Conversation Types

Luís Caires and Hugo Torres Vieira

CITI, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal

---

Abstract

We present a type theory for analyzing concurrent multiparty interactions as found in service-oriented computing. Our theory introduces a novel and flexible type structure, able to uniformly describe both the internal and the interface behavior of systems, referred respectively as choreographies and contracts in web-services terminology. The notion of conversation builds on the fundamental concept of session, but generalizes it along directions up to now unexplored; in particular, conversation types discipline interactions in conversations while accounting for dynamical join and leave of an unanticipated number of participants. We prove that well-typed systems never violate the prescribed conversation constraints. We also present techniques to ensure progress of systems involving several interleaved conversations, a previously open problem.

Key words: Behavioral Types, Distributed Systems, Program Analysis, Service-Based Systems, Session Types

---

1. Introduction

While most issues arising in the context of communication-based software systems do not appear to be new when considered in isolation, the analysis of loosely-coupled distributed systems involving type based discovery, and multiparty collaborations such as those supported by web-services technology raises many challenges and calls for new concepts, specially crafted models, and formal analysis techniques (e.g., [1, 2, 4, 5, 6, 10, 12, 16, 19, 22]). In previous work [28] we introduced the Conversation Calculus (CC), a π-calculus based model for service-oriented computing that builds on the concepts of process delegation, loose-coupling, and, crucially, conversation contexts.

A key concept for the organization of service-oriented computing systems is the notion of conversation. A conversation is a structured, not centrally coordinated, possibly concurrent, set of interactions between several participants. Then, a conversation context is a medium where partners may interact in a conversation. It can be distributed in many pieces, and processes in any piece may seamlessly talk to processes in the same or any other piece of the same conversation context. Intuitively a conversation context may be seen as a virtual chat room where remote participants exchange messages according to some discipline, while simultaneously engaged in other conversations. Conversation context identities can be passed around, allowing participants to dynamically join conversations. To join an ongoing

conversation, a process may perform a remote conversation access using the conversation context identifier. It is then able to participate in the conversation to which it has joined, while being able to interact back with the caller context through the access point. To discipline multiparty conversations we introduce conversation types, a novel and flexible type structure, able to uniformly describe both the internal and the interface behavior of systems, referred respectively as choreographies and contracts in web-services terminology.

We give substantial evidence that our minimal extension to the π-calculus is already effective enough to model and type sophisticated service-based systems, at a fairly high level of abstraction. Examples of such systems include challenging scenarios involving simultaneous multiparty conversations, with concurrency and access to local resources, and conversations with a dynamically changing and unanticipated number of participants, that fall out of scope of other approaches for modeling and typing of service-based systems.

On the opposite direction, we show that the key ideas behind conversation types can already be developed in much more canonic models (without explicit conversation contexts, and thus with some loss of expressiveness) such as a simple labeled π-calculus, thus demonstrating the generality and essence of our approach to typing multi-party interactions.

1.1. Conversation Contexts and Conversation Types

We explain the key ideas of our development by going through a motivating example. Consider the following composition of two conversation contexts, named *Buyer* and *Seller*, modeling a typical service collaboration:

$$\begin{array}{l} \mathit{Buyer} \blacktriangleleft [\ \textbf{new}\ \mathit{Seller} \cdot \mathtt{startBuy} \Leftarrow \mathtt{buy}!(\mathit{prod}).\mathtt{price}?(v)\ ] \\ \mid \\ \mathit{Seller} \blacktriangleleft [\ \mathit{PriceDB} \mid \\ \qquad \textbf{def}\ \mathtt{startBuy} \Rightarrow \mathtt{buy}?(\mathit{prod}).\mathtt{askPrice}^{\uparrow}!(\mathit{prod}). \\ \qquad\qquad \mathtt{readVal}^{\uparrow}?(v).\mathtt{price}!(v)\ ] \end{array}$$

Notice that in the core CC, the bounded communication medium provided by a conversation context may also be used to model a partner local context, avoiding the introduction of a primitive notion of site. The code in *Buyer* starts a new conversation by calling service `startBuy` located at *Seller* using the service instantiation idiom $\textbf{new}\ \mathit{Seller} \cdot \mathtt{startBuy} \Leftarrow \mathtt{buy}!(\mathit{prod}).\mathtt{price}?(v)$. The code $\mathtt{buy}!(\mathit{prod}).\mathtt{price}?(v)$ describes the role of *Buyer* in the conversation: a `buy` message is sent, and afterwards a `price` message should be received. Upon service instantiation, the system evolves to:

$$\begin{array}{l} (\boldsymbol{\nu} c)(\ \mathit{Buyer} \blacktriangleleft [\ c \blacktriangleleft [\ \mathtt{buy}!(\mathit{prod}).\mathtt{price}?(v)\ ]\ ] \\ \qquad \mid \\ \qquad \mathit{Seller} \blacktriangleleft [\ \mathit{PriceDB} \mid \\ \qquad\qquad c \blacktriangleleft [\ \mathtt{buy}?(\mathit{prod}).\mathtt{askPrice}^{\uparrow}!(\mathit{prod}). \\ \qquad\qquad\qquad \mathtt{readVal}^{\uparrow}?(v).\mathtt{price}!(v)\ ]\ \ ) \end{array}$$

where c is the fresh name of the newly created conversation (with two pieces). The code:

$$\mathtt{buy}?(\mathit{prod}).\mathtt{askPrice}^{\uparrow}!(\mathit{prod}).\mathtt{readVal}^{\uparrow}?(v).\mathtt{price}!(v)$$

describes the participation of *Seller* in the conversation c: a `buy` message is received, and in the end, `price` message should be sent. In between, database *PriceDB* located in the *Seller* context is consulted through a pair of $\uparrow$ directed message exchanges (`askPrice` and `readVal`). Such messages are targeted to the parent conversation (*Seller*), rather than to the current conversation (c).

In our theory, message exchanges *inside* and *at* the interface of subsystems are captured by conversation types, which describe both internal and external participation of processes in conversations. The *Buyer* and *Seller* conversation is described by type:

$$BSChat \triangleq \tau\,\mathtt{buy}(Tp).\tau\,\mathtt{price}(Tm)$$

specifying the two interactions that occur sequentially within the conversation c, first a message `buy` and after a message `price` (Tp and Tm represent basic value types).

The τ in, e.g., $\tau\,\mathtt{buy}(Tp)$ means that the interaction is internal. A declaration such as $\tau\,\mathtt{buy}(Tp)$ is like an assertion such as $\mathtt{buy}(Tp) : Buyer \rightarrow Seller$ in a message sequence chart, or in the global types of [19], except that in our case participant identities are abstracted away, increasing flexibility. In general, the interactions described by a type such as *BSChat* may be realized in several ways, by different participants. Technically, we specify the several possibilities by a (ternary) merge relation between types, noted $B = B_1 \bowtie B_2$, stating how a behavior B may be projected in two independent matching behaviors B_1 and B_2. In particular, we have (among others) the projection:

$$BSChat = \,!\,\mathtt{buy}(Tp).?\,\mathtt{price}(Tm) \bowtie\, ?\,\mathtt{buy}(Tp).!\,\mathtt{price}(Tm)$$

The type $!\,\mathtt{buy}(Tp).?\,\mathtt{price}(Tm)$ will be used to type the *Buyer* participation, and the type $?\,\mathtt{buy}(Tp).!\,\mathtt{price}(Tm)$ will be used to type the *Seller* participation (in conversation *BSChat*). Thus, in our first example, the conversation type *BSChat* is decomposed in a pair of "dual" conversation types, as in classical session types [17, 18]; this does not need to be always the case, however. In fact, the notion of conversation builds on the fundamental concept of session but extends it along unexplored directions, as we now discuss. Consider a three-party variation (from [10]) of the example above:

$$\begin{array}{l}
Buyer \blacktriangleleft [\ \ \textbf{new}\ Seller \cdot \mathtt{startBuy} \Leftarrow \mathtt{buy}!(prod).\mathtt{price}?(p).\mathtt{details}?(d)\ ] \\
\mid \\
Seller \blacktriangleleft [\ \ PriceDB \mid \\
\qquad\qquad \textbf{def}\ \mathtt{startBuy} \Rightarrow \mathtt{buy}?(prod).\mathtt{askPrice}^{\uparrow}!(prod). \\
\qquad\qquad\qquad\qquad\qquad\ \mathtt{readVal}^{\uparrow}?(p).\mathtt{price}!(p). \\
\qquad\qquad\qquad\qquad\qquad\ \textbf{join}\ Shipper \cdot \mathtt{newDelivery} \Leftarrow \mathtt{product}!(prod)\ ] \\
\mid \\
Shipper \blacktriangleleft [\ \ \textbf{def}\ \mathtt{newDelivery} \Rightarrow \mathtt{product}?(p).\mathtt{details}!(data)\ ]
\end{array}$$

The role of *Shipper* is to inform the client on the delivery details. The code is composed of three conversation contexts, representing the three partners *Buyer*, *Seller* and *Shipper*. The system progresses as in the first example: messages `buy` and `price` are exchanged between

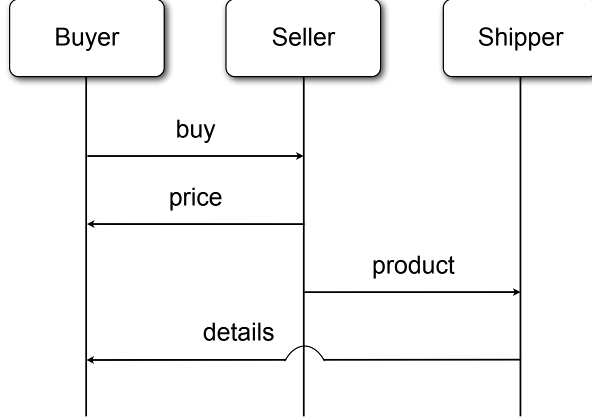

Figure 1: BSSChat Message Sequence Chart.

Buyer and *Seller* in the fresh conversation. After that, *Shipper* is asked by *Seller*, using idiom **join** $\mathit{Shipper} \cdot \mathtt{newDelivery} \Leftarrow \cdots$, to join the ongoing conversation (till then involving only *Buyer* and *Seller*). The system then evolves to:

$$
\begin{array}{ll}
(\boldsymbol{\nu} a)(& \mathit{Buyer} \blacktriangleleft [\ a \blacktriangleleft [\ \mathtt{details}?(d)\ ]\ ] \\
& | \\
& \mathit{Seller} \blacktriangleleft [\ a \blacktriangleleft [\ \mathtt{product}!(\mathit{prod})\ ]\ \ |\ \ \ldots\ ]\ | \\
& | \\
& \mathit{Shipper} \blacktriangleleft [\ a \blacktriangleleft [\ \mathtt{product}?(p).\mathtt{details}!(\mathit{data})\ ]\ ]\ \ )
\end{array}
$$

Notice that *Seller does not lose access* to the conversation after asking service $\mathit{Shipper} \cdot \mathtt{newDelivery}$ to join in the current conversation a (partial session delegation). In fact, *Seller* and *Shipper* will interact later on in the very same conversation, by exchanging a `product` message. Finally, *Shipper* sends a message `details` directly to *Buyer*. In this case, the global conversation a is initially assigned type:

$$\mathit{BSSChat} \triangleq \tau\,\mathtt{buy}(\mathit{Tp}).\tau\,\mathtt{price}(\mathit{Tm}).\tau\,\mathtt{product}(\mathit{Tp}).\tau\,\mathtt{details}(\mathit{Td})$$

BSSChat type may be depicted as the message sequence chart shown in Figure 1. We decompose type *BSSChat* in three "projections" (B_{bu}, B_{se}, and B_{sh}), by means of the *merge* $\bowtie$, first by $\mathit{BSSChat} = B_{bu} \bowtie B_{ss}$, and then by $B_{ss} = B_{se} \bowtie B_{sh}$, where:

$$
\begin{array}{lcl}
B_{bu} & \triangleq & !\,\mathtt{buy}(\mathit{Tp}).?\,\mathtt{price}(\mathit{Tm}).?\,\mathtt{details}(\mathit{Td}) \\
B_{ss} & \triangleq & ?\,\mathtt{buy}(\mathit{Tp}).!\,\mathtt{price}(\mathit{Tm}).\tau\,\mathtt{product}(\mathit{Tp}).!\,\mathtt{details}(\mathit{Td}) \\
B_{se} & \triangleq & ?\,\mathtt{buy}(\mathit{Tp}).!\,\mathtt{price}(\mathit{Tm}).!\,\mathtt{product}(\mathit{Tp}) \\
B_{sh} & \triangleq & ?\,\mathtt{product}(\mathit{Tp}).!\,\mathtt{details}(\mathit{Td})
\end{array}
$$

These various "local" types are merged by our type system in a compositional way, allowing e.g., service `startBuy` to be assigned type $?\,\mathtt{startBuy}([B_{ss}])$, and the contribution of each

partner in the conversation to be properly determined. At the point where **join** operation above gets typed, the (residual) conversation type corresponding to the participation of *Seller* is typed $\tau\,\mathtt{product}(Tp).!\,\mathtt{details}(Td)$. At this stage, extrusion of the conversation name a to service *Seller* $\cdot$ `newDelivery` will occur, to enable *Shipper* to join in. Notice that the global conversation *BSSChat* discipline will nevertheless be respected, since the conversation fragment delegated to *Shipper* is typed $?\,\mathtt{product}(Tp).!\,\mathtt{details}(Td)$ while the conversation fragment retained by *Seller* is typed $!\,\mathtt{product}(Tp)$. Also notice that since conversation types abstract away from participant identities, the overall conversation type can be projected into the types of the individual roles in several ways, allowing for different implementations of the roles of a given conversation (cf. loose-coupling). It is even possible to type systems with an unbounded number of different participants, as needed to type, e.g., a service broker.

Our type system combines techniques from linear, behavioral, session and spatial types (see [7, 18, 20, 21]): the type structure features prefix $M.B$, parallel composition $B_1 \mid B_2$, and other operators. Messages M describe external (receive ? / send !) exchanges in two views: with the *caller / parent* conversation ($\uparrow$), and in the *current* conversation ($\downarrow$). They also describe internal message exchanges (τ). Key technical ingredients in our approach to conversation types are the amalgamation of global types and of local types (in the general sense of [19]) in the same type language, and the definition of a merge relation ensuring, by construction, that participants typed by the projected views of a type will behave well under composition. Merge subsumes duality, in the sense that for each τ-free B there are types $\overline{B}, B'$ such that $B \bowtie \overline{B} = \tau(B')$ (where $\tau(B')$ is defined exclusively on τ message types), so sessions are special cases of conversations. But merge of types allows for extra flexibility on the manipulation of projections of conversation types, in an open-ended way, as illustrated above. In particular, our approach allows fragments of a conversation type (e.g., a choreography) to be dynamically distributed among participants, while statically ensuring that interactions follow the prescribed discipline.

The technical contributions of this work may be summarized as follows. First, we define the new notion of conversation type. Conversation types are a generalization of session types to loosely-coupled, possibly concurrent, multiparty conversations, allowing mixed global / local behavioral descriptions to be expressed at the same level, while supporting the analysis of systems with dynamic delegation of fragments of ongoing conversations. Second, we advance new techniques to certify safety and liveness properties of service-based systems. We propose a type system for assigning conversation types to core CC systems. Processes that get past our typing rules are ensured to be free of communication errors, and races on plain messages (Corollary 3.24): this also implies that well-typed systems enjoy a conversation fidelity property (i.e., all conversations follow the prescribed protocols). Finally, we present techniques to establish progress of systems with several interleaved conversations (Theorem 4.7), exploiting the combination of conversation names with message labels in event orderings, and, more crucially, propagation of orderings in communications, solving a previously open problem. Before concluding and discussing related work, we demonstrate that our concepts and techniques—both the conversation type system and the progress analysis—are not specific to the core CC model, by showing they can be smoothly adapted

and applied to systems specified in a simple labeled π-calculus.

This paper is an extended revised version of [8]. The main difference with respect to the original presentation in [8] is the inclusion of Section 5, where we demonstrate the applicability of our techniques to a simple labeled π-calculus.

2. The Core Conversation Calculus

In this section, we present the syntax of our calculus, and formally define its operational semantics, by means of a labeled transition system. The core Conversation Calculus (core CC) extends the static fragment of the π-calculus [24] with the conversation construct $n \blacktriangleleft [P]$, and replaces channel based communication with context-sensitive message based communication. For simplicity, we present a monadic version of the calculus.

Definition 2.1 (Core CC Syntax). *The syntax of core CC processes ($P, Q, \ldots$), message directions ($d, d', \ldots$), and actions ($\alpha, \alpha_1, \ldots$) is given in Figure 2.*

We assume given an infinite set of names Λ, an infinite set of variables $\mathcal{V}$, an infinite set of labels $\mathcal{L}$, and an infinite set of process variables χ. The static fragment is defined by the inaction $\mathbf{0}$, parallel composition $P \mid Q$, name restriction $(\boldsymbol{\nu} a)P$ and recursion $\mathbf{rec}\,\mathcal{X}.P$. The conversation access construct $n \blacktriangleleft [P]$, allows a process to initiate interactions, as specified by P, in the conversation n.

Communication is expressed by the guarded choice construct $\Sigma_{i \in I}\ \alpha_i.P_i$, meaning that the process may select some initial action α_i and then progress as P_i. Communication actions are of two forms: $l^d!(n)$ for sending messages (e.g., $\texttt{askPrice}^{\uparrow}!(prod)$) and $l^d?(x)$ for receiving messages (e.g., $\texttt{price}^{\downarrow}?(p)$). Thus, message communication is defined by the label l and the direction d. There are two message directions: $\downarrow$ (read "here") meaning that the interaction should take place in the current conversation or $\uparrow$ (read "up") meaning that the interaction should take place in the caller conversation. N.B.: to lighten notation we omit the $\downarrow$ in messages, without any ambiguity. A basic action may also be of the form $\mathbf{this}(x)$, allowing the process to dynamically access the identity of the current conversation.

Notice that message labels (from $l \in \mathcal{L}$) are not names but free identifiers (cf. record labels or XML tags), and therefore not subject to fresh generation, restriction or binding. Only conversation names may be subject to binding, and freshly generated via $(\nu a)P$.

The distinguished occurrences of a, x, x and $\mathcal{X}$ are binding occurrences in $(\boldsymbol{\nu} a)P$, $l^d?(x).P$, $\mathbf{this}(x).P$, and $\mathbf{rec}\,\mathcal{X}.P$, respectively. The sets of free ($fn(P)$) and bound ($bn(P)$) names, free variables ($fv(P)$), and free process variables ($fpv(P)$) in a process P are defined as usual. We implicitly identify α-equivalent processes. We denote by $P\{x \leftarrow a\}$ the process obtained by replacing all free occurrences of x by a (likewise for $P\{\mathcal{X} \leftarrow Q\}$).

2.1. Operational Semantics

The operational semantics of the core CC is defined by a labeled transition system. For clarity, we split the presentation into two sets of rules, one (in Figure 3) containing the rules

$$
\begin{array}{lcll}
a, b, c, \ldots & \in & \Lambda & \text{(Names)} \\
x, y, z, \ldots & \in & \mathcal{V} & \text{(Variables)} \\
n, m, o \ldots & \in & \Lambda \cup \mathcal{V} & \\
l, s \ldots & \in & \mathcal{L} & \text{(Labels)} \\
\mathcal{X}, \mathcal{Y}, \ldots & \in & \chi & \text{(Process Vars)}
\end{array}
$$

$$
\begin{array}{rcll}
P, Q & ::= & \mathbf{0} & \text{(Inaction)} \\
& \mid & P \mid Q & \text{(Parallel Composition)} \\
& \mid & (\boldsymbol{\nu} a)P & \text{(Name Restriction)} \\
& \mid & \mathsf{rec}\, \mathcal{X}.P & \text{(Recursion)} \\
& \mid & \mathcal{X} & \text{(Variable)} \\
& \mid & n \blacktriangleleft [P] & \text{(Conversation Access)} \\
& \mid & \Sigma_{i \in I}\ \alpha_i.P_i & \text{(Prefix Guarded Choice)}
\end{array}
$$

$$
\begin{array}{rcll}
d & ::= & \downarrow \mid \uparrow & \text{(Directions)} \\
\alpha & ::= & l^d!(n) & \text{(Output)} \\
& \mid & l^d?(x) & \text{(Input)} \\
& \mid & \mathsf{this}(x) & \text{(Conversation Awareness)}
\end{array}
$$

Figure 2: The Core Conversation Calculus Syntax.

for the basic operators, which are essentially identical to the corresponding ones in the π-calculus (see [27]), and the other (in Figure 4) grouping the rules specific to the Conversation Calculus.

A transition $P \xrightarrow{\lambda} Q$ states that process P may evolve to process Q by performing the action represented by the transition label λ. We define transition labels and actions.

Definition 2.2 (Transition Labels and Actions). *Transition labels and actions are defined as follows:*

$$
\begin{array}{rcll}
\sigma & ::= & \tau \mid l^d!(a) \mid l^d?(a) \mid \mathtt{this} & \text{(Actions)} \\
\lambda & ::= & c\ \sigma \mid \sigma \mid (\nu a)\lambda & \text{(Transition Labels)}
\end{array}
$$

An action τ denotes an internal communication, actions $l^d!(a)$ and $l^d?(a)$ represent communications with the environment, and `this` represents a conversation identity access. To capture the observational semantics of processes, transition labels need to register not only the action but also the conversation where the action takes place. So, a transition label λ containing $c\ \sigma$ is said to be *located at* conversation c (or just *located*), otherwise is said to be *unlocated*. In $(\nu a)\lambda$ the distinguished occurrence of a is bound with scope λ (cf., the π-calculus bound output actions). For a communication label λ we denote by $\overline{\lambda}$ the dual matching label obtaining by swapping inputs with outputs, such that, e.g., $\overline{l^d!(a)} = l^d?(a)$ and $\overline{l^d?(a)} = l^d!(a)$. We denote by $fn(\lambda)$ and $bn(\lambda)$ (respectively) the free and bound names of a transition label, and by $na(\lambda)$ both free and bound names of a transition label.

The `this` transition label represents a conversation identity access. Processes can explicitly access the identity of the conversation in which they are located (which is captured by a `this` label), and synchronizations between processes may also require such contextual information. Since messages do not explicitly refer the conversation to which they pertain, the operational semantics of the core CC must locally account for synchronizations which may arise depending on the surrounding context. For example, consider the following process:

$$l^{\downarrow}!().P \mid n \blacktriangleleft \left[l^{\downarrow}?().Q\right] \tag{1}$$

which specifies that a message l is to be sent at the current conversation, after which P is activated, and that a message l is to be received at conversation n, after which Q is activated. If such a process is to be placed in a piece of conversation n, yielding the following process:

$$n \blacktriangleleft \left[l^{\downarrow}!().P \mid n \blacktriangleleft \left[l^{\downarrow}?().Q\right]\right]$$

then both input and output refer to the same conversation n and therefore the message may be exchanged under conversation n. Thus, we can observe the following τ transition:

$$n \blacktriangleleft \left[l^{\downarrow}!().P \mid n \blacktriangleleft \left[l^{\downarrow}?().Q\right]\right] \xrightarrow{\tau} n \blacktriangleleft [P \mid n \blacktriangleleft [Q]]$$

When locally describing the behavior of the process shown in (1) we must account for the possible synchronization *if* the current conversation *is* the n conversation. This is realized by means of a n `this` transition, which may only progress under a piece of conversation n. We then have the following transition:

$$l^{\downarrow}!().P \mid n \blacktriangleleft \left[l^{\downarrow}?().Q\right] \xrightarrow{n\,\mathtt{this}} P \mid n \blacktriangleleft [Q]$$

We may now define the transition relation.

Definition 2.3 (Transition Relation). *The transition relation ($P \xrightarrow{\lambda} Q$) is the least relation that satisfies the rules of Figure 3 and of Figure 4.*

Remark 2.4. *Given processes P and Q, if $P \xrightarrow{\lambda} Q$ then either $\lambda = \tau$ or $\lambda = l^d!(a)$ or $\lambda = (\nu a)l^d!(a)$ or $\lambda = l^d?(a)$ or $\lambda = c\,\mathtt{this}$ or $\lambda = c\,l^{\downarrow}!(a)$ or $\lambda = (\nu a)c\,l^{\downarrow}!(a)$ or $\lambda = c\,l^{\downarrow}?(a)$.*

Transition rules presented in Figure 3 closely follow the ones for the π-calculus and should be fairly clear to a reader familiar with mobile process calculi. For example, rule (*Open*) corresponds to the bound output or extrusion rule, in which a bound name a is extruded to the environment in an output message λ: we define $out(\lambda) = a$ if $\lambda = l^d!(a)$ or $\lambda = c\ l^d!(a)$ and $c \neq a$. We omit the rules symmetric to (*Par-l*) and (*Close-l*). In rule (*Par-l*) we use predicate $\#$ to denote disjoint sets: $A \# B$ iff $A \cap B = \emptyset$.

It would be useful however to discuss the intuitions behind the rules for conversation contexts (Figure 4). In rule (*Here*) an $\uparrow$ directed message (to the enclosing conversation) becomes $\downarrow$ (in the current conversation), after passing through the conversation access boundary . We note by λ^d a transition label containing the direction d ($\uparrow, \downarrow$), and by $\lambda^{d'}$ the label obtained by replacing d by d' in λ^d (e.g., if $\lambda^{\uparrow}$ is $\mathtt{askPrice}^{\uparrow}?(a)$ then $\lambda^{\downarrow}$ is $\mathtt{askPrice}^{\downarrow}?(a)$).

$$l^d!(a).P \xrightarrow{l^d!(a)} P\ (Out) \qquad l^d?(x).P \xrightarrow{l^d?(a)} P\{x \leftarrow a\}\ (In) \qquad \frac{\alpha_j.P_j \xrightarrow{\lambda} Q \quad j \in I}{\Sigma_{i\in I}\ \alpha_i.P_i \xrightarrow{\lambda} Q}(Sum)$$

$$\frac{P \xrightarrow{\lambda} Q \quad a \in out(\lambda)}{(\boldsymbol{\nu} a)P \xrightarrow{(\nu a)\lambda} Q}(Open) \qquad \frac{P \xrightarrow{\lambda} Q \quad a \notin na(\lambda)}{(\boldsymbol{\nu} a)P \xrightarrow{\lambda} (\boldsymbol{\nu} a)Q}(Res)$$

$$\frac{P \xrightarrow{\lambda} Q \quad bn(\lambda)\,\#\,fn(R)}{P \mid R \xrightarrow{\lambda} Q \mid R}(Par\text{-}l) \qquad \frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\overline{\lambda}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}(Comm)$$

$$\frac{P \xrightarrow{(\nu a)\overline{\lambda}} P' \quad Q \xrightarrow{\lambda} Q' \quad a \notin fn(Q)}{P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu} a)(P' \mid Q')}(Close\text{-}l) \qquad \frac{P\{\mathcal{X} \leftarrow \mathbf{rec}\,\mathcal{X}.P\} \xrightarrow{\lambda} Q}{\mathbf{rec}\,\mathcal{X}.P \xrightarrow{\lambda} Q}(Rec)$$

Figure 3: Basic Operators (π-calculus).

