Disciplined structured communications with disciplined runtime adaptation

Di Giusto, Cinzia; Perez, Jorge A.

Published in:
Science of computer programming

DOI:
10.1016/j.scico.2014.04.017

Publication date:
2015

Science of Computer Programming 97 (2015) 235–265

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Disciplined structured communications with disciplined runtime adaptation

Cinzia Di Giusto [a,*], Jorge A. Pérez [b,c,*]

[a] Université d'Evry – Val d'Essonne, Laboratoire IBISC, France
[b] CITI and Departamento de Informática, FCT Universidade Nova de Lisboa, Portugal
[c] Johann Bernoulli Institute for Mathematics and Computer Science, University of Groningen, The Netherlands

HIGHLIGHTS

- A model of session communications with located and update processes.
- A type discipline that ensures absence of communication errors and consistent updates.
- Several examples of runtime adaptation in the typed process framework.

ARTICLE INFO

Article history:
Received 1 July 2013
Received in revised form 1 March 2014
Accepted 29 April 2014
Available online 22 May 2014

Keywords:
Types for structured communications
Session types
Process calculi
Runtime adaptation
Adaptable processes

ABSTRACT

Session types offer a powerful type-theoretic foundation for the analysis of structured communications, as commonly found in service-oriented systems. They are defined upon core programming calculi which offer only limited support for expressing requirements related to *runtime adaptation*. This is unfortunate, as service-oriented systems are increasingly being deployed upon highly dynamic infrastructures in which such requirements are central concerns. In previous work, we developed a process calculi framework of *adaptable processes*, in which concurrent processes can be replaced, suspended, or discarded at runtime. In this paper, we propose a session type discipline for a calculus with adaptable processes. Our typed framework offers a simple alternative for integrating runtime adaptation mechanisms in the modeling and analysis of structured communications. We show that well-typed processes enjoy *safety* and *consistency* properties: while the former property ensures the absence of communication errors at runtime, the latter guarantees that active session behavior is never disrupted by adaptation actions.

1. Introduction

Session types offer a powerful type-theoretic foundation for the analysis of complex scenarios of structured communications, as frequently found in service-oriented systems. They abstract communication protocols as basic interaction patterns, which may then be checked against specifications in some core programming calculus—typically, a variant of the π-calculus [24]. Introduced in [19,20], session type theories have been extended in multiple directions—see [12] for a survey. Their practical relevance is witnessed by, e.g., their successful application to the analysis of collaborative, distributed workflows in healthcare services [18].

In spite of these developments, we find that existing process frameworks based on session types do not adequately support mechanisms for *runtime adaptation*. As distributed systems and applications are being deployed in open, highly dynamic

* Corresponding authors.

http://dx.doi.org/10.1016/j.scico.2014.04.017
0167-6423/

infrastructures (such as cloud computing platforms), runtime adaptation appears as a key feature to ensure continued system operation, reduce costs, and achieve business agility. We understand runtime adaptation as the dynamic modification of (the behavior of) the system as a response to an exceptional external event. Even if such events may not be catastrophic, they are often hard to predict. The initial system specification must explicitly describe the sub-systems amenable to adaptation and their exceptional events. Then, on the basis of this initial specification and its projected evolution, an array of possible adaptation routines is defined at design time. Runtime adaptation then denotes potential behavior, in the sense that a given adaptation routine is triggered only if its associated exceptional event takes place. While channel mobility present in session languages (commonly referred to as *delegation*) is useful to specify distribution of processing via types [20], we find that runtime adaptation, in the sense just discussed, is hard to specify and reason about in those languages.

We thus observe a substantial gap between (i) the adaptation capabilities of communication-based systems in practice, and (ii) the forms of interaction available in existing (typed) process frameworks developed to reason about the correctness of such systems.

In this paper we propose an alternative for filling in this gap. We introduce a session type discipline for a language equipped with mechanisms for runtime adaptation. Rather than developing yet another session type discipline *from scratch*, we have deliberately preferred to build upon existing lines of work. Our proposal builds upon the framework of *adaptable processes*, an attempt for enhancing process calculi specifications with evolvability mechanisms which we have developed together with Bravetti and Zavattaro in [3]. We combine the constructs for adaptable processes with the main insights of the session type system put forward by Garralda et al. [16] for the Boxed Ambient calculus [6]. Since the type system in [16] does not support delegation, we incorporate this key mechanism by relying on the "liberal" typing system developed by Yoshida and Vasconcelos in [29]. As a result of this integration of concepts, we obtain a simple yet expressive model of structured communications with explicit mechanisms for runtime adaptation.

We briefly describe our approach and results. Our process language includes the usual π-calculus constructs for session communication, but extended with the *located processes* and the *update processes* introduced in [3]. Given a location l, a process P, and a context Q (i.e. a process with zero or more free occurrences of variable X), these two processes are noted $l[P]$ and $l\{(\mathsf{X}).Q\}$, respectively. They may synchronize on l so as to evolve into process $Q[^{P}/_{\mathsf{X}}]$—the process Q in which all free occurrences of X are replaced with P. This interaction represents the *update* of process P at l with an *adaptation routine* embodied by Q, thus realizing the vision of runtime adaptation hinted at above. Locations can be nested and are transparent: within $l[P]$, process P may evolve autonomously, with the potential of interacting with some update process for l.

In our language, communicating behavior coexists with update actions. This raises the need for disciplining both forms of interaction, in a way such that protocol abstractions given by session types are respected and evolvability requirements are enforced. To this end, by observing that our update actions are a simple form of (higher-order) process mobility [26], we draw inspiration from the session types in [16], which ensure that sessions within Ambient hierarchies are never disrupted by Ambient mobility steps. By generalizing this insight to the context of (session) processes which execute in arbitrary, possibly nested locations, we obtain a property which we call *consistency*: update actions over located processes which are engaged in active session behavior cannot be enabled.

To show how located and update processes fit in a session-typed process language, and to illustrate our notion of consistency, we consider a simple distributed client/server scenario, conveniently represented as located processes:

$$\mathsf{Sys} \triangleq l_1[C_1] \mid l_2[r[S] \mid R] \quad \text{where:}$$

$$C_1 \triangleq \mathsf{request}\, a(x).x\langle \mathrm{u}_1, \mathrm{p}_1\rangle.x \lhd n_1.P_1.\mathsf{close}\,(x)$$

$$S \triangleq !\,\mathsf{accept}\, a(y).y(u,p).y \rhd \{n_1{:}Q_1.\mathsf{close}\,(y) \,[\!]\, n_2{:}Q_2.\mathsf{close}\,(y)\}$$

Intuitively, Sys consists of a replicated server S and a client C_1, hosted in different locations r and l_1, respectively. Process R, in location l_2, represents the platform in which S is deployed. The client C_1 and the (persistent) server S may synchronize on name a to establish a new session. After that, the client first sends its credentials to the server; then, she chooses one of the two labeled alternatives offered by the server. Above, client C_1 selects the alternative on label n_1; the subsequent client and server behaviors are abstracted by processes P_1 and Q_1, respectively. Finally, server and client synchronize to close the session.

Starting from Sys, let us suppose that a new session is indeed established by synchronization on a. Our semantics decrees a reduction step $\mathsf{Sys} \longrightarrow \mathsf{Sys}'$:

$$\mathsf{Sys}' = (\nu\kappa)(l_1[\kappa^+\langle \mathrm{u}_1, \mathrm{p}_1\rangle.\kappa^+ \lhd n_1.P_1.\mathsf{close}\,(\kappa^+)] \mid l_2[r[\kappa^-(u,p).\kappa^- \rhd \{n_1{:}Q_1.\mathsf{close}\,(\kappa^-) \,[\!]\, n_2{:}Q_2.\mathsf{close}\,(\kappa^-)\}] \mid R])$$

where κ^+ and κ^- denote the two *end-points* of channel κ [17]. Suppose now that R simply represents an upgrade process, which is ready to synchronize with r, the location in which S resides:

$$R = r\{(\mathsf{X}).NewS\}$$

From Sys′, an update on r would be highly inconvenient for at least two reasons:

(a) First, since r contains the local server behavior for an already established session, a careless update action on r could potentially discard such behavior. This would leave the client in l_1 without a partner—the protocol agreed upon session establishment would not be respected.
(b) Second, since r contains also the actual service definition S, an undisciplined update action on r could affect the service on a in a variety of ways—for instance, it could destroy it. This clearly goes against the expected nature of services, which should be always available.

Above, item (a) concerns our intended notion of consistency, which ensures that any update actions on r (such as those involving R above) are only enabled when r contains no active sessions. Closely related, item (b) concerns a most desirable principle for services, namely that "services should always be available in multiple copies"—this is the Service Channel Principle (SCP) given in [9].

Contributions. The main contribution of this paper is a session typed framework with runtime adaptation. Our framework lies upon two main technical ingredients:

(1) An *operational semantics* for our session language with located and update processes [3]. The semantics enables adaptation actions within *arbitrary process hierarchies* and, following [16], endows each located process with a suitable *runtime annotation*, which describes its active session behavior. Runtime annotations for locations are key in avoiding undesirable update actions such as the described in (a) above.
(2) A *typing system* which extends existing session type systems [29,16] with the notion of *interface*, which allows for simple and intuitive static checking rules for evolvability constructs. In particular, interfaces are essential to rule out careless updates such as the described in (b) above. This typing system provides a static analysis technique for ensuring not only *safety*, i.e., absence of communication errors at runtime, but also consistency, as described above.

To the best of our knowledge, our framework is the first in amalgamating structured communications and runtime adaptation from a session types perspective (either binary or multiparty).

Organization. The following section introduces our process language, a session π-calculus with adaptable processes. In Section 3 our session type system is presented; its main properties, namely safety and consistency, are defined and investigated in Section 4. The typed approach is illustrated via examples in Section 5, where the client/server scenario discussed above is revisited. Extensions and enhancements for our framework are discussed in Section 6: they concern the runtime adaptation of processes with active sessions, and the incorporation of recursive types and subtyping. Finally, Section 7 discusses related work and Section 8 collects some concluding remarks.

This paper is a revised version of the conference paper [14], extended with further examples and discussions. See Section 7 for further comparisons with respect to [14]. This presentation also contains the proofs of the main technical results; most of them are collected in Appendix B.

2. A process language with runtime adaptation

We extend standard session-typed languages (see, e.g., [20,29]) with *located processes* and *update actions*. These two constructs, extracted from [3], allow us to explicitly represent runtime adaptation within models of structured communications. This section introduces the syntax and semantics of our process model, and illustrates some of the adaptation patterns expressible in it.

2.1. Syntax

Our syntax builds upon the following (disjoint) base sets: *names*, ranged over by $a, b, \ldots$; *locations*, ranged over by $l, l', \ldots$; *labels*, ranged over by $n, n', \ldots$; *constants* (integers, booleans, names), ranged over by $c, c', \ldots$; *process variables*, ranged over by $\mathsf{X}, \mathsf{X}', \ldots$. Then, *processes*, ranged over by $P, Q, R, \ldots$ and *expressions*, ranged over $e, e', \ldots$ are given by the grammar in Table 1. We write $\widetilde{e}$ to denote a finite sequence of expressions $e_1, \ldots, e_n$; a similar convention applies for variables. Notice that we also use *(polarized) channels*, ranged over by $\kappa^p, \kappa_1^p, \ldots$, where $p \in \{+, -\}$. In the following, $j, h, m, \ldots$ range over $\mathbb{N}$.

We now comment on constructs in Table 1; in most cases, intuitions and conventions are as expected [20]. Prefixes $\mathsf{accept}\, a(x)$ and $\mathsf{request}\, a(y)$ use name a to establish a new session. Once a session is established, structured behavior on channels is possible. We sometimes refer to $\mathsf{accept}\, a(x).P$ as a *service* and to $!\mathsf{accept}\, a(x).P$ as a *persistent service*. In the same vein, we sometimes refer to $\mathsf{request}\, a(x).Q$ as a *service request*. Prefix $\mathsf{close}\,(k)$ is used to explicitly terminate session k. Having this prefix is crucial to our approach, for it allows us to keep track of the open sessions at any given time. The exchange of expressions is as usual; channel passing (delegation) is also supported. Thus, our language supports transparent distribution of processing via channel passing as well as the more expressive runtime adaptation via *located* and *update processes*, as we discuss below. With the aim of highlighting the novel aspect of runtime adaptation, in this section we consider infinite behavior in the form of replicated services; in Section 6.2 we show how to include recursion in the language. Also, we consider a restriction operator over channels only—restriction over names is not supported.

Table 1
Process syntax.

a, b, c	::=	x, y, z	variables
	|	u, v, w	names
k	::=	x, y, z	variables
	|	κ^+, κ^-	channels
P	::=	$\mathsf{request}\, a(x).P$	session request
	|	$\mathsf{accept}\, a(x).P$	session acceptance
	|	$!\,\mathsf{accept}\, a(x).P$	persistent session acceptance
	|	$k\langle\widetilde{e}\rangle.P$	data output
	|	$k(\widetilde{x}).P$	data input
	|	$k\langle\langle k'\rangle\rangle.P$	channel output
	|	$k((x)).P$	channel input
	|	$k \lhd n; P$	selection
	|	$k \rhd \{n_1{:}P_1 \,[\!]\, \cdots \,[\!]\, n_m{:}P_m\}$	branching
	|	$l[P]$	located process
	|	$l\{(\mathsf{X}).P\}$	update process
	|	X	process variable
	|	if e then P else Q	conditional
	|	$P \mid P$	parallel composition
	|	$\mathsf{close}\,(k).P$	close session
	|	$(\nu k)P$	channel hiding
	|	$\mathbf{0}$	inaction
e	::=	c	constants
	|	$e_1 + e_2 \mid e_1 - e_2 \mid \mathtt{not}(e) \mid \ldots$	expressions

Table 2
Reduction semantics.

(R:Par)	if $P \longrightarrow P'$ then $P \mid Q \longrightarrow P' \mid Q$
(R:Res)	if $P \longrightarrow P'$ then $(\nu\kappa)P \longrightarrow (\nu\kappa)P'$
(R:Str)	if $P \equiv P'$, $P' \longrightarrow Q'$, and $Q' \equiv Q$ then $P \longrightarrow Q$
(R:Open)	$E\{C\{\mathsf{accept}\, a(x).P\} \mid D\{\mathsf{request}\, a(y).Q\}\} \longrightarrow E^{++}\{(\nu\kappa)(C^+\{P[\kappa^+/x]\} \mid D^+\{Q[\kappa^-/y]\})\}$
(R:ROpen)	$E\{C\{!\,\mathsf{accept}\, a(x).P\} \mid D\{\mathsf{request}\, a(y).Q\}\} \longrightarrow E^{++}\{(\nu\kappa)(C^+\{P[\kappa^+/x] \mid\, !\,\mathsf{accept}\, a(x).P\} \mid D^+\{Q[\kappa^-/y]\})\}$
(R:Upd)	$E\{C\{l^0[P]\} \mid D\{l\{(X).Q\}\}\} \longrightarrow E\{C\{Q[^P/\mathsf{X}]\} \mid D\{\mathbf{0}\}\}$
(R:I/O)	$E\{C\{\kappa^{\,p}\langle\widetilde{e}\rangle.P\} \mid D\{\kappa^{\,\overline{p}}(\widetilde{x}).Q\}\} \longrightarrow E\{C\{P\} \mid D\{Q[\widetilde{c}/\widetilde{x}]\}\} \quad (\widetilde{e} \downarrow \widetilde{c})$
(R:Pass)	$E\{C\{\kappa^{\,p}\langle\langle\kappa'^{\,q}\rangle\rangle.P\} \mid D\{\kappa^{\,\overline{p}}((x)).Q\}\} \longrightarrow E\{C^-\{P\} \mid D^+\{Q[\kappa'^{\,q}/x]\}\}$
(R:Sel)	$E\{C\{\kappa^{\,p} \rhd \{n_1{:}P_1 \,[\!]\, \cdots \,[\!]\, n_m{:}P_m\}\} \mid D\{\kappa^{\,\overline{p}} \lhd n_j; Q\}\} \longrightarrow E\{C\{P_j\} \mid D\{Q\}\} \quad (1 \le j \le m)$
(R:Close)	$E\{C\{\mathsf{close}\,(\kappa^{\,p}).P\} \mid D\{\mathsf{close}\,(\kappa^{\,\overline{p}}).Q\}\} \longrightarrow E^{--}\{C^-\{P\} \mid D^-\{Q\}\}$
(R:IfTr)	$C\{\text{if } e \text{ then } P \text{ else } Q\} \longrightarrow C\{P\} \quad (e \downarrow \mathtt{true})$
(R:IfFa)	$C\{\text{if } e \text{ then } P \text{ else } Q\} \longrightarrow C\{Q\} \quad (e \downarrow \mathtt{false})$

As hinted at in the Introduction, a *located process* $l[P]$ denotes a process P deployed at location l. Inside l, process P can evolve on its own and interact with external processes. In $l[P]$, we use l as a reference for a potential update action, which occurs by interaction with an *update process* $l\{(\mathsf{X}).Q\}$—a built-in adaptation mechanism. In $l\{(\mathsf{X}).Q\}$, we use Q to denote a process with zero or more occurrences of process variable X. As formalized by the operational semantics in Section 2.2, an *update action* at l corresponds to the interaction of $l[P]$ and $l\{(\mathsf{X}).Q\}$ which leads to process $Q[^P/\mathsf{X}]$—the process Q in which free occurrences of X are substituted with P. In the semantics, we shall consider *annotated* located processes $l^h[P]$, where h stands for the number of active sessions in P. This runtime annotation is used by the type discipline in Section 3 to ensure that update actions do not disrupt the active session behavior deployed at a given location—this is the *consistency* guarantee, formally addressed in Section 4.2.

Binding is as follows: variable x is bound in processes $\mathsf{request}\, a(x).P$, $!\,\mathsf{accept}\, a(x).P$, and $\mathsf{accept}\, a(x).P$; similarly, $\widetilde{x}$ is bound in $k(\widetilde{x}).P$ (variables $x_1, \ldots, x_n$ are all distinct). Also, process variable X is bound in the update process $l\{(\mathsf{X}).P\}$. Based on these intuitions, given a process P, its sets of free/bound channels, variables, and process variables—noted $\mathsf{fc}(P)$, $\mathsf{fv}(P)$, $\mathsf{fpv}(P)$, $\mathsf{bc}(P)$, $\mathsf{bv}(P)$, and $\mathsf{bpv}(P)$, respectively—are defined as expected. In all cases, we rely on expected notions of α-conversion (noted $\equiv_\alpha$) and (capture-avoiding, simultaneous) substitution, noted $[\widetilde{c}/\widetilde{x}]$ (for data), $[\kappa^{\,p}/x]$ (for channels), and $[^P/\mathsf{X}]$ (for processes). We work only with closed processes, and often omit the trailing $\mathbf{0}$.

2.2. Semantics

The semantics of our process language is given by a *reduction semantics*, denoted $P \longrightarrow P'$, the smallest relation on processes generated by the rules in Table 2. As customary, it relies on an evaluation relation on expressions (noted $e \downarrow c$) and on a structural congruence relation, denoted $\equiv$, which we define next.

Definition 1 *(Structural congruence).* Structural congruence is the smallest congruence relation on processes that is generated by the following laws:

$$P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R)$$
$$P \mid \mathbf{0} \equiv P \qquad P \equiv Q \text{ if } P \equiv_\alpha Q$$
$$(\nu\kappa)\mathbf{0} \equiv \mathbf{0} \qquad (\nu\kappa)(\nu\kappa')P \equiv (\nu\kappa')(\nu\kappa)P$$
$$(\nu\kappa)P \mid Q \equiv (\nu\kappa)(P \mid Q) \quad (\text{if } \kappa \notin \mathsf{fc}(Q)) \qquad (\nu\kappa)l^h[P] \equiv l^h[(\nu\kappa)P]$$

In Table 2, duality for polarities p is as expected: $\overline{+} = -$ and $\overline{-} = +$. We write $\longrightarrow^*$ for the reflexive, transitive closure of $\longrightarrow$. As processes can be arbitrarily nested inside locations, reduction rules use *contexts*, i.e., processes with a *hole* $\bullet$.

Definition 2 *(Contexts).* The set of *contexts* is defined by the following syntax:

$$C, D, E, \ldots ::= \bullet \mid l^h[C \mid P]$$

Given a context C and a process P, we write $C\{P\}$ to denote the process obtained by filling in the occurrences of hole $\bullet$ in C with P. The intention is that P may reduce inside C, thus reflecting the transparent containment realized by location nesting.

We assume the expected extension of $\equiv$ to contexts; in particular, we tacitly use $\equiv$ to enlarge the scope of restricted channels in contexts as needed. As mentioned above, reduction relies on located processes with annotations h, which denote the number of active sessions at every location. To ensure the coherence of such annotations along reduction, we define operations over contexts which allow us to increase/decrease the annotation h on every location contained in a context.

Definition 3 *(Operations on contexts).* Given a context C as in Definition 2, a natural number j, and an operator $* \in \{+, -\}$, we define C^{*j} as follows:

$$(\bullet)^{*j} = \bullet \qquad (l^h[C \mid P])^{*j} = l^{h*j}[C^{*j} \mid P]$$

This way, for instance, C^{+1} denotes the context C in which the runtime annotations for all locations have been incremented by one. We write C^-, C^+, C^{--}, and C^{++} to stand for C^{-1}, C^{+1}, C^{-2}, and C^{+2}, respectively.

We now comment on reduction rules (R:Open), (R:Upd), (R:Pass), and (R:Close) in Table 2; other rules are either standard or self-explanatory.

Rule (R:Open) formalizes session establishment. There are three distinguished contexts: while C contains the service offer and D encloses the service request, context E encloses both C and D. By matching on name a a session is established; following [17,29], this is realized by endowing each partner with a fresh polarized channel (or end-point) κ^p. As such, channels κ^+ and κ^- are runtime entities. Because of session initiation, the number of active sessions should be increased across all enclosing contexts: relying on the operators given in Definition 3, we increment by one in contexts C and D and by two in context E, for it encloses both endpoints. This increment realizes a form of protection against careless update actions; observe that due to the transparency of locations, any updates may take place independently from the actual nested structure.

The reasons for the increment just described should be clear from rule (R:Upd), which formalizes the update/reconfiguration of a location l, enclosed in contexts C and E. Notice how the update action can only occur if the number of open (active) sessions in l is 0. By forcing updates to occur only when any (located) session behavior has completed (cf. rule (R:Close)), reduction ensures that active sessions are not inadvertently disrupted. When enabled, an update action is realized by substituting, within location l, all free occurrences of X in Q with P (the current behavior at l). Hence, it is the adaptation routine which "moves" to realize reconfiguration. This is a form of *objective update*, as the located process does not contain information on future update actions: it reduces autonomously until it is adapted by an update process in its environment.

Rule (R:Pass) is the standard rule for delegation; in our setting, endpoint mobility is reflected by appropriate decrements/increments in the contexts enclosing the session sender/receiver. Rule (R:Close) formalizes the synchronized session closure. In analogy with rule (R:Open), session closure should decrease the active session annotation in all enclosing contexts.

2.3. Examples of runtime adaptation

We now present some patterns of runtime adaptation that can be expressed in our process framework. Recall that $l_1, l_2, \ldots$ denote identifiers for locations. We discuss different reductions, resulting from the interaction between a located process and a corresponding adaptable process. These reductions are *enabled*; for the sake of readability, however, we elide the runtime annotation associated to the locations ($h = 0$ in all cases).

Relocation. One of the simplest reconfiguration actions that can be represented in our model is the relocation of an arbitrary behavior to a completely different computation site. The following reduction illustrates how a synchronization on location l_1 leads to a relocation from l_1 to l_2:

$$l_1[\,l_4[\,Q\,]\,] \mid l_1\{(\mathsf{X}).l_2[\mathsf{X}]\} \longrightarrow l_2[\,l_4[\,Q\,]\,] \mid \mathbf{0}$$

It is worth observing how a relocation does not alter the behavior at l_1. In particular, relocations are harmless to open sessions in Q, if any.

