Dynamic Roles in Multiparty Communicating Systems

Pedro Baltazar
Universidade de Lisboa, Faculdade de Ciências, LaSIGE

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 March 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 the communication protocols without relying on centralized control, and verification mechanisms that automatically check if the code meets some common protocol specification are 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 a single site changes accordingly.

A challenge that needs to be overcome 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 [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 [2]. While 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 realistic 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:

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

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). A non surprising implementation of this interaction is given by:

$$\mathit{chat} \lhd_{\mathsf{Sender}} \mathit{hello}().\mathit{chat} \lhd_{\mathsf{Sender}} \mathit{bye}() \mid \mathit{chat} \rhd_{\mathsf{Receiver}} \mathit{hello}().\mathit{chat} \rhd_{\mathsf{Receiver}} \mathit{bye}()$$

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

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:

$$\mathit{chat} \lhd_{\mathsf{Sender}} \mathit{hello}().\mathit{chat} \rhd_{\mathsf{Receiver}} \mathit{bye}() \mid \mathit{chat} \rhd_{\mathsf{Receiver}} \mathit{hello}().\mathit{chat} \lhd_{\mathsf{Sender}} \mathit{bye}()$$

where the process on the left sends message *hello* as Sender and then receives message *bye* as Receiver, described by type $!\mathsf{Sender}\ \mathit{hello}().\ ?\mathsf{Receiver}\ \mathit{bye}()$, and the process on the right first acts first as Receiver and then as Sender, described by type $?\mathsf{Receiver}\,\mathit{hello}().\ !\mathsf{Sender}\ \mathit{bye}()$. 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

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

is decomposed in "complementary" types that describe the behavior of the individual processes (for instance, in type $!\mathsf{Sender}\,\mathit{hello}().\,?\mathsf{Receiver}\,\mathit{bye}()$ and type $?\mathsf{Receiver}\ \mathit{hello}().\ !\mathsf{Sender}\ \mathit{bye}()$).

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

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

which captures the interactions in a purchase system involving three parties, and which captures the interaction illustrated in Fig. 1. 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

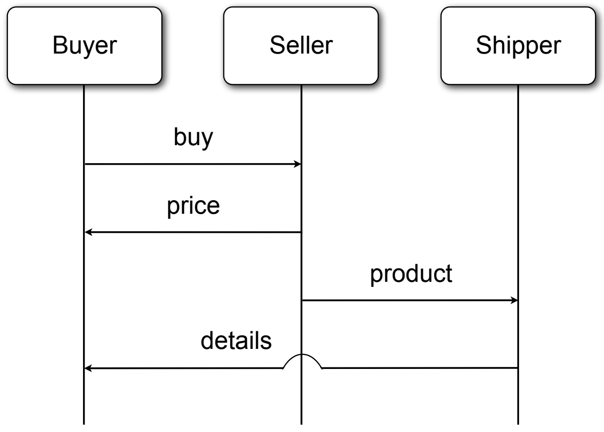

Figure 1: Purchase Interaction Message Sequence Chart.

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

Figure 2: Purchase System Code (a).

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. 2 shows a possible implementation of the purchase interaction, that follows the protocol specified in (1) and where roles and processes are in a one to one correspondence. And in Fig. 3 a different implementation is shown, where now the roles and the processes are intermixed. Using the **new** construct, process `Buyer` creates a fresh chat that will host the purchase interaction. This newly created name is passed to a shop, via message *buyService*. Code $\mathit{Shop} \triangleleft_{\mathsf{Buyer}} \mathit{buyService}(\mathit{chat})$ represents the output of message *buyService* on channel *Shop*, passing name *chat* under role Buyer.

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

Figure 3: Purchase System Code (b).

The `Buyer` process then sends message *buy*, after which 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 exchange 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. 3 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*):

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

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

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

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. 3 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 $?\mathsf{Buyer}\ \mathit{price}()$ and $\Diamond?\mathsf{Buyer}\ \mathit{details}()$, 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 message *product*, in $!\mathsf{Seller}\ \mathit{product}()$ and $?\mathsf{Shipper}\ \mathit{product}().!\mathsf{Shipper}\ \mathit{details}()$ 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, e.g.,:

$$\begin{aligned}&\mathsf{Seller} \to \mathsf{Shipper}\ \mathit{product}().\ !\mathsf{Shipper}\ \mathit{details}() = \\ &\qquad\qquad ?\mathsf{Shipper}\ \mathit{product}().\ !\mathsf{Shipper}\ \mathit{details}()\ \circ\ !\mathsf{Seller}\ \mathit{product}()\end{aligned}$$

The purchase interaction of the system shown in Fig. 3 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. 4, 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

$$P ::= \mathbf{0} \;[\!]\; (\mathbf{new}\ x)P \;[\!]\; P_1 \,|\, P_2 \;[\!]\; *P \;[\!]\; x \triangleright_r \{l_i(x_i).P_i\}_{i \in I} \;[\!]\; x \triangleleft_r l(y).P$$
$$l \in \mathcal{L}(abels) \qquad x, y \in \mathcal{N}(ames) \qquad r, s \in \mathcal{R}(oles)$$

Figure 4: Process Syntax.

