the session. A distinguishing parallel process is $C' \stackrel{\text{def}}{=} s(y).\overline{\omega}\langle y\rangle$: such process can synchronize with P' and subsequently show a signal ω which is not observable when C' runs in parallel with the inert process. However, the composition of P' and C' is rejected by the type system; this holds since the channel s is already used both in input and output by P', and since the affine discipline forbids any further use of the channel. We therefore must regulate the behavior of the observers by enforcing type checking under adequate assumptions.

Given a type environment Δ representing the knowledge of the observer, and assuming Δ to be *compatible* with some Γ' and Γ'' such that $\Gamma' \vdash P'$ and $\Gamma'' \vdash \mathbf{0}$, we infer the following typed equation stating the equivalence (hence indistinguishability) of the two processes in all contexts type checked by Δ:

$$\Delta \models P' \cong \mathbf{0}$$

Intuitively, two environments are compatible when a) they do not contain the same end point of the session and b) they agree on the type of the messages exchanged. These conditions prevent the context to interfere with the session and in turn to break the desired equivalences.

Incidentally, we note that the presence of types is necessary in our behavioral theory because the constructs of pi calculus cannot be used to offer adequate protection for session abstractions. Indeed, while the previous example can be recovered by defending the session channel from the interference of the context by means of the restriction operator, this cannot be done when the session has been delegated by the context. Process P'' below is obtained by prefixing the process P' with an input, so that the channel s is a place-holder for a session received from the context. As in the previous example, the behaviour of the continuation should be indistinguishable from the inert process: the following typed equation formalizes this idea.

$$\Delta, a\colon !T.\mathsf{end} \models P'' \cong a(s\colon T).\mathbf{0} \qquad\qquad P'' \stackrel{\text{def}}{=} a(s\colon T).P'$$

However, without type checking of contexts this equation cannot be enforced. An untyped process can indeed discern P' from the inert process by first delegating a session to P'' and then by using it, which is forbidden in session-oriented programming. A distinguishing parallel process is $C'' \stackrel{\text{def}}{=} (\nu s')(\overline{a}\langle s'\rangle \mid s'(y).\overline{\omega}\langle y\rangle)$ where $a, \omega \neq s'$: we have that C'' and P'' do interact giving raise to a signal ω, while interaction of C'' with $a(s\colon T).\mathbf{0}$ does

not permit to observe such signal. Note that since the session channel s' is generated by the context, the only way to impose a session-based discipline for its use is by enforcing type checking.

1.1. *Contribution*

We introduce an affine type discipline for a polyadic pi calculus with matching which is useful for session-oriented programming. We contrast typed processes by introducing a notion of contextual behavioural equivalence where the behaviour of contexts is regulated by the type checker. In particular, admissible contexts cannot break the affine discipline for sessions, that is contexts are allowed to use an end of a session only if such end is not already exercised by one of the contrasted processes. To avoid universal quantification over contexts in proving our behavioural equations, we provide for a characterization of observational equivalence based on typed labelled semantics. By assessing the soundness and the completeness of bisimulation semantics, we supply a proof technique for observational equivalence and in turn an effective method to analyze the behavior of processes described by a session typing system. We show the usefulness of our technique for reason on the behaviour of programs by establishing liveness errors and security guarantees in code specifications.

1.2. *Plan of the paper*

Section 2 introduces a typed polyadic pi calculus with matching and its session-based type discipline. A behavioral theory for the calculus based on the notion of barbs is defined in Section 3. In Section 4 we introduce typed bisimulation semantics, and we prove that bisimilar process do exhibit the same contextual behavior. Completeness of bisimilarity with respect to observational equivalence is established in Section 5. Section 6 draws some example of the application of our technique. Lastly, in Section 7 we discuss related work and underline the limitations of our approach.

2. Pi calculus

This section introduces the syntax and the semantics of our typed polyadic pi calculus. The grammar of types is below.

T	$::= (S, S) \mid S \mid \top$	Types
$\tilde{T}$	$::= T_1, \ldots, T_n,\ n \geq 0$	Tuple
S	$::= ?\tilde{T}.S \mid !\tilde{T}.S \mid E$	End Point
E	$::= ?\tilde{T} \mid !\tilde{T} \mid \mathsf{end}$	Termination

We let T range over types, and let $\tilde{T}$ indicate a tuple of types. The *top* type, noted $\top$, permits to perform a local test for the identity of a variable by means of if-then-else. We consider *channel* types of the form (S, S) where S is a type describing the behavior of a channel end point. An *end point* type S finishes with a *termination* type E. A

type of the form $!\tilde{T}.S$ describes a channel end able to send at most once variables of type $\tilde{T} = T_1, \ldots, T_n$ and to proceed as prescribed by S, following an affine discipline. Similarly, $?\tilde{T}.S$ describes a channel end able to receive at most once variable of types $\tilde{T}$ and to continue as S. A type of the form $?\tilde{T}$ could be used in an unrestricted way to receive variables of types $\tilde{T}$. Similarly $!\tilde{T}$ is used zero or more times to send variables of types $\tilde{T}$. Type end describes a channel end on which no further interaction is possible. We say that a type T is a *session type* if it is of the form $?\tilde{T}.S$, of the form $!\tilde{T}.S$, or of the form (S_1, S_2) with S_1, S_2 session types.

End point type duality plays a central role ensuring that communication between the two ends of a channel proceeds smoothly. Intuitively, the dual of an output end point is an input end point and the dual of input end point is an output end point. In particular if S_2 is dual of S_1, noted $\overline{S}_1 = S_2$, then $?\tilde{T}.S_1$ is dual of $!\tilde{T}.S_2$. Note that the tuple $\tilde{T}$ expected in input corresponds to the tuple $\tilde{T}$ sent in output.

$$\overline{T_1, \ldots, T_n} = \overline{T}_1, \ldots, \overline{T}_n \qquad \overline{?\tilde{T}.S} = !\tilde{T}.\overline{S} \qquad \overline{!\tilde{T}.S} = ?\tilde{T}.\overline{S}$$
$$\overline{?\tilde{T}} = !\tilde{T} \qquad \overline{!\tilde{T}} = ?\tilde{T} \qquad \overline{\mathsf{end}} = \mathsf{end}$$

Not all types are available to the programmer. Namely, the type declarations in the syntax of processes include only *balanced types* generated by the sub-syntax below. Non-balanced types are used in the proof's derivations by the type system and must be hidden to the programmer, since they can lead to unsound protocols.

$B ::= (C, \overline{C}) \mid C \mid \top$	Balanced Types
$\tilde{B} ::= B_1, \ldots, B_n,\ n \geq 0$	Tuple
$C ::= ?\tilde{B}.C \mid !\tilde{B}.C \mid D$	End Point
$D ::= ?\tilde{B} \mid !\tilde{B} \mid \mathsf{end}$	Termination

The syntax of pi calculus processes is below.

$$P, Q ::= \overline{x}\langle\tilde{v}\rangle.P \mid x(\tilde{y}\colon \tilde{B}).P \mid \mathsf{if}\, x = y\, \mathsf{then}\, P\, \mathsf{else}\, Q \mid (\nu y\colon B)P \mid P \mid Q \mid !P \mid \mathbf{0}$$

We rely on a set of variables, ranged over by $a, b, \ldots, u, v, \ldots x, y, z$. We let $\tilde{v}$ indicate a tuple of variables $v_1, \ldots, v_n$, $n \geq 0$. We consider synchronous, polyadic output and input processes, in the forms $\overline{x}\langle\tilde{v}\rangle.P$ and $x(\tilde{y}\colon \tilde{B}).P$, where we assume each variable v_i in the tuple $\tilde{v} = v_1, \ldots, v_n$ to be distinct, and each variable y_i in the tuple $\tilde{y}\colon \tilde{B} = y_1\colon B_1, \ldots, y_n\colon B_n$ to be distinct and to be decorated with the balanced type B_i. The matching process $\mathsf{if}\, x = y\, \mathsf{then}\, P\, \mathsf{else}\, Q$ allows for comparison of variables. The restricted process $(\nu y\colon B)P$ provides for create a variable y decorated with the balanced type B. The remaining processes are parallel composition, replication, and inaction. The binders for the language appear in parentheses: variables $\tilde{y}$ are bound in $x(\tilde{y}\colon \tilde{B}).P$ and variable y is bound in $(\nu y\colon B)P$. Free and bound variables in processes are defined accordingly, and so is alpha conversion, substitution of variables $\tilde{y}$ by variables $\tilde{v}$ in a process P given $|\tilde{y}| = |\tilde{v}|$, denoted $P[\tilde{v}/\tilde{y}]$. We follow Barendregt's variable convention, requiring bound variables to be distinct from each other and from free variables in any mathematical context. We let $(\nu\tilde{y}\colon \tilde{B})P$ to abbreviate process $(\nu y_1\colon B_1)\cdots(\nu y_n\colon B_n)P$ whenever $\tilde{y} =$

Rules for structural congruence

$$P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R) \qquad P \mid \mathbf{0} \equiv P \qquad !P \equiv P \mid !P$$
$$(\nu y\colon B)P \mid Q \equiv (\nu y\colon B)(P \mid Q) \quad (\nu y\colon B)\mathbf{0} \equiv \mathbf{0}$$
$$(\nu y_1\colon B_1)(\nu y_2\colon B_2)P \equiv (\nu y_2\colon B_2)(\nu y_1\colon B_1)P$$

Rules for reduction

$$\frac{|\tilde{v}| = |\tilde{y}|}{\overline{x}\langle\tilde{v}\rangle.P \mid x(\tilde{y}\colon \tilde{B}).Q \xrightarrow{x} P \mid Q[\tilde{v}/\tilde{y}]} \qquad \text{[R-Com]}$$

$$\frac{P \xrightarrow{y} P' \qquad B \rightarrowtail B'}{(\nu y\colon B)P \xrightarrow{\tau} (\nu y\colon B')P'} \qquad \frac{P \xrightarrow{\mu} P' \qquad \mu \neq y}{(\nu y\colon B)P \xrightarrow{\mu} (\nu y\colon B)P'} \qquad \text{[R-Res-T],[R-Res]}$$

$$\frac{}{\mathsf{if}\, x = x\, \mathsf{then}\, P\, \mathsf{else}\, Q \xrightarrow{\tau} P} \qquad \frac{x \neq y}{\mathsf{if}\, x = y\, \mathsf{then}\, P\, \mathsf{else}\, Q \xrightarrow{\tau} Q} \qquad \text{[R-IfT] [R-IfF]}$$

$$\frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q} \qquad \frac{P \equiv Q \quad Q \xrightarrow{\mu} Q' \quad Q' \equiv P'}{P \xrightarrow{\mu} P'} \qquad \text{[R-Par] [R-Struct]}$$

Fig. 1. Pi calculus: Reduction semantics

$y_1, \cdots, y_n$ with $n \geq 1$, and be a short of process P whenever $\tilde{y}$ is empty. We will avoid often to write trailing and branching inert processes and abbreviate processes $\overline{x}\langle\tilde{v}\rangle.\mathbf{0}$ and $\mathsf{if}\, x = y\, \mathsf{then}\, P\, \mathsf{else}\, \mathbf{0}$ respectively to $\overline{x}\langle\tilde{v}\rangle$ and to $\mathsf{if}\, x = y\, \mathsf{then}\, P$.

Structural congruence is the smallest relation on processes including the rules in Figure 1. The definition is standard. In the first rule on the second line, scope extrusion allows the scope of x to encompass Q; due to variable convention, variable x bound in $(\nu y\colon B)P$ cannot be free in Q. The reduction is the smallest relation $\xrightarrow{\mu}$ on processes including the rules in Figure 1. To represent the evolution of types associated to restricted variables, we record the variable x on the arrow and let μ range over the internal transition τ and variables x, y, z. This is only for convenience, and has no impact on the behavioral theory (cf. (Kobayashi, Pierce and Turner 1999)).

The [R-Com] rule permits to communicate variables $\tilde{v}$ from an output prefixed one $\overline{x}\langle\tilde{v}\rangle.P$ to an input prefixed process $x(\tilde{y}\colon \tilde{B}).Q$ given that $\tilde{v}$ and $\tilde{y}$ have the same arity; the result is the parallel composition of the continuation processes, where the bound variables $\tilde{y}$ in the input process are replaced by the variables $\tilde{v}$. Moreover, we record on the arrow the channel on which the synchronization takes place; this will be exploited in rule [R-Res-T] to infer the type of a redex bound by restriction. To this aim, we introduce a notion of type reduction. A balanced type of the form $B = (C, \overline{C})$ evolves to B', noted $B \rightarrowtail B'$, by firing a deterministic step:

$$(?\tilde{B}.C, !\tilde{B}.\overline{C}) \rightarrowtail (C, \overline{C}) \qquad (!\tilde{B}.C, ?\tilde{B}.\overline{C}) \rightarrowtail (C, \overline{C}) \qquad (D, \overline{D}) \rightarrowtail (D, \overline{D}) \quad D \neq \mathsf{end}$$

Rule [R-Res] applies when the reduction is inferred from a channel not bound by the restriction. Rule [R-IfT] and [R-IfF] are respectively for matching and mismatching of variables. Rule [R-Par] permits to compose processes and rule [R-Struct] allows for rearrangement of processes by using structural congruence. We will often abuse the

Composition of Types

$$S_1 \otimes S_2 = \begin{cases} E & S_1 = E = S_2 \\ (S_1, S_2) & \text{else} \end{cases}$$

$$(S_1, S_2) \otimes E = \begin{cases} (S_1, S_2) & \exists i \in \{1,2\} \,.\, S_i = E \\ \text{undefined} & \text{else} \end{cases}$$

$$E \otimes (S_1, S_2) = (S_1, S_2) \otimes E$$

$$(E_1, E_2) \otimes (E_1, E_2) = (E_1, E_2)$$

$$T \otimes \top = T$$

$$\top \otimes T = T$$

Composition of Type Environments

$$\mathrm{dom}(\Gamma_1 \otimes \Gamma_2) = \mathrm{dom}(\Gamma_1) = \mathrm{dom}(\Gamma_2)$$

$$(\Gamma_1 \otimes \Gamma_2)(x) = \begin{cases} \Gamma_1(x) \otimes \Gamma_2(x) & \text{if } (\Gamma_1(x) \otimes \Gamma_2(x)) \downarrow \\ \text{undefined} & \text{else} \end{cases}$$

Fig. 2. Composition of types and of type environments

notation and write $P \to P'$ to indicate that there is μ such that $P \xrightarrow{\mu} P'$. Similarly, we write $P \Rightarrow P'$ to indicate that a) $P \to \cdots \to P'$ or that b) $P' = P$.

Type system Type environments Γ, Δ are a possibly empty map from variables to types. In an environment $\Gamma, x\colon T$ we assume that x does not occur in Γ; we also assume the variable bindings in Γ to be unordered. We let the predicate term be true of a) the empty environment, as well as of b) environment $\Gamma, x\colon T$ whenever $\mathsf{term}(\Gamma)$ and T is a termination type E, or T is a channel type (E_1, E_2) with E_1, E_2 termination types, or T is the type $\top$.

Types and type environments can be combined; this will be exploited in the typing rule for parallel processes and in the rules for sending variables. The definition of the partial *composition* operation over types, noted $\otimes\colon$ *Types* $\times$ *Types* $\to$ *Types*, is in Figure 2. We use infix notation an write $T_1 \otimes T_2$ to indicate the application $\otimes(T_1, T_2)$, and let $\tilde{T} \otimes \tilde{T'}$ indicate the applications $T_1 \otimes T'_1, \ldots, T_n \otimes T'_n$ whenever $\tilde{T} = T_1, \ldots, T_n$, and $\tilde{T'} = T'_1, \ldots, T'_n$ with $n \geq 1$, and let $\tilde{T} \otimes \tilde{T'}$ be null when $n = 0$. Whenever the composition of T_1 and T_2 is defined we write $(T_1 \otimes T_2) \downarrow$. The operator $\otimes$ is extended point-wise to type environments with equal domain. We let $\Gamma_1 \otimes \Gamma_2$ be defined, noted $(\Gamma_1 \otimes \Gamma_2) \downarrow$, whenever for all entries $x \in \mathrm{dom}(\Gamma_1)$ we have $(\Gamma_1(x) \otimes \Gamma_2(x)) \downarrow$.