Deep update. Because our locations are transparent, an update action may have influence on located processes not necessarily at top-level in the nested process structure. The reduction below illustrates how the influence of an update process on name l_3 can cross locations l_1 and l_2 in order to realize an adaptation routine, represented by process S':

$$l_1[Q \mid l_2[R \mid l_3[S_1]]] \mid l_3\{(\mathsf{X}).l_4[S']\} \longrightarrow l_1[Q \mid l_2[R \mid l_4[S_2]]] \mid \mathbf{0}$$

where $S_2 = S'[^{S_1}/_{\mathsf{X}}]$. That is, by virtue of the update action on l_3, its current behavior (denoted S_1) is suspended and possibly used in the new behavior S', which is now located at l_4.

Upgrade. Interestingly, update actions do not need to preserve the current behavior at a given location. In fact, if the adaptation routine embodied by the updated process does not contain a process variable, then the current behavior at the location will be discarded. This feature is illustrated by the following reduction, in which we assume that $\mathsf{X} \notin \mathsf{fv}(Q)$:

$$l_1[P] \mid l_1\{(\mathsf{X}).Q\} \longrightarrow Q \mid \mathbf{0}$$

Observe that the location on which the update action takes place does not need to be preserved either: had we wanted to only replace the behavior at l_1, then it would have sufficed to enclose the runtime adaptation code Q within a located process named l_1, i.e., $l_1\{(\mathsf{X}).l_1[\,Q\,]\}$.

Conditional backup. The current behavior of a location may be used more than once by an adaptation routine. Consider process B_e below:

$$B_e = l_1[\,Q\,] \mid l_5[\,\mathsf{if}\ e\ \mathsf{then}\ l_1\{(\mathsf{X}).l_2[\mathsf{X}]\}\ \mathsf{else}\ l_1\{(\mathsf{X}).l_1[\mathsf{X}] \mid l_3[\mathsf{X}]\}\,]$$

Depending on the boolean expression e reduces to, B_e may either (i) simply relocate the behavior at l_1, or (ii) define a "backup copy" of Q at l_3:

$$\begin{array}{lll} B_e & \longrightarrow^* & l_2[\,Q\,] \mid l_5[\mathbf{0}] \qquad \text{if } e \downarrow \texttt{true} \\ B_e & \longrightarrow^* & l_1[\,Q\,] \mid l_3[\,Q\,] \mid l_5[\mathbf{0}] \quad \text{if } e \downarrow \texttt{false} \end{array}$$

The previous examples are useful to illustrate the expressiveness of adaptable processes for representing rich adaptation mechanisms. As our process model includes both communication and update actions, we require mechanisms for harmonizing them, avoiding undesirable disruptions of communication behavior by updates. In the next section, we define a static analysis technique that enables update actions when the given location does not enclose open sessions.

3. The type system

Our type system builds upon the one in [29], extending it so as to account for disciplined runtime adaptation. A main criteria in the design of our type discipline is *conservativity*: we would like to enforce both structured communication and disciplined adaptation by preserving standard models of session types as much as possible.

3.1. Type syntax

We now define our type syntax, which is rather standard.

Definition 4 *(Types).* The syntax of *basic types* (ranged over by $\tau, \sigma, \ldots$) and *session types* (ranged over $\alpha, \beta, \ldots$) is given in Table 3.

We recall the intuitive meaning of session types. We write $\widetilde{\tau}$ to denote a sequence of base types $\tau_1, \ldots, \tau_n$. Type $?(\widetilde{\tau}).\alpha$ (resp. $?(\beta).\alpha$) abstracts the behavior of a channel which receives values of types $\widetilde{\tau}$ (resp. a channel of type β) and continues as α afterwards. Complementarily, type $!(\widetilde{\tau}).\alpha$ (resp. $!(\beta).\alpha$) represents the behavior of a channel which sends values (resp. a channel) and that continues as α afterwards. Type $\&\{n_1 : \alpha_1, \ldots, n_m : \alpha_m\}$ describes a branching behavior, or external choice, along a channel: it offers m behaviors, and if the j-th alternative is selected then it behaves as described by type α_j $(1 \leq j \leq m)$. In turn, type $\oplus\{n_1 : \alpha_1, \ldots, n_m : \alpha_m\}$ describes the behavior of a channel which may select a single behavior among $\alpha_1, \ldots, \alpha_m$. This is an internal choice, which continues as α_j afterwards. Finally, type ϵ represents a channel with no communication behavior. We now introduce the central notion of *duality* for (session) types.

Table 3
Types and typing environments. Interfaces $\mathcal{I}$ are formally introduced in Definition 6.

TYPES			
τ, σ	$::=$	$\mathsf{int} \mid \mathsf{bool} \mid \ldots$	basic types
α, β	$::=$	$!(\widetilde{\tau}).\alpha \mid ?(\widetilde{\tau}).\alpha$	send, receive
	$\mid$	$!(\beta).\alpha \mid ?(\beta).\alpha$	throw, catch
	$\mid$	$\&\{n_1 : \alpha_1, \ldots, n_m : \alpha_m\}$	branch
	$\mid$	$\oplus\{n_1 : \alpha_1, \ldots, n_m : \alpha_m\}$	select
	$\mid$	ϵ	closed session
ENVIRONMENTS			
q	$::=$	$\mathsf{lin} \mid \mathsf{un}$	type qualifiers
Δ	$::=$	$\emptyset \mid \Delta, k : \alpha \mid \Delta, [k : \alpha]$	typing with active sessions
Γ	$::=$	$\emptyset \mid \Gamma, e : \tau \mid \Gamma, a : \langle \alpha_{\mathsf{q}}, \overline{\alpha}_{\mathsf{q}} \rangle$	first-order environment
Θ	$::=$	$\emptyset \mid \Theta, \mathsf{X} : \mathcal{I} \mid \Theta, l : \mathcal{I}$	higher-order environment

Table 4
Dual of session types.

$\overline{\epsilon}$	$=$	ϵ
$\overline{!(\widetilde{\tau}).\alpha}$	$=$	$?(\widetilde{\tau}).\overline{\alpha}$
$\overline{?(\widetilde{\tau}).\alpha}$	$=$	$!(\widetilde{\tau}).\overline{\alpha}$
$\overline{!(\beta).\alpha}$	$=$	$?(\beta).\overline{\alpha}$
$\overline{?(\beta).\alpha}$	$=$	$!(\beta).\overline{\alpha}$
$\overline{\&\{n_1 : \alpha_1, \ldots, n_m : \alpha_m\}}$	$=$	$\oplus\{n_1 : \overline{\alpha_1}, \ldots, n_m : \overline{\alpha_m}\}$
$\overline{\oplus\{n_1 : \alpha_1, \ldots, n_m : \alpha_m\}}$	$=$	$\&\{n_1 : \overline{\alpha_1}, \ldots, n_m : \overline{\alpha_m}\}$

Definition 5 *(Duality).* Given a session type α, its *dual* (noted $\overline{\alpha}$) is inductively defined in Table 4.

Our typing judgments generalize usual notions with an *interface* $\mathcal{I}$ (see Definition 6). Based on the syntactic occurrences of prefixes $\mathsf{request}\, a(x)$, $\mathsf{accept}\, a(x)$, and $\mathbf{!}\mathsf{accept}\, a(x)$, the interface of a process describes the (possibly persistent) services appearing in it. Thus, intuitively, the interface of a process gives an "upper bound" on the services that a process may execute. Formally, we have:

Definition 6 *(Interfaces).* We define an *interface* as the multiset whose underlying set of elements Int contains assignments from names to session types which are qualified (cf. Table 3). More precisely:

$$\mathrm{Int} = \left\{ \mathsf{q}\, a{:}\alpha \mid \mathsf{q} \in \{\mathsf{lin}, \mathsf{un}\},\ a \text{ is a name, and } \alpha \text{ is a session type} \right\}$$

We use $\mathcal{I}, \mathcal{I}', \ldots$ to range over interfaces. We sometimes write $\#_{\mathcal{I}}(a) = h$ to mean that element a occurs h times in $\mathcal{I}$.

Observe how several occurrences of the same service declaration are captured by the multiset nature of interfaces. The union of two interfaces $\mathcal{I}_1$ and $\mathcal{I}_2$ is essentially the union of their underlying multisets. We sometimes write $\mathcal{I} \uplus a : \alpha_{\mathsf{lin}}$ and $\mathcal{I} \uplus a : \alpha_{\mathsf{un}}$ to stand for $\mathcal{I} \uplus \{\mathsf{lin}\, a{:}\alpha\}$ and $\mathcal{I} \uplus \{\mathsf{un}\, a{:}\alpha\}$, respectively.

Notation 7 *(Interfaces).* We write $\mathcal{I}_{\mathsf{lin}}$ (resp. $\mathcal{I}_{\mathsf{un}}$) to denote the subset of $\mathcal{I}$ involving only assignments qualified with lin (resp. un). Moreover, we write $\mathcal{I}\uparrow^{\mathsf{un}}$ to denote the "persistent promotion" of $\mathcal{I}$. Formally, $\mathcal{I}\uparrow^{\mathsf{un}} = \mathcal{I} \setminus \mathcal{I}_{\mathsf{lin}} \uplus \{\mathsf{un}\, a{:}\alpha \mid \mathsf{lin}\, a{:}\alpha \in \mathcal{I}_{\mathsf{lin}}\}$.

It is useful to relate different interfaces. This is the intention of the relation $\sqsubseteq$ over interfaces, defined next.

Definition 8 *(Interface ordering).* Given interfaces $\mathcal{I}$ and $\mathcal{I}'$, we write $\mathcal{I} \sqsubseteq \mathcal{I}'$ iff

1. One of the following holds:
   (a) $\mathcal{I}_{\mathsf{lin}} \subseteq \mathcal{I}'_{\mathsf{lin}}$, where $\subseteq$ is the usual ordering on multisets, or
   (b) $\forall (\mathsf{lin}\, a{:}\alpha) \in \mathcal{I}_{\mathsf{lin}} \setminus \mathcal{I}'_{\mathsf{lin}}$ then $(\mathsf{un}\, a{:}\alpha) \in \mathcal{I}'_{\mathsf{un}}$
2. $\forall (\mathsf{un}\, a{:}\alpha) \in \mathcal{I}_{\mathsf{un}}$ then $(\mathsf{un}\, a{:}\alpha) \in \mathcal{I}'_{\mathsf{un}}$.

Interface equality is defined as: $\mathcal{I}_1 = \mathcal{I}_2$ iff $\mathcal{I}_1 \sqsubseteq \mathcal{I}_2$ and $\mathcal{I}_2 \sqsubseteq \mathcal{I}_1$.

In the light of the previous definitions, we may state:

Lemma 9. *Relation $\sqsubseteq$ is a preorder.*

Table 5
Well-typed processes (I).

$$(\text{T:Exp})\ \frac{}{\Gamma \vdash e : \tau}$$

$$(\text{T:NVar})\ \frac{}{\Gamma, x:\tau \vdash x : \tau}$$

$$(\text{T:Nil})\ \frac{}{\Gamma\,;\,\Theta \vdash \mathbf{0} \triangleright \emptyset;\emptyset}$$

$$(\text{T:LocEnv})\ \frac{}{\Theta, l:\mathcal{I} \vdash l : \mathcal{I}}$$

$$(\text{T:PVar})\ \frac{}{\Gamma;\Theta, \mathsf{X}:\mathcal{I} \vdash \mathsf{X} : \emptyset;\mathcal{I}}$$

$$(\text{T:Accept})\ \frac{\Gamma \vdash a : \langle \alpha_{\mathsf{lin}}, \overline{\alpha}_{\mathsf{lin}} \rangle \qquad \Gamma\,;\,\Theta \vdash P \triangleright \Delta, x:\alpha;\,\mathcal{I}}{\Gamma\,;\,\Theta \vdash \mathsf{accept}\, a(x).P \triangleright \Delta;\,\mathcal{I} \uplus a : \alpha_{\mathsf{lin}}}$$

$$(\text{T:RepAccept})\ \frac{\Gamma \vdash a : \langle \alpha_{\mathsf{un}}, \overline{\alpha}_{\mathsf{lin}} \rangle \qquad \Gamma\,;\,\Theta \vdash P \triangleright x:\alpha;\,\mathcal{I}}{\Gamma\,;\,\Theta \vdash \mathbf{!}\,\mathsf{accept}\, a(x).P \triangleright \emptyset;\,\mathcal{I}\!\uparrow^{\mathsf{un}} \uplus a : \alpha_{\mathsf{un}}}$$

$$(\text{T:Request})\ \frac{\Gamma \vdash a : \langle \alpha_{\mathsf{q}}, \overline{\alpha}_{\mathsf{lin}} \rangle \qquad \Gamma\,;\,\Theta \vdash P \triangleright \Delta, x:\overline{\alpha};\,\mathcal{I}}{\Gamma\,;\,\Theta \vdash \mathsf{request}\, a(x).P \triangleright \Delta;\,\mathcal{I} \uplus a : \overline{\alpha}_{\mathsf{lin}}}$$

$$(\text{T:Clo})\ \frac{\Gamma\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I} \qquad k \notin dom(\Delta)}{\Gamma\,;\,\Theta \vdash \mathsf{close}\,(k).P \triangleright \Delta, k:\epsilon;\mathcal{I}}$$

$$(\text{T:Loc})\ \frac{\Theta \vdash l : \mathcal{I} \qquad \Gamma\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I}' \qquad h = |\Delta| \qquad \mathcal{I}' \sqsubseteq \mathcal{I}}{\Gamma\,;\,\Theta \vdash l^h[P] \triangleright \Delta;\mathcal{I}'}$$

$$(\text{T:Adapt})\ \frac{\Theta \vdash l : \mathcal{I} \qquad \Gamma\,;\,\Theta, \mathsf{X}:\mathcal{I} \vdash P \triangleright \emptyset;\mathcal{I}'}{\Gamma\,;\,\Theta \vdash l\{(X).P\} \triangleright \emptyset;\emptyset}$$

$$(\text{T:CRes})\ \frac{\Gamma\,;\,\Theta \vdash P \triangleright \Delta, \kappa^- : \alpha, \kappa^+ : \overline{\alpha};\mathcal{I}}{\Gamma\,;\,\Theta \vdash (\nu\kappa)P \triangleright \Delta, [\kappa^- : \alpha], [\kappa^+ : \overline{\alpha}];\mathcal{I}}$$

$$(\text{T:Par})\ \frac{\Gamma\,;\,\Theta \vdash P \triangleright \Delta_1;\mathcal{I}_1 \qquad \Gamma\,;\,\Theta \vdash Q \triangleright \Delta_2;\mathcal{I}_2}{\Gamma\,;\,\Theta \vdash P \mid Q \triangleright \Delta_1 \cup \Delta_2;\mathcal{I}_1 \uplus \mathcal{I}_2}$$

3.2. Environments, judgments and typing rules

The typing environments we rely on are defined in the lower part of Table 3. In addition to interfaces $\mathcal{I}$, we consider typings Δ and environments Γ and Θ.

Typing Δ is commonly used to collect assignments from channels to session types; as such, it describes currently active sessions. In our discipline, in Δ we also include *bracketed assignments*, denoted $[\kappa^p : \alpha]$, which represent active but restricted sessions. As we discuss below, bracketed assignments arise in the typing of restriction, and are key to keep a precise count of the active sessions in a given located process. We write $dom(\Delta)$ to denote the set $\{k^p \mid k^p : \alpha \in \Delta \vee [k^p : \alpha] \in \Delta\}$. We write $\Delta, k:\alpha$ and $\Delta, [k:\alpha]$ where $k \notin dom(\Delta)$.

Γ is a first-order environment which maps expressions to basic types and names to pairs of *qualified* session types. In the interface, a session type is qualified with 'un' if it is associated to a persistent service; otherwise, it is qualified with 'lin'.

The higher-order environment Θ collects assignments of process variables and locations to interfaces. While the former kind of assignments is relevant to update processes, the latter concern located processes. As we explain next, by relying on the combination of these two pieces of information the type system ensures that runtime adaptation actions preserve the behavioral interfaces of a process. We write $vdom(\Theta) = \{\mathsf{X} \mid \mathsf{X} : \mathcal{I} \in \Theta\}$ to denote the variables in the domain of Θ. Given these environments, a *type judgment* is of form

$$\Gamma\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I}$$

meaning that, under environments Γ and Θ, process P has active sessions declared in Δ and interface $\mathcal{I}$. We then have:

Definition 10. A process is well-typed if it can be typed using the rules in Tables 5 and 6.

We comment on some rules in Table 5. Given a process which implements session type α on channel x, rule (T:Accept) types a service on name a. Observe how x is removed from Δ whereas $\mathcal{I}$ is appropriately extended with $a : \alpha_{\mathsf{lin}}$. Rule (T:RepAccept) is the analogous of (T:Accept) for persistent services. In that rule, observe how the linear services in $\mathcal{I}$ are "promoted" to persistent services via $\mathcal{I}\!\uparrow^{\mathsf{un}}$ (cf. Notation 7). Non-persistent services that appear in the context of a persistent service a are meant to be executed at most once for each instance of a. In fact, after promotion the declaration in Γ for a non-persistent service $b : \langle \alpha_{\mathsf{lin}}, \overline{\alpha}_{\mathsf{lin}} \rangle$ remains unchanged, but its entry in $\mathcal{I}$ should be $\mathsf{un}\, b{:}\alpha$, as we could now observe

Table 6
Well-typed processes (II).

$$(\text{T:Thr})\ \frac{\Gamma\,;\,\Theta \vdash P \triangleright \Delta, k:\beta;\mathcal{I}}{\Gamma\,;\,\Theta \vdash k\langle\langle k'\rangle\rangle.P \triangleright \Delta, k:!(\alpha).\beta, k':\alpha;\mathcal{I}}$$

$$(\text{T:Cat})\ \frac{\Gamma\,;\,\Theta \vdash P \triangleright \Delta, k:\beta, x:\alpha;\mathcal{I}}{\Gamma\,;\,\Theta \vdash k((x)).P \triangleright \Delta, k:?(\alpha).\beta;\mathcal{I}}$$

$$(\text{T:In})\ \frac{\Gamma, \widetilde{x}:\widetilde{\tau}\,;\,\Theta \vdash P \triangleright \Delta, k:\alpha;\mathcal{I}}{\Gamma\,;\,\Theta \vdash k(\widetilde{x}).P \triangleright \Delta, k:?(\widetilde{\tau}).\alpha;\mathcal{I}}$$

$$(\text{T:Out})\ \frac{\Gamma\,;\,\Theta \vdash P \triangleright \Delta, k:\alpha;\mathcal{I} \quad \Gamma \vdash \widetilde{e}:\widetilde{\tau}}{\Gamma\,;\,\Theta \vdash k(\widetilde{e}).P \triangleright \Delta, k:!(\widetilde{\tau}).\alpha;\mathcal{I}}$$

$$(\text{T:Weak})\ \frac{\Gamma\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I} \qquad \kappa^+, \kappa^- \notin dom(\Delta)}{\Gamma\,;\,\Theta \vdash (\nu\kappa)P \triangleright \Delta;\mathcal{I}}$$

$$(\text{T:If})\ \frac{\Gamma \vdash e:\mathsf{bool} \qquad \Gamma\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I} \qquad \Gamma\,;\,\Theta \vdash Q \triangleright \Delta;\mathcal{I}}{\Gamma\,;\,\Theta \vdash \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta;\mathcal{I}}$$

$$(\text{T:Bra})\ \frac{\Gamma\,;\,\Theta \vdash P_1 \triangleright \Delta, k:\alpha_1;\mathcal{I}_1 \quad \cdots \quad \Gamma\,;\,\Theta \vdash P_m \triangleright \Delta, k:\alpha_m;\mathcal{I}_m \quad \mathcal{I} = \mathcal{I}_1 \uplus \ldots \uplus \mathcal{I}_m}{\Gamma\,;\,\Theta \vdash k \rhd \{n_1:P_1 \,[\!]\, \cdots \,[\!]\, n_m:P_m\} \triangleright \Delta, k:\&\{n_1:\alpha_1, \ldots, n_m:\alpha_m\};\mathcal{I}}$$

$$(\text{T:Sel})\ \frac{\Gamma\,;\,\Theta \vdash P \triangleright \Delta, k:\alpha_i;\mathcal{I} \qquad 1 \le i \le m}{\Gamma\,;\,\Theta \vdash k \lhd n_i;\, P \triangleright \Delta, k:\oplus\{n_1:\alpha_1, \ldots, n_m:\alpha_m\};\mathcal{I}}$$

an unbounded number of executions of (non-persistent) service b. Given these typing rules, rule (T:Request) should be self-explanatory.

Rule (T:CRes) types channel restriction. The main difference wrt usual typing rules for channel restriction (cf. [29]) is that restricted end-points are not removed from Δ but kept in bracketed form, as motivated earlier. Using typing Δ, we can have an exact count of open, possibly restricted, sessions in a process. Rule (T:Close) types the explicit session closure construct, extending Δ with a fresh channel which is assigned to an empty session type. This may be useful to understand why our typing rule for the inactive process (rule (T:Nil)) requires an empty typing Δ.

Rule (T:Loc) performs two checks to type located processes. First, the runtime annotation h is computed by counting the assignments (standard and bracketed) declared in Δ (see Appendix A.1). Second, the rule checks that the interface of the located process is less or equal (in the sense of $\sqsubseteq$, cf. Notation 7) than the declared interface of the given location. Informally, this ensures that the process behavior does not "exceed" the expected behavior within the location. It is worth observing how a typed located processes has the exact same typing and interface of its contained process: this is how transparency of locations arises in typing. Finally, rule (T:Adapt) types update processes. Observe how the interface associated to the process variable of the given adaptation routine should match with the declared interfaces for the given location. However, for simplicity we establish no condition on the relation between $\mathcal{I}$ (the expected interface) and $\mathcal{I}'$ (the interface of the adaptation routine $P(\mathsf{X})$)—in Section 6.2 we discuss an alternative, more stringent, formulation.

Having introduced our typing system, the following section defines and states its main properties.

4. Session safety and consistency by typing

We now proceed to investigate *safety* and *consistency*, the main properties of our typing system. While safety (discussed in Section 4.1) corresponds to the expected guarantee of adherence to prescribed session types and absence of runtime errors, consistency (discussed in Section 4.2) formalizes a correct interplay between communication actions and update actions. Defining both properties requires the following notions of κ*-processes*, κ*-redexes*, and *error process*. These are classic ingredients in session types presentations (see, e.g., [20,29]); our notions generalize usual definitions to the case in which processes which may interact even if contained in arbitrarily nested transparent locations (formalized by the contexts of Definition 2).

Definition 11 *(κ-processes, κ-redexes, errors).* A process P is a κ*-process* if it is a prefixed process with subject κ, i.e., P is one of the following:

$$\begin{array}{ll}
\kappa^{\,p}(\widetilde{x}).P' & \kappa^{\,p}\langle v\rangle.P' \\
\kappa^{\,p}((x)).P' & \kappa^{\,p}\langle\langle k^{q}\rangle\rangle.P' \\
\kappa^{\,p} \rhd \{n_1:P_1 \,[\!]\, \cdots \,[\!]\, n_m:P_m\} & \kappa^{\,p} \lhd n.P' \\
\mathsf{close}\,(\kappa^{\,p}).P' &
\end{array}$$

Process P is a κ-*redex* if it contains the composition of exactly two κ-processes with opposing polarities, i.e., for some contexts C, D and E, and processes P_1, P_2, P_m, and P', we have that P is structurally congruent to one of the following:

$$(\nu\widetilde{\kappa})(E\{C\{\kappa^{\,p}(\widetilde{x}).P_1\} \mid D\{\kappa^{\,\bar{p}}\langle v\rangle.P_2\}\})$$
$$(\nu\widetilde{\kappa})(E\{C\{\kappa^{\,p}((x)).P_1\} \mid D\{\kappa^{\,\bar{p}}\langle\langle k^{\,q}\rangle\rangle.P_2\}\})$$
$$(\nu\widetilde{\kappa})(E\{C\{\kappa^{\,p} \rhd \{n_1{:}P_1 \,[\!]\, \cdots \,[\!]\, n_m{:}P_m\}\} \mid D\{\kappa^{\,\bar{p}} \lhd n_i;\, P'\}\})$$
$$(\nu\widetilde{\kappa})(E\{C\{\mathsf{close}\,(\kappa^{\,p}).P_1\} \mid D\{\mathsf{close}\,(\kappa^{\,\bar{p}}).P_2\}\})$$

We say a κ-redex is *located* if one or both of its κ-processes is inside at least one located process.