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 $P_1 \,|\, P_2$ 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 $fn(P)$ 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 $\equiv$, given by the rules in Fig. 5.

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:

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

Figure 5: Structural Congruence.

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

$$\frac{P_1 \xrightarrow{\lambda} P_1'}{P_1 \mid P_2 \xrightarrow{\lambda} P_1' \mid P_2} \qquad \frac{P \xrightarrow{\lambda} P' \qquad \lambda \in \{\tau, x : s \to rl\}}{(\mathbf{new}\ x)P \xrightarrow{\tau} (\mathbf{new}\ x)P'}$$
(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 6: Reduction Relation.

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. 6, 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 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.

We illustrate reduction using the purchase interaction example discussed in the Introduction. Consider the following reduction (the code is taken from

Fig. 2):

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

The only possible interaction in the system is the exchange of the *buyService* message in name *Shop* 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 $\mathit{Shop} : \mathsf{Buyer} \to \mathsf{Seller}\mathit{buyService}()$, leads to the system:

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

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:

$$
\begin{array}{l}
(\textbf{new}\ \mathit{chat}) \\
(\mathit{chat} \rhd_{\mathsf{Buyer}} \mathit{details}() \\
| \\
\mathit{Carrier} \lhd_{\mathsf{Seller}} \mathit{shipService}(\mathit{chat}).\mathit{chat} \lhd_{\mathsf{Seller}} \mathit{product}()) \\
| \\
\mathit{Carrier} \rhd_{\mathsf{Shipper}} \mathit{shipService}(x). \\
\quad x \rhd_{\mathsf{Shipper}} \mathit{product}().x \lhd_{\mathsf{Shipper}} \mathit{details}()
\end{array}
$$

By now message *shipService* may be exchanged in name *Carrier*, where channel *chat* is sent to *Carrier*, allowing for a third-party to join the ongoing interaction. Notice *Shop* and *Carrier* get to interact on the delegated channel *chat* afterwords, exchanging message *product*. Notice also that label

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

Figure 7: Conversation Types Syntax.

$Shop : \mathsf{Shipper} \to \mathsf{Seller} shipService()$ 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.

3 Type System

In this section we present our type system. The type language is given in Fig. 7, 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 sometime $\Diamond B$ 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 (if ρ is $?r$), an output choice (if 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

$$\mathbf{end} \vdash \mathbf{wf} \qquad \text{(WF-END)}$$

$$\frac{B_1 \# B_2 \qquad B_1 \vdash \mathbf{wf} \qquad B_2 \vdash \mathbf{wf}}{B_1 \,|\, B_2 \vdash \mathbf{wf}} \qquad \text{(WF-PAR)}$$

$$\frac{\forall i \in I \qquad B_i \vdash \mathbf{wf} \qquad \rho\{l_i(M_i).\mathbf{end}\} \# B_i}{\rho\{l_i(M_i).B_i\}_{i \in I} \vdash \mathbf{wf}} \qquad \text{(WF-COM)}$$

$$\frac{\rho\{l_i(M_i).B_i\} \vdash \mathbf{wf} \qquad \rho \in \{!s, ?r\}}{\Diamond \rho\{l_i(M_i).B_i\}_{i \in I} \vdash \mathbf{wf}} \qquad \text{(WF-SM1)}$$

$$\frac{\Diamond B_1 \vdash \mathbf{wf} \qquad \Diamond B_2 \vdash \mathbf{wf}}{\Diamond (B_1 \,|\, B_2) \vdash \mathbf{wf}} \qquad \text{(WF-SM2)}$$

Figure 8: Type Well-Formedness.

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 well-formed types to capture race-free behavioral descriptions. We say type B is well-formed if $B \vdash \mathbf{wf}$ can be derived by the rules given in Fig. 8. Informally, in a well-formed type labels do not appear repeatedly in parallel (to ensure race-free behavior) or in sequence (useful to simplify presentation). Also well-formed $\Diamond$ types are not applied directly to message exchange ($s \to r$) polarities. In the remaining technical presentation (e.g., in typing rules) we implicitly assume types are well-formed. Also used by our typing is the notion of matched types, which capture systems where all input actions have a matching output. We say that type B is matched, noted $matched(B)$, if all communication prefixes in B are of the form $s \to r$.