Essentially, the definition in Figure 2 says that we can spawn the use of termination types and of the top type in an unbounded manner. To illustrate, suppose that type $(?\tilde{T}.S, \mathsf{end})$ describes the local use of a variable x in process P_1, and that x is locally used at type end in process P_2. Then if we consider the global use of x in P_1, P_2, that is the use of x in the parallel process $P_1 \mid P_2$, we shall join the two types by using the $\otimes$ operator, which eventually leads to the type $(?\tilde{T}.S, \mathsf{end})$. Another example is when the session types $?\tilde{T}.S_1$ and $!\tilde{T}.S_2$ describe the use of x respectively in processes P_1 and P_2. Then the global use of x in $P_1 \mid P_2$ is described by the composition of the two session

$$\frac{\Gamma(v) = E_i \text{ or } \Gamma(v) = (E_1, E_2) \text{ and } i \in \{1,2\}}{\Gamma \vdash v\colon E_i} \quad \text{[T-VarT]}$$

$$\frac{\Gamma(x) = T_1 \qquad \Gamma(y) = T_2 \qquad (T_1 \otimes T_2)\downarrow \qquad \Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash \mathsf{if}\, x = y\, \mathsf{then}\, P\, \mathsf{else}\, Q} \quad \text{[T-If]}$$

$$\Gamma \vdash \mathbf{0} \qquad \frac{\Gamma_1 \vdash P_1 \qquad \Gamma_2 \vdash P_2 \qquad (\Gamma_1 \otimes \Gamma_2)\downarrow}{\Gamma_1 \otimes \Gamma_2 \vdash P_1 \mid P_2} \quad \text{[T-Inact],[T-Par]}$$

$$\frac{\Gamma \vdash P \qquad \mathsf{term}(\Gamma)}{\Gamma \vdash !P} \qquad \frac{\Gamma, x\colon B \vdash P}{\Gamma \vdash (\nu x\colon B)P} \quad \text{[T-Repl] [T-Res]}$$

$$\frac{\Gamma, x\colon S, \tilde{y}\colon \tilde{B} \vdash P}{\Gamma, x\colon ?\tilde{B}.S \vdash x(\tilde{y}\colon \tilde{B}).P} \qquad \frac{\Gamma, x\colon (S, S'), \tilde{y}\colon \tilde{B} \vdash P}{\Gamma, x\colon (?\tilde{B}.S, S') \vdash x(\tilde{y}\colon \tilde{B}).P} \quad \text{[T-In],[T-In-L]}$$

$$\frac{\Gamma \vdash x\colon ?\tilde{B} \qquad \Gamma, \tilde{y}\colon \tilde{B} \vdash P}{\Gamma \vdash x(\tilde{y}\colon \tilde{B}).P} \quad \text{[T-InT]}$$

$$\frac{\Gamma, \tilde{v}\colon \tilde{T}_j, x\colon S \vdash P \qquad (\tilde{T}_1 \otimes \tilde{T}_2)\downarrow \qquad (*)}{\Gamma, \tilde{v}\colon \tilde{T}_1 \otimes \tilde{T}_2, x\colon !\tilde{T}_i.S \vdash \overline{x}\langle\tilde{v}\rangle.P} \quad \text{[T-Out]}$$

$$\frac{\Gamma, \tilde{v}\colon \tilde{T}_j, x\colon (S, S') \vdash P \qquad (\tilde{T}_1 \otimes \tilde{T}_2)\downarrow \qquad (*)}{\Gamma, \tilde{v}\colon \tilde{T}_1 \otimes \tilde{T}_2, x\colon (!\tilde{T}_i.S, S') \vdash \overline{x}\langle\tilde{v}\rangle.P} \quad \text{[T-Out-L]}$$

$$\frac{\Gamma \vdash x\colon !\tilde{T}_i \qquad \Gamma, \tilde{v}\colon \tilde{T}_j \vdash P \qquad (\tilde{T}_1 \otimes \tilde{T}_2)\downarrow \qquad (*)}{\Gamma, \tilde{v}\colon \tilde{T}_1 \otimes \tilde{T}_2 \vdash \overline{x}\langle\tilde{v}\rangle.P} \quad \text{[T-OutT]}$$

$$(*)\ i \in \{1,2\},\ j = i + 1 (\mathrm{mod}\ 2)$$

Fig. 3. Type system

types which leads to the channel type $(?\tilde{T}.S_1, !\tilde{T}.S_2)$. This case illustrates indeed the essence of our session-based type discipline. On contrast, types $(?\tilde{T}.S_1, !\tilde{T}.S_2)$ and $!\tilde{T}.S$ cannot be composed. Intuitively, we cannot permit these two local uses of a type to be joined together since this would break the affine type discipline for sessions, which says that each end point of a session must be used at most once.

The typing system in Figure 3 is inspired by the linear type system for pi calculus without matching of (Giunti and Vasconcelos 2010). Differently from that paper, session types follow here an affine discipline. This permits a more compact and elegant formulation of our theory. Moreover, affine types permit to accept matching processes which do not exhibit the same behaviour in both branches, as for instance the process $\mathsf{if}\, x = y\, \mathsf{then}\, P\, \mathsf{else}\, \mathbf{0}$. Besides this, the same results would apply for a linear type system, while the definition of typed equivalent processes, which we will introduce in the next section, would be involved.

Rule [T-VarT] is to type a channel with a termination type, and is used in rules [T-InT] and [T-OutT] respectively to type input and output processes. The rule says that we can project termination types, and end points of termination channel types. Note that we do not have an analogous rule for session types, for which we inspect the

shape of the context. Rule [T-If] permits to type an if-then-else process which compares the identity of two variables. The types of the variables must be composable; this is to preserve the affine discipline of sessions, so that a process cannot use a session twice whenever the two contrasted variables are identical. Rule [T-Par] permits to type check two processes put in parallel such that the composition of the two respective type environments is defined. As introduced, the use of the $\otimes$ operator enforces that the local behaviour of processes can be joined together while preserving the affine invariant, so that we cannot have two processes using the same end of a session. In the replication rule, [T-Repl], we require the environment to be terminated, so that the process under replication could not contain free sessions. The rule for restriction [T-Res] permits to type a fresh variable having a balanced type.

For input we have three rules [T-In],[T-In-L], and [T-InT]; [T-In] and [T-In-L] are used whenever the input channel is a session, and [T-InT] is performed otherwise. We avoid to indicate rule [T-In-R] which is the analogous of [T-In-L] but that the end point of the session is on the right. Rule [T-In-L] permits to type an input channel x by using the end point type S on the left of a type (S, S'). If x is typed with $?\tilde{B}.S$, we know that the bound variables $\tilde{y}$ are of type $\tilde{B}$, and we type check P under the extra assumption $\tilde{y}\colon \tilde{B}$. Equally important is the fact that the continuation uses channel x at continuation type (S, S'), that is, process $x(\tilde{y}\colon \tilde{B}).P$ uses channel x at type $(?\tilde{B}.S, S')$ whereas P may use the *same* channel this time at type (S, S'). Rule [T-In] permits to type an input x described by a session type S by following the same mechanism: the continuation uses channel x at continuation type S while $x(\tilde{y}\colon \tilde{B}).P$ uses channel x at type $?\tilde{B}.S$. On contrast, rule [T-InT] permits to type process $x(\tilde{y}\colon \tilde{B}).P$ given that the judgement $\Gamma \vdash x\colon ?\tilde{B}$ can be inferred by using [T-VarT], and that the continuation P is typed by using the same context Γ extended with the variables $\tilde{y}$ at types $\tilde{B}$.

Then three rules for typing an output follow. Symmetrically, we have three rules for typing a session, [T-Out], [T-Out-L], and one rule for typing an output channel described by a termination type, [T-OutT]. Again, we avoid to indicate rule [T-Out-R] which is the analogous of [T-Out-L] but that the end point of the session is on the right. In the typing rules for output we account for both sending variables having a termination type and for sending variables having a session type, which means *delegating* one or both ends of a session. To illustrate the delegation mechanism, which relies on the type composition operator $\otimes$, we start with the rule for typing a channel with a termination type. Rule [T-OutT] permits to send variables $\tilde{v}$ at types $\tilde{T}$ on an output channel x given that we can infer $\Gamma \vdash x\colon !\tilde{T}$ by using [T-VarT]. If the continuation P is typed with the environment $\Gamma, \tilde{v}\colon \tilde{T}'$, and if the composition of $\tilde{T}$ and $\tilde{T}'$ is defined, then we can infer the judgement $\Gamma, \tilde{v}\colon \tilde{T} \otimes \tilde{T}' \vdash \overline{x}\langle\tilde{v}\rangle.P$. So if $\Gamma \vdash x\colon !(S_1, S_2)$ with S_1, S_2 session types, then the continuation P must be typed with $\Gamma, v\colon \top$, i.e. the session channel cannot be used in input or output. The indexes i, j in the formulation of the rule permit to switch types on the left and of the right of the composition. Rule [T-Out-L] permits to send variables $\tilde{v}$ at types $\tilde{T}_1$ on an output channel x having the session type $(!\tilde{T}_1.S, S')$. The continuation must be typed with an environment which uses channel x at type (S, S'), and which uses the variables at types $\tilde{T}_2$, provided that $(\tilde{T}_1 \otimes \tilde{T}_2)\downarrow$. This will permit us to infer the judgement $\Gamma, \tilde{v}\colon \tilde{T}_1 \otimes \tilde{T}_2, x\colon (!\tilde{T}_1.S, S') \vdash \overline{x}\langle\tilde{v}\rangle.P$. Similarly, given x hav-

ing the session type $(!\tilde{T}_2.S, S')$, and the continuation being typed by $\Gamma, x\colon (S, S'), \tilde{v}\colon \tilde{T}_1$, the rule permits to infer $\Gamma, \tilde{v}\colon \tilde{T}_1 \otimes \tilde{T}_2, x\colon (!\tilde{T}_2.S, S') \vdash \overline{x}\langle\tilde{v}\rangle.P$. This is the reason for the indexes in the three output rules (cf.$(*)$). Rule [T-Out] is to send variables $\tilde{v}$ at types $\tilde{T}_i$ on an output channel x having the session type $!\tilde{T}_i.S$. The continuation is typed with an environment which uses channel x at type S, and which uses the variables at types $\tilde{T}_j$, given that $\tilde{T}_i \otimes \tilde{T}_j$ is defined for $i = 1, 2$ and $j = i + 1(mod\, 2)$, following the same mechanism described above. If this is the case, then we can infer the jugdement $\Gamma, \tilde{v}\colon \tilde{T}_1 \otimes \tilde{T}_2, x\colon !\tilde{T}_i.S \vdash \overline{x}\langle\tilde{v}\rangle.P$.

Example 2.1. We show how we can type services using a protocol relying on a database of malicious sessions that offers operations to add and to check for untrusted channels; the database runs in parallel with the web service. Untrusted channels are stored in a set implemented as a pi calculus process and are typed with $\top$. By providing for channel matching, we can implement a process $\mathsf{if}\, y \in \mathsf{set}\, \mathsf{then}\, P\, \mathsf{else}\, Q$ that continues as P if channel y is stored in the set of untrusted channels, and continues as Q otherwise. We omit all details and refer to (Giunti *et al.* 2009) for an implementation of such protocol. What is interesting for our purposes is to show how we can control the behaviour of services using the database by using the typing discipline introduced in this section. More in detail, services may query the database for the trust of a session by sending the session identifier at type top without being obliged to delegate the session. The code below illustrates such core mechanism; we let d be a communication channel with the database carrying two values: the session identifier s, and an address a to acknowledge the trust of the session.

$$\begin{aligned} \mathsf{check}\ s\ \mathsf{in}\ P &\stackrel{\text{def}}{=} (\nu a\colon B)\overline{d}\langle s, a\rangle.a(x\colon \mathrm{Ack}).\mathsf{if}\, x = \mathrm{ok}\, \mathsf{then}\, P \\ B &\stackrel{\text{def}}{=} (?\mathrm{Ack}.\mathsf{end}, !\mathrm{Ack}.\mathsf{end}) \end{aligned}$$

Let Γ be an environment such that $\Gamma(d) = !T'$ with $T' = \{\top, !\mathrm{Ack}.\mathsf{end}\}$, and such that $\Gamma(\mathrm{ok}) = \top$. The delegation mechanism provides for sending one end of the channel for the ack to the database (the output part), while the other end (the input part) is retained by the process in order to receive the reply from the database. Now by assuming that there is some T such that $\Gamma, s : T, a\colon \mathsf{end}, x\colon \mathrm{Ack} \vdash P$, we can type the process above with the following derivation.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, s : T, a\colon \mathsf{end}, x\colon \mathrm{Ack} \vdash P}{\Gamma, s : T, a\colon \mathsf{end}, x\colon \mathrm{Ack} \vdash \mathsf{if}\, x = \mathrm{ok}\, \mathsf{then}\, P}\ ([\text{T-If}])}{\Gamma, s : T, a\colon ?\mathrm{Ack}.\mathsf{end} \vdash a(x\colon \mathrm{Ack}).\mathsf{if}\, x = \mathrm{ok}\, \mathsf{then}\, P}\ ([\text{T-In}])}{\Gamma, s : T \otimes \top, a\colon ?\mathrm{Ack}.\mathsf{end} \otimes !\mathrm{Ack}.\mathsf{end} \vdash \overline{d}\langle s, a\rangle.a(x\colon \mathrm{Ack}).\mathsf{if}\, x = \mathrm{ok}\, \mathsf{then}\, P}\ ([\text{T-OutT}])}{\Gamma, s : T \vdash \mathsf{check}\ s\ \mathsf{in}\ P}\ ([\text{T-Res}])$$

Example 2.2. We show that process P' of the introduction, which we claim to be deadlocked, is well-typed. Let the types I, U be a short respectively for RefId and User,

and define

$$P' \stackrel{\text{def}}{=} \overline{s}\langle u\rangle.s(x\colon I).\mathsf{if}\, x = r\ \mathsf{then}\, Q\, \mathsf{else}\, P_2$$
$$Q \stackrel{\text{def}}{=} s(y\colon U).\mathsf{if}\, y = g\, \mathsf{then}\, \overline{s}\langle m\rangle.\mathbf{0}$$
$$T \stackrel{\text{def}}{=} (!U.?I.\mathsf{end}, ?U.!I.\mathsf{end})$$

Take an environment Γ such that $\Gamma(g) = \top$ and $\Gamma(r) = \top$, i.e. only the identities of the user g and of the reference r are known to Γ, and assume that we have the judgement $(*)$ $\Gamma, s\colon (?U.!I.\mathsf{end}, \mathsf{end}), x\colon I, u : \top, m : I \vdash P_2$, which we will use to type the else branch of process P'. We have the following derivation.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, s\colon (\mathsf{end}, \mathsf{end}), x\colon I, y\colon U, u\colon \top, m\colon \top \vdash \mathbf{0}}\,([\text{T-Inact}])}{\Gamma, s\colon (!I.\mathsf{end}, \mathsf{end}), x\colon I, y\colon U, u\colon \top, m\colon I \vdash \overline{s}\langle m\rangle.\mathbf{0}}\,([\text{T-Out-L}])}{\Gamma, s\colon (!I.\mathsf{end}, \mathsf{end}), x\colon I, y\colon U, u\colon \top, m\colon I \vdash \mathsf{if}\, y = g\, \mathsf{then}\, \overline{s}\langle m\rangle.\mathbf{0}}\,([\text{T-If}])}{\Gamma, s\colon (?U.!I.\mathsf{end}, \mathsf{end}), x\colon I, u\colon \top, m\colon I \vdash Q \qquad (*)}\,([\text{T-In-L}])}{\Gamma, s\colon (?U.!I.\mathsf{end}, \mathsf{end}), x\colon I, u\colon \top, m\colon I \vdash \mathsf{if}\, x = r\ \mathsf{then}\, Q\, \mathsf{else}\, P_2}\,([\text{T-If}])}{\Gamma, s\colon (?U.!I.\mathsf{end}, ?I.\mathsf{end}), u\colon \top, m\colon I \vdash s(x\colon I).\mathsf{if}\, x = r\ \mathsf{then}\, Q\, \mathsf{else}\, P_2}\,([\text{T-In-R}])}{\Gamma, s\colon T, u\colon U, m\colon I \vdash P'}\,([\text{T-Out-R}])$$

The first lemma illustrates the properties of the type composition operator.

