A Typed Model for Multiparty Conversations

Luís Caires and Hugo Torres Vieira
CITI / Departamento de Informática, FCT Universidade Nova de Lisboa, Portugal

July 15, 2008
Revised October 21, 2008

Abstract

We present a type theory for analyzing concurrent multiparty interactions as found in service-oriented computing, using the Conversation Calculus as underlying model. 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.

1 Introduction

While most issues and questions 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, 3, 4, 5, 7, 6, 9, 10, 16, 19]). In previous work [25] we introduced the Conversation Calculus, a process 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, possible 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 and leave 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 through the access point with the caller context.

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 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 existing approaches for modeling and typing of service-based systems.

1.1 Conversation Contexts and Conversation Types

We explain the key ideas behind our development by going through some motivating examples. In the core Conversation Calculus (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. Consider the following composition of two conversation contexts, named *Buyer* and *Seller*, modeling a typical service collaboration:

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

The code in *Buyer* starts a new conversation by calling service `startBuy` located at *Seller* using service instantiation idiom **new** $\mathit{Seller}\cdot\texttt{startBuy} \Leftarrow \texttt{buy}!(\mathit{prod}).\texttt{price}?(v)$. The code $\texttt{buy}!(\mathit{prod}).\texttt{price}?(v)$ describes the participation 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

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

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

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

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 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).

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 may be described by the conversation type

$$\mathit{BSChat} \triangleq \boldsymbol{\tau}\texttt{buy}(P).\boldsymbol{\tau}\texttt{price}(M)$$

specifying the two interactions that occur sequentially within the conversation c, first a message `buy` and after a message `price`.

The $\boldsymbol{\tau}$s in e.g. $\boldsymbol{\tau}\texttt{buy}(P)$ means that the interaction is internal. A declaration such as $\boldsymbol{\tau}\texttt{buy}(P)$ is like an assertion such as $\texttt{buy}(P) : \mathit{Buyer} \to \mathit{Seller}$ in a message sequence chart, or in a global type [16], 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, 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 many others) the projection

$$\mathit{BSChat} = ?\,\texttt{buy}(P).!\,\texttt{price}(M) \bowtie !\,\texttt{buy}(P).?\,\texttt{price}(M)$$

The type $\texttt{buy}?(P).\texttt{price}!(M)$ will be used to type the *Buyer* participation, and the type $\texttt{buy}!(P).\texttt{price}?(M)$ 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 [14, 15]. The notion of conversation builds on the fundamental concept of session (introduced in [14, 15]) but extends it along unexplored directions, as we now discuss. Consider a three-party variation (from [9]) of the example above:

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

The role of *Shipper* is to inform the client on the delivery details of the chosen product. 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 *Buyer* and *Seller* in the fresh conversation. After that, *Shipper* is asked by *Seller*, using the idiom $\textbf{join}\ \mathit{Shipper} \cdot \texttt{newDelivery} \Leftarrow \cdots$, to join the ongoing conversation (till then involving only *Buyer* and *Seller*). The system then evolves to

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

Notice that *Seller does not lose access* to the conversation after asking service $\mathit{Shipper} \cdot \texttt{newDelivery}$ to join in the current conversation a (cf. a form of partial session delegation). In fact *Seller* and *Shipper* will later on interact 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

$$\mathit{BSSChat} \quad \triangleq \quad \boldsymbol{\tau}\ \texttt{buy}(P).\boldsymbol{\tau}\ \texttt{price}(D).\boldsymbol{\tau}\ \texttt{product}(P).\boldsymbol{\tau}\ \texttt{details}(I)$$

BSSChat type may be depicted as the message sequence chart shown in Figure 1.

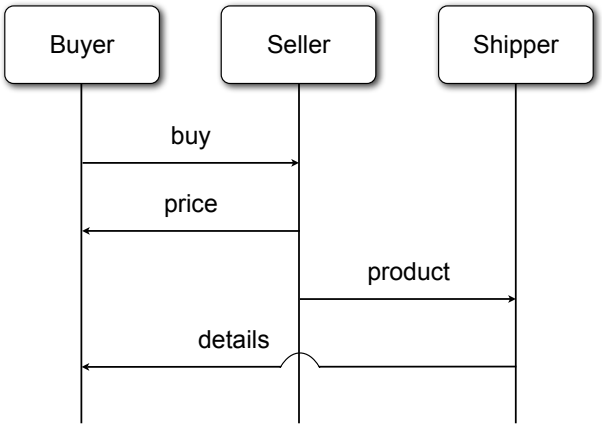

Figure 1: BSSChat Message Sequence Chart

We decompose type *BSSChat* in three "projections" (B_{bu}, B_{se}, and B_{sh}), by means of the *merge relation* $\bowtie$, first by $BSSChat = B_{bu} \bowtie B_{ss}$, and then by $B_{ss} = B_{se} \bowtie B_{sh}$, where

$$\begin{aligned} B_{bu} &\triangleq \ ?\ \texttt{buy}(P).!\ \texttt{price}(D).!\ \texttt{details}(I) \\ B_{ss} &\triangleq \ !\ \texttt{buy}(P).?\ \texttt{price}(D).\boldsymbol{\tau}\ \texttt{product}(P).?\ \texttt{details}(I) \\ B_{se} &\triangleq \ !\ \texttt{buy}(P).?\ \texttt{price}(D).?\ \texttt{product}(P) \\ B_{sh} &\triangleq \ !\ \texttt{product}(P).?\ \texttt{details}(I) \end{aligned}$$

These various "local" types are merged by our type system in a compositional way, allowing e.g., service `startBuy` to be assigned type $!\texttt{startBuy}([B_{ss}])$, 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 $\boldsymbol{\tau}\ \texttt{product}(I).?\ \texttt{details}(I)$. At this stage, extrusion of the conversation name a to service $Seller \cdot \texttt{newDelivery}$ will occur, to enable *Shipper* to join in. Notice that the global conversation will nevertheless be respected, since the conversation fragment delegated to *Shipper* is typed $!\ \texttt{product}(P).?\ \texttt{details}(I)$ while the conversation fragment still retained by *Seller* is typed $?\ \texttt{product}(P)$. To further illustrate the flexibility of our approach, we consider another variation of our running example. Now, *Buyer* does not need to know about the price, and instead it is *Shipper* that requires the pricing information.

$$\begin{aligned} &Buyer \blacleft [\ \textbf{new}\ Seller \cdot \texttt{startBuy} \Leftarrow \texttt{buy}!(prod).\texttt{details}?(d)\ ] \quad | \\ &Seller \blacleft [\ PriceDB \mid \textbf{def}\ \texttt{startBuy} \Rightarrow \texttt{buy}?(prod). \\ &\qquad\qquad\qquad\qquad \texttt{askPrice}^{\uparrow}!(prod).\texttt{readVal}^{\uparrow}?(p). \\ &\qquad\qquad\qquad\qquad \textbf{join}\ Shipper \cdot \texttt{newDelivery} \Leftarrow \\ &\qquad\qquad\qquad\qquad\qquad \texttt{price}!(p).\texttt{product}!(prod)\ ] \quad | \\ &Shipper \blacleft [\ \textbf{def}\ \texttt{newDelivery} \Rightarrow \texttt{price}?(v).\texttt{product}?(p).\texttt{details}!(data)\ ] \end{aligned}$$

When compared with the previous example, the roles of the participants have changed. However, the global conversation type is the same (*BSSChat*), just the decomposition into individual types is realized in a different way. Since our types

abstract away from participant identities, we allow roles in a given conversations to be implemented in various ways (cf. loose-coupling). It is even possible to type a system with an unbounded number of different participants, as needed to type e.g., a broker, or registry for services of a given type.

Our type system combines techniques from linear, behavioral, session and spatial types (see [8, 15, 17, 18]): 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 ($\boldsymbol{\tau}$). Key technical ingredients in our approach to conversation types are the uniform treatment of global types and of local types (in the sense introduced by [16]) 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 $\boldsymbol{\tau}$-free B there are types $\overline{B}, B'$ such that $B \bowtie \overline{B} = \boldsymbol{\tau}(B')$, 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 and formalize a new notion of conversation type. Conversation types are a powerful 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 (Theorem 3.18): this also implies that well-typed systems enjoy a conversation fidelity property. Finally, we present provably correct techniques to establish progress of systems with several interleaved conversations (Theorem 4.6), exploiting the combination of conversation names with message labels in event orderings, and, more crucially, propagation of orderings in communications.

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 extends the π-calculus [21] static fragment 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, and assume the expected extension in examples as needed.