In rule (*Loc*) an unlocated $\downarrow$ message (in the current conversation) gets explicitly located at the conversation c in which it originates. Given an unlocated label λ, we represent by $c \cdot \lambda$ the label obtained by locating λ at c (e.g., if $\lambda^{\downarrow}$ is $\texttt{askPrice}^{\downarrow}?(p)$ then $c \cdot \lambda^{\downarrow}$ is $c\ \texttt{askPrice}^{\downarrow}?(p)$). In rule (*Through*) an already located communication label transparently crosses some other conversation boundary (by $a\ \lambda^{\downarrow}$ we denote a transition located at a), and likewise for a τ label in rule (*Tau*). In rule (*This*) a `this` label reads the current conversation identity, and originates a c `this` label. A c `this` labeled transition may only progress inside the c conversation, as expressed by the rule (*ThisLoc*), where a `this` label matches the enclosing conversation. In rules (*ThisComm-r*) and (*ThisClose-r*) an unlocated communication matches a communication located at c, originating a c `this` label, thus ensuring the interaction occurs in the given conversation c, as required. We omit the rules symmetric to (*ThisComm-r*) and (*ThisClose-r*).

The reduction relation is defined on top of the labeled transition system.

Definition 2.5 (Reduction). *The relation of reduction on processes, noted $P \to Q$, is defined as $P \xrightarrow{\tau} Q$.*

In the next section we describe how the basic set of primitives of the core CC can already be used to model useful service-oriented primitives.

2.2. Representing Service-Oriented Primitives

Our core model focuses on the fundamental notions of conversation context and message-based communication. From these basic mechanisms, useful programming abstractions for service-oriented systems may be idiomatically defined, namely service definition and instantiation constructs (defined as primitives in [28]), and the conversation join construct (introduced in [8]), which is crucial to our approach to multiparty conversations. These

$$\frac{P \xrightarrow{\lambda^{\uparrow}} Q}{c \blacktriangleleft [P] \xrightarrow{\lambda^{\downarrow}} c \blacktriangleleft [Q]} (Here) \qquad \frac{P \xrightarrow{\lambda^{\downarrow}} Q}{c \blacktriangleleft [P] \xrightarrow{c \cdot \lambda^{\downarrow}} c \blacktriangleleft [Q]} (Loc)$$

$$\frac{P \xrightarrow{a\ \lambda^{\downarrow}} Q}{c \blacktriangleleft [P] \xrightarrow{a\ \lambda^{\downarrow}} c \blacktriangleleft [Q]} (Through) \qquad \frac{P \xrightarrow{\tau} Q}{c \blacktriangleleft [P] \xrightarrow{\tau} c \blacktriangleleft [Q]} (Tau)$$

$$\textbf{this}(x).P \xrightarrow{c\ \texttt{this}} P\{x \leftarrow c\}\ (This) \qquad \frac{P \xrightarrow{c\ \texttt{this}} Q}{c \blacktriangleleft [P] \xrightarrow{\tau} c \blacktriangleleft [Q]} (ThisLoc)$$

$$\frac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{c\ \overline{\sigma}} Q'}{P \mid Q \xrightarrow{c\ \texttt{this}} P' \mid Q'} (ThisComm\text{-}r) \qquad \frac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{(\nu a)c\ \overline{\sigma}} Q'}{P \mid Q \xrightarrow{c\ \texttt{this}} (\boldsymbol{\nu} a)(P' \mid Q')} (ThisClose\text{-}r)$$

Figure 4: Conversation Operators.

$$\begin{array}{lll}
\textbf{def}\ s \Rightarrow P & \triangleq & s?(x).x \blacktriangleleft [P] \\
\textbf{new}\ n \cdot s \Leftarrow Q & \triangleq & (\boldsymbol{\nu} c)(n \blacktriangleleft [s!(c)] \mid c \blacktriangleleft [Q]) \\
\textbf{join}\ n \cdot s \Leftarrow Q & \triangleq & \textbf{this}(x).(n \blacktriangleleft [s!(x)] \mid Q) \\
\star\textbf{def}\ s \Rightarrow P & \triangleq & \textbf{rec}\ \mathcal{X}.s?(x).(\mathcal{X} \mid x \blacktriangleleft [P])
\end{array}$$

Figure 5: Service Idioms.

constructs may be embedded in a simple way in the minimal calculus, without hindering the flexibility of modeling and analysis.

We show in Figure 5 the derived forms along with their translation in the core CC. A service definition has the form **def** $s \Rightarrow P$ where s is the service name, and P is the process to be launched at the server side endpoint of the freshly created conversation (the service protocol). Service definitions must be placed in appropriate contexts (cf. methods in objects), e.g.:

$$Shipper \blacktriangleleft [\textbf{def}\ newDelivery \Rightarrow P \mid \cdots]$$

A new instance of a service s is created by **new** $n \cdot s \Leftarrow Q$, where n indicates the context where the service named s is published, and Q specifies the client protocol. For instance, a service definition as shown above may be instantiated by:

$$\textbf{new}\ Shipper \cdot newDelivery \Leftarrow Q$$

The process Q describes the client protocol that will run inside the freshly created conversation. The interaction between service instantiation (**new**) and service definition (**def**) results in the creation of a new conversation context n, in which the service interactions will take place. Such context is initially split in two pieces, one piece $c \blacktriangleleft [Q]$ residing in the context

of the client, the other piece $c \blacktriangleleft [P]$ placed in the context of the server. These newly created conversation access points appear to their caller contexts as any other local processes, as P and Q are able to continuously interact by means of $\uparrow$ directed messages. As expected, P and Q will interact in the new conversation by means of $\downarrow$ directed messages. Thus, conversation initiation via **new** and **def** is similar to session initiation in session calculi [18]. Typically, service definitions may also be replicated, written $\star$**def** $s \Rightarrow P$, in order to be usable an unbounded number of times.

In the core CC, conversation identifiers may be manipulated by processes if needed (via the **this**$(x).P$), passed around in messages and subject to scope extrusion: this corresponds, in our setting, to a generalization of session delegation, in which multiparty conversations are modeled by the progressive access of multiple, dynamically determined partners, to an ongoing conversation. Joining of another partner to an ongoing conversation is a frequent programming idiom, that may be conveniently abstracted by the **join** $n \cdot s \Leftarrow Q$ construct. The semantics of the **join** expression is similar to the service instantiation construct **new**: the key difference is that while **new** creates a fresh *new conversation*, **join** allows a service s defined at n to join in the *current conversation*, and continue interacting as specified by Q. Next, we illustrate typical reduction steps of systems involving the service oriented idioms.

$$n \blacktriangleleft [\mathbf{def}\ s_1 \Rightarrow P] \mid \cdots \mid m \blacktriangleleft [\mathbf{new}\ n \cdot s_1 \Rightarrow Q] \quad \rightarrow$$
$$(\boldsymbol{\nu} c)(n \blacktriangleleft [c \blacktriangleleft [P]] \mid \cdots \mid m \blacktriangleleft [c \blacktriangleleft [Q]])$$

Here, the service instantiation results in the creation of a new conversation c. The two partners n and m may then interact in the new conversation c by $\downarrow$ messages, exchanged by the processes P and Q. These processes while performing the conversation may also interact with their parent conversations n and m via $\uparrow$ messages.

$$o \blacktriangleleft [\mathbf{def}\ s_2 \Rightarrow P] \mid \cdots \mid c \blacktriangleleft [\mathbf{join}\ o \cdot s_2 \Rightarrow Q] \quad \rightarrow$$
$$o \blacktriangleleft [c \blacktriangleleft [P]] \mid \cdots \mid c \blacktriangleleft [Q]$$

This reduction step illustrates the situation where an ongoing conversation c asks a new partner o to join in c according to a published service definition s_2. It should be clear how building on these simple mechanisms, multiparty conversations may be progressively and dynamically formed, starting from dyadic ones created by service instantiation.

In the next section we will develop a fairly rich type theory for conversation contexts, using the core CC as the intended model. Our type system may be used to discipline and specify communication patterns in systems with complex interactive behavior including systems with dynamically assembled multiparty conversations, ensuring absence of certain kinds of erroneous behaviors as already mentioned in the Introduction.

3. Type System

In this section we formally present our type system for the core Conversation Calculus. As already motivated in the Introduction, our types specify the message protocols that flow between and within conversations.

$$
\begin{array}{llll}
B & ::= & B_1 \mid B_2 \;\big|\; \mathbf{0} \;\big|\; \mathtt{rec}\,\mathcal{X}.B \;\big|\; \mathcal{X} & \\
 & & \big|\; \oplus_{i\in I}\{M_i.B_i\} \;\big|\; \&_{i\in I}\{M_i.B_i\} & \text{(Behavioral)}\\
M & ::= & p\, l^d(C) & \text{(Message)}\\
p & ::= & \mathtt{!} \;\big|\; \mathtt{?} \;\big|\; \tau & \text{(Polarity)}\\
C & ::= & [B] & \text{(Conversation)}\\
L & ::= & n : C \;\big|\; L_1 \mid L_2 \;\big|\; \mathbf{0} & \text{(Located)}\\
T & ::= & L \mid B & \text{(Process)}
\end{array}
$$

Figure 6: Syntax of Types.

Definition 3.1 (Conversation Types). *The syntax of the conversation type language is given in Figure 6.*

Typing judgments have the form $P :: T$, where T is a process type. Intuitively, a type judgement $P :: T$ states that if process P is placed in an environment where a process of type T is expected, then the resulting system is safe, in a sense to be made precise below (Corollary 3.24). In general, a process type T has the form $L \mid B$, where L is a *located type* and B is a *behavioral type* which specifies the behavior of P in the current conversation (taking place in the context where P resides). An atomic located type associates a conversation type C to a conversation name n. Conversation types C are given by $[B]$, where B specifies the message interactions that may take place in the conversation.

Behavioral types B include the branch and the choice constructs ($\&_{i\in I}\{M_i.B_i\}$ and $\oplus_{i\in I}\{M_i.B_i\}$, respectively), specifying processes that can branch in either of the $M_i.B_i$ behaviors and choose between one of the $M_i.B_i$ behaviors, respectively. Prefix $M.B$ specifies a process that sends, receives, or internally exchanges a message M before proceeding with behavior B. We also have parallel composition $B_1 \mid B_2$, inaction $\mathbf{0}$, and recursion. Message types M are specified by a polarity p (either output $\mathtt{!}$, input $\mathtt{?}$ or internal action τ), a pair label-direction l^d, and the type C of the name communicated in the message. Notice that a message M may refer to an *internal* exchange between two partners, if it is of the form $\tau\, l^d(C)$. We write M for $M.\mathbf{0}$, where appropriate, and $p\, l(C)$ for $p\, l^{\downarrow}(C)$. The M_is in a branch type $\&_{i\in I}\{M_i.B_i\}$ are of polarity $\mathtt{?}$ and in a choice type $\oplus_{i\in I}\{M_i.B_i\}$ of polarity $\mathtt{!}$ or τ (so as to represent internal choice). We abbreviate $\oplus\{M.B\}$ and $\&\{M.B\}$ with $M.B$.

For typing purposes, we split the set of labels $\mathcal{L}$ into shared $\mathcal{L}_\star$ and plain $\mathcal{L}_p$ labels. Messages which are to be used linearly are defined with plain labels, and messages which are to be used exponentially are defined with shared labels. In such a way we distinguish two common interaction patterns in service-oriented computing: a service is expected to be available *exponentially* in the sense that there can be multiple clients trying to use the same service simultaneously. On the other hand during an ongoing service interaction messages that flow between collaborating partners are expected to be used *linearly*, in the sense that there should be a unique pair of parties that can interact on a specific message at a given moment (race absence).

Our types are related by a subtyping relation <:, which relies on some auxiliary operations we now introduce. The key ones are predicate *apartness* $B_1 \# B_2$ and direction projection $d(B)$. Intuitively, two types are apart when they may type subsystems that may be safely composed without undesirable interferences. Essentially, apartness ensures disjointness of plain ("linear") types, and consistency of shared ("exponential") types (cf. [21]). To characterize the plain and shared label sets of a type we introduce the set of message types (M) of a behavioral type B, noted $Msg_{\mathcal{L}}(B)$, and the set of directed labels (l^d) of a behavioral type B, noted $Lab_{\mathcal{L}}(B)$.

Definition 3.2 (Message Set). *We denote by $Msg_{\mathcal{L}}(B)$ the set of message types defined with labels in $\mathcal{L}$ of a behavioral type B, defined as follows:*

$$
\begin{array}{lcl}
Msg_{\mathcal{L}}(\mathbf{0}) & \triangleq & \emptyset \\
Msg_{\mathcal{L}}(B_1 \mid B_2) & \triangleq & Msg_{\mathcal{L}}(B_1) \cup Msg_{\mathcal{L}}(B_2) \\
Msg_{\mathcal{L}}(\mathcal{X}) & \triangleq & \emptyset \\
Msg_{\mathcal{L}}(\mathtt{rec}\,\mathcal{X}.B) & \triangleq & Msg_{\mathcal{L}}(B) \\
Msg_{\mathcal{L}}(p\,l^d(C).B) & \triangleq & \{(p\,l^d(C)) \mid l \in \mathcal{L}\} \cup Msg_{\mathcal{L}}(B) \\
Msg_{\mathcal{L}}(\oplus_{i\in I}\{M_i.B_i\}) & \triangleq & \bigcup_{i\in I} Msg_{\mathcal{L}}(M_i.B_i) \\
Msg_{\mathcal{L}}(\&_{i\in I}\{M_i.B_i\}) & \triangleq & \bigcup_{i\in I} Msg_{\mathcal{L}}(M_i.B_i)
\end{array}
$$

Definition 3.3 (Label Set). *We denote by $Lab_{\mathcal{L}}(B)$ the set of labels from $\mathcal{L}$ of a behavioral type B, defined as follows:*

$$Lab_{\mathcal{L}}(B) \triangleq \{l^d \mid (p\,l^d(C)) \in Msg_{\mathcal{L}}(B)\}$$

For example, given some behavioral type B, $Msg_{\mathcal{L}_p}(B)$ is the set of all plain (in $\mathcal{L}_p$) message types ($p\,l^d(C)$) occurring in B, leaving out message types defined on shared labels (those belonging to $\mathcal{L}_\star$).

To capture the consistency of shared types we introduce conformance $\asymp$. Given behavioral types B_1 and B_2, we let $B_1 \asymp B_2$ state that message types with shared labels occur both in B_1 and B_2 with identical argument types (so that B_1 and B_2 are compatible on shared labels).

Definition 3.4 (Conformance). *We say two behavioral types B_1, B_2 are conformant, noted $B_1 \asymp B_2$, if for any two message types $p_1\,l^d(C_1)$ and $p_2\,l^d(C_2)$ such that*

$$(p_1\,l^d(C_1)) \in Msg_{\mathcal{L}_\star}(B_1) \qquad \textit{and} \qquad (p_2\,l^d(C_2)) \in Msg_{\mathcal{L}_\star}(B_2)$$

then $C_1 = C_2$ and if $p_i = ?$ then $p_j = \tau$ for $\{i, j\} = \{1, 2\}$.

Conformance is also determined based on the polarities of the messages. For instance, two message types defined on shared labels and polarity ! are conformant as they represent compatible calls to the same service. We exclude the cases of messages presenting dual

polarities (! and ?) and when both messages present ? polarities: the former will be used to force such messages to synchronize, which means a τ will be introduced in the type to represent the possible synchronization; the latter is used to check the compatibility of two service definitions. In both cases the combination of such types is explained by the behavioral merge (Definition 3.11). We may now define apartness.

Definition 3.5 (Apartness). *Behavioral types B_1, B_2 are apart, noted $B_1 \# B_2$, if their plain label sets are disjoint ($Lab_{\mathcal{L}_p}(B_1) \# Lab_{\mathcal{L}_p}(B_2)$) and they are conformant ($B_1 \asymp B_2$).*

Two types are apart with respect to messages defined on plain labels if they are defined on disjoint sets of plain labeled messages ($Lab_{\mathcal{L}_p}$), and with respect to messages defined on shared labels if they are conformant $\asymp$.

The direction projection $d(B)$ is another important operation used in our subtyping rules. The projection $d(B)$ in the direction d of a behavioral type B consists in the selection of all messages that have the given direction d while filtering out the ones in the other direction, offering a partial view of behavior B from the viewpoint of d. We also write, e.g., $\uparrow B$ for $\uparrow (B)$, to lighten the notation. Informally, we sometimes refer to $\downarrow B$ as the "here interface" of B, and likewise for $\uparrow B$ as the "up interface". We show an example and define direction projection.

Example 3.6. *We illustrate the projection of choice and branch types. Consider type:*

$$\&\{?\,\texttt{bookA}^{\uparrow}().!\,\texttt{book}^{\downarrow}();\ ?\,\texttt{cancelA}^{\uparrow}().!\,\texttt{cancel}^{\downarrow}()\}$$

which describes a process that can either input message bookA *or message* cancelA *in the enclosing conversation, and afterwards output message* book *or message* cancel*, respectively. Projecting the type in the $\uparrow$ direction then results in the branch type of the two $\uparrow$ messages:*

$$\uparrow (\&\{?\,\texttt{bookA}^{\uparrow}().!\,\texttt{book}^{\downarrow}();\ ?\,\texttt{cancelA}^{\uparrow}().!\,\texttt{cancel}^{\downarrow}()\}) = \&\{?\,\texttt{bookA}^{\uparrow}();\ ?\,\texttt{cancelA}^{\uparrow}()\}$$

On the other hand, the $\downarrow$ projection describes that the process chooses one of the $\downarrow$ behaviors of the continuations (since the first branch is invisible from this view), as follows:

$$\downarrow (\&\{?\,\texttt{bookA}^{\uparrow}().!\,\texttt{book}^{\downarrow}();\ ?\,\texttt{cancelA}^{\uparrow}().!\,\texttt{cancel}^{\downarrow}()\}) = \oplus\{!\,\texttt{book}^{\downarrow}();\ !\,\texttt{cancel}^{\downarrow}()\}$$

Definition 3.7 (Direction Projection). *For each direction d, the projection $d(B)$ of behavioral type B along direction d is inductively defined as follows:*

$$\begin{array}{lll}
d(\mathbf{0}) & \triangleq & \mathbf{0} \\
d(\mathcal{X}) & \triangleq & \mathcal{X} \\
d(\texttt{rec}\,\mathcal{X}.B) & \triangleq & \texttt{rec}\,\mathcal{X}.d(B) \\
d(B_1 \mid B_2) & \triangleq & d(B_1) \mid d(B_2) \\
d(!\,l^{d'}(C).B) & \triangleq & d(B) \qquad\qquad (\textit{if } d \neq d') \\
d(\oplus_{i\in I}\{p_i\, l_i^d(C_i).B_i\}) & \triangleq & \oplus_{i\in I}\{p_i\, l_i^d(C_i).d(B_i)\} \\
d(\oplus_{i\in I}\{!\, l_i^{d'}(C_i).B_i\}) & \triangleq & \oplus_{i\in I}\{d(B_i)\} \qquad (\textit{if } d \neq d' \textit{ and } d(B_i) = !\,l^d(C).B, i \in I) \\
d(?\,l^{d'}(C).B) & \triangleq & d(B) \qquad\qquad (\textit{if } d \neq d') \\
d(\&_{i\in I}\{?\, l_i^d(C_i).B_i\}) & \triangleq & \&_{i\in I}\{?\, l_i^d(C_i).d(B_i)\} \\
d(\&_{i\in I}\{?\, l_i^{d'}(C_i).B_i\}) & \triangleq & \oplus_{i\in I}\{d(B_i)\} \qquad (\textit{if } d \neq d' \textit{ and } d(B_i) = !\,l^d(C).B, i \in I)
\end{array}$$

Our subtyping rules rely on some auxiliary notation used in characterizing our admissible recursive types. We introduce $B^\star$ which denotes a "shareable" behavioral type defined (exclusively) with shared labels (from $\mathcal{L}_\star$), hence not referring any plain label (from $\mathcal{L}_p$). Also we use $B\langle\mathcal{X}\rangle$ to represent a behavioral type where the recursion variable $\mathcal{X}$ may occur as a leaf, and all its plain labels appear in messages that prefix the recursion variable. We define $B^\star$ and $B\langle\mathcal{X}\rangle$.

Definition 3.8. *Shared messages, noted $M^\star$, shared behavioral types, noted $B^\star$, and recursive behavioral types, noted $B\langle\mathcal{X}\rangle$, are defined as follows ($l^\star$ ranges over labels in $\mathcal{L}_\star$):*

$$\begin{array}{lcl} M^\star & ::= & !\, l^{\star d}(C) \\ B^\star & ::= & B_1^\star \mid B_2^\star \;\big|\; \mathbf{0} \;\big|\; \oplus_{i\in I}\{M_i^\star.B_i^\star\} \\ B\langle\mathcal{X}\rangle & ::= & B\langle\mathcal{X}\rangle \mid B^\star \;\big|\; \mathbf{0} \;\big|\; \mathcal{X} \;\big|\; \oplus_{i\in I}\{M_i.B_i\langle\mathcal{X}\rangle\} \;\big|\; \&_{i\in I}\{M_i.B_i\langle\mathcal{X}\rangle\} \end{array}$$

Type $B\langle\mathcal{X}\rangle$ thus characterizes recursive processes that can safely have several active concurrent instances, where by "safely" we intend that the concurrent instances share only a message alphabet from $\mathcal{L}_\star$, hence do not share any (linear) message alphabet from $\mathcal{L}_p$. Also, when characterizing persistent messages we use $\star M$ as an abbreviation of $\mathtt{rec}\,\mathcal{X}.M.\mathcal{X}$.

We may now present the subtyping relation. Intuitively, we say type T_1 is a subtype of type T_2, noted $T_1 <: T_2$, when a process of type T_1 can safely be used in a context where a process of type T_2 is expected. Subtyping provides a way to generalize the typing characterization of processes, by its use in the subsumption rule:

$$\frac{P :: T_1 \quad T_1 <: T_2}{P :: T_2}$$

Our subtyping rules express expected relationships of types, such as the commutative monoid rules for $(- \mid -, \mathbf{0})$, congruence principles, and the split rule:

$$n : [B_1 \mid B_2] \equiv n : [B_1] \mid n : [B_2]$$

which captures the notion that the behavior in a single conversation can be described through distinct pieces. For types T_1 and T_2 we write $T_1 \equiv T_2$ if $T_1 <: T_2$ and $T_2 <: T_1$. We adopt an equi-recursive approach to recursive types [26], based on simple unfoldings of recursive type terms:

$$\mathtt{rec}\,\mathcal{X}.T \equiv T\{\mathcal{X} \leftarrow \mathtt{rec}\,\mathcal{X}.T\}$$

We could also have adopted a more flexible theory via coinductive definitions, along the lines of [15]. Rule:

$$\mathbf{0} <: n : [\mathbf{0}]$$

allows us to introduce names in the type that are not (yet) used by the process.

The following rule expresses a contraction principle for shared messages:

$$M^\star \mid \star M^\star <: \star M^\star$$

$$T_1 \mid T_2 \equiv T_2 \mid T_1 \;\; (1) \qquad T_1 \mid (T_2 \mid T_3) \equiv (T_1 \mid T_2) \mid T_3 \;\; (2) \qquad T \mid \mathbf{0} \equiv T \;\; (3)$$

$$\mathtt{rec}\,\mathcal{X}.T \equiv T\{\mathcal{X} \leftarrow \mathtt{rec}\,\mathcal{X}.T\} \;\; (4) \qquad n : [B_1 \mid B_2] \equiv n : [B_1] \mid n : [B_2] \;\; (5)$$

$$\frac{M_i.B_i <: M_i'.B_i' \;\; (i \in I)}{\oplus_{i \in I}\{M_i.B_i\} <: \oplus_{i \in I}\{M_i'.B_i'\}}(6) \qquad \frac{M_i.B_i <: M_i'.B_i' \;\; (i \in I)}{\&_{i \in I}\{M_i.B_i\} <: \&_{i \in I}\{M_i'.B_i'\}}(7)$$

$$\frac{B_1 <: B_2}{M.B_1 <: M.B_2}(8) \qquad \frac{B_1 <: B_2}{\mathtt{rec}\,\mathcal{X}.B_1 <: \mathtt{rec}\,\mathcal{X}.B_2}(9) \quad \frac{T_1 <: T_2}{T_3 \mid T_1 <: T_3 \mid T_2}(10)$$

$$\frac{B_1 <: B_2}{n : [B_1] <: n : [B_2]}(11) \quad \frac{T_1 <: T_3 \quad T_3 <: T_2}{T_1 <: T_2}(12) \qquad T <: T \;\; (13)$$

$$B <: \downarrow B \mid \uparrow B \;\; (14) \qquad M^\star \mid \star M^\star <: \star M^\star \;\; (15)$$

$$\mathtt{rec}\,\mathcal{X}.(M_1^\star \mid \ldots \mid M_k^\star \mid B\langle\mathcal{X}\rangle) <: \star M_1^\star \mid \ldots \mid \star M_k^\star \mid \mathtt{rec}\,\mathcal{X}.B\langle\mathcal{X}\rangle \;\; (16)$$

$$M.(B_1 \mid B_2) <: M.B_1 \mid B_2 \;\; (M \# B_2, fv(B_2) = \emptyset) \;\; (17) \qquad \mathbf{0} <: n : [\mathbf{0}] \;\; (18)$$

Figure 7: Subtyping Rules.

which describes that a process that independently outputs a message once and infinitely often can be safely used in a context where a process that sends such message infinitely often is expected.

The following rule allows for recursive types to export their shared interface separately:

$$\mathtt{rec}\,\mathcal{X}.(M_1^\star \mid \ldots \mid M_k^\star \mid B\langle\mathcal{X}\rangle) <: \star M_1^\star \mid \ldots \mid \star M_k^\star \mid \mathtt{rec}\,\mathcal{X}.B\langle\mathcal{X}\rangle$$

The rule then allows for a process that specifies a number of shared messages in between its repeated executions to be characterized by the type that separately specifies the shared message interface and the recursive behavior. A key subtyping rule that introduces some flexibility at the level of protocol specification is the following:

$$M.(B_1 \mid B_2) <: M.B_1 \mid B_2 \;\; (M \# B_2, fv(B_2) = \emptyset)$$

which allows for sequential protocols to export a more general concurrent interface, provided the behaviors specified in parallel are apart ($M \# B_2$). The intuition is that if a process performs action M and after which exhibits behavior B_2 then it can safely be used in a context that expects a process that exhibits simultaneously action M and behavior B_2. We use $fv(B)$ to denote the set of recursion variables of type B.

The following rule expresses a crucial subtyping principle, where we allow a behavioral type to be decomposed in its two projections according to the message directions:

$$B <: \downarrow B \mid \uparrow B$$

The rule characterizes that a process that specifies some, possibly interleaved, behavior in the current and enclosing conversations, can safely be used in a context where a process that exhibits such behaviors independently is expected. Figure 7 presents the subtyping rules and axioms.