Lemma 2.3. The following hold.

1 The composition operation $\otimes$ is associative, has type $\top$ as identity, and the idempotent elements E, (E_1, E_2);
2 For all types T_1, T_2, we have $(T_1 \otimes T_2)\downarrow$ iff $(T_2 \otimes T_1)\downarrow$.

Composition of types is not commutative; in fact, we have $S_1 \otimes S_2 = (S_1, S_2)$ while $S_2 \otimes S_1 = (S_2, S_1)$. However, the order of two end points in a type can be exchanged while preserving the typings, as stated in the next lemma. The proof of this result is by induction on the length of the inference for the judgement, and crucially relies on typing for termination channels ([T-VarT]), on symmetric rules for input ([T-In-L] and [T-In-R]) and output ([T-Out-L] and [T-Out-R]), and on the delegation mechanism in the rules for output, which permit to switch the order of compositions.

Lemma 2.4 (Exchange). If $\Gamma, x\colon (S_1, S_2) \vdash P$ then $\Gamma, x\colon (S_2, S_1) \vdash P$.

The next lemma says that we can weaken judgements by adding typings for channel types and for end point types. Note that add a session typing does preserve the affine type discipline. The proof is by induction on the structure of the process.

Lemma 2.5 (Weakening). The following hold.

1 If $\Gamma \vdash P$ then $\Gamma, x : T \vdash P$;
2 If $\Gamma, x : S \vdash P$ then $\Gamma, x : (S, S') \vdash P$.

The next lemma says that we can strengthen judgements by removing typings, whenever the variable for the typing does not occur in the free variables of the process. The proof is by induction on the structure of the process.

Lemma 2.6 (Strengthening). If $\Gamma, x\colon T \vdash P$ and $x \notin \mathrm{fv}(P)$ then $\Gamma \vdash P$.

Reduction preserves typability of balanced environments. An environment Γ is balanced if it is a) the empty environment or b) a map from variables to balanced types. To understand why subject reduction does not hold for unbalanced contexts, consider the process $P \stackrel{\text{def}}{=} a(x\colon B).(\overline{x}\langle z\rangle \mid x(y).P')$ where $B \stackrel{\text{def}}{=} (?C.\mathsf{end}, !C.\mathsf{end})$. Now if we let P to interact with $Q \stackrel{\text{def}}{=} (\nu b\colon B)\overline{a}\langle b\rangle.\overline{b}\langle w\rangle$, then the affine discipline for sessions is broken, since the synchronization of $P \mid Q$ eventually leads to a redex having two outputs over b. The problem is that in the type $T \stackrel{\text{def}}{=} (?B, !(?C.\mathsf{end}))$, which describes the use of a in $P \mid Q$, the type expected in input from a by P is different to the one sent in output over a by Q, i.e. T is unbalanced.

As usual, preservation of typability relies on a substitution Lemma. The proof of this result is similar to that of (Giunti and Vasconcelos 2010) but easier; see the appendix for more details. This is because the composition of two end point types is always defined whereas the corresponding split operation of (Giunti and Vasconcelos 2010) does not allow combination of unequal termination types.

Lemma 2.7 (Substitution). Let $\Gamma, \tilde{v}\colon \tilde{T}, \tilde{x}\colon \tilde{T}' \vdash P$ and assume $|\tilde{v}| = |\tilde{x}|$. If $(\tilde{T} \otimes \tilde{T}') \downarrow$ then $\Gamma, \tilde{v}\colon \tilde{T} \otimes \tilde{T}' \vdash P[\tilde{v}/\tilde{x}]$.

The next lemma is the core of subject reduction; for its proof we refer to the appendix. We let $\Gamma, x : T \stackrel{x}{\rightarrowtail} \Gamma, x : T'$ whenever $T \rightarrowtail T'$.

Lemma 2.8. Let $\Gamma \vdash P$ with Γ balanced and assume $P \stackrel{\mu}{\longrightarrow} P'$.

i) If $\mu = x$ then $\Gamma' \vdash P'$ with $\Gamma \stackrel{x}{\rightarrowtail} \Gamma'$;
ii) If $\mu = \tau$ then $\Gamma \vdash P'$.

By using Lemma 2.8, we obtain the main result of this section.

Theorem 2.9 (Subject Reduction). If $\Gamma \vdash P$ with Γ balanced and $P \rightarrow P'$, then $\Gamma' \vdash P'$ with Γ' balanced.

3. Typed behavioral theory

In this section we introduce a typed behavioral theory for pi calculus processes. We contrast typed processes with respect to contexts which behavior is regulated by type checking under adequate assumptions. We represent the knowledge of the observer on channels by means of a type environment. In particular, the type environment for the observer is allowed to contain a end of a session only if this is not exercised by one of the compared processes. The following definition formalizes this intuition.

Definition 3.1 (Compatibility). A type environment Δ is *compatible* with Γ, noted $\Delta \asymp \Gamma$, whenever a) $(\Delta \otimes \Gamma) \downarrow$ and b) $\Delta \otimes \Gamma$ is balanced.

Incidentally, we note that $\Delta \asymp \Gamma$ if and only if $\Gamma \asymp \Delta$.

We are interested in relating processes type checked by type environments that are compatible with the type environment for the observer. The definition below introduces the framework of the behavioural theory.

Definition 3.2 (Type-indexed relations). A type-indexed relation $\mathcal{R}$ is a family of binary relations over processes indexed by type environments. We let $\Delta \models P \mathcal{R} Q$ whenever there exist Γ_1, Γ_2 such that:

— $\Gamma_1 \vdash P$ and $\Gamma_2 \vdash Q$;
— $\Delta \asymp \Gamma_1$ and $\Delta \asymp \Gamma_2$.

In the definition above, the index corresponds to the type environment for the observer, since it possible to notice observables of processes only having the appropriate typing assumptions. The definition below formalizes this concept. Given a balanced type environment Δ such that $\Delta \vdash x\colon ?\tilde{U}.S$ or $\Delta \vdash x\colon ?\tilde{U}$, we let $\Delta^?(x)$ be defined, noted $\Delta^?(x)\downarrow$, and equal to $\tilde{U}\colon \Delta^?(x) = \tilde{U}$. Similarly whenever we have the judgment $\Delta \vdash x\colon !\tilde{U}.S$ or the judgment $\Delta \vdash x\colon !\tilde{U}$ we have $\Delta^!(x)\downarrow$ and $\Delta^!(x) = \tilde{U}$. Notice that the balanced hypothesis ensures that if $\Delta(x) = (B_1, B_2)$ then $\Delta^?(x) = \Delta^!(x)$.

The observables of our theory are defined in terms of barbs (Sangiorgi and Walker 2001). Whenever $\Delta^!(x)\downarrow$, an output barb on x can be observed if the process is structurally equivalent to a context containing the (free) output x. Conversely, whenever $\Delta^?(x)\downarrow$ an input barb on x can be observed if the process contains a (free) input over x.

Definition 3.3 (Barbs). Let Δ be a type environment and P be a process. We let:

— $\Delta \models P\downarrow_{x!}$ whenever $P \equiv (\nu\tilde{x}\colon \tilde{A})(\overline{x}\langle\tilde{v}\rangle.P \mid Q)$ and $x \notin \tilde{x}$ and $\Delta^?(x)\downarrow$;
— $\Delta \models P\downarrow_{x?}$ whenever $P \equiv (\nu\tilde{x}\colon \tilde{A})(x(\tilde{y}\colon \tilde{A}').P \mid Q)$ and $x \notin \tilde{x}$ and $\Delta^!(x)\downarrow$.

Weak barbs are defined in terms of weak reductions (cf. Section 2). For each observable $\mathcal{O} \in \{x?, x!\}$ we let $\Delta \models P\Downarrow_{\mathcal{O}}$ whenever $P \Rightarrow P'$ and $\Delta \models P'\downarrow_{\mathcal{O}}$. We write $\Delta \not\models P\Downarrow_{\mathcal{O}}$ to indicate that does not exist P' such that $P \Rightarrow P'$ and $\Delta \models P'\downarrow_{\mathcal{O}}$.

Definition 3.4 (Barb preservation). A type-indexed relation $\mathcal{R}$ is barb preserving whenever $\Delta \models P \mathcal{R} Q$ and

— $\Delta \models P\downarrow_{x!}$ implies $\Delta \models Q\Downarrow_{x!}$
— $\Delta \models P\downarrow_{x?}$ implies $\Delta \models Q\Downarrow_{x?}$.

Our behavioral theory ignores reductions of processes, since we assume such to be internal steps invisible to the observer.

Definition 3.5 (Reduction closure). A type-indexed relation $\mathcal{R}$ is reduction-closed if $\Delta \models P \mathcal{R} Q$ and $P \to P'$ implies that $Q \Rightarrow Q'$ and $\Delta' \models P' \mathcal{R} Q'$.

A crucial point is that related processes must maintain their behavior in all contexts type checked by the observing environment.

Definition 3.6 (Contextuality). A type-indexed relation $\mathcal{R}$ is contextual if

— $\Delta \otimes \Delta_R \models P \mathcal{R} Q$ and $\Delta_R \vdash R$ implies $\Delta \models P \mid R \mathcal{R} Q \mid R$;
— $\Delta \models P \mathcal{R} Q$ implies $\Delta, x : T \models P \mathcal{R} Q$;

— $\Delta, x : T \models P \,\mathcal{R}\, Q$ implies $\Delta \models (\nu x : A)P \,\mathcal{R}\, (\nu x\colon A)Q$.

We are ready to introduce the touchstone equivalence of our calculus.

Definition 3.7 (Typed behavioral equivalence). Typed behavioral equivalence, noted $\cong$, is the largest symmetric type-indexed relation which is barb preserving, reduction closed and contextual.

Whenever does not exist $\mathcal{R}$ such that $\Delta \models P \,\mathcal{R}\, Q$ and $\mathcal{R} \subseteq \cong$ we write $\Delta \not\models P \cong Q$.

Example 3.8. We show why our notion of contextuality, and in particular the clause inherent the immersion of parallel processes, relies on type checking under assumptions compatible with those used to type the related processes. Consider the process $P'' \stackrel{\text{def}}{=} a(s\colon T).P'$ of the introduction, where:

$$P' \stackrel{\text{def}}{=} \overline{s}\langle u\rangle.s(x\colon I).\mathsf{if}\, x = r\ \mathsf{then}\, Q\ \mathsf{else}\, P_2 \qquad Q \stackrel{\text{def}}{=} s(y\colon U).\mathsf{if}\, y = g\ \mathsf{then}\, \overline{s}\langle m\rangle.\mathbf{0}$$

$$\Gamma, a\colon ?T.\mathsf{end} \vdash P'' \qquad T \stackrel{\text{def}}{=} (!U.?I.\mathsf{end}, ?U.!I.\mathsf{end})$$

Our aim is to relate P'' and $a(s\colon T).\mathbf{0}$ with a type-index Δ such that $\Delta \models P'' \,\mathcal{R}\, a(s\colon T).\mathbf{0}$ is contextual. By definition, Δ needs to be compatible with $\Gamma, a\colon ?T.\mathsf{end}$ and Γ' such that $\Gamma' \vdash a(s\colon T).\mathbf{0}$; we avoid to analyze the last requirement, which does not add useful informations. In particular, the first requirement imposes that Δ can use only the output capability of a to send values at type T: $\Delta(a) = !T.\mathsf{end}$ or $\Delta(a) = \top$. Now we can understand how the observer's environment Δ can control the behaviour of contexts, and eventually rule out processes that cannot be composed with P'' and $a(s\colon T).\mathbf{0}$. Take the context of the introduction: $C'' \stackrel{\text{def}}{=} (\nu s'\colon T)(\overline{a}\langle s'\rangle \mid s'(y).\overline{\omega}\langle y\rangle)$ where $a, \omega \neq s'$, and assume that $\Delta^?(\omega)\downarrow$. We can easily see that C'' is not a valid context for $\mathcal{R}$, since there is H such that $P'' \mid C'' \Rightarrow H$ and $\Delta \models H\downarrow_{\omega!}$, while $\Delta \not\models (a(s\colon T).\mathbf{0} \mid C'')\Downarrow_{\omega!}$. In fact Δ does not accept this context: $\Delta \not\vdash C''$. In the non trivial case, that is $\Delta(a) \neq \top$, this holds since C'' both delegates the channel s' at type $\Delta^!(a) = T$, and use s' at the input type $?U.!I.\mathsf{end}$: that is, rule [T-Par] cannot be used to accept the parallel process $\overline{a}\langle s'\rangle \mid s'(y).\overline{\omega}\langle y\rangle$, because the composition of type T and type $?U.!I.\mathsf{end}$ is undefined.

4. Bisimulation semantics

In this section we provide for a co-inductive proof technique for typed behavioral equivalence based on bisimulation. We define semantics over typed *configurations* which depict the interaction among the process and the context represented by a type environment.

Definition 4.1. A pair $\Delta \triangleleft P$ is a configuration whenever there exist Γ such that
— $\Gamma \vdash P$;
— $\Delta \asymp \Gamma$.