The syntax of the calculus is defined 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 $\mathsf{rec}\, \mathcal{X}.P$. The

$$
\begin{array}{lcll}
a, b, c, \ldots & \in & \Lambda & (\textit{Names}) \\
x, y, z, \ldots & \in & \mathcal{V} & (\textit{Variables}) \\
n, m, o \ldots & \in & \Lambda \cup \mathcal{V} & \\
l, s \ldots & \in & \mathcal{L} & (\textit{Labels}) \\
\mathcal{X}, \mathcal{Y}, \ldots & \in & \chi & (\textit{Process Vars}) \\
 & & & \\
P, Q & ::= & \mathbf{0} & (\textit{Inaction}) \\
 & | & P \mid Q & (\textit{Parallel Composition}) \\
 & | & (\boldsymbol{\nu} a)P & (\textit{Name Restriction}) \\
 & | & \mathbf{rec}\, \mathcal{X}.P & (\textit{Recursion}) \\
 & | & \mathcal{X} & (\textit{Variable}) \\
 & | & n \blacktriangleleft [P] & (\textit{Conversation Access}) \\
 & | & \Sigma_{i \in I}\ \alpha_i.P_i & (\textit{Prefix Guarded Choice}) \\
 & & & \\
d & ::= & \downarrow \mid \uparrow & (\textit{Directions}) \\
\alpha & ::= & l^d!(n) & (\textit{Output}) \\
 & | & l^d?(x) & (\textit{Input}) \\
 & | & \mathbf{this}(x) & (\textit{Conversation Awareness})
\end{array}
$$

Figure 2: The Core Conversation Calculus.

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., $\texttt{askPrice}^{\uparrow}!(\textit{prod})$) 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 ($\mathit{fn}(P)$) and bound ($\mathit{bn}(P)$) names, free variables ($\mathit{fv}(P)$), and free process variables ($\mathit{fpv}(P)$) in a process P are defined as usual. We implicitly identify α-equivalent processes.

The operational semantics of the core CC is defined by a labeled transition system. For clarity, we split the presentation in two sets of rules, one (in Figure 3) containing the rules for the basic operators, which are essentially identical to the corresponding ones in the π-calculus (see [23]), and the other (in Figure 4) grouping

$$l^d!(a).P \xrightarrow{l^d!(a)} P \ (out) \qquad l^d?(x).P \xrightarrow{l^d?(a)} P\{x/a\} \ (inp)$$

$$\frac{\alpha_j.P_j \xrightarrow{\lambda} Q \quad j \in I}{\Sigma_{i\in I}\ \alpha_i.P_i \xrightarrow{\lambda} Q}(sum) \qquad \frac{P \xrightarrow{(\nu a)\overline{\lambda}} P' \quad Q \xrightarrow{\lambda} Q'}{P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu} a)(P' \mid Q')}(clo)$$

$$\frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\overline{\lambda}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}(com)$$

$$\frac{P \xrightarrow{\lambda} Q \quad a = out(\lambda)}{(\boldsymbol{\nu} a)P \xrightarrow{(\nu a)\lambda} Q}(opn) \qquad \frac{P \xrightarrow{\lambda} Q \quad a \notin na(\lambda)}{(\boldsymbol{\nu} a)P \xrightarrow{\lambda} (\boldsymbol{\nu} a)Q}(res)$$

$$\frac{P \xrightarrow{\lambda} Q}{P \mid R \xrightarrow{\lambda} Q \mid R}(par) \qquad \frac{P\{\mathcal{X}/\mathbf{rec}\,\mathcal{X}.P\} \xrightarrow{\lambda} Q}{\mathbf{rec}\,\mathcal{X}.P \xrightarrow{\lambda} Q}(rec)$$

Figure 3: Basic Operators (π-calculus).

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

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

$$\mathbf{this}(x).P \xrightarrow{c\ \mathtt{this}} P\{x/c\} \ (thi) \qquad \frac{P \xrightarrow{c\ \mathtt{this}} Q}{c \blacktriangleleft [P] \xrightarrow{\tau} c \blacktriangleleft [Q]}(thl)$$

$$\frac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{\overline{c\ \sigma}} Q'}{P \mid Q \xrightarrow{c\ \mathtt{this}} P' \mid Q'}(tco) \qquad \frac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{(\nu a)\overline{c\ \sigma}} Q'}{P \mid Q \xrightarrow{c\ \mathtt{this}} (\boldsymbol{\nu} a)(P' \mid Q')}(tcl)$$

Figure 4: Conversation Operators.

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. Transition labels (λ) and actions (σ) are given by

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

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; these correspond to the basic actions a process may perform in the context of a given conversation. 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 $\overline{l^d!(a)} = l^d?(a)$ and $\overline{l^d?(a)} = l^d!(a)$. We use $fn(\lambda)$ and $bn(\lambda)$ to denote (respectively) the free and bound names of a transition label, and $na(\lambda)$ to denote all names (both free and bound) of a transition label.

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 (*opn*) 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 $c \neq a$.

It would be useful however to discuss the intuitions behind the rules for conversation contexts (Figure 4). In rule (*her*) an $\uparrow$ directed message (to the caller conversation) becomes $\downarrow$ (in the current conversation), after passing through the conversation access boundary. We note by λ^d a transition label λ^d 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 $\texttt{askPrice}^\uparrow?(a)$ then $\lambda^\downarrow$ is $\texttt{askPrice}^\downarrow?(a)$).

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 (*thi*) a `this` label reads the 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 (*thl*), where a `this` label matches the enclosing conversation.

In rule (*thr*) an already located communication label transparently crosses some other conversation boundary, and likewise for a τ label in (*tau*). In rules (*tco*) and (*tcl*) an unlocated communication matches a communication located at c, originating a c `this` label, thus ensuring the interaction to occur inside the given conversation c, as required.

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

Definition 2.1 (Reduction) *The relation of reduction on processes, noted $P \rightarrow Q$, is defined as $P \xrightarrow{\tau} Q$.*

Although we will not exploit in this paper the behavioral semantics but rather the static semantics of our calculus, it is worthwhile reporting that the standard notion of bisimilarity, defined in terms of the above presented labeled transition system, has pleasant structural properties, establishing that all the constructs of our calculus may be soundly interpreted as compositional semantic operators on bisimilarity equivalence classes.

Definition 2.2 (Strong Bisimulation) *A (strong) bisimulation is a symmetric binary relation $\mathcal{R}$ on processes such that, for all processes P and Q, if $P\mathcal{R}Q$, we have:*

If $P \xrightarrow{\lambda} P'$ and $bn(\lambda) \cap fn(Q) = \emptyset$ then there is Q' such that
$Q \xrightarrow{\lambda} Q'$ and $P'\mathcal{R}Q'$.

We denote by $\sim$ (strong bisimilarity) the largest bisimulation.

Theorem 2.3 *Strong bisimilarity is a congruence.*

Proof. Direct from the proof of [24] Theorem 5.2. The core Conversation Calculus extends a subcalculus of the Conversation Calculus of [24] with the summation construct. For the summation we consider:

- If $\alpha_i.P_i \sim \alpha_i.Q_i$, for $i \in I$, then $\Sigma_{i\in I}\, \alpha_i.P_i \sim \Sigma_{i\in I}\, \alpha_i.Q_i$.

which proof follows the lines of the other axioms of [24] Theorem 5.2. ∎

N.B. For input, we consider the universal instantiation congruence principle: if $P\{x/a\} \sim Q\{x/a\}$ for all a then $l^d?(x).P \sim l^d?(x).Q$ (cf., [23] Theorem 2.2.8(2)). Like for the π-calculus, congruence does not hold for input, if input congruence is interpreted as a first order algebraic congruence. We may show interesting behavioral (in)equations, that confirm basic intuitions about our conversation-based communication model.

1. $n \blacktriangleleft [P] \mid n \blacktriangleleft [Q] \sim n \blacktriangleleft [P \mid Q]$.
2. $n \blacktriangleleft [m \blacktriangleleft [P] \mid Q] \not\sim m \blacktriangleleft [P] \mid n \blacktriangleleft [Q]$.
3. $m \blacktriangleleft [n \blacktriangleleft [o \blacktriangleleft [P]]] \sim n \blacktriangleleft [o \blacktriangleleft [P]]$.

Proof. (1) and (3) follow from [24] Proposition 5.4 (2) and (3). We show a counterexample to prove (2). Consider process P defined as $P \triangleq l^{\uparrow}(a)$ and Q defined as $Q \triangleq l^{\downarrow}(x)$, yielding processes P_1 and P_2 for the lhs and the rhs of the inequation, respectively, defined as

$$P_1 \triangleq n \blacktriangleleft \left[m \blacktriangleleft \left[l^{\uparrow}(a)\right] \mid l^{\downarrow}(x)\right] \qquad\qquad P_2 \triangleq m \blacktriangleleft \left[l^{\uparrow}(a)\right] \mid n \blacktriangleleft \left[l^{\downarrow}(x)\right]$$

Process P_1 has a τ transition ($P_1 \xrightarrow{\tau} n \blacktriangleleft [m \blacktriangleleft [\mathbf{0}] \mid \mathbf{0}]$) while process P_2 has no τ transitions, hence $P_1 \not\sim P_2$. ∎

(1) captures the notion of conversation context as a single medium accessible through distinct pieces. (2) contrasts with (1): the relation between a conversation and its caller must be preserved. On the other hand, (3) expresses the fact that P may only interact in its conversation and in the caller one (via $\uparrow$ communications).