The subtyping relation, noted $<:$, between behavioral types is given in Fig. 9, where we use $B_1 \equiv B_2$ when $B_1 <: B_2$ and $B_2 <: B_1$. We distinguish

$$B_1 <: B_1' \implies B_1 \,|\, B_2 <: B_1' \,|\, B_2$$
$$B_i <: B_i' \implies \rho\{l_i(M_i).B_i\}_{i\in I} <: \rho\{l_i(M_i).B_i'\}_{i\in I}$$
$$(B_1 \,|\, B_2) \,|\, B_3 \equiv B_1 \,|\, (B_2 \,|\, B_3)$$
$$B_1 \,|\, B_2 \equiv B_2 \,|\, B_1 \qquad B \,|\, \mathbf{end} \equiv B$$
$$\Diamond(B_1 \,|\, B_2) \equiv \Diamond B_1 \,|\, \Diamond B_2 \qquad \Diamond\mathbf{end} \equiv \mathbf{end} \qquad B <: \Diamond B$$

Figure 9: Subtyping Relation.

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. 10, 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 ($\Diamond$). 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-EQU) closes the relation under type equivalence.

To simplify the presentation we elide the symmetric counterparts of the rules shown in Fig. 10. Also, we sometimes use $B_1 \circ B_2$ 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

$$B = B \circ \mathbf{end} \qquad B = \mathbf{end} \circ B \qquad \frac{B_1 = B_1' \circ B_1'' \qquad B_2 = B_2' \circ B_2''}{B_1 \mid B_2 = B_1' \mid B_2' \circ B_1'' \mid B_2''}$$
(S-END,S-END-1,S-PAR)

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

$$\frac{B_i = B_i' \circ B_i'' \qquad \forall i \in I}{r_1 \to r_2\{l_i(M_i).B_i\}_{i\in I} = \Diamond !r_1\{l_i(M_i).B_i'\}_{i\in I} \circ ?r_2\{l_i(M_i).B_i''\}_{i\in I}}$$
(S-TAU-1)

$$\frac{B_i = B_i' \circ B_i'' \qquad \forall i \in I}{r_1 \to r_2\{l_i(M_i).B_i\}_{i\in I} = ?r_2\{l_i(M_i).B_i'\}_{i\in I} \circ \Diamond !r_1\{l_i(M_i).B_i''\}_{i\in I}}$$
(S-TAU-2)

$$\frac{B_i = B_i' \circ B_i'' \qquad \forall i \in I}{r_1 \to r_2\{l_i(M_i).B_i\}_{i\in I} = \Diamond ?r_2\{l_i(M_i).B_i'\}_{i\in I} \circ !r_1\{l_i(M_i).B_i''\}_{i\in I}}$$
(S-TAU-3)

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

$$\frac{B_i = \Diamond B \circ B_i' \qquad \forall i \in I}{\rho\{l_i(M_i).B_i\}_{i\in I} = \Diamond B \circ \rho\{l_i(M_i).B_i'\}_{i\in I}} \quad \text{(S-BRK-1)}$$

$$\frac{B_1' = B_2' \circ B_3' \qquad B_1 \equiv B_1' \qquad B_2 \equiv B_2' \qquad B_3 \equiv B_3'}{B_1 = B_2 \circ B_3} \quad \text{(S-EQU)}$$

Figure 10: Type Splitting Relation.

distinct types. Notice also that a type may be split in several ways. In prior work on conversation types [3], we have called "merge" to type-splitting, 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 state a basic property of splitting, crucial to our type system which relies on the flexibility of the type decomposition to address the behavioral contributions of multiple parties.

Proposition 3.1 (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 of $B = B_1 \circ B'$ (see Appendix).

We may now present the type system. A typing judgment is of the form $\Gamma; \Delta \vdash P$ where Γ is the typing environment which describes the interactions of P on linear channels, and Δ is the typing environment which describes the interactions of P on shared channels (we write $\Gamma; \Delta$ only when the domains of Γ and Δ are disjoint). Thus, a typing environment Γ is an assignment of identifiers to behavioral types ($\Gamma \triangleq x_1 : B_1, \ldots, x_k : B_k$) and a typing environment Δ is an assignment of identifiers to message exchange types ($\Delta \triangleq x_1 : T_1, \ldots, x_k : T_k$). We introduce some auxiliary notation to simplify presentation: we use $(x_1 : B'_1, \ldots, x_k : B'_k, \Gamma_1) \circ (x_1 : B''_1, \ldots, x_k : B''_k, \Gamma_2)$ to denote $x_1 : B_1, \ldots, x_k : B_k, \Gamma_1, \Gamma_2$ such that $B_i = B'_i \circ B''_i$, for all i in $1, \ldots, k$ and the domains of Γ_1 and Γ_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 $1, \ldots, k$.