The semantics of configurations are expressed in Figure 4 in terms of a labelled transition system. The labels are the following:

$$\alpha \ ::= \ (\tilde{y}\colon \tilde{T})\, x(\tilde{v}) \mid (\tilde{y})\, \overline{x}\langle \tilde{v}\rangle \mid \mu$$

$$\frac{|\tilde{y}| = |\tilde{v}| \qquad (*)}{\Delta, \tilde{v}\colon \tilde{B}_1 \otimes \tilde{B}_2, x\colon !\tilde{B}_i.C \triangleleft x(\tilde{y}\colon \tilde{B}_i).P \xrightarrow{x(\tilde{v})} \Delta, \tilde{v}\colon \tilde{B}_j, x\colon C \triangleleft P[\tilde{v}/\tilde{y}]} \quad [\text{L-In}]$$

$$\frac{}{\Delta, \tilde{v}\colon \tilde{B}_1, x\colon ?\tilde{B}_2.C \triangleleft \overline{x}\langle\tilde{v}\rangle.P \xrightarrow{\overline{x}\langle\tilde{v}\rangle} \Delta, \tilde{v}\colon \tilde{B}_1 \otimes \tilde{B}_2, x\colon C \triangleleft P} \quad [\text{L-Out}]$$

$$\frac{|\tilde{y}| = |\tilde{v}| \qquad (*)}{\Delta, \tilde{v}\colon \tilde{B}_1 \otimes \tilde{B}_2, x\colon (!\tilde{B}_i.C, C') \triangleleft x(\tilde{y}\colon \tilde{B}_i).P \xrightarrow{x(\tilde{v})} \Delta, \tilde{v}\colon \tilde{B}_j, x\colon (C, C') \triangleleft P[\tilde{v}/\tilde{y}]} \quad [\text{L-In-L}]$$

$$\frac{}{\Delta, \tilde{v}\colon \tilde{B}_1, x\colon (?\tilde{B}_2.C, C') \triangleleft \overline{x}\langle\tilde{v}\rangle.P \xrightarrow{\overline{x}\langle\tilde{v}\rangle} \Delta, \tilde{v}\colon \tilde{B}_1 \otimes \tilde{B}_2, x\colon (C, C') \triangleleft P} \quad [\text{L-Out-L}]$$

$$\frac{\Delta, \tilde{v} : \tilde{B}_1 \otimes \tilde{B}_2 \vdash x\colon !\tilde{B}_i \qquad |\tilde{y}| = |\tilde{v}| \qquad (*)}{\Delta, \tilde{v}\colon \tilde{B}_1 \otimes \tilde{B}_2 \triangleleft x(\tilde{y}\colon \tilde{B}_i).P \xrightarrow{x(\tilde{v})} \Delta, \tilde{v}\colon \tilde{B}_j \triangleleft P[\tilde{v}/\tilde{y}]} \quad [\text{L-InT}]$$

$$\frac{\Delta, \tilde{v}\colon \tilde{B}_1 \vdash x\colon ?\tilde{B}_2}{\Delta, \tilde{v}\colon \tilde{B}_1 \triangleleft \overline{x}\langle\tilde{v}\rangle.P \xrightarrow{\overline{x}\langle\tilde{v}\rangle} \Delta, \tilde{v}\colon \tilde{B}_1 \otimes \tilde{B}_2 \triangleleft P} \quad [\text{L-OutT}]$$

$$\frac{\Delta, y\colon B \triangleleft P \xrightarrow{(\tilde{y}:\tilde{B})x(\tilde{v})} \Delta' \triangleleft P' \qquad y \notin \{x, \tilde{y}\}}{\Delta \triangleleft P \xrightarrow{(y\colon B, \tilde{y}\colon \tilde{B})\, x(\tilde{v})} \Delta' \triangleleft P'} \qquad \frac{\Delta, y\colon \top \triangleleft P \xrightarrow{(\tilde{y})\overline{x}\langle\tilde{v}\rangle} \Delta' \triangleleft P'}{\Delta \triangleleft (\nu y\colon B)P \xrightarrow{(y,\tilde{y})\,\overline{x}\langle\tilde{v}\rangle} \Delta' \triangleleft P'} \quad [\text{L-Weak}],[\text{L-Open}]$$

$$\frac{P \xrightarrow{\mu} P'}{\Delta \triangleleft P \xrightarrow{\mu} \Delta \triangleleft P'} \qquad \frac{\Delta \triangleleft P \xrightarrow{\alpha} \Delta' \triangleleft P' \qquad \mathrm{bv}(\alpha) \cap \mathrm{fv}(Q) = \emptyset}{\Delta \triangleleft P \mid Q \xrightarrow{\alpha} \Delta' \triangleleft P' \mid Q} \quad [\text{L-Red}],[\text{L-Par}]$$

$$\frac{\Delta, x\colon \top \triangleleft P \xrightarrow{x} \Delta, x\colon \top \triangleleft P' \qquad B \rightarrowtail B'}{\Delta \triangleleft (\nu x\colon B)P \xrightarrow{\tau} \Delta \triangleleft (\nu x\colon B')P'} \quad [\text{L-Res-T}]$$

$$\frac{\Delta, x\colon \top \triangleleft P \xrightarrow{\alpha} \Delta', x\colon \top \triangleleft P' \qquad x \notin \mathrm{var}(\alpha)}{\Delta \triangleleft (\nu x\colon B)P \xrightarrow{\alpha} \Delta' \triangleleft (\nu x\colon B)P'} \quad [\text{L-Res}]$$

$$\frac{\Delta \triangleleft P \xrightarrow{\alpha} \Delta' \triangleleft P'}{\Delta \triangleleft !P \xrightarrow{\alpha} \Delta' \triangleleft P' \mid !P} \quad [\text{L-Repl}]$$

$$(*)\ i \in \{1, 2\},\ j = i + 1 (\mathrm{mod}\ 2)$$

Fig. 4. Typed labelled semantics

The first label denotes possible fresh input of variables $\tilde{v}$ on a variable x; we assume $\tilde{y} \subseteq \tilde{v}$. Notice that the types for the variables are indicated. The second label denotes an output of possibly fresh variables $\tilde{v}$ on a variable x; we let $\tilde{y} \subseteq \tilde{v}$. The last label, μ, which ranges over τ and variables x, y, z as in Section 2, denotes an internal transition which is not observable by the typed observer. We let the bound variables of α, noted $\mathrm{bv}(\alpha)$, be the set $\{\tilde{y}\}$ whenever $\alpha = (\tilde{y}\colon \tilde{T})\, x(\tilde{v})$ or $\alpha = (\tilde{y})\, \overline{x}\langle\tilde{v}\rangle$, and the empty set otherwise.

The variables of α, noted $\mathrm{var}(\alpha)$, are all variables occurring in α. The free variables of α, noted $\mathrm{fv}(\alpha)$, are the set $\mathrm{var}(\alpha) \setminus \mathrm{bv}(\alpha)$.

Symmetrically to the type system of Figure 3, in the typed transitions of Figure 4 we have three rules for input, [L-In],[L-In-L], and [L-InT], and three rules for output, [L-Out],[L-Out-L], and [L-OutT]. Rule [L-In] and [L-Out] apply when the type for the input/output channel of the observer's environment is a session type of the form S, while rules [L-In-L] and [L-Out-L] apply when such type has the form (S, S') with S a session type. We avoid to indicate the symmetric right rules of [L-In-L] and [L-Out-L] that apply when the observer's type for the channel has the form (S', S) with S a session type. Then we have rules [L-InT] and [L-OutT], which apply when the type for the channel is a termination type. In rule [L-In], an environment which knows x at session type $!\tilde{B}_i.C$, $i \in \{1, 2\}$, and possess variables $\tilde{v}$ at types $\tilde{B}_1 \otimes \tilde{B}_2$, interacts with a process waiting for input on x by sending to it the variables $\tilde{v}$. The result of the interaction is that the variables $\tilde{v}$ are assigned to the formal parameters of the input in the continuation, and that the environment for the redex looses the capabilities $\tilde{B}_i$ over $\tilde{v}$ and unrolls the type for x to C. For instance if the initial environment is $\Delta, v\colon (?B.\mathsf{end}, !B.\mathsf{end}), x\colon !(?B.\mathsf{end}).C$ then the environment for the redex is $\Delta, v\colon !B.\mathsf{end}, x\colon C$. Rule [L-In-L] performs the same behaviour when x is known at type $(!\tilde{B}_i.C, C')$. Rule [L-InT] is similar to rule [L-In] and applies when the input channel x of the process is known by the environment at termination type $!\tilde{B}_i$. Note that in the environment of the redex the type for x does not change, since it is terminated. In rule [L-Out], an environment which knows x at session type $?\tilde{B}_2.C$ interacts with a process sending variables $\tilde{v}$ over x. The result of the interaction is that the process progresses to the continuation while the environment acquires the variables $\tilde{v}$ at types $\tilde{B}_2$ and unrolls the type for x to C. For instance if the initial environment is $\Delta, v\colon !B.\mathsf{end}, x\colon ?(?B.\mathsf{end}).C$ then the environment for the redex is $\Delta, v\colon (!B.\mathsf{end}, ?B.\mathsf{end}), x\colon C$. Note that the balanced hypothesis makes sure the types for the variables $\tilde{v}$ in the redex environment to be defined. Rule [L-Out-L] is analogous, while in rule [L-OutT] the environment of the redex assigns to the output channel of the process the same termination type of the hypothesis. In rule [L-Weak] the context dynamically creates new variables and it sends them to the receiving process. Scope extrusion is performed by rule [L-Open]. Rule [L-Red] says that reduction defined in Figure 1 is invisible to the context. The two rules for restriction [L-Res-T] and [L-Res] are the counterparts respectively of rules [R-Res-T] and [R-Res] in Figure 3.

The following result establishes that the rules in Figure 3 do represent a transition system. The proof is in the appendix.

Proposition 4.2. Let $\Delta \triangleleft P$ be a configuration and assume that $\Delta \triangleleft P \xrightarrow{\alpha} \Delta' \triangleleft P'$. Then $\Delta' \triangleleft P'$ is a configuration.

We define bisimilarity over the typed labelled transition system.

Definition 4.3 (Bisimilarity). A symmetric relation $\mathcal{R}$ over configurations is a bisimulation if whenever $(\Delta \triangleleft P)\,\mathcal{R}\,(\Delta \triangleleft Q)$ and $\Delta \triangleleft P \xrightarrow{\alpha} \Delta' \triangleleft P'$ then $\Delta \triangleleft Q \stackrel{\hat{\alpha}}{\Longrightarrow} \Delta' \triangleleft Q'$ and $(\Delta' \triangleleft P')\,\mathcal{R}\,(\Delta' \triangleleft Q')$, where $\hat{\mu}$ is the empty string and $\hat{\alpha} = \alpha$ otherwise. Bisimilarity, noted $\approx$, is the largest bisimulation. We write $\Delta \models P \approx Q$ as a short for $\Delta \triangleleft P \approx \Delta \triangleleft Q$.

The first lemma describes the shape of a process executing a typed action and of the typed observer. The proof is a simple argument by induction on the length of the reference; we omit all the details and refer to (Hennessy 2007).

Lemma 4.4. The following hold.

1 If $\Delta \triangleleft P \xrightarrow{(\tilde{y}:\tilde{T})\,x(\tilde{v})} \Delta' \triangleleft P'$ then $\Delta^{!}(x)\downarrow$, $P \equiv (\nu\tilde{x}\colon \tilde{A})(x(\tilde{z}\colon \tilde{B}).Q_1 \mid Q_2)$ with $|\tilde{x} = \tilde{v}$— and $x \notin \tilde{x}$, and $P' \equiv (\nu\tilde{x}\colon \tilde{A})(Q_1[\tilde{v}/\tilde{z}] \mid Q_2)$;

2 If $\Delta \triangleleft P \xrightarrow{(\tilde{y})\overline{x}\langle\tilde{v}\rangle} \Delta' \triangleleft P'$ then $\Delta^{?}(x)\downarrow$, $P \equiv (\nu\tilde{y}\colon \tilde{A}, \tilde{z}\colon \tilde{B})(\overline{x}\langle\tilde{v}\rangle.Q_1 \mid Q_2)$ with $x \notin \tilde{z}$, and $P' \equiv (\nu\tilde{z}\colon \tilde{A})(Q_1 \mid Q_2)$.

The first proposition establishes the correspondence among the μ-actions of Figure 1 and the μ-actions of Figure 4.

Proposition 4.5. $P \xrightarrow{\mu} P'$ if and only if $\Delta \triangleleft P \xrightarrow{\mu} \Delta \triangleleft P'$.

Proof. The left direction is immediate since from $P \xrightarrow{\mu} P'$ and [L-Red] we infer $\Delta \triangleleft P \xrightarrow{\mu} \Delta \triangleleft P'$. For the opposite direction, we proceed by induction on the length of the reference. In case [L-Red] from $\Delta \triangleleft P \xrightarrow{\mu} \Delta \triangleleft P'$ we infer $P \xrightarrow{\mu} P'$ and we have done. In case [L-Par] we have $\Delta \triangleleft P \mid Q \xrightarrow{\mu} \Delta \triangleleft P' \mid Q$ inferred from $\Delta \triangleleft P \xrightarrow{\mu} \Delta' \triangleleft P'$. By I.H. we have $P \xrightarrow{\mu} P'$. We apply [R-Par] and infer $P \mid Q \xrightarrow{\mu} P' \mid Q'$. In case [L-Res-T] we have $\Delta \triangleleft (\nu x : T)P \xrightarrow{\tau} \Delta \triangleleft (\nu x : T)P'$ inferred from $\Delta \triangleleft P \xrightarrow{x} \Delta \triangleleft P'$ and $T \rightarrowtail T'$. By I.H. we have $P \xrightarrow{x} P'$. We apply [R-Res-T] and infer $(\nu x : T)P \xrightarrow{\tau} (\nu x : T')P'$. Case [L-Res] is analogous. In case [L-Repl] we have $\Delta \triangleleft !P \xrightarrow{\mu} \Delta \triangleleft P' \mid !P$ inferred from $\Delta \triangleleft P \xrightarrow{\mu} \Delta \triangleleft P'$. By I.H. we have $P \xrightarrow{\mu} P'$. We apply [R-Par] and infer $P \mid !P \xrightarrow{\mu} P' \mid !P$. An application of [R-Struct] with $!P \equiv P \mid !P$ give us the desired result, $!P \xrightarrow{\mu} P' \mid !P$. □

In the following, we establish the contextuality of $\approx$. We start by proving that $\approx$ is preserved by restriction. We need a preparatory lemma, which says that the type $\top$ is less informative than the other types. The proof is in the appendix.

Lemma 4.6. If $\Delta, x\colon T \models P \approx Q$ then $\Delta, x\colon \top \models P \approx Q$.

Proposition 4.7 (Closure under restriction). If $\Delta, x\colon T \models P \approx Q$ then $\Delta \models (\nu x\colon A)P \approx (\nu x\colon B)Q$.

Proof. Let $\Delta \models M\,\mathcal{R}\,N$ whenever a) $M = (\nu y\colon A)P$ and $N = (\nu y\colon B)Q$ and $\Delta, y\colon \top \models P \approx Q$ or b) $\Delta \models M \approx N$. We proceed by co-induction and show that $\mathcal{R}$ is contained in $\approx$, hence proving that $\Delta, x\colon \top \models P \approx Q$ implies $\Delta \models (\nu x\colon A)P \approx (\nu x\colon B)Q$. The result then follows by Lemma 4.6. It is easy to see that $\mathcal{R}$ is a type-indexed relation. To see that $\mathcal{R}$ is a bisimulation, we show case (a); case (b) is obtained directly by exploiting the properties of $\approx$. Let $\Delta \triangleleft M \xrightarrow{\alpha} \Delta' \triangleleft P'$ be inferred by [L-Open] since $\alpha = (y, \tilde{y})\overline{x}\langle\tilde{v}\rangle$ and $\Delta, y\colon \top \triangleleft P \xrightarrow{(\tilde{y})\overline{x}\langle\tilde{v}\rangle} \Delta' \triangleleft P'$. We find Q' s.t.

$\Delta, y\colon \top \triangleleft Q \Longrightarrow \xrightarrow{(\tilde{y})\overline{x}\langle\tilde{v}\rangle} \Longrightarrow \Delta' \triangleleft Q'$ and $\Delta' \models P' \approx Q'$. By applications of [L-Res] and [L-Open] we infer $\Delta \triangleleft N \Longrightarrow \xrightarrow{(y,\tilde{y})\overline{x}\langle\tilde{v}\rangle} \Longrightarrow \Delta' \triangleleft Q'$. From $\Delta' \models P' \mathcal{R} Q'$ we obtain the desired result. Consider case [L-Res-T]. We have $\Delta \triangleleft (\nu y\colon A)P \xrightarrow{\tau} \Delta \triangleleft (\nu y\colon A')P'$ inferred from $\Delta, y\colon \top \triangleleft P \xrightarrow{y} \Delta, y\colon \top \triangleleft P'$ and $A \rightarrowtail A'$. We infer that there is Q' such that $\Delta \triangleleft Q \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_n} \Delta \triangleleft Q'$ and $\Delta, y\colon \top \models P' \approx Q'$. By applications of [L-Res] and [L-Res-T] we infer $\Delta \triangleleft (\nu y\colon B)Q \Longrightarrow \Delta \triangleleft (\nu y\colon B')Q'$ where $B \rightarrowtail \cdots \rightarrowtail B'$. We deduce that $\Delta \models (\nu y\colon A')P' \mathcal{R} (\nu y\colon B')Q'$, as required. Case [L-Res] is similar. We have that $\Delta \triangleleft (\nu y\colon A)P \xrightarrow{\alpha} \Delta' \triangleleft (\nu y\colon A)P'$ is inferred from $\Delta, y\colon \top \triangleleft P \xrightarrow{\alpha} \Delta', y\colon \top \triangleleft P'$ with $y \notin \mathrm{var}(\alpha)$. We infer that there is Q' such that $\Delta \triangleleft Q \stackrel{\alpha}{\Longrightarrow} \Delta' \triangleleft Q'$ and $\Delta, y\colon \top \models P' \approx Q'$. By applications of [L-Res] and [L-Res-T] we obtain $\Delta \triangleleft (\nu y\colon B)Q \stackrel{\alpha}{\Longrightarrow} \Delta \triangleleft (\nu y\colon B')Q'$ with $B \rightarrowtail \cdots \rightarrowtail B'$. We conclude that $\Delta \models (\nu y\colon A)P' \mathcal{R} (\nu y\colon B')Q'$. □

To prove that bisimulation is closed under creation of fresh variables, we need the following lemma. The proof is by induction on the length of the reference (cf. (Bugliesi and Giunti 2005)).