2.1 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 [25]), and the conversation join construct, which is crucial to our approach to multiparty conversations. These 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.,

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

Figure 5: Service Idioms.

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

A new instance of a service s is created by $\textbf{new}\ n \cdot s \Leftarrow Q$, 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 [15]. Typically, service definitions may also be replicated, written $\star\textbf{def}\ s \Rightarrow P$, 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 $\textbf{this}(x).P$), 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 $\textbf{join}\ n \cdot s \Leftarrow Q$ 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.

$$
\begin{array}{l}
n \blacktriangleleft [\textbf{def}\ s_1 \Rightarrow P] \mid \cdots \mid m \blacktriangleleft [\textbf{new}\ n \cdot s_1 \Rightarrow Q] \quad \rightarrow \\
\qquad\qquad (\boldsymbol{\nu} c)(n \blacktriangleleft [c \blacktriangleleft [P]] \mid \cdots \mid m \blacktriangleleft [c \blacktriangleleft [Q]])
\end{array}
$$

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.

$$
\begin{array}{l}
o \blacktriangleleft [\textbf{def}\ s_2 \Rightarrow P] \mid \cdots \mid c \blacktriangleleft [\textbf{join}\ o \cdot s_2 \Rightarrow Q] \quad \rightarrow \\
\qquad\qquad o \blacktriangleleft [c \blacktriangleleft [P]] \mid \cdots \mid c \blacktriangleleft [Q]
\end{array}
$$

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 rest of the paper 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. Before closing the section, we invite the reader to revisit the examples presented in the Introduction, an then go through the next example in Figure 6. The Flight Booking Service example illustrates a familiar service composition scenario, involving a conversation with concurrent message exchanges and branching on choices. We assume an extension of the language with a standard **if** statement, and introduce anonymous contexts, defined as $[P] \triangleq (\boldsymbol{\nu} c)(c \blacktriangleleft [P])$.

An instance of the `flightBooking` service is expected to receive a `flightReq` message from the client and then reply a `flightResp` message after finding a suitable flight. The implementation of the service relies on subsidiary services provided by *alphaAir* and *deltaAir*, used to identify the best pricing. After reception of the `flightReq` message, the flight information is forwarded to the service instances, that will reply informing on the price. The prices are compared and a suitable response to the `flightBooking` service client will be sent back.

Notice that the body of the `flightBooking` service is built from three components: a new instance of $\mathit{alphaAir} \cdot \texttt{flightBook}$, a new instance of $\mathit{deltaAir} \cdot \texttt{flightBook}$, and a process starting with the $\texttt{flightReq}^{\uparrow}?(\mathit{flight})$ message, to which we will conventionally refer to as the "orchestration script". The instantiation of the two subsidiary services will result in the creation of two new conversation access pieces, that will act as local processes. The orchestration script coordinates the interaction between such local processes and the remote client of the `flightBooking` instance. The local process created by the instance of *alphaAir* will interact with the script by means of messages `reqA` and `replyA`, and the local process created by the instance of *deltaAir* will interact with the orchestration script by means of messages `reqD` and `replyD`.

These message exchanges form a loosely-coupled interaction between the orchestration script and the subsidiary service conversations, where the protocols provided in **new** $\mathit{alphaAir} \cdot \texttt{flightBook}$ and **new** $\mathit{deltaAir} \cdot \texttt{flightBook}$ take care of adapting the conversation protocol expected by the airline service providers to the messages required by the orchestration script (e.g., mapping message `reqA` to message `req`, etc).

After choosing the flight the `flightBooking` service instance informs the partner services of the choice taken, confirming or canceling the reservations accordingly. The partner service instances branch in the two possible behaviors, being set to receive either `bookA`/`bookD` or `cancelA`/`cancelD` messages, after which the service providers are notified accordingly.

```
⋆ def flightBooking ⇒ [
    new alphaAir · flightBook ⇐
      reqA↑?(flight).req!(flight).
      reply?(price).replyA↑!(price).
      (bookA↑?().book!() + cancelA↑?().cancel!())
   |
    new deltaAir · flightBook ⇐
      reqD↑?(flight).req!(flight).
      reply?(price).replyD↑!(price).
      (bookD↑?().book!() + cancelD↑?().cancel!())
   |
    flightReq↑?(flight).
     (reqA!(flight) | reqD!(flight)
      |
      replyA?(priceA).
      replyD?(priceD).
      if (priceD < priceA) then
        flightResp↑!(priceD).(bookD!() | cancelA!())
      else
        flightResp↑!(priceA).(bookA!() | cancelD!()))  ]
```

Figure 6: Flight Booking Service.

3 Type System

In this section we formally present our type system for the core CC. The syntax for the types is show in Figure 7. As already motivated in the Introduction, our types specify the message protocols that flow between and within conversations.

Typing judgments have the form $P :: T$, where T is a process type. A process type T is a type of the form $L \mid B$, where L is a located type and B is a behavioral type. A located type associates conversation types to conversation names; a conversation type C may then declare its intended local conversations, in terms of a behavioral type. Such a type judgement $P :: T$ intuitively states that if process P is placed in an environment that complies to type T, then one obtains a safe system. The intended safety property will be formally stated in Theorem 3.18: it implies conversations agree to declared protocols, and the absence of certain kind of runtime errors.

For behavioral types B we have the branch and the choice constructs ($\oplus_{i\in I}\{M_i.B_i\}$ and $\&_{i\in I}\{M_i.B_i\}$, 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. The prefix $M.B$ specifies a process that sends, receives, or internally exchanges a message M before proceeding with behavior B. Then we have parallel composition $B_1 \mid B_2$, inaction $\mathbf{0}$, and recursion. Message types M are specified by a polarity p (either output !, input ? or internal action $\boldsymbol{\tau}$), a pair label-direction l^d, and the type C of what is communicated in the message (for simplicity, we restrict to monadic messages). Notice that a message M may refer to an *internal* exchange between two

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

Figure 7: Syntax of Types

partners, if it is of the form $\boldsymbol{\tau} l^{\downarrow}(C)$ (such internal interactions are always specified in the local ("here") conversation). We write M for $M.\mathbf{0}$, and $p\,l(C)$ for $p\,l^{\downarrow}(C)$. Also, we write $\overline{B}$ for the dual type of B, obtained from B by swapping polarities ! and ?. We abbreviate both $\oplus\{M.B\}$ 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 (recursive processes are defined using shared labels).

Conversation types C are given by $[B]$, where B specifies the message interactions that may take place in the conversation. Located types L collect (using composition) type associations between conversation names and their types. A located type specifies the conversation type of each visible conversation. Recall that the unlocated parts of the types appearing in a typing judgment $P :: T$ refer to properties of the current (not yet located) conversation.

In order to type recursive processes we introduce $B\langle\mathcal{X}\rangle$, used in characterizing our admissible recursive types. First off, we denote by $B^\star$ a "shareable" behavioral type defined (exclusively) with shared labels (from $\mathcal{L}_\star$), hence not referring any plain label (from $\mathcal{L}_p$). We define $B^\star$ and $B\langle\mathcal{X}\rangle$.

Definition 3.1 *Shared behavioral types, noted $B^\star$, and recursive behavioral types, noted $B\langle\mathcal{X}\rangle$, are defined as follows:*

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

Essentially, type $B\langle\mathcal{X}\rangle$ is 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. Type $B\langle\mathcal{X}\rangle$ thus characterizes recursive processes that can safely have several active concurrent instances, whereby 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 $\mathcal{L}_p$, and hence do not interfere and are apart (apartness is formalized in Definition 3.10). We then use $B\langle M_1,\ldots,M_k,\mathcal{X}\rangle$ to refer to a type with $k+1$ leafs, where all M_i are defined with shared labels and with polarity ?.

