Dynamic Roles in Multiparty Communicating Systems

Pedro Baltazar
Instituto de Telecomunicações, Technical University of Lisbon

Luís Caires
CITI-DI, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa

Vasco T. Vasconcelos
Universidade de Lisboa, Faculdade de Ciências, LaSIGE

Hugo T. Vieira
CITI-DI, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa

February 21, 2012,
Revised June 20, 2012

Abstract

Communication protocols in distributed systems often specify the roles of the parties involved in the communications, e.g., for enforcing security policies or task assignment purposes. Ensuring that implementations follow role-based protocol specifications is challenging, especially in scenarios found, e.g., in business processes and web applications, where multiple peers are involved, single peers participate in several roles, or single roles are carried out by several peers. We present a type-based analysis for statically verifying role-based multiparty interactions, based on a simple π-calculus model and prior work on conversation types. Our main result ensures well-typed systems follow the role-based protocols prescribed by the types, and addresses systems where roles have dynamic distributed implementations.

1 Introduction

Communication is a central feature of nowadays software systems, as more and more often systems are built using computational resources that are concurrently available and distributed in the web. Examples range from operating systems where functionality is distributed between distinct threads in the system, to services available on the Internet which rely on third-party (remote) service providers to carry out subsidiary tasks, following the emerging model of SaaS (software as a service) and cloud computing. Building software from the composition of communicating interacting pieces is very flexible, at least in principle, since resources can be dynamically discovered and chosen according to criteria such as declared functionality, availability and work load. In such a setting, all interacting parties must agree on communication protocols without relying on centralized control. Verification mechanisms that automatically check whether the code meets some common protocol specification become then of crucial importance.

A protocol specification describes a set of message exchanges, recording when these should occur as well as the parties involved in the interaction. A party involved in a protocol may have a spatial meaning, for instance denoting a distinguished site or process, or, more generally, a party may have a behavioral meaning, a *role* in the interaction that may be realized by one or more processes or sites. Conversely, a process may impersonate different roles throughout its execution. Such flexibility is essential to address systems, e.g., where a leader role is impersonated by different sites at different stages of the protocol, and the role of each site changes accordingly.

A challenge that arises is then to devise techniques to verify whether a system complies to a protocol specification, given such dynamic and distributed implementation of roles, just by inspecting the source code. A particular situation where roles must be traced is when checking conformance against security policies like, for example, those involving separation of duties.

In this paper we present a type-based analysis for verifying if systems defined in a model programming language follow the role-based protocol descriptions as prescribed by types. Our development is based on conversation type theory [2, 3], extending it with the ability to specify and analyze the roles involved in the interactions. The underlying model of our analysis is based an extremely parsimonious extension to the π-calculus [13], where communication actions specify a message label and the role performing the action, inspired by TyCO [14]. Conversations generalize sessions [9, 11] with support to multiparty interaction, addressing dynamically established collaborations between an unanticipated number of partners. A distinguishing feature of the conversation types approach is that multiple parties interact using labeled messages in a single medium of communication, while other works support multiparty communication via message queues [10] and indexed communication channels [1]. We choose to adopt the simplest possible setting where session-like multiparty interaction may be studied, and extend it in a minimal way so as to support general reasoning about roles. So, apart from retaining the simplicity of conversation types, our theory addresses systems where a single role may be realized by several parties and where processes may dynamically change the role on behalf of which they are interacting, as needed to model communicating workflows as present in actual business processes. This contrasts with related approaches (see, e.g., [6, 10]) where roles have a “spatial” meaning, as they are mapped into the structure of systems or sites in a static way.

In the remainder of this section we informally describe our type analysis by going through some examples. Consider the protocol specification given by type:

$$\mathsf{Sender} \to \mathsf{Receiver}\ hello().\mathsf{Sender} \to \mathsf{Receiver}\ bye()$$

which captures a binary interaction where messages *hello* and *bye* are sequentially exchanged, and the communicating partners are identified by Sender and Receiver which send and receive the messages, respectively (read $\to$ as "sends to"). A non surprising implementation of this interaction is given by:

$$chat \triangleleft_{\mathsf{Sender}}\ hello().chat \triangleleft_{\mathsf{Sender}}\ bye() \mid chat \triangleright_{\mathsf{Receiver}}\ hello().chat \triangleright_{\mathsf{Receiver}}\ bye()$$

where two concurrent processes interact on channel *chat* following the protocol above. The process on the left sends the two messages under role Sender ($\triangleleft_{\mathsf{Sender}}$), as described by type $!\mathsf{Sender}\ hello().\ !\mathsf{Sender}\ bye()$, while the process on the right receives the two messages under role Receiver ($\triangleright_{\mathsf{Receiver}}$), described by type $?\mathsf{Receiver}\ hello().\ ?\mathsf{Receiver}\ bye()$.

In this first example there is a perfect match between processes and the roles under which the processes interact. However, this does not need to be the case. Consider a different implementation of the same protocol:

$$chat \triangleleft_{\mathsf{Sender}}\ hello().chat \triangleright_{\mathsf{Receiver}}\ bye() \mid chat \triangleright_{\mathsf{Receiver}}\ hello().chat \triangleleft_{\mathsf{Sender}}\ bye()$$

where the process on the left sends message *hello* as Sender and then receives message *bye* as Receiver, described by type $!\mathsf{Sender}\ hello().\ ?\mathsf{Receiver}\ bye()$, and the process on the right first acts first as Receiver and then as Sender, described by type $?\mathsf{Receiver}\ hello().\ !\mathsf{Sender}\ bye()$. Notice each role is carried out by two distinct processes and each process implements two distinct roles.

Our type analysis ensures that both implementations follow the prescribed protocol, since the protocol

$\mathsf{Sender} \to \mathsf{Receiver}\ hello().\mathsf{Sender} \to \mathsf{Receiver}\ bye()$

is decomposed in "complementary" types that describe the behavior of the individual processes (for instance, in type $!\mathsf{Sender}\ hello().\ ?\mathsf{Receiver}\ bye()$ and type $?\mathsf{Receiver}\ hello().\ !\mathsf{Sender}\ bye()$). Although very simple, this example already distinguishes our approach from previous works, since the ability to specify roles is absent in [2, 3] while [6, 10] do not support such role distribution. Conceivably channel delegation (channel-passing) supported by previous works may be used to represent a similar notion but, for this example in particular, two channel delegations would be involved, it would not be possible to directly observe that the two interactions take place in a related medium (in our case the *chat* channel) and the ability to audit role participation locally would be lost (as the personification of a different role would be a consequence of channel-passing).

$$
\begin{array}{lcl}
\texttt{Buyer} & \triangleq & (\mathbf{new}\ chat) \\
 & & \quad Shop \triangleleft_{\mathsf{Buyer}} buyService(chat). \\
 & & \qquad chat \triangleleft_{\mathsf{Buyer}} buy(). \\
 & & \qquad\quad (chat \triangleright_{\mathsf{Buyer}} price() \mid MailBox \triangleleft_{\mathsf{Buyer}} storeService(chat)) \\
\texttt{Shop} & \triangleq & Shop \triangleright_{\mathsf{Shop}} buyService(x). \\
 & & \qquad x \triangleright_{\mathsf{Seller}} buy(). \\
 & & \qquad x \triangleleft_{\mathsf{Seller}} price(). \\
 & & \qquad\quad (x \triangleleft_{\mathsf{Seller}} product() \mid x \triangleright_{\mathsf{Shipper}} product().x \triangleleft_{\mathsf{Shipper}} details()) \\
\texttt{Mail} & \triangleq & MailBox \triangleright_{\mathsf{Mail}} storeService(x). \\
 & & \qquad x \triangleright_{\mathsf{Buyer}} details() \\
\texttt{System} & \triangleq & (*\texttt{Buyer} \mid *\texttt{Mail} \mid *\texttt{Shop})
\end{array}
$$

Figure 1: Code for the Purchase System.

Now consider a more realistic scenario (adapted from [3]) described by type:

$$
\begin{aligned}
&\mathsf{Buyer} \to \mathsf{Seller}\, buy().\ \mathsf{Seller} \to \mathsf{Buyer}\, price(). \\
&\qquad\qquad \mathsf{Seller} \to \mathsf{Shipper}\, product().\ \mathsf{Shipper} \to \mathsf{Buyer}\, details() \qquad (1)
\end{aligned}
$$

which captures the interactions in a purchase system involving three parties. Messages *buy*, *price*, *product* and *details* are exchanged between a Buyer, a Seller, and a Shipper. First, the buyer sends the seller a buy request, then the seller replies the price back to the buyer. After that, the seller informs the shipper of the chosen product and the shipper sends the buyer the delivery details.

Fig. 1 shows a possible implementation of the purchase interaction system. Using the **new** construct, process `Buyer` creates a fresh channel chat that will host the purchase interaction. This newly created name is passed to a shop, via message *buyService*. Code $Shop \triangleleft_{\mathsf{Buyer}} buyService(chat)$ represents the output of message *buyService* on channel *Shop*, passing name *chat* under role Buyer. The `Buyer` process then sends message *buy*, after which it is simultaneously active to receive *price* and to send *storeService* to *MailBox*, passing name *chat*.

The `Shop` process starts by receiving a channel name (that instantiates variable x) in message *buyService*. Then, in this received channel the `Shop` impersonates the Seller role and receives message *buy*, after which it sends message *price*. At this point, the `Shop` simultaneously impersonates Seller and Shipper which exchanges message *product*, after which message *details* is sent. Notice that this particular `Shop` carries out both the role of the Seller and the role of the Shipper, allowing to represent a shop equipped with its own shipping service.