We say process P is well-typed if $\Gamma; \Delta \vdash P$ may be derived using the rules given in Fig. 11. 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). Rules (T-LSIN) and (T-LSOUT) follow similar lines, addressing the case

$$x : \mathbf{end}; \Delta \vdash \mathbf{0} \qquad \frac{\Gamma_1; \Delta \vdash P_1 \qquad \Gamma_2; \Delta \vdash P_2}{\Gamma_1 \circ \Gamma_2; \Delta \vdash P_1 \,|\, P_2} \qquad \text{(T-END,T-PAR)}$$

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

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

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

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

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

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

$$\frac{\Gamma_1; \Delta \vdash P \quad \Gamma_1 <: \Gamma_2}{\Gamma_2; \Delta \vdash P} \qquad \frac{x : \mathbf{end}; \Delta \vdash P}{x : \mathbf{end}; \Delta \vdash *P} \qquad \text{(T-SUB,T-REP)}$$

Figure 11: Typing Rules.

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 may now present our results, starting by the key auxiliary lemmas that ensure that typing is preserved by substitution and by structural congruence.

Lemma 3.2 (Substitution) *If* $\Gamma; \Delta \vdash P$ *and*

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

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

Figure 12: Type Reduction.

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

Lemma 3.3 (Subject Congruence) *If* $\Gamma; \Delta \vdash P$ *and* $P \equiv P'$ *then* $\Gamma; \Delta \vdash P'$.

Proof. By induction on the length of the derivation of $P \equiv P'$ following expected lines. The proof crucially builds on the fact that the split relation is symmetric (by definition) and associative (Proposition 3.1).

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. 12. 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.

We state our main result that explains process reduction via type reduction.

Theorem 3.4 (Type Preservation) *Let* $\Gamma; \Delta \vdash P$ *and* $P \xrightarrow{\lambda} P'$.

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

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

Theorem 3.4 states that any reduction of a well-typed process is explained by the corresponding type reduction, thus ensuring 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 state this property as a direct conclusion of Theorem 3.4.

Corollary 3.5 (Role-Based Protocol Fidelity) *If* P *is well-typed then interactions in* P *follow the role-based protocols prescribed by the types.*

We proceed to typing an example to provide further intuition. Returning to Fig. 2, 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 the following decomposition by using rules [S-END] and [S-TAU]. ($\mathsf{e} = \mathsf{S\underline{e}ller}$ and $\mathsf{h} = \mathsf{S\underline{h}ipper}$)

$$\frac{!\mathsf{h}\{\mathit{details}().\mathbf{end}\} = \mathbf{end} \circ !\mathsf{h}\{\mathit{details}().\mathbf{end}\}}{\mathsf{e} \to \mathsf{h}\{\mathit{product}().!\mathsf{h}\{\mathit{details}().\mathbf{end}\}\} = !\mathsf{e}\{\mathit{product}().\mathbf{end}\} \circ ?\mathsf{h}\{\mathit{product}().!\mathsf{h}\{\mathit{details}().\mathbf{end}\}\}}$$

Now, the splitting example given above appears when typing the subprocess

$$\begin{array}{c} \mathit{Carrier} \lhd_{\mathsf{Seller}} \mathit{shipService}(\mathit{chat}).\mathit{chat} \lhd_{\mathsf{Seller}} \mathit{product}() \mid \\ \mathit{Carrier} \rhd_{\mathsf{Shipper}} \mathit{shipService}(x).x \rhd_{\mathsf{Shipper}} \mathit{product}().x \lhd_{\mathsf{Shipper}} \mathit{details}() \end{array}$$

Here, the delegation of name *chat*, through service *shipService*, requires that the behavior of *chat* to be split between the two processes.

In Figure 13, we show another variation of our example, where now Buyer tells the Seller which shipping service he prefers. This example illustrates the use of different IN/OUT typing rules. In the case of

$$\mathit{Shop} \lhd_{\mathsf{Buyer}} \mathit{buyService}(\mathit{chat})$$

an output of a linear on a shared name is typed by rule [T-SOUT]. The inverse case, a shared name is sent through a linear name

$$\mathit{chat} \lhd_{\mathsf{Buyer}} \mathit{buy}(\mathit{Shipper})$$

is covered by rule [T-LSOUT]. On the other hand, receiving a shared in a linear name

$$x \rhd_{\mathsf{Seller}} \mathit{buy}(y)$$

corresponds to rule [T-LSIN].