Types are related by the subtyping relation $<:$ defined in Figure 10. The subtyping rules express expected relationships of types, such as the commutative monoid rules for $(- \mid -, \mathbf{0})$, congruence principles, and the split rule (5). We adopt an iso-recursive approach to recursive types [22], based on simple unfoldings of recursive

$$d(\mathbf{0}) \triangleq \mathbf{0} \qquad d(\mathcal{X}) \triangleq \mathcal{X} \qquad d(\mathtt{rec}\,\mathcal{X}.B) \triangleq \mathtt{rec}\,\mathcal{X}.d(B)$$

$$\begin{array}{lcll}
d(B_1 \mid B_2) & \triangleq & d(B_1) \mid d(B_2) & \\
d(\&_{i\in I}\{?\ l_i^d(C_i).B_i\}) & \triangleq & \&_{i\in I}\{?\ l_i^d(C_i).d(B_i)\} & \\
d(\&_{i\in I}\{?\ l_i^{d'}(C_i).B_i\}) & \triangleq & \&_{i\in I}\{d(B_i)\} & \\
 & & & \text{if } d \neq d' \text{ and } d(B_i) = ?\ l^d(C).B' \\
d(\oplus_{i\in I}\{p\ l_i^d(C_i).B_i\}) & \triangleq & \oplus_{i\in I}\{p\ l_i^d(C_i).d(B_i)\} & \\
d(\oplus_{i\in I}\{p\ l_i^{d'}(C_i).B_i\}) & \triangleq & \&_{i\in I}\{d(B_i)\} & \\
 & & & \text{if } d \neq d' \text{ and } d(B_i) = ?\ l^d(C).B' \\
d(\oplus_{i\in I}\{p\ l_i^{d'}(C_i).B_i\}) & \triangleq & \oplus_{i\in I}\{d(B_i)\} & \\
 & & & \text{if } d \neq d' \text{ and } d(B_i) = !\ l^d(C).B'
\end{array}$$

Figure 8: Direction Projection

type terms (4). We could also have adopted a more flexible theory of equi-recursive subtyping, along the lines of [13]. For types T_1 and T_2 we write $T_1 \equiv T_2$ if $T_1 <: T_2$ and $T_2 <: T_1$. A key subtyping principle is (12), that allows a behavioral type to be decomposed (in the subtype) in its two projections according to the message directions $\downarrow$ and $\uparrow$. An important subtyping principle is (14), that allows a message to be serialized (in the supertype). In (13) we use $\star M$ as an abbreviation of $\mathtt{rec}\,\mathcal{X}.M.\mathcal{X}$. In (16) we require that all M_i ($i \in J$) message types have the same direction (either $\uparrow$ or $\downarrow$). Notice that we allow width subtyping for branch type in (16) and do not allow width subtyping in choice type in (15). Essentially we can safely consider more branches in the branch type, while we can not forget some choices in the choice type, since this would allow undesired matches between choice and branch types: if the environment expected by a process does not fully reveal the options it might undertake, then when the process is placed in such an environment there might be unexpected behaviors, i.e., behaviors not described by the type (cf., [10] where a similar problem arises in contract compliance).

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.

Definition 3.2 *For each direction d, the projection $d(B)$ of type B along direction d is inductively defined in Figure 8.*

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".

Example 3.3 *We illustrate the projection of choice and branch types with an example. Consider type*

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

which specifies a process environment that can either output message `bookA` *or message* `cancelA` *in the enclosing conversation, and afterwards receive message* `book` *or message* `cancel`*, respectively. Projecting the type in the $\uparrow$ direction then results in the choice type of the two $\uparrow$ messages, as follows:*

$$
\begin{array}{lcl}
p(\mathbf{0}) \triangleq \mathbf{0} & & p(T_1 \mid T_2) \triangleq p(T_1) \mid p(T_2) \\
p(\mathcal{X}) \triangleq \mathcal{X} & & p(\mathtt{rec}\,\mathcal{X}.B) \triangleq \mathtt{rec}\,\mathcal{X}.p(B) \\
p(\&_{i\in I}\{p\ l_i^d(C_i).B_i\}) & \triangleq & \&_{i\in I}\{p\ l_i^d(C_i).p(B_i)\} \\
p(\oplus_{i\in I}\{p\ l_i^d(C_i).B_i\}) & \triangleq & \oplus_{i\in I}\{p\ l_i^d(C_i).p(B_i)\} \\
p(p'\ l^d(C).B) & \triangleq & p(B) \\
p(n : [B]) & \triangleq & n : [p(B)]
\end{array}
$$

Figure 9: Polarity Projection

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

On the other hand, the $\downarrow$ projection specifies that the process environment must branch in all $\downarrow$ behaviors of the continuations (since the first choice is invisible from this view), as follows:

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

If p is a polarity (!, ?, $\boldsymbol{\tau}$), we denote by $p(B)$ the projection type that selects all messages that have polarity p, while filtering out the ones of other polarities.

Definition 3.4 *For each polarity p, the projection $p(T)$ of type T along a polarity p is inductively defined in Figure 9.*

We may now present our typing rules in Figure 12. They rely on several auxiliary operations and predicates on types. The key ones are *apartness* $T_1 \# T_2$ (a predicate) and *merge* $T = T_1 \bowtie T_2$ (a relation). Intuitively, two types are apart when they may type subsystems that may be safely composed without undesirable interferences. To define apartness we introduce the set $\mathit{Labels}_{\mathcal{L}}(B)$.

Definition 3.5 *We denote by $\mathit{Labels}_{\mathcal{L}}(B)$ the set of message types defined with labels in $\mathcal{L}$ of a behavioral B, defined as follows:*

$$
\begin{array}{lcl}
\mathit{Labels}_{\mathcal{L}}(\mathbf{0}) \triangleq \emptyset \qquad \mathit{Labels}_{\mathcal{L}}(\mathcal{X}) \triangleq \emptyset \qquad \mathit{Labels}_{\mathcal{L}}(\mathbf{rec}\mathcal{X}.T) \triangleq \mathit{Labels}_{\mathcal{L}}(T) & & \\
\mathit{Labels}_{\mathcal{L}}(T_1 \mid T_2) & \triangleq & \mathit{Labels}_{\mathcal{L}}(T_1) \cup \mathit{Labels}_{\mathcal{L}}(T_2) \\
\mathit{Labels}_{\mathcal{L}}(\oplus_{i\in I}\{p\ l_i^d(C_i).B_i\}) & \triangleq & \{(p\,l_i^d(C_i)) : l_i \in \mathcal{L}\} \cup \bigcup_{i\in I} \mathit{Labels}_{\mathcal{L}}(B_i) \\
\mathit{Labels}_{\mathcal{L}}(\&_{i\in I}\{p\ l_i^d(C_i).B_i\}) & \triangleq & \{(p\,l_i^d(C_i)) : l_i \in \mathcal{L}\} \cup \bigcup_{i\in I} \mathit{Labels}_{\mathcal{L}}(B_i)
\end{array}
$$

For example, given some behavioral type B, $\mathit{Labels}_{\mathcal{L}_p}(B)$ is the set of all plain (in $\mathcal{L}_p$) message types ($p\,l^d(C)$) occurring in T, leaving out shared labels (those belonging to $\mathcal{L}_\star$). We define the set of directed labels of a behavioral type $\mathit{LLabels}_{\mathcal{L}}(T)$.

Definition 3.6 *We denote by $\mathit{LLabels}_{\mathcal{L}}(T)$ the set of directed labels from $\mathcal{L}$ of a behavioral type defined as:*

$$\mathit{LLabels}_{\mathcal{L}}(T) \triangleq \{l^d \ :\ (p\,l^d(C)) \in \mathit{Labels}_{\mathcal{L}}(T)\}$$

We then define $\mathit{Labels}(n, T)$ and $\mathit{LLabels}(n, T)$, which denote the set of message types and directed labels, respectively, located at n in type T.

$$T_1 \mid T_2 <:> T_2 \mid T_1 \ \ (1) \quad T \mid \mathbf{0} <:> T \ \ (2)$$

$$T_1 \mid (T_2 \mid T_3) <:> (T_1 \mid T_2) \mid T_3 \ \ (3)$$

$$\mathtt{rec}\, \mathcal{X}.T <:> T\{\mathcal{X}/\mathtt{rec}\, \mathcal{X}.T\} \ \ (4)$$

$$n : [B_1 \mid B_2] <:> n : [B_1] \mid n : [B_2] \ \ (5)$$

$$\frac{B_1 <: B_2}{M.B_1 <: M.B_2}(6) \qquad \frac{T_1 <: T_2}{T_3 \mid T_1 <: T_3 \mid T_2}(7)$$

$$\frac{C_1 <: C_2}{n : C_1 <: n : C_2}(8) \quad \frac{T_1 <: T_3 \quad T_3 <: T_2}{T_1 <: T_2}(9)$$

$$T <: T \ \ (10) \qquad \frac{B_1 <: B_2}{[B_1] <: [B_2]}(11)$$

$$\downarrow B \mid \uparrow B <: B \ \ (12)$$

$$\star M_1 \mid \ldots \mid \star M_k \mid \mathtt{rec}\, \mathcal{X}.B\langle \mathbf{0}, \ldots, \mathbf{0}, \mathcal{X}\rangle <: \mathtt{rec}\, \mathcal{X}.B\langle M_1, \ldots, M_k, \mathcal{X}\rangle \ \ (13)$$

$$M.B_1 \mid B_2 <: M.(B_1 \mid B_2) \ \ (M \ \# \ B_2) \ \ (14)$$

$$\frac{M_i.B_i <: M_i'.B_i' \ \ (i \in I)}{\oplus_{i\in I}\{M_i.B_i\} <: \oplus_{i\in I}\{M_i'.B_i'\}}(15)$$

$$\frac{M_i.B_i <: M_i'.B_i' \ \ (i \in I) \quad I \subseteq J}{\&_{i\in J}\{M_i.B_i\} <: \&_{i\in I}\{M_i'.B_i'\}}(16)$$

Figure 10: Subtyping Rules.