The `Mail` process defines a message storage service that impersonates the buyer in receiving the shipping delivery details. Notice that the buyer passes name *chat* to the mailbox, allowing in this way a third party to dynamically join the ongoing interaction, while still interacting on the delegated channel (via message *price*). Hence, in this system the Buyer role is actually carried out by two distinct processes (`Buyer` and `Mail`), which can be simultaneously active.

The implementation shown in Fig. 1 involves three distinguished processes that carry out the three roles identified in the protocol, albeit not in a one-to-one-correspondence. The type given in (1) captures the interaction in channel *chat*, which is passed from the buyer to the shop and to the mailbox in messages *buyService* and *storeService*, respectively. In order to analyze the protocol distribution between the three parties, we must consider the "slices" of protocol that are delegated in messages. Namely, the overall protocol is *split* in the type that captures the behavior that is sent to the shop (via message *buyService*):

$$?\mathsf{Seller}\ buy().\ !\mathsf{Seller}\ price().\ \mathsf{Seller} \to \mathsf{Shipper}\ product().\ !\mathsf{Shipper}\ details()$$

and in the type that captures the behavior retained by the buyer:

$$!\mathsf{Buyer}\ buy().?\mathsf{Buyer}\ price().\Diamond ?\mathsf{Buyer}\ details()$$

The $\Diamond$ type expresses the fact that the input of message *details* occurs "sometime", i.e., it does not necessarily occur exactly after the input of message *price*. In fact the `Buyer` process illustrated in Fig. 1 does not guarantee that the input is active only after the reception of message *price*. However, the sequentiality of the message exchanges is ensured by the `Seller` process, since the output of message *details* only occurs after the output of message *price*. A type $\Diamond B$ denotes a behavior that must occur sometime, but not necessarily "now" — $\Diamond B$ types obey the basic laws of the eventually temporal logic operator.

The type of the buyer is further decomposed, at the level of messages *price* and *details*, in $?\mathsf{Buyer}\ price()$ and $\Diamond ?\mathsf{Buyer}\ details()$, the former being retained by the buyer process and the latter delegated to the mailbox. The type of the shop is further decomposed, at the level of message *product*, in $!\mathsf{Seller}\ product()$ and $?\mathsf{Shipper}\ product().!\mathsf{Shipper}\ details()$ which explain the behaviors of the parallel processes in the shop code. All decompositions sketched above are captured by a *type split*, $\circ$, relation that explains how protocols may be split in two complementary slices, along with subtyping. $\Diamond B$ types are crucial to the definition of type splitting, as they provide algebraic support to the flexibility required to sequentially order message exchanges between multiple parties.

In the previous example, the fact that message `details` is exchanged after message `price` is not observable just by looking at the source code of the buyer and mail. However, such ordering is guaranteed by the shop.

$$P ::= \mathbf{0} \;[\!]\; (\mathbf{new}\ x)P \;[\!]\; P_1 \,|\, P_2 \;[\!]\; *P \;[\!]\; x \triangleright_r \{l_i(x_i).P_i\}_{i \in I} \;[\!]\; x \triangleleft_r l(y).P$$

$$l \in \mathcal{L}(abels) \qquad x, y \in \mathcal{N}(ames) \qquad r, s \in \mathcal{R}(oles)$$

Figure 2: Process Syntax.

If we specify that the client, in general, exhibits such behaviors concurrently (e.g., $?\mathsf{Buyer}\ price() \,|\, ?\mathsf{Buyer}\ details()$) we would require (order preserving) decompositions of protocols into multiple threads of behavior. The flexibility introduced by $\diamond B$ types solves this problem as they support the specification of orderings that are guaranteed via synchronization, e.g., type $?\mathsf{Buyer}\ price().\diamond ?\mathsf{Buyer}\ details()$ says that the reception of message details may take place after the reception of message price, which along with the behavior of the shop captured by the type

$!\mathsf{Seller}\ price().\ \mathsf{Seller} \to \mathsf{Shipper}\ product().\ !\mathsf{Shipper}\ details()$

that says that the output of details necessarily occurs after the output of message price, guarantees the overall ordering: first message *price*, then *product* and finally *details*.

The purchase interaction of the system shown in Fig. 1 follows the protocol specification given in (1). Notice that the Buyer role is distributed between two processes (`Buyer` and `Mail`), and that roles Seller and Shipper are carried out by a single process (`Shop`). From the point of view of our type analysis the system follows the prescribed protocol, regardless of the spatial configuration of the processes that implement the roles.

2 Process Language

In this section we present the process model, first by introducing the syntax and second by defining the operational semantics. Our process language is the π-calculus [13] extended with labeled communication and role-based annotations. The syntax, inspired on TyCO [14], is illustrated in Fig. 2, where we consider given infinite sets of labels $\mathcal{L}$, of channel names $\mathcal{N}$ and of roles $\mathcal{R}$. Labels are used to index communication and are static identifiers that may neither be created nor communicated (e.g., XML tags). Names are used to identify mediums of communication. For typing purposes, we distinguish two distinct usages of channels: public (*shared*) communication mediums (e.g., gateways to service providers) and private (*linear*) mediums of communication, where a set of related interactions between several parties may take place (capturing, e.g., service instance interactions, where related communications share correlation tokens). Roles are used to identify the parties involved in communications.

A process is either an inactive process $\mathbf{0}$, a name restriction $(\mathbf{new}\ x)P$ where fresh name x is known only to process P, a parallel composition $P_1 \,|\, P_2$

$$P \,|\, \mathbf{0} \equiv P \qquad P_1 \,|\, P_2 \equiv P_2 \,|\, P_1 \qquad (P_1 \,|\, P_2) \,|\, P_3 \equiv P_1 \,|\, (P_2 \,|\, P_3)$$
$$(\mathbf{new}\ x)(\mathbf{new}\ y)P \equiv (\mathbf{new}\ y)(\mathbf{new}\ x)P$$
$$P_1 \,|\, (\mathbf{new}\ x)P_2 \equiv (\mathbf{new}\ x)(P_1 \,|\, P_2) \ \ (\text{if } x \notin \text{fn}(P_1))$$
$$(\mathbf{new}\ x)\mathbf{0} \equiv \mathbf{0} \qquad *\,P \equiv *P \,|\, P \qquad P_1 \equiv P_2 \ \ (\text{if } P_1 \equiv_\alpha P_2)$$

Figure 3: Structural Congruence.

where P_1 and P_2 are simultaneously active or a replication $*P$ where unlimited copies of P are simultaneously active. Process constructs described up to here (the static fragment) correspond exactly to the ones found in π-calculus. As for communication primitives, we extend (monadic) π-calculus input and output with labeled communication and role annotations: the input summation process $x \triangleright_r \{l_i(x_i).P_i\}_{i\in I}$ is able to receive one message in name x, under role r, labeled by any of the l_is, where i ranges over index set I (we assume that all labels l_i in an input prefix are distinct). Upon synchronization with a l_j labeled message, the respective parameter x_j is instantiated and the respective continuation activated. Notice that the r annotation identifies the role in which the reception is performed. Process $x \triangleleft_r l(y).P$ is able to send a message on channel x, under role r, labeled by l. Upon synchronization the name y is sent and the continuation P activated. In $(\mathbf{new}\ x)P$ all occurrences of x are bound in P, and in $x \triangleright_r \{l_i(x_i).P_i\}_{i\in I}$ all occurrences of x_i are bound in P_i, for each $i = 1, \ldots, n$.

We introduce some auxiliary notions: we use $\mathit{fn}(P)$ to denote the set of free names of process P, defined as expected, and $P[x \leftarrow y]$ to denote the process obtained by replacing all free occurrences of x by y in P. As usual, we omit inactive continuations (e.g., $x \triangleleft_r l(y)$ stands for $x \triangleleft_r l(y).\mathbf{0}$).

The operational semantics is given by a reduction relation and by a structural congruence. We consider the standard definition of structural congruence, noted by $\equiv$, defined as the least congruence that satisfies the rules in Fig. 3. Structural congruence is used in the definition of the reduction relation to syntactically rearrange the process, in order to allow reduction to be defined, as usual, by capturing the basic case for synchronization and identifying the active contexts in which a synchronization may take place.

For typing purposes, since we intend to match process behaviors with type specifications, our reduction relation records (public) synchronization information in labels. Reduction labels (ranged over by λ) are of two forms: a τ label captures a private internal interaction, whereas an $x : s \to rl$ label captures an l-labeled message exchange on channel x, between roles s(ender) and r(eceiver).