We now introduce a key operation in which our typing rules rely: the ternary relation merge $B = B_1 \bowtie B_2$. The merge relation is used to define the composition of two types, so that if $B = B_1 \bowtie B_2$ then B is a particular (in general not unique) behavioral combination of the types B_1 and B_2. Merge is defined not only in terms of spatial separation, but also, and crucially, in terms of merging behavioral "traces". Notice also that it is not always the case that there is B such that $B = B_1 \bowtie B_2$. On the other hand, if some such B exists, we use $B_1 \bowtie B_2$ to non-deterministically denote any such B (e.g., in conclusions of type rules). Intuitively, $B = B_1 \bowtie B_2$ holds if B_1 and B_2 may safely synchronize or interleave so as to produce behavioral type B.

Before presenting the definition of the merge relation we introduce some auxiliary operations: the initial label set of a behavioral type B, noted $\mathcal{I}(B)$, and message type substitution, noted $B\{M_1 \leftarrow M_2\}$. The initial label set of a behavioral type B collects the set of labels of the actions immediately active in B.

Definition 3.9 (Message Type Substitution). *We denote by $B\{M_1 \leftarrow M_2\}$ the type obtained by replacing all occurrences of message type M_1 with message type M_2 in type B, defined inductively in the structure of types as follows:*

$$
\begin{array}{lcll}
\mathbf{0}\{M_1 \leftarrow M_2\} & \triangleq & \mathbf{0} & \\
(B_1 \mid B_2)\{M_1 \leftarrow M_2\} & \triangleq & (B_1\{M_1 \leftarrow M_2\}) \mid (B_2\{M_1 \leftarrow M_2\}) & \\
\mathcal{X}\{M_1 \leftarrow M_2\} & \triangleq & \mathcal{X} & \\
(\mathtt{rec}\,\mathcal{X}.B)\{M_1 \leftarrow M_2\} & \triangleq & \mathtt{rec}\,\mathcal{X}.(B\{M_1 \leftarrow M_2\}) & \\
(M_1.B)\{M_1 \leftarrow M_2\} & \triangleq & M_2.(B\{M_1 \leftarrow M_2\}) & \\
(M.B)\{M_1 \leftarrow M_2\} & \triangleq & M.(B\{M_1 \leftarrow M_2\}) & (\text{if } M \neq M_1) \\
(\oplus_{i\in I}\{M_i.B_i\})\{M_1 \leftarrow M_2\} & \triangleq & \oplus_{i\in I}\{(M_i.B_i)\{M_1 \leftarrow M_2\}\} & \\
(\&_{i\in I}\{M_i.B_i\})\{M_1 \leftarrow M_2\} & \triangleq & \&_{i\in I}\{(M_i.B_i)\{M_1 \leftarrow M_2\}\} &
\end{array}
$$

Definition 3.10 (Initial Label Set). *The initial label set of a behavioral type B, noted $\mathcal{I}(B)$, is defined as follows:*

$$
\begin{array}{lcllcl}
\mathcal{I}(\mathbf{0}) & \triangleq & \emptyset & \mathcal{I}(B_1 \mid B_2) & \triangleq & \mathcal{I}(B_1) \cup \mathcal{I}(B_2) \\
\mathcal{I}(\mathcal{X}) & \triangleq & \emptyset & \mathcal{I}(\mathtt{rec}\,\mathcal{X}.B) & \triangleq & \mathcal{I}(B) \\
\mathcal{I}(\oplus_{i\in I}\{M_i.B_i\}) & \triangleq & \bigcup_{i\in I} \mathcal{I}(M_i.B_i) & \mathcal{I}(\&_{i\in I}\{M_i.B_i\}) & \triangleq & \bigcup_{i\in I} \mathcal{I}(M_i.B_i) \\
\mathcal{I}(p\, l^d(C).B) & \triangleq & \{l^d\} & & &
\end{array}
$$

We discuss the key rules of the merge relation, then present its definition. Rule:

$$
\frac{B_1 \# B_2}{B_1 \mid B_2 = B_1 \bowtie B_2}(Apart)
$$

captures the composition of two independent behaviors B_1 and B_2, by specifying them in parallel in the resulting merge. The behaviors are independent since they are apart $\#$. The merge of behaviors which are not independent must synchronize the actions that are not independent. There are two rules that explain such synchronizations, one for messages defined on plain labels, and the other for messages defined on shared labels. For plain messages synchronization we have the following rule:

$$\frac{\forall_{i\in I}(B_i = B_i^- \bowtie B_i^+ \qquad l_i \in \mathcal{L}_p)}{\oplus_{i\in I}\{\tau\, l_i^{\downarrow}(C_i).B_i\} = \&_{i\in I}\{?\, l_i^{\downarrow}(C_i).B_i^-\} \bowtie \oplus_{i\in I}\{!\, l_i^{\downarrow}(C_i).B_i^+\}}(\textit{Plain-r})$$

that merges a branch and choice type which are dual in an "internal" choice type, i.e., a choice between messages with polarity τ. The continuations are merges of the corresponding continuations of the branches and choices. Rule (*Plain-r*) thus allows for $\tau\, l^{\downarrow}$ plain message types ("here" internal interactions) to be separated into send ! and receive ? capabilities in respective choice and branch constructs.

Shared message synchronization is captured by rule:

$$\frac{B \asymp !\, l^d(C) \qquad l \in \mathcal{L}_\star}{B\{!\, l^d(C) \leftarrow \tau\, l^d(C)\} \mid \star\ ?\, l^d(C) = B \bowtie \star\, ?\, l^d(C)}(\textit{Shared-r})$$

which synchronizes a persistently available input message type with all corresponding output message types. The resulting merge is then the type obtained replacing all $!\, l^d$ message types with $\tau\, l^d$ in B, in parallel with the persistent input message type: shared labels synchronize and leave open the possibility for further synchronizations, expecting further outputs from the environment, while plain message synchronization characterizes the uniquely determined synchronization on that plain label.

The following rule ensures compatibility of persistent shared input specifications:

$$\frac{l \in \mathcal{L}_\star}{\star ?\, l^d(C) = \star ?\, l^d(C) \bowtie \star ?\, l^d(C)}(\textit{SharedInp})$$

Thus, two persistent shared inputs may be merged if they are characterized by exactly the same type. The following rule allows for the merge to interleave a message prefix:

$$\frac{M \# B_2 \qquad B' \mid B'' \equiv B_1 \bowtie B_2 \qquad M \# B'' \qquad \mathcal{I}(B') \subseteq \mathcal{I}(B_1) \qquad \mathcal{I}(B'') \subseteq \mathcal{I}(B_2)}{M.B' \mid B'' = M.B_1 \bowtie B_2}(\textit{Shuffle-l})$$

Rule (*Shuffle-l*) explains the composition of behaviors $M.B_1$ and B_2 by first composing B_1 and B_2 (since M is apart from B_2 it does not interfere with B_2) and second by placing the message prefix $M.B'$ so as to maintain (some of) the sequentiality information originally specified in $M.B_1$. On the one hand, no extra sequentiality may be imposed by prefixing B' with M in the resulting merge, with respect to the one originally specified in $M.B_1$. This is guaranteed by condition $\mathcal{I}(B') \subseteq \mathcal{I}(B_1)$, which says that the labels of the immediately active messages of B' are a subset of the labels of the immediately active messages of B_1. On the other hand the behavior B'' which is specified in parallel to M has its initial actions defined

by a subset of the ones specified by B_2, so no behaviors that occurred only in the continuation of M will be exposed in parallel. In such way, we allow for type synchronizations to occur in the continuation of message prefixes. The following rule:

$$\frac{M \not\# B_1 \qquad B' = B_1 \bowtie M.B_2 \qquad B = B' \bowtie B_3}{B = B_1 \bowtie M.B_2 \mid B_3}(\mathit{MsgPar\text{-}r})$$

explains the merge of the parallel composition $M.B_2 \mid B_3$ with type B_1 by first merging B_1 with $M.B_2$ then merging the resulting type with B_3, provided M is not apart $\#$ from B_1. This rule allows for several messages that are originally specified in parallel to merge with the same thread, e.g., in the merge:

$$\begin{aligned}&\tau\,\texttt{askPrice}^{\downarrow}(Tp).\tau\,\texttt{readVal}^{\downarrow}(Tm) = \\ &\qquad !\,\texttt{askPrice}^{\downarrow}(Tp).?\,\texttt{readVal}^{\downarrow}(Tm) \bowtie ?\,\texttt{askPrice}^{\downarrow}(Tp) \mid !\,\texttt{readVal}^{\downarrow}(Tm)\end{aligned}$$

We may now define the behavioral types merge relation.

Definition 3.11 (Behavioral Types Merge Relation). *The merge ternary relation, defined on behavioral types $B = B_1 \bowtie B_2$, is inductively defined in Figure 8.*
N.B. We omit from the Figure rules symmetric to (MsgPar-r), (MsgPar-l) and (Par).

We state some properties of the behavioral types merge relation.

Lemma 3.12. *The behavioral types merge relation is commutative and associative:*

(*1*). *If $B = B_1 \bowtie B_2$ then $B = B_2 \bowtie B_1$.*

(*2*). *If $B' = B_1 \bowtie B_2$ and $B = B' \bowtie B_3$ then there is B'' such that $B'' = B_2 \bowtie B_3$ and $B = B_1 \bowtie B''$.*

Proof. (*1*): follows immediately from the definition. (*2*) by induction on the derivation of $B' = B_1 \bowtie B_2$ and $B = B' \bowtie B_3$ (see Appendix A). ■

We show a couple of examples that illustrate how types may be merged.

Example 3.13. *Consider type:*

$$?\,\texttt{buy}^{\downarrow}(Tp).!\,\texttt{price}^{\downarrow}(Tm).?\,\texttt{accept}^{\downarrow}().\tau\,\texttt{product}^{\downarrow}(Tp).!\,\texttt{details}^{\downarrow}(Td)$$

which describes a process that inputs message `buy`*, then outputs message* `price`*, then inputs messages* `accept`*, then internally exchanges message* `product`*, and finally outputs message* `details`*. When merged with the type:*

$$?\,\texttt{price}^{\downarrow}(Tm).!\,\texttt{accept}^{\downarrow}()$$

specifying the dual polarities for `price` *and* `accept`*, it yields type:*

$$?\,\texttt{buy}^{\downarrow}(Tp).\tau\,\texttt{price}^{\downarrow}(Tm).\tau\,\texttt{accept}^{\downarrow}().\tau\,\texttt{product}^{\downarrow}(Tp).!\,\texttt{details}^{\downarrow}(Td)$$

which specifies the composite behavior that inputs message `buy`*, then has internal interactions on messages* `price`*,* `accept` *and* `product`*, and finally outputs message* `details`*.*

$$\frac{l \in \mathcal{L}_\star}{\star ?\, l^d(C) = \star ?\, l^d(C) \bowtie \star ?\, l^d(C)}(\mathit{SharedInp})$$

$$\frac{B \asymp\, !\, l^d(C) \qquad l \in \mathcal{L}_\star}{B\{!\, l^d(C) \leftarrow \tau\, l^d(C)\} \mid \star\ ?\, l^d(C) = B \bowtie \star\, ?\, l^d(C)}(\mathit{Shared}\text{-}r)$$

$$\frac{B \asymp\, !\, l^d(C) \qquad l \in \mathcal{L}_\star}{B\{!\, l^d(C) \leftarrow \tau\, l^d(C)\} \mid \star\ ?\, l^d(C) = \star\, ?\, l^d(C) \bowtie B}(\mathit{Shared}\text{-}l)$$

$$\frac{\forall_{i \in I}(B_i = B_i^- \bowtie B_i^+ \qquad l_i \in \mathcal{L}_p)}{\oplus_{i \in I}\{\tau\, l_i^{\downarrow}(C_i).B_i\} = \&_{i \in I}\{?\, l_i^{\downarrow}(C_i).B_i^-\} \bowtie \oplus_{i \in I}\{!\, l_i^{\downarrow}(C_i).B_i^+\}}(\mathit{Plain}\text{-}r)$$

$$\frac{\forall_{i \in I}(B_i = B_i^+ \bowtie B_i^- \qquad l_i \in \mathcal{L}_p)}{\oplus_{i \in I}\{\tau\, l_i^{\downarrow}(C_i).B_i\} = \oplus_{i \in I}\{!\, l_i^{\downarrow}(C_i).B_i^+\} \bowtie \&_{i \in I}\{?\, l_i^{\downarrow}(C_i).B_i^-\}}(\mathit{Plain}\text{-}l)$$

$$\frac{M \# B_1 \qquad B' \mid B'' \equiv B_1 \bowtie B_2 \qquad M \# B' \qquad \mathcal{I}(B'') \subseteq \mathcal{I}(B_2) \qquad \mathcal{I}(B') \subseteq \mathcal{I}(B_1)}{B' \mid M.B'' = B_1 \bowtie M.B_2}(\mathit{Shuffle}\text{-}r)$$

$$\frac{M \# B_2 \qquad B' \mid B'' \equiv B_1 \bowtie B_2 \qquad M \# B'' \qquad \mathcal{I}(B') \subseteq \mathcal{I}(B_1) \qquad \mathcal{I}(B'') \subseteq \mathcal{I}(B_2)}{M.B' \mid B'' = M.B_1 \bowtie B_2}(\mathit{Shuffle}\text{-}l)$$

$$\frac{M \not\# B_1 \qquad B' = B_1 \bowtie M.B_2 \qquad B = B' \bowtie B_3}{B = B_1 \bowtie M.B_2 \mid B_3}(\mathit{MsgPar}\text{-}r)$$

$$\frac{M \not\# B_3 \qquad B' = M.B_2 \bowtie B_3 \qquad B = B_1 \bowtie B'}{B = B_1 \mid M.B_2 \bowtie B_3}(\mathit{MsgPar}\text{-}l)$$

$$\frac{B = B_1 \bowtie B_2}{\texttt{rec}\, \mathcal{X}.B = \texttt{rec}\, \mathcal{X}.B_1 \bowtie \texttt{rec}\, \mathcal{X}.B_2}(\mathit{Rec}) \qquad \frac{}{\mathcal{X} = \mathcal{X} \bowtie \mathcal{X}}(\mathit{Var})$$

$$\frac{B' = B_1 \bowtie B_2 \qquad B'' = B_3 \bowtie B_4 \qquad B' \# B''}{B' \mid B'' = (B_1 \mid B_3) \bowtie (B_2 \mid B_4)}(\mathit{Par}) \qquad \frac{B_1 \# B_2}{B_1 \mid B_2 = B_1 \bowtie B_2}(\mathit{Apart})$$

Figure 8: Behavioral Type Merge Relation Rules.

Example 3.14. *Consider type:*

$$!\,\texttt{buy}(Tp).?\,\texttt{price}(Tm).?\,\texttt{details}(Td)$$

which characterizes a process that outputs message buy*, then inputs messages* price *and* details*. When merged with type:*

$$?\,\texttt{product}(Tp).!\,\texttt{details}(Td)$$

which characterizes a process that inputs message product *and then outputs message* details*, we obtain type (among other possibilities):*

$$!\,\texttt{buy}(Tp).?\,\texttt{price}(Tm) \mid ?\,\texttt{product}(Tp).\tau\,\texttt{details}(Td)$$

where message details *is specified to be internally exchanged after the reception of message* product*. In such case, the original sequentiality information tells us that the reception of message* details *happens after the reception of message* price *and also that the emission of message* details *happens after the reception of message* product*. Since* product *and* price *are temporally unrelated,* details *will be specified after one or the other. Thus, the above merge may also yield:*

$$!\,\texttt{buy}(Tp).?\,\texttt{price}(Tm).\tau\,\texttt{details}(Td) \mid ?\,\texttt{product}(Tp)$$

Such merges are explained by rule (Shuffle) which allow us to shuffle messages on the left and right hand sides, while making sure that no extra sequentiality is imposed.

The type system relies on a merge relation between process types, which lifts the merge between behavioral types by realizing per conversation behavioral type merges. We first define the domain of a located type.

Definition 3.15 (Domain of a Process Type). *The domain of a process type T, noted $dom(T)$, is defined as follows:*

$$dom(T) \triangleq \{n \mid T \equiv T' \mid n : C\}$$

Definition 3.16 (Process Types Merge Relation). *The merge relation $T = T_1 \bowtie T_2$ between process types is inductively defined as follows:*

$$\frac{B = B_1 \bowtie B_2}{n : [B] = n : [B_1] \bowtie n : [B_2]} \qquad T = T \bowtie \mathbf{0} \qquad T = \mathbf{0} \bowtie T \qquad \frac{dom(L_1) \# dom(L_2)}{L_1 \mid L_2 = L_1 \bowtie L_2}$$

$$\frac{\forall_{i \in 1,2}\ L_i = L_i^+ \bowtie L_i^- \quad dom(L_1) \# dom(L_2)}{L_1 \mid L_2 = L_1^+ \mid L_2^+ \bowtie L_1^- \mid L_2^-} \qquad \frac{B = B_1 \bowtie B_2 \quad L = L_1 \bowtie L_2}{L \mid B = L_1 \mid B_1 \bowtie L_2 \mid B_2}$$

We define *closed* behavioral types which characterize processes that have matching receives for all sends.

Definition 3.17 (Closed Types). *We say a behavioral type B is closed, noted closed(B), if for any message type $(\mathsf{p}\, l^d(C))$ such that $(\mathsf{p}\, l^d(C)) \in Msg_{\mathcal{L}}(B)$ then either $\mathsf{p} = \tau$, or $\mathsf{p} = ?$ and $l \in \mathcal{L}_\star$ and $B <: B' \mid \star\, ?\, l^d(C)$. We say a process type T is closed, noted closed(T), if for any type B such that $T \equiv T' \mid n : [B]$ we have closed(B).*

Figure 9 presents our typing rules. Rule (*Par*) types the parallel composition by merging the types of the branches. In rule (*Res*) we use *closed*(B), to avoid hiding conversation names where unmatched communications still persist (necessary to ensure deadlock absence). In rule (*Rec*) by L_M we denote a located type of the form $n_1 : [M_1^\star] \mid \ldots \mid n_k : [M_k^\star]$, then by $\star L_M$ we denote $n_1 : [\star M_1^\star] \mid \ldots \mid n_k : [\star M_k^\star]$. The rule states that the process is well typed under an environment that offers persistent messages M_i under conversations n_i, and offers persistent behavior B in the current conversation. Recursive processes define the intended shared behavior using shared messages, in such way allowing several instances of the (shared part) of the recursive process to be concurrently active - the types of these several instances must be apart # (see Definition 3.5).

Rule (*Piece*) types a (piece of a) conversation. Process P expects some behavior located in conversations L, and some behavior B in the current conversation. The type in the conclusion is obtained by merging the process type L with a type that describes the behavior of the new conversation piece, in parallel with the type of the toplevel conversation, the now current conversation. Essentially the type of each projection (along the two directions) is collected appropriately: the "here" behavior projection $\downarrow B$ is the behavior in conversation n, and the "up" behavior projection $\uparrow$ of P becomes the "here" behavior at the toplevel conversation, via $loc(\uparrow B)$. Type $loc(B)$ is obtained from B by setting the direction of all messages in B to $\downarrow$.

In rule (*Input*) the premise states that processes P_i require some located behavior L, some current conversation behavior B_i, and some behavior at conversation x_i. Then, the conclusion states that the input summation process is well-typed under type L, with the behavior interface becoming the branch of the types of the continuations prefixed by the messages $?\, l_i^d(C_i)$. In rule (*Output*) notice that the context type is a separate $\bowtie$ view of the context, which means that the type being sent may actually be some separate part of the type of some conversation, which will be (partially) delegated away. This mechanism is crucial to allow external partners to join in on ongoing conversations in a disciplined way. The behavioral interface of the output prefixed process is a choice type, where one of the choices corresponds to the message specified in the output prefix, and after which proceeds as specified in the type of the continuation of the output prefix.

Notice the asymmetry between the (*Output*) and (*Input*) rules: while we can safely consider that the process chooses the specified action in between any set of choices that contains it, we can not forget some branches in the branch type, since this would allow undesired matches between choice and branch types. If a process does not fully reveal the branches it offers, then placing such process in an environment that may actually choose the "forgotten" branch may give rise to unexpected behaviors, i.e., behaviors not described by the type (cf., [12] where a similar problem arises in contract compliance).

$$\frac{P :: T_1 \quad Q :: T_2}{P \mid Q :: T_1 \bowtie T_2}(Par) \qquad \frac{}{\mathbf{0} :: \mathbf{0}}(Stop)$$

$$\frac{P :: T \mid a : [B] \quad (closed(B), a \notin dom(T))}{(\boldsymbol{\nu} a)P :: T}(Res)$$

$$\frac{P :: L_M \mid B\langle\mathcal{X}\rangle}{\mathbf{rec}\,\mathcal{X}.P :: \star L_M \mid \mathtt{rec}\,\mathcal{X}.B\langle\mathcal{X}\rangle}(Rec) \qquad \frac{}{\mathcal{X} :: \mathcal{X}}(RecVar)$$

$$\frac{P :: L \mid B}{n \blacktriangleleft [P] :: (L \bowtie n : [\downarrow B]) \mid loc(\uparrow B)}(Piece)$$

$$\frac{\forall_{i \in I}(P_i :: L \mid B_i \mid x_i : C_i \quad (x_i \notin dom(L)))}{\Sigma_{i \in I}\ l_i^d?(x_i).P_i :: L \mid \&_{i \in I}\{?\ l_i^d(C_i).B_i\}}(Input)$$

$$\frac{P :: L \mid B \quad (\exists j \in I.M_j.B_j = !\ l^d(C).B)}{l^d!(n).P :: (L \bowtie n : C) \mid \oplus_{i \in I}\{M_i.B_i\}}(Output)$$

$$\frac{P :: L \mid B_1 \mid x : [B_2] \quad (x \notin dom(L))}{\mathbf{this}(x).P :: L \mid (B_1 \bowtie B_2)}(This)$$

$$\frac{P :: T_1 \quad T_1 <: T_2}{P :: T_2}(Sub)$$

Figure 9: Typing Rules.

Rule (*This*) types the conversation awareness primitive, requiring behavior B_2 of conversation x to be a separate (in general, just partial) view of the current conversation. This allows to bind the current conversation to name x, and possibly send it to other parties that may need to join it.

Our subject reduction result (Theorem 3.20) relies on a notion of reduction on types, since each reduction step at the process level may require a modification in the typing, as expected from a behavioral type system.

Definition 3.18 (Type Reduction). *The type reduction relation between process types, noted $T_1 \to T_2$, is the least relation that satisfies the rules of Figure 10.*

We describe type reduction. Essentially, a synchronization at the process level is characterized by the reduction of the corresponding $\boldsymbol{\tau}$ message type (1). The reduction can take place at the level of a choice type (2) (we use $\tilde{B}$ to abbreviate $B_1; B_2; \ldots; B_k$). We also have the expected congruence rules (3) and (4), and a rule that closes type reduction under type equivalence (5). The type reduction relation is reflexive (6): in such way we characterize reductions in a part of the type which is not visible (i.e., under a restricted conversation).

$$\tau\, l^d(C).B \to B \;\;(1) \qquad \frac{B \to B'}{\oplus\{\tilde{B_1}; B; \tilde{B_2}\} \to B'}(2) \qquad \frac{B_1 \to B_2}{n : [B_1] \to n : [B_2]}(3)$$

$$\frac{T_1 \to T_2}{T_1 \mid T_3 \to T_2 \mid T_3}(4) \qquad \frac{T_1 \equiv T_1' \to T_2' \equiv T_2}{T_1 \to T_2}(5) \qquad T \to T \;\;(6)$$

Figure 10: Type Reduction.

We may now present our main soundness results. We state a Substitution Lemma, main auxiliary result to Subject Reduction (Theorem 3.20).

Lemma 3.19 (Substitution). *Let P be a well-typed process such that $P :: T \mid x : C$, for $x \notin dom(T)$ and types T, C. If there is type T' such that $T' = T \bowtie a : C$ then $P\{x \leftarrow a\} :: T'$.*

Proof. By induction on the length of the derivation of $P :: T \mid x : C$ (see Appendix A). ■

We may now state our Subject Reduction result.

Theorem 3.20 (Subject Reduction). *Let P be a process and T a type such that $P :: T$. If $P \to Q$ then there is type T' such that $T \to T'$ and $Q :: T'$.*

Proof. By induction on the derivation of the reduction $P \to Q$ (see Appendix A). ■

Our safety result asserts certain error processes are unreachable from well-typed processes. To define error processes we introduce static process contexts.

Definition 3.21 (Static Context). *Static process contexts, noted $\mathcal{C}[\cdot]$, are defined as follows:*

$$\mathcal{C}[\cdot] ::= (\boldsymbol{\nu} a)\mathcal{C}[\cdot] \;\big|\; P \mid \mathcal{C}[\cdot] \;\big|\; c \blacktriangleleft [\mathcal{C}[\cdot]] \;\big|\; \textbf{rec}\, \mathcal{X}.\mathcal{C}[\cdot] \;\big|\; \cdot$$

We also use $w(\lambda)$ to denote the sequence $c\ l^d$ of elements in the action label λ, for example $w((\nu a)c\ l^d!(a)) = c\ l^d$ and $w((\nu a)l^d!(a)) = l^d$.

Definition 3.22 (Error Process). *P is an error process if there is a static context $\mathcal{C}$ with $P = \mathcal{C}[Q \mid R]$ and there are $Q', R', \lambda_1, \lambda_2$ such that $Q \xrightarrow{\lambda_1} Q'$, $R \xrightarrow{\lambda_2} R'$ and $w(\lambda) = w(\lambda')$, $\overline{\lambda} \neq \lambda'$ and $w(\lambda)$ is not defined with a shared label.*

A process is not an error only if for each possible immediate interaction in a plain message there is at most a single sender and a single receiver.

Proposition 3.23 (Error Freeness). *Let P be a process. If there is T such that $P :: T$ then P is not an error process.*

Proof. The result follows from the definition of merge $\bowtie$. A parallel composition is well typed if the types of the parallel branches can be merged. Since it is not possible to synchronize message types with the same polarity (which is the case for competing messages) and such types are not apart $\#$ (the label sets are not disjoint) it is not possible to merge them. Hence the composition of processes that exhibit competing messages is not typable. ∎

By subject reduction (Theorem 3.20), we conclude that any process reachable from a well-typed process $P :: T$ is not an error ($\xrightarrow{*}$ denotes the reflexive transitive closure of $\rightarrow$).

Corollary 3.24 (Type Safety). *Let P be a process such that $P :: T$ for some T. If there is Q such that $P \xrightarrow{*} Q$, then Q is not an error process.*

Proof. Immediate from Theorem 3.20 and Proposition 3.23. ∎

Our type safety result ensures that, in any reduction sequence arising from a well-typed process, for each plain-labeled message ready to communicate there is always at most a unique input / output outstanding synchronization. More: arbitrary interactions in shared labels do not invalidate this invariant. Another consequence of subject reduction (Theorem 3.20) is that any message exchange inside the process must be explained by a τM prefix in the related conversation type (via type reduction), thus implying conversation fidelity, i.e., all conversations follow the prescribed protocols.