P is an *error* if $P \equiv (\nu\widetilde{\kappa})(Q \mid R)$ where, for some κ, Q contains either exactly two κ-processes that do not form a κ-redex or three or more κ-processes.

4.1. Session safety

We now give subject congruence and subject reduction results for our typing discipline. Together with some auxiliary results, these provide the basis for the proof of type safety (Theorem 22 in p. 246). We start by giving three standard results, namely weakening, strengthening, and channel lemmas.

Lemma 12 *(Weakening). Let* $\Gamma\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I}$. *If* $\mathsf{X} \notin vdom(\Theta)$ *then* $\Gamma\,;\,\Theta, \mathsf{X}:\mathcal{I}' \vdash P \triangleright \Delta;\mathcal{I}$.

Proof. Easily shown by induction on the structure of P. □

Lemma 13 *(Strengthening). Let* $\Gamma\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I}$. *If* $\mathsf{X} \notin \mathsf{fv}(P)$ *then* $\Gamma\,;\,\Theta \setminus \mathsf{X}:\mathcal{I}' \vdash P \triangleright \Delta;\mathcal{I}$.

Proof. Easily shown by induction on the structure of P. □

Lemma 14 *(Channel lemma). Let* $\Gamma\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I}$, $\kappa \notin \mathsf{fc}(P) \cup \mathsf{bc}(P)$ *iff* $\kappa \notin dom(\Delta)$.

Proof. Easily shown by induction on the structure of P. □

We are ready to show the Subject Congruence Theorem:

Theorem 15 *(Subject congruence). If* $\Gamma\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I}$ *and* $P \equiv Q$ *then* $\Gamma\,;\,\Theta \vdash Q \triangleright \Delta;\mathcal{I}$.

Proof. By induction on the derivation of $P \equiv Q$, with a case analysis on the last applied rule. See Appendix B.1 for details. □

The following auxiliary result concerns substitutions for channels, expressions, and process variables. Observe how the case of process variables has been relaxed so as to allow substitution with a process with "smaller" interface (in the sense of $\sqsubseteq$, cf. Notation 7). This extra flexibility is in line with the typing rule for located processes (rule (T:Loc)), and will be useful later on in proofs.

Lemma 16 *(Substitution lemma).*

1. *If* $\Gamma\,;\,\Theta \vdash P \triangleright \Delta, x:\alpha;\mathcal{I}$ *then* $\Gamma\,;\,\Theta \vdash P[\kappa^{\,p}/x] \triangleright \Delta, \kappa^{\,p}:\alpha;\mathcal{I}$.
2. *If* $\Gamma, \widetilde{x}:\widetilde{\tau}\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I}$ *and* $\Gamma \vdash \widetilde{e}:\widetilde{\tau}$ *then* $\Gamma\,;\,\Theta \vdash P[\widetilde{e}/\widetilde{x}] \triangleright \Delta;\mathcal{I}$.
3. *If* $\Gamma\,;\,\Theta, \mathsf{X}:\mathcal{I} \vdash P \triangleright \emptyset;\mathcal{I}_1$ *and* $\Gamma\,;\,\Theta \vdash Q \triangleright \emptyset;\mathcal{I}'$ *with* $\mathcal{I}' \sqsubseteq \mathcal{I}$ *then, for some* $\mathcal{I}_1'$, *we have* $\Gamma\,;\,\Theta \vdash P[^{Q}/_{\mathsf{X}}] \triangleright \emptyset;\mathcal{I}_1'$ *with* $\mathcal{I}_1' \sqsubseteq \mathcal{I}_1$.

Proof. Easily shown by induction on the structure of P. □

As reduction may occur inside contexts, in proofs it is useful to have *typed contexts*, building upon Definition 2. We thus have contexts in which the hole has associated typing information—concretely, the typing for processes which may fill in the hole. Defining contexts requires a simple extension of judgments, in the following way:

$$\mathcal{H};\,\Gamma\,;\,\Theta \vdash C \triangleright \Delta;\mathcal{I}$$

Intuitively, $\mathcal{H}$ contains the description of the type associated to the hole in C. Typing rules in Tables 5 and 6 are extended in the expected way. Because contexts have a single hole, $\mathcal{H}$ is either empty of has exactly one element. When $\mathcal{H}$ is empty, we write $\Gamma\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I}$ instead of $\cdot\,;\,\Gamma\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I}$. Two additional typing rules are required:

$$(\text{T:HOLE})\ \frac{}{\bullet_{\Gamma;\Theta\vdash\Delta;\mathcal{I}}\,;\,\Gamma\,;\,\Theta \vdash \bullet \triangleright \Delta;\mathcal{I}}$$

$$(\text{T:FILL})\ \frac{\bullet_{\Gamma;\Theta\vdash\Delta;\mathcal{I}}\,;\,\Gamma\,;\,\Theta \vdash C \triangleright \Delta_1;\mathcal{I}_1 \qquad \Gamma\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I}}{\Gamma\,;\,\Theta \vdash C\{P\} \triangleright \Delta_1;\mathcal{I}_1}$$

Axiom (T:HOLE) allows us to introduce typed holes into contexts. In rule (T:FILL), P is a process (it does not have any holes), and C is a context with a hole of type $\Gamma;\Theta \vdash \Delta;\mathcal{I}$. The substitution of occurrences of $\bullet$ in C with P, noted $C\{P\}$ is sound as long as the typings of P coincide with those declared in $\mathcal{H}$ for C. Based on these rules and Definitions 2 and 3, the following two auxiliary lemmas on properties of typed contexts follow easily. We first introduce some convenient notation for typed holes.

Notation 17. Let us use $\mathcal{S}, \mathcal{S}', \ldots$ to range over judgments attached to typed holes. This way, $\bullet_{\mathcal{S}}$ denotes the valid typed hole associated to $\mathcal{S} = \Gamma;\Theta \vdash \Delta;\mathcal{I}$.

A typed context may contain a typed hole in parallel with arbitrary behaviors. This may have consequences on the typing and the interface, as the following lemma formalizes:

Lemma 18. *Let P and C be a process and a typed context such that*

$$\Gamma\,;\,\Theta \vdash C\{P\} \triangleright \Delta;\mathcal{I}$$

is a derivable judgment. There exist $\Delta_1, \mathcal{I}_1$ such that (i) $\Gamma\,;\,\Theta \vdash P \triangleright \Delta_1;\mathcal{I}_1$ is a well-typed process, and (ii) $\Delta_1 \subseteq \Delta$ and $\mathcal{I}_1 \sqsubseteq \mathcal{I}$.

The following property formalizes the effect that a type hole has in the typing judgment of a context: under certain conditions, if the typing and interface of the hole change, then the judgment for the whole context should change as well.

Lemma 19. *Let C be a context as in Definition 2.*

1. *Suppose $\bullet_{\mathcal{S}};\Gamma\,;\,\Theta \vdash C \triangleright \Delta_1, k_2 : \beta, \Delta';\mathcal{I}_1 \uplus \mathcal{I}_2 \uplus \mathcal{I}'$ with $\mathcal{S} = \Gamma;\Theta \vdash \Delta_1, k_2 : \beta;\mathcal{I}_1 \uplus \mathcal{I}_2$ is well-typed. Let $\mathcal{S}' = \Gamma;\Theta \vdash \Delta_1, k_1 : \alpha, k_2 : \beta';\mathcal{I}_1$. Then*

   $$\bullet_{\mathcal{S}'};\Gamma\,;\,\Theta \vdash C^{+} \triangleright \Delta_1, k_1 : \alpha, k_2 : \beta', \Delta';\mathcal{I}_1 \uplus \mathcal{I}'$$

   is a derivable judgment.
2. *Suppose $\bullet_{\mathcal{S}};\Gamma\,;\,\Theta \vdash C \triangleright \Delta_1, k_1{:}\alpha, k_2{:}\beta, \Delta';\mathcal{I}_1 \uplus \mathcal{I}_2 \uplus \mathcal{I}'$ with $\mathcal{S} = \Gamma;\Theta \vdash \Delta_1, k{:}\alpha, k_2{:}\beta;\mathcal{I}_1 \uplus \mathcal{I}_2$ is well-typed. Let $\mathcal{S}' = \Gamma;\Theta \vdash \Delta_1, k_2{:}\beta';\mathcal{I}_1$. Then*

   $$\bullet_{\mathcal{S}'};\Gamma\,;\,\Theta \vdash C^{-} \triangleright \Delta_1, k_2{:}\beta', \Delta;\mathcal{I}_1 \uplus \mathcal{I}'$$

   is a derivable judgment.
3. *Suppose $\bullet_{\mathcal{S}};\Gamma\,;\,\Theta \vdash C \triangleright \Delta_C \cup \Delta_{\mathcal{S}};\mathcal{I}_C \uplus \mathcal{I}_{\mathcal{S}}$ with $\mathcal{S} = \Gamma;\Theta \vdash \Delta_{\mathcal{S}};\mathcal{I}_{\mathcal{S}}$ is well-typed. Let $\mathcal{S}' = \Gamma;\Theta \vdash \Delta_{\mathcal{S}'};\mathcal{I}_{\mathcal{S}'}$. Then*

   $$\bullet_{\mathcal{S}'};\Gamma\,;\,\Theta \vdash C \triangleright \Delta_C \cup \Delta_{\mathcal{S}'};\mathcal{I}_C \uplus \mathcal{I}_{\mathcal{S}'}$$

   is a derivable judgment.

The analogous of (1) and (2), involving bracketed assignments, are as expected.

We now introduce the usual notion of *balanced typing* [29]:

Definition 20 *(Balanced typings).* We say a typing Δ is *balanced* iff for all $\kappa^{p} : \alpha \in \Delta$ (resp. $[\kappa^{p} : \alpha] \in \Delta$) then also $\kappa^{\bar{p}} : \overline{\alpha} \in \Delta$ (resp. $[\kappa^{\bar{p}} : \overline{\alpha}] \in \Delta$).

The final requirement for proving safety via typing is the subject reduction theorem below.

Theorem 21 *(Subject reduction). If $\Gamma\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I}$ with Δ balanced and $P \longrightarrow Q$ then $\Gamma\,;\,\Theta \vdash Q \triangleright \Delta';\mathcal{I}'$, for some $\mathcal{I}'$ and balanced Δ'.*

Proof. By induction on the last rule applied in the reduction. See Appendix B.2 for details. □

We are now ready to state our first main result: the *absence of communication errors* for well-typed processes. Recall that our notion of error process has been given in Definition 11.

Theorem 22 *(Typing ensures safety). If* $\Gamma\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I}$ *with* Δ *balanced then* P *never reduces into an error.*

Proof. We assume, towards a contradiction, that there exists a P_1 such that $P \longrightarrow^* P_1$ and P_1 is an error process (as in Definition 11). By Theorem 21 (Subject Reduction), P_1 is well-typed under a balanced typing Δ_1. Following Definition 11, there are two possibilities for P_1, namely it contains (i) exactly two κ-processes which do not form a κ-redex and (ii) three or more κ-processes. Consider the first possibility. There are several combinations; by inversion on rule (T:CRES) we have that, for some session types α_1 and α_2, $\{[\kappa^{\,p} : \alpha_1], [\kappa^{\,\overline{p}} : \alpha_2]\} \subseteq \Delta_1$. In all cases, since the two κ-processes do not form a κ-redex then, necessarily, $\alpha_1 \neq \overline{\alpha_2}$. This, however, contradicts the definition of balanced typings (Definition 20). The second possibility again contradicts Definition 20, as in that case Δ_1 would capture the fact that at least one κ-process does not have a complementary partner for forming a κ-redex. We thus conclude that well-typed processes never reduce to an error. □

4.2. Session consistency

We now investigate *session consistency*: this is to enforce a basic discipline on the interplay of communicating behavior (i.e., session interactions) and evolvability behavior (i.e., update actions). Informally, a process P is called *consistent* if whenever it has a κ-redex (cf. Definition 11) then possible interleaved update actions do not destroy such a redex.

Below, we formalize this intuition. Let us write $P \longrightarrow_{\mathrm{upd}} P'$ for any reduction inferred using rule (R:UPD), possibly followed by uses of rules (R:RES), (R:STR), and (R:PAR). We then define:

Definition 23 *(Consistency).* A process P is *update-consistent* if and only if, for all P' and κ such that $P \longrightarrow^* P'$ and P' contains a κ-redex, if $P' \longrightarrow_{\mathrm{upd}} P''$ then P'' contains a κ-redex.

Recall that a *located* κ-redex is a κ-redex in which one or both of its constituting κ-processes are contained by least one located process. This way, for instance,

$$l_2[\,l_1[\kappa^{\,p}(\widetilde{x}).P_1] \mid \kappa^{\,\overline{p}}\langle v\rangle.P_2\,]$$
$$l_1[\kappa^{\,p}(\widetilde{x}).P_1] \mid l_2[\kappa^{\,\overline{p}}\langle v\rangle.P_2]$$
$$l_1[\kappa^{\,p}(\widetilde{x}).P_1 \mid \kappa^{\,\overline{p}}\langle v\rangle.P_2]$$

are located κ-redexes, whereas $\kappa^{\,p}(\widetilde{x}).P_1 \mid \kappa^{\,\overline{p}}\langle v\rangle.P_2$ is not. From the point of view of consistency, the distinction between located and unlocated κ-redexes is relevant: since update actions result from synchronizations on located processes, unlocated κ-redexes are always preserved by update actions, whereas located κ-redexes may be destroyed by an update action. We have the following auxiliary proposition.

Proposition 24. *Let* $\Gamma\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I}$*, with* Δ *balanced, be a well-typed process containing a* κ*-redex, for some* κ*. We have:*

(a) $\Delta = \Delta', \kappa^{\,p} : \alpha, \kappa^{\,\overline{p}} : \overline{\alpha}$ *or* $\Delta = \Delta', [\kappa^{\,p} : \alpha], [\kappa^{\,\overline{p}} : \overline{\alpha}]$*, for some session type* α*, and a balanced* Δ'*.*
(b) If the κ*-redex is located, then the runtime annotation for the location(s) hosting its constituting* κ*-processes is different from zero.*

Proof. Part (a) is immediate from our definition of typing judgment, in particular from the fact that typing Δ records the types of currently active sessions, as implemented by channels such as κ. Part (b) follows directly by definition of typing rule (T:LOC) and part (a), observing that typing relies on the cardinality of Δ to compute (non-zero) runtime annotations for locations. □

Theorem 25 *(Typing ensures update consistency). If* $\Gamma\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I}$*, with* Δ *balanced, then* P *is update consistent.*

Proof. We assume, towards a contradiction, that there exist P_1, P_2, and κ_1 such that (i) $P \longrightarrow^* P_1$, (ii) P_1 has a κ_1-redex, (iii) $P_1 \longrightarrow_{\mathrm{upd}} P_2$, and (iv) P_2 does not have a κ_1-redex. Without loss of generality, we suppose that the reduction $P_1 \longrightarrow_{\mathrm{upd}} P_2$ is due to a synchronization on location $l_1 \in \Theta$. Since the κ_1-redex is destroyed by the update action from P_1 to P_2, the κ_1-redex in P_1 must necessarily be a located κ_1-redex, i.e., in P_1, one or both κ_1-processes are contained inside l_1. Now, our reduction semantics (rule (R:UPD)) decrees that for such an update action to be enabled, the runtime annotation for l_1 in P_1 should be zero. However, by Theorem 21 (Subject Reduction), we know that P_1 is well-typed under a balanced typing Δ_1. Then, using well-typedness and Proposition 24(b) we infer that the annotation for l_1 in P_1 must be different

from zero: contradiction. Hence, update steps which destroy a κ-redex (located and unlocated) can never be enabled from a well-typed process with a balanced typing (such as P) nor from any of its derivatives (such as P_1). We thus conclude that well-typedness implies update consistency. □

5. Example: an adaptable client/service scenario

To illustrate how located and update processes in our framework extend the expressiveness of session-typed languages, we revisit the client/server scenario discussed in the Introduction:

$$\begin{aligned}
\mathsf{Sys} &\triangleq l_1[C_1] \mid l_2[r[S] \mid R] \quad \text{where:}\\
C_1 &\triangleq \mathsf{request}\, a(x).x\langle \mathrm{u}_1, \mathrm{p}_1\rangle.x \lhd n_1.P_1.\mathsf{close}\,(x)\\
S &\triangleq \mathbf{!}\,\mathsf{accept}\, a(y).y(u,p).y \rhd \{n_1 : Q_1.\mathsf{close}\,(y) \,[\!]\, n_2 : Q_2.\mathsf{close}\,(y)\}
\end{aligned}$$

Recall that process R above represents the platform in which service S is deployed. We now discuss two different definitions for R: this is useful to illustrate the different facets that runtime adaptation may adopt in our typed framework.

In what follows, we assume that Q_i realizes a behavior denoted by type β_i ($i \in \{1,2\}$), and write α to stand for the session type $?(\tau_1,\tau_2).\&\{n_1 : \beta_1, n_2 : \beta_2\}$ for the server S. Dually, the type for C_1 is $\overline{\alpha} = !(\tau_1,\tau_2).\oplus\{n_1 : \overline{\beta_1}, n_2 : \overline{\beta_2}\}$; we assume that P_1 realizes the session type $\overline{\beta_1}$.

5.1. A basic service reconfiguration: relocation and upgrade

We first suppose that R stands for a simple adaptation routine for S which (i) relocates the service from r to l_4, and (ii) sets up a new adaptation routine at l_4 which upgrades S with adaptation mechanisms for Q_1 and Q_2 (denoted R_{11} and R_{12}, respectively, and left unspecified):

$$\begin{aligned}
R &\triangleq r\{(\mathsf{X}).l_4[\mathsf{X}] \mid l_4\{(\mathsf{X}_1).l_4[S_{\mathsf{new}}]\}\} \quad \text{where:}\\
S_{\mathsf{new}} &\triangleq \mathbf{!}\,\mathsf{accept}\, a(y).y(u,p).y \rhd \{n_1 : Q_1^*.\mathsf{close}\,(y) \,[\!]\, n_2 : Q_2^*.\mathsf{close}\,(y)\}\\
Q_1^* &\triangleq l_5[Q_1] \mid l_5\{(\mathsf{X}_2).R_{11}\}\\
Q_2^* &\triangleq l_6[Q_2] \mid l_6\{(\mathsf{X}_3).R_{12}\}
\end{aligned}$$

It is easy to see that the only difference between S and S_{new} is in the behavior given at label n_2—for simplicity, we assume that Q_2 and Q_2^* are implementations of the same typed behavior. For this R, using our type system, we can infer that

$$\Gamma\,;\,\Theta \vdash \mathsf{Sys} \triangleright \emptyset; \mathcal{I}_1 \quad \text{where:}$$

- $\{a : \langle \alpha_{\mathsf{un}}, \overline{\alpha}_{\mathsf{lin}}\rangle\} \subseteq \Gamma$
- $\{l_1 \mapsto a : \overline{\alpha}_{\mathsf{lin}},\ l_4 \mapsto a : \alpha_{\mathsf{un}},\ r \mapsto a : \alpha_{\mathsf{un}},\ \mathsf{X} \mapsto a : \alpha_{\mathsf{un}}\} \subseteq \Theta$
- $\mathcal{I}_1 = \{\mathsf{lin}\, a : \overline{\alpha},\ \mathsf{un}\, a : \alpha\}$

and where we have slightly simplified notation for Θ, for readability reasons.

By virtue of Theorem 22 typing ensures communications between C_1 and S which follow the prescribed session types. Moreover, by relying on Theorem 25, we know that well-typedness implies consistency, i.e., an update action on r will not be enabled if C_1 and S have already initiated a session. For the scenario above, our typed framework ensures that updates may only take place before a session related to the service on a is established. Suppose such an action occurs as the first action of Sys (i.e. service S is immediately relocated, from r to l_4):

$$\mathsf{Sys} \longrightarrow l_1^0[C_1] \mid l_2^0[l_4^0[S] \mid l_4\{(\mathsf{X}_1).l_4[S_{\mathsf{new}}]\}] = \mathsf{Sys}_1$$

The above reduction represents one of the simplest forms of reconfiguration, one in which the behavior of the located process is not explicitly changed. Still, we observe that runtime relocation may *indirectly* influence the future behavior of a process. For instance, after relocation the process may synchronize with reconfiguration routines (i.e., update processes) not defined for the previous location. This is exactly what occurs with the above definition for R when relocating S to l_4—see below.[1] Also, the new location may be associated to a larger interface (wrt $\sqsubseteq$) and this could be useful to eventually enable reconfiguration steps not possible for the process in the old location.

Starting from Sys_1, the service S can be then upgraded to S_{new} by synchronizing on l_4. We may have:

$$\mathsf{Sys}_1 \longrightarrow l_1^0[C_1] \mid l_2^0[l_4^0[S_{\mathsf{new}}]] = \mathsf{Sys}_2$$

[1] Conversely, update actions that remove the enclosing location(s) for a process, or relocate it to a location on which there are no update processes available are two ways of preventing future updates.

and by Theorem 21 we infer that

$$\Gamma\,;\,\Theta \vdash \mathsf{Sys}_2 \triangleright \emptyset; \mathcal{I}_1$$

5.2. Upgrading behavior and distributed structure

Suppose now that R stands for the following adaptation routine for S:

$$\begin{array}{lll} R & \triangleq & r\{(\mathsf{X}).r_1[S_{\mathrm{wra}}] \mid r_2[S_{\mathrm{rem}}]\} \quad \text{where:} \\ S_{\mathrm{wra}} & \triangleq & !\,\mathsf{accept}\, a(x).\mathsf{request}\, b(y).y\langle\langle x\rangle\rangle.\mathsf{close}\,(y) \\ S_{\mathrm{rem}} & \triangleq & !\,\mathsf{accept}\, b(z).z((x)).x(u,p).x \rhd \{n_1{:}Q_1.\mathsf{close}\,(x).\mathsf{close}\,(z) \,[\!]\, n_2{:}Q_2.\mathsf{close}\,(x).\mathsf{close}\,(z)\} \end{array}$$

Above, S_{wra} represents a service wrapper which, deployed at r_1, acts as a mediator: it redirects all client requests on name a to the remote service S_{rem}, defined on name b and deployed at r_2. Although simple, this service structure is quite appealing: by exploiting session delegation, it hides from clients certain implementation details (e.g., name b and location r_2), therefore simplifying future reconfiguration tasks—in fact, clients do not need to know about b to execute, and so S_{wra} can be transparently upgraded. This new definition for R illustrates an update process that may reconfigure both the behavior and distributed structure of S: in a single step, the monolithic service S is replaced by a more flexible distributed implementation based on S_{wra} and S_{rem} and deployed at r_1 and r_2 (rather than at r). As we discuss below, R above does not involve process variables, and so the current behavior at r is discarded. Using our type system, we may infer that

$$\Gamma\,;\,\Theta \vdash \mathsf{Sys} \triangleright \emptyset; \mathcal{I}_1 \quad \text{where:}$$

- $\{a : \langle \alpha_{\mathsf{un}}, \overline{\alpha}_{\mathsf{lin}}\rangle,\ b : \langle !(\alpha)_{\mathsf{lin}}, ?(\alpha)_{\mathsf{un}}\rangle\} \subseteq \Gamma$
- $\{l_1 \mapsto a : \overline{\alpha}_{\mathsf{lin}},\ l_2 \mapsto a : \alpha_{\mathsf{un}} \cup b : !(\alpha)_{\mathsf{un}} \cup b : ?(\alpha)_{\mathsf{un}},\ r \mapsto a : \alpha_{\mathsf{un}},\ r_1 \mapsto \alpha_{\mathsf{un}} \cup b : !(\alpha)_{\mathsf{un}},\ r_2 \mapsto b : ?(\alpha)_{\mathsf{un}}\} \subseteq \Theta$
- $\mathcal{I}_1 = \{\mathsf{lin}\, a : \overline{\alpha},\ \mathsf{un}\, a : \alpha\}$

and where we have slightly simplified notation for Θ, for readability reasons.