Definition 3.7 *We denote by $Labels(n, T)$ and by $LLabels(n, T)$ the set of message types and the set of directed labels, respectively, located at n in type T, defined as:*

$$\begin{aligned} Labels_{\mathcal{L}}(n, T) &\triangleq \{M \mid T \equiv T' \mid n : [B]\,, M \in Labels_{\mathcal{L}}(B)\} \\ LLabels_{\mathcal{L}}(n, T) &\triangleq \{l^d \ : \ (p\, l^d(C)) \in Labels_{\mathcal{L}}(n, T)\} \end{aligned}$$

Given behavioral types B_1 and B_2, we let $B_1 \asymp_u B_2$ state that message types with shared labels occur both B_1 and B_2 with identical argument types (so that B_1 and B_2 are compatible on shared labels).

Definition 3.8 *The conformance relation $B_1 \asymp_u B_2$ between behavioral types is defined as follows:*

$$\begin{aligned} T_1 \asymp_u T_2 \ \triangleq \ & \textit{if } (p_1\ l^d(C_1)) \in Labels_{\mathcal{L}_\star}(T_1) \\ & \textit{and } (p_2\ l^d(C_2)) \in Labels_{\mathcal{L}_\star}(T_2) \textit{ then } C_1 \equiv C_2 \\ & \textit{and either } p_1 = p_2 = ? \textit{ or } p_1 = p_2 = \boldsymbol{\tau} \textit{ or } p_i = ! \textit{ and } p_j = \boldsymbol{\tau} \end{aligned}$$

Conformance also characterizes the admissible shared message polarities, namely: two ! message types (introduced by two distinct input summations) are not conformant, ensuring an unique handling principle on inputs defined on shared labels; and that a ! message is not conformant with a ?, ensuring matching messages types must synchronize in a $\boldsymbol{\tau}$. We extend conformance to all (located and behavioral) types.

Definition 3.9 *The conformance relation $T_1 \asymp T_2$ between types is defined as follows:*

$$\begin{array}{rcl} T_1 \asymp T_2 & \triangleq & \textit{if } T_1 \equiv L_1 \mid B_1 \textit{ and } T_2 \equiv L_2 \mid B_2 \textit{ then} \\ & & B_1 \asymp_u B_2 \\ & & \textit{and for all } n. \textit{if } L_1 \equiv L_1' \mid n : [B_1'] \textit{ and } L_2 \equiv L_2' \mid n : [B_2'] \\ & & \textit{then } B_1' \asymp_u B_2' \end{array}$$

Type apartness is then defined by

Definition 3.10 *The apartness relation $T_1 \# T_2$ between types is defined as follows:*

$$\begin{array}{rcl} T_1 \# T_2 & \triangleq & T_1 \asymp T_2 \textit{ and } LLabels_{\mathcal{L}_p}(T_1) \# LLabels_{\mathcal{L}_p}(T_2) \\ & & \textit{and for all } n. LLabels_{\mathcal{L}_p}(n, T_1) \# LLabels_{\mathcal{L}_p}(n, T_2) \end{array}$$

N.B. We write $A \# B$ for $A \cap B = \emptyset$.

Essentially, apartness ensures disjointness of plain ("linear") types, and consistency of shared ("exponential") types, in the appropriate conversations (cf. [18]).

The merge relation is used to define merging of two types, so that if $T = T_1 \bowtie T_2$ then T is a particular (in general not unique) behavioral combination of the types T_1 and T_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 T such that $T = T_1 \bowtie T_2$. On the other hand, if some such T exists, we use $T_1 \bowtie T_2$ to non-deterministically denote any such T (e.g., in conclusions of type rules). Intuitively, $T = T_1 \bowtie T_2$ holds if T_1 and T_2 may safely synchronize or interleave so to produce behavioral type T.

Definition 3.11 *The merge relation $T = T_1 \bowtie T_2$, defined on located and behavioral types, is inductively defined in Figure 11.*

In rules (1) and (2) we require that l is a shared label; by $B\{?\ l^d(C)/\boldsymbol{\tau}\ l^d(C)\}$ we denote the type obtained by replacing all occurrences of $?\ l^d(C)$ by $\boldsymbol{\tau}\ l^d(C)$ in B. Instead in rules (3)-(6) the l_i labels must be plain. While shared labels synchronize and leave open the possibility for further synchronizations, expecting further outputs from the environment, plain message synchronization characterizes the uniquely determined synchronization on that plain label.

Merge $\bowtie$ thus allows $\boldsymbol{\tau}\ l^{\downarrow}$ plain message types ("here" internal interactions) to be separated into send ! and receive ? capabilities in respective choice and branch constructs. Also it is possible to decompose a sequence of prefixes Π, where Π abbreviates $M_1.(\ldots).M_k$, by interleaving with the continuation, if Π is apart from the prefixes of the behavior to be placed in parallel.

Example 3.12 *Consider type*

$$!\,\mathtt{buy}^{\downarrow}(F).?\,\mathtt{price}^{\downarrow}(F).!\,\mathtt{product}^{\downarrow}(D).?\,\mathtt{details}^{\downarrow}(D)$$

$$B\{?\ l^d(C)/\boldsymbol{\tau}\ l^d(C)\} \mid \star\,!\ l^d(C) \;=\; B \bowtie \star\,!\ l^d(C) \quad (1)$$

$$B\{?\ l^d(C)/\boldsymbol{\tau}\ l^d(C)\} \mid \star\,!\ l^d(C) \;=\; \star\,!\ l^d(C) \bowtie B \quad (2)$$

$$\Pi.\oplus_{i\in I}\{\boldsymbol{\tau}\ l_i^{\downarrow}(C_i).T_i\} \;=\; \Pi.\oplus_{i\in I}\{!\ l_i^{\downarrow}(C_i).T_i^+\} \bowtie \&_{i\in I}\{?\ l_i^{\downarrow}(C_i).T_i^-\} \quad (3)$$
$$\text{if } \Pi \,\#\, \&_{i\in I}\{?\ l_i^{\downarrow}(C_i)\} \text{ and } T_i = T_i^- \bowtie T_i^+$$

$$\Pi.\oplus_{i\in I}\{\boldsymbol{\tau}\ l_i^{\downarrow}(C_i).T_i\} \;=\; \oplus_{i\in I}\{!\ l_i^{\downarrow}(C_i).T_i^+\} \bowtie \Pi.\&_{i\in I}\{?\ l_i^{\downarrow}(C_i).T_i^-\} \quad (4)$$
$$\text{if } \Pi \,\#\, \oplus_{i\in I}\{!\ l_i^{\downarrow}(C_i)\} \text{ and } T_i = T_i^- \bowtie T_i^+$$

$$\Pi.\oplus_{i\in I}\{\boldsymbol{\tau}\ l_i^{\downarrow}(C_i).T_i\} \;=\; \Pi.\&_{i\in I}\{?\ l_i^{\downarrow}(C_i).T_i^+\} \bowtie \oplus_{i\in I}\{!\ l_i^{\downarrow}(C_i).T_i^-\} \quad (5)$$
$$\text{if } \Pi \,\#\, \oplus_{i\in I}\{!\ l_i^{\downarrow}(C_i)\} \text{ and } T_i = T_i^- \bowtie T_i^+$$

$$\Pi.\oplus_{i\in I}\{\boldsymbol{\tau}\ l_i^{\downarrow}(C_i).T_i\} \;=\; \&_{i\in I}\{?\ l_i^{\downarrow}(C_i).T_i^+\} \bowtie \Pi.\oplus_{i\in I}\{!\ l_i^{\downarrow}(C_i).T_i^-\} \quad (6)$$
$$\text{if } \Pi \,\#\, \&_{i\in I}\{?\ l_i^{\downarrow}(C_i)\} \text{ and } T_i = T_i^- \bowtie T_i^+$$

$$\mathtt{rec}\,\mathcal{X}.T \;=\; \mathtt{rec}\,\mathcal{X}.T^+ \bowtie \mathtt{rec}\,\mathcal{X}.T^- \quad (7)$$
$$\text{if } T = T^- \bowtie T^+$$

$$n : [T] \;=\; n : [T^+] \bowtie n : [T^-] \quad (8)$$
$$\text{if } T = T^- \bowtie T^+$$

$$T_1 \mid T_2 \;=\; T_1^+ \mid T_2^+ \bowtie T_1^- \mid T_2^- \quad (9)$$
$$\text{if } T_1 \,\#\, T_2 \text{ and } T_i = T_i^- \bowtie T_i^+$$

$$\mathcal{X} = \mathcal{X} \bowtie \mathcal{X} \quad (10) \qquad T = T \bowtie \mathbf{0} \quad (11) \qquad T = \mathbf{0} \bowtie T \quad (12)$$