From Proposition 3.1, we conclude that the order by which the participants enter the conversation is not relevant for its soundness. For example, if B is matched and $B = B_1 \circ B'$ with $B' = B_2 \circ B_3$, then associativity yields $B = B_1 \circ B_2 \circ B_3$. In other words, one can split B in three B_1, B_2, B_3 by any order. Now, suppose $\Gamma_i, x : B_i \vdash P_i$, $i = 1, 2, 3$. In order to re-construct the type B for x in $P_1 \mid P_2 \mid P_3$ one can also use any order. In our running example (Fig. 2) the types of channel *chat* in processes `Buyer` and `Shop` are, respectively:

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

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

where the latter shows that `Shop` retains the Shipper contribution to the overall interaction, which can then be later on delegated to `Carrier`. The latter

$$
\begin{array}{lcl}
\texttt{Buyer} & \triangleq & (\mathbf{new}\ chat) \\
 & & Shop \lhd_{\mathsf{Buyer}}\ buyService(chat). \\
 & & \quad chat \lhd_{\mathsf{Buyer}}\ buy(\ Shipper\ ). \\
 & & \quad chat \rhd_{\mathsf{Buyer}}\ price(). \\
 & & \quad chat \rhd_{\mathsf{Buyer}}\ details() \\
\texttt{Shop} & \triangleq & \ Shop \rhd_{\mathsf{Seller}}\ buyService(x). \\
 & & \quad x \rhd_{\mathsf{Seller}}\ buy(\ y\ ). \\
 & & \quad x \lhd_{\mathsf{Seller}}\ price(). \\
 & & \quad \ y\ \lhd_{\mathsf{Seller}}\ shipService(x). \\
 & & \qquad x \lhd_{\mathsf{Seller}}\ product()
\end{array}
$$

Figure 13: Purchase System Code (c).

is also specified in the argument type of message *buyService*, identifying the behavior delegated to the seller upon synchronization in the *buyService* message. The actual contribution of `Carrier` to the interaction in *chat* is captured by the following type (which is the argument type of message *shipService*):

$$?\mathsf{Shipper}\ product().!\mathsf{Shipper}\ details()$$

So `Shop` "directly" contributes to the interaction with the behavior:

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

The type specification (1) can be recovered by simulating the conversation taking place in *chat* and recording its trace. Typing the alternative purchase interaction shown in Fig. 3 follows similar lines, the difference is that type splitting allows to mix different roles in the contributions of each process to the overall interaction. Namely, the behavior of the `Buyer` process in the purchase interaction implementation shown in Fig. 3 is the same as before:

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

but where `Buyer` later on delegates the capability to receive *details* to `MailBox`, which behavior is captured by the type (specified in the *storeService* type):

$$?\mathsf{Buyer}\ details()$$

The `Shop` type is also the same as before (*buyService* has the same type):

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

but now no delegation of behavior to an external partner takes place, and `Shop` actually impersonates both the role of Seller and the role of Shipper.

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]) so as to address role-based protocol verification (labeled channels and role annotations) and extended the type analysis accordingly. Preliminary ideas of this work were presented in [1]. Noticeably, the splitting relation defined in this paper is much more readable and also more expressive — 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 can directly reuse the progress proof system introduced in [3] for a labeled π-calculus, which, 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-base 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 which have an unanticipated number of participants. Other works share the goal to address multiparty interaction, namely [5, 12, 2, 10]. We distinguish the approach of conversation types 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 [5, 2, 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] Pedro Baltazar, Vasco T. Vasconcelos, and Hugo T. Vieira. Typing Dynamic Roles in Multiparty Interaction. In *INForum 2011*. Universidade de Coimbra, 2011.

[2] 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.

[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

Proposition 3.1 (Associativity)
(repetition of the statement in page 13)
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, $B = B_1^\bullet \,|\, B_2^\bullet$ with $B_1 = B_{11}^\bullet \,|\, B_{21}^\bullet$ and $B' = B_{12}^\bullet \,|\, B_{22}^\bullet$, such that

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

(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

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

Where $B_2 = B^\bullet_{121} \mid B^\bullet_{221}$ and $B_3 = B^\bullet_{122} \mid B^\bullet_{222}$.
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

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

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

(c) The case $B' = B_2 \circ B_3$ by rule [S-EQU] is a straightforward application of the induction hypothesis and subtype relation.

2. [S-TAU] is the last rule in $B = B_1 \circ B'$. Without loss of generality we drop the modality from the prefixes, since is not relevant for this case.

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

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], [S-EQU] or [S-BRK]. The cases of rules [S-END] and [S-EQU] are straightforward. Now, if $B' = B_2 \circ B_3$ comes by rule [S-BRK] we have also two cases.

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

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

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

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

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

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

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

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

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

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

Once more, two cases to be considered.