As before, our typed framework ensures that consistent updates may only take place before a session related to the service on a is established. Suppose such an action occurs as the first action of Sys (i.e. the definition of service S is immediately updated):

$$\mathsf{Sys} \longrightarrow l_1^0[C_1] \mid l_2^0[r_1^0[S_{\mathrm{wra}}] \mid r_2^0[S_{\mathrm{rem}}]] = \mathsf{Sys}'$$

Because R declares no process variables, this step formalizes an update operation which *discards* the behavior located at r (i.e., $a : \alpha_{\mathsf{un}}$). To understand why this reconfiguration step is safe, it is crucial to observe that:

(i) the new service implementation, based on S_{wra} and S_{rem}, respects the prescribed interfaces of the involved locations (i.e., r_1 and r_2, not used in Sys);
(ii) the interface of location l_2 (which hosts the server implementation) can indeed contain the two services on name b implemented by S_{wra} and S_{rem}.

These two important conditions are statically checked by our typing system. After the reduction it is easy to see that

$$\Gamma\,;\,\Theta \vdash \mathsf{Sys}' \triangleright \emptyset; \mathcal{I}_2$$

where Γ, Θ are as above and $\mathcal{I}_2 = \{\mathsf{lin}\, a : \overline{\alpha},\ \mathsf{un}\, a : \alpha,\ \mathsf{un}\, b : ?(\alpha),\ \mathsf{un}\, b : !(\alpha)\}$. We have $\mathcal{I}_1 \sqsubseteq \mathcal{I}_2$: indeed the interface grows as the updated service now relies on two service definitions (on names a, b) rather than on a single definition.

6. Extensions and enhancements

This section discusses two possible extensions for our framework. The first one concerns the runtime adaptation of processes with active (running) sessions, while the second one concerns the inclusion of recursion and subtyping constructs. In both cases, concrete details on the technical machinery required are given, and the challenges involved are highlighted.

6.1. Runtime adaptation of processes with active sessions

Up to here, our notion of runtime adaptation concerns located processes with no active sessions. As already motivated, our intention is to rule careless update actions which may affect the session protocols implemented on such locations. Here we discuss generalizations of our framework so as to admit the runtime adaptation of located processes containing active sessions. As before, the goal will be to ensure that session communications are both safe and consistent.

To illustrate this point, consider process Sys′, discussed in the Introduction:

$$\begin{aligned}\mathsf{Sys}' = (\nu\kappa)(l_1[\kappa^+\langle \mathsf{u}_1, \mathsf{p}_1\rangle.\kappa^+ \lhd n_1.P_1.\mathsf{close}\,(\kappa^+)] \mid \\ l_2[r[\kappa^-(u,p).\kappa^- \rhd \{n_1{:}Q_1.\mathsf{close}\,(\kappa^-) \,[\!]\, n_2{:}Q_2.\mathsf{close}\,(\kappa^-)\}\,] \mid R\,])\end{aligned}$$

Focusing on location r, suppose that $R = r\{(\mathsf{X}).Q_\mathsf{X}\}$. There are at least two ways in which Q_X can implement a consistent update on r:

(a) Q_X *preserves* the behavior at r: Intuitively, this means that X occurs linearly (exactly once) in Q_X. This way, Q_X may implement a relocation (as in, e.g., $Q_\mathsf{X} = l'[\mathsf{X}]$, for some different location l') or it may place the behavior at r in a richer context (as in, e.g., $Q_\mathsf{X} = r[\mathsf{X} \mid R']$ in which the behavior at location r is *extended* with process R').
(b) Q_X *upgrades* the behavior at P: This is the case when, e.g., $\mathsf{X} \notin \mathsf{fpv}(Q_\mathsf{X})$. In order to ensure consistency, besides ensuring a compatible interface, the new behavior Q_X should implement all open sessions at r (namely κ^-, κ^+ above). Therefore, this possibility implies having precise information on the protocols implemented at r, for Q_X must continue with such protocols.

Next we separately consider each of these two alternatives.

6.1.1. Typing preserving updates

As mentioned above, the key issue in this class of updates is to ensure *linearity* of the processes variable. Given $l\{(\mathsf{X}).Q_\mathsf{X}\}$, we need to guarantee that $\mathsf{X} \in \mathsf{fpv}(Q_\mathsf{X})$ (to avoid discarding the behavior at l) but also that X occurs exactly once, for duplicating behaviors would be unsound. A first, but somewhat drastic, way of ensuring linearity would be by adding syntactic restrictions on the shape of update contexts (such as Q_X). We have formalized this alternative is in [3, §2.1.2], where behavioral characterizations of update processes are thoroughly analyzed.

Alternatively, we could exploit the type system, using the information in Θ to ensure linearity. This would require changing rule (T:Nil) so that $\mathbf{0}$ can only be typed in a higher-order environment that contains no process variables. Also, one would need to refine rule (T:Par) so to ensure that process variables are properly split. More precisely, we would need the following modified rules:

$$(\text{T:LNil})\ \frac{vdom(\Theta) = \emptyset}{\Gamma\,;\,\Theta \vdash \mathbf{0} \triangleright \emptyset;\emptyset}$$

$$(\text{T:LLoc})\ \frac{\Theta \vdash l : \mathcal{I} \qquad \Gamma\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I}' \qquad \mathcal{I}' \sqsubseteq \mathcal{I}}{\Gamma\,;\,\Theta \vdash l[P] \triangleright \Delta;\mathcal{I}'}$$

$$(\text{T:LPar})\ \frac{\Gamma\,;\,\Theta_1 \vdash P \triangleright \Delta_1;\mathcal{I}_1 \qquad \Gamma\,;\,\Theta_2 \vdash Q \triangleright \Delta_2;\mathcal{I}_2 \qquad \Theta = \Theta_1 \circ \Theta_2}{\Gamma\,;\,\Theta \vdash P \mid Q \triangleright \Delta_1 \cup \Delta_2;\mathcal{I}_1 \uplus \mathcal{I}_2}$$

where, in (T:LPar), the splitting $\Theta_1 \circ \Theta_2$ is defined if and only if $\Theta_1 \cap \Theta_2 = \emptyset$ and $vdom(\Theta_1) \cap vdom(\Theta_2) = \emptyset$. Observe that the interplay of these two rules suffices to guarantee linearity of process variables. Indeed, rule (T:LPar) ensures that variable X can be used in at most one subprocess in parallel, whereas rule (T:LNil) assures that it is used at least one. Notice also that we keep rule (T:Adapt) as in Table 5: its right-hand side typing ensures that the context does not introduce new open sessions.

With these changes in the typing system, the runtime annotation on the number of active sessions (occurring in located process) can be removed from the reduction semantics. The modified semantics can be found in Appendix C (Table C.9).

6.1.2. Typing runtime upgrades

We now generalize the mechanism in the previous section to include the *runtime upgrade* of a process $l[P]$ with a process Q that in particular provides an alternative implementation for the active protocols in P. As explained earlier, handling an upgrade entails having precise knowledge on the protocols running in the location. More precisely, a main challenge is to find a way of describing compatibility between the (non-bracketed) endpoints in P with those in Q. We now detail a possible solution to these issues, based on instrumenting the reduction semantics in Section 2.2 with typing environments. Let us consider *typed reductions* of the form:

$$\Gamma\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I} \longrightarrow \Gamma\,;\,\Theta \vdash P' \triangleright \Delta';\mathcal{I}'$$

thus defining how a well-typed process P (and their associated typing and interface) evolve as a result of an internal computation. We now discuss some selected rules for this typed semantics, given in Table 7; the complete set of rules can be found in Appendix C (Tables C.10 and C.11).

Main differences with respect to the semantics of Table 2 are: (i) runtime annotations on locations are no longer needed, and (ii) rule (R:UpdU) checks session consistency by appealing to appropriate typings (denoted Δ_1 and Δ_2 in the rule). More precisely, as runtime annotations are not considered, typed reduction rules are simpler than untyped ones. Notice how

Table 7
Typed reduction semantics (selected rules).

$$(\text{R:ParU})\quad \frac{\Gamma\,;\,\Theta \vdash P \triangleright \Delta_1;\mathcal{I}_1 \longrightarrow \Gamma\,;\,\Theta \vdash P' \triangleright \Delta_1';\mathcal{I}_1'}{\Gamma\,;\,\Theta \vdash P \mid Q \triangleright \Delta_1 \cup \Delta_2;\mathcal{I}_1 \cup \mathcal{I}_2 \longrightarrow \Gamma\,;\,\Theta \vdash P' \mid Q \triangleright \Delta_1' \cup \Delta_2;\mathcal{I}_1' \uplus \mathcal{I}_2'}$$

$$(\text{R:LocU})\quad \frac{\Gamma\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I} \longrightarrow \Gamma\,;\,\Theta \vdash P' \triangleright \Delta';\mathcal{I}'}{\Gamma\,;\,\Theta \vdash l[P] \triangleright \Delta;\mathcal{I} \longrightarrow \Gamma\,;\,\Theta \vdash l[P'] \triangleright \Delta';\mathcal{I}'}$$

$$(\text{R:UpdU})\quad \frac{\Gamma\,;\,\Theta \vdash P \triangleright \Delta_1;\mathcal{I}_1 \qquad \Gamma\,;\,\Theta, \mathsf{X} : \Delta_1, \mathcal{I}_1 \vdash Q \triangleright \Delta_2;\mathcal{I}_2 \qquad \Delta_1 = \rho(\Delta_2)}{\begin{array}{c}\Gamma\,;\,\Theta \vdash C\{l[P]\} \mid D\{l[(X).Q\}] \triangleright \Delta;\mathcal{I} \\ \longrightarrow \\ \Gamma\,;\,\Theta \vdash C\{\rho(Q)[^{P}/_{\mathsf{X}}]\} \mid D\{\mathbf{0}\} \triangleright \Delta;(\mathcal{I} \setminus \mathcal{I}_1) \uplus \mathcal{I}_2\end{array}}$$

$$(\text{R:OpenU})\quad \frac{\begin{array}{c}\Gamma\,;\,\Theta \vdash C\{\mathsf{accept}\, a(x).P\} \mid D\{\mathsf{request}\, a(y).Q\} \triangleright \Delta;\mathcal{I}, a : \alpha_{\mathsf{lin}}, a : \overline{\alpha}_{\mathsf{lin}} \\ \longrightarrow \\ \Gamma\,;\,\Theta \vdash (\nu\kappa)C\{P[^{\kappa^+}/_{X}]\} \mid D\{Q[^{\kappa^-}/_{y}]\} \triangleright \Delta, [\kappa^+ : \alpha], [\kappa^- : \overline{\alpha}];\mathcal{I}\end{array}}{}$$

rule (R:LocU) allows us to infer reductions with a single location; in general, given a context C (as in Definition 2) and a process P which may reduce, a corresponding typed reduction for process $C\{P\}$ can be inferred by combining rules (R:LocU) and (R:ParU). Rule (R:UpdU) concerns the update of a located process P with a context Q. A typed update reduction will depend on their associated typings, denoted Δ_1 and Δ_2, respectively. Intuitively, Δ_1 and Δ_2 should be identical, up to a substitution ρ from channel variables $x_1, \ldots, x_m$ in Δ_2 to non-bracketed channels $\kappa_1^p, \ldots, \kappa_m^p$ in Δ_1. Substitution ρ works then as an adaptor; to highlight its role, in rule (R:UpdU) we write $\rho(\Delta)$ and $\rho(P)$ to denote the application of ρ to typing Δ and process P; the formal definition of these notations is as expected. This is how endpoint compatibility between P and Q is enforced. Provided a suitable ρ exists, the upgrade can take place and $l[P]$ is substituted with $\rho(Q)[^{P}/_{\mathsf{X}}]$.

For the system with the typed reduction semantics, we require the typing rules in Tables 5 and 6, replacing rules (T:Pvar), (T:Adapt) and (T:Loc) with rules (T:PVarU), (T:AdaptU) and (T:LocU) below:

$$(\text{T:PVarU})\ \frac{}{\Gamma;\Theta, \mathsf{X} : \Delta;\mathcal{I} \vdash \mathsf{X} : \Delta;\mathcal{I}}$$

$$(\text{T:AdaptU})\ \frac{\Theta \vdash l : \mathcal{I} \qquad \Gamma\,;\,\Theta, \mathsf{X} : \emptyset;\mathcal{I} \vdash P \triangleright \Delta;\mathcal{I}}{\Gamma\,;\,\Theta \vdash l\{(\mathsf{X}).P\} \triangleright \emptyset;\emptyset}$$

$$(\text{T:LocU})\ \frac{\Theta \vdash l : \mathcal{I} \qquad \Gamma\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I}' \qquad \mathcal{I}' \sqsubseteq \mathcal{I}}{\Gamma\,;\,\Theta \vdash l[P] \triangleright \Delta;\mathcal{I}'}$$

Intuitively, as made explicit by rule (T:PVarU), process variables must now record both a typing Δ and an interface $\mathcal{I}$. This refines the intrinsic meaning of an update operation, as there is an explicit reference to required open sessions. Based on this enhancement, rule (T:AdaptU) is a variant of rule (T:Adapt) in which the process variable occurs annotated with its typing and interface and where we admit a non-empty typing Δ, thus allowing process P to introduce active sessions. Finally, rule (T:LocU) simplifies rule (T:Loc) by eliminating runtime annotations. We are confident that session processes in this modified framework also enjoy our safety and consistency results (Theorems 21, 22, and 25) with little modifications.

6.2. Adding recursive types and subtyping

This section discusses the extension of our approach with *recursive types* and *subtyping*. These are two well-known ingredients of session type theories: while recursion is present in early papers in binary session types [20], subtyping was first studied by Gay and Hole in [17]. Notice that by extending the types in Table 3 with recursive types we obtain a type syntax that coincides with that in [20,29], and is quite similar to that in [17]. As such, the incorporation of both recursive types and subtyping closely follows prior works, and entails unsurprising technical details. Still, the extension is interesting: on the one hand, recursive types increase the expressiveness of our language; on the other, subtyping allows us to refine the notion of interface (and interface ordering), thus enhancing our typed constructs for runtime adaptation.

6.2.1. Extended process and type syntax

We begin by re-defining the process language. Consider an additional base set of *recursion variables*, ranged over by $\mathcal{Y}, \mathcal{Y}', \ldots$. In essence, we consider the language in Table 1 without replicated session acceptance and with the addition of *recursive calls*, denoted $\mathsf{rec}\,\mathcal{Y}.P$:

Table 8
Well-typed processes with recursion and subtyping: new and/or modified rules.

$$\frac{}{\Gamma;\Theta,\mathcal{Y}:\Delta,\mathcal{I}\vdash\mathcal{Y}:\Delta;\mathcal{I}}\ (\text{T:RVar})$$

$$\frac{\Gamma;\Theta,\mathcal{Y}:\Delta;\mathcal{I}\vdash P\triangleright\Delta;\mathcal{I}}{\Gamma;\Theta\vdash \mathsf{rec}\,\mathcal{Y}.P\triangleright\Delta;\mathcal{I}}\ (\text{T:Rec})$$

$$\frac{\Gamma;\Theta\vdash P\triangleright\Delta;\mathcal{I}\quad \Delta\leq_{\mathrm{C}}\Delta'\quad \mathcal{I}\leq_{\mathrm{C}}\mathcal{I}'}{\Gamma;\Theta\vdash P\triangleright\Delta';\mathcal{I}'}\ (\text{T:Subs})$$

$$\frac{\Gamma;\Theta\vdash P\triangleright\Delta,\kappa^-:\alpha_1,\kappa^+:\alpha_2;\mathcal{I}\quad \alpha_1\perp_{\mathrm{C}}\alpha_2}{\Gamma;\Theta\vdash(\nu\kappa)P\triangleright\Delta,[\kappa^-:\alpha_1],[\kappa^+:\alpha_2];\mathcal{I}}\ (\text{T:CResD})$$

$$\frac{\Theta\vdash l:\mathcal{I}_1\qquad \Gamma;\Theta,\mathsf{X}:\mathcal{I}_1\vdash P\triangleright\emptyset;\mathcal{I}_2\qquad \mathcal{I}_1\sqsubseteq\mathcal{I}_2}{\Gamma;\Theta\vdash l\{(\mathsf{X}).P\}\triangleright\emptyset;\emptyset}\ (\text{T:MonAdapt})$$

$$\begin{aligned}P,Q,R ::= \;& \mathsf{request}\,a(x).P \mid \mathsf{accept}\,a(x).P\\ \mid\;& k\langle\tilde{e}\rangle.P \mid k(\tilde{x}).P \mid k\langle\langle k'\rangle\rangle.P \mid k((x)).P\\ \mid\;& k\lhd n;P \mid k\rhd\{n_1{:}P_1\,[\!]\cdots[\!]\,n_m{:}P_m\} \mid \mathsf{close}\,(k).P \mid (\nu k)P\\ \mid\;& l[P] \mid l\{(\mathsf{X}).P\} \mid \mathsf{X} \mid \mathsf{rec}\,\mathcal{Y}.P \mid \mathcal{Y}\\ \mid\;& P\mid P \mid \mathsf{if}\,e\,\mathsf{then}\,P\,\mathsf{else}\,Q \mid \mathbf{0}\end{aligned}$$

Observe how a different font style distinguishes process variables (used in update processes) from recursion variables. Notions of binding, α-conversion, and substitution for recursion variables are completely standard; given a process P, we write $\mathsf{frv}(P)$ and $\mathsf{brv}(P)$ to denote its sets of free/bound recursion variables.

The operational semantics for the modified language requires minor modifications. Notions of structural congruence (Definition 1), contexts (Definition 2), and operations over contexts (Definition 3) are kept unchanged. The reduction semantics is the smallest relation on processes generated by the rules in Table 2, excepting rule (R:ROpen) and adding the following additional rule:

$$(\text{R:Rec})\quad C\{\mathsf{rec}\,\mathcal{Y}.P\}\longrightarrow C\{P[^{\mathsf{rec}\,\mathcal{Y}.P}/_{\mathcal{Y}}]\}$$

As for the type syntax, the only addition are recursive types. Let us write $t,t',\ldots$ to denote recursive type variables. The extended syntax for types is then as follows:

$$\begin{aligned}\alpha,\beta ::=\;& t && \text{type variable}\\ \mid\;& \mu t.\alpha && \text{recursive type}\\ \mid\;& \cdots && \{\text{the other type constructs, as in Table 3}\}\end{aligned}$$

As customary, we adopt an *equi-recursive* approach to recursive types, not distinguishing between $\mu t.\alpha$ and its unfolding $\alpha[^{\mu t.\alpha}/_t]$. We restrict to *contractive types*: a type is contractive if for each of its sub-expressions $\mu t.\mu t_1.\ldots.\mu t_n.\alpha$, the body α is not t. Concerning typing environments, we assume that recursion variables are included in the higher-order environment Θ, thus extending the definition given in Table 3 (lower part). Finally, we shall require two typing rules for recursion variables and recursive calls; these are the first two rules in Table 8.

An example. As a simple illustration, consider a typical client/server scenario (*Client* | *Server*) realized by the processes below:

$$\begin{aligned}\mathit{Server} &:= \mathsf{rec}\,\mathcal{Z}.\mathsf{accept}\,a(x).(\mathcal{Z}\mid\mathsf{rec}\,\mathcal{Y}.x\rhd\{n_1{:}x(v).S_{\mathcal{Y}}\,[\!]\,n_2{:}\mathsf{close}\,(x)\})\\ S_{\mathcal{Y}} &:= \mathsf{request}\,b(z).z\langle v\rangle.z(r).\mathsf{close}\,(z).x\langle r\rangle.\mathcal{Y}\\ \mathit{Client} &:= \mathsf{request}\,a(x).x\lhd n_1.x\langle d_1\rangle.x(r).x\lhd n_1.x\langle d_2\rangle.x(r).x\lhd n_2.\mathsf{close}\,(x)\end{aligned}$$

We use recursion to implement (i) persistent services and (ii) services with recursive behaviors. Process *Server* shows how to encode a behavior similar to $!\,\mathsf{accept}\,a(x)$: indeed once a session on a is established a new copy of *Server* is spawned. The body of the session has a recursive behavior: the client can choose between doing some computation on the server or closing the communication. If she chooses the first branch, then she sends some data to the server, the server processes them and sends back an answer r. At this point the client can choose again what to do: another computation or ending the session.

Processes *Server* and *Client* are well-typed under environments $\Gamma := v : \tau_1, r : \tau_2$ and $\Theta = \emptyset$. Indeed we can obtain the following type judgments:

$$\Gamma\,;\ \Theta \vdash \mathit{Server} \triangleright \emptyset; \mathsf{un}\, a : \mu t.\&\{n_1{:}?(\tau_1).!(\tau_2).t, n_2{:}\epsilon\}, \mathsf{un}\, b : \&\{n_1{:}!(\tau_1).?(\tau_2).\epsilon$$
$$\Gamma\,;\ \Theta \vdash \mathit{Client} \triangleright \emptyset; \mathcal{I}$$

where $\mathcal{I} = \mathsf{un}\, a : \oplus\{n_1{:}!(\tau_1).?(\tau_2). \oplus \{n_1{:}!(\tau_1).?(\tau_2). \oplus \{n_1{:}!(\tau_1).?(\tau_2).\epsilon, n_2{:}\epsilon\}, n_2{:}\epsilon\}, n_2{:}\epsilon\}$.

6.2.2. Coinductive subtyping and duality

We now introduce subtyping and duality, by adapting notions and definitions from [17]. Intuitively, given session types α, β, we shall say that α is a *subtype* of β if any process of type α can safely be used in a context where a process of type β is expected. More formally, let us write $\mathcal{T}$ to refer to the set of types, including both session types $\alpha, \beta, \ldots$ and base types $\tau, \sigma, \ldots$. We shall write $T, S, \ldots$ to range over $\mathcal{T}$. For all types, define $\mathbf{unfold}(T)$ by recursion on the structure of T: $\mathbf{unfold}(\mu t.T) = \mathbf{unfold}(T[^{\mu t.T}/_t])$ and $\mathbf{unfold}(T) = T$ otherwise. Our definition of coinductive subtyping is given next; it relies on a subtyping relation $\leq_{\mathrm{B}}$ over base types, which arises from subset relations (as in, e.g., $\mathsf{int} \leq_{\mathrm{B}} \mathsf{real}$).

Definition 26. A relation $\mathcal{R} \subseteq \mathcal{T} \times \mathcal{T}$ is a *type simulation* if $(T, S) \in \mathcal{R}$ implies the following conditions:

1. If $\mathbf{unfold}(T) = \tau$ then $\mathbf{unfold}(S) = \sigma$ and $\tau \leq_{\mathrm{B}} \sigma$.
2. If $\mathbf{unfold}(T) = \epsilon$ then $\mathbf{unfold}(S) = \epsilon$.
3. If $\mathbf{unfold}(T) = ?(T_2).T_1$ then $\mathbf{unfold}(S) = ?(S_2).S_1$ and $(T_1, S_1) \in \mathcal{R}$ and $(T_2, S_2) \in \mathcal{R}$.
4. If $\mathbf{unfold}(T) = !(T_2).T_1$ then $\mathbf{unfold}(S) = !(S_2).S_1$ and $(T_1, S_1) \in \mathcal{R}$ and $(S_2, T_2) \in \mathcal{R}$.
5. If $\mathbf{unfold}(T) = ?(\tau_1, \ldots, \tau_n).T_1$ then $\mathbf{unfold}(S) = ?(\sigma_1, \ldots, \sigma_n).S_1$ then for all $i \in [1..n]$, we have that $(\tau_i, \sigma_i) \in \mathcal{R}$ and $(T_1, S_1) \in \mathcal{R}$.
6. If $\mathbf{unfold}(T) = !(\tau_1, \ldots, \tau_n).T_1$ then $\mathbf{unfold}(S) = !(\sigma_1, \ldots, \sigma_n).S_1$ then for all $i \in [1..n]$, we have that $(\sigma_i, \tau_i) \in \mathcal{R}$ and $(T_1, S_1) \in \mathcal{R}$.
7. If $\mathbf{unfold}(T) = \&\{n_1 : T_1, \ldots, n_m : T_m\}$ then $\mathbf{unfold}(S) = \&\{n_1 : S_1, \ldots, n_h : S_h\}$ and $m \leq h$ for all $i \in [1..m]$, we have that $(T_i, S_i) \in \mathcal{R}$.
8. If $\mathbf{unfold}(T) = \oplus\{n_1 : T_1, \ldots, n_m : T_m\}$ then $\mathbf{unfold}(S) = \oplus\{n_1 : S_1, \ldots, n_h : S_h\}$ and $h \leq m$ for all $i \in [1..m]$, we have that $(T_i, S_i) \in \mathcal{R}$.