Corollary 3.25 (Conversation Fidelity). *Let P be a process such that $P :: T$ for some T. Then all conversations in P follow the protocols prescribed by T.*

Example 3.26. *Consider for instance the typing for the purchase conversation presented in the Introduction:*

$$\tau\,\texttt{buy}(Tp).\tau\,\texttt{price}(Tm).\tau\,\texttt{product}(Tp).\tau\,\texttt{details}(Td)$$

Such a (closed) type characterizes the global interaction scheme (the choreography*) of the interaction between parties Buyer, Seller and Shipper. In the light of Theorem 3.20 we then have that each interaction in the process is explained by a reduction of a τ message type. For instance, when message* buy *is exchanged between Buyer and Seller, such synchronization in the process is explained by the following reduction in the type:*

$$\begin{array}{l}\tau\,\texttt{buy}(Tp).\tau\,\texttt{price}(Tm).\tau\,\texttt{product}(Tp).\tau\,\texttt{details}(Td)\\ \rightarrow\\ \tau\,\texttt{price}(Tm).\tau\,\texttt{product}(Tp).\tau\,\texttt{details}(Td)\end{array}$$

after which the synchronization in message price *is explained by the following type reduction:*

$$\begin{array}{l}\tau\,\texttt{price}(Tm).\tau\,\texttt{product}(Tp).\tau\,\texttt{details}(Td)\\ \rightarrow\\ \tau\,\texttt{product}(Tp).\tau\,\texttt{details}(Td)\end{array}$$

and so on and so forth in the successive synchronizations. Thus, the type reductions actually capture the evolution of the choreographies throughout system execution.

$$\frac{P :: L \mid B}{\textsf{def}\ \mathsf{s} \Rightarrow P :: L \mid \texttt{?}\, s^{\downarrow}([\downarrow B]).(loc(\uparrow B))}(Def)$$

$$\frac{P :: L \mid B \quad (closed(\downarrow B \bowtie B_1))}{\textsf{new}\ n \cdot s \Leftarrow P :: L \mid n : [\texttt{!}\ s^{\downarrow}([B_1])] \mid loc(\uparrow B)}(New)$$

$$\frac{P :: L \mid B}{\textsf{join}\ n \cdot s \Leftarrow P :: L \mid n : [\texttt{!}\ s^{\downarrow}([B_1])] \mid (B \bowtie B_1)}(Join)$$

$$\frac{\textsf{def}\ s \Rightarrow P :: L_M \mid \texttt{?}\, s^{\downarrow}(C).B}{\star\textsf{def}\ s \Rightarrow P :: \star L_M \mid \texttt{rec}\, \mathcal{X}.\texttt{?}\, s^{\downarrow}(C).(B \mid \mathcal{X})}(RepDef)$$

Figure 11: Derived Typings.

In the expected polyadic extension of core CC and type system, considering also basic values and basic types, we would also exclude arity mismatch and type mismatch errors.

Remark 3.27. *Extending the conversation type language with basic types at the level of argument message types (e.g., $C ::= [B] \mid \texttt{Int} \mid \texttt{String} \mid \ldots$) would directly allow us to exclude systems where message argument types mismatch (via conformance $\asymp$ and merge $\bowtie$). To type identifiers which carry basic types we would impose a conformance check—such identifiers are always used with the same type—and exclude their use as conversation names.*

An essential property of any type system is the ability to automate the type checking procedure. Although we have not yet fully addressed the implementation issues, we may already state a crucial property that asserts the existence of such a type checking procedure.

Theorem 3.28 (Decidability of Type Checking). *Let P be a process where all bound names are type annotated. Then checking if $P :: T$ for some T is decidable.*

Proof. By induction on the derivation of $P :: T$, following expected lines. ∎

We prove decidability of our system, if binders are type annotated. This is an expected result, since our typing rules are syntax-directed, our merge relation is finitary, and typability is witnessed by a proof tree (as usual).

3.1. Derived Typings for Service Idioms

We show in Figure 11 the typing rules for the service idioms defined in Figure 5. Notice that these are admissible rules, mechanically derived from the typings of the encodings, not primitive rules. It is remarkable that the typings of these idiomatic constructs, defined from the small set of primitives in the core CC, admit the intended high level typings. Ignoring the continuation $loc(\uparrow B)$, the type of a service definition **def** has the form $\texttt{?}\, s([S])$, where S describes the service behavior. The dual type is required for a service instantiation, of the form $\texttt{!}\ s([S])$. However, such type must be located at some context n, cf. the semantics of the

$$
\begin{array}{lcl}
B_c & \triangleq & \tau\,\mathtt{buy}(Tp).\tau\,\mathtt{price}(Tm).\tau\,\mathtt{product}(Tp).\tau\,\mathtt{details}(Td) \\
B_{bu} & \triangleq & !\,\mathtt{buy}(Tp).?\,\mathtt{price}(Tm).?\,\mathtt{details}(Td) \\
B_{ss} & \triangleq & ?\,\mathtt{buy}(Tp).!\,\mathtt{price}(Tm).\tau\,\mathtt{product}(Tp).!\,\mathtt{details}(Td) \\
B_{se} & \triangleq & ?\,\mathtt{buy}(Tp).!\,\mathtt{price}(Tm).!\,\mathtt{product}(Tp) \\
B_{sh} & \triangleq & ?\,\mathtt{product}(Tp).!\,\mathtt{details}(Td) \\
T_{bu} & \triangleq & \mathit{Seller} : [!\,\mathtt{startBuy}([B_{ss}])] \\
T_{se} & \triangleq & \mathit{Seller} : [?\,\mathtt{startBuy}([B_{ss}]).B_{db}\,] \\
 & & \mid \mathit{Shipper} : [!\,\mathtt{newDelivery}([B_{sh}])] \\
T_{sh} & \triangleq & \mathit{Shipper} : [?\,\mathtt{newDelivery}([B_{sh}])] \\
T_{sys} & \triangleq & \mathit{Seller} : [\tau\,\mathtt{startBuy}([B_{ss}]).B_{db}] \\
 & & \mid \mathit{Shipper} : [\tau\,\mathtt{newDelivery}([B_{sh}])]
\end{array}
$$

$$
\mathit{Buyer} :: T_{bu} \qquad \mathit{Seller} :: T_{se} \qquad \mathit{Shipper} :: T_{sh}
$$
$$
\mathit{BuySystem} :: T_{sys}
$$

Figure 12: Typings for Buy System.

new idiom. The typing for **join** clearly displays the partial delegation of a conversation type fragment: B_1 represents the conversation type defining the participation of the incoming partner, while B specifies the residual that remains owned by the current process.

3.2. Typing Conversations

We illustrate the expressiveness of our type system by typing a couple of examples.

3.2.1. The Purchase Conversation

In Figure 12 we depict the types for the *Buyer*-*Seller*-*Shipper* example shown in Section 1.1. The type B_c describes all the interactions that take place under the three-party conversation, which consist in the sequence of internal actions on messages `buy`, `price`, `product` and `details`. Upon `startBuy` service instantiation the overall conversation type B_c is separated, so that *Buyer* retains its role in the conversation (B_{bu}) and *Seller* gets the rest of the conversation behavioral type (B_{ss}). The separation is such that $B_c = B_{bu} \bowtie B_{ss}$.

The type of the *Buyer* role in the conversation is then given by type B_{bu}, which specifies that *Buyer* first outputs message `buy`, then receives messages `price` and `details`. Notice that the type makes no explicit mention of who is the communicating partner, allowing for whoever to fulfill the intended protocol. Type B_{ss}, which specifies the behavior of the subsystem consisting of *Seller* and *Shipper*, is dual to B_{bu} in messages `buy`, `price` and `details`, and accounts for the internal interaction between *Seller* and *Shipper* in message `product`. Type B_{ss} is further separated in the type of the *Seller* role in the conversation (B_{se}) and the type of the *Shipper* role in the conversation (B_{sh}), such that $B_{ss} = B_{se} \bowtie B_{sh}$. When *Shipper* is asked to join in on the ongoing conversation it will be assigned the type of its own role in the conversation B_{sh}.

$$
\begin{array}{l}
\mathit{Client} \blacktriangleleft [\\
\quad \textbf{new}\ \mathit{NewsSite} \cdot \texttt{Newsfeed} \Leftarrow\\
\quad\quad \textbf{rec}\ \mathcal{X}.\texttt{post}?(\mathit{info}).\mathcal{X}\ \ ]\\
|\\
\mathit{NewsSite} \blacktriangleleft [\\
\quad \star\textbf{def}\ \texttt{Newsfeed} \Rightarrow\\
\quad\quad \textbf{rec}\ \mathcal{X}.\textbf{join}\ \mathit{NewsPortal} \cdot \texttt{NewsService} \Leftarrow \mathcal{X}\ \ ]\\
|\\
\mathit{BBC} \blacktriangleleft [\\
\quad \mathit{NewsPortal} \blacktriangleleft [\\
\quad\quad \star\textbf{def}\ \texttt{NewsService} \Rightarrow \texttt{post}!(\mathit{info})\ \ ]\ \ ]\\
|\\
\mathit{CNN} \blacktriangleleft [\\
\quad \mathit{NewsPortal} \blacktriangleleft [\\
\quad\quad \star\textbf{def}\ \texttt{NewsService} \Rightarrow \texttt{post}!(\mathit{info})\ \ ]\ \ ]
\end{array}
$$

Figure 13: The Newsfeed Conversation CC Code.

The types of each individual party are then T_{bu}, T_{se} and T_{sh}, for *Buyer*, *Seller* and *Shipper*, respectively. The type specified in the service message `startBuy` is then the type of the *Seller-Shipper* subsystem B_{ss} and the type of the service message `newDelivery` is the type of the *Shipper* role in the conversation B_{sh}. To type the *Seller* participant we consider the type of the *PriceDB* process to be $?\,\texttt{askPrice}(Tp).!\,\texttt{readVal}(Tm)$ so that the merge with the `startBuy` service "up" interface results in type B_{db}, such that $B_{db} \triangleq \tau\,\texttt{askPrice}(Tp).\tau\,\texttt{readVal}(Tm)$. The type of the whole system is then given by T_{sys} which specifies the two service instantiations as internal interactions τ.

3.2.2. The Newsfeed Conversation

Our next example shows a scenario where an unbounded number of parties may join a single conversation. We consider a `Newsfeed` service that, upon instantiation, asks an undetermined number of news service providers to join the conversation. Each one of the news service providers that join the conversation send a message `post` (containing some news information) that is picked up by the `Newsfeed` service client.

The CC implementation of this scenario is given in Figure 13. We define two particular news service providers (*BBC* and *CNN*) but the system is open to an unbounded number of such news providers. Notice that the `Newsfeed` service code continuously calls external news services to join in the conversation, and, in particular, countless copies of *BBC* and *CNN* news services may get to join the conversation. Notice also that the `Newsfeed` service client is continuously able to receive `post` messages, regardless of who is sending them.

The conversation types that capture the `Newsfeed` interaction are shown in Figure 14. The type of the `Newsfeed` conversation (given by *NewsfeedConversationT*) says that infinitely many `post` messages are exchanged, and that the system is still open to receive further `post` messages. Type *NewsfeedConversationT* is split in the types of the `Newsfeed`

$$
\begin{array}{lcl}
\mathit{NewsfeedConversationT} & \triangleq & \star ?\,\mathtt{post}(\mathit{infoT}) \mid \star\tau\,\mathtt{post}(\mathit{infoT}) \\
\mathit{NewsfeedClientT} & \triangleq & \star ?\,\mathtt{post}(\mathit{infoT}) \\
\mathit{NewsfeedServiceT} & \triangleq & \star !\,\mathtt{post}(\mathit{infoT}) \\
\mathit{NewsServiceT} & \triangleq & !\,\mathtt{post}(\mathit{infoT})
\end{array}
$$

$$\mathit{Client} :: \mathit{NewsSite} : [!\,\mathtt{Newsfeed}([\mathit{NewsfeedServiceT}])]$$

$$
\begin{array}{l}
\mathit{NewsSite} :: \mathit{NewsSite} : [\star ?\,\mathtt{Newsfeed}([\mathit{NewsfeedServiceT}])] \\
\qquad\qquad \mid \mathit{NewsPortal} : [\star !\,\mathtt{NewsService}([\mathit{NewsServiceT}])]
\end{array}
$$

$$\mathit{BBC} :: \mathit{NewsPortal} : [\star ?\,\mathtt{NewsService}([\mathit{NewsServiceT}])]$$

$$\mathit{CNN} :: \mathit{NewsPortal} : [\star ?\,\mathtt{NewsService}([\mathit{NewsServiceT}])]$$

$$
\begin{array}{l}
\mathit{NewsfeedSystem} \\
:: \\
\mathit{NewsSite} : [\tau\,\mathtt{Newsfeed}([\mathit{NewsfeedServiceT}]) \mid \star ?\,\mathtt{Newsfeed}([\mathit{NewsfeedServiceT}])] \\
\mid \\
\mathit{NewsPortal} : [\star\tau\,\mathtt{NewsService}([\mathit{NewsServiceT}]) \mid \star ?\,\mathtt{NewsService}([\mathit{NewsServiceT}])]
\end{array}
$$

Figure 14: The Newsfeed System Typing.

client and provider, where the first specifies the reception and the second the emission (both infinitely many times) of message `post`. The contribution of each `NewsService` is characterized by type *NewsServiceT* which specifies the output of a single `post` message.

The typings of the four participants individually specify that: the *Client* expects a `Newsfeed` service is available at conversation *NewsSite*; the *NewsSite* publishes a `Newsfeed` service and uses (an infinite number of times) service `NewsService` available at conversation *NewsPortal*; both *BBC* and *CNN* publish `NewsService` in conversation *NewsPortal*. The typing for the whole system (*NewsfeedSystem*) specifies the interactions in services `Newsfeed` and `NewsService` in conversations *NewsSite* and *NewsPortal*, respectively.

4. Progress

In this section, we develop an auxiliary proof system to enforce progress properties on systems. As most traditional deadlock detection methods (e.g., see [14, 23, 25]), we build on the construction of a well-founded ordering on events. In our case, events are message synchronizations occurring under conversations. Thus the ordering must relate pairs (conversation identifier,message label), which allows us to cope with systems with multiple interleaved conversations, and back and forth communications between two or more conversations in the same thread. Since references to conversations can be passed in message

synchronization, the ordering also considers for each message the ordering associated to the conversation which is communicated in the message. These ingredients allow us to check that all events in the continuation of a prefix are of greater rank than the event of the prefix, thus guaranteeing the event dependencies are acyclic.

We motivate our development with an example. Consider the following specification:

$$Amazon \blacktriangleleft [\ \texttt{buy}^{\downarrow}?(product).\texttt{price}^{\downarrow}!(price).eBay \blacktriangleleft [\ \texttt{buy}^{\downarrow}!(product).\texttt{price}^{\downarrow}?(p)\ \ ]\ ]$$

representing an application that is trying to sell some product at *Amazon* and then uses *eBay* to restock the bought item. Consider another application performing a similar task:

$$eBay \blacktriangleleft [\ \texttt{buy}^{\downarrow}?(product).\texttt{price}^{\downarrow}!(price).Amazon \blacktriangleleft [\ \texttt{buy}^{\downarrow}!(product).\texttt{price}^{\downarrow}?(p)\ \ ]\ ]$$

The only difference with respect to the previous process is that this one is working the other way around: selling at *eBay* and restocking at *Amazon*. When considering the system obtained by composing these two processes in parallel we may observe that the system is deadlocked since both processes are waiting to receive a message. Notice, however, that well-defined conversation protocols are followed ($\boldsymbol{\tau}\,\texttt{buy}^{\downarrow}(Tp).\boldsymbol{\tau}\,\texttt{price}^{\downarrow}(Tm)$ in each conversation). For the first process we have that the underlying event ordering is such that event $Amazon.\texttt{buy}$ is smaller than $Amazon.\texttt{price}$, and so forth, which we denote by:

$$Amazon.\texttt{buy} \prec Amazon.\texttt{price} \prec eBay.\texttt{buy} \prec eBay.\texttt{price}$$

Instead for the second process the event ordering is such that

$$eBay.\texttt{buy} \prec eBay.\texttt{price} \prec Amazon.\texttt{buy} \prec Amazon.\texttt{price}$$

Thus, to satisfy the requirements of both processes, an ordering would have to be such that:

$$Amazon.\texttt{buy} \prec \ \ldots\ \prec eBay.\texttt{buy} \prec \ \ldots\ \prec Amazon.\texttt{buy} \prec \ \ldots$$

which is not well-founded. In fact there is no well-founded ordering for the parallel composition of the two processes given above. In this simple example the deadlock is perhaps easy to detect, however when the interleaving is performed over conversations which will only be dynamically instantiated then detecting such deadlocked configurations is harder, as the next example illustrates. Consider a variation of the previous code:

$$\begin{array}{l} eBayReseller \blacktriangleleft [\\ \quad \texttt{sellAt}^{\downarrow}?(x). \\ \qquad x \blacktriangleleft [\ \texttt{buy}^{\downarrow}?(product).\texttt{price}^{\downarrow}!(price).eBay \blacktriangleleft [\ \texttt{buy}^{\downarrow}!(product).\texttt{price}^{\downarrow}?(p)\ \ ]\ ]\ ] \end{array}$$

that specifies a purchase broker *eBayReseller* that performs the previously described functionality: sell in a conversation, restock in another. However, the conversation in which the broker is to sell at is now instructed by a user of the broker, by means of message `sellAt`,

while the restocking is performed at *eBay*. The code for a similar broker that restocks at *Amazon* is:

$$\begin{array}{l} \quad AmazonReseller \blacktriangleleft [\\ \quad\quad \texttt{sellAt}^{\downarrow}?(x). \\ \quad\quad\quad x \blacktriangleleft [\ \texttt{buy}^{\downarrow}?(product).\texttt{price}^{\downarrow}!(price).Amazon \blacktriangleleft [\ \texttt{buy}^{\downarrow}!(product).\texttt{price}^{\downarrow}?(p)\ ]\ ]\ ] \end{array}$$

If we consider the system obtained by composing the two brokers in parallel then the problem is not evident anymore, since it depends on the conversations where the brokers will sell at, namely if placed in the parallel with the process:

$$eBayReseller \blacktriangleleft [\ \texttt{sellAt}^{\downarrow}!(Amazon)\ ]\ \mid\ AmazonReseller \blacktriangleleft [\ \texttt{sellAt}^{\downarrow}!(eBay)\ ]$$

then the system will end up in a deadlocked configuration similar to the one shown before.

The problem in this example can only be detected if we analyze how the conversation references being passed along must be ordered. The event ordering for the *eBayReseller* must be such that $x.\texttt{buy} \prec x.\texttt{price} \prec eBay.\texttt{buy} \prec eBay.\texttt{price}$ so any name instantiation of x must respect this ordering. Likewise for *AmazonReseller* we have the following prescribed ordering: $x.\texttt{buy} \prec x.\texttt{price} \prec Amazon.\texttt{buy} \prec Amazon.\texttt{price}$. Technically, we proceed by attaching such orderings to the events where the conversation references are passed, e.g.,:

$$eBayReseller.\texttt{sellAt}.(x)(x.\texttt{buy} \prec x.\texttt{price} \prec eBay.\texttt{buy} \prec eBay.\texttt{price})$$

and

$$AmazonReseller.\texttt{sellAt}.(x)(x.\texttt{buy} \prec x.\texttt{price} \prec Amazon.\texttt{buy} \prec Amazon.\texttt{price})$$

which then allows us to check if the name that is actually sent in such a message respects (or not) the ordering expected by the process that receives the name. We may thus exclude the resellers system with our technique, since there is no such well-founded ordering for the events in the system: name *Amazon* is sent in event $eBayReseller.\texttt{sellAt}$ where a conversation x is expected such that $x.\texttt{buy} \prec \ldots \prec eBay.\texttt{buy}$, and name *eBay* is sent in event $AmazonReseller.\texttt{sellAt}$ where a conversation x is expected such that $x.\texttt{buy} \prec \ldots \prec Amazon.\texttt{buy}$.

The proof system, depicted in Figure 15, is presented by means of judgments of the form $\Gamma \vdash_{\ell} P$. The judgment $\Gamma \vdash_{\ell} P$ states that the communications of process P follow a well determined order, specified by Γ. In such a judgment we note by Γ an event ordering: a well-founded partial order of events. Events consist of both a pair (name,label) ($(\Lambda \cup \mathcal{V}) \times \mathcal{L}$) and an event ordering abstraction, i.e., a parameterized event ordering, noted $(x)\Gamma$ (where x is a binding occurrence with scope Γ), which represents the ordering of the conversation which is to be communicated in the message. We range over events with $e, e_1, \ldots$ and denote by $n.l.(x)\Gamma$ an event where n is the conversation name, l is the message label and $(x)\Gamma$ is the event ordering abstraction. In $\Gamma \vdash_{\ell} P$, we use ℓ to keep track of the names of the current conversation ($\ell(\downarrow)$) and of the enclosing conversation ($\ell(\uparrow)$); if $\ell = (n, m)$ then $\ell(\uparrow) = n$ and $\ell(\downarrow) = m$. We reserve variables z', z to represent the top level conversation, so initial judgments are of the form $\Gamma \vdash_{(z',z)} P$. We define some operations over event orderings Γ.

$$\frac{\Gamma \vdash_\ell P \quad \Gamma \vdash_\ell Q}{\Gamma \vdash_\ell P \mid Q}(\mathit{Par}) \qquad \frac{}{\Gamma \vdash_\ell \mathbf{0}}(\mathit{Stop})$$

$$\frac{\Gamma \vdash_\ell P}{\Gamma \setminus a \vdash_\ell (\boldsymbol{\nu} a)P}(\mathit{Res}) \quad \frac{\Gamma \vdash_{(\ell(\downarrow),n)} P}{\Gamma \vdash_\ell n \blacktriangleleft [P]}(\mathit{Piece})$$

$$\frac{\Gamma \vdash_\ell P}{\Gamma \vdash_\ell \mathbf{rec}\, \mathcal{X}.P}(\mathit{Rec}) \quad \frac{\mathcal{X} \in \chi_u}{\Gamma \vdash_\ell \mathbf{rec}\, \mathcal{X}.P}(\mathit{RecUnfold}) \quad \frac{}{\Gamma \vdash_\ell \mathcal{X}}(\mathit{RecVar})$$

$$\frac{\forall_{i \in I}((\ell(d).l_i.(y)\Gamma'_i \perp \Gamma) \cup \Gamma'_i\{y \leftarrow x_i\} \vdash_\ell P_i)}{\Gamma \vdash_\ell \Sigma_{i \in I}\ l_i^d?(x_i).P_i}(\mathit{Input})$$

$$\frac{(\ell(d).l.(x)\Gamma' \perp \Gamma) \vdash_\ell P \quad \Gamma'\{x \leftarrow n\} \subseteq (\ell(d).l.(x)\Gamma' \perp \Gamma)}{\Gamma \vdash_\ell l^d!(n).P}(\mathit{Output})$$

$$\frac{\Gamma \cup \{(e_1 \prec e_2) \mid (e_1\{x \leftarrow \ell(\downarrow)\} \prec_\Gamma e_2\{x \leftarrow \ell(\downarrow)\})\} \vdash_\ell P \quad (\ell(\downarrow) \neq z)}{\Gamma \vdash_\ell \mathbf{this}(x).P}(\mathit{This})$$

Figure 15: Proof Rules for Progress.

Definition 4.1. *Given event ordering* Γ *and conversation name* n *we denote by* $\Gamma \setminus n$ *the event ordering obtained from* Γ *by removing all events that have as conversation identifier the name* n*, while keeping the overall ordering, defined as follows:*

$$\Gamma \setminus n \triangleq \{(e_1(m) \prec e_2(o)) \mid (e_1(m) \prec_\Gamma e_2(o)) \wedge n \neq m \wedge n \neq o\}$$

By $e_1 \prec_\Gamma e_2$ we denote that e_1 is smaller than e_2 under Γ, and by $e(n)$ an event of conversation n, i.e., e is of the form $n.l.(x)\Gamma$ for some l and $(x)\Gamma$.

Definition 4.2. *The domain of an event ordering* Γ*, noted* $dom(\Gamma)$*, is the set of events which are related by* Γ*, defined as follows:*

$$dom(\Gamma) \triangleq \{e \mid \exists e'.(e \prec_\Gamma e') \text{ or } (e' \prec_\Gamma e)\}$$

Definition 4.3. *Given event* e *and event ordering* Γ *such that* $e \in dom(\Gamma)$ *we define* $e \perp \Gamma$ *as the subrelation of* Γ *where all events are greater than* e*, as follows:*

$$e \perp \Gamma \triangleq \{(e_1 \prec e_2) \mid (e_1 \prec_\Gamma e_2) \wedge (e \prec_\Gamma e_1)\}$$

We discuss the key proof rules of Figure 15. In rule (*Res*) the event ordering considered in the conclusion is obtained from the one in the premise by removing all events that have as conversation the restricted name a. In rule (*Piece*) the current conversation and enclosing conversation are updated, so that in the premise the current conversation is n and the enclosing conversation is $\ell(\downarrow)$, which is the current conversation in the conclusion. Rule (*Rec*) states a recursive process is well-ordered if the body is well-ordered. Instead in rule (*RecUnfold*) a recursive process that originates from an unfolding is always well-ordered:

we verify the ordering of the recursive process body only once. To simplify presentation, we consider that each time a recursive definition is unfolded then the recursion variable is replaced by a variable from a dedicated set χ_u ($\chi_u \subseteq \chi$), so as to distinguish unfoldings.

Rules (*Input*) and (*Output*) ensure communications originating in the continuations, including the ones in the conversation being received/sent, are of a greater order. In rule (*Input*) the event ordering considered in the premise is such that it contains elements greater than $\ell(d).l_i.(y)\Gamma'_i$, the event associated with the input, enlarged with the event ordering abstraction $(y)\Gamma'$ of the event associated with the input, where the bound y is replaced by the input parameter x_i. Notice that $\ell(d).l_i.(y)\Gamma'_i$ is necessarily in the domain of the event ordering Γ, by definition of $\ell(d).l_i.(y)\Gamma'_i \perp \Gamma$. In rule (*Output*) the event ordering considered in the premise is such that it contains elements greater than $\ell(d).l.(x)\Gamma'$, the event associated to the output. Also the premise states that the event ordering abstraction $(x)\Gamma'$ of the event associated to the output is a subrelation of the event ordering Γ, when the parameter x is replaced by the name to be sent in the output n. Again, notice that $\ell(d).l_i.(x)\Gamma'$ is necessarily in the domain of the event ordering Γ, by definition of $\ell(d).l_i.(x)\Gamma' \perp \Gamma$.

Rule (*This*) ensures interactions in conversation x follow the ordering defined for the current conversation. The event ordering given in the premise is enlarged with events which have as conversation x, following the ordering given for events that have as conversation $\ell(\downarrow)$. Condition $\ell(\downarrow) \neq z$ ensures the **this** is inside a conversation piece (not at toplevel).