We may now present the reduction relation, defined by the rules given in Fig. 4, where we use $P_1 \xrightarrow{\lambda} P_2$ to represent that process P_1 reduces to P_2 with label λ. Rule (Red-Comm) means that parallel output and input

$$x \triangleright_r \{l_i(x_i).P_i\}_{i\in I} \,|\, x \triangleleft_s l_k(y).P \xrightarrow{x:s\to r l_k} P_k[x_k \leftarrow y] \,|\, P \quad (k \in I)$$
(Red-Comm)

$$\frac{P_1 \xrightarrow{\lambda} P_1'}{P_1 \,|\, P_2 \xrightarrow{\lambda} P_1' \,|\, P_2} \qquad \frac{P \xrightarrow{\lambda} P' \qquad \lambda \in \{\tau, x : s \to rl\}}{(\mathbf{new}\ x)P \xrightarrow{\tau} (\mathbf{new}\ x)P'}$$
(Red-Par,Red-New1)

$$\frac{P \xrightarrow{\lambda} P' \qquad \lambda = x : s \to rl \qquad y \neq x}{(\mathbf{new}\ y)P \xrightarrow{\lambda} (\mathbf{new}\ y)P'} \qquad \frac{P_1 \equiv P_1' \qquad P_1' \xrightarrow{\lambda} P_2' \qquad P_2' \equiv P_2}{P_1 \xrightarrow{\lambda} P_2}$$
(Red-New2,Red-Struct)

Figure 4: Reduction Relation.

$$\begin{array}{rl} B ::= & \mathbf{end} \;[\!]\; B \,|\, B \;[\!]\; \Diamond B \;[\!]\; \rho\{l_i(M_i).B_i\}_{i\in I} \\ T ::= \; l(B) \qquad M ::= & B \;[\!]\; T \qquad \rho ::= \,!s \;[\!]\; ?r \;[\!]\; s \to r \end{array}$$

Figure 5: Conversation Types Syntax.

processes may exchange message l_k on channel x, the interaction being captured by label $x : s \to rl_k$, where also the roles involved in the interaction are recorded. As the result of the synchronization, name y is sent to the receiving process which activates the continuation (respective to l_k) instantiating parameter x_k. The continuation of the output prefix is also activated as a consequence of the synchronization. Rule (Red-Par) closes reduction under parallel contexts, while rules (Red-New1) and (Red-New2) close reduction under name restriction. (Red-New1) captures synchronization in private names in the scope of the name restriction, either by "hiding" a public synchronization in the restricted name or by allowing private synchronizations. (Red-New2) captures public synchronizations in the scope of the name restriction, not involving the restricted name. (Red-Struct) closes reduction under structural congruence.

3 Type System

In this section we present our type system. The type language is given in Fig. 5, where we distinguish between behavioral types that describe linear interactions (B) from types that describe shared message exchanges (T) (cf. conversation [3] or session [11] initiation primitives). We also use message (argument) types (M) that specify either a linear protocol or a shared message type, and communication prefixes (ρ) that describe role-based communication actions.

A behavioral type B specifies either the inactive behavior **end**, the parallel composition $B_1 \,|\, B_2$ of two independent behaviors B_1 and B_2, the some-

$$\vdash \mathbf{end} \qquad \vdash \Diamond\mathbf{end} \qquad \frac{B_1 \# B_2 \quad \vdash B_1 \quad \vdash B_2}{\vdash B_1 \,|\, B_2} \qquad \frac{\vdash B_1 \,|\, B_2 \quad \vdash \Diamond B_1 \quad \vdash \Diamond B_2}{\vdash \Diamond(B_1 \,|\, B_2)}$$

$$\frac{\forall i \in I \quad \vdash B_i \quad \rho\{l_i(M_i).\mathbf{end}\}\# B_i}{\vdash \rho\{l_i(M_i).B_i\}_{i\in I}} \qquad \frac{\vdash \rho\{l_i(M_i).B_i\} \quad \rho \in \{!s, ?r\}}{\Diamond\rho\{l_i(M_i).B_i\}_{i\in I} \vdash}$$

Figure 6: Type Well-Formedness.

time $\Diamond B$ which says that behavior B may occur at any point in time, or a menu of labeled actions $\rho\{l_i(M_i).B_i\}_{i\in I}$, each one specifying the type of the name communicated in the message M_i, and the respective continuation behavior B_i. Depending on the communication prefix ρ, an action menu represents either an input summation branching (when ρ is $?r$), an output choice (when ρ is $!s$)—cf. branch and choice session types [11]—or an internal choice $s \to r$, i.e., a matched communication between an output and an input. Notice that the communication roles are identified in the communication prefixes: the *s*ender role in $!s$, the *r*eceiver role in $?r$, and the two roles involved in the interaction in $s \to r$ (s sends to r). Notice also that input and output actions (interface types that capture interactions with the environment) are mixed with matched actions (capturing internal interactions) at the same level in the type language.

The Conversation Type language is extended with role-based annotations and sometime types ($\Diamond B$). Although a specification is not expected to use $\Diamond B$ types, these are crucial to allow the decomposition of protocols into slices, some of which related to interactions that occur later in the protocol.

A message argument type M either specifies a behavioral linear type B, in case a linear name is communicated in the message, or a shared message exchange type T, in case a shared name is communicated in the message. A shared message exchange type T abbreviates $l(B)$, identifying the label of the message exchanged and the (linear) type of the name sent in the message — to simplify the presentation we consider that only linear names can be communicated in shared messages (communicating shared names can be easily encoded).

We now introduce some auxiliary notions, namely the type apartness, well-formed types, and matched types, all defined as predicates. Type apartness is used to identify non-interfering concurrent behaviors that may be safely composed in a linear interaction. To define type-apartness we use $lab(B)$ to denote the set of labels occurring in type B, defined as expected. We say that two types B_1 and B_2 are apart, and we write $B_1 \# B_2$, if the set of labels of B_1 is disjoint from the set of labels of B_2 ($lab(B_1) \cap lab(B_2) = \emptyset$).

Building on apartness, we introduce type well-formedness, noted $\vdash B$, given by the rules in Fig. 6. Informally, in a well-formed type labels do not appear repeatedly in parallel (to ensure race-free behavior) or in sequence

$$\frac{B_1 <: B_1'}{B_1 \,|\, B_2 <: B_1' \,|\, B_2} \qquad \frac{B_i <: B_i'}{\rho\{l_i(M_i).B_i\}_{i\in I} <: \rho\{l_i(M_i).B_i'\}_{i\in I}} \qquad \frac{\vdash \Diamond B}{B <: \Diamond B}$$

$$(B_1 \,|\, B_2) \,|\, B_3 \equiv B_1 \,|\, (B_2 \,|\, B_3) \qquad B_1 \,|\, B_2 \equiv B_2 \,|\, B_1 \qquad B \,|\, \mathbf{end} \equiv B$$

$$\Diamond(B_1 \,|\, B_2) \equiv \Diamond B_1 \,|\, \Diamond B_2 \qquad \Diamond\mathbf{end} \equiv \mathbf{end}$$

Figure 7: Subtyping Relation.

$$B = \mathbf{end} \circ B \qquad \frac{B_1 = B_1' \circ B_1'' \quad B_2 = B_2' \circ B_2'' \quad \vdash B_1 \,|\, B_2}{B_1 \,|\, B_2 = B_1' \,|\, B_2' \circ B_1'' \,|\, B_2''} \tag{S-END,S-PAR}$$

$$\frac{\forall i \in I \quad B_i = B_i' \circ B_i'' \quad \{\rho_1, \rho_2\} = \{!r_1, ?r_2\} \quad \vdash r_1 \to r_2\{l_i(M_i).B_i\}_{i\in I}}{r_1 \to r_2\{l_i(M_i).B_i\}_{i\in I} = \rho_1\{l_i(M_i).B_i'\}_{i\in I} \circ \Diamond\rho_2\{l_i(M_i).B_i''\}_{i\in I}} \tag{S-TAU}$$

$$\frac{\forall i \in I \quad B_i = B_i' \circ \Diamond B \quad \vdash \rho\{l_i(M_i).B_i\}_{i\in I}}{\rho\{l_i(M_i).B_i\}_{i\in I} = \rho\{l_i(M_i).B_i'\}_{i\in I} \circ \Diamond B} \tag{S-BRK}$$

$$\frac{\forall i \in I \quad B_i = B_i' \circ \Diamond B \quad \vdash \Diamond\rho\{l_i(M_i).B_i\}_{i\in I}}{\Diamond\rho\{l_i(M_i).B_i\}_{i\in I} = \Diamond\rho\{l_i(M_i).B_i'\}_{i\in I} \circ \Diamond B} \tag{S-BRKS}$$

$$\frac{B = B_2 \circ B_1}{B = B_1 \circ B_2} \qquad \frac{B_1' = B_2' \circ B_3' \quad B_1 \equiv B_1' \quad B_2 \equiv B_2' \quad B_3 \equiv B_3'}{B_1 = B_2 \circ B_3} \tag{S-SYM,S-EQU}$$

Figure 8: Type Splitting Relation.

(useful to simplify presentation). Also well-formed $\Diamond$ types are not applied directly to message exchanges ($s \to r$), since we are interested in specifying message exchanges that happen exactly at some point in the protocol. Also used by our typing is the notion of matched types, which captures systems where all input actions have a matching output. We say that type B is matched, noted $\mathit{matched}(B)$, if all communication prefixes in B are of the form $s \to r$.

The subtyping relation, noted $<:$, between behavioral types is given in Fig. 7, where we use $B_1 \equiv B_2$ when $B_1 <: B_2$ and $B_2 <: B_1$. We distinguish the use of subtyping to introduce flexibility at the level of $\Diamond$ types: type B is a subtype of $\Diamond B$ which, intuitively, means that carrying out behavior B immediately is a safe implementation of eventually carrying out behavior B.

We may now introduce type split, a ternary relation that explains how a behavioral type may be safely decomposed in two slices of behavior, capturing, in a compositional way, the behavioral contribution of distinct processes to the overall interaction. The type splitting relation is defined by the rules given in Fig. 8, where we use $B = B_1 \circ B_2$ to denote that type B may be decomposed in parts B_1 and B_2. We briefly discuss the splitting rules. Rule (S-END) specifies that a behavioral type may be decomposed in itself and the inactive behavior, typing processes that contribute "all or nothing" to the interaction. Rule (S-PAR) explains the decomposition of two independent behaviors in two slices of behaviors each, capturing the decomposition of a system in two processes that contribute both to independent interactions. Rule (S-TAU) separates a matched communication, between roles r_1 and r_2, in the respective output by role r_1 and input by role r_2, given a splitting of the continuation behaviors. The rule captures the decomposition of a system in two processes that synchronize in a message, each with a given role in the interaction, where one of them carries out the behavior immediately, while the other may carry out the behavior at some point in time ($\Diamond$). In such way, given that one of the behaviors is guaranteed to occur immediately we may ensure that also the message exchange takes place immediately. Notice that a rule that separates the message exchange in two immediate behaviors is not necessary since the sometime behavior may also take place immediately (via subtyping).