Observe how $\preceq$ is co-variant for input prefixes and contra-variant for outputs, whereas it is co-variant for branching and contra-variant for choices. We have the following definition:

Definition 27. The coinductive subtyping relation, denoted $\leq_{\mathrm{C}}$, is defined by $T \leq_{\mathrm{C}} S$ if and only if there exists a type simulation $\mathcal{R}$ such that $(T, S) \in \mathcal{R}$.

Following standard arguments (as in, e.g., [17]), one may show:

Lemma 28. *$\leq_{\mathrm{C}}$ is a preorder.*

We now move on to define duality. It is known that an inductive definition of duality (cf. Definition 5) is no longer appropriate in the presence of recursive types (see, e.g., [28]). This justifies the need for a *coinductive* notion of duality over session types; it is defined similarly as subtyping above.

Definition 29. A relation $\mathcal{R} \subseteq \mathcal{T} \times \mathcal{T}$ is a *duality relation* if $(T, S) \in \mathcal{R}$ implies the following conditions:

1. If $\mathbf{unfold}(T) = \tau$ then $\mathbf{unfold}(S) = \sigma$ and $\tau \leq_{\mathrm{C}} \sigma$ and $\sigma \leq_{\mathrm{C}} \tau$.
2. If $\mathbf{unfold}(T) = \epsilon$ then $\mathbf{unfold}(S) = \epsilon$.
3. If $\mathbf{unfold}(T) = ?(T_2).T_1$ then $\mathbf{unfold}(S) = !(S_2).S_1$ and $(T_1, S_1) \in \mathcal{R}$ and $T_2 \leq_{\mathrm{C}} S_2$ and $S_2 \leq_{\mathrm{C}} T_2$.
4. If $\mathbf{unfold}(T) = !(T_2).T_1$ then $\mathbf{unfold}(S) = ?(S_2).S_1$ and $(T_1, S_1) \in \mathcal{R}$ and $T_2 \leq_{\mathrm{C}} S_2$ and $S_2 \leq_{\mathrm{C}} T_2$.
5. If $\mathbf{unfold}(T) = ?(\tau_1, \ldots, \tau_n).T_1$ then $\mathbf{unfold}(S) = ?(\sigma_1, \ldots, \sigma_n).S_1$ then for all $i \in [1..n]$, we have that $(T_1, S_1) \in \mathcal{R}$ and $\tau_i \leq_{\mathrm{C}} \sigma_i$ and $\sigma_i \leq_{\mathrm{C}} \tau_i$.
6. If $\mathbf{unfold}(T) = !(\tau_1, \ldots, \tau_n).T_1$ then $\mathbf{unfold}(S) = ?(\sigma_1, \ldots, \sigma_n).S_1$ then for all $i \in [1..n]$, we have that $(T_1, S_1) \in \mathcal{R}$ and $\tau_i \leq_{\mathrm{C}} \sigma_i$ and $\sigma_i \leq_{\mathrm{C}} \tau_i$.
7. If $\mathbf{unfold}(T) = \&\{n_1 : T_1, \ldots, n_m : T_m\}$ then $\mathbf{unfold}(S) = \oplus\{n_1 : S_1, \ldots, n_m : S_m\}$ and for all $i \in [1..m]$, we have that $(T_i, S_i) \in \mathcal{R}$.
8. If $\mathbf{unfold}(T) = \oplus\{n_1 : T_1, \ldots, n_m : T_m\}$ then $\mathbf{unfold}(S) = \&\{n_1 : S_1, \ldots, n_m : S_m\}$ and for all $i \in [1..m]$, we have that $(T_i, S_i) \in \mathcal{R}$.

We may now define:

Definition 30. The coinductive duality relation, denoted $\perp_{\mathrm{C}}$, is defined by $T \perp_{\mathrm{C}} S$ if and only if there exists a duality relation $\mathcal{R}$ such that $(T, S) \in \mathcal{R}$. The extension of $\leq_{\mathrm{C}}$ to typings and interfaces, written $\Delta \leq_{\mathrm{C}} \Delta'$ and $\mathcal{I} \leq_{\mathrm{C}} \mathcal{I}'$, respectively, arise as expected.

6.2.3. Refining interfaces via subtyping

In most session type disciplines, the main use of duality typically arises in the rule for channel restriction; similarly, the main use of (coinductive) subtyping is in the subsumption rule, which enables us to incorporate the flexibility of subtyping in derivations, increasing typability. Table 8 presents these rules for our framework, denoted (T:CRESD) and (T:SUBS), respectively. While the former represents the key duality check performed by the typing system, the latter covers both typing Δ and interface $\mathcal{I}$. In the following we elaborate on another consequence of adding subtyping, namely its interplay with interface ordering (cf. Definition 8).

As argued along the paper, a main contribution of this work is the extension of session type disciplines with a simple notion of *interface*. Using interfaces, we are able to give simple and intuitive typing rules for located and update processes—see rules (T:LOC) and (T:ADAPT) in Table 5. It is thus legitimate to investigate how to enhance the notion of interface and its associated definitions. In particular, we discuss an alternative based on subtyping. Consider rule (T:MONADAPT) in Table 8: it is intended as an alternative formulation for typing rule (T:ADAPT) in Table 5. Although rule (T:MONADAPT) is more restrictive than (T:ADAPT) (i.e., it accepts less update processes as typable), it captures a requirement that may be desirable in several practical settings, namely that the behavior after adaptation is "at least" the behavior offered before, possibly adding new behaviors. Indeed, by disallowing adaptation routines that discard behavior, rule (T:MONADAPT) is suitable to reason about settings in which adaptation/upgrade actions need to be tightly controlled.

In the context of more stringent typing rules such as (T:MONADAPT), it is convenient to find ways for adding flexibility to the interface preorder $\sqsubseteq$ in Definition 8. As this preorder is central to our approach for disciplined runtime adaptation, a relaxed definition for it may lead to more flexible typing disciplines. One alternative is to rely on $\leq_{\mathrm{C}}$ for such a relaxed definition:

Definition 31 *(Refined interface preorder).* Given interfaces $\mathcal{I}$ and $\mathcal{I}'$, we write $\mathcal{I} \sqsubseteq_{\mathrm{sub}} \mathcal{I}'$ iff

1. $\forall(\mathsf{lin}\, a{:}\alpha)$ such that $\#_{\mathcal{I}_{\mathsf{lin}}}(\mathsf{lin}\, a{:}\alpha) = h$ with $h > 0$, then one of the following holds:
   (a) there exists h distinct elements $(\mathsf{lin}\, a{:}\beta_i) \in \mathcal{I}'_{\mathsf{lin}}$ such that $\alpha \leq_{\mathrm{C}} \beta_i$ for $i \in [1..h]$;
   (b) there exists $(\mathsf{un}\, a{:}\beta) \in \mathcal{I}'_{\mathsf{un}}$ such that $\alpha \leq_{\mathrm{C}} \beta$.
2. $\forall(\mathsf{un}\, a{:}\alpha) \in \mathcal{I}_{\mathsf{un}}$ then $(\mathsf{un}\, a{:}\beta) \in \mathcal{I}'_{\mathsf{un}}$ and $\alpha \leq_{\mathrm{C}} \beta$, for some β.

It is immediate to see how $\sqsubseteq_{\mathrm{sub}}$ improves over $\sqsubseteq$ by offering a more flexible and fine-grained relation over interfaces, in which subtyping replaces strict type equality.

We are now finally ready to state the notion of well-typedness that concerns the processes introduced in this section:

Definition 32. A (possibly recursive) process is well-typed if it can be typed using the rules in Table 5 (excepting (T:CRES) and (T:ADAPT)), Table 6, and Table 8.

We are confident that our main results (Theorems 21, 22, and 25) also hold, with minor modifications, for the enhanced framework that results from: (a) replacing replication with recursion, using coinductive duality, and (b) incorporating subtyping, also replacing (T:ADAPT) with rule (T:MONADAPT) above, using $\sqsubseteq_{\mathrm{sub}}$ in place of $\sqsubseteq$ in the appropriate places in Table 5.

7. Related work

Adaptation in structured communications. Binary session types were first introduced in [19,20]. They have been the subject of intense research in the last two decades, and many developments have followed. Two notable such developments concern asynchronous communications (see, e.g., [22]) and multiparty communications (see, e.g., [21], respectively). To the best of our knowledge, our paper [14] was the first work to incorporate constructs for runtime adaptation (or evolvability) into a session process language (either binary or multiparty). As stated in the Introduction, the present paper is an extended, revised version of [14]. In particular, in this presentation the operational semantics and typing system of [14] have been much simplified. Following [16], the operational semantics in [14] was instrumented with several elements to support static analysis. For instance, one such elements is a local association between names and session types; such an association is explicitly tracked by a type annotation in prefixes for session acceptance and request. In contrast, for the sake of clarity, the semantics given here relies only on a simple runtime annotation for located processes; the other elements are now subsumed by the (revised) typing discipline. Also, to highlight on the simplicity and novelty of our approach, in this presentation we focus on replicated processes, rather than on recursion (the form of infinite behavior given in [14]). This is an issue orthogonal to our approach, as discussed in Section 6.

After our paper [14] appeared, some works have addressed the challenging issue of analyzing runtime adaptation in scenarios of *multiparty* communications, in which protocols involve more than two partners. In such settings, there is a global specification (or *choreography*) to which all interacting partners, from their local perspectives, should adhere. In the setting of multiparty session types (see, e.g., [21]), these two visions are described by a global type and by local types, respectively; there is a *projection function* which formally relates the two. The work [2] defines both global and local languages for choreographies in which adaptation is represented by generalized forms of the adaptable processes in [3]. It is then fair to say that [2] defines the foundations for extending the present approach to a multiparty setting. The recent work [11] proposes a model of self-adaptation for multiparty sessions. Key technical novelties in this approach are *monitors* (obtained via projection of global types) and *adaptation functions*. Monitors are coupled to processes and govern their behavior; together with global types, adaptation functions embody the runtime adaptation strategy. An associated typed system ensures communication safety and progress.

Adaptation and higher-order process communication. Our constructs for runtime adaptation (inherited from [3]) can be seen as a particular form of *higher-order* communication (or *process-passing* [26]). In fact, as explained in Section 2.2, our notion of runtime adaptation formally relies on the objective *movement* of an adaptation routine to a designated location. Session type disciplines for a higher-order π-calculus have been studied in [25]. Besides data and session names, session communications in [25] may also involve code (i.e., process terms), leading to succinct and flexible protocol descriptions. The proposed typing system ensures the disciplined use of mobile code exchanged in communications; in particular, to avoid communication errors due to unmatched or uncompleted sessions, typing ensures that mobile code containing session endpoints is run exactly once. In contrast to the process framework in [25], our model has a rather implicit higher-order character, for we do not allow process communication within sessions. Relating our process language with the model in [25] is a promising topic for future work. Our constructs for runtime adaptation are also related to *passivation* operators found in higher-order process calculi (see, e.g., [23]). There are significant differences between adaptable processes and passivation; in particular, adaptable processes distinguish from calculi with passivation in that process update is defined without assuming constructs for higher-order process communication. See [3, §9.1] for a detailed comparison between adaptable processes and passivation.

As already discussed, our approach has been influenced by [16] and our previous work [3]. Nevertheless, there are several major differences between our framework and those works. (i) Unlike the system in [16], our framework supports channel passing (delegation). As delegation is already useful to represent forms of dynamic reconfiguration, its interplay with update actions is very appealing (cf. the example in Section 5.2). (ii) We have extended typing judgments in [16] with *interfaces* $\mathcal{I}$, which are central to characterize located processes which can be safely updated. (iii) While in [3] adaptable processes are defined for a fragment of CCS, here we consider them within a typed π-calculus. (iv) Adaptation steps in [3] are completely unrestricted. Here we have considered annotated versions of the constructs in [3]: they offer a more realistic account of update actions, as they are supported by runtime conditions based on session types.

Runtime adaptation. Recently, there has been an increasing interest in the specification and verification of autonomic and (self-)adaptive computer systems. As a result, several definitions and characterizations of (self-)adaptation have been put forward: they reflect the different perspectives and approaches of the several research communities interested in this class of systems. In the Introduction, we have stated our vision of runtime adaptation for communication-based systems. Here we then limit to remark that this vision is well-aligned with the conceptual definition of adaptation recently given in [5]: "Given a component with a distinguished collection of control data, adaptation is the runtime modification of such control data." In the case of the calculus of adaptable processes, *control data* refers to the located processes in which behavior can be structured. The same observation applies for the session calculus considered here, noticing that our framework precisely defines the runtime conditions in which adaptation may consistently occur.

Adaptation vs exceptions and compensations. From a high-level perspective, our typed framework can be related to formal models for service-oriented systems with constructs such as exceptions and compensations (e.g., [10,15]). In particular, [10] develops a typeful approach for *interactional exceptions* in asynchronous, binary session types. There, services define a default process and an exception handler, and may be influenced by a `throw` construct used to throw exceptions; the associated type system ensures consistent dynamic escaping from possibly nested exception scopes in a concerted way. We find conceptual differences between our intended notion of runtime adaptation (described in the Introduction) and mechanisms for exceptions and compensations. In our view, such mechanisms are concerned only with a particular instance of adaptation/evolvability scenarios. This way, forms of exceptions (e.g., the one in [10]) should typically handle *internal events* which arise during the program's execution and may affect its flow of control.

Similar in several respects to exceptions, constructs for compensations are usually conceived for handling events which are often exceptional *and* catastrophic, such as runtime errors. This is in contrast with runtime adaptation in modern distributed systems, which relies on *external* mechanisms tailored to handle general exceptional events, not necessarily catastrophic. As an example, consider elasticity in cloud-based applications, i.e., the ability such applications have to acquire/release computing resources based on user demand. Although elasticity triggers system adaptation, it does not represent a catastrophic event; rather, it represents an acceptable (yet hard to predict) state of the system. Due to this conceptual difference, we do not have a clear perspective as to how formally relate our approach with known models of exceptions/compensations.

Somewhat related to our approach is the work [1], where dynamic runtime update for message passing programs with queues is studied, and a form of consistency for updates over threads is ensured using multiparty session types. As in our setting, the notion of update in [1] does not require a complete system shutdown, while ensuring that the state of the program remains consistent. However, there are significant conceptual and technical differences. First, unlike our approach, in [1] update actions (or adaptation routines) are defined independently and externally from the program's syntax. Second, here we have considered synchronous communication, whereas in [1] an asynchronous thread model with queues is used. Third, while we have analyzed updates for a language endowed with binary session types, in [1] the need for reaching agreements on multiple threads leads to the use of multiparty session types to ensure consistent updates.

Adaptation in other settings. It is instructive to note that approaches similar to ours can also be found in the sequential formalisms for reasoning at lower levels of abstraction. For instance, in [27] the authors introduce dynamic updates to C-like languages, with a focus on runtime update of functions and type declarations. They show that dynamic updates are type-safe (consistent) as it cannot happen that different parts of the program expect different representations of a type. Although the aim is similar, it is difficult to establish more precise comparisons, for our interest is in high-level reasoning for communicating programs with precise protocol descriptions. In [4] the authors also investigate behavioral types for adaptation, but in the different setting of component-based systems. Their notion of runtime adaptation relies on a notion of interface which describes the behavior of each component. These interfaces are to be used to implement inter-component communication. As it could that composition might not work because of interface mismatching, the authors introduce the notion of *adaptors*, i.e., a software piece that acts as mediator between two communicating components. It is interesting to notice that a similar mediator behavior could be implemented in our setting, as described in Section 5.2.

8. Concluding remarks

A main motivation for our work is the observation that while paradigms such as service-oriented computing are increasingly popular among practitioners, formal models based on them—such as analysis techniques based on session types—fail to capture distinctive aspects of such paradigms. Here we have addressed, for the first time, one of such aspects, namely *runtime adaptation*. In our view, it represents an increasingly important concern when analyzing the behavior of communicating systems in open, context-aware computing infrastructures.

We have proposed a framework for reasoning about runtime adaptation in the context of structured communications described by session types. Amalgamating a static analysis technique for correct communications (session types) with a novel, inherently dynamic form of interaction (runtime adaptation actions on located processes) is challenging, for it requires balancing two different (but equally important) concerns in modern interacting systems. In our approach, we are concerned with a session type discipline for an extended π-calculus, in which channel mobility (delegation) is enhanced with the possibility of performing sophisticated update actions on located processes. We purposefully aimed at integrating existing, well-studied lines of work: we expect this to be beneficial for the enhancement of other known session type disciplines with adaptation concerns. In particular, we built upon our previous work on process abstractions for evolvability [3] and on session types for mobile calculi [16], relying also on a modern account of binary session types [29]. In addition to runtime correctness (safety), our typing discipline ensures consistency: this guarantees that communication behavior (as declared by types) is not disrupted by potential update actions.

There are several avenues for future developments. In ongoing work, we are exploring the use of a *typecase* operator (similar to the one proposed in [22]) to support the runtime adaptation of processes with running sessions. As detailed in Section 6, this is a challenging issue, for it requires having a uniform treatment of process behavior and the current state of sessions. Also, we intend to continue the study of runtime adaptation in a multiparty setting, developing further the approach recently introduced in [2]. Furthermore, we plan to investigate the interplay of runtime adaptation with issues such as deadlock-freedom/progress [13,8] and properties related to security, such as access control and information flow [7].

Acknowledgements

We thank the anonymous reviewers for their detailed remarks. Most of this research was carried out when the second author was a postdoctoral fellow at CITI – Departamento de Informatica, FCT – Universidade Nova de Lisboa, Portugal. This work was partially supported by the French project ANR BLANC 0307 01 – SYNBIOTIC, as well as by the Portuguese Foundation for Science and Technology (FCT) grants SFRH/BPD/84067/2012 and CITI.

Appendix A. Auxiliary definitions

A.1. Cardinality of typings

Definition 33. Let Δ be a typing, as in Table 3. The cardinality of Δ, denoted $|\Delta|$, is inductively defined as follows:

$$\begin{aligned} |\emptyset| &= 0 \\ |\Delta', k:\alpha| &= 1 + |\Delta'| \\ |\Delta', [k:\alpha]| &= 1 + |\Delta'| \end{aligned}$$

Appendix B. Proofs from Section 4

B.1. Proof of Theorem 15

We repeat the statement given in p. 244 and give its proof.

Theorem 34 *(Subject congruence). If $\Gamma\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I}$ and $P \equiv Q$ then $\Gamma\,;\,\Theta \vdash Q \triangleright \Delta;\mathcal{I}$.*

Proof. By induction on the derivation of $P \equiv Q$, with a case analysis on the last applied rule.

Case $(\nu\kappa)(l^h[P]) \equiv l^h[(\nu\kappa)P]$**.**
We examine the left to right direction: we show that if $\Gamma\,;\,\Theta \vdash (\nu\kappa)(l^h[P]) \triangleright \Delta;\mathcal{I}$ then $\Gamma\,;\,\Theta \vdash l^h[(\nu\kappa)P] \triangleright \Delta;\mathcal{I}$. Since $(\nu\kappa)(l^h[P])$ is well-typed, by inversion on rules (T:Loc) and (T:CRes), for some α, Δ' we have:

$$\dfrac{\dfrac{\dfrac{\Theta \vdash l : \mathcal{I}' \quad \mathcal{I} \sqsubseteq \mathcal{I}' \qquad \Gamma\,;\,\Theta \vdash P \triangleright \Delta', \kappa^- : \alpha, \kappa^+ : \overline{\alpha};\mathcal{I} \quad h = |\Delta', \kappa^- : \alpha, \kappa^+ : \overline{\alpha}|}{\Gamma\,;\,\Theta \vdash l^h[P] \triangleright \Delta', \kappa^- : \alpha, \kappa^+ : \overline{\alpha};\mathcal{I}}}{\Gamma\,;\,\Theta \vdash (\nu\kappa)(l^h[P]) \triangleright \Delta', [\kappa^- : \alpha], [\kappa^+ : \overline{\alpha}];\mathcal{I}}}{}$$

Hence $\Gamma\,;\,\Theta \vdash P \triangleright \Delta', \kappa^- : \alpha, \kappa^+ : \overline{\alpha};\mathcal{I}$, where $\Delta = \Delta', [\kappa^- : \alpha], [\kappa^+ : \overline{\alpha}]$. Now, starting from P, and by applying first rule (T:CRes) and then rule (Loc) we obtain:

$$\dfrac{\dfrac{\Gamma\,;\,\Theta \vdash P \triangleright \Delta', \kappa^- : \alpha, \kappa^+ : \overline{\alpha};\mathcal{I}}{\Gamma\,;\,\Theta \vdash (\nu\kappa)P \triangleright \Delta', [\kappa^- : \alpha], [\kappa^+ : \overline{\alpha}];\mathcal{I}} \quad \Theta \vdash l : \mathcal{I}' \qquad \mathcal{I} \sqsubseteq \mathcal{I}' \quad h = |\Delta', [\kappa^- : \alpha], [\kappa^+ : \overline{\alpha}]|}{\Gamma\,;\,\Theta \vdash l^h[(\nu\kappa)P] \triangleright \Delta', [\kappa^- : \alpha], [\kappa^+ : \overline{\alpha}];\mathcal{I}}$$

Observe that $h = |\Delta', [\kappa^- : \alpha], [\kappa^+ : \overline{\alpha}]| = |\Delta', \kappa^- : \alpha, \kappa^+ : \overline{\alpha}|$—bracketing does not influence h, i.e., the reasoning for the right to left direction is analogous and omitted.

Case $P \mid \mathbf{0} \equiv P$**.**
We examine only the left to right direction; the converse direction is similar. We then show that if $\Gamma\,;\,\Theta \vdash P \mid \mathbf{0} \triangleright \Delta;\mathcal{I}$ then $\Gamma\,;\,\Theta \vdash P \triangleright \Delta;\mathcal{I}$. By inversion on rule (T:Par) there exist $\Delta_1, \mathcal{I}_1$ such that

$$\Gamma\,;\,\Theta \vdash P \triangleright \Delta_1;\mathcal{I}_1 \quad \text{and} \quad \Gamma\,;\,\Theta \vdash \mathbf{0} \triangleright \emptyset;\emptyset$$

with $\Delta = \Delta_1 \cup \emptyset = \Delta_1$ and $\mathcal{I} = \mathcal{I}_1 \uplus \emptyset = \mathcal{I}_1$ and so the thesis follows.