We may now present our progress results, starting by Theorem 4.5 which states that orderings are preserved in reductions, and its auxiliary Substitution Lemma.

Lemma 4.4 (Substitution). *Let P be a process and Γ, Γ' event orderings such that $\Gamma \cup \Gamma' \vdash_\ell P$ and $\Gamma'\{x \leftarrow n\} \subseteq \Gamma$. Then $\Gamma \vdash_{\ell\{x \leftarrow n\}} P\{x \leftarrow n\}$.*

Proof. By induction on the structure of P. Essentially, $\Gamma'\{x \leftarrow n\} \subseteq \Gamma$ ensures the ordering prescribed for n in Γ copes with the ordering required for x (see Appendix B). ■

Theorem 4.5 (Preservation of Event Ordering). *Let P be a process and Γ an event ordering such that $\Gamma \vdash_\ell P$, for some ℓ. If $P \to Q$ then $\Gamma \vdash_\ell Q$.*

Proof. By induction on the length of the derivation of the reduction (see Appendix B). ■

In order to characterize the absence of stuck processes, we first need to distinguish finished processes from stuck processes.

Definition 4.6 (Finished Process). *P is a finished process if for any static context $\mathcal{C}$ and process Q such that $P = \mathcal{C}[Q]$ then Q has no immediate output ($\lambda = l^d!(a)$) transitions.*

Finished processes have no reductions and also have no pending requests (outputs), hence are in a stable state, but may have some active inputs (e.g., persistent definitions).

Theorem 4.7 (Progress). *Let P be a well-typed process such that $P :: T$, where $closed(T)$, and Γ an event ordering such that $\Gamma \vdash_{(z',z)} P$. If P is not a finished process then $P \to Q$.*

Proof. The result follows from auxiliary Lemmas (see Appendix B). ∎

Theorem 4.7 ensures that well-typed and well-ordered processes are never stuck on an output that has no matching input. This property entails that services are always available upon request and protocols involving interleaving conversations never get stuck. As for conversation type-checking the procedure to check if $\Gamma \vdash_\ell P$ is decidable, provided bound names are annotated with the orderings.

4.1. Proving Progress in Conversations

In this section we revisit the running example to show they enjoy the progress property. For the *Buyer-Seller-Shipper* example of Section 1.1, which we denote by *BuySystem*, we have that *BuySystem* :: T_{sys} and $closed(T_{sys})$ (see Figure 12).

We consider Γ is such that:

$$\begin{array}{l} \mathit{Seller}.\mathtt{startBuy}.(x_1)\Gamma_1 \prec_\Gamma \mathit{Seller}.\mathtt{askPrice} \prec_\Gamma \\ \qquad \mathit{Seller}.\mathtt{readVal} \prec_\Gamma \mathit{Shipper}.\mathtt{newDelivery}.(x_2)\Gamma_2 \end{array}$$

where Γ_1 describes the ordering of the conversation which is passed in the `startBuy` service instantiation and Γ_2 describes the ordering of the conversation which is passed in the `newDelivery` service instantiation (we omit the associate event ordering of events when it is empty, e.g., in events *Seller*.`askPrice` and *Seller*.`readVal`).

We then have that Γ_2 is such that:

$$x_2.\mathtt{product} \prec_{\Gamma_2} x_2.\mathtt{details}$$

which captures the ordering of the events associated to the *Shipper*, and Γ_1 is such that:

$$\begin{array}{l} x_1.\mathtt{buy} \prec_{\Gamma_1} \mathit{Seller}.\mathtt{askPrice} \prec_{\Gamma_1} \mathit{Seller}.\mathtt{readVal} \prec_{\Gamma_1} x_1.\mathtt{price} \prec_{\Gamma_1} \\ \qquad \mathit{Shipper}.\mathtt{newDelivery}.(x_2)\Gamma_2 \prec_{\Gamma_1} x_1.\mathtt{product} \prec_{\Gamma_1} x_1.\mathtt{details} \end{array}$$

which captures the ordering of the events associated to the *Seller-Shipper* subsystem (recall that *Seller* dynamically invites *Shipper* to join the conversation). When analyzing within the scope of the restricted name corresponding to the service conversation, which we identify with name c, Γ is extended to Γ' such that:

$$\begin{array}{l} \mathit{Seller}.\mathtt{startBuy}.(x_1)\Gamma_1 \prec_{\Gamma'} c.\mathtt{buy} \prec_{\Gamma'} \mathit{Seller}.\mathtt{askPrice} \prec_{\Gamma'} \mathit{Seller}.\mathtt{readVal} \prec_{\Gamma'} \\ \qquad c.\mathtt{price} \prec_{\Gamma'} \mathit{Shipper}.\mathtt{newDelivery}.(x_2)\Gamma_2 \prec_{\Gamma'} c.\mathtt{product} \prec_{\Gamma'} c.\mathtt{details} \end{array}$$

Notice such ordering corresponds to the vertical timeline of the message sequence chart shown in Figure 1. We can then state $\Gamma \vdash_\ell$ *BuySystem* which combined with *BuySystem* :: T_{sys} and $closed(T_{sys})$ guarantees, considering the results of Theorem 4.5 and Theorem 4.7, that the process *BuySystem* reduces endlessly, or until it is a finished process.

Notice our progress results apply to systems where conversations can be interleaved, including delegated conversations. For instance, consider the *Seller* code which interleaves the received service conversation with the enclosing conversation (to access the price database) and with conversation *Shipper* (to ask *Shipper* to join in the ongoing conversation).

$$
\begin{array}{rcll}
P, Q, R & ::= & \mathbf{0} & \text{(Inaction)} \\
& | & P \mid Q & \text{(Parallel Composition)} \\
& | & (\boldsymbol{\nu} a)P & \text{(Name Restriction)} \\
& | & \textsf{rec}\, \mathcal{X}.P & \text{(Recursion)} \\
& | & \mathcal{X} & \text{(Variable)} \\
& | & \Sigma_{i \in I}\ \alpha_i.P_i & \text{(Prefix Guarded Choice)} \\
\\
\alpha & ::= & n \cdot l!(m) & \text{(Output)} \\
& | & n \cdot l?(x) & \text{(Input)}
\end{array}
$$

Figure 16: The Labeled π-calculus (π_{lab}-calculus) Syntax.

5. Conversation Types in a Labeled π-calculus

Our general framework of conversations was developed having in mind our view of service oriented computing as modeled by the CC. In this section, we describe how the key ideas behind conversation types can be reproduced in a more canonical and simpler model, an elementary labeled π-calculus, thus showing how the generalization of binary session types to conversations types are not specific to the core CC, in the same way as session types are not specific to the π-calculus with session constructs (cf., [11]), and may conceivably be imposed in a π-calculus with labels. We start by presenting the labeled π-calculus (π_{lab}-calculus), which is an extension of the π-calculus obtained by indexing channel communication with labels.

The syntax of the π_{lab}-calculus is shown in Figure 16. We reuse names, variables, labels and process variables from the syntax of core CC—Definition 2.1. Notice the only difference of the π_{lab}-calculus with respect to the π-calculus is the label l suffix of channel n communications.

We briefly describe the operational semantics of the π_{lab}-calculus. The operational semantics of the π_{lab}-calculus is defined by means of a labeled transition system. A transition $P \xrightarrow{\lambda} Q$ states that process P may evolve to process Q by performing the action represented by the transition label λ. Transition labels are defined as follows:

$$\lambda \ ::= \ \tau \mid c \cdot l!(a) \mid c \cdot l?(a) \mid (\nu a)\lambda$$

Transition labels represent internal interaction (τ), and interactions with the external environment, either output ($c \cdot l!(a)$) or input ($c \cdot l?(a)$) , and may carry a bound name ($(\boldsymbol{\nu} a)\lambda$).

The transition relation ($P \xrightarrow{\lambda} Q$) is the least relation that satisfies the rules of Figure 3, where rules (Out) and (In) are replaced by the following:

$$c \cdot l!(a).P \xrightarrow{c \cdot l!(a)} P \ (Out) \qquad c \cdot l?(x).P \xrightarrow{c \cdot l?(a)} P\{x \leftarrow a\} \ (In)$$

The main difference with respect to core CC is that communications in the π_{lab}-calculus are located a priori, unlike in core CC where communications' location depends on contextual

information. Notice that π_{lab}-calculus processes can be translated into core CC processes by encoding the inputs and outputs through the combination of conversation contexts and labeled messages:

$$[\![n \cdot l!(m).P]\!] \triangleq n \blacktriangleleft [\, l!(m).[\![P]\!] \,] \qquad\qquad [\![n \cdot l?(x).P]\!] \triangleq n \blacktriangleleft [\, l?(x).[\![P]\!] \,]$$

However, the typing analysis presented in Section 3 is (as usual) driven by syntactic information, and uses information about the current (contextually determined) conversation which cannot be represented in the π_{lab}-calculus, so some care is needed in the application of conversation type analysis of the π_{lab}-calculus. On the other hand the progress proof system presented in Section 4 can be directly applied to the π_{lab}-calculus. We discuss such applications in the following sections.

5.1. Conversation Types for the π_{lab}-calculus

In this section we show how the conversation types presented in Section 3 can be used to type π_{lab} systems. We consider the direct restriction of our type language and operators to messages defined exclusively with $\downarrow$ direction (which is the same as no direction at all).

We characterize the interactions that a π_{lab}-calculus process has in a determined channel n by means of a located type $n : [B]$, where B is the behavioral type that collects the labels and polarities of the interactions in channel n. Since π_{lab}-calculus process communications are originally located, the typing judgment for a π_{lab}-calculus process P is of the form $P :: L$, unlike the judgment for core CC processes which uses process types (the unlocated part) to specify the interactions in the current conversation. Figure 17 shows the typing rules for π_{lab}-calculus processes.

The main differences with respect to the rules shown in Figure 9 are in rules (*Output*) and (*Input*). While for the core CC we specify the choice or branch types (respectively) that characterize the action prefix at the level of the current conversation (in the conclusion of the rule), for the π_{lab}-calculus we must specify the choice or branch types in the corresponding located type. To that end, in the premise of rule (*Output*), we take a partial view of the corresponding located type, through the merge $L \bowtie n : [B]$, and use this view as the continuation of the message interaction. In the conclusion we merge the resulting located choice type $n : [\oplus_{i \in I}\{M_i.B_i\}]$ back to the located type L. Likewise for rule (*Input*).

Intuitively, rules (*Input*) and (*Output*) for π_{lab}-calculus processes are a combination of the respective rules for core CC processes together with rule (*Piece*), which is where behaviors get located in the core CC type system.

We state our main soundness results for the labeled π-calculus.

Theorem 5.1 (π_{lab}-calculus Subject Reduction). *Let P be a π_{lab}-calculus process and L a type such that $P :: L$. If $P \to Q$ then there is L' such that $L \to L'$ and $Q :: L'$.*

Proof. Follows the lines of the proof of Theorem 3.20. ∎

The definition of error π_{lab}-calculus processes follows the lines of Definition 3.22: processes that exhibit competing linear messages.

$$\frac{P :: L_1 \quad Q :: L_2}{P \mid Q :: L_1 \bowtie L_2}(Par) \qquad \frac{}{\mathbf{0} :: \mathbf{0}}(Stop)$$

$$\frac{P :: L \mid a : [B] \quad (closed(B), a \notin dom(L))}{(\boldsymbol{\nu} a)P :: L}(Res)$$

$$\frac{P :: L_M}{\texttt{rec}\,\mathcal{X}.P :: \star L_M}(Rec) \qquad \frac{}{\mathcal{X} :: \mathbf{0}}(RecVar)$$

$$\frac{\forall_{i \in I}\,(P_i :: (L \bowtie n : [B_i]) \mid x_i : C_i \quad (x_i \notin dom(L)))}{\Sigma_{i \in I}\; n \cdot l_i?(x_i).P_i :: L \bowtie n : [\&_{i \in I}\{?\, l_i(C_i).B_i\}]}(Input)$$

$$\frac{P :: L \bowtie n : [B] \quad (\exists j \in I.M_j.B_j = \,!\, l(C).B)}{n \cdot l!(m).P :: (L \bowtie n : [\oplus_{i \in I}\{M_i.B_i\}]) \bowtie (m : C)}(Output)$$

$$\frac{P :: L_1 \quad L_1 <: L_2}{P :: L_2}(Sub)$$

Figure 17: π_{lab}-calculus Typing Rules.

Proposition 5.2 (π_{lab}-calculus Error Freeness). *Let P be a π_{lab}-calculus process. If there is type L such that $P :: L$ then P is not an error process.*

Proof. Follows the lines of the proof of Proposition 3.23. ∎

Corollary 5.3 (π_{lab}-calculus Type Safety). *Let P be a π_{lab}-calculus process such that $P :: L$ for some L. If there is Q such that $P \xrightarrow{*} Q$, then Q is not an error process.*

Proof. Immediate from Theorem 5.1 and Proposition 5.2. ∎

Corollary 5.4 (Conversation Fidelity). *Let P be a π_{lab}-calculus process such that $P :: L$ for some L. Then all conversations in P follow the protocols prescribed by L.*

5.2. Progress Analysis for the π_{lab}-calculus

In this section we show how the progress proof system presented in Section 4 applies to the π_{lab}-calculus. Event orderings and their operations are exactly the same as defined in Section 4 in this setting. The only difference is that given communications are always located a priori we no longer need to keep track of the current location information, so the judgment now takes the form of $\Gamma \vdash P$. The rules of the progress proof system are shown in Figure 18.

We state our progress results for the labeled π-calculus.

Theorem 5.5 (π_{lab}-calculus Preservation of Event Ordering). *Let P be a π_{lab}-calculus process and Γ an event ordering such that $\Gamma \vdash P$. If $P \to Q$ then $\Gamma \vdash Q$.*

$$\frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \mid Q}(Par) \qquad \frac{}{\Gamma \vdash \mathbf{0}}(Stop) \qquad \frac{\Gamma \vdash P}{\Gamma \setminus a \vdash (\boldsymbol{\nu} a)P}(Res)$$

$$\frac{\Gamma \vdash P}{\Gamma \vdash \mathbf{rec}\,\mathcal{X}.P}(Rec) \qquad \frac{\mathcal{X} \in \chi_u}{\Gamma \vdash \mathbf{rec}\,\mathcal{X}.P}(RecUnfold) \qquad \frac{}{\Gamma \vdash \mathcal{X}}(RecVar)$$

$$\frac{\forall_{i\in I}((n.l_i.(y)\Gamma'_i \perp \Gamma) \cup \Gamma'_i\{y \leftarrow x_i\} \vdash P_i)}{\Gamma \vdash \Sigma_{i\in I}\ n \cdot l_i?(x_i).P_i}(Input)$$

$$\frac{(n.l.(x)\Gamma' \perp \Gamma) \vdash P \quad \Gamma'\{x \leftarrow m\} \subseteq (n.l.(x)\Gamma' \perp \Gamma)}{\Gamma \vdash n \cdot l!(m).P}(Output)$$

Figure 18: π_{lab}-calculus Proof Rules for Progress.

Proof. Follows directly from Theorem 4.5. ■

The definition of finished π_{lab}-calculus processes is a direct extension of Definition 4.6: processes that have no immediate output ($\lambda = c\ l!(a)$) transitions.

Theorem 5.6 (π_{lab}-calculus Progress). *Let P be a well-typed π_{lab}-calculus process such that $P :: L$, where closed(L), and Γ an event ordering such that $\Gamma \vdash P$. If P is not a finished process then $P \rightarrow Q$.*

Proof. Follows the lines of the proof of Theorem 4.7. ■

6. Related Work

Behavioral Type Systems As most behavioral type systems (see [13, 20]), we describe a conversation behavior by some kind of abstract process. However, fundamental ideas behind the conversation type structure, in particular the composition / decomposition of behaviors via merge, as captured, e.g., in the typing rule for $P \mid Q$, and used to model delegation of conversation fragments, have not been explored before.

Binary Sessions The notion of conversation originates in that of session (introduced in [17, 18]). Sessions are a medium for two-party interaction, where session participants access the session through a session endpoint. On the other hand conversations are also a single medium but for multiparty interaction, where any of the conversation participants accesses the conversation through a conversation endpoint (pieces). Session channels support single-threaded interaction protocols between the two session participants. Conversation contexts, on the other hand, support concurrent interaction protocols between multiple participants. Sessions always have two endpoints, created at session initialization. Participants can delegate their participation in a session, but the delegation is full as the delegating party loses access to the session. Conversations also initially have two endpoints. However the number of endpoints may increase (decrease) as participants join in on (leave) ongoing conversations. Participants can ask a party to join in on a conversation and not lose access to it (partial delegation). Since there are only two session participants, session types

may describe the entire protocol by describing the behavior of just one of the participants (the type of the other participant is dual). Conversation types, on the other hand, describe the interactions between multiple parties so they specify the entire conversation protocol (a choreography description) that decomposes in the types of the several participants (e.g., $B_t = B_{bu} \bowtie B_{se} \bowtie B_{sh}$).

Multiparty Sessions The goals of the works [2, 3, 19] are similar to ours. To support multiparty interaction, [19] considers multiple session channels, while [2] considers a multiple indexed session channel, both resorting to multiple communication pathways. We follow an essentially different approach, by letting a single medium of interaction support concurrent multiparty interaction via labeled messages. In [2, 19] sessions are established simultaneously between several parties through a multicast session request. As in binary sessions, session delegation is full so the number of initial participants is kept invariant, unlike in conversations where parties can keep joining in. The approach of [2, 19] builds on two-level descriptions of service collaborations (global and local types), first introduced in a theory of endpoint projection [10]. The global types mention the identities of the communicating partners, being the types of the individual participants projections of the global type with respect to these annotations. Our merge operation $\bowtie$ is inspired by the idea of projection [10], but we follow a different approach where "global" and "local" types are treated at the same level in the type language and types do not explicitly mention the participants identities, so that each given protocol may be realized by different sets of participants, provided that the composition of the types of the several participants produce (via $\bowtie$) the appropriate invariant. Our approach thus supports conversations with dynamically changing number of partners, ensuring a higher degree of loose-coupling. We do not see how this could be encoded in the approach of [19]. On the other hand, we believe that core CC with conversation types can express the same kind of systems as [19].

Progress in Session Types There are a number of progress studies for binary sessions (e.g., [1, 6, 14]), and for multiparty sessions [2, 19]. The techniques of [2, 14] are nearer to ours as orderings on channels are imposed to guarantee the absence of cyclic dependencies. However they disallow processes that get back to interact in a session after interacting in another, and exclude interleaving on received sessions, while we allow processes that re-interact in a conversation and interleave received conversations.

7. Concluding Remarks

We have presented a core typed model for expressing and analyzing service and communication based systems, building on the notions of conversation, conversation context, and context-dependent communication. We believe that, operationally, the core CC can be seen as a specialized idiom of the π-calculus [27], if one considers π extended with labeled channels or pattern matching. However, for the purpose of studying communication disciplines for service-oriented computing and their typings, it is much more convenient to adopt a primitive conversation context construct, for it allows the conversation identity to be kept implicit until needed.

Conversation types elucidate the intended dynamic structure of conversations, in particular how freshly instantiated conversations may dynamically engage and dismiss participants, modeling in a fairly abstract way, the much lower level correlation mechanisms available in Web-Services technology. Conversation types also describe the information and control flow of general service-based collaborations, in particular they may describe the behavior of orchestrations and choreographies. We have established subject reduction and type safety theorems, which entail that well-typed systems follow the defined protocols. We also have studied a progress property, proving that well-ordered systems never get stuck, even when participants are engaged in multiple interleaved conversations, as is often the case in applications. Conversation types extend the notion of binary session types to multiple participants, but discipline their communication by exploiting distinctions between labeled messages in a single shared communication medium, rather than by introducing multiple or indexed communication channels, where interaction in each one is captured by a more traditional session type, as, e.g., [19]. This approach allows us to unify the notions of global type and local type, and type highly dynamic scenarios of multiparty concurrent conversations not covered by other approaches. On the other hand, being more abstract and uniform, our type system does not explicitly keep track of participant identities. It would be interesting to investigate to what extent both approaches could be conciliated, for instance, by specializing our approach so as to consider extra constraints on projections on types and merges, restricting particular message exchanges to some roles.

Acknowledgments We acknowledge CITI, IP Sensoria and CMU-PT. We thank Mariangiola Dezani-Ciancaglini and Nobuko Yoshida for insightful discussions, and the anonymous referees for their detailed remarks.

References

[1] L. Acciai and M. Boreale. A Type System for Client Progress in a Service-Oriented Calculus. In P. Degano, R. De Nicola, and J. Meseguer, editors, *Concurrency, Graphs and Models, Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday*, volume 5065 of *Lecture Notes in Computer Science*, pages 642–658. Springer-Verlag, 2008.

[2] L. Bettini, M. Coppo, L. D'Antoni, M. De Luca, M. Dezani-Ciancaglini, and N. Yoshida. Global Progress in Dynamically Interleaved Multiparty Sessions. In F. van Breugel and M. Chechik, editors, *CONCUR 2008, 19th International Conference on Concurrency Theory, Proceedings*, volume 5201 of *Lecture Notes in Computer Science*, pages 418–433. Springer-Verlag, 2008.

[3] E. Bonelli and A. Compagnoni. Multipoint Session Types for a Distributed Calculus. In G. Barthe and C. Fournet, editors, *Trustworthy Global Computing, Third Symposium, TGC 2007, Revised Selected Papers*, volume 4912 of *Lecture Notes in Computer Science*, pages 240–256. Springer-Verlag, 2008.

[4] M. Bravetti, I. Lanese, and G. Zavattaro. Contract-Driven Implementation of Choreographies. In C. Kaklamanis and F. Nielson, editors, *Trustworthy Global Computing, 4th Symposium, TGC 2008, Revised Selected Papers*, volume 5474 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, 2009.

[5] M. Bravetti and G. Zavattaro. A Foundational Theory of Contracts for Multi-party Service Composition. *Fundamenta Informaticae*, 89(4):451–478, 2008.

[6] R. Bruni and L. G. Mezzina. Types and Deadlock Freedom in a Calculus of Services, Sessions and Pipelines. In J. Meseguer and G. Rosu, editors, *Algebraic Methodology and Software Technology,*

12th International Conference, AMAST 2008, Proceedings, volume 5140 of *Lecture Notes in Computer Science*, pages 100–115. Springer-Verlag, 2008.

[7] L. Caires. Spatial-Behavioral Types for Concurrency and Resource Control in Distributed Systems. *Theoretical Computer Science*, 402(2-3):120–141, 2008.

[8] L. Caires and H. T. Vieira. Conversation Types. In G. Castagna, editor, *Programming Languages and Systems, 18th European Symposium on Programming, ESOP 2009, Proceedings*, volume 5502 of *Lecture Notes in Computer Science*, pages 285–300. Springer-Verlag, 2009.

[9] L. Caires and H. T. Vieira. Conversation Types. Technical Report UNL-DI-01-09, Departamento de Informática, Universidade Nova de Lisboa, 2009.

[10] M. Carbone, K. Honda, and N. Yoshida. Structured Communication-Centred Programming for Web Services. In R. De Nicola, editor, *Programming Languages and Systems, 16th European Symposium on Programming, ESOP 2007, Proceedings*, volume 4421 of *Lecture Notes in Computer Science*, pages 2–17. Springer-Verlag, 2007.

[11] G. Castagna, M. Dezani-Ciancaglini, E. Giachino, and L. Padovani. Foundations of Session Types. In A. Porto and F. J. López-Fraguas, editors, *PPDP 2009, 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, Proceedings*, pages 219–230. ACM, 2009.

[12] G. Castagna, N. Gesbert, and L. Padovani. A Theory of Contracts for Web Services. *ACM Transactions on Programming Languages and Systems*, 31(5), 2009.

[13] S. Chaki, S. K. Rajamani, and J. Rehof. Types as models: Model Checking Message-Passing Programs. In *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2002*, pages 45–57. ACM, 2002.

[14] M. Dezani-Ciancaglini, U. de' Liguoro, and N. Yoshida. On Progress for Structured Communications. In G. Barthe and C. Fournet, editors, *Trustworthy Global Computing, Third Symposium, TGC 2007, Revised Selected Papers*, volume 4912 of *Lecture Notes in Computer Science*, pages 257–275. Springer-Verlag, 2008.

[15] S. Gay and M. Hole. Subtyping for Session Types in the Pi Calculus. *Acta Informatica*, 42(2-3):191–225, 2005.

[16] E. Giachino, M. Sackman, S. Drossopoulou, and S. Eisenbach. Softly Safely Spoken: Role Playing for Session Types. In *PLACES 2009, 2nd International Workshop on Programming Language Approaches to Concurrency and Communication-Centric Software, Proceedings*. To appear.

[17] K. Honda. Types for Dyadic Interaction. In E. Best, editor, *CONCUR 1993, 4th International Conference on Concurrency Theory, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer-Verlag, 1993.

[18] K. Honda, V. T. Vasconcelos, and M. Kubo. Language Primitives and Type Discipline for Structured Communication-Based Programming. In C. Hankin, editor, *Programming Languages and Systems, 7th European Symposium on Programming, ESOP 1998, Proceedings*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer-Verlag, 1998.

[19] K. Honda, N. Yoshida, and M. Carbone. Multiparty Asynchronous Session Types. In G. C. Necula and P. Wadler, editors, *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008*, pages 273–284. ACM, 2008.

[20] A. Igarashi and N. Kobayashi. A Generic Type System for the Pi-Calculus. *Theoretical Computer Science*, 311(1-3):121–163, 2004.

[21] N. Kobayashi, B. C. Pierce, and D. N. Turner. Linearity and the Pi-Calculus. In *Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 1996*, pages 358–371. ACM, 1996.

[22] I. Lanese, C. Guidi, F. Montesi, and G. Zavattaro. Bridging the Gap between Interaction- and Process-Oriented Choreographies. In A. Cerone and S. Gruner, editors, *SEFM 2008, Sixth IEEE International Conference on Software Engineering and Formal Methods, Proceedings*, pages 323–332. IEEE Computer Society, 2008.

[23] N. Lynch. Fast Allocation of Nearby Resources in a Distributed System. In *Conference Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing, STOC 1980*, pages 70–81. ACM, 1980.

[24] R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, Part I + II. *Information and Computation*, 100(1):1–77, 1992.
[25] N. Kobayashi. A New Type System for Deadlock-Free Processes. In C. Baier and H. Hermanns, editors, *CONCUR 2006, 17th International Conference on Concurrency Theory, Proceedings*, volume 4137 of *Lecture Notes in Computer Science*, pages 233–247. Springer-Verlag, 2006.
[26] B. Pierce. *Types and Programming Languages.* MIT Press, 2002.
[27] D. Sangiorgi and D. Walker. *The π-calculus: A Theory of Mobile Processes.* Cambridge University Press, 2001.
[28] H. T. Vieira, L. Caires, and J. C. Seco. The Conversation Calculus: A Model of Service-Oriented Computation. In S. Drossopoulou, editor, *Programming Languages and Systems, 17th European Symposium on Programming, ESOP 2008, Proceedings*, volume 4960 of *Lecture Notes in Computer Science*, pages 269–283. Springer-Verlag, 2008.