Rule (S-BRK) separates a $\Diamond$ (sometime) distinguished slice of behavior from a communication prefixed type, provided this behavior can be split out of (all) the continuations. The rule thus captures the decomposition of a system in two parts, where one retains the (entire) interaction capability specified by the communication prefixed type while the other contributes to ensuing interactions—singled out by the $\Diamond$. Notice that (S-BRK) allows to split behaviors such that the same slice is shared between all branches, useful when addressing, e.g., a branching protocol where every branch terminates with an *ok* or *ack* message. Rule (S-BRKS) expresses the same principle as (S-BRK) but for $\Diamond$ prefixed types. Rule (S-SYM) closes the relation under symmetry and rule (S-EQU) closes the relation under type equivalence.

To simplify the presentation, we sometimes write $B_1 \circ B_2$ to represent a type B such that $B = B_1 \circ B_2$ (if any such B exists). Notice that $B_1 \circ B_2$ does not uniquely identify a type, as B_1 and B_2 may be the result of splitting distinct types. Notice also that a type may be split in several ways. In prior work on conversation types [3], we use "merge" instead of "split", in the sense that if $B = B_1 \circ B_2$ then we may see B as the result of merging the behaviors B_1 and B_2. The merge was originally inspired in the (non-algebraic) end-point projection introduced in [4]. We can show that splitting is an associative relation, which is crucial property to our type system since we rely on the flexibility of the type decomposition to address the behavioral contributions of multiple parties.

We may now present the type system. A typing judgment is of the form $\Delta;\Gamma \vdash P$ where Δ is the typing environment which describes the interactions of P on linear channels, and Γ is the typing environment which describes the interactions of P on shared channels (we write $\Delta;\Gamma$ only when the domains of Δ and Γ are disjoint). Thus, a typing environment Δ is an assignment of identifiers to behavioral types ($\Delta \triangleq x_1 : B_1, \ldots, x_k : B_k$) and a typing environment Γ is an assignment of identifiers to message exchange types ($\Gamma \triangleq x_1 : T_1, \ldots, x_k : T_k$). We introduce some auxiliary notation to simplify presentation: we use $(x_1 : B'_1, \ldots, x_k : B'_k, \Delta_1) \circ (x_1 : B''_1, \ldots, x_k : B''_k, \Delta_2)$ to denote $x_1 : B_1, \ldots, x_k : B_k, \Delta_1, \Delta_2$ such that $B_i = B'_i \circ B''_i$, for all i in $1, \ldots, k$ and the domains of Δ_1 and Δ_2 are disjoint. Also, we use $x_1 : B_1, \ldots, x_k : B_k <: x_1 : B'_1, \ldots, x_k : B'_k$ when $B_i <: B'_i$, for all i in $1, \ldots, k$, and $\Delta_{\mathbf{end}}$ to denote $x_1 : \mathbf{end}, \ldots, x_k : \mathbf{end}$.

We say process P is well-typed if $\Delta;\Gamma \vdash P$ may be derived using the rules given in Fig. 9. We discuss the key features of the typing rules. Rule (T-END) says the inactive process has no linear behavior (but complies to any shared behavior specification). Rule (T-PAR) types the parallel composition process with the linear types which may be split in the behaviors of the two parallel branches, while ensuring both branches comply to the same usage of shared types. Rule (T-NEW) types a restricted linear name provided its usage is matched, i.e., it has no outstanding unmatched (? or !) communications. Rule (T-SNEW) types a restricted shared name, if it is used according to a shared message exchange.

Rules for communication prefixes are divided in three groups, depending on the shared or linear usage of both communication subject and object. Rules (T-SIN) and (T-SOUT) address the case when the communication subject has shared usage while the object has linear usage. Notice that the behavioral type B, specified in the argument type of the shared message exchange type $l(B)$ of x, captures the slice of behavior which is delegated in the communication. Type B describes the linear usage of the input parameter in the premise of (T-SIN), and is singled out via splitting in the conclusion of (T-SOUT), where splitting is used so as to take into account the usage of y (the sent name) by the continuation (crucial to type processes that delegate a name and continue to interact in it).

Rules (T-IN) and (T-OUT) address the cases when both the communication subject and object have linear usage, and follow the same lines as described above. Both rules record the prefixed type $\rho\{l_i(B'_i).B_i\}_{i \in I}$ in the conclusions, where ρ is either $?r$ or $!r$ for input and output, respectively. A single output is typed with a communication menu (containing the label of the emitted message) so as to directly match input summation menus. Notice that the prefixed type is taken up to subtyping, so as to allow to introduce $\diamond$ types that may be necessary for the split in the conclusion. Notice also that the prefixed type is singled out via splitting, so as to take into account behaviors of x originally assigned to other threads (due to name delegation).

$$\Delta_{\mathbf{end}};\Gamma \vdash \mathbf{0} \qquad \frac{\Delta_1;\Gamma \vdash P_1 \qquad \Delta_2;\Gamma \vdash P_2}{\Delta_1 \circ \Delta_2;\Gamma \vdash P_1 \mid P_2} \qquad \text{(T-END,T-PAR)}$$

$$\frac{\Delta, x : B;\Gamma \vdash P \qquad matched(B)}{\Delta;\Gamma \vdash (\mathbf{new}\ x)P} \qquad \frac{\Delta;\Gamma, x : l(B) \vdash P}{\Delta;\Gamma \vdash (\mathbf{new}\ x)P} \qquad \text{(T-NEW,T-SNEW)}$$

$$\frac{\forall i \in I \qquad \Delta \circ x : B_i, y_i : B_i';\Gamma \vdash P_i \qquad ?r\{l_i(B_i').B_i\}_{i\in I} <: B}{\Delta \circ x : B;\Gamma \vdash x \triangleright_r \{l_i(y_i).P_i\}_{i\in I}} \qquad \text{(T-IN)}$$

$$\frac{k \in I \qquad \Delta \circ x : B_k;\Gamma \vdash P \qquad !r\{l_i(B_i').B_i\}_{i\in I} <: B}{\Delta \circ x : B \circ y : B_k';\Gamma \vdash x \triangleleft_r l_k(y).P} \qquad \text{(T-OUT)}$$

$$\frac{\forall i \in I \qquad \Delta \circ x : B_i';\Gamma, y_i : T_i \vdash P_i \qquad ?r\{l_i(T_i).B_i'\}_{i\in I} <: B}{\Delta \circ x : B;\Gamma \vdash x \triangleright_r \{l_i(y_i).P_i\}_{i\in I}} \qquad \text{(T-LSIN)}$$

$$\frac{\Delta \circ x : B_k';\Gamma, y : T_k \vdash P \qquad !r\{l_i(T_i).B_i'\}_{i\in I} <: B}{\Delta \circ x : B;\Gamma, y : T_k \vdash x \triangleleft_r l_k(y).P} \qquad \text{(T-LSOUT)}$$

$$\frac{\Delta, y : B;\Gamma, x : l(B) \vdash P}{\Delta;\Gamma, x : l(B) \vdash x \triangleright_r \{l(y).P\}} \qquad \frac{\Delta;\Gamma, x\colon l(B) \vdash P}{\Delta \circ y\colon B;\Gamma, x\colon l(B) \vdash x \triangleleft_r l(y).P} \qquad \text{(T-SIN,T-SOUT)}$$

$$\frac{\Delta_1;\Gamma \vdash P \quad \Delta_1 <: \Delta_2}{\Delta_2;\Gamma \vdash P} \qquad \frac{\Delta_{\mathbf{end}};\Gamma \vdash P}{\Delta_{\mathbf{end}};\Gamma \vdash *P} \qquad \text{(T-SUB,T-REP)}$$

Figure 9: Typing Rules.

Rules (T-LSIN) and (T-LSOUT) follow similar lines, addressing the case when the communication subject / object have linear / shared use. The last two rules are (T-REP) which types the replicated process, provided it uses no linear names, and the subsumption rule (T-SUB).

We can show that typing is preserved by substitution and by structural congruence. Given that our main result involves relating process actions and type specifications, we introduce type reduction, defined by the rules given in Fig. 10. In this way, we are able to precisely describe process reductions via the corresponding type reductions. Type reduction specifies how matched types reduce, explaining a message exchange that activates the respective continuation. Type reduction relies on reduction labels of the form $s \to rl$, identifying the roles involved in the communication and the label of the exchanged message.

$$s \to r\{l_i(M_i).B_i\}_{i\in I} \xrightarrow{s\to rl_k} B_k \ \ (k \in I) \qquad \frac{B_1 \xrightarrow{s\to rl} B_2}{B_1 \mid B \xrightarrow{s\to rl} B_2 \mid B} \qquad \frac{B_1 \xrightarrow{s\to rl} B_2}{B \mid B_1 \xrightarrow{s\to rl} B \mid B_2}$$

Figure 10: Type Reduction.

$$\Delta; \Gamma \xrightarrow{\tau} \Delta; \Gamma \qquad \Delta; \Gamma, x : T \xrightarrow{x:s\to rl} \Delta; \Gamma, x : T \qquad \frac{B_1 \xrightarrow{s\to rl} B_2}{\Delta, x : B_1; \Gamma \xrightarrow{x:s\to rl} \Delta, x : B_2; \Gamma}$$

Figure 11: Typing Environment Reduction.