(a) $B_1 = \rho\{l_i(M_i).B^\bullet_i\}_i$ and $B' = \Diamond B^\bullet$. Clearly, if $B' = B_2 \circ B_3$, then $\Diamond B^\bullet = \Diamond B^\bullet_2 \circ \Diamond B^\bullet_3$. Now, applying the induction hypothesis to $\Diamond B^\bullet_3$ and $B_i = B^\bullet_i \circ \Diamond B^\bullet$ we have $B_i = (B^\bullet_i \circ \Diamond B^\bullet_2) \circ \Diamond B^\bullet_3$. From applying [S-BRK] and then [S-TAU] we conclude that $\rho\{l_i(M_i).B_i\}_i = \rho\{l_i(M_i).(B^\bullet_i \circ \Diamond B^\bullet_2)\}_i \circ \Diamond B^\bullet_3$ and that $\rho\{l_i(M_i).(B^\bullet_i \circ \Diamond B^\bullet_2)\}_i = \rho\{l_i(M_i).B^\bullet_i\}_i \circ \Diamond B^\bullet_2$. Hence, $\rho\{l_i(M_i).B_i\}_i = (\rho\{l_i(M_i).B^\bullet_i\}_i \circ \Diamond B^\bullet_2) \circ \Diamond B^\bullet_3$.

(b) $B' = \rho\{l_i(M_i).B_i^\bullet\}_i$ and $B_1 = \diamond B^\bullet$. In this case, we have to consider all the possible last rule of the split $B' = \rho\{l_i(M_i).B_i^\bullet\}_i = B_2 \circ B_3$. The cases [S-END] and [S-EQU] are straightforward.

   i. [S-TAU] Analogous to the sub-case $(a) - (i)$ of case (2).

   ii. [S-BRK] Analogous to the sub-case $(a) - (ii)$ of case (2).

4. [S-EQU] is the last rule in $B = B_1 \circ B'$. Straightforward.

Theorem 3.4 (Type Preservation)
(repetition of the statement in page 16)
If $\Gamma; \Delta \vdash P$ *and* $P \xrightarrow{\lambda} P'$ *and*

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

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

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

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

(Assumption)

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

$$\Gamma_1; \Delta', x : l_1(B) \vdash x \triangleright_r \{l_1(x_1).P_1\} \tag{3}$$

$$I = \{1\}, k = 1 \tag{4}$$

((T-SIN))

$$\Gamma_2 \circ y : B; \Delta', x : l_1(B) \vdash x \triangleleft_s l_1(y).P \tag{5}$$

((T-SOUT))

$$\Gamma = \Gamma_1 \circ \Gamma_2 \circ y : B \tag{6}$$

$$\Delta = \Delta', x : l_1(B) \tag{7}$$

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

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

$$\Gamma_1, x_1 : B; \Delta', x : l_1(B) \vdash P_1 \tag{9}$$

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

$\Gamma_1 \circ y : B$ defined (10)

((6))

$\Gamma_1 \circ y : B; \Delta', x : l_1(B) \vdash P_1[x_1 \leftarrow y]$ (11)

((9) and (10) and Lemma 3.2)

$\Gamma_2; \Delta', x : l_1(B) \vdash P$ (12)

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

$\Gamma; \Delta \vdash P_1[x_1 \leftarrow y] \,|\, P$ (13)

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

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

$\Gamma_1 \circ x : B_1; \Delta \vdash x \triangleright_r \{l_i(x_i).P_i\}_{i \in I}$ (14)

((T-IN))

$\Gamma_2 \circ x : B_2 \circ y : B'_k; \Delta \vdash x \triangleleft_s l_k(y).P$ (15)

((T-OUT))

$\Gamma = \Gamma_1 \circ \Gamma_2 \circ x : B_1 \circ x : B_2 \circ y : B'_k$ (16)

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

$\forall i \in I \qquad \Gamma_1 \circ x : B_i, x_i : B'_i; \Delta \vdash P_i$ (17)

$?r\{l_i(B'_i).B_i\}_{i \in I} <: B_1$ (18)

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

$\Gamma_2 \circ x : B''_k; \Delta \vdash P$ (19)

$!s\{l_i(B'_i).B''_i\}_{i \in I} <: B_2$ (20)

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

(**Case** $B_1 \equiv ?r\{l_i(B'_i).B_{1_i}\}_{i \in I} \wedge B_2 \equiv \Diamond !s\{l_i(B'_i).B_{2_i}\}_{i \in I}$)

$B_1 \equiv ?r\{l_i(B'_i).B_{1_i}\}_{i \in I}$ (21)

$B_2 \equiv \Diamond !s\{l_i(B'_i).B_{2_i}\}_{i \in I}$ (22)

((Assumption))

$\forall_{i \in I}\ B_i <: B_{1_i}$ (23)

((18) and (21))

$\Gamma_1 \circ x : B_{1_k} \circ y : B'_k$ defined (24)

((16), (18), (23) and (21))

$\Gamma_1 \circ x : B_{1_k} \circ y : B'_k; \Delta \vdash P_k[x_k \leftarrow y]$ (25)