Lemma 4.8. The following hold.

1 If $\Delta, y\colon T \triangleleft P \xrightarrow{\alpha} \Delta', y\colon T \triangleleft P'$ and $y \notin \mathrm{fv}(P)$ then $\Delta \triangleleft P \xrightarrow{\alpha} \Delta' \triangleleft P'$.
2 If $\Delta \triangleleft P \xrightarrow{\alpha} \Delta' \triangleleft P'$ and y is fresh to Δ' then $\Delta, y\colon T \triangleleft P \xrightarrow{\alpha} \Delta', y\colon T \triangleleft P'$.

Proposition 4.9 (Closure under new variables). If $\Delta \models P \approx Q$ then $\Delta, x\colon T \models P \approx Q$.

Proof. Let $\Delta \models P \mathcal{R} Q$ whenever a) $\Delta = \Delta_1, y\colon T$ and $\Delta_1 \models P \approx Q$ or b) $\Delta \models P \approx Q$. We show that $\mathcal{R}$ is a bisimulation. First, we note that $\mathcal{R}$ is a type-indexed relation; this result is straightforward obtained by the bisimilarity hypothesis. The interesting case is (a); (b) follows directly by co-induction. First notice that in case (a) we know that $y \notin \mathrm{fv}(P, Q)$ since from $\Delta_1 \models P \approx Q$ we infer $\Gamma_1 \vdash P$ and $\Gamma \vdash Q$ with $\mathrm{dom}(\Gamma_1) = \mathrm{dom}(\Delta_1) = \mathrm{dom}(\Gamma_2)$. Assume $\Delta \triangleleft P \xrightarrow{\alpha} \Delta' \triangleleft P'$. If $\alpha = (\tilde{y}\colon \tilde{T})x(\tilde{v})$ with $y \notin \tilde{v}$, or $\alpha = (\tilde{y})\overline{x}\langle\tilde{v}\rangle$, or $\alpha = \mu$ then we infer $y \notin \mathrm{fv}(\alpha)$. From this we deduce that $\Delta' = \Delta_2, y\colon T$ for some Δ_2. We apply Lemma 4.8(i) and infer $\Delta_1 \triangleleft P \xrightarrow{\alpha} \Delta_2 \triangleleft P'$. From $\Delta_1 \models P \approx Q$ we deduce that there is Q' such that $\Delta_1 \triangleleft Q \stackrel{\alpha}{\Longrightarrow} \Delta_2 \triangleleft Q'$ and $\Delta_2 \models P' \approx Q'$. We apply Lemma 4.8(ii) and infer the following reductions: $\Delta_1, y : T \triangleleft Q \stackrel{\alpha}{\Longrightarrow} \Delta_2, y : T \triangleleft Q'$. From a) we infer $\Delta' \models P' \mathcal{R} Q'$, as required. Otherwise, consider the case $\alpha = (\tilde{y} : \tilde{T})x(\tilde{v})$ with $y \in \tilde{v}$. We apply [L-Weak] and infer $\Delta_1 \triangleleft P \xrightarrow{\alpha} \Delta' \triangleleft P'$. From $\Delta_1 \models P \approx Q$ we deduce that $\Delta_1 \triangleleft Q \stackrel{\alpha}{\Longrightarrow} \Delta' \triangleleft Q'$ and $\Delta' \models P' \approx Q'$. From b) we infer $\Delta' \models P' \mathcal{R} Q'$, as requested. □

Proposition 4.10 (Closure under composition). If $\Delta \otimes \Delta_R \models P \approx Q$ and $\Delta_R \vdash R$ then $\Delta \models P \mid R \approx Q \mid R$.

Proof. Let $\Delta \models (\nu\tilde{x} : \tilde{A})(P \mid R)\ \mathcal{R}\ (\nu\tilde{y} : \tilde{B})(Q \mid R)$ whenever $\Delta, \tilde{z} : \tilde{\top} \otimes \Delta_R, \tilde{z} : \tilde{C} \models P \approx Q$ with $\tilde{z} = \tilde{x} \cup \tilde{y}$, $\Delta_R, \tilde{z} : \tilde{C} \vdash R$. We show that $\mathcal{R}$ is a bisimulation; the statement of the proposition then follows by letting $\tilde{z}$ be the empty tuple. We first

prove that $\mathcal{R}$ is a type-indexed relation. In the following, we show that each of the related processes is typed by an environment compatible with the type-index. By the bisimilarity hypothesis we infer that there are $\Gamma_1 \vdash P, \Gamma_2 \vdash Q$ such that $\Delta, \tilde{z} : \tilde{\top} \otimes \Delta_R, \tilde{z} : \tilde{C} \asymp \Gamma_1$ and $\Delta, \tilde{z} : \tilde{\top} \otimes \Delta_R, \tilde{z} : \tilde{C} \asymp \Gamma_2$. This means that $\Delta_R, \tilde{z} : \tilde{C} \otimes \Gamma_1$ and $\Delta_R, \tilde{z} : \tilde{C} \otimes \Gamma_2$ are defined. From $\Delta_R, \tilde{z} : \tilde{C} \vdash R$, and from the rule for typing for parallel processes, [T-Par], we obtain that $\Delta_R, \tilde{z} : \tilde{C} \otimes \Gamma_1 \vdash P \mid R$ and $\Delta_R, \tilde{z} : \tilde{C} \otimes \Gamma_2 \vdash Q \mid R$. From multiple applications of the rule for restriction, [T-Res], we infer $\Delta_R, \tilde{a} \colon \tilde{A}' \otimes \Gamma_1 \setminus \tilde{x} \vdash (\nu \tilde{x} \colon \tilde{A})(P \mid R)$ and $\Delta_R, \tilde{b} \colon \tilde{B}' \otimes \Gamma_2 \setminus \tilde{y} \vdash (\nu \tilde{y} \colon \tilde{B})(Q \mid R)$ where $\tilde{a} \cup \tilde{x} = \tilde{y}$, and $\tilde{b} \cup \tilde{y} = \tilde{x}$; we let $\Gamma \setminus \tilde{x}$ be the union of typings $y \colon \Gamma(y)$ such that $y \notin \tilde{x}$. Now we know that $\tilde{a} \cap \mathrm{fv}(P, R) = \emptyset$, and $\tilde{b} \cap \mathrm{fv}(Q, R) = \emptyset$, eventually by alpha-renaming $\tilde{x}$ and $\tilde{y}$. By strengthening (Lemma 2.6) we infer $\Delta_R \otimes \Gamma_1 \setminus \tilde{z} \vdash (\nu \tilde{x} \colon \tilde{A})(P \mid R)$ and $\Delta_R \otimes \Gamma_2 \setminus \tilde{z} \vdash (\nu \tilde{y} \colon \tilde{B})(Q \mid R)$. We can now deduce that $\Delta \asymp \Delta_R \otimes \Gamma_1 \setminus \tilde{z}$ and $\Delta \asymp \Delta_R \otimes \Gamma_2 \setminus \tilde{z}$, as required.

To see that $\mathcal{R} \subseteq \approx$, let $\Delta \triangleleft (\nu \tilde{x} : \tilde{A})(P \mid R) \xrightarrow{\alpha} \Delta_1 \triangleleft M$. The interesting situation to analyze is communication among P and R, which arises when $\alpha = \mu$. To tackle the proof, we mimic the synchronization among processes P and R by letting P to interact with the type environment Δ_R. First, notice that when $\alpha = \mu$ we have that $\Delta_1 = \Delta$. In the following, we show the case whether the synchronization arises from process P sending variables to R over a free channel x, i.e. $\alpha = x$. The case whether R sends variables to P is analogous, while the case $\alpha = \tau$ is similar, but simpler. To ease the notation we avoid type decorations whenever unnecessary for the proof. Without loss of generality, let $(\nu \tilde{x})(P \mid R) \xrightarrow{x} M$ be inferred by using [R-Com],[R-Res] and [R-Struct] since $M \stackrel{\text{def}}{=} (\nu \tilde{x}, \tilde{k})(P' \mid R')$, $P \equiv (\nu \tilde{p}, \tilde{k})(\overline{x}\langle \tilde{v} \rangle.P_1 \mid P_2)$, $R \equiv (\nu \tilde{r})(x(\tilde{y} \colon \tilde{B}).R_1 \mid R_2)$, $|\tilde{v}| = |\tilde{y}|$, $P' \equiv (\nu \tilde{p})(P_1 \mid P_2)$, and $R' \equiv (\nu \tilde{r})(R_1[\tilde{v}/\tilde{y}] \mid R_2)$.

From the hypothesis $\Delta_R \vdash R$, we infer that $\Delta_R^?(x)$ is defined and equal to $\tilde{B} \colon \Delta_R^?(x) = \tilde{B}$. Let $\Delta, \tilde{z} : \tilde{\top} \otimes \Delta_R, \tilde{z} : \tilde{C} = \Delta_1$. By applications of [L-Res], [L-Par], and [L-Open] we deduce $\Delta_1 \triangleleft P \xrightarrow{(\tilde{k})\overline{x}\langle \tilde{v} \rangle} \Delta' \triangleleft (\nu \tilde{p})(P_1 \mid P_2)$. From the bisimilarity hypothesis, we find Q' such that $\Delta_1 \triangleleft Q \xRightarrow{(\tilde{k})\overline{x}\langle \tilde{v} \rangle} \Delta' \triangleleft Q'$ and $\Delta' \models (\nu \tilde{p})(P_1 \mid P_2) \approx Q'$. This means that there are Q_0, Q_1 such that $\Delta_1 \triangleleft Q \Longrightarrow \Delta \triangleleft Q_0 \xrightarrow{(\tilde{k})\overline{x}\langle \tilde{v} \rangle} \Delta' \triangleleft Q_1 \Longrightarrow \Delta' \triangleleft Q'$. We infer the shape of Q_0 and Q_1 (cf. Lemma 4.4): $Q_0 \equiv (\nu \tilde{q}, \tilde{k})(\overline{x}\langle \tilde{v} \rangle.Q_1 \mid Q_2)$ and $Q_1 \equiv (\nu \tilde{q})(Q_1 \mid Q_2)$. By applications of [L-Res-T],[L-Res],[L-Par] and [L-Red] we infer $\Delta \triangleleft (\nu \tilde{y})(Q \mid R) \Longrightarrow \Delta \triangleleft N$ where $N \equiv (\nu \tilde{y}, \tilde{k})(Q' \mid R')$. To infer that $\Delta \models M \mathcal{R} N$ we need to show that R' is typed by a suitable environment. This follows by subject reduction (Lemma 2.8): we have $\Delta_R \otimes \Gamma_1 \setminus \tilde{z} \overset{x}{\rightarrowtail} \Omega$ and $\Omega \vdash M$, and $\Delta_R \otimes \Gamma_2 \setminus \tilde{z} \overset{x}{\rightarrowtail} \Omega'$ and $\Omega' \vdash N$. From the judgement above we find $\Gamma_{R'}$ such that $\Gamma_{R'}, \tilde{z} \colon \tilde{C}, \tilde{k} \colon \tilde{K} \vdash R'$ and $\Delta'' \otimes \Gamma_{R'}, \tilde{z} \colon \tilde{C}, \tilde{k} \colon \tilde{K} = \Delta'$ for some tuple of types $\tilde{K}$ and type environment Δ''. □

Based on these results, we establish the soundness of bisimulation as a proof technique for observational equivalence.

Proposition 4.11 (Soundness). If $\Delta \models P \approx Q$ then $\Delta \models P \cong Q$.

Proof. Let $\Delta \models P \mathcal{R} Q$ whenever $\Delta \models P \approx Q$. First, notice that $\mathcal{R}$ is a type-indexed relation since there are Γ_1, Γ_2 such that $\Gamma_1 \vdash P, \Gamma_2 \vdash Q$, $\Delta \asymp \Gamma_1$ and $\Delta \asymp \Gamma_2$. We

show that $\mathcal{R}$ is a typed behavioral equivalence, i.e. we prove that $\mathcal{R} \subseteq \cong$. To see barb preservation, assume $\Delta \models P \downarrow_{x!}$. We have $P \equiv (\nu \tilde{y} : \tilde{T})(\overline{x}\langle \tilde{v} \rangle.P_1 \mid P_2)$ and $x \notin \tilde{y}$ and $\Delta^?(x) \downarrow$. There are two cases corresponding to $\tilde{v} \cap \tilde{y} = \emptyset$ or not. We draw the latter case, the former is easier. By multiple applications of [L-Res], [L-Open],[L-Par] and [L-Out], where in the last we exploit the hypothesis $\Delta^?(x) \downarrow$, we infer $\Delta \triangleleft P \xrightarrow{\alpha} \Delta' \triangleleft P'$ with $\alpha = (\tilde{z})\,\overline{x}\langle \tilde{v} \rangle$ with $\tilde{v} \cap \tilde{y} = \tilde{z}$. From $\Delta \models P \approx Q$ we infer that there is Q' such that $\Delta \triangleleft Q \stackrel{\alpha}{\Longrightarrow} \Delta' \triangleleft Q'$ with $\Delta' \models P' \approx Q'$. This means that there is Q_1 such that $\Delta \triangleleft Q \Longrightarrow \Delta \triangleleft Q_1 \xrightarrow{\alpha} \Longrightarrow \Delta' \triangleleft Q'$. We apply Lemma 4.4 and infer that there are variables $\tilde{a}$ and types $\tilde{A}$ such that $Q_1 \equiv (\nu \tilde{a} : \tilde{A})(\overline{x}\langle \tilde{v} \rangle.R_1 \mid R_2)$ where $x \notin \tilde{a}$ and $\tilde{z} \subseteq \tilde{a}$. From $\Delta^?(x) \downarrow$ and the results above we infer $\Delta \models Q \Downarrow_{x!}$. Now assume $\Delta \models P \downarrow_{x?}$. We have $P \equiv (\nu \tilde{x} : \tilde{T})(x(\tilde{y}).P_1 \mid P_2)$ and $x \notin \tilde{x}$ and $\Delta^!(x) \downarrow$. By multiple applications of [L-Res], [L-Weak],[L-Par] and [L-In], where in the last we exploit the hypothesis $\Delta^!(x) \downarrow$, we infer $\Delta \triangleleft P \xrightarrow{\alpha} \Delta' \triangleleft P'$ with $\alpha = (\tilde{y} : \tilde{T})\,x(\tilde{v})$ for types $\tilde{T}$. From $\Delta \models P \approx Q$ we infer that there is Q' such that $\Delta \triangleleft Q \stackrel{\alpha}{\Longrightarrow} \Delta' \triangleleft Q'$ with $\Delta' \models P' \approx Q'$. As in the previous case, by exploting Lemma 4.4 we find a suitable process Q_1 such that $\Delta \triangleleft Q \Longrightarrow \Delta \triangleleft Q_1$ and $Q_1 \equiv (\nu \tilde{z} : \tilde{T}')(x(\tilde{y}).R_1 \mid R_2)$ with $x \notin \tilde{z}$; from this we infer $\Delta \models Q \Downarrow_{x?}$, as required. Next we show reduction closure. Assume $P \xrightarrow{\mu} P'$. We apply the left direction of Lemma 4.5 and infer $\Delta \triangleleft P \xrightarrow{\mu} \Delta \triangleleft P'$. From $\Delta \models P \approx Q$ we infer that $\Delta \triangleleft Q \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_n} \Delta \triangleleft Q'$ with $\Delta \models P' \approx Q'$. By applications of the right direction of Lemma 4.5 we infer that $Q \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_n} Q'$. Since $\Delta \models P' \,\mathcal{R}\, Q'$, we are done. Lastly, contextuality of $\mathcal{R}$ is assured by Propositions 4.7, 4.9 and 4.10. □