A. Proofs for Results of Section 3

We show the main cases and refer the interested reader to [9] for more detailed proofs.

Proof of Lemma 3.12

(2) If $B' = B_1 \bowtie B_2$ *and* $B = B' \bowtie B_3$ *then there is* B'' *such that* $B'' = B_2 \bowtie B_3$ *and* $B = B_1 \bowtie B''$*.*

Proof. By induction on the derivation of $B' = B_1 \bowtie B_2$ and $B = B' \bowtie B_3$. We show the case when $B' = B_1 \bowtie B_2$ is derived using rule (*Plain-l*) and $B = B' \bowtie B_3$ is derived using rule (*Shuffle-l*), hence $B_1 \equiv\ !\, l(C).B_a$ and $B_2 \equiv\ ?\, l(C).B_b$. We have:

$$\tau\, l(C).B' \mid B'' = \tau\, l(C).(B_a \bowtie B_b) \bowtie B_3 \tag{1}$$

derived from:

$$B' \mid B'' \equiv (B_a \bowtie B_b) \bowtie B_3 \tag{2}$$

and $\tau\, l(C) \# B_3$, $\tau\, l(C) \# B''$ and $\mathcal{I}(B') \subseteq \mathcal{I}(B_a \bowtie B_b)$ and $\mathcal{I}(B'') \subseteq \mathcal{I}(B_3)$. We also have:

$$\tau\, l(C).(B_a \bowtie B_b) =\ !\, l(C).B_a \bowtie\ ?\, l(C).B_b \tag{3}$$

We intend to prove: $\tau\, l(C).B' \mid B'' =\ !\, l(C).B_a \bowtie (?\, l(C).B_b \bowtie B_3)$. By induction hypothesis on (2) we have:

$$B' \mid B'' \equiv B_a \bowtie (B_b \bowtie B_3) \tag{4}$$

From (4) and $\tau\, l(C) \# B_3$ we have that there is B_c, B_d such that:

$$?\, l(C).B_c \mid B_d =\ ?\, l(C).B_b \bowtie B_3 \tag{5}$$

derived from:

$$B_c \mid B_d \equiv B_b \bowtie B_3 \tag{6}$$

and $?\, l(C) \# B_d$ and $\mathcal{I}(B_c) \subseteq \mathcal{I}(B_b)$ and $\mathcal{I}(B_d) \subseteq \mathcal{I}(B_3)$. We then derive ((*MsgPar-r*)):

$$(!\, l(C).B_a \bowtie\ ?\, l(C).B_c) \bowtie B_d =\ !\, l(C).B_a \bowtie\ ?\, l(C).B_c \mid B_d \tag{7}$$

and then by rule (*Plain-l*) we derive:

$$\tau\, l(C).(B_a \bowtie B_c) \bowtie B_d = (!\, l(C).B_a \bowtie ?\, l(C).B_c) \bowtie B_d \tag{8}$$

We then have, by rule (*Shuffle-l*):

$$\tau\, l(C).B_e \mid B_f = \tau\, l(C).(B_a \bowtie B_c) \bowtie B_d \tag{9}$$

derived from:

$$B_e \mid B_f \equiv (B_a \bowtie B_c) \bowtie B_d \tag{10}$$

and $\tau\, l(C) \# B_f$ and $\mathcal{I}(B_e) \subseteq \mathcal{I}(B_a \bowtie B_c)$ and $\mathcal{I}(B_f) \subseteq \mathcal{I}(B_d)$. By induction hypothesis on (10) we conclude:

$$(B_a \bowtie B_c) \bowtie B_d \equiv B_a \bowtie (B_c \bowtie B_d) \tag{11}$$

From (6) we have that B_c and B_d are apart, hence:

$$B_a \bowtie (B_c \bowtie B_d) \equiv B_a \bowtie (B_c \mid B_d) \tag{12}$$

Also from (6) we conclude:

$$B_a \bowtie (B_c \mid B_d) \equiv B_a \bowtie (B_b \bowtie B_3) \tag{13}$$

From (13), (12), (11) and (10) we conclude:

$$B_e \mid B_f \equiv B_a \bowtie (B_b \bowtie B_3) \tag{14}$$

From (14) and (4) we conclude:

$$B_e \mid B_f \equiv B' \mid B'' \tag{15}$$

From $\mathcal{I}(B_f) \subseteq \mathcal{I}(B_d) \subseteq \mathcal{I}(B_3)$ and $\mathcal{I}(B'') \subseteq B_3$ we have $B_e \equiv B'$ and $B_f \equiv B''$, from which we conclude—rule (*Shuffle-l*):

$$\tau\, l(C).B' \mid B'' = \tau\, l(C).(B_a \bowtie B_c) \bowtie B_d \tag{16}$$

which completes the proof. ∎

Proof of Lemma 3.19 (Substitution)

Let P be a well-typed process such that $P :: T \mid x : C$, for $x \notin dom(T)$ and types T, C. If there is type T' such that $T' = T \bowtie a : C$ then $P\{x \leftarrow a\} :: T'$.

Proof. By induction on the length of the derivation of $P :: T \mid x : C$. We show the case of rule (*Piece*), when P of the form $x \blacktriangleleft [Q]$.

(*Case* (*Piece*))

We have:

$$x \blacktriangleleft [Q] :: T \mid x : C \tag{1}$$

where $x \notin dom(T)$. Since (1) is a conclusion of rule (*Piece*) we have that:

$$T \mid x : C \equiv (L \bowtie x : [\downarrow B]) \mid loc(\uparrow B)\ \ (i) \qquad \text{and} \qquad Q :: L \mid B\ \ (ii) \tag{2}$$

In order to apply the induction hypothesis we first characterize types L, C and T. We separate L into two parts L' and $x : C'$ such that L' does not mention x (i.e., $x \notin dom(L')$):

$$L \equiv L' \mid x : C' \tag{3}$$

From (2)(i) we have that $x : C$ is the result of the merge of $x : [\downarrow B]$ and the type that L specifies for x (which we identify in (3) as $x : C'$), hence:

$$x : C = x : C' \bowtie x : [\downarrow B] \tag{4}$$

Also, since L' does not mention x we have that $(L' \mid x : C') \bowtie x : [\downarrow B]$ yields the same type as $L' \mid (x : C' \bowtie x : [\downarrow B])$, hence from (2)($i$) we have that T is such that:

$$T \equiv L' \mid loc(\uparrow B) \tag{5}$$

We now assume the hypothesis in the statement of the Lemma: there is type T' such that:

$$T' = T \bowtie a : C \tag{6}$$

Since L' is a part of T (5) and C' is a partial view of C (4), from (6) we conclude there is type T'' such that $T'' = L' \bowtie a : C'$ and hence:

$$T'' \mid B = (L' \mid B) \bowtie a : C' \tag{7}$$

We rewrite (2)(ii) considering (3), using the subsumption rule, and obtain:

$$Q :: L' \mid B \mid x : C' \tag{8}$$

By induction hypothesis on (8) and (7) we conclude $Q\{x \leftarrow a\} :: (L' \mid B) \bowtie a : C'$, from which we obtain:

$$Q\{x \leftarrow a\} :: (L' \bowtie a : C') \mid B \tag{9}$$

From (6), considering L' is a part of T (5) and separating C in the partial views given by (4) we obtain there is type T''' such that:

$$T''' = L' \bowtie (a : C' \bowtie a : [\downarrow B]) \tag{10}$$

From (9) and (10) and considering rule (*Piece*) we derive:

$$a \blacktriangleleft [(Q\{x \leftarrow a\})] :: ((L' \bowtie a : C') \bowtie a : [\downarrow B]) \mid loc(\uparrow B) \tag{11}$$

from which we conclude:

$$a \blacktriangleleft [(Q\{x \leftarrow a\})] :: (L' \mid loc(\uparrow B)) \bowtie (a : C' \bowtie a : [\downarrow B]) \tag{12}$$

From (12), considering (5) and (4), we conclude:

$$a \blacktriangleleft [(Q\{x \leftarrow a\})] :: T \bowtie a : C \tag{13}$$

which completes the proof for this case. ∎

We state some auxiliary results to the proof of Theorem 3.20.

Lemma A.1. *Let process P be such that $P :: T$. If $P \xrightarrow{l^d?(a)} Q$ then there are $T', C, B, \widetilde{B}$ such that either $T \equiv T' \mid \&\{?\, l^d(C).B; \widetilde{B}\}$ or $T \equiv T' \mid B$ and $\tau\, l^d(C) \in Msg(B)$. Furthermore if $T \equiv T' \mid \&\{?\, l^d(C).B; \widetilde{B}\}$ and there is $T'' = (T' \mid B) \bowtie a : C$ then $Q :: T''$.*

Proof. By induction on the derivation of the transition $P \xrightarrow{l^d?(a)} Q$. We show the cases when the transition results from a input summation and from a parallel composition.

(*Case* $\Sigma_{i\in I}\, l_i^d?(x_i).P_i \xrightarrow{l_j^d?(a)} P_j\{x_j \leftarrow a\}$)
We have that:

$$\Sigma_{i\in I}\, l_i^d?(x_i).P_i :: T \qquad \text{and} \qquad \Sigma_{i\in I}\, l_i^d?(x_i).P_i \xrightarrow{l_j^d?(a)} P_j\{x_j \leftarrow a\} \tag{1}$$

From (1) and considering rule (*Input*) we have there is $L, C_j, B_j, \widetilde{B}$ such that:

$$L \mid \&\{?\, l_j^d(C_j).B_j; \widetilde{B}\} <: T \tag{2}$$

and:

$$\Sigma_{i\in I}\, l_i^d?(x_i).P_i :: L \mid \&\{?\, l_j^d(C_j).B_j; \widetilde{B}\} \qquad \text{and} \qquad P_j :: L \mid B_j \mid x_j : C_j \tag{3}$$

From (2) we conclude there are $T', B', B'_j, \widetilde{B}'$ such that:

$$T \equiv T' \mid B' \mid \&\{?\, l_j^d(C_j).B'_j; \widetilde{B}'\} \tag{4}$$

and:

$$L <: T' \qquad \text{and} \qquad \&\{?\, l_j^d(C_j).B_j; \widetilde{B}\} <: B' \mid \&\{?\, l_j^d(C_j).B'_j; \widetilde{B}'\} \tag{5}$$

Let us now consider there is T'' such that:

$$T'' \equiv (T' \mid B' \mid B'_j) \bowtie a : C_j \tag{6}$$

Since $L <: T'$ from (6) we directly have that there is L' such that

$$L' \mid B_j \equiv (L \mid B_j) \bowtie a : C_j \tag{7}$$

Considering Lemma 3.19 we then have:

$$P_j\{x_j \leftarrow a\} :: (L \mid B_j) \bowtie a : C_j \tag{8}$$

and hence:

$$P_j\{x_j \leftarrow a\} :: (T' \mid B' \mid B'_j) \bowtie a : C_j \tag{9}$$

which completes the proof for this case.

(*Case* $P' \mid R \xrightarrow{l^d?(a)} Q' \mid R$)
We have that:

$$P' \mid R :: T \qquad \text{and} \qquad P' \mid R \xrightarrow{l^d?(a)} Q' \mid R \tag{10}$$

where the transition is derived from:

$$P' \xrightarrow{l^d?(a)} Q' \tag{11}$$

Considering (10) and rule (Par) we have that there is T_1, T_2 such that $T_1 \bowtie T_2 <: T$ and:

$$P' \mid R :: T_1 \bowtie T_2 \qquad \text{and} \qquad P' :: T_1 \qquad \text{and} \qquad R :: T_2 \tag{12}$$

By induction hypothesis on (11) and (12) we have that there is $T', C, B, \widetilde{B}$ such that:

$$T_1 \equiv T' \mid \&\{?\, l^d(C).B; \widetilde{B}\} \tag{13}$$

or:

$$T_1 \equiv T' \mid B \qquad \text{and} \qquad \tau\, l^d(C) \in \mathit{Msg}(B) \tag{14}$$

In case of (13), from $T_1 \bowtie T_2 <: T$ we conclude there is T'' such that either:

$$T \equiv T'' \mid \&\{?\, l^d(C).B; \widetilde{B}\} \tag{15}$$

or:

$$T \equiv T'' \mid B'' \qquad \text{and} \qquad \tau\, l^d(C) \in \mathit{Msg}(B'') \tag{16}$$

In case of (14), from $T_1 \bowtie T_2 <: T$ we directly have that $T \equiv T'' \mid B''$ where $\tau\, l^d(C) \in \mathit{Msg}(B'')$. Let us now consider (15) and that there is T_3 such that:

$$T_3 \equiv (T'' \mid B) \bowtie a : C \tag{17}$$

From (17), (15) and $T_1 \bowtie T_2 <: T$ we have that there is $(T' \mid B) \bowtie a : C$, and thus by induction hypothesis we conclude:

$$Q' :: (T' \mid B) \bowtie a : C \tag{18}$$

From (18) and (12) we derive:

$$Q' \mid R :: ((T' \mid B) \bowtie a : C) \bowtie T_2 \tag{19}$$

From (19), (15), (13) and $T <: T_1 \bowtie T_2$ we conclude:

$$Q' \mid R :: (T'' \mid B) \bowtie a : C \tag{20}$$

which completes the proof for this case. ∎

Lemma A.2. *Let P be a process such that $P :: T$. If $P \xrightarrow{c\ l^{\downarrow}?(a)} Q$ then there are $T', C, B, \widetilde{B}$ s.t. $T \equiv T' \mid c : [\&\{?\ l^{\downarrow}(C).B; \widetilde{B}\}]$ or $T \equiv T' \mid c : [B]$ and $\tau\, l^{\downarrow}(C) \in \mathit{Msg}(B)$. Furthermore if $T \equiv T' \mid c : [\&\{?\ l^{\downarrow}(C).B; \widetilde{B}\}]$ and there is $T'' = (T' \mid c : [B]) \bowtie a : C$ then $Q :: T''$.*

Proof. By induction on the derivation of the transition $P \xrightarrow{c\ l^{\downarrow}?(a)} Q$, similarly to the proof of Lemma A.1. We show the case of $l^{\downarrow}?(a)$ transition originating from a context piece.

(*Case* $c \blacktriangleleft [P'] \xrightarrow{c\ l^{\downarrow}?(a)} c \blacktriangleleft [Q']$)

We have that:

$$c \blacktriangleleft [P'] :: T \qquad \text{and} \qquad c \blacktriangleleft [P'] \xrightarrow{c\ l^{\downarrow}?(a)} c \blacktriangleleft [Q'] \tag{1}$$

From (1) and considering rule (*Piece*) we have there is L, B such that:

$$(L \bowtie c : [\downarrow B]) \mid loc(\uparrow B) <: T \tag{2}$$

and:

$$c \blacktriangleleft [P'] :: (L \bowtie c : [\downarrow B]) \mid loc(\uparrow B) \qquad \text{and} \qquad P' :: L \mid B \tag{3}$$

We also have that (1) is derived from:

$$P' \xrightarrow{l^{\downarrow}?(a)} Q' \tag{4}$$

Considering Lemma A.1, from (4) and (3) we have there is $T', C, B', \widetilde{B}$ such that either:

$$L \mid B \equiv T' \mid \&\{?\, l^{\downarrow}(C).B'; \widetilde{B}\} \tag{5}$$

or:

$$\tau\, l^{\downarrow}(C) \in Msg(B) \tag{6}$$

In case of (5) we have that there is B_1 such that:

$$B \equiv B_1 \mid \&\{?\, l^{\downarrow}(C).B'; \widetilde{B}\} \tag{7}$$

and there is L' such that $T' \equiv L' \mid B_1$ and $L \equiv L'$. We then have:

$$\downarrow B \equiv \downarrow B_1 \mid \&\{?\, l^{\downarrow}(C). \downarrow B'; \downarrow \widetilde{B}\} \tag{8}$$

From (2), (5) and (8) we conclude:

$$(L' \bowtie c : [\downarrow B_1 \mid \&\{?\, l^{\downarrow}(C). \downarrow B'; \downarrow \widetilde{B}\}]) \mid loc(\uparrow B_1) \mid loc(\uparrow (\&\{?\, l^{\downarrow}(C).B'; \widetilde{B}\}) <: T \tag{9}$$

From (9) we have that either:

$$\begin{aligned}&(L' \bowtie c : [\downarrow B_1 \mid \&\{?\, l^{\downarrow}(C). \downarrow B'; \downarrow \widetilde{B}\}]) \mid loc(\uparrow B_1) \mid loc(\uparrow (\&\{?\, l^{\downarrow}(C).B'; \widetilde{B}\})\\ &\equiv L'' \mid c : [\&\{?\, l^{\downarrow}(C).B''; \widetilde{B}'\}] \mid loc(\uparrow B_1) \mid loc(\uparrow (\&\{?\, l^{\downarrow}(C).B'; \widetilde{B}\})\end{aligned} \tag{10}$$

or:

$$\begin{aligned}&(L' \bowtie c : [\downarrow B_1 \mid \&\{?\, l^{\downarrow}(C). \downarrow B'; \downarrow \widetilde{B}\}]) \mid loc(\uparrow B_1) \mid loc(\uparrow (\&\{?\, l^{\downarrow}(C).B'; \widetilde{B}\})\\ &\equiv L'' \mid c : [\oplus\{\tau\, l^{\downarrow}(C).B''; \widetilde{B}'\}] \mid loc(\uparrow B_1) \mid loc(\uparrow (\&\{?\, l^{\downarrow}(C).B'; \widetilde{B}\})\end{aligned} \tag{11}$$

Then we have that there is $T_1', B_1', B''', \widetilde{B}''$ such that either:

$$T \equiv T_1' \mid c : [B_1' \mid \&\{?\, l^{\downarrow}(C).B'''; \widetilde{B}''\}] \tag{12}$$

where:

$$L'' \mid loc(\uparrow B_1) \mid loc(\uparrow (\&\{?\, l^{\downarrow}(C).B'; \widetilde{B}\})) <: T_1' \tag{13}$$

and:

$$c : [\&\{?\, l^{\downarrow}(C).B''; \widetilde{B}'\}] <: c : [B_1' \mid \&\{?\, l^{\downarrow}(C).B'''; \widetilde{B}''\}] \tag{14}$$

or:

$$T \equiv T_1' \mid c : [B_1' \mid \&\{?\, l^{\downarrow}(C).B'''; \widetilde{B}''\}] \tag{15}$$

In case of (6) proof that $T \equiv T' \mid c : [B']$ and $\tau\, l^{\downarrow}(C) \in Msg(B')$ from $\tau\, l^{\downarrow}(C) \in Msg(B)$ follows expected lines. Let us now consider (12) and that there is T_2 such that:

$$T_2 \equiv (T_1' \mid c : [B_1' \mid B''']) \bowtie a : C \tag{16}$$

From (16), (13) and (5) we conclude there is T'' such that:

$$T'' \equiv (T' \mid B') \bowtie a : C \tag{17}$$

From Lemma A.1 we then have:

$$Q' :: (T' \mid B') \bowtie a : C \tag{18}$$

and thus:

$$Q' :: (L' \mid B_1 \mid B') \bowtie a : C \tag{19}$$

From (19) we derive:

$$c \blacktriangleleft [Q'] :: ((L' \bowtie a : C) \bowtie c : [\downarrow B_1 \mid \ \downarrow B']) \mid loc(\uparrow (B_1 \mid B')) \tag{20}$$

From (10), (13) and (14), we conclude:

$$c \blacktriangleleft [Q'] :: (T_1' \mid c : [B_1' \mid B''']) \bowtie a : C \tag{21}$$

which completes the proof for this case. ∎

Lemma A.3. *Let P be a process such that $P :: T$. If $P \xrightarrow{l^d!(a)} Q$ then there are $T', C, B, \widetilde{B}$ s.t. either $T \equiv T' \mid \oplus\{!\, l^d(C).B; \widetilde{B}\}$ or $T \equiv T' \mid B$ and $\tau\, l^d(C) \in Msg(B)$. Furthermore if $T \equiv T' \mid \oplus\{!\, l^d(C).B; \widetilde{B}\}$ or $l \in \mathcal{L}_\star$ then there is T'' s.t. $T' = T'' \bowtie a : C$ and $Q :: T'' \mid B$.*

Proof. By induction on the length of the derivation of the transition $P \xrightarrow{l^d!(a)} Q$. We show the case when the transition results from a output prefix.

(*Case* $l^d!(a).P' \xrightarrow{l^d!(a)} P'$)

We have that:

$$l^d!(a).P' :: T \qquad \text{and} \qquad l^d!(a).P' \xrightarrow{l^d!(a)} P' \tag{1}$$

From (1) and considering rule (*Output*) we have there is $L, C, B, \widetilde{B}$ such that:

$$(L \bowtie a : C) \mid \oplus\{!\ l^d(C).B; \widetilde{B}\} <: T \tag{2}$$

and:

$$l^d!(a).P' :: (L \bowtie a : C) \mid \oplus\{!\ l^d(C).B; \widetilde{B}\} \qquad \text{and} \qquad P' :: L \mid B \tag{3}$$

From (2) we have there is $T', B', B'', \widetilde{B}'$ such that:

$$T \equiv T' \mid B' \mid \oplus\{!\ l^d(C).B''; \widetilde{B}'\} \tag{4}$$

where:

$$L \bowtie a : C <: T' \tag{5}$$

and:

$$\oplus\{!\ l^d(C).B; \widetilde{B}\} <: B' \mid \oplus\{!\ l^d(C).B''; \widetilde{B}'\} \tag{6}$$

From (5) we have that there is T'' such that $L <: T''$ and:

$$T' \equiv T'' \bowtie a : C \tag{7}$$

From (3) and (6) and $L <: T''$ we conclude:

$$P' :: T'' \mid B' \mid B'' \tag{8}$$

which completes the proof for this case. ■

Lemma A.4. *Let P be a process such that $P :: T$. If $P \xrightarrow{c\ l^{\downarrow}!(a)} Q$ then there are $T', C, B, \widetilde{B}$ s.t. $T \equiv T' \mid c : [\oplus\{!\ l^{\downarrow}(C).B; \widetilde{B}\}]$ or $T \equiv T' \mid c : [B]$ and $\tau\, l^{\downarrow}(C) \in Msg(B)$. Furthermore if $T \equiv T' \mid c : [\oplus\{!\ l^{\downarrow}(C).B; \widetilde{B}\}]$ or $l \in \mathcal{L}_\star$ then there is $T' \equiv T'' \bowtie a : C$ and $Q :: T'' \mid c : [B]$.*

Proof. By induction on the derivation of the transition, following expected lines. ■

Lemma A.5. *Let P be a process such that $P :: T$. If $P \xrightarrow{(\nu a)l^d!(a)} Q$ then there are $T', C, B, \widetilde{B}$ s.t. $T \equiv T' \mid \oplus\{!\ l^d(C).B; \widetilde{B}\}$ or $T \equiv T' \mid B$ and $\tau\, l^d(C) \in Msg(B)$. Furthermore if $T \equiv T' \mid \oplus\{!\ l^d(C).B; \widetilde{B}\}$ or $l \in \mathcal{L}_\star$ then there are B', C' such that $closed(B')$, $a : [B'] = a : C' \bowtie a : C$ and $Q :: T' \mid B \mid a : C'$.*

Proof. By induction on the length of the derivation of the transition $P \xrightarrow{(\nu a)l^d!(a)} Q$. The proof follows similar lines to that of Lemma A.3. We show the case of (*Open*) (Figure 3).

(*Case* $(\boldsymbol{\nu} a)P' \xrightarrow{(\nu a)l^d!(a)} Q'$)

We have that:

$$(\boldsymbol{\nu} a)P' :: T \qquad \text{and} \qquad (\boldsymbol{\nu} a)P' \xrightarrow{(\nu a)l^d!(a)} Q' \tag{1}$$

which is derived from:

$$P' \xrightarrow{l^d!(a)} Q' \tag{2}$$

From (1) and considering rule (*Res*) we have that there is T', B such that:

$$T' <: T \qquad \text{and} \qquad (\boldsymbol{\nu} a)P' :: T' \qquad \text{and} \qquad P' :: T' \mid a : [B] \tag{3}$$

and $closed(B)$. Considering Lemma A.3 and (2) and (3) we have that there are $T_1, C, B', \widetilde{B}$ such that either:

$$T' \mid a : [B] \equiv T_1 \mid \oplus\{!\, l^d(C).B'; \widetilde{B}\} \tag{4}$$

or:

$$T' \mid a : [B] \equiv T_1 \mid B' \qquad \text{and} \qquad \boldsymbol{\tau}\, l^d(C) \in Msg(B') \tag{5}$$

We show the case of (4). We have that there is T_1' such that $T_1 \equiv T_1' \mid a : [B]$ and:

$$T' \equiv T_1' \mid \oplus\{!\, l^d(C).B'; \widetilde{B}\} \tag{6}$$

From (6) and $T' <: T$ we conclude there is $T'', B'', B''', \widetilde{B}'$ such that:

$$T \equiv T'' \mid B'' \mid \oplus\{!\, l^d(C).B'''; \widetilde{B}'\} \tag{7}$$

where:

$$T_1' <: T'' \tag{8}$$

and:

$$\oplus\{!\, l^d(C).B'; \widetilde{B}\} <: B'' \mid \oplus\{!\, l^d(C).B'''; \widetilde{B}'\} \tag{9}$$

Let us now consider (6) (proof when $l \in \mathcal{L}_\star$ follows similar lines). We then have that must be the case of (4), hence, from Lemma A.3 we conclude:

$$T_1 \equiv T_1'' \bowtie a : C \qquad \text{and} \qquad Q' :: T_1'' \mid B' \tag{10}$$

From $T_1 \equiv T_1' \mid a : [B]$ we then have:

$$T_1' \mid a : [B] \equiv T_1'' \bowtie a : C \tag{11}$$

From (11) we conclude there are T_2, C' such that $T_2 \equiv T_1'$:

$$T_1'' \equiv T_2 \mid a : C' \qquad \text{and} \qquad a : [B] = a : C' \bowtie a : C \tag{12}$$

From (10) and (12) and $T_2 \equiv T_1'$ we have:

$$Q' :: T_1' \mid a : C' \mid B' \tag{13}$$

From (13) and $T'_1 <: T''$—(8)—we then have:

$$Q' :: T'' \mid a : C' \mid B' \tag{14}$$

Finally, from (14) and (9) we conclude:

$$Q' :: T'' \mid a : C' \mid B'' \mid B''' \tag{15}$$

which completes the proof for this case. ∎

Lemma A.6. *Let P be a process such that $P :: T$. If $P \xrightarrow{(\nu a)c\ l^{\downarrow}!(a)} Q$ then there are $T', C, B, \widetilde{B}$ such that $T \equiv T' \mid c : [\oplus\{!\,l^{\downarrow}(C).B; \widetilde{B}\}]$ or $T \equiv T' \mid c : [B]$ and $\tau\, l^{\downarrow}(C).B \in Msg(B)$. Furthermore if $T \equiv T' \mid c : [\oplus\{!\,l^{\downarrow}(C).B; \widetilde{B}\}]$ or $l \in \mathcal{L}_\star$ then there are B', C' such that $closed(B')$ and $a : [B'] = a : C' \bowtie a : C$ and $Q :: (T' \bowtie c : [B]) \mid a : C'$.*

Proof. By induction on the length of the derivation of the transition $P \xrightarrow{(\nu a)c\ l^{\downarrow}!(a)} Q$, following the lines of the proof of Lemma A.5. ∎

Lemma A.7. *Let P be a well-typed process such that $P :: T$. If $P \xrightarrow{c\ \mathtt{this}} Q$ due to a* **this** *prefix, then there are L, B_1, B_2 such that $T \equiv L \mid (B_1 \bowtie (\downarrow B_2))$. Furthermore if there is T' such that $T' \equiv (L \mid B_1) \bowtie (c : [\downarrow B_2])$ then $Q :: (L \mid B_1) \bowtie (c : [\downarrow B_2])$.*

Proof. By induction on the derivation of $P \xrightarrow{c\ \mathtt{this}} Q$, following expected lines. ∎

Lemma A.8. *Let P be a process such that $P :: T$. If $P \xrightarrow{c\ \mathtt{this}} Q$ then there are $T', B_1, B_2, \widetilde{B}, \widetilde{B}', C_1, C_2, l$ such that $T \equiv T' \mid \{p_1\, l^{\downarrow}(C_1).B_1; \widetilde{B}\} \mid c : [\{p_2\, l^{\downarrow}(C_2).B_2; \widetilde{B}'\}]$ where $p_i = !$ and $p_j = ?$ for $\{i, j\} = \{1, 2\}$, or $T \equiv T' \mid B_1 \mid c : [B_2]$ and $p_1\, l^{\downarrow}(C_1) \in Msg(B_1)$ and $p_2\, l^{\downarrow}(C_2) \in Msg(B_2)$. Furthermore if $T \equiv T' \mid \{p_1\, l^{\downarrow}(C_1).B_1; \widetilde{B}\} \mid c : [\{p_2\, l^{\downarrow}(C_2).B_2; \widetilde{B}'\}]$ and $p_i = !$ or $l \in \mathcal{L}_\star$ and $p_j = ?$ and $C_1 \equiv C_2$ then we have that $Q :: T' \mid B_1 \mid c : [B_2]$.*

Proof. By induction on the length of the derivation of the transition $P \xrightarrow{c\ \mathtt{this}} Q$. We show the case of a (*ThisComm*) synchronization.