Case $(\nu\kappa)P \mid Q \equiv (\nu\kappa)(P \mid Q)$ **with** $\kappa \notin \mathsf{fc}(Q)$**.**
We examine only the right to left direction; the other direction is analogous. We show that if $\Gamma\,;\,\Theta \vdash (\nu\kappa)(P \mid Q) \triangleright \Delta;\mathcal{I}$ (with $\kappa \notin \mathsf{fc}(Q)$) then also $\Gamma\,;\,\Theta \vdash (\nu\kappa)P \mid Q \triangleright \Delta;\mathcal{I}$. By inversion on rules (T:CRes) and (T:Par) we have:

$$\dfrac{\dfrac{\Gamma\,;\,\Theta \vdash P \triangleright \Delta_1, \kappa^- : \alpha, \kappa^+ : \overline{\alpha};\mathcal{I}_1 \qquad \Gamma\,;\,\Theta \vdash Q \triangleright \Delta_2;\mathcal{I}_2 \qquad \mathcal{I} = \mathcal{I}_1 \uplus \mathcal{I}_2}{\Gamma\,;\,\Theta \vdash P \mid Q \triangleright \Delta_1 \cup \Delta_2, \kappa^- : \alpha, \kappa^+ : \overline{\alpha};\mathcal{I}'}}{\Gamma\,;\,\Theta \vdash (\nu\kappa)(P \mid Q) \triangleright \Delta_1 \cup \Delta_2, [\kappa^- : \alpha], [\kappa^+ : \overline{\alpha}];\mathcal{I}}$$

where $\Delta = \Delta_1 \cup \Delta_2, [\kappa^- : \alpha], [\kappa^+ : \overline{\alpha}]$. Observe how in the inversion of rule (T:Par) we may combine assumption $\kappa \notin \mathsf{fc}(Q)$ with α-conversion to infer $\kappa \notin \mathsf{fc}(Q) \cup \mathsf{bc}(Q)$. We may then use Lemma 14 to infer $\Gamma\,;\,\Theta \vdash P \triangleright \Delta_1, \kappa^- : \alpha, \kappa^+ : \overline{\alpha};\mathcal{I}_1$ and $\Gamma\,;\,\Theta \vdash Q \triangleright \Delta_2;\mathcal{I}_2$. Now, using first rule (T:CRes) and then rule (T:Par) we have:

$$\dfrac{\dfrac{\Gamma\,;\,\Theta \vdash P \triangleright \Delta_1, \kappa^- : \alpha, \kappa^+ : \overline{\alpha};\mathcal{I}_1}{\Gamma\,;\,\Theta \vdash (\nu\kappa)P \triangleright \Delta_1, [\kappa^- : \alpha], [\kappa^+ : \overline{\alpha}];\mathcal{I}_1} \qquad \begin{array}{c}\Gamma\,;\,\Theta \vdash Q \triangleright \Delta_2;\mathcal{I}_2 \\ \mathcal{I} = \mathcal{I}_1 \uplus \mathcal{I}_2\end{array}}{\Gamma\,;\,\Theta \vdash (\nu\kappa)P \mid Q \triangleright \Delta_1 \cup \Delta_2, [\kappa^- : \alpha], [\kappa^+ : \overline{\alpha}];\mathcal{I}}$$

Case $(\nu\kappa)\mathbf{0} \equiv \mathbf{0}$**.**
This case is easily proven by appealing to rule (T:Weak).

Cases $P \mid Q \equiv Q \mid P$ **and** $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$**.**
In both cases, the proof follows by commutativity and associativity of $\cup$ and $\uplus$ (cf. Definition 6).

Case $(\nu\kappa)(\nu\kappa')P \equiv (\nu\kappa')(\nu\kappa)P$**.**
This case is similar to previous ones. □

B.2. Proof of Theorem 21

The proof of Theorem 21 relies on the following lemma that allows to reverse typing rules.

Lemma 35 *(Inversion lemma).*

(T:ACCEPT): *if* $\Gamma\,;\,\Theta \vdash \mathsf{accept}\,a(x).P \triangleright \Delta; \mathcal{I} \uplus a : \alpha_{\mathsf{lin}}$ *then* $\Gamma\,;\,\Theta \vdash P \triangleright \Delta, x : \alpha; \mathcal{I}$;
(T:REPACCEPT): *if* $\Gamma\,;\,\Theta \vdash \mathbf{!}\,\mathsf{accept}\,a(x).P \triangleright \Delta; \mathcal{I} \uplus a : \alpha_{\mathsf{un}}$ *then there exists* $\mathcal{I}'$ *such that* $\Gamma\,;\,\Theta \vdash P \triangleright \Delta, x : \alpha; \mathcal{I}'$;
(T:REQUEST): *if* $\Gamma\,;\,\Theta \vdash \mathsf{request}\,a(x).P \triangleright \Delta; \mathcal{I} \uplus a : \overline{\alpha}_{\mathsf{lin}}$ *then* $\Gamma\,;\,\Theta \vdash P \triangleright \Delta, x : \overline{\alpha}; \mathcal{I}$;
(T:CLOSE): *if* $\Gamma\,;\,\Theta \vdash \mathsf{close}\,(k).P \triangleright \Delta, k : \epsilon; \mathcal{I}$ *then* $\Gamma\,;\,\Theta \vdash P \triangleright \Delta; \mathcal{I}$;
(T:LOC): *if* $\Gamma\,;\,\Theta \vdash l^h[P] \triangleright \Delta; \mathcal{I}$ *then* $\Theta \vdash l : \mathcal{I}'$, $\Gamma\,;\,\Theta \vdash P \triangleright \Delta; \mathcal{I}$, $h = |\Delta|$ *and* $\mathcal{I} \sqsubseteq \mathcal{I}'$;
(T:ADAPT): *if* $\Gamma\,;\,\Theta \vdash l\{(X).P\} \triangleright \emptyset; \emptyset$ *then* $\Theta \vdash l : \mathcal{I}$ *and there exists* $\mathcal{I}'$ *such that* $\Gamma\,;\,\Theta, \mathsf{X} : \mathcal{I} \vdash P \triangleright \emptyset; \mathcal{I}'$;
(T:CRES): *if* $\Gamma\,;\,\Theta \vdash (\nu\kappa)P \triangleright \Delta, [\kappa^- : \alpha], [\kappa^+ : \overline{\alpha}]; \mathcal{I}$ *then* $\Gamma\,;\,\Theta \vdash P \triangleright \Delta, \kappa^- : \alpha, \kappa^+ : \overline{\alpha}; \mathcal{I}$;
(T:PAR): *if* $\Gamma\,;\,\Theta \vdash P \mid Q \triangleright \Delta; \mathcal{I}$ *then there exists* $\Delta_1, \Delta_2, \mathcal{I}_1, \mathcal{I}_2$ *such that* $\Gamma\,;\,\Theta \vdash P \triangleright \Delta_1; \mathcal{I}_1$, $\Gamma\,;\,\Theta \vdash Q \triangleright \Delta_2; \mathcal{I}_2$, $\Delta = \Delta_1 \cup \Delta_2$ *and* $\mathcal{I} = \mathcal{I}_1 \uplus \mathcal{I}_2$;
(T:THR): *if* $\Gamma\,;\,\Theta \vdash k\langle\langle k' \rangle\rangle.P \triangleright \Delta, k : !(\alpha).\beta, k' : \alpha; \mathcal{I}$ *then* $\Gamma\,;\,\Theta \vdash P \triangleright \Delta, k : \beta; \mathcal{I}$;
(T:CAT): *if* $\Gamma\,;\,\Theta \vdash k((x)).P \triangleright \Delta, k ?(\alpha).\beta; \mathcal{I}$ *then* $\Gamma\,;\,\Theta \vdash P \triangleright \Delta, k : \beta, x : \alpha; \mathcal{I}$;
(T:IN): *if* $\Gamma\,;\,\Theta \vdash k(\widetilde{x}).P \triangleright \Delta, k : ?(\widetilde{\tau}).\alpha; \mathcal{I}$ *then* $\Gamma, \widetilde{x} : \widetilde{\tau}\,;\,\Theta \vdash P \triangleright \Delta, k : \alpha; \mathcal{I}$ *and* $\Gamma \vdash \widetilde{e} : \widetilde{\tau}$;
(T:OUT): *if* $\Gamma\,;\,\Theta \vdash k\langle\widetilde{e}\rangle.P \triangleright \Delta, k : !(\widetilde{\tau}).\alpha; \mathcal{I}$ *then* $\Gamma\,;\,\Theta \vdash P \triangleright \Delta, k : !(\widetilde{\tau}).\alpha; \mathcal{I}$;
(T:IF): *if* $\Gamma\,;\,\Theta \vdash \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta; \mathcal{I}$ *then* $\Gamma\,;\,\Theta \vdash P \triangleright \Delta; \mathcal{I}$, $\Gamma\,;\,\Theta \vdash Q \triangleright \Delta; \mathcal{I}$ *and* $\Gamma \vdash e : \mathsf{bool}$;
(T:BRA): *if* $\Gamma\,;\,\Theta \vdash k \rhd \{n_1 : P_1 \,[]\, \ldots \,[]\, n_m : P_m\} \triangleright \Delta, k : \&\{n_1 : \alpha_1, \ldots, n_m : \alpha_m\}; \mathcal{I}$ *then there exists* $\mathcal{I}_1, \ldots, \mathcal{I}_m$ *such that for all* $i \in [1..m]$, $\Gamma\,;\,\Theta \vdash P_i \triangleright \Delta, k : \alpha_i; \mathcal{I}_i$;
(T:SEL): *if* $\Gamma\,;\,\Theta \vdash k \lhd n_i.P \triangleright \Delta, k : \oplus\{n_1 : \alpha_1, \ldots, n_m : \alpha_m\}; \mathcal{I}$ *then* $i \in [1..m]$ *and* $\Gamma\,;\,\Theta \vdash P \triangleright \Delta, k : \alpha_i; \mathcal{I}$.

Proof. Follows directly from the definition of typing system. □

We repeat the statement in p. 245 and present its proof.

Theorem 36 *(Subject reduction). If* $\Gamma\,;\,\Theta \vdash P \triangleright \Delta; \mathcal{I}$ *with* Δ *balanced and* $P \longrightarrow Q$ *then* $\Gamma\,;\,\Theta \vdash Q \triangleright \Delta'; \mathcal{I}'$, *for some* $\mathcal{I}'$ *and balanced* Δ'.

Proof. By induction on the last rule applied in the reduction. We assume that $\widetilde{e} \downarrow \widetilde{c}$ is a type preserving operation, for every $\widetilde{e}$.

Case (R:OPEN). From Table 2 we have:

$$E\{C\{\mathsf{accept}\,a(x).P_1\} \mid D\{\mathsf{request}\,a(y).P_2\}\} \longrightarrow E^{++}\{(\nu\kappa)(C^+\{P_1[\kappa^+/x]\} \mid D^+\{P_2[\kappa^-/y]\})\}$$

By assumption $\Gamma\,;\,\Theta \vdash E\{C\{\mathsf{accept}\,a(x).P_1\} \mid D\{\mathsf{request}\,a(y).P_2\}\} \triangleright \Delta; \mathcal{I}$ with balanced Δ. Then, by inversion on typing, using rules (T:FILL), (T:ACCEPT), (T:REQUEST), and (T:PAR) we infer there exist $\Delta', \mathcal{I}'$ such that

$$\frac{\bullet_{\mathcal{S}_0};\ \Gamma\,;\,\Theta \vdash E \triangleright \Delta; \mathcal{I} \qquad \dfrac{\text{(B.3)} \quad \text{(B.5)}}{\Gamma\,;\,\Theta \vdash C\{\mathsf{accept}\,a(x).P_1\} \mid D\{\mathsf{request}\,a(y).P_2\} \triangleright \Delta'; \mathcal{I}'}}{\Gamma\,;\,\Theta \vdash E\{C\{\mathsf{accept}\,a(x).P_1\} \mid D\{\mathsf{request}\,a(y).P_2\}\} \triangleright \Delta; \mathcal{I}} \tag{B.1}$$

where $\mathcal{I}' = (\mathcal{I}'_1 \uplus a : \alpha_{\mathsf{lin}}) \uplus (\mathcal{I}'_2 \uplus a : \overline{\alpha}_{\mathsf{lin}})$ and

$$\mathcal{S}_0 = \Gamma; \Theta \vdash \Delta'; \mathcal{I}' \tag{B.2}$$

By Lemma 18, $\Delta' \subseteq \Delta$ and $\mathcal{I}' \sqsubseteq \mathcal{I}$. Then, letting $\Delta' = \Delta'_1 \cup \Delta'_2$, subtree (B.3) is as follows:

$$\frac{\bullet_{\mathcal{S}_1};\ \Gamma\,;\,\Theta \vdash C \triangleright \Delta'_1; \mathcal{I}'_1 \uplus a : \alpha_{\mathsf{lin}} \qquad \dfrac{\Gamma \vdash a : \langle \alpha_{\mathsf{lin}}, \overline{\alpha}_{\mathsf{lin}} \rangle \quad \Gamma\,;\,\Theta \vdash P_1 \triangleright \Delta_1, x : \alpha; \mathcal{I}_1}{\Gamma\,;\,\Theta \vdash \mathsf{accept}\,a(x).P_1 \triangleright \Delta_1; \mathcal{I}_1 \uplus a : \alpha_{\mathsf{lin}}}}{\Gamma\,;\,\Theta \vdash C\{\mathsf{accept}\,a(x).P_1\} \triangleright \Delta'_1; \mathcal{I}'_1 \uplus a : \alpha_{\mathsf{lin}}} \tag{B.3}$$

with

$$\mathcal{S}_1 = \Gamma; \Theta \vdash \Delta_1; \mathcal{I}_1 \uplus a : \alpha_{\mathsf{lin}} \tag{B.4}$$

Then, subtree (B.5) is as follows:

$$\frac{\bullet_{\mathcal{S}_2};\ \Gamma\,;\,\Theta \vdash D \triangleright \Delta'_2; \mathcal{I}'_2 \uplus a : \overline{\alpha}_{\mathsf{lin}} \qquad \dfrac{\Gamma \vdash a : \langle \alpha_{\mathsf{lin}}, \overline{\alpha}_{\mathsf{lin}} \rangle \quad \Gamma\,;\,\Theta \vdash P_2 \triangleright \Delta_2, y : \overline{\alpha}; \mathcal{I}_2}{\Gamma\,;\,\Theta \vdash \mathsf{request}\,a(y).P_2 \triangleright \Delta_2; \mathcal{I}_2 \uplus a : \overline{\alpha}_{\mathsf{lin}}}}{\Gamma\,;\,\Theta \vdash D\{\mathsf{request}\,a(y).P_2\} \triangleright \Delta'_2; \mathcal{I}'_2 \uplus a : \overline{\alpha}_{\mathsf{lin}}} \tag{B.5}$$

with

$$\mathcal{S}_2 = \Gamma; \Theta \vdash \Delta_2; \mathcal{I}_2 \uplus a : \overline{\alpha}_{\mathsf{lin}} \tag{B.6}$$

By Lemma 18 we have that $\Delta_1 \subseteq \Delta_1'$ and $\Delta_2 \subseteq \Delta_2'$. We also infer $\mathcal{I}_1 \sqsubseteq \mathcal{I}_1'$, $\mathcal{I}_2 \sqsubseteq \mathcal{I}_2'$, and $\mathcal{I}' \sqsubseteq \mathcal{I}$. Now, using Lemma 16(1) on judgments for P_1 and P_2, we obtain:

(a) $\Gamma\,;\,\Theta \vdash P_1[\kappa^+/x] \triangleright \Delta_1, \kappa^+ : \alpha; \mathcal{I}_1$.
(b) $\Gamma\,;\,\Theta \vdash P_2[\kappa^-/y] \triangleright \Delta_2, \kappa^- : \overline{\alpha}; \mathcal{I}_2$.

We now describe how to obtain appropriately typed contexts C^+, D^+, and E^{++} based on the information inferred up to here on contexts C, D, and E. We first describe the case of C^+. From (B.3) above we obtained $\bullet_{\mathcal{S}_1}; \Gamma\,;\,\Theta \vdash C \triangleright \Delta_1'; \mathcal{I}_1' \uplus a : \alpha_{\mathsf{lin}}$ with $\mathcal{S}_1$ as in (B.4). Then, using Lemma 19(1), we infer $\bullet_{\mathcal{S}_3}; \Gamma\,;\,\Theta \vdash C^+ \triangleright \Delta_1', \kappa^+ : \alpha; \mathcal{I}_1'$ with

$$\mathcal{S}_3 = \Gamma; \Theta \vdash \Delta_1, \kappa^+ : \alpha; \mathcal{I}_1 \tag{B.7}$$

We may now reconstruct the derivation given in (B.3):

$$\frac{\bullet_{\mathcal{S}_3}; \Gamma\,;\,\Theta \vdash C^+ \triangleright \Delta_1', \kappa^+ : \alpha\,;\,\mathcal{I}_1' \quad \Gamma\,;\,\Theta \vdash P_1[\kappa^+/x] \triangleright \Delta_1, \kappa^+ : \alpha; \mathcal{I}_1}{\Gamma\,;\,\Theta \vdash C^+\{P_1[\kappa^+/x]\} \triangleright \Delta_1', \kappa^+ : \alpha; \mathcal{I}_1'} \tag{B.8}$$

For D^+, we proceed analogously from (B.5) and infer:

$$\frac{\bullet_{\mathcal{S}_4}; \Gamma\,;\,\Theta \vdash D^+ \triangleright \Delta_2', \kappa^- : \overline{\alpha}; \mathcal{I}_2' \quad \Gamma\,;\,\Theta \vdash P_2[\kappa^-/y] \triangleright \Delta_2, \kappa^- : \overline{\alpha}; \mathcal{I}_2}{\Gamma\,;\,\Theta \vdash D^+\{P_2[\kappa^-/y]\} \triangleright \Delta_2', \kappa^- : \overline{\alpha}; \mathcal{I}_2'} \tag{B.9}$$

with

$$\mathcal{S}_4 = \Gamma; \Theta \vdash \Delta_2, \kappa^- : \overline{\alpha}; \mathcal{I}_2 \tag{B.10}$$

To infer the type of E^{++} we proceed as before using twice Lemma 19(1), combined with (B.2). We may finally derive the type for the result of the reduction: using rules (T:Par), (T:CRes), and (T:Fill) we obtain:

$$\frac{\dfrac{\dfrac{\text{(B.8)} \quad \text{(B.9)}}{\Gamma\,;\,\Theta \vdash C^+\{P_1[\kappa^+/x]\} \mid D^+\{P_2[\kappa^-/y]\} \triangleright \Delta', \kappa^+ : \alpha, \kappa^- : \overline{\alpha}; \mathcal{I}_1' \uplus \mathcal{I}_2'}}{\Gamma\,;\,\Theta \vdash (\nu\kappa)C^+\{P_1[\kappa^+/x]\} \mid D^+\{P_2[\kappa^-/y]\} \triangleright \Delta', [\kappa^+ : \alpha], [\kappa^- : \overline{\alpha}]; \mathcal{I}_1' \uplus \mathcal{I}_2'}\ \text{(B.11)}}{\Gamma\,;\,\Theta \vdash E^{++}\{(\nu\kappa)C^+\{P_1[\kappa^+/x]\} \mid D^+\{P_2[\kappa^-/y]\}\} \triangleright \Delta, [\kappa^+ : \alpha], [\kappa^- : \overline{\alpha}]; \mathcal{I}''}$$

with

$$\bullet_{\mathcal{S}_5}; \Gamma\,;\,\Theta \vdash E \triangleright \Delta, [\kappa^+ : \alpha], [\kappa^- : \overline{\alpha}]; \mathcal{I}'' \tag{B.11}$$

and

$$\mathcal{S}_5 = \Gamma; \Theta \vdash \Delta', \kappa^+ : \alpha, \kappa^- : \overline{\alpha}; \mathcal{I}_1' \uplus \mathcal{I}_2'$$

Notice that by Lemma 18, we have $\mathcal{I}'' \sqsubseteq \mathcal{I}_1' \cup \mathcal{I}_2'$. Also, observe that by assumption Δ is balanced; therefore, by Definition 20 the resulting typing $\Delta, [\kappa^+ : \alpha], [\kappa^- : \overline{\alpha}]$ is balanced too. This concludes this case.

Case (R:ROpen). From Table 2 we have:

$$E\{C\{!\,\mathsf{accept}\,a(x).P_1\} \mid D\{\mathsf{request}\,a(y).P_2\}\} \longrightarrow$$
$$E^{++}\{(\nu\kappa)(C^+\{P_1[\kappa^+/x] \mid !\,\mathsf{accept}\,a(x).P_1\} \mid D^+\{P_2[\kappa^-/y]\})\}$$

By assumption $\Gamma\,;\,\Theta \vdash E\{C\{!\,\mathsf{accept}\,a(x).P_1\} \mid D\{\mathsf{request}\,a(y).P_2\}\} \triangleright \Delta; \mathcal{I}$, with balanced Δ. Then, by inversion on typing, using rules (T:Fill), (T:RepAccept), (T:Request), and (T:Par), we infer there exist Δ', $\mathcal{I}'$ such that:

$$\frac{\bullet_{\mathcal{S}_0}; \Gamma\,;\,\Theta \vdash E \triangleright \Delta; \mathcal{I} \quad \dfrac{\text{(B.13)} \quad \text{(B.15)}}{\Gamma\,;\,\Theta \vdash C\{!\,\mathsf{accept}\,a(x).P_1\} \mid D\{\mathsf{request}\,a(y).P_2\} \triangleright \Delta'; \mathcal{I}'}}{\Gamma\,;\,\Theta \vdash E\{C\{!\,\mathsf{accept}\,a(x).P_1\} \mid D\{\mathsf{request}\,a(y).P_2\}\} \triangleright \Delta; \mathcal{I}}$$

where $\mathcal{I}' = (\mathcal{I}_1' \uplus a : \alpha_{\mathsf{un}}) \uplus (\mathcal{I}_2' \uplus a : \overline{\alpha}_{\mathsf{lin}})$ and

$$\mathcal{S}_0 = \Gamma; \Theta \vdash \Delta'; \mathcal{I}' \tag{B.12}$$

By Lemma 18, $\Delta' \subseteq \Delta$ and $\mathcal{I}' \sqsubseteq \mathcal{I}$. Then, letting $\Delta' = \Delta'_1 \cup \Delta'_2$, subtree (B.13) is as follows:

$$\dfrac{\bullet_{\mathcal{S}_1};\ \Gamma\ ;\ \Theta \vdash C \triangleright \Delta'_1; \mathcal{I}'_1 \uplus a : \alpha_{\mathsf{un}} \qquad \dfrac{\Gamma \vdash a : \langle \alpha_{\mathsf{un}}, \overline{\alpha}_{\mathsf{lin}} \rangle \quad \Gamma\ ;\ \Theta \vdash P_1 \triangleright x : \alpha; \mathcal{I}_1}{\Gamma\ ;\ \Theta \vdash \mathbf{!}\,\mathsf{accept}\, a(x).P_1 \triangleright \emptyset; \uparrow^{\mathsf{un}}(\mathcal{I}_1) \uplus a : \alpha_{\mathsf{un}}}}{\Gamma\ ;\ \Theta \vdash C\{\mathbf{!}\,\mathsf{accept}\, a(x).P_1\} \triangleright \Delta'_1; \mathcal{I}'_1 \uplus a : \alpha_{\mathsf{un}}} \tag{B.13}$$

with

$$\mathcal{S}_1 = \Gamma; \Theta \vdash \emptyset; \uparrow^{\mathsf{un}}(\mathcal{I}_1) \uplus a : \alpha_{\mathsf{un}} \tag{B.14}$$

Then, subtree (B.15) is as follows:

$$\dfrac{\bullet_{\mathcal{S}_2};\ \Gamma\ ;\ \Theta \vdash D \triangleright \Delta'_2; \mathcal{I}'_2 \uplus a : \overline{\alpha}_{\mathsf{lin}} \qquad \dfrac{\Gamma \vdash a : \langle \alpha_{\mathsf{un}}, \overline{\alpha}_{\mathsf{lin}} \rangle \quad \Gamma\ ;\ \Theta \vdash P_2 \triangleright \Delta_2, y : \overline{\alpha}; \mathcal{I}_2}{\Gamma\ ;\ \Theta \vdash \mathsf{request}\, a(y).P_2 \triangleright \Delta_2; \mathcal{I}_2 \uplus a : \overline{\alpha}_{\mathsf{lin}}}}{\Gamma\ ;\ \Theta \vdash D\{\mathsf{request}\, a(y).P_2\} \triangleright \Delta'_2; \mathcal{I}'_2 \uplus a : \overline{\alpha}_{\mathsf{lin}}} \tag{B.15}$$

with

$$\mathcal{S}_2 = \Gamma; \Theta \vdash \Delta_2; \mathcal{I}_2 \uplus a : \overline{\alpha}_{\mathsf{lin}} \tag{B.16}$$

By Lemma 18 we have $\Delta_1 \subseteq \Delta'_1$ and $\Delta_2 \subseteq \Delta'_2$. Moreover, $\mathcal{I}_1 \sqsubseteq \mathcal{I}'_1$, $\mathcal{I}_2 \sqsubseteq \mathcal{I}'_2$ and $\mathcal{I}' \sqsubseteq \mathcal{I}$. Now, using Lemma 16(1) on P_1 and P_2, we have:

(a) $\Gamma\ ;\ \Theta \vdash P_1[\kappa^+/x] \triangleright \kappa^+ : \alpha; \mathcal{I}_1$.
(b) $\Gamma\ ;\ \Theta \vdash P_2[\kappa^-/y] \triangleright \Delta_2, \kappa^- : \overline{\alpha}; \mathcal{I}_2$.