((17) and (24) and Lemma 3.2)

$$s \to r\{l_i(B_i').(B_{1_i} \circ B_{2_i})\}_{i\in I} = B_1 \circ B_2 \tag{26}$$
((21) and (22) and (16))

$$\forall_{i\in I}\, B_i'' <: B_{2_i} \tag{27}$$
((20) and (22))

$$\Gamma_2 \circ x : B_{2_k}; \Delta \vdash P \tag{28}$$
((19) and (27) and (T-SUB))

$$\Gamma_1 \circ \Gamma_2 \circ x : (B_{1_k} \circ B_{2_k}) \circ y : B_k'; \Delta \vdash P_k[x_k \leftarrow y] \,|\, P \tag{29}$$
((28), (25), (16) and (T-PAR))

$$\Gamma = \Gamma_1 \circ \Gamma_2 \circ x : s \to r\{l_i(B_i').(B_{1_i} \circ B_{2_i})\}_{i\in I} \circ y : B_k' \tag{30}$$
((16) and (26))

$$\Gamma \xrightarrow{s\to r l_k} \Gamma_1 \circ \Gamma_2 \circ x : (B_{1_k} \circ B_{2_k}) \circ y : B_k' \tag{31}$$
((30))

(**Case** $B_1 \equiv \Diamond ?r\{l_i(B_i').B_{1_i}\}_{i\in I} \wedge B_2 \equiv !s\{l_i(B_i').B_{2_i}\}_{i\in I}$)

Follows lines similar to the previous case.

(**Case** $B_1 \equiv ?r\{l_i(B_i').B_{1_i}\}_{i\in I} \wedge B_2 \equiv !s\{l_i(B_i').B_{2_i}\}_{i\in I}$)

Impossible since $B_1 \circ B_2$ is not defined which contradicts (16).

(**Case** $B_1 \equiv \Diamond ?r\{l_i(B_i').B_{1_i}\}_{i\in I} \wedge B_2 \equiv \Diamond !s\{l_i(B_i').B_{2_i}\}_{i\in I}$)

Impossible since $B_1 \circ B_2$ is not defined which contradicts (16).

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

Follows lines similar to the previous case.

(*Case* (Red-Par))

$$\Gamma;\Delta \vdash P_1 \,|\, P_2 \tag{32}$$

$$P_1 \,|\, P_2 \xrightarrow{\lambda} P_1' \,|\, P_2 \tag{33}$$

(Assumption)

$$P_1 \xrightarrow{\lambda} P_1' \tag{34}$$

(Inversion on (Red-Par) and (33))

$$\Gamma = \Gamma_1 \circ \Gamma_2 \tag{35}$$

$$\Gamma_2;\Delta \vdash P_2 \tag{36}$$

$$\Gamma_1;\Delta \vdash P_1 \tag{37}$$

(Inversion on (T-PAR) and (32))

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

$$\Gamma_1;\Delta \vdash P_1' \tag{38}$$

(Induction hypothesis on (37) and (34))

$$\Gamma;\Delta \vdash P_1' \,|\, P_2 \tag{39}$$

((38), (36), (35) and (Red-Par))

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

$$\Gamma_1 = \Gamma_1', x : B_1 \tag{40}$$

$$B_1 \xrightarrow{s\to rl} B_1' \tag{41}$$

$$\Gamma_1', x : B_1';\Delta \vdash P_1' \tag{42}$$

(Induction hypothesis on (37) and (34))

$$\Gamma_2 = \Gamma_2', x : B_2 \tag{43}$$

$$\Gamma = \Gamma', x : B \tag{44}$$

$$\Gamma' = \Gamma_1' \circ \Gamma_2' \tag{45}$$

$$B = B_1 \circ B_2 \tag{46}$$

((35))

$$B \xrightarrow{s\to rl} B' \tag{47}$$

$$B' = B_1' \circ B_2 \tag{48}$$

((46) and (41) and $B \vdash \mathbf{wf}$)

$$\Gamma', x : B' \vdash P_1' \,|\, P_2 \tag{49}$$

((48), (45), (42), (36), (43) and (T-PAR))

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

$\Delta = \Delta', x : T$ (50)

$\Gamma_1; \Delta \vdash P_1'$ (51)

(Induction hypothesis on (37) and (34))

$\Gamma; \Delta \vdash P_1' \,|\, P_2$ (52)

((51), (35), and (36))

(*Case* (Red-New1))

$\Gamma; \Delta \vdash (\textbf{new } x)P$ (53)

$(\textbf{new } x)P \xrightarrow{\tau} (\textbf{new } x)P'$ (54)

(Assumption)

$P \xrightarrow{\lambda} P'$ (55)

$\lambda = x : s \rightarrow rl \quad \vee \quad \lambda = \tau$ (56)

(Inversion on (Red-New1) and (54))

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

$\Gamma, x : B; \Delta \vdash P$ (57)