Type reduction provides the expected semantics of behavioral types. Building on type reduction and in order to simplify the presentation of the results we introduce typing environment reduction, given by the rules in Fig. 11. Typing environment reduction specifies that environments seamlessly mimic internal τ (non public) reductions as well as synchronizations on shared channels. Also, typing environments exhibit linear reductions provided the reduction is observable at the level of the type of the respective channel. We may now state our main result that explains process reduction via typing environment reduction.

Theorem 3.1 (Type Preservation) *Let* $\Delta; \Gamma \vdash P$. *If* $P \xrightarrow{\lambda} P'$ *then* $\Delta; \Gamma \xrightarrow{\lambda} \Delta'; \Gamma$ *and* $\Delta'; \Gamma \vdash P'$.

Proof. By induction on the length of the derivation of $P \xrightarrow{\lambda} P'$ (see Appendix).

Theorem 3.1 states that any reduction of a well-typed process is explained by the corresponding type reduction, thus ensuring that processes interact according to the protocols prescribed by the types. Notice that this compliance entails that the protocols are actually carried out by the roles accordingly to the type specifications. We provide a precise characterization of this property as follows.

Definition 3.2 (Role-Based Protocol Fidelity) *Let P be a process and Δ, Γ typing environments. We say P follows the role-based protocols prescribed by Δ, Γ if for any reduction sequence of the process:*

$P \xrightarrow{\lambda_1} P_1 \xrightarrow{\lambda_2} \dots \xrightarrow{\lambda_k} P_k$

there is a matching reduction sequence of the typing environments:

$\Delta, \Gamma \xrightarrow{\lambda_1} \Delta_1, \Gamma \xrightarrow{\lambda_2} \dots \xrightarrow{\lambda_k} \Delta_k, \Gamma$

We have that well-typed processes satisfy role-based protocol fidelity as a direct consequence of Theorem 3.1.

Corollary 3.3 (Role-Based Protocol Fidelity) *Let* $\Delta; \Gamma \vdash P$. *We have that P follows the role-based protocols prescribed by* Δ, Γ.

In order to provide further intuition we proceed to typing an example. Returning to Fig. 1, the type of name *chat*, as described in (1), page 4, is checked by successively splitting and matching resulting types with subprocesses. In this case, for example, we have that the type of Buyer after the first delegation can be decomposed by using rules (S-END), (S-BRK) and (S-BRK) ($\mathsf{b} = \underline{\mathsf{B}}\mathsf{uyer}$).

$$\dfrac{\dfrac{\overline{\Diamond ?\mathsf{b}\{\mathit{details}().\mathbf{end}\} = \mathbf{end} \circ \Diamond ?\mathsf{b}\{\mathit{details}().\mathbf{end}\}}}{?\mathsf{b}\{\mathit{price}().\Diamond ?\mathsf{b}\{\mathit{details}().\mathbf{end}\}\} = ?\mathsf{b}\{\mathit{price}().\mathbf{end}\} \circ \Diamond ?\mathsf{b}\{\mathit{details}().\mathbf{end}\}}}{!\mathsf{b}\{\mathit{buy}().?\mathsf{b}\{\mathit{price}().\Diamond ?\mathsf{b}\{\mathit{details}().\mathbf{end}\}\}\} = !\mathsf{b}\{\mathit{buy}().?\mathsf{b}\{\mathit{price}().\mathbf{end}\}\} \circ \Diamond ?\mathsf{b}\{\mathit{details}().\mathbf{end}\}}$$

Now, the splitting given above appears when typing the subprocess

$$\mathit{chat} \lhd_{\mathsf{Buyer}} \mathit{buy}().(\mathit{chat} \rhd_{\mathsf{Buyer}} \mathit{price}() \mid \mathit{MailBox} \lhd_{\mathsf{Buyer}} \mathit{storeService}(\mathit{chat}))$$

Here, the delegation of name *chat*, through service *storeService*, requires that the behavior of *chat* to be split between the two processes. Using (T-SUB), (T-SOUT) and (T-END) we have:

$$\dfrac{\dfrac{\overline{\mathit{chat} : \mathbf{end}; \mathit{MailBox} : \mathit{storeService}(?\mathsf{b}\{\mathit{details}().\mathbf{end}\}) \vdash \mathbf{0}}}{\mathit{chat} : ?\mathsf{b}\{\mathit{details}().\mathbf{end}\}; \mathit{MailBox} : \mathit{storeService}(\cdots) \vdash \mathit{MailBox} \lhd_{\mathsf{Buyer}} \mathit{storeService}(\mathit{chat})}}{\mathit{chat} : \Diamond ?\mathsf{b}\{\mathit{details}().\mathbf{end}\}; \mathit{MailBox} : \mathit{storeService}(\cdots) \vdash \mathit{MailBox} \lhd_{\mathsf{Buyer}} \mathit{storeService}(\mathit{chat})}$$

The example shows that the *sometime* operator behaves as a delayed choice between a *dot*, which expresses the sequentially of behaviors, and a *paralell* composition, which types concurrent actions. These alternatives are introduced by rules (S-TAU), in only one of the branches of the split types, and in order to preserve, globally, the specified order of labels. Conceivably, the same flexibility would be achieved by a different (S-TAU) rule, which would immediately select between *dot* and *parallel.* Nevertheless, such rule would need to "look inside" the types and pull *parallels* to the top level. Therefore, this extension of session types with a new modality for breaking sequentiality, enriches the languages of types with an operator that enables us to perform choices locally and as needed. Such innovation yields a split operation which is associative and structural on language of processes.

4 Concluding Remarks

Our development is based on previous work on conversation types [3], extended so as to address assignment of dynamic roles to the several parties involved. Technically, we identified a minimal set of ingredients to add to a core process specification language (the π-calculus [13], TyCO [14] more precisely) so as to address role-based protocol verification (labeled channels and role annotations) and extended the type analysis accordingly. Noticeably, the splitting relation defined in this paper is much more readable and also more expressive than the merge relation in [3] — in particular, it allows for splitting (the same) behavior out of the continuations of a branching behavior. Crucial to our development is the introduction of the $\diamond$ type which allows to control behavior interleaving.

We discuss some extensions to our development. An essential feature of any type analysis is a verification procedure. We are yet to implement such a procedure, but we may already assert there exists such a procedure in a setting where all bound names are type annotated. Another crucial property left out of this paper is progress. However, we expect that the progress analysis introduced in [3] for a labeled π-calculus, combined with our typing analysis, may be used to single-out systems that enjoy progress. An interesting further development to be addressed is the dynamic delegation of roles. In our setting roles are statically annotated in processes. Extending the language with role delegation would allow parties to dynamically assume unanticipated roles.

Several works address role-based type specifications to enforce security concerns (for example [7] introduces a type analysis to discipline role-based access control to data). We focus on communication protocol assignment and leave security to be handled orthogonally. Our approach builds on conversation type theory, introduced as a generalization of session types [9, 11] to discipline multiparty interaction, including dynamically established conversations with an unanticipated number of participants. Other works share the goal to address multiparty interaction, namely [1, 5, 10, 12], with respect to which we distinguish the approach of conversation since it addresses multiparty interaction where the number of participants is not fixed a priori, while considering a simpler underlying model. We remark that in [1, 5, 10] a notion of role assignment is explicit, unlike in [3] where types do not mention identities of communicating partners. However, such role assignment is achieved via a structural projection, forcing single roles to be carried out by single threads. A different notion of dynamic roles is also considered in the approaches described in [6, 8], allowing for several processes, much like a thread pool, to simultaneously carry out a single role.

In this work we have presented a type-based analysis which ensures that systems follow the prescribed role-based protocol specifications. Novel to our approach is the flexibility of role assignment, allowing us to address dynamic distributed implementations of role specifications, where a single role can be distributed between several processes and a single process can dynamically switch between roles. To the best of our knowledge, ours is the only (session-type like) approach that addresses such configurations, that are actually found in, e.g., real world business protocols. Our development extends conversation types with role-based protocol specifications, retaining the simplicity of the approach, simplifying and generalizing the underlying technical framework, and contrasting with related approaches in the dynamic and flexible nature of roles.

References

[1] Lorenzo Bettini, Mario Coppo, Loris D'Antoni, Marco De Luca, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. Global Progress in Dynamically Interleaved Multiparty Sessions. In *CONCUR 2008*, volume 5201 of *LNCS*, pages 418–433. Springer, 2008.

[2] Luís Caires and Hugo T. Vieira. Conversation Types. In *ESOP 2009, 18th European Symposium on Programming, Proceedings*, volume 5502 of *LNCS*, pages 285–300. Springer, 2009.

[3] Luís Caires and Hugo T. Vieira. Conversation Types. *Theoretical Computer Science*, 411(51-52):4399–4440, 2010.

[4] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured Communication-Centred Programming for Web Services. In *ESOP 2007*, volume 4421 of *LNCS*, pages 2–17. Springer, 2007.

[5] Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, and Luca Padovani. On Global Types and Multi-party Sessions. In *FMOODS/FORTE 2011*, volume 6722 of *LNCS*, pages 1–28. Springer, 2011.

[6] Pierre-Malo Deniélou and Nobuko Yoshida. Dynamic Multirole Session Types. In *POPL 2011*, pages 435–446. ACM, 2011.

[7] Silvia Ghilezan, Svetlana Jaksic, Jovanka Pantovic, and Mariangiola Dezani-Ciancaglini. Types and Roles for Web Security. *Transactions on Advanced Research*, 8(2):16–21, 2012.

[8] Elena Giachino, Matthew Sackman, Sophia Drossopoulou, and Susan Eisenbach. Softly Safely Spoken: Role Playing for Session Types. In *PLACES 2009*, 2009.