5. Completeness

In this section we show that typed behavioral equivalence coincides with bisimulation. As in (Hennessy 2007; Bugliesi and Giunti 2005) the completeness result relies on the fact that the observations of processes we make in typed actions in Figure 4 are contextually valid. The idea is to represent each label with a context, in the following way. Whenever $\Delta = \tilde{x} : \tilde{T}$, we let $\overline{p}\langle \Delta \rangle$ be a short of $\overline{p}\langle \tilde{x} \rangle.\mathbf{0}$ and (Δ) be a short of $\tilde{T}$. Given a type environment Δ, we let Δ after α be defined and equal to Δ', if:

$$\begin{array}{ll}
\Delta \triangleleft x(\tilde{y}).\mathbf{0} \xrightarrow{\alpha} \Delta' \triangleleft \mathbf{0} & \alpha = (\tilde{z} \colon \tilde{B})\,x(\tilde{v}), \quad |\tilde{y}| = |\tilde{v}| \\
\Delta \triangleleft \overline{x}\langle \tilde{v} \rangle.\mathbf{0} \xrightarrow{\alpha} \Delta' \triangleleft \mathbf{0} & \alpha = \overline{x}\langle \tilde{v} \rangle \\
\Delta \triangleleft (\nu \tilde{y} \colon \tilde{B})\overline{x}\langle \tilde{v} \rangle.\mathbf{0} \xrightarrow{\alpha} \Delta' \triangleleft \mathbf{0} & \alpha = (\tilde{y})\overline{x}\langle \tilde{v} \rangle \\
\Delta' = \Delta & \alpha = \mu
\end{array}$$

Lemma 5.1. For each label $\alpha \neq \mu$ and environment Δ there exists a process C_α^Δ and a name e fresh to Δ such that:

1 if $\Delta \triangleleft P \xrightarrow{\alpha} \Delta' \triangleleft P'$ then $\Delta, e : (?(\Delta'), !(\Delta')) \vdash C_\alpha^\Delta$ and $P \mid C_\alpha^\Delta \Longrightarrow (\nu \tilde{x} : \tilde{T})(P' \mid \overline{e}\langle \Delta' \rangle)$;
2 if $C_\alpha^\Delta \mid P \Longrightarrow (\nu \tilde{x} : \tilde{T})(P' \mid \overline{e}\langle \tilde{v} \rangle)$ and $\Delta, e : (?(\Delta'), !(\Delta')) \vdash C_\alpha^\Delta$ with $\Delta' = \Delta$ after α and $\tilde{x} = \mathrm{bv}(\alpha)$ and $e \notin \tilde{x}$, then $\Delta \triangleleft P \stackrel{\alpha}{\Longrightarrow} \Delta' \triangleleft P'$.

Proof. For the sake of compactness, we define the contexts for the monadic calculus. The definition of contexts for the polyadic calculus is a simple but annoying exercise (cf. (Giunti 2007, Chapter 2)). In the following, the process $\mathsf{if}\, y \notin \Delta\, \mathsf{then}\, P\, \mathsf{else}\, Q$ is implemented by means of nested conditionals; we omit all the details.

$$C^{\Delta}_{\overline{x}\langle v\rangle} = x(y\colon A).\mathsf{if}\, y = v\, \mathsf{then}\, \overline{e}\langle\Delta_1\rangle\, \mathsf{else}\, \mathbf{0} \qquad A \stackrel{\text{def}}{=} \Delta^{?}(x),\ \Delta_1 \stackrel{\text{def}}{=} \Delta \text{ after } \overline{x}\langle v\rangle$$

$$C^{\Delta}_{(v)\overline{x}\langle v\rangle} = x(y\colon A).\mathsf{if}\, y \notin \Delta\, \mathsf{then}\, \overline{e}\langle\Delta_1\rangle\, \mathsf{else}\, \mathbf{0} \qquad A \stackrel{\text{def}}{=} \Delta^{!}(x),\ \Delta_1 \stackrel{\text{def}}{=} \Delta \text{ after } (v)\overline{x}\langle v\rangle$$

$$C^{\Delta}_{x(v)} = \overline{x}\langle v\rangle.\overline{e}\langle\Delta_1\rangle \qquad \Delta_1 \stackrel{\text{def}}{=} \Delta \text{ after } x(v)$$

$$C^{\Delta}_{(v:T)x(v)} = (\nu v : T)\overline{x}\langle v\rangle.\overline{e}\langle\Delta_1\rangle \qquad \Delta_1 \stackrel{\text{def}}{=} \Delta \text{ after } (v : T)x(v)$$

To show (1), assume $\Delta \triangleleft P \xrightarrow{\alpha} \Delta' \triangleleft P'$. First notice that given e fresh to Δ we have $e \notin \mathrm{fv}(P)$. Consider the case $\alpha = \overline{x}\langle v\rangle$. From Lemma 4.4 we deduce that $\Delta^{?}(x) \downarrow$ and in turn infer the shape of P: $P \equiv (\nu\tilde{x} : \tilde{T})(\overline{x}\langle v\rangle.P_1 \mid P_2)$ with $\{x, v\} \cap \tilde{x} = \emptyset$. First we infer $\Delta, e : (?(\Delta'), !(\Delta')) \vdash C^{\Delta}_{\alpha}$. Next, by applications of [R-Com] we have $P \mid C^{\Delta}_{\overline{x}\langle v\rangle} \Longrightarrow (\nu\tilde{x} : \tilde{T})(P_1 \mid P_2 \mid \overline{e}\langle\Delta'\rangle)$. Now consider the case $\alpha = (v\colon T)x(v)$. From $\Delta^{w}(x) \downarrow$ and $\Delta' = \Delta_1$ we infer $\Delta, e : (?(\Delta'), !(\Delta')) \vdash C^{\Delta}_{\alpha}$. We infer the shape of P: $P \equiv (\nu\tilde{x} : \tilde{T})(x(y).P_1 \mid P_2)$ with $x \notin \tilde{x}$. By applications of [R-Com] we infer the desired result, $P \mid C^{\Delta}_{(v:T)x(v)} \Longrightarrow (\nu v : T, \tilde{x} : \tilde{T})(P_1[v/x] \mid P_2 \mid \overline{e}\langle\Delta'\rangle)$.

Consider case (2) and assume $C^{\Delta}_{\alpha} \mid P \Longrightarrow (\nu\tilde{x} : \tilde{T})(P' \mid \overline{e}\langle\tilde{v}\rangle)$. Let $\alpha = \overline{x}\langle v\rangle$. Therefore to unblock the signal $\overline{e}\langle\tilde{v}\rangle$ it must be that P has interacted with C^{Δ}_{α}, because $e \notin \mathrm{fv}(P)$. Moreover the output on e is unblocked only if v is sent over y. We infer that $P \Longrightarrow P_1 \equiv (\nu\tilde{x} : \tilde{T})(\overline{x}\langle v\rangle.P_1 \mid P_2)$ with $\{x, v\} \cap \tilde{x} = \emptyset$. Moreover, we infer that $(\nu\tilde{x} : \tilde{T})(P_1 \mid P_2 \Longrightarrow P')$, since after the interaction over x with C^{Δ}_{α} no other synchronization is possible. Since $\Delta, e : (?(\Delta'), !(\Delta')) \vdash C^{\Delta}_{\alpha}$, with appropriate type Δ', we deduce that $\Delta^{?}(x) \downarrow$. By applications of [L-Res],[L-Par] and [L-Out], followed by [L-Red] we infer that $\Delta \triangleleft P' \stackrel{\alpha}{\Longrightarrow} \Delta' \triangleleft P'$. □

Based on the lemma above, we prove that bisimulation is complete with respect to typed behavioral equivalence.

Proposition 5.2 (Completeness). If $\Delta \models P \cong Q$ then $\Delta \models P \approx Q$.

Proof. We follow the construction of (Hennessy 2007). Let $\Delta \models P\,\mathcal{R}\,Q$ whenever $\Delta \models P \cong Q$. Assume $\Delta \triangleleft P \xrightarrow{\alpha} \Delta' \triangleleft P'$. If $\alpha = \mu$ we have $\Delta' = \Delta$ and we find Q' such that $\Delta \triangleleft P \Longrightarrow \Delta \triangleleft Q'$ with $\Delta \models P' \cong Q'$, as requested. Let $\alpha = (\tilde{y})\overline{x}\langle\tilde{v}\rangle$. Therefore $\Delta^{?}(x) \downarrow$. We take e fresh to Δ and infer $\Delta, e : (?(\Delta'), !(\Delta')) \vdash C^{\Delta}_{\alpha}$. We apply Lemma 5.1 and infer $P \mid C^{\Delta}_{\alpha} \Longrightarrow (\nu\tilde{x} : \tilde{T})(P' \mid \overline{e}\langle\Delta'\rangle)$. Take f fresh to Δ, $e \neq f$, and let $C_f = \overline{g}\langle\rangle \mid e(\tilde{x}).g().\overline{f}\langle\tilde{x}\rangle$ where $|\tilde{x}| = |\,\mathrm{dom}(\Delta)|$. Defining $\Omega = \Delta, e : (?(\Delta'), !(\Delta')), f : (?(\Delta'), !(\Delta'))$ we infer that (i) $\Omega \vdash C_f$ and (ii) $\Omega \models C_f \mid C^{\Delta}_{\alpha} \mid P \cong C_f \mid C^{\Delta}_{\alpha} \mid Q$, by contextuality of $\cong$. We infer synchronization of C_f with the redex of $P \mid C^{\Delta}_{\alpha}$:

$$P \mid C^{\Delta}_{\alpha} \mid C_f \Longrightarrow (\nu\tilde{x} : \tilde{T})(P' \mid \overline{e}\langle\Delta'\rangle) \mid C_f \Longrightarrow (\nu\tilde{x} : \tilde{T})(P' \mid \overline{f}\langle\Delta'\rangle)$$

Let C^P be the reached redex. The process on the right matches these moves:

$$Q \mid C^{\Delta}_{\alpha} \mid C^{\Delta}_{f} \Longrightarrow C^{Q}$$

with $\Omega \models C^P \cong C^Q$. Since $\Omega \models C^P \downarrow_{f!}$ and $\Omega \not\models C^P \downarrow_{g!}$, by barb preservation $\Omega \models C^Q \Downarrow_{f!}$ and $\Omega \not\models C^Q \Downarrow_{g!}$. We find Q' such that $C^Q \equiv (\nu \tilde{\tilde{x}} : \tilde{\tilde{T}})(Q' \mid \overline{f}\langle \tilde{v} \rangle)$. Therefore the signal g in C_f has been consumed by unblocking e, which means that $\tilde{v} = (\Delta')$. We infer the following reductions:

$$C^{\Delta}_{\alpha} \mid Q \Longrightarrow (\nu \tilde{x} : \tilde{T})(Q' \mid \overline{e}\langle \Delta' \rangle.)$$

We conclude that $C^Q \equiv (\nu \tilde{x} : \tilde{T})(Q' \mid \overline{f}\langle \Delta' \rangle)$. We apply Lemma 5.1 and infer that $\Delta \triangleleft Q \Longrightarrow \Delta' \triangleleft Q'$. To conclude we need the following result: $\Delta, e : (?(\Delta'), !(\Delta')) \models (\nu \tilde{x} : \tilde{T})(P \mid \overline{e}\langle \Delta' \rangle) \cong (\nu \tilde{x} : \tilde{T})(Q \mid \overline{e}\langle \Delta' \rangle)$ with e fresh to P, Q, for any environment Δ, implies $\Delta' \models P \cong Q$. The proof of this lemma is similar to the one in (Giunti 2007, Chapter 2). We conclude that $\Delta \models P' \mathcal{R} Q'$. The case $\alpha = (\tilde{y}\colon \tilde{T})x(\tilde{v})$ is similar, but easier. □

Propositions 4.11 and 5.2 let us prove that typed behavioral equivalence coincides with typed bisimulation.

Theorem 5.3 (Full abstraction). $\Delta \models P \cong Q$ if and only if $\Delta \models P \approx Q$.

6. Applications

In this section we apply our theory and we show a couple of interesting equalities.

As a first example, we show that the process P' introduced in Section 1 is *deadlocked*. We show this by proving that $a(s\colon T).P'$ is bisimilar to $a(s\colon T).\mathbf{0}$. That P' is deadlocked then follows as a by-product. We have the following equation:

$$\Delta, a\colon !T.\mathsf{end} \models a(s\colon T).P' \cong a(s\colon T).\mathbf{0} \tag{3}$$

where we remind that $T \stackrel{\text{def}}{=} (?U.!I.\mathsf{end}, !U.?I.\mathsf{end})$ and $P' \stackrel{\text{def}}{=} (\overline{s}\langle u \rangle.s(x\colon I).\mathsf{if}\, x = r\, \mathsf{then}\, Q \mid P_2$. To prove equation (3), take the relation defined by $\Delta_1 \models M \mathcal{R} N$ whenever (a) $M = a(s\colon T).P'$ and $N = a(s\colon T).\mathbf{0}$ and $\Delta_1 = \Delta, a\colon !T.\mathsf{end}$ and $\Gamma, a\colon ?T.\mathsf{end} \vdash a(s\colon T).P'$ and $\Gamma', a\colon ?T.\mathsf{end} \vdash a(s\colon T).\mathbf{0}$ and $\Delta \asymp \Gamma$,$\Delta \asymp \Gamma'$ or (b) $M = P'[t/s]$ and $N = \mathbf{0}$ and $\Delta_1 = \Delta, a\colon \mathsf{end}$ and $\Gamma, a\colon \mathsf{end}, t\colon T \vdash M$ and $\Gamma'' \vdash \mathbf{0}$ and $\Delta \asymp \Gamma, t\colon T$ and $\Delta_1 \asymp \Gamma'$. It is not difficult to show that $\mathcal{R}$ is a bisimulation, and hence a proof of (3) given that $\cong = \approx$ (cf. Theorem 5.3). That $\mathcal{R}$ is a type-indexed relation is straightforward. Let $\Delta \triangleleft M \stackrel{\alpha}{\longrightarrow} \Delta' \triangleleft M'$. In case a) we infer that rule [L-In] or rule [L-Weak] has been applied, and in turn that $\alpha = (\tilde{t}\colon T)a(t)$ for some t; therefore M' has the shape of the left process of b). We match this move with $\Delta \triangleleft N \stackrel{\alpha}{\longrightarrow} \Delta' \triangleleft \mathbf{0}$. To use b) and conclude that $\Delta' \models M' \mathcal{R} \mathbf{0}$ we need to check the desired properties for Δ' and for the type environment for M' and $\mathbf{0}$. Let $\tilde{t} = t$ and $\tilde{T} = T'$; the case $\tilde{T}$ empty is easier. We note that the environment Δ' is equal to Δ after $(t\colon T')a(t)$, which is the environment $\Delta, t\colon \top, a\colon \mathsf{end}$. This follows from $T' = T$ and $(T \otimes T'') \downarrow$ iff $T'' = \top$. Next, we take $\Gamma, a\colon ?T.\mathsf{end} \vdash a(s\colon T).P'$ and weaken it to $\Gamma, a\colon ?T.\mathsf{end}, t : \top \vdash a(s\colon T).P'$. By the rule for typing an input, [T-In], and substitution (cf. Lemma 2.7), we infer $\Gamma, a\colon \mathsf{end}, t : T \vdash P'[t/s]$. The compatibility result follows then

by the hypothesis. The case for the environment typing process **0** is straightforward. This complete the proof since in case b) process M is stuck because of $\Delta(t) = \top$. This follows from $\Delta \asymp \Gamma, t\colon T$. The opposite direction is analogous.