We now describe how to obtain appropriately typed contexts C^+, D^+, and E^{++} based on the information inferred up to here on contexts C, D, and E. We first describe the case of C^+. From (B.13) above we obtained $\bullet_{\mathcal{S}_1};\ \Gamma\ ;\ \Theta \vdash C \triangleright \Delta'_1; \mathcal{I}'_1 \uplus a : \alpha_{\mathsf{un}}$ with $\mathcal{S}_1$ as in (B.14). Then, using Lemma 19(1), we infer $\bullet_{\mathcal{S}_3};\ \Gamma\ ;\ \Theta \vdash C^+ \triangleright \Delta'_1, \kappa^+ : \alpha; \mathcal{I}'_1$ with

$$\mathcal{S}_3 = \Gamma; \Theta \vdash \kappa^+ : \alpha; \uparrow^{\mathsf{un}}(\mathcal{I}_1) \uplus a : \alpha_{\mathsf{un}} \tag{B.17}$$

We may now reconstruct the derivation given in (B.13):

$$\dfrac{\bullet_{\mathcal{S}_3};\ \Gamma\ ;\ \Theta \vdash C^+ \triangleright \Delta'_1, \kappa^+ : \alpha\ ; \mathcal{I}'_1 \uplus a : \alpha_{\mathsf{un}} \quad \Gamma\ ;\ \Theta \vdash P_1[\kappa^+/x] \triangleright \kappa^+ : \alpha; \uparrow^{\mathsf{un}}(\mathcal{I}_1) \uplus a : \alpha_{\mathsf{un}}}{\Gamma\ ;\ \Theta \vdash C^+\{P_1[\kappa^+/x]\} \triangleright \Delta'_1, \kappa^+ : \alpha; \mathcal{I}'_1 \uplus a : \alpha_{\mathsf{un}}} \tag{B.18}$$

For D^+, we proceed analogously from (B.15) and infer:

$$\dfrac{\bullet_{\mathcal{S}_4};\ \Gamma\ ;\ \Theta \vdash D^+ \triangleright \Delta'_2, \kappa^- : \overline{\alpha}; \mathcal{I}'_2 \quad \Gamma\ ;\ \Theta \vdash P_2[\kappa^-/y] \triangleright \Delta_2, \kappa^- : \overline{\alpha}; \mathcal{I}_2}{\Gamma\ ;\ \Theta \vdash D^+\{P_2[\kappa^-/y]\} \triangleright \Delta'_2, \kappa^- : \overline{\alpha}; \mathcal{I}'_2} \tag{B.19}$$

with

$$\mathcal{S}_4 = \Gamma; \Theta \vdash \Delta_2, \kappa^- : \overline{\alpha}; \mathcal{I}_2 \tag{B.20}$$

To infer the type of E^{++} we proceed as before using twice Lemma 19(1), combined with (B.12). We may finally derive the type for the result of the reduction: using rules (T:Par), (T:Cres), and (T:Fill) we obtain:

$$\dfrac{\dfrac{\dfrac{\text{(B.18)} \quad \text{(B.19)}}{\Gamma\ ;\ \Theta \vdash C^+\{P_1[\kappa^+/x]\} \mid D^+\{P_2[\kappa^-/y]\} \triangleright \Delta', \kappa^+ : \alpha, \kappa^- : \overline{\alpha}; \mathcal{I}'_1 \uplus \mathcal{I}'_2 \uplus a : \alpha_{\mathsf{un}}}}{\text{(B.21)} \quad \Gamma\ ;\ \Theta \vdash (\nu\kappa)C^+\{P_1[\kappa^+/x]\} \mid D^+\{P_2[\kappa^-/y]\} \triangleright \Delta', [\kappa^+ : \alpha], [\kappa^- : \overline{\alpha}]; \mathcal{I}'_1 \uplus \mathcal{I}'_2 \uplus a : \alpha_{\mathsf{un}}}}{\Gamma\ ;\ \Theta \vdash E^{++}\{(\nu\kappa)C^+\{P_1[\kappa^+/x]\} \mid D^+\{P_2[\kappa^-/y]\}\} \triangleright \Delta, [\kappa^+ : \alpha], [\kappa^- : \overline{\alpha}]; \mathcal{I}''}$$

with

$$\bullet_{\mathcal{S}_5};\ \Gamma\ ;\ \Theta \vdash E \triangleright \Delta, [\kappa^+ : \alpha], [\kappa^- : \overline{\alpha}]; \mathcal{I}'' \tag{B.21}$$

and

$$\mathcal{S}_5 = \Gamma; \Theta \vdash \Delta', \kappa^+ : \alpha, \kappa^- : \overline{\alpha}; \mathcal{I}'_1 \uplus \mathcal{I}'_2 \uplus a : \alpha_{\mathsf{un}}$$

Notice that by Lemma 18, we have $\mathcal{I}'' \sqsubseteq \mathcal{I}'_1 \cup \mathcal{I}'_2 \uplus a : \alpha_{\mathsf{un}}$. Also, observe that by assumption Δ is balanced; therefore, by Definition 20 the resulting typing $\Delta, [\kappa^+ : \alpha], [\kappa^- : \overline{\alpha}]$ is balanced too. This concludes this case.

Case (R:Upd). From Table 2 we have:

$$E\{C\{l^0[P_1]\} \mid D\{l\{(\mathsf{X}).P_2\}\}\} \longrightarrow E\{C\{P_2[^{P_1}/_{\mathsf{X}}]\} \mid D\{\mathbf{0}\}\}$$

By assumption we have $\Gamma\,;\,\Theta \vdash E\{C\{l^0[P_1]\} \mid D\{l\{(\mathsf{X}).P_2\}\}\} \triangleright \Delta;\mathcal{I}$, with Δ balanced. Then, by inversion on typing, using rules (T:Fill), (T:Par), (T:Adapt), and (T:Loc) we infer:

$$\dfrac{\bullet_{\mathcal{S}_0};\Gamma\,;\,\Theta \vdash E \triangleright \Delta;\mathcal{I} \quad \dfrac{\text{(B.23)}\quad\text{(B.24)}}{\Gamma\,;\,\Theta \vdash C\{l^0[P_1]\} \mid D\{l\{(\mathsf{X}).P_2\}\} \triangleright \Delta';\mathcal{I}'}}{\Gamma\,;\,\Theta \vdash E\{C\{l^0[P_1]\} \mid D\{l\{(\mathsf{X}).P_2\}\}\} \triangleright \Delta;\mathcal{I}} \tag{B.22}$$

with $\mathcal{S}_0 = \Gamma;\Theta \vdash \Delta';\mathcal{I}'$. By Lemma 18, we have $\Delta' \subseteq \Delta'$ and $\mathcal{I}' \sqsubseteq \mathcal{I}$. Moreover, letting $\Delta' = \Delta'_1 \cup \Delta'_2$ and $\mathcal{I}' = \mathcal{I}'_1 \uplus \mathcal{I}'_2$, subtree (B.23) is as follows:

$$\dfrac{\bullet_{\mathcal{S}_1};\Gamma\,;\,\Theta \vdash C \triangleright \Delta'_1;\mathcal{I}'_1 \quad \dfrac{\mathcal{I}''_1 \sqsubseteq \mathcal{I}^*_1 \quad \Theta \vdash l:\mathcal{I}^*_1 \quad \Gamma\,;\,\Theta \vdash P_1 \triangleright \emptyset;\mathcal{I}''_1}{\Gamma\,;\,\Theta \vdash l^0[P_1] \triangleright \emptyset;\mathcal{I}''_1}}{\Gamma\,;\,\Theta \vdash C\{l^0[P_1]\} \triangleright \Delta'_1;\mathcal{I}'_1} \tag{B.23}$$

with $\mathcal{S}_1 = \Gamma;\Theta \vdash \emptyset;\mathcal{I}''_1$, and $\mathcal{I}''_1 \sqsubseteq \mathcal{I}'_1$ (by Lemma 18). Subtree (B.24) is as follows:

$$\dfrac{\bullet_{\mathcal{S}_2};\Gamma\,;\,\Theta \vdash D \triangleright \Delta'_2;\mathcal{I}'_2 \quad \dfrac{\Theta \vdash l:\mathcal{I}^*_1 \quad \Gamma\,;\,\Theta,\mathsf{X}:\mathcal{I}^*_1 \vdash P_2 \triangleright \emptyset;\mathcal{I}_3}{\Gamma\,;\,\Theta \vdash l\{(\mathsf{X}).P_2\} \triangleright \emptyset;\emptyset}}{\Gamma\,;\,\Theta \vdash D\{l\{(\mathsf{X}).P_2\}\} \triangleright \Delta'_2;\mathcal{I}'_2} \tag{B.24}$$

with $\mathcal{S}_2 = \Gamma;\Theta \vdash \emptyset;\emptyset$. By Lemma 16(3) we have $\Gamma\,;\,\Theta \vdash P_2[^{P_1}/_{\mathsf{X}}] \triangleright \emptyset;\mathcal{I}'_3$, for some $\mathcal{I}'_3$ such that $\mathcal{I}'_3 \sqsubseteq \mathcal{I}_3$. We now reconstruct the derivation in (B.22), using rules (T:Par), (T:Fill) and Lemma 19(3). Let

$$\dfrac{\dfrac{\bullet_{\mathcal{S}_3};\Gamma\,;\,\Theta \vdash D \triangleright \Delta'_1;\mathcal{I}''_3 \quad \Gamma\,;\,\Theta \vdash P_2[^{P_1}/_{\mathsf{X}}] \triangleright \emptyset;\mathcal{I}'_3}{\Gamma\,;\,\Theta \vdash C\{P_2[^{P_1}/_{\mathsf{X}}]\} \triangleright \Delta'_1;\mathcal{I}''_3} \quad \dfrac{\bullet_{\mathcal{S}_4};\Gamma\,;\,\Theta \vdash D \triangleright \Delta'_2;\mathcal{I}'_2 \quad \Gamma\,;\,\Theta \vdash \mathbf{0} \triangleright \emptyset;\emptyset}{\Gamma\,;\,\Theta \vdash D\{\mathbf{0}\} \triangleright \Delta'_2;\mathcal{I}'_2}}{\Gamma\,;\,\Theta \vdash C\{P_2[^{P_1}/_{\mathsf{X}}]\} \mid D\{\mathbf{0}\} \triangleright \Delta'_1 \cup \Delta'_2;\mathcal{I}''_3 \uplus \mathcal{I}'_2} \tag{B.25}$$

and

$$\dfrac{\bullet_{\mathcal{S}_5};\Gamma\,;\,\Theta \vdash E \triangleright \Delta;\mathcal{I}' \quad \text{(B.25)}}{\Gamma\,;\,\Theta \vdash E\{C\{P_2[^{P_1}/_{\mathsf{X}}]\} \mid D\{\mathbf{0}\}\} \triangleright \Delta;\mathcal{I}'}$$

with

$$\mathcal{S}_5 = \Gamma;\Theta \vdash \Delta'_1 \cup \Delta'_2;\mathcal{I}''_3 \uplus \mathcal{I}'_2$$

where by Lemma 18 we know $\mathcal{I}''_3 \sqsubseteq \mathcal{I}'_3$ and $\mathcal{I}''_3 \uplus \mathcal{I}'_2 \sqsubseteq \mathcal{I}'$. This concludes the analysis for this case.

Case (R:I/O). From Table 2 we have:

$$E\{C\{\kappa^{\,p}\langle\widetilde{e}\rangle.P_1\} \mid D\{\kappa^{\,\overline{p}}(\widetilde{x}).P_2\}\} \longrightarrow E\{C\{P_1\} \mid D\{P_2[^{\widetilde{c}}/_{\widetilde{x}}]\}\} \quad (\widetilde{e}\downarrow\widetilde{c})$$

By assumption, we have $\Gamma\,;\,\Theta \vdash E\{C\{\kappa^{\,p}\langle\widetilde{e}\rangle.P_1\} \mid D\{\kappa^{\,\overline{p}}(\widetilde{x}).P_2\}\} \triangleright \Delta;\mathcal{I}$, with Δ balanced. By inversion on typing, using rules (T:Fill), (T:Par), (T:In), and (T:Out), we infer:

$$\dfrac{\bullet_{\mathcal{S}_0};\Gamma\,;\,\Theta \vdash E \triangleright \Delta;\mathcal{I} \quad \dfrac{\text{(B.29)}\quad\text{(B.30)}}{\Gamma\,;\,\Theta \vdash C\{\kappa^{\,p}\langle\widetilde{e}\rangle.P_1\} \mid D\{\kappa^{\,\overline{p}}(\widetilde{x}).P_2\} \triangleright \Delta';\mathcal{I}'_1 \uplus \mathcal{I}'_2}}{\Gamma\,;\,\Theta \vdash E\{C\{\kappa^{\,p}\langle\widetilde{e}\rangle.P_1\} \mid D\{\kappa^{\,\overline{p}}(\widetilde{x}).P_2\}\} \triangleright \Delta;\mathcal{I}}$$

where:

$$\Delta' = \Delta'_1 \cup \Delta'_2, \kappa^{\,p} :!(\widetilde{\tau}).\alpha, \kappa^{\,\overline{p}} :?(\widetilde{\tau}).\overline{\alpha} \tag{B.26}$$

$$\mathcal{I} = \mathcal{I}'_1 \uplus \mathcal{I}'_2 \tag{B.27}$$

$$\mathcal{S}_0 = \Gamma;\Theta \vdash \Delta';\mathcal{I}'_1 \uplus \mathcal{I}'_2 \tag{B.28}$$

Moreover, by Lemma 18, we infer $\Delta' \subseteq \Delta$ and $\mathcal{I}'_1 \uplus \mathcal{I}'_2 \sqsubseteq \mathcal{I}$. Also, we have that subtree (B.29) is as follows:

$$\dfrac{\bullet_{\mathcal{S}_1};\Gamma\,;\,\Theta \vdash C \triangleright \Delta_1;\mathcal{I}'_2 \quad \dfrac{\Gamma\,;\,\Theta \vdash P_1 \triangleright \Delta_1,\kappa^{\,p}:\alpha;\mathcal{I}_1 \quad \Gamma \vdash \widetilde{e}:\widetilde{\tau}}{\Gamma\,;\,\Theta \vdash \kappa^{\,p}\langle\widetilde{e}\rangle.P_1 \triangleright \Delta_1,\kappa^{\,p} :!(\widetilde{\tau}).\alpha;\mathcal{I}_1}}{\Gamma\,;\,\Theta \vdash C\{\kappa^{\,p}\langle\widetilde{e}\rangle.P_1\} \triangleright \Delta'_1,\kappa^{\,p} :!(\widetilde{\tau}).\alpha;\mathcal{I}'_1} \tag{B.29}$$

with

$$\mathcal{S}_1 = \Gamma; \Theta \vdash \Delta_1, \kappa^p : !(\widetilde{\tau}).\alpha; \mathcal{I}_1$$

Also, subtree (B.30) is as follows:

$$\dfrac{\bullet_{\mathcal{S}_2}; \Gamma;\ \Theta \vdash D \triangleright \Delta_1; \mathcal{I}_2' \qquad \dfrac{\Gamma, \widetilde{x} : \widetilde{\tau};\ \Theta \vdash P_2 \triangleright \Delta_2, \kappa^{\overline{p}} : \overline{\alpha}; \mathcal{I}_2}{\Gamma;\ \Theta \vdash \kappa^{\overline{p}}(\widetilde{x}).P_1 \triangleright \Delta_2, \kappa^{\overline{p}} : ?(\widetilde{\tau}).\overline{\alpha}; \mathcal{I}_2}}{\Gamma;\ \Theta \vdash D\{\kappa^{\overline{p}}(\widetilde{x}).P_2\} \triangleright \Delta_2', \kappa^{\overline{p}} : ?(\widetilde{\tau}).\overline{\alpha}; \mathcal{I}_2'} \tag{B.30}$$

with

$$\mathcal{S}_2 = \Gamma; \Theta \vdash \Delta_2, \kappa^{\overline{p}} : ?(\widetilde{\tau}).\overline{\alpha}; \mathcal{I}_2$$

where Lemma 18 ensures $\Delta_1 \subseteq \Delta_1'$, $\Delta_2 \subseteq \Delta_2'$, $\Delta \subseteq \Delta_1' \cup \Delta_2', \kappa^p : !(\widetilde{\tau}).\alpha, \kappa^{\overline{p}} : ?(\widetilde{\tau}).\overline{\alpha}$, $\mathcal{I}_1 \sqsubseteq \mathcal{I}_1'$, $\mathcal{I}_2 \sqsubseteq \mathcal{I}_2'$, and $\mathcal{I} \sqsubseteq \mathcal{I}_1 \uplus \mathcal{I}_2$.

Now, by Lemma 16(2) we know $\Gamma;\ \Theta \vdash P_2[\widetilde{c}/\widetilde{x}] \triangleright \Delta_2, \kappa^{\overline{p}} : \overline{\alpha}; \mathcal{I}_2$ with $\widetilde{e} \downarrow \widetilde{c}$. Moreover by Lemma 19(3) and rules (T:Par) and (T:Fill) we obtain the following type derivations:

$$\dfrac{\bullet_{\mathcal{S}_3}; \Gamma;\ \Theta \vdash D \triangleright \Delta_1', \kappa^p : \alpha; \mathcal{I}_1' \quad \Gamma;\ \Theta \vdash P_1 \triangleright \Delta_1, \kappa^p : \alpha; \mathcal{I}_1}{\Gamma;\ \Theta \vdash C\{P_1\} \triangleright \Delta_1', \kappa^p : \alpha; \mathcal{I}_1'} \tag{B.31}$$

$$\dfrac{\bullet_{\mathcal{S}_4}; \Gamma;\ \Theta \vdash D \triangleright \Delta_2', \kappa^{\overline{p}} : \overline{\alpha}; \mathcal{I}_2' \quad \Gamma;\ \Theta \vdash P_2[\widetilde{c}/\widetilde{x}] \triangleright \Delta_2, \kappa^{\overline{p}} : \overline{\alpha}; \mathcal{I}_2}{\Gamma;\ \Theta \vdash D\{P_2[\widetilde{c}/\widetilde{x}]\} \triangleright \Delta_2', \kappa^{\overline{p}} : \overline{\alpha}; \mathcal{I}_2'} \tag{B.32}$$

$$\dfrac{\bullet_{\mathcal{S}_5}; \Gamma;\ \Theta \vdash E \triangleright \Delta'; \mathcal{I} \quad \dfrac{\text{(B.31)} \quad \text{(B.32)}}{\Gamma;\ \Theta \vdash C\{P_1\} \mid D\{P_2[\widetilde{c}/\widetilde{x}]\} \triangleright \Delta_1' \cup \Delta_2', \kappa^p : \alpha, \kappa^{\overline{p}}\overline{\alpha}; \mathcal{I}_1' \uplus \mathcal{I}_2'}}{\Gamma;\ \Theta \vdash E\{C\{P_1\} \mid D\{P_2[\widetilde{c}/\widetilde{x}]\}\} \triangleright \Delta'; \mathcal{I}}$$

with

$$\mathcal{S}_3 = \Gamma; \Theta \vdash \Delta_1, \kappa^p : \alpha; \mathcal{I}_1$$

$$\mathcal{S}_4 = \Gamma; \Theta \vdash \Delta_2, \kappa^{\overline{p}} : \overline{\alpha}; \mathcal{I}_2$$

$$\mathcal{S}_5 = \Gamma; \Theta \vdash \Delta_1' \cup \Delta_2', \kappa^p : \alpha, \kappa^{\overline{p}}\overline{\alpha}; \mathcal{I}_1' \uplus \mathcal{I}_2'$$

Since by inductive hypothesis Δ_1' and Δ_2' are balanced, we infer that $\Delta_1' \cup \Delta_2', \kappa^p : \alpha, \kappa^{\overline{p}} : \overline{\alpha}$ is balanced as well; this concludes the proof for this case.

Case (R:Pass). From Table 2 we have:

$$E\{C\{\kappa^p\langle\langle\kappa_1^q\rangle\rangle.P_1\} \mid D\{\kappa^{\overline{p}}((x)).P_2\}\} \longrightarrow E\{C^-\{P_1\} \mid D^+\{P_2[\kappa_1^q/x]\}\}$$

By assumption we have $\Gamma;\ \Theta \vdash E\{C\{\kappa^p\langle\langle\kappa_1^q\rangle\rangle.P_1\} \mid D\{\kappa^{\overline{p}}((x)).P_2\}\} \triangleright \Delta; \mathcal{I}$, with Δ balanced. By typing inversion on rules (T:Fill), (T:Par), (T:Cat), and (T:Thr) we infer:

$$\dfrac{\bullet_{\mathcal{S}_0}; \Gamma;\ \Theta \vdash E \triangleright \Delta; \mathcal{I} \quad \dfrac{\text{(B.36)} \quad \text{(B.38)}}{\Gamma;\ \Theta \vdash C\{\kappa^p\langle\langle\kappa_1^q\rangle\rangle.P_1\} \mid D\{\kappa^{\overline{p}}((x)).P_2\} \triangleright \Delta'; \mathcal{I}'}}{\Gamma;\ \Theta \vdash E\{C\{\kappa^p\langle\langle\kappa_1^q\rangle\rangle.P_1\} \mid D\{\kappa^{\overline{p}}((x)).P_2\}\} \triangleright \Delta; \mathcal{I}}$$

with:

$$\Delta = \Delta_1, \kappa^p : !(\alpha).\overline{\beta}, \kappa_1^q : \alpha, \Delta_2, \kappa^{\overline{p}} : ?(\alpha).\beta \tag{B.33}$$

$$\Delta' = \Delta_1', \kappa^p : !(\alpha).\overline{\beta}, \kappa_1^q : \alpha, \Delta_2', , \kappa^{\overline{p}} : ?(\alpha).\beta \tag{B.34}$$

$$\mathcal{S}_0 = \Gamma; \Theta \vdash \Delta'; \mathcal{I}' \tag{B.35}$$

and, by Lemma 18, we infer $\Delta_1' \subseteq \Delta_1$, $\Delta_2' \subseteq \Delta_2$, and $\mathcal{I}' \sqsubseteq \mathcal{I}$. Moreover, (B.36) corresponds to the subtree:

$$\dfrac{\bullet_{\mathcal{S}_1}; \Gamma;\ \Theta \vdash C \triangleright \Delta_1', \kappa^p : !(\alpha).\overline{\beta}, \kappa_1^q : \alpha; \mathcal{I}_1' \quad \dfrac{\Gamma;\ \Theta \vdash P_1 \triangleright \Delta_1'', \kappa^p : \overline{\beta}; \mathcal{I}_1''}{\Gamma;\ \Theta \vdash \kappa^p\langle\langle\kappa_1^q\rangle\rangle.P_1 \triangleright \Delta_1'', \kappa^p : !(\alpha).\overline{\beta}, \kappa_1^q : \alpha; \mathcal{I}_1''}}{\Gamma;\ \Theta \vdash C\{\kappa^p\langle\langle\kappa_1^q\rangle\rangle.P_1\} \triangleright \Delta_1', \kappa^p : !(\alpha).\overline{\beta}, \kappa_1^q : \alpha; \mathcal{I}_1'} \tag{B.36}$$

with $\Delta_1'' \subseteq \Delta_1'$ and $\mathcal{I}_1'' \sqsubseteq \mathcal{I}_1'$ (by Lemma 18) and

$$\mathcal{S}_1 = \Gamma; \Theta \vdash \Delta_1'', \kappa^p :!(\alpha).\overline{\beta}, \kappa_1^q : \alpha; \mathcal{I}_1'' \tag{B.37}$$

while (B.38) corresponds to the subtree:

$$\frac{\bullet_{\mathcal{S}_2}; \Gamma\,;\, \Theta \vdash D \triangleright \Delta_2', \kappa^{\overline{p}} :?(\alpha).\beta; \mathcal{I}_2' \quad \dfrac{\Gamma\,;\, \Theta \vdash P_2 \triangleright \Delta_2'', \kappa^{\overline{p}} : \beta, x : \alpha; \mathcal{I}_2''}{\Gamma\,;\, \Theta \vdash \kappa^{\overline{p}}((x)).P_2 \triangleright \Delta_2'', \kappa^{\overline{p}} :?(\alpha).\beta; \mathcal{I}_2''}}{\Gamma\,;\, \Theta \vdash D\{\kappa^{\overline{p}}((x)).P_2\} \triangleright \Delta_2', \kappa^{\overline{p}} :?(\alpha).\beta; \mathcal{I}_2'} \tag{B.38}$$

with $\Delta_2'' \subseteq \Delta_2'$ and $\mathcal{I}_2'' \sqsubseteq \mathcal{I}_2'$ (by Lemma 18) and

$$\mathcal{S}_2 = \Gamma; \Theta \vdash \Delta_2'', \kappa^{\overline{p}} :?(\alpha).\beta; \mathcal{I}_2'' \tag{B.39}$$