Figure 11: Merge Relation.

which specifies a process environment that can output message `buy`*, then input message* `price`*, then output message* `product`*, and finally input message* `details`*. When merged with the type*

$$!\,\mathtt{price}^{\downarrow}(F).?\,\mathtt{product}^{\downarrow}(D)$$

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

$$!\,\mathtt{buy}^{\downarrow}(F).\boldsymbol{\tau}\,\mathtt{price}^{\downarrow}(F).\boldsymbol{\tau}\,\mathtt{product}^{\downarrow}(D).?\,\mathtt{details}^{\downarrow}(D)$$

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

We define *closed* behavioral type used in characterizing processes that have a matching input for all outputs, both for plain and shared messages, and are still open for further outputs on shared label messages.

Definition 3.13 *We denote by* $closed(B)$ *a behavioral type, defined as follows*

$$closed(B) \;\triangleq\; \text{if } (p\ l^d(C)) \in Labels_{\mathcal{L}}(B) \text{ then } p = \boldsymbol{\tau} \text{ or } p = ! \text{ and } l \in \mathcal{L}_\star$$

We discuss some key typing rules. In rule (Stop) we use $\boldsymbol{\tau}(L)$ to refer to a located type defined exclusively with messages of polarity $\boldsymbol{\tau}$. In rule (Res) we use $closed(B)$ as given in Definition 3.13. In rule (Rec) we use $B\langle\mathcal{X}\rangle$ as given in Definition 3.1. By L_M we denote a located type of the form $n_1 : [M_1] \mid \ldots \mid n_k : [M_k]$, then by $\star L_M$ we denote $n_1 : [\star M_1] \mid \ldots \mid n_k : [\star M_k]$. The rule states that the process is well typed

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

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

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

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

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

$$\frac{P :: L \mid B}{l^d!(n).P :: (L \bowtie n : C) \mid ?l^d(C).B}(\text{Output})$$

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

$$\frac{T <: T' \quad P :: T'}{P :: T}(\text{Sub})$$

Figure 12: Typing Rules.

under an environment that offers persistent messages M_i under conversations n_i, and offers persistent behavior B in the current conversation. We require that message types M_i are defined on shared labels and with polarity ?. 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 # Definition 3.10.

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 is now the "here" behavior at the toplevel conversation, via $loc(\uparrow B)$. The effect of $loc(B)$ is to set the direction of all messages in B to $\downarrow$.

We use $dom(L)$ to refer to the set of conversation identifiers of the located type L. In Rule (Input) we may read the premise as follows: 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 choice of the types of the continuations prefixed by the messages $!\, l_i^d(C_i)$, where the output capability $!$ corresponds to the message capability expected from the external environment (as

$$\boldsymbol{\tau}\, l^d(C).B \to B \qquad \frac{B \to B'}{\&\{\tilde{B_1}; B; \tilde{B_2}\} \to B'} \qquad \frac{B \to B'}{\oplus\{\tilde{B_1}; B; \tilde{B_2}\} \to B'}$$

$$\frac{T_1 \to T_2}{n : [T_1] \to n : [T_2]} \qquad \frac{T_1 \to T_2}{T_1 \mid T_3 \to T_2 \mid T_3} \qquad T \to T$$

Figure 13: Type Reduction.

well as the choice that also refers to the capability of performing a choice expected from the external environment). We require that message labels l_i for $i \in I$ are either all plain or all shared. In the 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 an input type, as an input is expected from the external environment. For (Input) and (Output) rules when the label used in the prefix is shared, the message type is required to conform with the continuation ($l_i^d(C_i) \asymp B_i$).

In rule (This), typing of the conversation awareness primitive requires the "here" behavior B_2 of conversation x to be a separate view of the current conversation. This allows the current conversation to be bound to the name x, and possibly sent to other partners that may need to join it. Notice that the type assigned to x is in general just a partial view of the current conversation type.

We may now present our main soundness results. Subject reduction is defined using 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. Type reduction is given in Figure 13. We use $\tilde{B}$ to denote $B_1; B_2; \ldots; B_k$.

We now state a Substitution Lemma, main auxiliary result to Subject Reduction.

Lemma 3.14 (Substitution Lemma) *Let P be a well typed process such that $P :: T \mid x : C$ and $x \notin dom(T)$. If there is T' such that $T' = T \bowtie a : C$ then $P\{x/a\} :: T'$.*

Proof. Follows by induction on the structure of P. Essentially the merge $T \bowtie a : C$ ensures the existence of the new type derivation (see Appendix A). ∎

We may now state our Subject Reduction result.

Theorem 3.15 (Subject Reduction) *Let P be a process and T a type such that $P :: T$. If $P \to Q$ then there is T' such that $T \to T'$ and $Q :: T'$.*

Proof. See Appendix B. ∎

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

Definition 3.16 *Static process contexts, noted $\mathcal{C}[\cdot]$, are defined as follows:*

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

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 $w((\nu a)l^d!(a)) = l^d$.

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

$$\frac{P :: L \mid B \quad (closed(B'), B' = \downarrow B \bowtie B_1)}{\textbf{new}\ n \cdot s \Rightarrow P :: L \mid n : [?\, s^{\downarrow}([B_1])] \mid loc(\uparrow B)}(\text{New})$$

$$\frac{P :: L \mid B}{\textbf{join}\ n \cdot s \Rightarrow P :: L \mid n : [?\, s^{\downarrow}([B_1])] \mid (B \bowtie B_1)}(\text{Join})$$

$$\frac{\textbf{def}\ s \Rightarrow P :: L_M \mid !\, s^{\downarrow}(C).B}{\star\textbf{def}\ s \Rightarrow P :: \star L_M \mid \mathtt{rec}\, \mathcal{X}.(!\, s^{\downarrow}(C).(B \mid \mathcal{X})}(\text{RepDef})$$

Figure 14: Derived Typings.

Definition 3.17 (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 $Q', R', \lambda, \lambda'$ such that $Q \xrightarrow{\lambda} Q'$, $R \xrightarrow{\lambda'} R'$ and $w(\lambda) = w(\lambda')$, $\overline{\lambda} \neq \lambda'$ and $w(\lambda)$ is not 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 3.18 (Error Freeness) *Let P be a process such that $P :: T$ for some T. Then P is not an error process.*

Proof. See Appendix C. ∎

By subject reduction (Theorem 3.15), we conclude that any process reachable from a well-typed process $P :: T$ is not an error. 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 3.15) is that any message exchange inside the process must be explained by a $\boldsymbol{\tau} M$ prefix in the related conversation type (via type reduction), thus implying conversation fidelity, i.e., all conversations follow the prescribed protocols. In the expected polyadic extension of core CC and type system we would also exclude arity mismatch errors.

3.1 Derived Typings for Service Idioms

We show in Figure 14 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 $!s([S])$, where S describes the compatible client behavior. The dual type is required for a service instantiation, of the form $?s([S])$. However, such type must be located at some context n, cf. the semantics of the **new** idiom. The typing for **join** clearly displays the partial delegation of a conversation

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

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

$$
\mathit{BuySystem} :: T_{sys}
$$

Figure 15: Typings for Buy System.

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

In this section we illustrate the expressiveness of our type system by typing the running examples. In Figure 15 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 expects someone to receive message `buy`, then to send message `price` and then to send message `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 receive the type of its own role in the conversation B_{sh}.