[9] Kohei Honda. Types for Dyadic Interaction. In *CONCUR 1993*, volume 715 of *LNCS*, pages 509–523. Springer, 1993.

[10] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty Asynchronous Session Types. In *POPL 2008*, pages 273–284. ACM Press, 2008.

[11] Kokei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language Primitives and Type Discipline for Structured Communication-Based Programming. In *ESOP 1998*, volume 1381 of *LNCS*, pages 122–138. Springer, 1998.

[12] Luca Padovani. Session Types at the Mirror. In *ICE 2009*, volume 12 of *EPTCS*, pages 71–86, 2009.

[13] Davide Sangiorgi and David Walker. *The π-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.

[14] Vasco T. Vasconcelos and Mario Tokoro. A typing system for a calculus of objects. In *ISOTAS 1993*, volume 472 of *LNCS*, pages 460–474. Springer, 1993.

A Proofs

B Example:

We illustrate reduction using the purchase interaction example discussed in the Introduction. Consider the following code:

$$
\begin{array}{l}
(\mathbf{new}\ chat)\\
(Seller \lhd_{\mathsf{Buyer}} buyService(chat).\\
\quad chat \lhd_{\mathsf{Buyer}} buy().chat \rhd_{\mathsf{Buyer}} price().chat \rhd_{\mathsf{Buyer}} details())\\
|\\
Seller \rhd_{\mathsf{Seller}} buyService(x).\\
\quad x \rhd_{\mathsf{Seller}} buy().x \lhd_{\mathsf{Seller}} price().Shipper \lhd_{\mathsf{Seller}} shipService(x).\\
\hfill x \lhd_{\mathsf{Seller}} product()\\
|\\
Shipper \rhd_{\mathsf{Shipper}} shipService(x).\\
\quad x \rhd_{\mathsf{Shipper}} product().x \lhd_{\mathsf{Shipper}} details()
\end{array}
$$

The only possible interaction in the system is the exchange of the *buyService* message in name *Seller* between Buyer and Seller. As the result of this communication, channel *chat* is sent from the buyer to the seller, allowing them to share a private medium of communication. The *buyService* message exchange, recorded in label $Seller : \mathsf{Buyer} \to \mathsf{Seller} buyService()$, leads to the system:

$$
\begin{array}{l}
(\mathbf{new}\ chat)\\
(chat \lhd_{\mathsf{Buyer}} buy().chat \rhd_{\mathsf{Buyer}} price().chat \rhd_{\mathsf{Buyer}} details()\\
|\\
chat \rhd_{\mathsf{Seller}} buy().chat \lhd_{\mathsf{Seller}} price().Shipper \lhd_{\mathsf{Seller}} shipService(chat).\\
\hfill chat \lhd_{\mathsf{Seller}} product())\\
|\\
Shipper \rhd_{\mathsf{Shipper}} shipService(x).\\
\quad x \rhd_{\mathsf{Shipper}} product().x \lhd_{\mathsf{Shipper}} details()
\end{array}
$$

At this point the exchange of messages *buy* and then *price* between Buyer and Seller in channel *chat* may take place (described via τ labels, since *chat* is private to the system), leading to the following configuration:

$$
\begin{array}{l}
(\textbf{new}\ \mathit{chat}) \\
(\mathit{chat} \triangleright_{\mathsf{Buyer}} \mathit{details}() \\
\mid \\
\mathit{Shipper} \triangleleft_{\mathsf{Seller}} \mathit{shipService}(\mathit{chat}).\mathit{chat} \triangleleft_{\mathsf{Seller}} \mathit{product}()) \\
\mid \\
\mathit{Shipper} \triangleright_{\mathsf{Shipper}} \mathit{shipService}(x). \\
\quad x \triangleright_{\mathsf{Shipper}} \mathit{product}().x \triangleleft_{\mathsf{Shipper}} \mathit{details}()
\end{array}
$$

By now message *shipService* may be exchanged in name *Shipper*, where channel *chat* is sent to *Shipper*, allowing for a third-party to join the ongoing interaction. Notice *Seller* and *Shipper* get to interact on the delegated channel *chat* afterwords, exchanging message *product*. Notice also that label $\mathit{Seller} : \mathsf{Shipper} \rightarrow \mathsf{Seller}\mathit{shipService}()$ describes the *shipService* message exchange, identifying the roles involved in the interaction. This information is relevant to our typing analysis, presented in the next section, where process behaviors are checked against type specifications.

C Proofs

Proposition C.1 (Associativity) *If $B = B_1 \circ B'$ and $B' = B_2 \circ B_3$ then there exists B'' such that $B = B'' \circ B_3$ and $B'' = B_1 \circ B_2$.*

Proof. By induction on the length of the derivation tree of $B = B_1 \circ B'$.

Base: $B = B_1 \circ B'$ by rule [S-END]. We have two cases.

1. B_1 is **end** and $B = B'$. In this case, let $B'' = B_2$. Hence, $B_2 = \mathbf{end} \circ B_2$ and $B = B_2 \circ B_3$.

2. B' is **end** and $B = B_1$. In this case, we set $B'' = \mathbf{end} = \mathbf{end} \circ \mathbf{end}$, with $B_2 = B_3 = \mathbf{end}$.

Step: Suppose the Proposition is valid for all derivation trees of degree $\leq k$. Let $B = B_1 \circ B'$ be a derivation with degree $k + 1$. Lets analyze all possible cases.

1. [S-PAR] is the last rule in the derivation tree of $B = B_1 \circ B'$. Hence, $B = B_1^\bullet \mid B_2^\bullet$ with $B_1 = B_{11}^\bullet \mid B_{21}^\bullet$ and $B' = B_{12}^\bullet \mid B_{22}^\bullet$, such that

$$
\dfrac{\dfrac{\vdots}{B_1^\bullet = B_{11}^\bullet \circ B_{12}^\bullet}\ \mathrm{R1} \qquad \dfrac{\vdots}{B_2^\bullet = B_{21}^\bullet \circ B_{22}^\bullet}\ \mathrm{R2}}{B_1^\bullet \mid B_2^\bullet = B_{11}^\bullet \mid B_{21}^\bullet \circ B_{12}^\bullet \mid B_{22}^\bullet}\ \text{[S-PAR]}
$$

(a) If $B' = B_2 \circ B_3$ by rule [S-END], then we have two cases: $B_2 = \textbf{end}$ and $B_3 = \textbf{end}$. If $B_2 = \textbf{end}$, then we let $B'' = B_1$. And, if $B_3 = \textbf{end}$, then we set $B'' = B$.

(b) Suppose $B' = B_2 \circ B_3$ by rule [S-PAR]. Hence

$$\frac{B^\bullet_{12} = B^\bullet_{121} \circ B^\bullet_{122} \qquad B^\bullet_{22} = B^\bullet_{221} \circ B^\bullet_{222}}{B^\bullet_{12} \,|\, B^\bullet_{22} = B^\bullet_{121} \,|\, B^\bullet_{221} \circ B^\bullet_{122} \,|\, B^\bullet_{222}}\ \text{[S-PAR]}$$

Where $B_2 = B^\bullet_{121} \,|\, B^\bullet_{221}$ and $B_3 = B^\bullet_{122} \,|\, B^\bullet_{222}$.
Now, applying the induction hypothesis to branch R1 and $B^\bullet_{12} = B^\bullet_{121} \circ B^\bullet_{122}$ yields $B^\bullet_1 = (B^\bullet_{11} \circ B^\bullet_{121}) \circ B^\bullet_{122}$. [1] In the same manner, applying the induction hypothesis to branch R2 yields $B^\bullet_2 = (B^\bullet_{21} \circ B^\bullet_{221}) \circ B^\bullet_{222}$.
Therefore, by applying rule [S-PAR] we have

$$\frac{\dfrac{\vdots}{B^\bullet_1 = (B^\bullet_{11} \circ B^\bullet_{121}) \circ B^\bullet_{122}}\ \text{R1'} \qquad \dfrac{\vdots}{B^\bullet_2 = (B^\bullet_{21} \circ B^\bullet_{221}) \circ B^\bullet_{222}}\ \text{R2'}}{B^\bullet_1 \,|\, B^\bullet_2 = (B^\bullet_{11} \circ B^\bullet_{121}) \,|\, (B^\bullet_{21} \circ B^\bullet_{221}) \circ B^\bullet_{122} \,|\, B^\bullet_{222}}\ \text{[S-PAR]}$$

Hence, we let $B'' = (B^\bullet_{11} \circ B^\bullet_{121}) \,|\, (B^\bullet_{21} \circ B^\bullet_{221})$, and by rule [S-PAR] we get $B'' = B_1 \circ B_2$.

2. [S-TAU] is the last rule in $B = B_1 \circ B'$.

$$\frac{B_i = B_{1i} \circ B_{2i} \quad \forall i \in I}{r \to s\{l_i(B'_i).B_i\}_i = !r\{l_i(B'_i).B_{1i}\}_i \circ ?s\{l_i(B'_i).B_{2i}\}_i}\ \text{[S-TAU]}$$

We need to consider two cases.