(*Case* $P_1 \mid P_2 \xrightarrow{c\ \mathtt{this}} Q_1 \mid Q_2$)

We have that:

$$P_1 \mid P_2 \xrightarrow{c\ \mathtt{this}} Q_1 \mid Q_2\ (i) \qquad \text{and} \qquad P_1 \mid P_2 :: T\ (ii) \tag{1}$$

(1)(i) is derived from:

$$P_1 \xrightarrow{c\ l^{\downarrow}!(a)} Q_1 \qquad \text{and} \qquad P_2 \xrightarrow{l^{\downarrow}?(a)} Q_2 \tag{2}$$

From (1)(ii) we have there are T_1, T_2 such that:

$$T_1 \bowtie T_2 <: T \qquad \text{and} \qquad P_1 :: T_1 \qquad \text{and} \qquad P_2 :: T_2 \tag{3}$$

Considering Lemma A.4, (2) and (3) we have there are $T_1', C_1, B_1, \widetilde{B}$ such that either:

$$T_1 \equiv T_1' \mid c : [\oplus\{ ! \, l^{\downarrow}(C_1).B_1; \widetilde{B}\}] \tag{4}$$

or:

$$T_1 \equiv T_1' \mid c : [B_1] \qquad \text{and} \qquad \tau \, l^{\downarrow}(C_1) \in Msg(B_1) \tag{5}$$

Considering Lemma A.1, (2) and (3) we have there are $T_2', C_2, B_2, \widetilde{B}'$ such that:

$$T_2 \equiv T_2' \mid \&\{ ? \, l^{\downarrow}(C_2).B_2; \widetilde{B}'\} \tag{6}$$

or:

$$T_2 \equiv T_2' \mid B_2 \qquad \text{and} \qquad \tau \, l^{\downarrow}(C_2) \in Msg(B_2) \tag{7}$$

From $T_1 \bowtie T_2 <: T$ and (4), (5), (6) and (7) we directly have that $T \equiv T' \mid c : [B_1] \mid B_2$ such that $p_1 \, l^{\downarrow}(C_1) \in Msg(B_1)$ and $p_2 \, l^{\downarrow}(C_2) \in Msg(B_2)$.

Let us now consider the case of (4) and (6 and also that $! \, l^{\downarrow}(C_1) \# T_2$ and $! \, l^{\downarrow}(C_2) \# T_1$. We then have that:

$$\begin{aligned} T &\equiv (T_1'' \mid c : [B_1' \mid \oplus\{ ! \, l^{\downarrow}(C_1).B_1''; \widetilde{B}''\}]) \bowtie (T_2'' \mid B_2' \mid \&\{ ? \, l^{\downarrow}(C_2).B_2''; \widetilde{B}'''\}) \\ &\equiv T' \mid c : [\oplus\{ ! \, l^{\downarrow}(C_1).B'; (\cdots)\}] \mid \&\{ ? \, l^{\downarrow}(C_2).B''; (\cdots)\} \end{aligned} \tag{8}$$

where $T_1' <: T_1''$ and $T_2' <: T_2''$ and:

$$\oplus\{ ! \, l^{\downarrow}(C_1).B_1; \widetilde{B}\} <: B_1' \mid \oplus\{ ! \, l^{\downarrow}(C_1).B_1''; \widetilde{B}''\} \tag{9}$$

and:

$$\&\{ ? \, l^{\downarrow}(C_2).B_2; \widetilde{B}'\} <: B_2' \mid \&\{ ? \, l^{\downarrow}(C_2).B_2''; \widetilde{B}'''\} \tag{10}$$

Let us now consider $(C \equiv) C_1 \equiv C_2$. Considering Lemma A.4 we have there is T_1''' such that:

$$T_1' \equiv T_1''' \bowtie a : C \qquad \text{and} \qquad Q_1 :: T_1''' \mid c : [B_1] \tag{11}$$

Then, via Lemma A.1, considering (11) and $T_1' <: T_1''$ and (8) and $T_2' <: T_2''$ we conclude that there is T_2''' such that:

$$T_2''' = (T_2' \mid B_2) \bowtie a : C \qquad \text{and} \qquad Q_2 :: T_2''' \tag{12}$$

From (11) and (9) we conclude:

$$Q_1 :: T_1''' \mid c : [B_1' \mid B_1''] \tag{13}$$

Likewise from (12), (10) we conclude:

$$Q_2 :: (T_2' \mid a : C) \mid B_2' \mid B_2'' \tag{14}$$

Then from (13) and (14) we have:

$$Q_1 \mid Q_2 :: (T_1''' \mid c : [B_1' \mid B_1'']) \bowtie ((T_2' \bowtie a : C) \mid B_2' \mid B_2'') \tag{15}$$

which, considering (11) leads to:

$$Q_1 \mid Q_2 :: (T_1' \mid c : [B_1' \mid B_1'']) \bowtie (T_2' \mid B_2' \mid B_2'') \tag{16}$$

Since $T_1' <: T_1''$ and $T_2' <: T_2''$ we derive:

$$Q_1 \mid Q_2 :: (T_1'' \mid c : [B_1' \mid B_1'']) \bowtie (T_2'' \mid B_2' \mid B_2'') \tag{17}$$

which, along with (8), gives us:

$$Q_1 \mid Q_2 :: T' \mid c : [B'] \mid B'' \tag{18}$$

which completes the proof for this case. ∎

Lemma A.9. *Let types T_1, T_1', T_2' be such that $T_1' <: T_1$ and $T_1' \to T_2'$. Then we have that there is type T_2 such that $T_1 \to T_2$ and $P :: T_2'$ implies $P :: T_2$.*

Proof. By induction on the length of the derivation of $T_1' <: T_1$, following expected lines. ∎

Proof of Theorem 3.20 (Subject Reduction)
Let P be a process and T a type such that $P :: T$. If $P \to Q$ then there is type T' such that $T \to T'$ and $Q :: T'$.
Proof. By induction on the derivation of the reduction $P \to Q$. We show the case of a reduction derived from a synchronization on an unlocated—at the level of the current conversation—message, distinguishing between when the message is defined on a plain label and on a shared label, and the case of a τ derived from a c `this` transition originating from a conversation piece.

(*Case unlocated message synchronization*)

We have:

$$P_1 \mid P_2 \xrightarrow{\tau} Q_1 \mid Q_2 \tag{1}$$

derived from:

$$P_1 \xrightarrow{l^{\downarrow}!(a)} Q_1 \ \ (i) \qquad \text{and} \qquad P_2 \xrightarrow{l^{\downarrow}?(a)} Q_2 \ \ (ii) \tag{2}$$

Since $P_1 \mid P_2$ is a well-typed process, we have $P_1 \mid P_2 :: T$ for some T such that $T' <: T$ and:

$$P_1 \mid P_2 :: T' \tag{3}$$

where (3) is derived from (rule (Par)):

$$P_1 :: T_1 \ \ (i) \qquad \text{and} \qquad P_2 :: T_2 \ \ (ii) \qquad \text{and} \qquad T' = T_1 \bowtie T_2 \ \ (iii) \tag{4}$$

From (2)(i) and (4)(i) and considering Lemma A.3 we conclude that there are $T_1', C_1, B_1, \widetilde{B}$ such that either:

$$T_1 \equiv T_1' \mid \oplus\{!\, l^{\downarrow}(C_1).B_1; \widetilde{B}\} \tag{5}$$

or:

$$T_1 \equiv T_1' \mid B \qquad \text{and} \qquad \tau\, l^{\downarrow}(C_1) \in \mathit{Msg}(B_1) \tag{6}$$

From (2)(ii) and (4)(ii), considering Lemma A.1, we conclude that there are $T_2', C_2, B_2, \widetilde{B}'$ such that either:

$$T_2 \equiv T_2' \mid \&\{?\, l^{\downarrow}(C_2).B_2; \widetilde{B}'\} \tag{7}$$

or:

$$T_2 \equiv T_2' \mid B \qquad \text{and} \qquad \tau\, l^{\downarrow}(C_2) \in \mathit{Msg}(B_2) \tag{8}$$

We consider the two possible cases: either the label is plain ($l \in \mathcal{L}_p$) or it is shared ($l \in \mathcal{L}_\star$).

(*Plain label*) If l is a plain label, from (4)(iii) we have that it must be the case that in T' there is a τ introduced by rule (*Plain*) for this synchronization which is only possible if (5) and (7) and also that $C_1 \equiv C_2$ ($\equiv C$), otherwise the merge $T_1 \bowtie T_2$ (4)(iii) would not be defined. We then have, from Lemma A.3 that there is T'' such that:

$$T_1' = T_1'' \bowtie a : C \;\; (i) \qquad \text{and} \qquad Q_1 :: T_1'' \mid B_1 \;\; (ii) \tag{9}$$

From the merge in (4)(iii), considering (5), (9)(i) and (7) we conclude:

$$T' = ((T_1'' \bowtie a : C) \mid \oplus\{!\, l^{\downarrow}(C).B_1; \widetilde{B}\}) \bowtie (T_2' \mid \&\{?\, l^{\downarrow}(C).B_2; \widetilde{B}'\}) \tag{10}$$

From (10) we have that there is $T'' = (T_2' \mid B_2) \bowtie a : C$. From Lemma A.1 we have:

$$Q_2 :: (T_2' \mid B_2) \bowtie a : C \tag{11}$$

From (10) we also have that there is T''' such that $T''' = (T_1'' \bowtie B_1) \bowtie ((T_2' \bowtie B_2) \bowtie a : C)$, hence from (9)($ii$) and (11) we conclude:

$$Q_1 \mid Q_2 :: (T_1'' \mid B_1) \bowtie ((T_2' \mid B_2) \bowtie a : C) \tag{12}$$

The merge of plain label message types necessarily yields a τ message type. We can thus derive a reduction for type T' as follows:

$$\begin{array}{l}
((T_1'' \bowtie a : C) \mid \oplus\{!\, l^{\downarrow}(C).B_1; \widetilde{B}\}) \bowtie (T_2' \mid \&\{?\, l^{\downarrow}(C).B_2; \widetilde{B}'\}) \\
\equiv \\
((T_1'' \bowtie a : C) \bowtie T_2') \bowtie \oplus\{\tau\, l^{\downarrow}(C).(B_1 \bowtie B_2); (\cdots)\} \\
\rightarrow \\
((T_1'' \bowtie a : C) \bowtie T_2') \bowtie (B_1 \bowtie B_2) \\
\equiv \\
(T_1'' \mid B_1) \bowtie ((T_2' \mid B_2) \bowtie a : C)
\end{array}$$

Since $T' <: T$ from Lemma A.9 we have that there is T'' such that $T \rightarrow T''$ and $Q_1 \mid Q_2 :: T''$ which completes the proof for this case.

(*Shared label*) If l is a shared label then by conformance we have that $C_1 \equiv C_2 (\equiv C)$. From the definition of merge and (4)(iii) we conclude it must be the case of (7) and

furthermore we have that $\&\{?\, l^d(C).B_2; \widetilde{B'}\} \equiv \star ?\, l^d(C)$. Also, considering Lemma A.3, from either (5) or (6), since $l \in \mathcal{L}_\star$, we have there is T'' such that:

$$T_1' = T_1'' \bowtie a : C_1 \;\; (i) \qquad \text{and} \qquad Q_1 :: T_1'' \mid B_1 \;\; (ii) \tag{13}$$

We show only the proof for (5), as the proof for (6) follows similar lines. From the merge in (4)(iii), and (13)(i) and (7) we conclude:

$$T' = ((T_1'' \bowtie a : C) \mid \oplus\{?\, l^{\downarrow}(C).B_1; \widetilde{B}\}) \bowtie (T_2' \mid \star ?\, l^d(C)) \tag{14}$$

which leads to:

$$\begin{array}{l} ((T_1'' \bowtie a : C) \bowtie T_2') \bowtie (\oplus\{?\, l^{\downarrow}(C).B_1; \widetilde{B}\} \bowtie \star ?\, l^d(C)) \\ \equiv \\ ((T_1'' \bowtie a : C) \bowtie T_2') \\ \bowtie (\oplus\{\tau\, l^{\downarrow}(C).(B_1\{!\, l^{\downarrow}(C) \leftarrow \tau\, l^{\downarrow}(C)\}); (\widetilde{B}\{!\, l^{\downarrow}(C) \leftarrow \tau\, l^{\downarrow}(C)\})\} \mid \star ?\, l^d(C)) \end{array} \tag{15}$$

From (15) we conclude:

$$(T_2' \mid \star ?\, l^d(C)) \bowtie a : C \tag{16}$$

Then from Lemma A.1 and (16) we have:

$$Q_2 :: (T_2' \mid \star ?\, l^d(C)) \bowtie a : C \tag{17}$$

From (13)(ii), (17) and (15) we derive:

$$Q_1 \mid Q_2 :: (T_1'' \mid B_1) \bowtie ((T_2' \mid \star ?\, l^d(C)) \bowtie a : C) \tag{18}$$

From (15) we have:

$$\begin{array}{l} ((T_1'' \bowtie a : C) \bowtie T_2') \\ \bowtie (\oplus\{\tau\, l^{\downarrow}(C).(B_1\{!\, l^{\downarrow}(C) \leftarrow \tau\, l^{\downarrow}(C)\}); (\widetilde{B}\{!\, l^{\downarrow}(C) \leftarrow \tau\, l^{\downarrow}(C)\})\} \mid \star ?\, l^d(C)) \\ \rightarrow \\ ((T_1'' \bowtie a : C) \bowtie T_2') \bowtie (B_1\{!\, l^{\downarrow}(C) \leftarrow \tau\, l^{\downarrow}(C)\}) \mid \star ?\, l^d(C)) \\ \equiv \\ (T_1'' \mid B_1) \bowtie ((T_2' \mid \star ?\, l^d(C)) \bowtie a : C) \end{array} \tag{19}$$

We then have, from $T' <: T$, (14), (15) and (19) and Lemma A.9 that there is T'' such that $T \rightarrow T''$ and $Q_1 \mid Q_2 :: T''$ which completes the proof for this case.

(*Case* c `this`)

We have:

$$c \blacktriangleleft [P] \xrightarrow{\tau} c \blacktriangleleft [Q] \tag{20}$$

derived from:

$$P \xrightarrow{c\ \mathtt{this}} Q \tag{21}$$

We have that $c \blacktriangleleft [P] :: T$. Also we have that there is T' such that $T' <: T$ and:

$$c \blacktriangleleft [P] :: T' \tag{22}$$

where (22) is derived from (rule (*Piece*)):

$$P :: L \mid B \;\; (i) \qquad \text{and} \qquad T' = (L \bowtie c : [\downarrow B]) \mid loc(\uparrow B) \;\; (ii) \tag{23}$$

We must consider the two distinct cases: either the transition originates from a **this** prefix or from a message synchronization. The proof of the first case follows expected lines. We show the proof for the second case, when the transition originates in a message synchronization. Considering Lemma A.8 and from (23)(i) and (21) we conclude that there exist $T'', B_1, B_2, \widetilde{B}, \widetilde{B}', C_1, C_2, l$ such that:

$$L \mid B \equiv T'' \mid \{p_1\, l^{\downarrow}(C_1).B_1; \widetilde{B}\} \mid c : [\{p_2\, l^{\downarrow}(C_2).B_2; \widetilde{B}'\}] \tag{24}$$

and $p_i = \,!$ and $p_j = \,?$, or:

$$L \mid B \equiv T'' \mid B_1 \mid c : [B_2] \tag{25}$$

and $p_1\, l^{\downarrow}(C_1) \in Msg(B_1)$ and $p_2\, l^{\downarrow}(C_2) \in Msg(B_2)$. From $(L \bowtie c : [\downarrow B]) \mid loc(\uparrow B)$ we directly have that it must be the case of (24) otherwise the merge would be undefined. We show the case when $i = 1$ and $j = 2$ and l is a plain label (the proofs for the other cases follow similar lines). From (24) we conclude there exist L_1, B_3 such that $T'' \equiv L_1 \mid B_3$ and:

$$L \equiv L_1 \mid c : [\&\{?\, l^{\downarrow}(C_2).B_2; \widetilde{B}'\}] \qquad \text{and} \qquad B \equiv B_3 \mid \oplus\{!\, l^{\downarrow}(C_1).B_1; \widetilde{B}\} \tag{26}$$

From (26) we have that:

$$\downarrow B \equiv \downarrow B_3 \mid \oplus\{!\, l^{\downarrow}(C_1).(\downarrow B_1); (\downarrow \widetilde{B})\} \tag{27}$$

and:

$$loc(\uparrow B) \equiv loc(\uparrow B_3) \mid loc(\uparrow \oplus\{!\, l^{\downarrow}(C_1).B_1; \widetilde{B}\}) \tag{28}$$

From (23)(ii), (26) and (27) and (28) we have:

$$\begin{array}{rcl} T' & = & ((L_1 \mid c : [\&\{?\, l^{\downarrow}(C_2).B_2; \widetilde{B}'\}]) \bowtie c : [\downarrow B_3 \mid \oplus\{!\, l^{\downarrow}(C_1).(\downarrow B_1); (\downarrow \widetilde{B})\}]) \\ & & \mid loc(\uparrow B_3) \mid loc(\uparrow \oplus\{!\, l^{\downarrow}(C_1).B_1; \widetilde{B}\}) \end{array} \tag{29}$$

We also have that $C_1 \equiv C_2$ ($\equiv C$). From Lemma A.8 we then have:

$$Q :: T'' \mid B_1 \mid c : [B_2] \tag{30}$$

which, since $T'' \equiv L_1 \mid B_3$ gives us:

$$Q :: L_1 \mid B_3 \mid B_1 \mid c : [B_2] \tag{31}$$

From (31) we derive:

$$c \blacktriangleleft [Q] :: ((L_1 \mid c : [B_2]) \bowtie c : [\downarrow (B_3 \mid B_1)]) \mid loc(\uparrow (B_3 \mid B_1)) \tag{32}$$

We conclude:

$$c \blacktriangleleft [Q] :: ((L_1 \mid c : [B_2]) \bowtie c : [\downarrow (B_3 \mid B_1)]) \mid loc(\uparrow B_3) \mid loc(\uparrow \{p_1\, l^{\downarrow}(C_1).B_1; \widetilde{B}\}) \tag{33}$$

and derive the following type reduction:

$$
\begin{array}{l}
((L_1 \mid c : [\&\{?\, l^{\downarrow}(C_2).B_2; \widetilde{B}'\}]) \bowtie c : [\downarrow B_3 \mid \oplus\{!\, l^{\downarrow}(C_1).(\downarrow B_1); (\downarrow \widetilde{B})\}]) \\
\ \mid loc(\uparrow B_3) \mid loc(\uparrow \oplus\{!\, l^{\downarrow}(C_1).B_1; \widetilde{B}\}) \\
\equiv \\
((L_1 \bowtie c : [\downarrow B_3]) \bowtie c : [\oplus\{\tau\, l^{\downarrow}(C).((\downarrow B_1) \bowtie B_2); \widetilde{B}'''\}]) \mid loc(\uparrow B_3) \mid loc(\uparrow \oplus\{!\, l^{\downarrow}(C_1).B_1; \widetilde{B}\}) \\
\rightarrow \\
((L_1 \bowtie c : [\downarrow B_3]) \bowtie c : [(\downarrow B_1) \bowtie B_2]) \mid loc(\uparrow B_3) \mid loc(\uparrow \oplus\{!\, l^{\downarrow}(C_1).B_1; \widetilde{B}\}) \\
\equiv \\
((L_1 \mid c : [B_2]) \bowtie c : [\downarrow (B_3 \mid B_1)]) \mid loc(\uparrow B_3) \mid loc(\uparrow \oplus\{!\, l^{\downarrow}(C_1).B_1; \widetilde{B}\})
\end{array}
\tag{34}
$$

which, along with (29) and $T' <: T$, considering Lemma A.9 gives us there is T''' such that $T \rightarrow T'''$ and $c \blacktriangleleft [Q] :: T'''$, which completes the proof for this case. ■

B. Proofs for Results of Section 4

We first state a weakening property for the progress proof system judgments.

Lemma B.1. *Let P be a well-typed process and Γ an event ordering such that $\Gamma \vdash_\ell P$. If $\Gamma \cup \Gamma'$ is an event ordering then $\Gamma \cup \Gamma' \vdash_\ell P$.*

Proof. By induction on the structure of P, following expected lines. Intuitively if Γ already proves that events are well ordered in P then Γ' describes an ordering of events that do not pertain to P, and hence Γ' does not interfere in verifying the event ordering of P. ■

Proof of Lemma 4.4 (Substitution)

Let P be a process and Γ, Γ' event orderings such that $\Gamma \cup \Gamma' \vdash_\ell P$ and $\Gamma'\{x \leftarrow n\} \subseteq \Gamma$. Then $\Gamma \vdash_{\ell\{x \leftarrow n\}} P\{x \leftarrow n\}$.

Proof. By induction on the structure of P. We show the case when P is a conversation piece $x \blacktriangleleft [P]$ and an output prefixed process $l^d!(x).P$.

(*Case* $x \blacktriangleleft [P]$)

We have that

$$\Gamma \cup \Gamma' \vdash_\ell x \blacktriangleleft [P] \quad (i) \qquad \text{and} \qquad \Gamma'\{x \leftarrow n\} \subseteq \Gamma \quad (ii) \tag{1}$$

derived from

$$\Gamma \cup \Gamma' \vdash_{(\ell(\downarrow),x)} P \tag{2}$$

By induction hypothesis on (2) and (1)(ii) we have

$$\Gamma \vdash_{(\ell(\downarrow)\{x \leftarrow n\},n)} P\{x \leftarrow n\} \tag{3}$$

From (3) we derive

$$\Gamma \vdash_{\ell\{x \leftarrow n\}} n \blacktriangleleft [P\{x \leftarrow n\}] \tag{4}$$

which completes the proof for this case.

(*Case* $l^d!(x).P$) We have that:

$$\Gamma \cup \Gamma' \vdash_\ell l^d!(x).P \qquad \text{and} \qquad \Gamma'\{x \leftarrow n\} \subseteq \Gamma \tag{5}$$

(5) is derived from (rule (*Output*)):

$$(\ell(d).l.(y)\Gamma'' \bot (\Gamma \cup \Gamma')) \vdash_\ell P \;\; (i) \qquad \text{and} \qquad \Gamma''\{y \leftarrow x\} \subseteq (\ell(d).l.(y)\Gamma'' \bot (\Gamma \cup \Gamma')) \;\; (ii) \tag{6}$$

From (6)(i), considering Lemma B.1, we conclude $\Gamma \cup \Gamma' \vdash_\ell P$. By induction hypothesis:

$$\Gamma \vdash_{\ell\{x \leftarrow n\}} P\{x \leftarrow n\} \tag{7}$$

We have that $\ell(d).l.(y)\Gamma'' \in dom(\Gamma \cup \Gamma')$ and hence:

$$(\ell(d).l.(y)\Gamma'')\{x \leftarrow n\} \in dom(\Gamma) \tag{8}$$

Also from (6)(i) we have that events in P are of greater order w.r.t. event $\ell(d).l.(y)\Gamma''$, so events in $P\{x \leftarrow n\}$ are of greater order w.r.t. $(\ell(d).l.(y)\Gamma'')\{x \leftarrow n\}$:

$$((\ell(d).l.(y)\Gamma'')\{x \leftarrow n\} \bot \Gamma) \vdash_{\ell\{x \leftarrow n\}} P\{x \leftarrow n\} \tag{9}$$

From (6)(ii) and $\Gamma'\{x \leftarrow n\} \subseteq \Gamma$—(5)—we conclude:

$$(\Gamma''\{y \leftarrow x\})\{x \leftarrow n\} \subseteq (\ell(d).l.(y)\Gamma'' \bot (\Gamma \cup \Gamma'))\{x \leftarrow n\} \subseteq ((\ell(d).l.(y)\Gamma'')\{x \leftarrow n\} \bot \Gamma) \tag{10}$$

and hence $\Gamma''\{y \leftarrow n\} \subseteq ((\ell(d).l.(y)\Gamma'')\{x \leftarrow n\} \bot \Gamma)$. Then, considering also (9), we conclude $\Gamma \vdash_{\ell\{x \leftarrow n\}} l^d!(n).(P\{x \leftarrow n\})$ which completes the proof for this case. ∎

Lemma B.2. *Let P be a well-typed process and Γ an event ordering such that $\Gamma \vdash_\ell P$. If $P \xrightarrow{l^d?(a)} Q$ and $(\ell(d).l.(x)\Gamma') \in dom(\Gamma)$ and $\Gamma'\{x \leftarrow a\} \subseteq (\ell(d).l.(x)\Gamma') \bot \Gamma$ then $\Gamma \vdash_\ell Q$.*

Proof. By induction on the derivation of the transition. We show the case when P is an input summation.