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

$$
\begin{array}{lcl}
B_s & \triangleq & !\,\texttt{flightReq}(F).?\,\texttt{flightResp}(D) \\
B_t & \triangleq & !\,\texttt{flightReq}^{\uparrow}(F). \\
 & & \quad (\boldsymbol{\tau}\,\texttt{reqA}(F) \mid \boldsymbol{\tau}\,\texttt{reqD}(F) \\
 & & \quad \mid \boldsymbol{\tau}\,\texttt{replyA}(D).\boldsymbol{\tau}\,\texttt{replyD}(D). \\
 & & \quad\quad ?\,\texttt{flightResp}^{\uparrow}(D). \\
 & & \quad\quad\quad (\oplus\{\boldsymbol{\tau}\,\texttt{bookD}(); \boldsymbol{\tau}\,\texttt{cancelD}()\} \\
 & & \quad\quad\quad \mid \oplus\{\boldsymbol{\tau}\,\texttt{bookA}(); \boldsymbol{\tau}\,\texttt{cancelA}()\})) \\
B_O & \triangleq & !\,\texttt{flightReq}^{\uparrow}(F). \\
 & & \quad (?\,\texttt{reqA}(F) \mid ?\,\texttt{reqD}(F) \\
 & & \quad \mid !\,\texttt{replyA}(D).!\,\texttt{replyD}(D). \\
 & & \quad\quad ?\,\texttt{flightResp}^{\uparrow}(D). \\
 & & \quad\quad\quad (\&\{?\,\texttt{bookD}(); ?\,\texttt{cancelD}()\} \\
 & & \quad\quad\quad \mid \&\{?\,\texttt{bookA}(); ?\,\texttt{cancelA}()\}) \\
B_A & \triangleq & !\,\texttt{reqA}(F).?\,\texttt{replyA}(D). \oplus \{!\,\texttt{bookA}(); !\,\texttt{cancelA}()\} \\
B_D & \triangleq & !\,\texttt{reqD}(F).?\,\texttt{replyD}(D). \oplus \{!\,\texttt{bookD}(); !\,\texttt{cancelD}()\} \\
L_a & \triangleq & \mathit{alphaAir} : [?\,\texttt{flightBook}([B_p])] \\
L_d & \triangleq & \mathit{deltaAir} : [?\,\texttt{flightBook}([B_p])] \\
B_p & \triangleq & ?\,\texttt{req}(F).!\,\texttt{reply}(D).\&\{?\,\texttt{book}(); ?\,\texttt{cancel}()\}
\end{array}
$$

$$
\begin{array}{c}
OR :: B_O \qquad AA :: L_a \mid B_A \qquad DA :: L_d \mid B_D \\
(AA \mid DA \mid OR) :: L_a \mid L_d \mid B_t \\
[AA \mid DA \mid OR] :: L_a \mid L_d \mid B_s \\
\star\,\textbf{def}\ \texttt{flightBooking} \Rightarrow [AA \mid DA \mid OR] :: \\
\quad \star L_a \mid \star\, L_d \mid \texttt{rec}\,\mathcal{X}.!\texttt{flightBooking}([B_s]).\mathcal{X}
\end{array}
$$

Figure 16: Typings for Flight Booking.

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 $!\,\texttt{askPrice}(P).?\,\texttt{readVal}(D)$ so that the merge with the `startBuy` service "up" interface results in type B_{db}, such that $B_{db} \triangleq \boldsymbol{\tau}\,\texttt{askPrice}(P).\boldsymbol{\tau}\,\texttt{readVal}(D)$.

The type of the whole system is then given by T_{sys} which specifies the two service instantiations as internal interactions $\boldsymbol{\tau}$.

We now turn to the types of the `flightBooking` service (Figure 6) which are depicted in Figure 16. We consider the **if** statement to be well typed if both branches have the same type. Whenever `flightBooking` is instantiated, the service conversation is described by the type B_s, specifying that the service expects the client to output message `fightReq` and then input message `flightResp`.

The internal anonymous conversation is described by the type B_t, where the interactions between the subsidiary service instances ($\mathit{alphaAir} \cdot \texttt{flightBook}$ and $\mathit{deltaAir} \cdot \texttt{flightBook}$) and the orchestrating process take place. Such a type may be seen as a so-called choreographic description, that describes the internal message exchanges between the different partners. We consider AA, DA and OR denote the three elements (*partner alphaAir*, *partner deltaAir* and *Orchestration*) in the body

of the definition for service `flighBooking`. Notice how B_O, B_A and B_D specify message exchanges from the local viewpoint of OR, AA and DA respectively. Notice also that $B_t \equiv B_O \bowtie B_{A<:} \bowtie B_{D<:}$, so that the behavior of the three elements merge on the "global" conversation type B_t, which still specifies interactions with the enclosing conversation (messages $\texttt{flightReq}^{\uparrow}$ and $\texttt{flightResp}^{\uparrow}$). In particular, the internal choice specified in type B_t results from matching the behavior of the orchestration script B_O, which specifies the branch types, with the behavior interface of the service instances B_A and B_D, which specify the choice types.

The choice type described in type B_A is obtained from the "up" projection of the behavioral type of the body of the $alphaAir \cdot \texttt{flightBook}$ service (process P in **new** $alphaAir \cdot \texttt{flightBook} \Leftarrow P$), where the following choice type is specified:

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

describing that a choice is expected from the enclosing environment on either message `bookA` or message `cancelA`, after which the service provider side is expected to receive either message `book` or message `cancel`. As described in Example 3.3 the $\downarrow$ and $\uparrow$ projections yield the choice type $\oplus\{!\,\texttt{bookA}^{\uparrow}();\,!\,\texttt{cancelA}^{\uparrow}()\}$ in the type of the behavioral interface of the service endpoint (B_A) and the branch type $\&\{?\,\texttt{book}();\,?\,\texttt{cancel}()\}$ in the type of the `flightBook` service conversation (B_p).

4 Progress

In this section, we propose an auxiliary proof system to enforce progress properties on systems. As most traditional deadlock detection methods (e.g., see [20]), we build on the construction of an well-founded ordering on events. In our case, since communication depends both on a context and a label, we are able to cope with systems with multiple interleaved conversations, and back and forth communications between two or more conversations in the same thread.

Essentially the proof system ensures that the events in a process can be related by a well founded partial order, where the events that are to be considered here are synchronization on messages. Since messages can be exchanged in different conversation contexts, the ordering must consider both the message label and the conversation identifier. Furthermore since references to conversations can be passed in message synchronization, the ordering also takes into account for each message the ordering associated to the conversation which is communicated in the message. We consider that events that come first to be minimal in the ordering (it would be analogous to consider first events as maximal). We can then use such an ordering to verify that all events in the continuation of an input/output prefix are greater in the ordering than the message specified in the prefix, thus guaranteeing the intended acyclic nature in event dependencies.

The proof system, depicted in Figure 17, 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, which is as 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). We note by e an event defined as $e \triangleq n.l.(x)\Gamma$ where n is the conversation,

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

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

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

$$\frac{(\ell(d).l_i.(y)\Gamma'_i \bot \Gamma) \cup \Gamma'_i\{y/x_i\} \vdash_\ell P_i}{\Gamma \vdash_\ell \Sigma_{i\in I}\ l_i^d?(x_i).P_i}(\text{Input})$$

$$\frac{(\ell(d).l.(x)\Gamma' \bot \Gamma) \vdash_\ell P \quad \Gamma'\{x/n\} \subseteq (\ell(d).l.(x)\Gamma' \bot \Gamma)}{\Gamma \vdash_\ell l^d!(n).P}(\text{Output})$$

$$\frac{\Gamma \cup \{(e_1 \prec e_2) \mid (e_1\{x/\ell(\downarrow)\} \prec_\Gamma e_2\{x/\ell(\downarrow)\})\} \vdash_\ell P}{\Gamma \vdash_\ell \mathbf{this}(x).P}(\text{This})$$

Figure 17: Proof Rules for Progress.

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 define some operations over event orderings Γ.

Definition 4.1 *The event ordering $\Gamma \setminus n$ is obtained from Γ by removing all events that have as conversation identifier the name n, while keeping the overall ordering.*

$$\begin{aligned}\Gamma \setminus n \triangleq\ & \{(e_1(m) \prec e_2(o)) \mid (e_1(m) \prec_\Gamma e_2(o)) \wedge n \neq m \wedge n \neq o\} \\ & \cup \{(e_1(m) \prec e_2(o)) \mid (e_1(m) \prec_\Gamma e_3(n)) \wedge (e_3(n) \prec_\Gamma e_2(o))\}\end{aligned}$$

We use $e_1 \prec_\Gamma e_2$ to say that e_1 is smaller than e_2 under Γ. Also, we use $e(n)$ to refer to an event such that e is of the form $n.l.(x)\Gamma$ for some l and $(x)\Gamma$.

Definition 4.2 *The event ordering $e \bot \Gamma$ is the subrelation of Γ where all events are smaller than e (with respect to Γ) defined as*

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

We briefly discuss the key proof rules of Figure 17. 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 need to verify the ordering of the recursive process body only "once". To simplify presentation, we annotate unfoldings by considering a specialized set of process variables χ_u.