We now describe how to obtain appropriately typed contexts C^- and D^+, based on the information already inferred on contexts C and D. We first consider the case of C^-. From (B.36), we obtained

$$\bullet_{\mathcal{S}_1}; \Gamma\,;\, \Theta \vdash C \triangleright \Delta_1', \kappa^p :!(\alpha).\overline{\beta}, \kappa_1^q : \alpha; \mathcal{I}_1'$$

with $\mathcal{S}_1$ as in (B.37). Then, using Lemma 19(2), we infer

$$\bullet_{\mathcal{S}_3}; \Gamma\,;\, \Theta \vdash C^- \triangleright \Delta_1', \kappa^p : \overline{\beta}; \mathcal{I}_1'$$

where

$$\mathcal{S}_3 = \Gamma; \Theta \vdash \Delta_1'', \kappa^p : \overline{\beta}; \mathcal{I}_1'' \tag{B.40}$$

We may now reconstruct the derivation in (B.36), as follows:

$$\frac{\bullet_{\mathcal{S}_3}; \Gamma\,;\, \Theta \vdash C^- \triangleright \Delta_1', \kappa^p : \overline{\beta}; \mathcal{I}_1' \quad \Gamma\,;\, \Theta \vdash P_1 \triangleright \Delta_1'', \kappa^p : \overline{\beta}; \mathcal{I}_1''}{\Gamma\,;\, \Theta \vdash C^-\{P_1\} \triangleright \Delta_1', \kappa^p : \overline{\beta}; \mathcal{I}_1'} \tag{B.41}$$

We now consider the case of D^+. By applying Lemma 16(1) on the premise concerning P_2 in (B.38), we obtain

$$\Gamma\,;\, \Theta \vdash P_2[\kappa_1^q/x] \triangleright \Delta_2'', \kappa^{\overline{p}} : \beta, \kappa_1^q : \alpha; \mathcal{I}_2''$$

From (B.38) we obtained

$$\bullet_{\mathcal{S}_2}; \Gamma\,;\, \Theta \vdash D \triangleright \Delta_2', \kappa^{\overline{p}} :?(\alpha).\beta; \mathcal{I}_2'$$

with $\mathcal{S}_2$ as in (B.39). Then, using Lemma 19(1), we infer

$$\bullet_{\mathcal{S}_4}; \Gamma\,;\, \Theta \vdash D^+ \triangleright \Delta_2', \kappa^{\overline{p}} : \beta, \kappa_1^q : \alpha; \mathcal{I}_2'$$

where

$$\mathcal{S}_4 = \Gamma; \Theta \vdash \Delta_2'', \kappa^{\overline{p}} : \beta, \kappa_1^q : \alpha; \mathcal{I}_2'' \tag{B.42}$$

We can reconstruct the derivation depicted by (B.38):

$$\frac{\bullet_{\mathcal{S}_4}; \Gamma\,;\, \Theta \vdash D^+ \triangleright \Delta_2', \kappa^{\overline{p}} : \beta, \kappa_1^q : \alpha; \mathcal{I}_2' \quad \Gamma\,;\, \Theta \vdash P_2[\kappa_1^q/x] \triangleright \Delta_2'', \kappa^+ : \beta, \kappa_1^q : \alpha; \mathcal{I}_2''}{\Gamma\,;\, \Theta \vdash D^+\{P_2[\kappa_1^q/x]\} \triangleright \Delta_2', \kappa^{\overline{p}} : \beta, \kappa_1^q : \alpha; \mathcal{I}_2'} \tag{B.43}$$

Combining (B.41) and (B.43), we may finally derive the type for the result of the reduction. Using rules (T:Par) and (T:Fill) we obtain:

$$\frac{\bullet_{\mathcal{S}_5}; \Gamma\,;\, \Theta \vdash E \triangleright \Delta^*; \mathcal{I} \quad \dfrac{\text{(B.41)} \quad \text{(B.43)}}{\Gamma\,;\, \Theta \vdash C^-\{P_1\} \mid D^+\{P_2[\kappa_1^q/x]\} \triangleright \Delta_1' \cup \Delta_2', \kappa^p : \overline{\beta}, \kappa^{\overline{p}} : \beta, \kappa_1^q : \alpha; \mathcal{I}'}}{\Gamma\,;\, \Theta \vdash E\{C^-\{P_1\} \mid D^+\{P_2[\kappa_1^q/x]\}\} \triangleright \Delta^*; \mathcal{I}}$$

with

$$\mathcal{S}_5 = \Gamma; \Theta \vdash \Delta_1' \cup \Delta_2', \kappa^p : \overline{\beta}, \kappa^{\overline{p}} : \beta, \kappa_1^q : \alpha; \mathcal{I}'$$

Since by assumption Δ is balanced, we have that by construction Δ^* is balanced as well. It is worth observing how contexts C^- and D^+ correctly implement the fact that the number of active sessions is changed after delegating session κ_1^q to process P_2. This concludes the proof for this case.

Cases (R:IfTr) and (R:IfFa). Follows by an ease induction on the derivation tree.

Case (R:Close). These follow by the same reasoning as in (R:Open) case.

Case (R:Branch). This case is similar to previous (R:I/O) case.

Case (R:Str). Follows from Theorem 15 (Subject Congruence).

Case (R:Par). Follows by induction and by applying rule (T:Par).

Case (R:Res). Follows by induction and by the fact that Δ is balanced. Indeed, by hypothesis and by inversion on rule (T:CRes) all the occurrences of bracketed assignments ($[\kappa^{p} : \alpha]$) are necessarily balanced thus making it possible to apply the inductive hypothesis to the premise of the rule and concluding the analysis of this case and the proof of the theorem. □

Appendix C. Additional material for Section 6

Table C.9
Reduction semantics without annotations.

(R:LPar)	if $P \longrightarrow P'$ then $P \mid Q \longrightarrow P' \mid Q$
(R:LRes)	if $P \longrightarrow P'$ then $(\nu\kappa)P \longrightarrow (\nu\kappa)P'$
(R:LStr)	if $P \equiv P'$, $P' \longrightarrow Q'$, and $Q' \equiv Q$ then $P \longrightarrow Q$
(R:LLoc)	if $P \longrightarrow P'$ then $l[P] \longrightarrow l[P']$
(R:LOpen)	$C\{\mathsf{accept}\, a(x).P\} \mid D\{\mathsf{request}\, a(y).Q\} \longrightarrow (\nu\kappa)(C\{P[\kappa^{+}/x]\} \mid D\{Q[\kappa^{-}/y]\})$
(R:LROpen)	$C\{!\,\mathsf{accept}\, a(x).P\} \mid D\{\mathsf{request}\, a(y).Q\} \longrightarrow (\nu\kappa)(C\{P[\kappa^{+}/x] \mid !\,\mathsf{accept}\, a(x).P\} \mid D\{Q[\kappa^{-}/y]\})$
(R:LUpd)	$C\{l[P]\} \mid D\{l\{(X).Q\}\} \longrightarrow C\{Q[^{P}/_{\mathsf{X}}]\} \mid D\{\mathbf{0}\}$
(R:LI/O)	$C\{\kappa^{p}\langle\widetilde{e}\rangle.P\} \mid D\{\kappa^{\bar{p}}(\widetilde{x}).Q\} \longrightarrow C\{P\} \mid D\{Q[\widetilde{c}/\widetilde{x}]\} \quad (\widetilde{e} \downarrow \widetilde{c})$
(R:LPass)	$C\{\kappa^{p}\langle\langle\kappa'^{q}\rangle\rangle.P\} \mid D\{\kappa^{\bar{p}}((x)).Q\} \longrightarrow C\{P\} \mid D\{Q[\kappa'^{q}/x]\}$
(R:LSel)	$C\{\kappa^{p} \rhd \{n_1 : P_1 \,[]\, \cdots \,[]\, n_m : P_m\}\} \mid D\{\kappa^{\bar{p}} \lhd n_j; Q\} \longrightarrow C\{P_j\} \mid D\{Q\} \quad (1 \le j \le m)$
(R:LClose)	$C\{\mathsf{close}\,(\kappa^{p}).P\} \mid D\{\mathsf{close}\,(\kappa^{\bar{p}}).Q\} \longrightarrow C\{P\} \mid D\{Q\}$
(R:LIfTr)	if e then P else $Q \longrightarrow P \quad (e \downarrow \mathtt{true})$
(R:LIfFa)	if e then P else $Q \longrightarrow Q \quad (e \downarrow \mathtt{false})$

Table C.10
Typed reduction semantics (I).

(R:ParU)	$\dfrac{\Gamma\,;\,\Theta \vdash P \triangleright \Delta_1\,;\,\mathcal{I}_1 \longrightarrow \Gamma\,;\,\Theta \vdash P' \triangleright \Delta_1'\,;\,\mathcal{I}_1'}{\Gamma\,;\,\Theta \vdash P \mid Q \triangleright \Delta_1 \cup \Delta_2\,;\,\mathcal{I}_1 \cup \mathcal{I}_2 \longrightarrow \Gamma\,;\,\Theta \vdash P' \mid Q \triangleright \Delta_1' \cup \Delta_2\,;\,\mathcal{I}_1' \uplus \mathcal{I}_2'}$
(R:ResU)	$\dfrac{\Gamma\,;\,\Theta \vdash P \triangleright \kappa^{+} : \alpha, \kappa^{-} : \overline{\alpha}, \Delta\,;\,\mathcal{I} \longrightarrow \Gamma\,;\,\Theta \vdash P' \triangleright \kappa^{+} : \alpha', \kappa^{-} : \overline{\alpha'}, \Delta'\,;\,\mathcal{I}'}{\begin{array}{c}\Gamma\,;\,\Theta \vdash (\nu\kappa)P \triangleright [\kappa^{+} : \alpha], [\kappa^{-} : \overline{\alpha}], \Delta\,;\,\mathcal{I} \longrightarrow \\ \Gamma\,;\,\Theta \vdash (\nu\kappa)P' \triangleright [\kappa^{+} : \alpha'], [\kappa^{-} : \overline{\alpha'}], \Delta'\,;\,\mathcal{I}'\end{array}}$
(R:StrU)	$\dfrac{P \equiv Q \quad \Gamma\,;\,\Theta \vdash Q \triangleright \Delta\,;\,\mathcal{I} \longrightarrow \Gamma\,;\,\Theta \vdash Q' \triangleright \Delta'\,;\,\mathcal{I}' \quad P' \equiv Q'}{\Gamma\,;\,\Theta \vdash P \triangleright \Delta\,;\,\mathcal{I} \longrightarrow \Gamma\,;\,\Theta \vdash P' \triangleright \Delta'\,;\,\mathcal{I}'}$
(R:LocU)	$\dfrac{\Gamma\,;\,\Theta \vdash P \triangleright \Delta\,;\,\mathcal{I} \longrightarrow \Gamma\,;\,\Theta \vdash P' \triangleright \Delta'\,;\,\mathcal{I}'}{\Gamma\,;\,\Theta \vdash l[P] \triangleright \Delta\,;\,\mathcal{I} \longrightarrow \Gamma\,;\,\Theta \vdash l[P'] \triangleright \Delta'\,;\,\mathcal{I}'}$
(R:UpdU)	$\dfrac{\Gamma\,;\,\Theta \vdash P \triangleright \Delta_1\,;\,\mathcal{I}_1 \quad \Gamma\,;\,\Theta, \mathsf{X} : \Delta_1, \mathcal{I}_1 \vdash Q \triangleright \Delta_2\,;\,\mathcal{I}_2 \quad \Delta_1 = \rho(\Delta_2)}{\begin{array}{c}\Gamma\,;\,\Theta \vdash C\{l[P]\} \mid D\{l\{(X).Q\}\} \triangleright \Delta\,;\,\mathcal{I} \\ \longrightarrow \\ \Gamma\,;\,\Theta \vdash C\{\rho(Q)[^{P}/_{\mathsf{X}}]\} \mid D\{\mathbf{0}\} \triangleright \Delta\,;\,(\mathcal{I} \setminus \mathcal{I}_1) \uplus \mathcal{I}_2\end{array}}$
(R:IfTrU)	$\Gamma\,;\,\Theta \vdash \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta\,;\,\mathcal{I} \longrightarrow \Gamma\,;\,\Theta \vdash P \triangleright \Delta\,;\,\mathcal{I} \quad (e \downarrow \mathtt{true})$
(R:IfFaU)	$\Gamma\,;\,\Theta \vdash \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta\,;\,\mathcal{I} \longrightarrow \Gamma\,;\,\Theta \vdash Q \triangleright \Delta\,;\,\mathcal{I} \quad (e \downarrow \mathtt{false})$

Table C.11
Typed reduction semantics (II).

(R:OpenU)
$$\Gamma;\ \Theta \vdash C\{\mathsf{accept}\, a(x).P\} \mid D\{\mathsf{request}\, a(y).Q\} \triangleright \Delta; \mathcal{I}, a:\alpha_{\mathsf{lin}}, a:\overline{\alpha}_{\mathsf{lin}} \longrightarrow \Gamma;\ \Theta \vdash (\nu\kappa)C\{P[\kappa^+/x]\} \mid D\{Q[\kappa^-/y]\} \triangleright \Delta, [\kappa^+:\alpha], [\kappa^-:\overline{\alpha}]; \mathcal{I}$$

(R:ROpenU)
$$\Gamma;\ \Theta \vdash C\{!\,\mathsf{accept}\, a(x).P\} \mid D\{\mathsf{request}\, a(y).Q\} \triangleright \Delta; \mathcal{I}, a:\alpha_{\mathsf{un}}, a:\overline{\alpha}_{\mathsf{lin}} \longrightarrow \Gamma;\ \Theta \vdash (\nu\kappa)C\{P[\kappa^+/x]\} \mid !\,\mathsf{accept}\, a(x).P\} \mid D\{Q[\kappa^-/y]\} \triangleright \Delta, [\kappa^+:\alpha], [\kappa^-:\overline{\alpha}]; \mathcal{I}, a:\alpha_{\mathsf{un}}$$

(R:I/OU)
$$\Gamma;\ \Theta \vdash C\{\kappa^{\,p}\langle\widetilde{e}\rangle.P\} \mid D\{\kappa^{\,\overline{p}}(\widetilde{x}).Q\} \triangleright \Delta, \kappa^{\,p}:!(\widetilde{\tau}).\alpha, \kappa^{\,\overline{p}}:?(\widetilde{\tau}).\overline{\alpha}; \mathcal{I} \longrightarrow \Gamma;\ \Theta \vdash C\{P\} \mid D\{Q[\widetilde{C}/\widetilde{x}]\} \triangleright \Delta, \kappa^{\,p}:\alpha, \kappa^{\,\overline{p}}:\overline{\alpha}; \mathcal{I}$$

(R:PassU)
$$\Gamma;\ \Theta \vdash C\{\kappa^{\,p}\langle\langle\kappa'^{\,q}\rangle\rangle.P\} \mid D\{\kappa^{\,\overline{p}}((x)).Q\} \triangleright \Delta, \kappa^{\,p}:!(\alpha).\beta, \kappa'^{\,q}:\alpha, \kappa^{\,\overline{p}}:?(\alpha).\overline{\beta}; \mathcal{I} \longrightarrow \Gamma;\ \Theta \vdash C\{P\} \mid D\{Q[\kappa'^{\,q}/x]\} \triangleright \Delta, \kappa^{\,p}:\beta, \kappa^{\,\overline{p}}:\overline{\beta}, \kappa'^{\,q}:\alpha; \mathcal{I}$$

(R:SelU)
$$\Gamma;\ \Theta \vdash C\{\kappa^{\,p} \rhd \{n_1{:}P_1 \,[\!]\, \cdots \,[\!]\, n_m{:}P_m\}\} \mid D\{\kappa^{\,\overline{p}} \lhd n_j;\ Q\} \triangleright \Delta, \kappa^{\,p}:\&\{n_1{:}\alpha_1, \ldots, n_k{:}\alpha_m\}, \kappa^{\,\overline{p}}:\oplus\{n_1:\overline{\alpha_1}, \ldots, n_m:\overline{\alpha_m}\}; \mathcal{I} \longrightarrow \Gamma;\ \Theta \vdash C\{P_j\} \mid D\{Q\} \triangleright \Delta, \kappa^{\,p}:\alpha_j, \kappa^{\,\overline{p}}:\overline{\alpha_j}; \mathcal{I}$$

(R:CloseU)
$$\Gamma;\ \Theta \vdash C\{\mathsf{close}\,(\kappa^{\,p}).P\} \mid D\{\mathsf{close}\,(\kappa^{\,\overline{p}}).Q\} \triangleright \Delta, \kappa^{\,p}:\epsilon, \kappa^{\,\overline{p}}:\epsilon; \mathcal{I} \longrightarrow \Gamma;\ \Theta \vdash C\{P\} \mid D\{Q\} \triangleright \Delta; \mathcal{I}$$

References

[1] G. Anderson, J. Rathke, Dynamic software update for message passing programs, in: R. Jhala, A. Igarashi (Eds.), APLAS, in: Lecture Notes in Computer Science, vol. 7705, Springer, 2012, pp. 207–222.
[2] M. Bravetti, M. Carbone, T. Hildebrandt, I. Lanese, J. Mauro, J.A. Pérez, G. Zavattaro, Towards global and local types for adaptation, in: SEFM 2013 Collocated Workshops, in: Lecture Notes in Computer Science, vol. 8368, Springer, 2014.
[3] M. Bravetti, C. Di Giusto, J.A. Pérez, G. Zavattaro, Adaptable processes, Log. Methods Comput. Sci. 8 (4) (2012). Extended abstract in: Proc. of FMOODS-FORTE'11, in: LNCS, vol. 6722, Springer, 2012.
[4] A. Brogi, C. Canal, E. Pimentel, Behavioural types and component adaptation, in: C. Rattray, S. Maharaj, C. Shankland (Eds.), AMAST, in: LNCS, vol. 3116, Springer, 2004, pp. 42–56.
[5] R. Bruni, A. Corradini, F. Gadducci, A. Lluch-Lafuente, A. Vandin, A conceptual framework for adaptation, in: J. de Lara, A. Zisman (Eds.), FASE, in: Lecture Notes in Computer Science, vol. 7212, Springer, 2012, pp. 240–254.
[6] M. Bugliesi, G. Castagna, S. Crafa, Access control for mobile agents: the calculus of boxed ambients, ACM Trans. Program. Lang. Syst. 26 (1) (2004) 57–124.
[7] S. Capecchi, I. Castellani, M. Dezani-Ciancaglini, T. Rezk, Session types for access and information flow control, in: P. Gastin, F. Laroussinie (Eds.), CONCUR, in: Lecture Notes in Computer Science, vol. 6269, Springer, 2010, pp. 237–252.
[8] M. Carbone, S. Debois, A graphical approach to progress for structured communication in web services, in: S. Bliudze, R. Bruni, D. Grohmann, A. Silva (Eds.), ICE, in: EPTCS, vol. 38, 2010, pp. 13–27.
[9] M. Carbone, K. Honda, N. Yoshida, Structured communication-centred programming for web services, in: R. De Nicola (Ed.), ESOP, in: Lecture Notes in Computer Science, vol. 4421, Springer, 2007, pp. 2–17.
[10] M. Carbone, K. Honda, N. Yoshida, Structured interactional exceptions in session types, in: CONCUR, in: LNCS, vol. 5201, Springer, 2008, pp. 402–417.
[11] M. Coppo, M. Dezani-Ciancaglini, B. Venneri, Self-adaptive monitors for multiparty sessions, in: PDP'14, 2014, http://doi.ieeecomputersociety.org/10.1109/PDP.2014.18, in press.
[12] M. Dezani-Ciancaglini, U. de'Liguoro, Sessions and session types: an overview, in: WS-FM, in: LNCS, vol. 6194, Springer, 2009, pp. 1–28.
[13] M. Dezani-Ciancaglini, U. de'Liguoro, N. Yoshida, On progress for structured communications, in: G. Barthe, C. Fournet (Eds.), TGC, in: Lecture Notes in Computer Science, vol. 4912, Springer, 2007, pp. 257–275.
[14] C. Di Giusto, J.A. Pérez, Disciplined structured communications with consistent runtime adaptation, in: SAC, ACM, 2013, pp. 1913–1918.
[15] C. Ferreira, I. Lanese, A. Ravara, H.T. Vieira, G. Zavattaro, Advanced mechanisms for service combination and transactions, in: Results of the SENSORIA Project, in: LNCS, vol. 6582, Springer, 2011, pp. 302–325.
[16] P. Garralda, A.B. Compagnoni, M. Dezani-Ciancaglini, Bass: boxed ambients with safe sessions, in: PPDP, ACM, 2006, pp. 61–72.
[17] S.J. Gay, M. Hole, Subtyping for session types in the pi calculus, Acta Inform. 42 (2–3) (2005) 191–225.
[18] A.S. Henriksen, L. Nielsen, T.T. Hildebrandt, N. Yoshida, F. Henglein, Trustworthy pervasive healthcare services via multiparty session types, in: J. Weber, I. Perseil (Eds.), FHIES, in: Lecture Notes in Computer Science, vol. 7789, Springer, 2012, pp. 124–141.
[19] K. Honda, Types for dyadic interaction, in: CONCUR, in: LNCS, vol. 715, Springer, 1993, pp. 509–523.
[20] K. Honda, V.T. Vasconcelos, M. Kubo, Language primitives and type discipline for structured communication-based programming, in: ESOP, in: LNCS, vol. 1381, Springer, 1998, pp. 122–138.
[21] K. Honda, N. Yoshida, M. Carbone, Multiparty asynchronous session types, in: G.C. Necula, P. Wadler (Eds.), POPL, ACM, 2008, pp. 273–284.
[22] D. Kouzapas, N. Yoshida, K. Honda, On asynchronous session semantics, in: FMOODS/FORTE, in: LNCS, vol. 6722, Springer, 2011, pp. 228–243.
[23] S. Lenglet, A. Schmitt, J.-B. Stefani, Characterizing contextual equivalence in calculi with passivation, Inf. Comput. 209 (11) (2011) 1390–1433.
[24] R. Milner, J. Parrow, D. Walker, A calculus of mobile processes, I, Inf. Comput. 100 (1) (1992) 1–40.
[25] D. Mostrous, N. Yoshida, Two session typing systems for higher-order mobile processes, in: TLCA, in: LNCS, vol. 4583, Springer, 2007, pp. 321–335.
[26] D. Sangiorgi, Expressing mobility in process algebras: first-order and higher-order paradigms, PhD thesis CST-99-93, Department of Computer Science, University of Edinburgh, 1992.

[27] G. Stoyle, M.W. Hicks, G.M. Bierman, P. Sewell, I. Neamtiu, *Mutatis Mutandis*: safe and predictable dynamic software updating, ACM Trans. Program. Lang. Syst. 29 (4) (2007).
[28] A. Vallecillo, V.T. Vasconcelos, A. Ravara, Typing the behavior of objects and component using session types, Electron. Notes Theor. Comput. Sci. 68 (3) (2003) 439–456.
[29] N. Yoshida, V.T. Vasconcelos, Language primitives and type discipline for structured communication-based programming revisited: two systems for higher-order session communication, Electron. Notes Theor. Comput. Sci. 171 (4) (2007) 73–93.