$\mathit{matched}(B)$ (58)

(Inversion on (T-NEW) and (53))

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

$\Gamma, x : B; \Delta \vdash P'$ (59)

(Induction hypothesis on (57) and (55))

$\Gamma; \Delta \vdash (\textbf{new } x)P'$ (60)

((59), (58) and (T-NEW))

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

$B \xrightarrow{s \rightarrow rl} B'$ (61)

$\Gamma, x : B'; \Delta \vdash P'$ (62)

(Induction hypothesis on (57) and (55))

$\mathit{matched}(B')$ (63)

((58) and (61))

$\Gamma; \Delta \vdash (\textbf{new } x)P'$ (64)

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

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

$\Gamma; \Delta, x : l(B) \vdash P$ (65)

(Inversion on (T-SNEW) and (53))

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

$\Gamma; \Delta, x : l(B) \vdash P'$ (66)

(Induction hypothesis on (65) and (55))

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

((66) and (T-SNEW))

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

$\Gamma; \Delta, x : l(B) \vdash P'$ (68)

(Induction hypothesis on (65) and (55))

$\Gamma; \Delta \vdash (\mathbf{new}\ x)P'$ (69)

((68) and (T-SNEW))

(*Case* (Red-New2))

$\Gamma; \Delta \vdash (\mathbf{new}\ y)P$ (70)

$(\mathbf{new}\ y)P \xrightarrow{\lambda} (\mathbf{new}\ y)P'$ (71)

(Assumption)

$P \xrightarrow{\lambda} P'$ (72)

$\lambda = x : s \to rl \ \ (x \neq y)$ (73)

(Inversion on (Red-New2) and (71))

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

$\Gamma, y : B_1; \Delta \vdash P$ (74)

$matched(B_1)$ (75)

(Inversion on (T-NEW) and (70))

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

$\Gamma, y : B_1 = \Gamma', y : B_1, x : B$ (76)

$B \xrightarrow{s \to rl} B'$ (77)

$\Gamma', y : B_1, x : B'; \Delta \vdash P'$ (78)

(Induction hypothesis on (74) and (72))

$\Gamma', x : B'; \Delta \vdash (\mathbf{new}\ y)P'$ (79)

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

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

$\Delta = \Delta', x : T$ (80)

$\Gamma, y : B_1; \Delta \vdash P'$ (81)

(Induction hypothesis on (74) and (72))

$\Gamma; \Delta \vdash (\mathbf{new}\ y)P'$ (82)

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

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

$\Gamma; \Delta, y : l(B_1) \vdash P$ (83)

(Inversion on (T-SNEW) and (70))

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

$\Gamma = \Gamma', x : B$ (84)

$B \xrightarrow{s \to rl} B'$ (85)

$\Gamma', x : B'; \Delta, y : l(B_1) \vdash P'$ (86)

(Induction hypothesis on (83) and (72))

$\Gamma', x : B'; \Delta \vdash (\mathbf{new}\ y)P'$ (87)

((86) and (T-SNEW))

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

$\Delta, y : l(B_1) = \Delta', y : l(B_1), x : T$ (88)

$\Gamma; \Delta, y : l(B_1) \vdash P'$ (89)

(Induction hypothesis on (83) and (72))

$\Gamma; \Delta \vdash (\mathbf{new}\ y)P'$ (90)

((89) and (T-SNEW))

(*Case* (Red-Struct))

$\Gamma; \Delta \vdash P_1$ (91)

$P_1 \xrightarrow{\lambda} P_2$ (92)

(Assumption)

$P_1 \equiv P_1'$ (93)

$P_1' \xrightarrow{\lambda} P_2'$ (94)

$P_2 \equiv P_2'$ (95)

(Inversion on (Red-Struct) and (92))

$\Gamma; \Delta \vdash P_1'$ (96)

((91) and Lemma 3.3)

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

$$\Gamma; \Delta \vdash P_2' \tag{97}$$

(Induction hypothesis on (96) and (94))

$$\Gamma; \Delta \vdash P_2 \tag{98}$$

((97) and Lemma 3.3)

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

$$\Gamma = \Gamma', x : B \tag{99}$$

$$B \xrightarrow{s \to rl} B' \tag{100}$$

$$\Gamma', x : B'; \Delta \vdash P_2' \tag{101}$$

(Induction hypothesis on (96) and (94))

$$\Gamma', x : B'; \Delta \vdash P_2 \tag{102}$$

((101) and Lemma 3.3)

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

$$\Delta = \Delta', x : T \tag{103}$$

$$\Gamma; \Delta \vdash P_2' \tag{104}$$

(Induction hypothesis on (96) and (94))

$$\Gamma; \Delta \vdash P_2 \tag{105}$$

((104) and Lemma 3.3)