The second equation establishes the *secrecy* of an exchange of a confidential information p which occurs on a session b that is not accessible by the context. This example shows that the typed discipline based on sessions can also be used to enforce discretionary access policies to communication channels, and that we can establishes our security equations by reasoning on the typed knowledge of the context of such channels.

$$\begin{aligned} P &\stackrel{\text{def}}{=} (\nu p\colon C)(\bar{b}\langle p\rangle) \mid b(x).Q \quad p \neq b \\ \Delta, b : \top &\models P \cong (\nu p\colon C)Q[p/x] \end{aligned} \tag{4}$$

To prove equation (4), let $\Delta \models M \,\mathcal{R}\, N$ whenever a) $M = P$ and $N = (\nu p\colon C)Q[p/x]$ and $\Gamma \vdash M$ and $\Gamma' \vdash N$ and $\Delta = \Delta_1, B\colon \top$ and $\Delta \asymp \Gamma$ and $\Delta \asymp \Gamma'$ or b) $M \equiv N$ and $\Gamma \vdash M$ and $\Gamma' \vdash N$ and $\Delta \asymp \Gamma$ and $\Delta \asymp \Gamma'$. We show that $\mathcal{R}$ is a bisimulation up to structural equivalence, which in turn implies that $\mathcal{R}$ is a bisimulation; the proof of this claim follows standard arguments. Let $\Delta \triangleleft M \stackrel{\alpha}{\longrightarrow} \Delta' \triangleleft M'$. In case a) we do not match this move since we claim that $M' \equiv N$. Indeed, we can prove that the only applicable rule is [L-Red], since the channel b is known by the environment at top type. Therefore $\alpha = b$ and $\Delta' = \Delta$, and by b) we infer infer $\Delta' \models M' \,\mathcal{R}\, N$. Case (b) follows immediately from the fact that structural congruent processes have the same moves. For the opposite direction, assume $\Delta \triangleleft N \stackrel{\alpha}{\longrightarrow} \Delta' \triangleleft N'$. We match this move with a) $\Delta \triangleleft M \stackrel{\tau}{\longrightarrow} \stackrel{\alpha}{\longrightarrow} \equiv N'$ or b) $\Delta \triangleleft M \stackrel{\alpha}{\longrightarrow} \equiv N'$.

7. Conclusions

We have proposed a notion of typed observational equivalence for a pi calculus with session types where the discerning capability of the observer is regulated by the type system. Type checking forces contexts to not interfere with a session shared by two participants. To avoid universal quantification over contexts, we provided for a proof technique based on typed bisimulation semantics. To motivate our approach, we applied the typed behavioral reasoning to three examples and 1) proved a deadlock due to a programming error, 2) established the secrecy of the exchange of a confidential information, and 3) showing the ability of contexts to compare sessions by performing identity tests. We are also applying our technique to the type checking algorithm for session types of (Giunti 2011) in order to prove that all processes with a correct behavior are type checked.

Session types, introduced in (Takeuchi, Honda and Kubo 1994; Honda, Vasconcelos and Kubo 1998) for a dialect of pi calculus, are now a well established static analysis technique for service oriented protocols which is utilized in various paradigms including functional languages, object-oriented programming and operating systems; see (Dezani-Ciancaglini and de'Liguoro 2009) for a recent overview. Our session types are based on a channel type construct of the form (S_1, S_2) where each S_i describes the behavior of one end of the session (Giunti and Vasconcelos 2010). Slightly differently to (Giunti and Vasconcelos 2010), the type system analyzed in this paper follows an affine discipline

and considers both polyadic communications and the ability to compare the identity of names. We do not envisage difficulties in establishing the results of this paper in a linear setting; we chose a more liberal typing discipline for the sake of a clean formulation of the behavioral theory and to type more processes.

The presence of the type checker in our behavioural theory is necessary. Indeed, the programming constructs of the pi calculus do not guarantee adequate protection for session channel abstractions. While on hand the restriction operator can be used to create channels which are free from the interference of the context, on the other hand pi calculus processes can receive channels from the context, which in service-oriented terminology consists in receiving sessions that have been *delegated*. In this situation, an untyped context can easily cheat and break the desired equivalences by avoiding to follow a correct discipline for the session it has delegated, as showed in Section 1.

Notions of untyped bisimulation have already been proposed for service oriented calculi; see (Lanese, Ravara and Vieira 2011) for a comparison of behavioural theories for session-based and correlation-based calculi for service-oriented computing. Independently from us, typed bisimulation for asynchronous session types has been recently considered in (Kouzapas, Yoshida and Honda 2011) for a dialect of pi calculus with primitives for session initiation and buffers. Bisimulation is defined over a labelled transition system that omits session delegation and relies on a notion of process localisation. On contrast, we tackle the full communication mechanism of pi calculus and provide for delegating a session already known to the receiver, which is a non trivial feature in service oriented computing (Yoshida and Vasconcelos 2007).

In the linear setting, the work close to ours is (Kobayashi, Pierce and Turner 1999), which introduces a notion of barbed bisimulation for pi calculus in the presence of linear types. While in (Kobayashi, Pierce and Turner 1999) linear types are distinct from unrestricted types, in our approach affine types evolve to unrestricted types; indeed the linear types of (Kobayashi, Pierce and Turner 1999) can be encoded in the linear session types of (Giunti and Vasconcelos 2010) by preserving typing, as shown by the author and Vasconcelos. An untyped testing theory for a linear pi calculus with choices and subtyping has been recently used in (Demangeon and Honda 2011) to prove full abstraction for an encoding of (Honda, Vasconcelos and Kubo 1998).

Our results and techniques draw on typed equivalences for pi calculus in the presence of I\O types and subtyping (Hennessy 2007; Bugliesi and Giunti 2005). We do not envisage difficulties in introducing subtyping for unrestricted types à la (Pierce and Sangiorgi 1996); however, this seems to go in the opposite direction of the idea of channel types. We therefore need to investigate subtyping solutions which take into account the channel type construct. This would permit us to type more processes, as $Q \stackrel{\text{def}}{=} a(\mathsf{item}).a(\mathsf{min\_bid}).(!a(\mathsf{bid}).P \mid \overline{a}\langle 20\$\rangle)$. The process abstracts one end of a session established by an auction service in order to receive the name of an item and the minimum bid. After that this mandatory information is received, the process waits for an unbounded number of bids, and also proposes itself a bid. Unfortunately, we cannot describe Q with an end point type of the form S since in the (termination) continuation type both the input and the output capabilities would be needed.

Lastly, we believe that the presence of matching in a pi calculus with session types is of practical interest, since it permits to reason on the identity of sessions after that they have been delegated. This point has been already discussed in (Giunti *et al.* 2009), which introduces a session typing system for a variant of pi calculus with accept/request primitives (Honda, Vasconcelos and Kubo 1998) and matching, and illustrates a system with long running sessions of different degrees of trust relying on a database of malicious sessions.

7.1. *Acknowledgements*

This work was carried out during the tenure of an "Alain Bensoussan" Fellowship Programme. This programme is supported by the Marie Curie Co-Funding of Regional, National and International Programmes (COFUND) of the European Commission. I would like to thank Silvia Crafa, which gave me precious advices that permitted to improve the presentation of the paper. I also thank the anonymous referees for constructive criticism on a previous draft of the paper.

References

Michele Bugliesi and Marco Giunti. Typed processes in untyped contexts. In *TGC*, volume 3705 of *Lecture Notes in Computer Science*, pages 19–32. Springer, 2005.

Mariangiola Dezani-Ciancaglini and Ugo de'Liguoro. Sessions and session types: An overview. In *WS-FM*, volume 6194 of *Lectures Notes in Computer Science*, pages 1–28. Springer, 2009.

Romain Demangeon and Kohei Honda. Full abstraction in a subtyped pi-calculus with linear types. In *CONCUR*, volume 6901 of *Lecture Notes in Computer Science*, pages 280–296. Springer, 2011.

Marco Giunti. *Secure Implementations of Typed Channel Abstractions.* PhD thesis, Ca'Foscari University, Venice, 2007.

Marco Giunti. A type checking algorithm for qualified session types. In *WWV*, volume 61 of *Electronic Proceedings in Theoretical Computer Science*, pages 96-114. 2011.

Marco Giunti and Vasco Thudichum Vasconcelos. A linear account of session types in the pi calculus. In *CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 432–446. Springer, 2010.

Marco Giunti, Vasco T. Vasconcelos, Kohei Honda, and Nobuko Yoshida. Session-based type discipline for pi calculus with matching. In *PLACES*, 2009.

Matthew Hennessy. *A Distributed Pi-calculus.* Cambridge University Press, 2007.

Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP*, volume 1381 of *Lectures Notes in Computer Science*, pages 22–138. Springer, 1998.

Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the pi-calculus. *ACM Transactions on Programming Languages and Systems*, 21(5):914–947, 1999.

Dimitrios Kouzapas, Nobuko Yoshida, and Kohei Honda. On asynchronous session semantics. In *FMOODS/FORTE*, volume 6722 of *Lecture Notes in Computer Science*, pages 228–243, 2011.

Ivan Lanese, António Ravara, and Hugo Torres Vieira. Behavioral theory for session-oriented calculi. In *Results of the SENSORIA Project*, pages 189–213. 2011.

B. C. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–453, 1996.

Davide Sangiorgi and David Walker. *The Pi-calculus, a Theory of Mobile Processes*. Cambridge University Press, 2001.

Kaku Takeuchi, Kohei Honda, and Makoto Kubo. An Interaction-based Language and its Typing System. In *PARLE*, volume 817 of *Lectures Notes in Computer Science*, pages 398–413. Springer-Verlag, 1994.

Nobuko Yoshida and Vasco T. Vasconcelos. Language primitives and type discipline for structured communication-based programming revisited: Two systems for higher-order session communication. In *SecReT'07*, volume 171(4) of *ENTCS*, pages 73–93. Elsevier Science Publishers, 2007.

Appendix A. Additional proofs

The first two proofs show respectively closure under substitution and subject reduction for the type system presented in Section 2. The proof follows a schema similar to the corresponding results in (Giunti and Vasconcelos 2010).

Proof of Lemma 2.7. We rely on an analogous result for values, namely that (1) if $\Gamma, \tilde{x}\colon \tilde{T}, \tilde{v}\colon \tilde{T}' \vdash P$ and $(\tilde{T} \otimes \tilde{T}') \downarrow$ then $\Gamma, \tilde{v}\colon \tilde{T} \otimes \tilde{T}' \vdash v\colon T$. The interesting case is session delegation, which arises in rules [T-Out-L],[T-Out] and [T-Out]. It is sufficient to analyze case [T-OutT]; we show the monadic case, the polyadic case is analogous but the notation is heavier. We first tackle the case whether the variable to be substituted is the object of the communication. Let $\Gamma, y\colon T_1 \otimes T_2, v : T \vdash \overline{x}\langle y\rangle.P$ be inferred from $\Gamma, y\colon T_1 \otimes T_2, v : T \vdash x\colon !T_i$ and $\Gamma, y\colon T_j, v : T \vdash P$ given that $T_i \otimes T_j$ is defined for i, j with the conditions in $(*)$. We need to prove that if $A \stackrel{\text{def}}{=} T \otimes (T_1 \otimes T_2)$ is defined then $\Gamma, v : A \vdash \overline{x}\langle y\rangle.P[v/y]$. Indeed, when A is defined we have that $T \otimes T_j$ is defined, and then by applying (1) we infer $\Gamma, v\colon T \otimes T_j \vdash x[v/y]\colon !T_i$, and by induction we have $\Gamma, v\colon T \otimes T_j \vdash P[v/y]$. When $j = 1$, and in turn $i = 2$, the result follows directly by applying [T-OutT]: $\Gamma, v : (T \otimes T_1) \otimes T_2 \vdash \overline{x}\langle y\rangle.P[v/y]$. Otherwise, when $j = 2$, by applying [T-OutT] we obtain: $\Gamma, v\colon T_2 \otimes (T \otimes T_1) \vdash \overline{x}\langle y\rangle.P[v/y]$. We exchange the order of the end points by applying Lemma 2.4, and we infer $\Gamma, v\colon (T \otimes T_1) \otimes T_2 \vdash \overline{x}\langle y\rangle.P[v/y]$, as desired. Consider now the case whether the variable x to be substituted in [T-OutT] is the subject of the communication, and let $\Gamma, x\colon T_2, y\colon A, v : T_1 \vdash \overline{x}\langle y\rangle.P$ be inferred from $\Gamma, x\colon T_2, v\colon T_1 \vdash x\colon ?B$ and $\Gamma, x\colon T_2, y\colon C, v\colon T_1 \vdash P$, where $A = B \otimes C$, or $A = C \otimes B$. Assume that $T_1 \otimes T_2$ is defined. We need to prove that $\Gamma, y\colon A, v\colon T_1 \otimes T_2 \vdash \overline{x}\langle y\rangle.P[v/y]$. We apply (1) and infer $\Gamma, v\colon T_1 \otimes T_2 \vdash v\colon ?B$. By induction we obtain $\Gamma, y\colon C, v\colon T_1 \otimes T_2 \vdash P[v/y]$. We apply [T-OutT] and conclude.