(*Case* $\Sigma_{i \in I}\, l_i^d?(x_i).P_i \xrightarrow{l_j^d?(a)} P_j\{x_j \leftarrow a\}$)

We have that

$$\Gamma \vdash_\ell \Sigma_{i \in I}\, l_i^d?(x_i).P_i \qquad \text{and} \qquad \Sigma_{i \in I}\, l_i^d?(x_i).P_i \xrightarrow{l_j^d?(a)} P_j\{x_j \leftarrow a\} \tag{1}$$

and

$$(\ell(d).l_i.(y)\Gamma'_j \in dom(\Gamma)) \qquad \text{and} \qquad \Gamma'_j\{y \leftarrow a\} \subseteq (\ell(d).l_i.(y)\Gamma'_j \bot \Gamma) \tag{2}$$

We have that (1) is derived from

$$(\ell(d).l_i.(y)\Gamma'_i \bot \Gamma) \cup \Gamma'_i\{y \leftarrow x_i\} \vdash_\ell P_i \tag{3}$$

in particular for j we have

$$(\ell(d).l_j.(y)\Gamma'_j \bot \Gamma) \cup \Gamma'_j\{y \leftarrow x_j\} \vdash_\ell P_j \tag{4}$$

From Lemma 4.4 considering (4) and (2) we then have

$$(\ell(d).l_j.(y)\Gamma'_j \bot \Gamma) \vdash_\ell P_j\{x_j \leftarrow a\} \tag{5}$$

From (5) and considering Lemma B.1 we have

$$\Gamma \vdash_\ell P_j\{x_j \leftarrow a\} \tag{6}$$

which completes the proof for this case. ∎

Lemma B.3. *Let P be a well-typed process and Γ an event ordering such that $\Gamma \vdash_\ell P$. If $P \xrightarrow{c\ l^{\downarrow}?(a)} Q$ and $(c.l.(x)\Gamma') \in dom(\Gamma)$ and $\Gamma'\{x \leftarrow a\} \subseteq (c.l.(x)\Gamma')\bot\Gamma$ then $\Gamma \vdash_\ell Q$.*

Proof. By induction on the derivation of the label. We show the base case.

(*Case* $c \blacktriangleleft [P'] \xrightarrow{c\ l^{\downarrow}?(a)} c \blacktriangleleft [Q']$)

We have that

$$\Gamma \vdash_\ell c \blacktriangleleft [P'] \tag{1}$$

Let us consider

$$c \blacktriangleleft [P'] \xrightarrow{c\ l^{\downarrow}?(a)} c \blacktriangleleft [Q'] \quad \text{and} \quad (c.l.(x)\Gamma' \in dom(\Gamma)) \quad \text{and} \quad \Gamma'\{x \leftarrow a\} \subseteq (c.l.(x)\Gamma' \bot \Gamma) \tag{2}$$

We have that (1) is derived from

$$\Gamma \vdash_{(\ell(\downarrow),c)} P' \tag{3}$$

and (2) is derived from

$$P' \xrightarrow{l^{\downarrow}?(a)} Q' \tag{4}$$

From Lemma B.2 considering (4), (3) and (2) we have

$$\Gamma \vdash_{(\ell(\downarrow),c)} Q' \tag{5}$$

From (5) we derive

$$\Gamma \vdash_\ell c \blacktriangleleft [Q'] \tag{6}$$

which completes the proof for this case. ∎

Lemma B.4. *Let P be a well-typed process and Γ an event ordering such that $\Gamma \vdash_\ell P$. If $P \xrightarrow{l^d!(a)} Q$ then $\Gamma \vdash_\ell Q$ and $(\ell(d).l.(x)\Gamma') \in dom(\Gamma)$ and $\Gamma'\{x \leftarrow a\} \subseteq (\ell(d).l.(x)\Gamma')\bot\Gamma$.*

Proof. By induction on the derivation of the transition. We show the case when P is an output prefix.

(*Case* $l^d!(a).P' \xrightarrow{l^d!(a)} P'$)

We have that

$$\Gamma \vdash_\ell l^d!(a).P' \;\; (i) \qquad \text{and} \qquad l^d!(a).P' \xrightarrow{l^d!(a)} P' \;\; (ii) \tag{1}$$

We have that (1)(i) is derived from

$$(\ell(d).l.(x)\Gamma' \bot \Gamma) \vdash_\ell P' \qquad \text{and} \qquad \Gamma'\{x \leftarrow a\} \subseteq (\ell(d).l.(x)\Gamma' \bot \Gamma) \tag{2}$$

From (2) and considering Lemma B.1 we have

$$\Gamma \vdash_\ell P' \tag{3}$$

which completes the proof for this case. ■

Lemma B.5. *Let P be a well-typed process and Γ an event ordering such that $\Gamma \vdash_\ell P$. If $P \xrightarrow{c\ l^\downarrow!(a)} Q$ then $\Gamma \vdash_\ell Q$ and $(c.l.(x)\Gamma') \in dom(\Gamma)$ and $\Gamma'\{x \leftarrow a\} \subseteq (c.l.(x)\Gamma')\bot\Gamma$.*

Proof. By induction on the derivation of the transition. We show the base case.

(*Case* $c \blacktriangleleft [P'] \xrightarrow{c\ l^\downarrow!(a)} c \blacktriangleleft [Q']$)

We have that

$$\Gamma \vdash_\ell c \blacktriangleleft [P'] \tag{1}$$

Let us consider

$$c \blacktriangleleft [P'] \xrightarrow{c\ l^\downarrow!(a)} c \blacktriangleleft [Q'] \tag{2}$$

We have that (1) is derived from

$$\Gamma \vdash_{(\ell(\downarrow),c)} P' \tag{3}$$

and (2) is derived from

$$P' \xrightarrow{l^\downarrow!(a)} Q' \tag{4}$$

From (3) and (4), considering Lemma B.4 we have

$$(c.l.(x)\Gamma' \bot \Gamma) \vdash_\ell P' \qquad \text{and} \qquad \Gamma'\{x \leftarrow a\} \subseteq (c.l.(x)\Gamma' \bot \Gamma) \qquad \text{and} \qquad \Gamma \vdash_{(\ell(\downarrow),c)} Q' \tag{5}$$

From (5) we conclude

$$\Gamma \vdash_\ell c \blacktriangleleft [Q'] \tag{6}$$

which completes the proof for this case. ■

Lemma B.6. *Let P be a well-typed process and Γ an event ordering such that $\Gamma \vdash_\ell P$. If $P \xrightarrow{(\nu a)l^d!(a)} Q$ then there is Γ' such that $\Gamma \cup \Gamma' \vdash_\ell Q$ and $(\Gamma \cup \Gamma') \setminus a \subseteq \Gamma$ and $(\ell(d).l.(x)\Gamma'') \in dom(\Gamma)$ and $\Gamma''\{x \leftarrow a\} \subseteq ((\ell(d).l.(x)\Gamma'')\bot(\Gamma \cup \Gamma'))$.*

Proof. By induction on the derivation of the transition. We show the base case of restriction open (Figure 3 (*Open*)).

($Case\ (\nu a)P' \xrightarrow{(\nu a)l^d!(a)} Q'$)

We have that

$$\Gamma \vdash_\ell (\nu a)P' \tag{1}$$

Let us consider

$$(\nu a)P' \xrightarrow{(\nu a)l^d!(a)} Q' \tag{2}$$

We have that (1) is derived from

$$\Gamma' \vdash_\ell P' \tag{3}$$

where $\Gamma = \Gamma' \setminus a$. (2) is derived from

$$P' \xrightarrow{l^d!(a)} Q' \tag{4}$$

From (3) and (4), considering Lemma B.4 we have

$$(\ell(d).l.(x)\Gamma''\bot\Gamma') \vdash_\ell P' \quad \text{and} \quad \Gamma''\{x \leftarrow a\} \subseteq (\ell(d).l.(x)\Gamma''\bot\Gamma') \quad \text{and} \quad \Gamma' \vdash_\ell Q' \tag{5}$$

Since $\ell(d) \neq a$ from (5) we conclude $(\ell(d).l.(x)\Gamma'') \in dom(\Gamma)$ which completes the proof. ∎

Lemma B.7. *Let P be a well-typed process and Γ an event ordering such that $\Gamma \vdash_\ell P$. If $P \xrightarrow{(\nu a)c\ l^\downarrow!(a)} Q$ then there is Γ' such that $\Gamma \cup \Gamma' \vdash_\ell Q$ and $(\Gamma \cup \Gamma') \setminus a \subseteq \Gamma$ and $(c.l.(x)\Gamma'') \in dom(\Gamma)$ and $\Gamma''\{x \leftarrow a\} \subseteq ((c.l.(x)\Gamma'')\bot(\Gamma \cup \Gamma'))$.*

Proof. By induction on the derivation of the transition. We show the base cases of restriction open (Figure 3 (*Open*)) and $(\nu a)l^\downarrow!(a)$ transition originating from within a context piece.

($Case\ c \blacktriangleleft [P'] \xrightarrow{(\nu a)c\ l^\downarrow!(a)} c \blacktriangleleft [Q']$)

We have that

$$\Gamma \vdash_\ell c \blacktriangleleft [P'] \tag{1}$$

Let us consider

$$c \blacktriangleleft [P'] \xrightarrow{(\nu a)c\ l^\downarrow!(a)} c \blacktriangleleft [Q'] \tag{2}$$

We have that (1) is derived from

$$\Gamma \vdash_{(\ell(\downarrow),c)} P' \tag{3}$$

and (2) is derived from

$$P' \xrightarrow{(\nu a)l^\downarrow!(a)} Q' \tag{4}$$

From (3) and (4), considering Lemma B.6 we have there is Γ' such that

$$\Gamma \cup \Gamma' \vdash_{(\ell(\downarrow),c)} Q' \tag{5}$$

and $(\Gamma \cup \Gamma') \setminus a \subseteq \Gamma$ and

$$(c.l.(x)\Gamma'') \in dom(\Gamma) \quad \text{and} \quad \Gamma''\{x \leftarrow a\} \subseteq (c.l.(x)\Gamma''\bot(\Gamma \cup \Gamma')) \tag{6}$$

From (5) we conclude

$$\Gamma \cup \Gamma' \vdash_\ell c \blacktriangleleft [Q'] \tag{7}$$

which completes the proof for this case.

($Case$ $(\nu a)P' \xrightarrow{(\nu a)c\ l^{d}!(a)} Q'$)

We have that

$$\Gamma \vdash_\ell (\nu a)P' \tag{8}$$

Let us consider

$$(\nu a)P' \xrightarrow{(\nu a)c\ l^{\downarrow}!(a)} Q' \tag{9}$$

We have that (8) is derived from

$$\Gamma' \vdash_\ell P' \tag{10}$$

where $\Gamma = \Gamma' \setminus a$. (9) is derived from

$$P' \xrightarrow{c\ l^{\downarrow}!(a)} Q' \tag{11}$$

From (10) and (11), considering Lemma B.5 we have

$$(c.l.(x)\Gamma''\bot\Gamma') \vdash_\ell P' \quad \text{and} \quad \Gamma''\{x \leftarrow a\} \subseteq (c.l.(x)\Gamma''\bot\Gamma') \quad \text{and} \quad \Gamma' \vdash_\ell Q' \tag{12}$$

Since $c \neq a$ from (12) we conclude

$$(c.l.(x)\Gamma'') \in dom(\Gamma) \tag{13}$$

which completes the proof for this case. ∎

Lemma B.8. *Let P be a well-typed process and Γ an event ordering such that $\Gamma \vdash_\ell P$. If $P \xrightarrow{c\ \mathsf{this}} Q$ and $\ell(\downarrow) = c$ then $\Gamma \vdash_\ell Q$.*

Proof. By induction on the derivation of the transition. We show the base case when P is a **this** prefixed process.

($Case$ $\mathbf{this}(x).P' \xrightarrow{c\ \mathsf{this}} P'\{x \leftarrow c\}$)

We have that

$$\Gamma \vdash_\ell \mathbf{this}(x).P' \tag{1}$$

Let us consider

$$\mathbf{this}(x).P' \xrightarrow{c\ \mathsf{this}} P'\{x \leftarrow c\} \tag{2}$$

and $\ell(\downarrow) = c$. We have that (1) is derived from

$$\Gamma \cup \Gamma' \vdash_\ell P' \tag{3}$$

where $\Gamma'\{x \leftarrow \ell(\downarrow)\} \subseteq \Gamma$, hence $\Gamma'\{x \leftarrow c\} \subseteq \Gamma$. From Lemma 4.4 we then have

$$\Gamma \vdash_\ell P'\{x \leftarrow c\} \tag{4}$$

which completes the proof for this case. ∎

Proof of Theorem 4.5 (Preservation of Event Ordering)

Let P be a process and Γ an event ordering such that $\Gamma \vdash_\ell P$. If $P \to Q$ then $\Gamma \vdash_\ell Q$.

Proof. By induction on the length of the derivation of the reduction. We show the case for a message exchanged at the level of the current conversation, and the case of a message exchanged at the level of the current conversation carrying a bound name.

(*Case* $P_1 \xrightarrow{l^d!(a)} Q_1$ *and* $P_2 \xrightarrow{l^d?(a)} Q_2$)

We have that

$$P_1 \mid P_2 \to Q_1 \mid Q_2 \quad \text{and} \quad \Gamma \vdash_\ell P_1 \mid P_2 \tag{1}$$

From (1) we have that

$$\Gamma \vdash_\ell P_1 \quad \text{and} \quad \Gamma \vdash_\ell P_2 \tag{2}$$

(1) is derived from

$$P_1 \xrightarrow{l^d!(a)} Q_1 \quad \text{and} \quad P_2 \xrightarrow{l^d?(a)} Q_2 \tag{3}$$

From Lemma B.4 and (2) and (3) we have

$$(\ell(d).l.(x)\Gamma') \in dom(\Gamma) \quad \text{and} \quad \Gamma'\{x \leftarrow a\} \subseteq ((\ell(d).l.(x)\Gamma')\bot\Gamma) \quad \text{and} \quad \Gamma \vdash_\ell Q_1 \tag{4}$$

From Lemma B.2 and (2) and (3) and (4) we have

$$\Gamma \vdash_\ell Q_2 \tag{5}$$

From (4) and (5) we have

$$\Gamma \vdash_\ell Q_1 \mid Q_2 \tag{6}$$

which completes the proof for this case.

(*Case* $P_1 \xrightarrow{(\nu a)l^d!(a)} Q_1$ *and* $P_2 \xrightarrow{l^d?(a)} Q_2$)

We have that

$$P_1 \mid P_2 \to (\nu a)(Q_1 \mid Q_2) \quad \text{and} \quad \Gamma \vdash_\ell P_1 \mid P_2 \tag{7}$$

From (7) we have that

$$\Gamma \vdash_\ell P_1 \quad \text{and} \quad \Gamma \vdash_\ell P_2 \tag{8}$$

(7) is derived from

$$P_1 \xrightarrow{(\nu a)l^d!(a)} Q_1 \quad \text{and} \quad P_2 \xrightarrow{l^d?(a)} Q_2 \tag{9}$$

From Lemma B.6 and (8) and (9) we have there is Γ' such that

$$\Gamma \cup \Gamma' \vdash_\ell Q_1 \quad \text{and} \quad (\Gamma \cup \Gamma') \setminus a \subseteq \Gamma \quad \text{and} \quad (\Gamma(\ell(d).l).(x)\Gamma'') \in dom(\Gamma) \tag{10}$$

and

$$\Gamma''\{x \leftarrow a\} \subseteq (\Gamma(\ell(d).l).(x)\Gamma'')\bot(\Gamma \cup \Gamma') \tag{11}$$

From Lemma B.1 and (8), since (10) gives us that $\Gamma \cup \Gamma'$ is a well founded order, we have

$$\Gamma \cup \Gamma' \vdash_\ell P_2 \tag{12}$$

From Lemma B.2 and (12) and (9) and (10) and (11) we have

$$\Gamma \cup \Gamma' \vdash_\ell Q_2 \tag{13}$$

From (10) and (13) we have

$$\Gamma \cup \Gamma' \vdash_\ell Q_1 \mid Q_2 \tag{14}$$

From (14) and (10) we conclude

$$\Gamma \vdash_\ell (\nu a)(Q_1 \mid Q_2) \tag{15}$$

which completes the proof for this case. ∎

We introduce some notions auxiliary to the proof of Theorem 4.7.

Notation B.9. *We say M^w is an initial message type of T if $T \equiv \oplus\{\widetilde{B}_1; p\, l^d(C).B; \widetilde{B}_2\} \mid T'$ and $M = p\, l^d(C)$ and $w = d$, or $T \equiv \&\{\widetilde{B}_1; p\, l^d(C).B; \widetilde{B}_2\} \mid T'$ and $M = p\, l^d(C)$ and $w = d$, or $T \equiv c : [\oplus\{\widetilde{B}_1; M.B; \widetilde{B}_2\}] \mid T'$ and $w = c$, or $T \equiv c : [\&\{\widetilde{B}_1; M.B; \widetilde{B}_2\}] \mid T'$ and $w = c$.*

Notation B.10. *We denote by $\mathrm{p}_1\, l_1^d(C_1)^d \prec_\Gamma^\ell \mathrm{p}_2\, l_2^{d'}(C_2)^{d'}$ that $\ell(d).l_1.(x)\Gamma' \prec_\Gamma \ell(d').l_2.(x)\Gamma''$ and by $\mathrm{p}_1\, l_1^d(C_1)^a \prec_\Gamma^\ell \mathrm{p}_2\, l_2^{d'}(C_2)^b$ that $a.l_1.(x)\Gamma' \prec_\Gamma b.l_2.(x)\Gamma''$.*

Notation B.11. *We denote by $P \xrightarrow{M^w} P'$ a transition $P \xrightarrow{\lambda} P'$ such that either:*

- *$\lambda = l^d!(a)$ (or $\lambda = (\nu a)l^d!(a)$) and $M = !\, l^d(C)$ and $w = d$ for some a, C*
- *$\lambda = l^d?(a)$ and $M = ?\, l^d(C)$ and $w = d$ for some a, C*
- *$\lambda = c\ l^\downarrow!(a)$ (or $\lambda = (\nu a)c\ l^\downarrow!(a)$) and $M = !\, l^\downarrow(C)$ and $w = c$ for some a, C*
- *$\lambda = c\ l^\downarrow?(a)$ and $M = ?\, l^\downarrow(C)$ and $w = c$ for some a, C*
- *$\lambda = \tau$ and $M = \boldsymbol{\tau}\, l^d(C)$*

Notation B.12. *We say M^w is an minimal initial message of T w.r.t. Γ, ℓ if M^w is an initial message of T and there is no $M'^{w'}$ initial message type of T such that $M'^{w'} \prec_\Gamma^\ell M^w$.*

Lemma B.13. *Let P be a well-typed process. If $P \xrightarrow{c\ \sigma} P'$ and $P \xrightarrow{\overline{\sigma}} P''$ then $P \xrightarrow{c\ \texttt{this}} P'''$.*

Proof. Follows by induction on the structure of P in expected lines. ∎

Lemma B.14. *Let P be a process such that $P :: T$ and there is T' such that closed$(T \bowtie T')$ and $\Gamma \vdash_\ell P$ and M_1^w and $M_2^{w'}$ initial messages of T. If $M_1^w \prec_\Gamma^\ell M_2^{w'}$ and $M_1 = ?\, l_1^d(C_1)$ and $l_1 \in \mathcal{L}_\star$ and $M_2 = \boldsymbol{\tau}\, l_2^d(C_2)$ then there is $\boldsymbol{\tau}\, l_1^d(C_1)$ initial of T.*

Proof. By induction on the length of the derivation of $P :: T$ following expected lines. Notice that shared inputs expose a shared message interface (outputs on shared labels) and hence any greater τ initial message type is introduced by an output that matches the shared input, which type is initial and minimal since the shared input is initial and minimal. ∎

Notation B.15. *We say process P is final if it has no active* **this** *prefixes, hence if there is no $\mathcal{C}$ and Q such that $P = \mathcal{C}[\mathbf{this}(x).P]$.*

Lemma B.16. *Let P be a final process such that $P :: T$ and $\Gamma \vdash_\ell P$ and M^w a minimal initial message of T. Then there is P' such that either $P \to P'$ or $P \xrightarrow{M^w} P'$.*

Proof. By induction on the length of the derivation of $P :: T$. We show the cases when the last rule applied is (*Par*), (*Res*) and (*Output*).

(*Case* (*Par*))

We have that:

$$P_1 \mid P_2 :: T_1 \bowtie T_2 \ \ (i) \qquad \text{and} \qquad \Gamma \vdash_\ell P_1 \mid P_2 \ \ (ii) \tag{1}$$

and that M^w is a minimal initial message of $T_1 \bowtie T_2$. (1)(i) is derived from:

$$P_1 :: T_1 \ \ (i) \qquad \text{and} \qquad P_2 :: T_2 \ \ (ii) \tag{2}$$

(1)(ii) is derived from:

$$\Gamma \vdash_\ell P_1 \ \ (i) \qquad \text{and} \qquad \Gamma \vdash_\ell P_2 \ \ (ii) \tag{3}$$

Since M^w is a minimal initial message of $T_1 \bowtie T_2$ we have that either (*1*) M^w is a minimal initial message of T_1 or (*2*) M^w is a minimal initial message of T_2 or (*3*) $M^w = \tau\, l^d(C)^w$ and $?\, l^d(C)^w$ is a minimal initial message of T_i and $!\, l^d(C)^w$ is a minimal initial message of T_j for $\{i, j\} = \{1, 2\}$.

(*Case* (*1*)) By induction hypothesis on (2)(i) and (3)(i) and (*1*) we have that there is P_1' such that either:

$$P_1 \to P_1' \qquad \text{or} \qquad P_1 \xrightarrow{M^w} P_1' \tag{4}$$

We then directly have that:

$$P_1 \mid P_2 \to P_1' \mid P_2 \qquad \text{or} \qquad P_1 \mid P_2 \xrightarrow{M^w} P_1' \mid P_2 \tag{5}$$

respectively, which completes the proof for this case.

(*Case* (*2*)) Analogous to (*1*).

(*Case* (*3*)) Let us consider $i = 1$ and $j = 2$, hence, $?\, l^d(C)^w$ is a minimal initial message of T_1 and $!\, l^d(C)^w$ is a minimal initial message of T_2. By induction hypothesis on (2)(i) and (3)(i) and $?\, l^d(C)^w$ is a minimal initial message of T_1 we conclude that either:

$$P_1 \to P_1' \ \ (i) \qquad \text{or} \qquad P_1 \xrightarrow{?\, l^d(C)^w} P_1' \ \ (ii) \tag{6}$$

Likewise by induction hypothesis on (2)(ii) and (3)(ii) and $!\,l^d(C)^w$ is a minimal initial message of T_2 we conclude that either:

$$P_2 \to P_2' \;\; (i) \qquad \text{or} \qquad P_2 \xrightarrow{!\,l^d(C)^w} P_2' \;\; (ii) \tag{7}$$

If either (6)(i) or (7)(i) we have that:

$$P_1 \mid P_2 \to P' \tag{8}$$

and the proof is complete. If (6)(ii) and (7)(ii) we derive:

$$P_1 \mid P_2 \to P' \tag{9}$$

and, given that $M^w = \tau\, l^d(C)^w$, we conclude:

$$P_1 \mid P_2 \xrightarrow{M^w} P' \tag{10}$$

which completes the proof for this case.

(*Case* (*Res*))

We have that:

$$(\boldsymbol{\nu} a)P :: T \;\; (i) \qquad \text{and} \qquad \Gamma \vdash_\ell (\boldsymbol{\nu} a)P \;\; (ii) \tag{11}$$

and that M^w is a minimal initial message of T. (11)(i) is derived from:

$$P :: T \mid a : [B] \tag{12}$$

and $closed(B)$. (11)(ii) is derived from:

$$\Gamma' \vdash_\ell P \tag{13}$$

and $\Gamma = \Gamma' \setminus a$. Since M^w is a minimal initial message of T we have that either (*1*) M^w is a minimal initial message of $T \mid a : [B]$ (w.r.t. Γ', ℓ) or (*2*) there is M'^a minimal initial message of $T \mid a : [B]$ (w.r.t. Γ', ℓ). If (*1*) then the result follows from induction hypothesis. If (*2*) we have, since $closed(B)$, that $M'^a = \tau\, l^{\downarrow}(C)^a$. By induction hypothesis on (12) and (13) and (*2*) we conclude:

$$P \to P' \qquad \text{or} \qquad P \xrightarrow{M'^a} P' \tag{14}$$

Since $M'^a = \tau\, l^{\downarrow}(C)^a$ we conclude

$$P \to P' \tag{15}$$

and hence:

$$(\boldsymbol{\nu} a)P \to (\boldsymbol{\nu} a)P' \tag{16}$$

which completes the proof for this case.

(*Case* (*Output*))

We have that:

$$l^d!(n).P :: (L \bowtie n : C) \mid \oplus_{i \in I}\{M_i.B_i\} \ \ (i) \qquad \text{and} \qquad \Gamma \vdash_\ell l^d!(n).P \ \ (ii) \tag{17}$$

and there is $j \in I$ such that $M_j.B_j = \,!\,l^d(C).B$ and that M^w is a minimal initial message of $(L \bowtie n : C) \mid \oplus_{i\in I}\{M_i.B_i\}$. (17)($i$) is derived from:

$$P :: L \mid B \tag{18}$$

(17)(ii) is derived from:

$$(\ell(d).l.(x)\Gamma' \bot \Gamma) \vdash_\ell P \qquad \text{and} \qquad \Gamma'\{x \leftarrow n\} \subseteq (\ell(d).l.(x)\Gamma' \bot \Gamma) \tag{19}$$

From (18) and (19) we conclude $M^w = \,!\,l^d(C)^d$ and we have that:

$$l^d!(n).P \xrightarrow{!\,l^d(C)^d} P \tag{20}$$

which completes the proof for this case. ∎

Lemma B.17. *Let P be final process such that $P :: T$, where closed(T), and $\Gamma \vdash_{(z',z)} P$. If P is not a finished process then there are $\mathcal{C}, Q, T', l, d, C$ such that $P = \mathcal{C}[Q]$ and $Q :: T'$ and $\tau\, l^d(C)^w$ is minimal to T'.*

Proof. Follows by induction on the length of the derivation of $P :: T$ in expected lines. Since *closed*(T) all message types in T are either of polarity τ or are of polarity ? and defined on a shared label. If all initial message types are of the second type then the process is finished, otherwise if there is an initial τ message type which is not minimal then Lemma B.14 gives us there is a minimal initial message type. ∎

Proof of Theorem 4.7 (Progress)

Let P be a well-typed process such that $P :: T$, where closed(T), and Γ an event ordering such that $\Gamma \vdash_{(z',z)} P$. If P is not a finished process then $P \to Q$.

Proof. Follows directly from Lemma B.16 and Lemma B.17. If P is has an active **this** prefix we directly have that $P \to P'$. Otherwise P is final and Lemma B.17 gives us that there is a minimal initial τ message type and hence from Lemma B.16 we conclude $P \to P'$. ∎