(a) $B_1 = !r\{l_i(B'_i).B_{1i}\}_i$. If $B' = B_2 \circ B_3$, then it must be by rule [S-END] or [S-BRK]. The case of rule [S-END] is straightforward as before. Now, if $B' = B_2 \circ B_3$ comes by rule [S-BRK] we have also two cases.

$$\frac{B_{2i} = B''_i \circ B''' \quad \forall i \in I}{?s\{l_i(B'_i).B_{2i}\}_i = ?s\{l_i(B'_i).B''_i\}_i \circ B'''}\ \text{[S-BRK]}$$

[1] Here, we use $(B' \circ B'')$ to denote a B, such that $B = B' \circ B''$.

i. $B_2 = ?s\{l_i(B'_i).B''_i\}_i$ and $B_3 = B'''$. In this case, we apply the induction hypothesis on B_i and get $B_i = (B_{1i} \circ B''_i) \circ B'''$, for all $i \in I$. Hence, by rule [S-BRK] we have

$$\frac{B_i = (B_{1i} \circ B''_i) \circ B''' \quad \forall i \in I}{r \to s\{l_i(B'_i).B_i\}_i = r \to s\{l_i(B'_i).(B_{1i} \circ B''_i)\}_i \circ B'''} \text{[S-BRK]}$$

Hence, taking $B'' = r \to s\{l_i(B'_i).(B_{1i} \circ B''_i)\}_i$ yields, by rule [S-TAU],

$$B'' = !r\{l_i(B'_i).B_{1i}\}_i \circ ?s\{l_i(B'_i).B''_i\}_i = B_1 \circ B_2.$$

ii. $B_3 = ?s\{l_i(B'_i).B''_i\}_i$ and $B_2 = B'''$. Once more, by applying the hypothesis to B_i we have $B_i = (B_{1i} \circ B''') \circ B_{2i}$, for all $i \in I$. Applying rule [S-TAU] yields

$$r \to s\{l_i(B'_i).(B_{1i} \circ B''_i)\}_i = !r\{l_i(B'_i).(B_{1i} \circ B''')\}_i \circ ?s\{l_i(B'_i).B_{2i}\}_i$$

Now, we set $B'' = !r\{l_i(B'_i).(B_{1i} \circ B''')\}_i$, because by rule [S-BRK] we have $B'' = !r\{l_i(B'_i).(B_{1i} \circ B''')\}_i = !r\{l_i(B'_i).B_{1i}\}_i \circ B''' = B_2 \circ B_3$.

(b) $B_1 = ?r\{l_i(B'_i).B_i\}_i$. This case is analogous to the previous case (a).

3. [S-BRK] is the last rule in $B = B_1 \circ B'$.

$$\frac{B_i = B''_i \circ B''' \quad \forall i \in I}{\rho\{l_i(B'_i).B_i\}_i = \rho\{l_i(B'_i).B''_i\}_i \circ B'''} \text{[S-BRK]}$$

Once more, two cases to be considered.

(a) $B_2 = \rho\{l_i(B'_i).B''_i\}_i$ and $B_3 = B'''$. This case is analogous to the previous case (a) (i).

(b) $B_3 = \rho\{l_i(B'_i).B''_i\}_i$ and $B_2 = B'''$. This case is analogous to the previous case (a) (ii).

4. [S-BRKS] is the last rule in $B = B_1 \circ B'$. Analogous to the previous case.

Lemma C.2 (Substitution) *If* $\Delta; \Gamma \vdash P$ *and*

1. $\Delta = \Delta', x : B$ *and* $\Delta' \circ y : B$ *is defined then* $\Delta' \circ y : B; \Gamma \vdash P[x \leftarrow y]$.

2. $\Gamma = \Gamma', x : T$ *and* $\Gamma, y : T$ *is defined then* $\Delta; \Gamma', y : T \vdash P[x \leftarrow y]$.

Proof. By induction on the length of the derivation of $\Delta; \Gamma \vdash P$.

Lemma C.3 (Subject Congruence) *If* $\Delta; \Gamma \vdash P$ *and* $P \equiv P'$ *then* $\Delta; \Gamma \vdash P'$.

Proof. By induction on the length of the derivation of $P \equiv P'$.

Theorem 3.1 (Type Preservation)
(repetition of the statement in page 14)
If $\Delta; \Gamma \vdash P$ *and* $P \xrightarrow{\lambda} P'$ *and*

- $\lambda = \tau$ *then* $\Delta; \Gamma \vdash P'$;
- $\lambda = x : s \to rl$ *then (1)* $\Delta = \Delta', x : B$ *and* $B \xrightarrow{s \to rl} B'$ *and* $\Delta', x : B'; \Gamma \vdash P'$ *or (2)* $\Gamma = \Gamma', x : T$ *and* $\Delta; \Gamma \vdash P'$.

Proof. By induction on the length of the derivation of $P \xrightarrow{\lambda} P'$.
(*Case* (Red-Comm))

$$\Delta; \Gamma \vdash x \triangleright_r \{l_i(x_i).P_i\}_{i \in I} \mid x \triangleleft_s l_k(y).P \quad (1)$$

$$x \triangleright_r \{l_i(x_i).P_i\}_{i \in I} \mid x \triangleleft_s l_k(y).P \xrightarrow{x:s \to rl_k} P_k[x_k \leftarrow y] \mid P \quad (2)$$

(Assumption)

(**Case** $x \in dom(\Gamma) \wedge y \in dom(\Delta)$)

$$\Delta_1; \Gamma', x : l_1(B) \vdash x \triangleright_r \{l_1(x_1).P_1\} \quad (3)$$

$$I = \{1\}, k = 1 \quad (4)$$

((T-SIN))

$$\Delta_2 \circ y : B; \Gamma', x : l_1(B) \vdash x \triangleleft_s l_1(y).P \quad (5)$$

((T-SOUT))

$$\Delta = \Delta_1 \circ \Delta_2 \circ y : B \quad (6)$$

$$\Gamma = \Gamma', x : l_1(B) \quad (7)$$

$$\Delta; \Gamma \vdash x \triangleright_r \{l_1(x_1).P_1\} \mid x \triangleleft_s l_1(y).P \quad (8)$$

((3), (5) and (1))

$$\Delta_1, x_1 : B; \Gamma', x : l_1(B) \vdash P_1 \quad (9)$$

(Inversion on (T-SIN) and (3))

$$\Delta_1 \circ y : B \text{ defined} \tag{10}$$

((6))

$$\Delta_1 \circ y : B; \Gamma', x : l_1(B) \vdash P_1[x_1 \leftarrow y] \tag{11}$$

((9) and (10) and Lemma C.2)

$$\Delta_2; \Gamma', x : l_1(B) \vdash P \tag{12}$$

(Inversion on (T-SOUT) and (5))

$$\Delta; \Gamma \vdash P_1[x_1 \leftarrow y] \,|\, P \tag{13}$$

((12), (11), (6), (7) and (T-PAR))

(**Case** $x \in dom(\Delta) \wedge y \in dom(\Delta)$)

$$\Delta_1 \circ x : ?r\{l_i(B'_i).B_i\}_{i\in I}; \Gamma \vdash x \triangleright_r \{l_i(x_i).P_i\}_{i\in I} \tag{14}$$

((T-IN))

$$\Delta_2 \circ x : !s\{l_i(B'_i).B''_i\}_{i\in I} \circ y : B'_k; \Gamma \vdash x \triangleleft_s l_k(y).P \tag{15}$$

((T-OUT))

$$\Delta = \Delta_1 \circ \Delta_2 \circ x : ?r\{l_i(B'_i).B_i\}_{i\in I} \circ x : !s\{l_i(B'_i).B''_i\}_{i\in I} \circ y : B'_k \tag{16}$$

((14), (15) and (1))

$$\forall i \in I \qquad \Delta_1 \circ x : B_i, x_i : B'_i; \Gamma \vdash P_i \tag{17}$$

(Inversion on (T-IN) and (14))

$$\Delta_1 \circ x : B_k \circ y : B'_k \text{ defined} \tag{18}$$

((16))

$$\Delta_1 \circ x : B_k \circ y : B'_k; \Gamma \vdash P_k[x_k \leftarrow y] \tag{19}$$

((17) and (18) and Lemma C.2)

$$\Delta_2 \circ x : B''_k; \Gamma \vdash P \tag{20}$$

(Inversion on (T-OUT) and (15))

$$\Delta_1 \circ \Delta_2 \circ x : (B_k \circ B''_k) \circ y : B'_k; \Gamma \vdash P_k[x_k \leftarrow y] \,|\, P \tag{21}$$

((20), (19), (16) and (T-PAR))

$$\Delta = \Delta_1 \circ \Delta_2 \circ x : s \to r\{l_i(B'_i).(B_i \circ B''_i)\}_{i\in I} \circ y : B'_k \tag{22}$$

((16))

$$\Delta \xrightarrow{s \to r l_k} \Delta_1 \circ \Delta_2 \circ x : (B_k \circ B''_k) \circ y : B'_k \tag{23}$$

((22))

(**Case** $x \in dom(\Delta) \wedge y \in dom(\Gamma)$)

$$\Delta_1 \circ x : ?r\{l_i(T).B'_i\}_{i\in I}; \Gamma \vdash x \triangleright_r \{l_i(x_i).P_i\}_{i\in I} \tag{24}$$
((T-LSIN))
$$\Delta_2 \circ x : !s\{l_i(T).B''_i\}_{i\in I}; \Gamma', y : T \vdash x \triangleleft_s l_k(y).P \tag{25}$$
((T-LSOUT))
$$\Gamma = \Gamma', y : T \tag{26}$$
$$\Delta = \Delta_1 \circ \Delta_2 \circ x : ?r\{l_i(T).B'_i\}_{i\in I} \circ x : !s\{l_i(T).B''_i\}_{i\in I} \tag{27}$$
((24), (25) and (1))
$$\forall i \in I \qquad \Delta_1 \circ x : B'_i; \Gamma, x_i : T \vdash P_i \tag{28}$$
(Inversion on (T-LSIN) and (24))
$$\Gamma', y : T \text{ defined} \tag{29}$$
((26))
$$\Delta_1 \circ x : B'_k; \Gamma', y : T \vdash P_k[x_k \leftarrow y] \tag{30}$$
((28) and (29) and Lemma C.2)
$$\Delta_2 \circ x : B''_k; \Gamma', y : T \vdash P \tag{31}$$
(Inversion on (T-LSOUT) and (25))
$$\Delta_1 \circ \Delta_2 \circ x : (B'_k \circ B''_k); \Gamma \vdash P_k[x_k \leftarrow y] \mid P \tag{32}$$
((31), (30), (27), (26) and (T-PAR))
$$\Delta = \Delta_1 \circ \Delta_2 \circ x : s \to r\{l_i(T).(B'_i \circ B''_i)\}_{i\in I} \tag{33}$$
((27))
$$\Delta \xrightarrow{s\to rl_k} \Delta_1 \circ \Delta_2 \circ x : (B'_k \circ B''_k) \tag{34}$$
((33))