Proof of Lemma 2.8. The proof is by induction on the reduction derivation, and uses substitution (Lemma 2.7). Most inductive cases are straightforward; we draw some example below. In case [R-Struct] we need a Lemma saying that typing preserves structural congruence. The most interesting case is when the derivation of the reduction step ends with rule [R-Com]. Assume $\overline{x}\langle\tilde{v}\rangle.P \mid x(\tilde{y}).Q \stackrel{x}{\longrightarrow} P \mid Q[\tilde{v}/\tilde{y}]$ and let $\Omega \vdash \overline{x}\langle v\rangle.P \mid$

$x(y).Q$. The judgment above has been inferred by using [T-Par] because

$$\Gamma \vdash \overline{x}\langle\tilde{v}\rangle.P \tag{5}$$
$$\Delta \vdash x(\tilde{y}).Q \tag{6}$$
$$\Omega = \Gamma \otimes \Delta \tag{7}$$

First notice that $\Omega(x)$ has the form $(B, \overline{B})$, because the environment Ω does permit to type both the output and the input capability of x, which in our system is possible only by types of the form (S_1, S_2), and because Ω is balanced. We have sub-cases corresponding of combination of use of rules [T-Out] and [T-Out-L] to infer (5) and of rules [T-In] and [T-In-L] to infer (6). The symmetric cases inferred from [T-In-R],[T-Out-R] can be obtained by using Lemma 2.4.

a)	([T-Out] – [T-In])	$\Gamma(x) = !\tilde{B}_i.C$	$\Delta(x) = ?\tilde{B}_i.\overline{C}$
b)	([T-OutT] – [T-InT])	$\Gamma(x) = (!\tilde{B}_i, ?\tilde{B}_i)$	$\Delta(x) = (!\tilde{B}_i, ?\tilde{B}_i)$
c)	([T-OutT] – [T-InT])	$\Gamma(x) = (!\tilde{B}_i, ?\tilde{B}_i)$	$\Delta(x) = ?\tilde{B}_i$
d)	([T-OutT] – [T-InT])	$\Gamma(x) = !\tilde{B}_i$	$\Delta(x) = (!\tilde{B}_i, ?\tilde{B}_i)$
e)	([T-OutT] – [T-InT])	$\Gamma(x) = !\tilde{B}_i$	$\Delta(x) = ?\tilde{B}_i$

(a) From (5) and [T-Out] we infer $\Gamma = \Gamma_1, \tilde{v} : \tilde{B}_1 \otimes \tilde{B}_2, x\colon !\tilde{B}_i.C$ and $\Gamma_1, \tilde{v} : \tilde{B}_j, x : C \vdash P$ with i, j as in $(*)$. From (6) and [T-In] we infer $\Delta = \Delta_1, \tilde{v} : \tilde{T}, x\colon ?\tilde{B}_i.\overline{C}$ and $\Delta_1, \tilde{v} : \tilde{T}, x\colon \overline{C}, \tilde{y} : \tilde{B}_i \vdash P$. From $(\Gamma \otimes \Delta) \downarrow$ we infer that $\tilde{B}_i \otimes \tilde{T} \downarrow$, which implies $\tilde{T} \otimes \tilde{B}_i \downarrow$ (cf. Lemma 2.3). We can apply substitution, Lemma 2.7, and infer $\Delta_1, \tilde{v} : \tilde{T} \otimes \tilde{B}_i, x\colon \overline{C} \vdash Q[\tilde{v}/\tilde{y}]$. Then we can apply [T-Par] and obtain:

$$(\Gamma_1, \tilde{v}\colon \tilde{B}_j, x : C) \otimes (\Delta_1, \tilde{v} : \tilde{T} \otimes \tilde{B}_i, x\colon \overline{C}) \vdash P \mid Q[\tilde{v}/\tilde{y}]\ .$$

(b-e) From (5) and [T-OutT] we infer $\Gamma = \Gamma_1, \tilde{v}\colon \tilde{B}_1 \otimes \tilde{B}_2$ and $\Gamma, \tilde{v}\colon \tilde{B}_j \vdash P$. From (6) and [T-In] we infer $\Delta_1, \tilde{v}\colon \tilde{T}, \tilde{y}\colon \tilde{B}_i \vdash Q$. We apply substitution (Lemma 2.7) and infer $\Delta_1, \tilde{v}\colon \tilde{T} \otimes \tilde{B}_i \vdash Q[\tilde{v}/\tilde{y}]$. An application of [T-Par] give us the expected result: $\Gamma, \tilde{v}\colon \tilde{B}_j \otimes \Delta_1, \tilde{v}\colon \tilde{T} \otimes \tilde{B}_i \vdash P \mid Q[\tilde{v}/\tilde{y}]$.

Case ([R-Par]). Assume $P_1 \mid P_2 \xrightarrow{\mu} P' \mid P_2$ and let $\Gamma \vdash P_1 \mid P_2$ with Γ balanced. Therefore rule [T-Par] has been applied with the following hypothesis: $\Gamma_1 \vdash P_1$ and $\Gamma_2 \vdash P_2$ and $\Gamma = \Gamma_1 \otimes \Gamma_2$. First, we note that Γ_1, Γ_2 are both balanced. We let the reduction be inferred from $P_1 \xrightarrow{\mu} Q_1$ and assume by induction hypothesis that $\Delta_1 \vdash Q_1$ with $\Gamma_1 \overset{x}{\rightarrowtail} \Delta_1$ whenever $\mu = x$ and $\Delta_1(x) = \Gamma_1(x)$ otherwise. In the latter case we apply [T-Par] and infer the desired result, $\Gamma_1 \otimes \Gamma_2 \vdash Q_1 \mid P_2$. In the former case we infer the type $\Delta_1(x)$. Whenever $\Gamma_1(x) = (?\tilde{B}, !\tilde{B})$ we have that $\Delta_1 = \Gamma_1$, and we proceed as above. Otherwise we have that $\Gamma_1(x) = (?\tilde{B}.C, !\tilde{B}.\overline{C})$, $\Delta_1(x) = (C, \overline{C})$, and $\Gamma_2(x) = \top$. We apply [T-Par] and infer the desired result, $\Delta_1 \otimes \Gamma_2 \vdash Q_1 \mid P_2$.

Case ([R-Res-T]). We have $(\nu x : B)P \xrightarrow{\tau} (\nu x : B')P'$ inferred from $P \xrightarrow{x} P'$ and $B \overset{x}{\rightarrowtail} B'$. We apply [T-Res] and let $\Gamma \vdash (\nu x\colon B)P$ be inferred from $\Gamma, x : B \vdash P$. By

induction hypothesis $\Gamma' \vdash P'$ with $\Gamma, x : B \overset{x}{\rightarrowtail} \Gamma'$. Therefore $\Gamma' = \Gamma, x : B'$. We apply [T-Res] and conclude: $\Gamma \vdash (\nu x : B')P'$.

The next result shows that the typed actions introduced for processes in Section 4 do indeed represent a labelled transition system.

Proof of Proposition 4.2. We rely on the following lemma which explains the behavior of the semantics in Figure 4.

Lemma A.1. Let $\Delta_1 \triangleleft P_1$ be a configuration and let $\Gamma_1 \vdash P_1$ with $\Delta_1 \asymp \Gamma_1$. Assume that $\Delta_1 \triangleleft P_1 \overset{\alpha}{\longrightarrow} \Delta_2 \triangleleft P_2$.

($\alpha = \tau$) we have that $\Delta_2 = \Delta_1$ and $\Gamma_1 \vdash P_2$;

($\alpha = x$) we have that $\Delta_2 = \Delta_1$ and $\Gamma_2 \vdash P_2$ with $\Gamma_1 \overset{x}{\rightarrowtail} \Gamma_2$;

($\alpha = (\tilde{y}\colon \tilde{T})x(\tilde{v})$) we have that $(\Delta_1, \tilde{y}\colon \tilde{T} \otimes (\Gamma_1, \tilde{y}\colon \tilde{\top}) \overset{x}{\rightarrowtail} \Delta_2 \otimes \Gamma_2$ with $\Gamma_2 \vdash P$ and $\Delta_2 \asymp \Gamma_2$;

($\alpha = (\tilde{y})\overline{x}\langle\tilde{v}\rangle$) there are $\tilde{T}$ such that $(\Delta_1, \tilde{y}\colon \tilde{\top}) \otimes (\Gamma_1, y\colon \tilde{T}) \overset{x}{\rightarrowtail} \Delta_2 \otimes \Gamma_2$ with $\Gamma_2 \vdash P$ and $\Delta_2 \asymp \Gamma_2$;

Proof. The first two items are a consequence of Lemma 2.8. To see the third result, let $\Delta_1 \triangleleft P_1 \overset{(\tilde{y}\colon \tilde{T})x(\tilde{v})}{\longrightarrow} \Delta_2 \triangleleft P_2$. We proceed by induction on the last rule of the inference. In case [L-In] we have that $\Gamma_1 \vdash x(\tilde{z} : \tilde{B}_i).P$ $(**)$ with the conditions $(*)$ over i has been inferred by using [T-In] since $\Gamma_1 = \Gamma', x :?\tilde{B}_i.C$. Notice indeed that both [T-In-L] and [T-In-R] do not apply since $\Delta_1^!(x)$ is defined and $\Delta_1 \asymp \Gamma_1$; similarly, [T-InT] does not apply since $\Delta_1 \otimes \Gamma_1$ is balanced. Therefore the judgement $(**)$ has been inferred by $\Gamma', x : C, \tilde{z}\colon \tilde{B}_i \vdash P$. We rewrite Γ' as $\Gamma'', \tilde{v}\colon \tilde{T}$. Next, note that $\Delta_1 = \Delta', \tilde{v}\colon \tilde{B}_1 \otimes \tilde{B}_2, x :!\tilde{B}_i.\overline{C}$. We have $\Delta_1 \otimes \Gamma_1 \overset{x}{\rightarrowtail} \Delta', \tilde{v}\colon \tilde{B}_j, x\colon \overline{C} \otimes \Gamma'', \tilde{v}\colon \tilde{B}_i \otimes \tilde{T}, x\colon C$. We close the proof by applying substitution (Lemma 2.7), which permits to infer $\Gamma'', \tilde{v}\colon \tilde{B}_i \otimes \tilde{T}, x : C \vdash P[\tilde{v}/\tilde{z}]$. Case [L-InT] is analogous, but easier. In case [L-Weak] assume $\Delta_1 \triangleleft P_1 \overset{(y\colon T, \tilde{y}\colon \tilde{T})x(\tilde{v})}{\longrightarrow} \Delta_2 \triangleleft P_2$ and let the I.H. be $\Delta_1, y : T \triangleleft P \overset{(\tilde{y}:\tilde{T})x(\tilde{v})}{\longrightarrow} \Delta_2 \triangleleft P_2$ with $(\Delta_1, y : T, \tilde{y}\colon \tilde{T} \otimes (\Gamma_1, y\colon T', \tilde{y}\colon \tilde{\top}) \overset{x}{\rightarrowtail} \Delta_2 \otimes \Gamma_2$ with $\Gamma_2 \vdash P$ and $\Delta_2 \asymp \Gamma_2$. Note that $y \notin \mathrm{fv}(P)$, because y is bound in α. From this fact we infer that $(\Gamma_1, y\colon \top, \tilde{y}\colon \tilde{\top}) \vdash P_1$ and $(\Gamma_2 \backslash y), y\colon T \vdash P_2$, where the notation $\Gamma\backslash z$ stands for Γ less the entry for z. From $(\Delta_1, y : T, \tilde{y}\colon \tilde{T} \otimes (\Gamma_1, y\colon \top, \tilde{y}\colon \tilde{\top}) \overset{x}{\rightarrowtail} \Delta_2 \otimes (\Gamma_2 \backslash y), y\colon T$ we obtain the desired result. The remaining cases follow from induction. To show the fourth item, assume that $\Delta_1 \triangleleft P_1 \overset{(\tilde{y})\overline{x}\langle\tilde{v}\rangle}{\longrightarrow} \Delta_2 \triangleleft P_2$. Case [L-Out] is specular to [L-In]. We have that [T-Out] has been used in order to infer $\Gamma_1 \vdash \overline{x}\langle\tilde{v}\rangle.P_2$, since rules [T-Out-L] and [T-Out-R] do not apply because of $\Delta_1^!(x)$ defined, and rule [T-OutT] does not apply because of $\Delta_1 \otimes \Gamma_1$ is balanced. We infer the shape of Δ_1 and Γ_1, and proceed similarly to case [L-In] and obtain the desired result. Case [L-OutT] is similar, but easier. Take now case [L-Open] and let $\Delta_1 \triangleleft (\nu y : T)P_1 \overset{(y,\tilde{y})\,\overline{x}\langle\tilde{v}\rangle}{\longrightarrow} \Delta_2 \triangleleft P_2$ be inferred from $\Delta_1, y : \top \triangleleft P_1 \overset{(\tilde{y})\overline{x}\langle\tilde{v}\rangle}{\longrightarrow} \Delta_2 \triangleleft P_2$ where $\Gamma_1 \vdash (\nu y : T)P_1$ and $\Delta_1 \asymp \Gamma_1$. From [T-Res] we infer $\Gamma_1, y : T \vdash P_1$. By I.H. there are $\tilde{T}$ such that $\Delta_1, y : \top, \tilde{y} : \tilde{\top} \otimes \Gamma_1, y : T, \tilde{y} : \tilde{T} \overset{x}{\rightarrowtail} \Delta_2 \otimes \Gamma_2$ where $\Gamma_2 \vdash P_2$ and $\Delta_2 \asymp \Gamma_2$. This is the desired result. □

Based on the lemma above, we can easily prove that typed actions preserve configurations, and in turn that Figure 4 is a labelled transition system.

The next result shows the type $\top$ is less informative than the other types.

Proof of Lemma 4.6 We proceed by co-induction and show that $\Delta, x\colon \top \otimes T_2 \models P \,\mathcal{R}\, Q$ whenever $\Delta, x\colon T_1 \otimes T_2 \models P \approx Q$ is a bisimulation. The result follows by letting $T_2 = \top$. That $\mathcal{R}$ is a typed-index relation follows from $T_1 \otimes T_2 \otimes T_3$ balanced implies $T_2 \otimes T$ balanced. To see that $\mathcal{R} \subseteq \approx$, let $\Delta, x\colon \top \otimes T_2 \triangleleft P \xrightarrow{\alpha} \Delta' \triangleleft P'$. If $x \notin \mathrm{fv}(\alpha)$ we easily infer $\Delta' = \Delta_1, x\colon \top \otimes T_2$ and $\Delta, x\colon T_1 \otimes T_2 \triangleleft P \xrightarrow{\alpha} \Delta_1, x\colon T_1 \otimes T_2 \triangleleft P'$. We find Q' such that $\Delta, x\colon T_1 \otimes T_2 \triangleleft Q \stackrel{\alpha}{\Longrightarrow} \Delta_1, x\colon T_1 \otimes T_2 \triangleleft Q'$ with $\Delta_1, x\colon T_1 \otimes T_2 \models P' \approx Q'$. From $x \notin \mathrm{fv}(\alpha)$ we infer $\Delta, x\colon \top \otimes T_2 \triangleleft Q \stackrel{\alpha}{\Longrightarrow} \Delta_1, x\colon \top \otimes T_2 \triangleleft Q'$. We infer that $\Delta_1, x\colon \top \otimes T_2 \models P' \,\mathcal{R}\, Q'$, as required. Now assume that $\alpha = (\tilde{y}\colon \tilde{T})x(\tilde{v})$ or $\alpha = (\tilde{y})\overline{x}\langle\tilde{v}\rangle$ with $x \notin \tilde{v}$. Therefore the reduction has been allowed by the type T_2, since type top forbids any interaction over x. We infer $\Delta' = \Delta_1, x\colon \top \otimes T_2'$ where $T_2' = T_2$ when T_2 is a termination type and $T_2' = C$ when $T_2 = ?\tilde{B}.C$ or $T_2 = !\tilde{B}.C$. Notice that other cases are not possible, because of the compatibility hypothesis $\Delta, x\colon \top \otimes T_2 \asymp \Gamma_1$ for some Γ_1 such that $\Gamma_1 \vdash P$. We infer that $\Delta, x\colon T_1 \otimes T_2 \triangleleft P \xrightarrow{\alpha} \Delta_1, x\colon T_1 \otimes T_2' \triangleleft P'$. We then close this case by exploiting the properties of $\approx$, following the same steps as above. Now consider the case $\alpha = (\tilde{y})\overline{z}\langle x, \tilde{v}\rangle$ with $x \notin \tilde{y}$. Therefore there is T_3 such that $\Delta' = \Delta_1, x\colon (\top \otimes T_2) \otimes T_3$ and $\Delta, x\colon T_1 \otimes T_2 \triangleleft P \xrightarrow{\alpha} \Delta'' \triangleleft P'$ with $\Delta'' = \Delta_1, x\colon (T_1 \otimes T_2) \otimes T_3$. We find Q' such that $\Delta, x\colon T_1 \otimes T_2 \triangleleft Q \stackrel{\alpha}{\Longrightarrow} \Delta'' \triangleleft Q'$ and $\Delta'' \models P' \approx Q'$. From the reductions above we easily infer $\Delta, x\colon \top \otimes T_2 \triangleleft Q \stackrel{\alpha}{\Longrightarrow} \Delta_1, x\colon (\top \otimes T_2) \otimes T_3 \triangleleft Q'$. From associativity of $\otimes$ we obtain $\Delta' \models P' \,\mathcal{R}\, Q'$, and we are done.