(*Case* (Red-Par))

$$\Delta; \Gamma \vdash P_1 \mid P_2 \tag{35}$$
$$P_1 \mid P_2 \xrightarrow{\lambda} P'_1 \mid P_2 \tag{36}$$
(Assumption)
$$P_1 \xrightarrow{\lambda} P'_1 \tag{37}$$
(Inversion on (Red-Par) and (36))
$$\Delta = \Delta_1 \circ \Delta_2 \tag{38}$$
$$\Delta_2; \Gamma \vdash P_2 \tag{39}$$
$$\Delta_1; \Gamma \vdash P_1 \tag{40}$$
(Inversion on (T-PAR) and (35))

(**Case** $\lambda = \tau$)

$$\Delta_1; \Gamma \vdash P_1' \tag{41}$$
(Induction hypothesis on (40) and (37))
$$\Delta; \Gamma \vdash P_1' \,|\, P_2 \tag{42}$$
((41), (39), (38) and (Red-Par))

(**Case** $\lambda = x : s \to rl$ (1))

$$\Delta_1 = \Delta_1', x : B_1 \tag{43}$$
$$B_1 \xrightarrow{s \to rl} B_1' \tag{44}$$
$$\Delta_1', x : B_1'; \Gamma \vdash P_1' \tag{45}$$
(Induction hypothesis on (40) and (37))
$$\Delta_2 = \Delta_2', x : B_2 \tag{46}$$
$$\Delta = \Delta', x : B \tag{47}$$
$$\Delta' = \Delta_1' \circ \Delta_2' \tag{48}$$
$$B = B_1 \circ B_2 \tag{49}$$
((38))
$$B \xrightarrow{s \to rl} B' \tag{50}$$
$$B' = B_1' \circ B_2 \tag{51}$$
((49) and (44))
$$\Delta', x : B' \vdash P_1' \,|\, P_2 \tag{52}$$
((51), (48), (45), (39), (46) and (T-PAR))

(**Case** $\lambda = x : s \to rl$ (2))

$$\Gamma = \Gamma', x : T \tag{53}$$
$$\Delta_1; \Gamma \vdash P_1' \tag{54}$$
(Induction hypothesis on (40) and (37))
$$\Delta; \Gamma \vdash P_1' \,|\, P_2 \tag{55}$$
((54), (38), and (39))

($Case$ (RED-NEW1))

$$\Delta; \Gamma \vdash (\mathbf{new}\ x)P \tag{56}$$

$$(\mathbf{new}\ x)P \xrightarrow{\tau} (\mathbf{new}\ x)P' \tag{57}$$

(Assumption)

$$P \xrightarrow{\lambda} P' \tag{58}$$

$$\lambda = x : s \to rl \quad \vee \quad \lambda = \tau \tag{59}$$

(Inversion on (RED-NEW1) and (57))

(**Case** (T-NEW))

$$\Delta, x : B; \Gamma \vdash P \tag{60}$$

$$matched(B) \tag{61}$$

(Inversion on (T-NEW) and (56))

(**Case** $\lambda = \tau$)

$$\Delta, x : B; \Gamma \vdash P' \tag{62}$$

(Induction hypothesis on (60) and (58))

$$\Delta; \Gamma \vdash (\mathbf{new}\ x)P' \tag{63}$$

((62), (61) and (T-NEW))

(**Case** $\lambda = x : s \to rl$)

$$B \xrightarrow{s \to rl} B' \tag{64}$$

$$\Delta, x : B'; \Gamma \vdash P' \tag{65}$$

(Induction hypothesis on (60) and (58))

$$matched(B') \tag{66}$$

((61) and (64))

$$\Delta; \Gamma \vdash (\mathbf{new}\ x)P' \tag{67}$$

((65), (66) and (T-NEW))

(**Case** (T-SNEW))

$$\Delta; \Gamma, x : l(B) \vdash P \tag{68}$$

(Inversion on (T-SNEW) and (56))

(**Case** $\lambda = \tau$)

$$\Delta; \Gamma, x : l(B) \vdash P' \tag{69}$$

(Induction hypothesis on (68) and (58))

$$\Delta; \Gamma \vdash (\mathbf{new}\ x)P' \tag{70}$$

((69) and (T-SNEW))

(**Case** $\lambda = x : s \to rl$)

$$\Delta; \Gamma, x : l(B) \vdash P' \tag{71}$$

(Induction hypothesis on (68) and (58))

$$\Delta; \Gamma \vdash (\mathbf{new}\ x)P' \tag{72}$$

((71) and (T-SNEW))

(*Case* (Red-New2))

$$\Delta; \Gamma \vdash (\mathbf{new}\ y)P \tag{73}$$

$$(\mathbf{new}\ y)P \xrightarrow{\lambda} (\mathbf{new}\ y)P' \tag{74}$$

(Assumption)

$$P \xrightarrow{\lambda} P' \tag{75}$$

$$\lambda = x : s \to rl \ \ (x \neq y) \tag{76}$$

(Inversion on (Red-New2) and (74))

(**Case** (T-NEW))

$$\Delta, y : B_1; \Gamma \vdash P \tag{77}$$

$$matched(B_1) \tag{78}$$

(Inversion on (T-NEW) and (73))

(**Case** $\lambda = x : s \to rl$ (1))

$$\Delta, y : B_1 = \Delta', y : B_1, x : B \tag{79}$$

$$B \xrightarrow{s \to rl} B' \tag{80}$$

$$\Delta', y : B_1, x : B'; \Gamma \vdash P' \tag{81}$$

(Induction hypothesis on (77) and (75))

$$\Delta', x : B'; \Gamma \vdash (\mathbf{new}\ y)P' \tag{82}$$

((81), (78) and (T-NEW))

($\mathbf{Case}$ $\lambda = x : s \to rl$ (2))

$$\Gamma = \Gamma', x : T \tag{83}$$

$$\Delta, y : B_1; \Gamma \vdash P' \tag{84}$$

(Induction hypothesis on (77) and (75))

$$\Delta; \Gamma \vdash (\mathbf{new}\ y)P' \tag{85}$$

((84), (78) and (T-NEW))

($\mathbf{Case}$ (T-SNEW))

$$\Delta; \Gamma, y : l(B_1) \vdash P \tag{86}$$

(Inversion on (T-SNEW) and (73))

($\mathbf{Case}$ $\lambda = x : s \to rl$ (1))

$$\Delta = \Delta', x : B \tag{87}$$

$$B \xrightarrow{s \to rl} B' \tag{88}$$

$$\Delta', x : B'; \Gamma, y : l(B_1) \vdash P' \tag{89}$$

(Induction hypothesis on (86) and (75))

$$\Delta', x : B'; \Gamma \vdash (\mathbf{new}\ y)P' \tag{90}$$

((89) and (T-SNEW))

($\mathbf{Case}$ $\lambda = x : s \to rl$ (2))

$$\Gamma, y : l(B_1) = \Gamma', y : l(B_1), x : T \tag{91}$$

$$\Delta; \Gamma, y : l(B_1) \vdash P' \tag{92}$$

(Induction hypothesis on (86) and (75))

$$\Delta; \Gamma \vdash (\mathbf{new}\ y)P' \tag{93}$$

((92) and (T-SNEW))

(*Case* (Red-Struct))

$$\Delta; \Gamma \vdash P_1 \tag{94}$$

$$P_1 \xrightarrow{\lambda} P_2 \tag{95}$$

(Assumption)

$$P_1 \equiv P_1' \tag{96}$$

$$P_1' \xrightarrow{\lambda} P_2' \tag{97}$$

$$P_2 \equiv P_2' \tag{98}$$

(Inversion on (Red-Struct) and (95))

$$\Delta; \Gamma \vdash P_1' \tag{99}$$

((94) and Lemma C.3)

(**Case** $\lambda = \tau$)

$$\Delta; \Gamma \vdash P_2' \tag{100}$$

(Induction hypothesis on (99) and (97))

$$\Delta; \Gamma \vdash P_2 \tag{101}$$

((100) and Lemma C.3)

(**Case** $\lambda = x : s \to rl$ (1))

$$\Delta = \Delta', x : B \tag{102}$$

$$B \xrightarrow{s \to rl} B' \tag{103}$$

$$\Delta', x : B'; \Gamma \vdash P_2' \tag{104}$$

(Induction hypothesis on (99) and (97))

$$\Delta', x : B'; \Gamma \vdash P_2 \tag{105}$$

((104) and Lemma C.3)

(**Case** $\lambda = x : s \to rl$ (2))

$$\Gamma = \Gamma', x : T \tag{106}$$

$$\Delta; \Gamma \vdash P_2' \tag{107}$$

(Induction hypothesis on (99) and (97))

$$\Delta; \Gamma \vdash P_2 \tag{108}$$

((107) and Lemma C.3)