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 $\ell(d).l_i.(x)\Gamma'$, the event associated with the input, enlarged with the event ordering abstraction $(x)\Gamma'$ of the event associated with the input, where the bound y is replaced by the input parameter x_i. 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. In both rules (Output) and (Input) we require there is e' such that $\ell(d).l_i.(y)\Gamma'_i \prec_\Gamma e'$.

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 $\ell(\downarrow)$. We assume that all **this** occur inside context pieces.

We now present our progress results, starting by Theorem 4.4 (Subject Reduction), and the auxiliary Lemma 4.3 (Substitution Lemma), which states that orderings are preserved in reductions.

Lemma 4.3 *Let P be a process and Γ, Γ' event orderings such that $\Gamma \cup \Gamma' \vdash_\ell P$ and $\Gamma'\{x/n\} \subseteq \Gamma$. Then $\Gamma \vdash_{\ell\{x/n\}} P\{x/n\}$.*

Proof. Follows by induction on the structure of P. Essentially the condition $\Gamma'\{x/n\} \subseteq \Gamma$ ensures the ordering already prescribed for n in Γ copes with the ordering required for conversation x (see Appendix D). ∎

Theorem 4.4 *Let P be a well typed process $P :: T$ and Γ an event ordering such that $\Gamma \vdash_\ell P$. If there is Q such that $P \to Q$ then $\Gamma \vdash_\ell Q$.*

Proof. See Appendix E. ∎

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

Definition 4.5 (Finished Process) *P is a finished process if for every static context $\mathcal{C}$ (Definition 3.16) and process Q such that $P = \mathcal{C}[Q]$ there are no l^d, a, R such that $Q \xrightarrow{l^d!(a)} R$.*

Finished processes are processes that have no active outputs. With finished processes we thus intend to characterize processes that are in a stable state (have no reductions) where there are no pending requests (outputs), but there can be some processes (e.g., persistent definitions) still listening on messages (cf., Definition 1 (Normal Form) [6] where a similar notion is presented, and Definition 4.6 (Progress) [12] where a notion of stable process is presented as a process that does not contain live channels). We may now state our progress result.

Theorem 4.6 *Let P be a well typed process such that $P :: T$, where closed(T), and Γ an event ordering and a, b names ($a, b \notin fn(P)$) such that $\Gamma \vdash_{(a,b)} P$. If P is not a finished process (Definition 4.5) then there is Q such that $P \to Q$.*

Proof. See Appendix G. ∎

Theorem 4.6 ensures that well-typed and well-ordered processes are never stuck on an output that has no matching input. This property entails services are always available upon request and protocols involving interleaving conversations never get stuck.

4.1 Proving Progress in Conversations

In this section we revisit the running examples 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 $\mathit{BuySystem} :: T_{sys}$ and $\mathit{closed}(T_{sys})$ (see Figure 15).

We consider Γ such that

$$\mathit{Seller}.\texttt{startBuy} \prec_\Gamma \mathit{Seller}.\texttt{askPrice} \prec_\Gamma \mathit{Seller}.\texttt{readVal} \prec_\Gamma \mathit{Shipper}.\texttt{newDelivery}$$

When analyzing within the scope of the restricted name corresponding to the service conversation, which we identify with *BuyChat*, Γ is extended to Γ' such that

$$\mathit{Seller}.\texttt{startBuy} \prec_{\Gamma'} \mathit{BuyChat}.\texttt{buy} \prec_{\Gamma'} \ldots \prec_{\Gamma'} \mathit{BuyChat}.\texttt{details}$$

Intuitively such ordering corresponds to the protocol ordering of messages a programmer has in mind when writing such code. Our proof system guarantees this order is coherently followed globally.

We can then state $\Gamma \vdash_\ell \mathit{BuySystem}$ which combined with $\mathit{BuySystem} :: T_{sys}$ and $\mathit{closed}(T_{sys})$ guarantees, considering the results of Theorem 4.4 and Theorem 4.6, that the process *BuySystem* reduces until it is a finished process (Definition 4.5).

Notice that since we combine conversation names and message labels in the ordering we are able to interleave conversations, for instance in the *Seller* code where after receiving a `buy` message under the service conversation, the process then accesses the enclosing conversation to consult the service provider database, and then re-interacts in the service conversation. We believe the ability to express this control flow pattern is essential in service-oriented computing, as server side resources need to be accessed in between the handling of a service instance.

We can also state that our `flightBooking` service, shown in Figure 6, enjoys the progress property, if placed in a system with a service client, *alphaAir* and *deltaAir* service providers, such that the type of the composite system is a *closed* type. If the service client persistently requires the service to be instantiated then the system has infinite reductions.

5 Related Work

Behavioral Type Systems As most behavioral type systems (see [17, 18, 11]), we build on describing 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 for instance in the typing rule for $P \mid Q$, and explored to model partial delegation of conversation fragments, have not been considered before.

Binary Sessions The notion of conversation originates in that of session (introduced in [14, 15]). 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 time. 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). Conversations 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, 16] are similar to ours. To support multiparty interaction, in particular [16] 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, 16] 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 conversations where parties can keep joining in. The approach of [2, 16] builds on two-level descriptions of service collaborations (global and local types) first introduced in a theory of endpoint projection [9]. 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 in the idea of projection [9], 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, and allows us to obtain a more flexible delegation mechanism.

Progress in Session Types There are a number of progress studies for binary sessions (e.g., [1, 6, 12]), and more recently for multiparty sessions [2, 16]. The techniques of [2, 12] 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 interacting in a session after interacting in another, and exclude interleaving on received sessions, while we consider processes that re-interact in a conversation, and

allow interleaving on received conversations.

Service-Oriented Calculi There have been a number of proposals of models for service-oriented computing [4, 5, 19, 7]. The Conversation Calculus originated in [25], where we consider service-oriented primitives as primitive constructs. Instead, in the work presented in this paper we consider service-oriented primitives as idioms of more lower-level communication primitives, allowing for a more fundamental typing analysis. We also introduce choice which turns out to have a specific handling in our model.

6 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 [23], 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, which is the focus of this paper, it is much more convenient to adopt a primitive conversation context construct, for it allows the conversation identity to be kept implicit until needed (cf. continuations in λ+callcc).

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 conversation, 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 more traditional session typed communication channels as, e.g., [16]. 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, in fixed system configurations. It would be interesting to investigate to what extent both approaches could be conciliated.

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. Boreale, R. Bruni, L. Caires, R. De Nicola, I. Lanese, M. Loreti, F. Martins, U. Montanari, A. Ravara, D. Sangiorgi, V. Vasconcelos, and G. Zavattaro. SCC: a Service Centered Calculus. In M. Bravetti, M. Núñez, and G. Zavattaro, editors, *Web Services and Formal Methods, Third International Workshop, WS-FM 2006, Proceedings*, volume 4184 of *Lecture Notes in Computer Science*, pages 38–57. Springer-Verlag, 2006.

[5] M. Boreale, R. Bruni, R. De Nicola, and M. Loreti. Sessions and Pipelines for Structured Service Programming. In G. Barthe and F. de Boer, editors, *Formal Methods for Open Object-Based Distributed Systems, 10th IFIP WG 6.1 International Conference, FMOODS 2008, Proceedings*, volume 5051 of *Lecture Notes in Computer Science*, pages 19–38. Springer-Verlag, 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] R. Bruni, I. Lanese, H. C. Melgratti, and E. Tuosto. Multiparty Sessions in SOC. In D. Lea and G. Zavattaro, editors, *Coordination Models and Languages, 10th International Conference, COORDINATION 2008, Proceedings*, volume 5052 of *Lecture Notes in Computer Science*, pages 67–82. Springer-Verlag, 2008.

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

[9] 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, Held as Part of the European Joint Conferences on Theory and Practice of*

Software, ETAPS 2007, Proceedings, volume 4421 of *Lecture Notes in Computer Science*, pages 2–17. Springer-Verlag, 2007.

[10] G. Castagna, N. Gesbert, and L. Padovani. A Theory of Contracts for Web Services. In G. C. Necula and P. Wadler, editors, *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008*, pages 261–272. ACM, 2008.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS 1998, Proceedings*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer-Verlag, 1998.

[16] 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.

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

[18] 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.

[19] I. Lanese, F. Martins, V. T. Vasconcelos, and A. Ravara. Disciplining Orchestration and Conversation in Service-Oriented Computing. In *Fifth IEEE International Conference on Software Engineering and Formal Methods, SEFM 2007, Proceedings*, pages 305–314. IEEE Computer Society Press, 2007.

[20] 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.

[21] R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, Part I + II. *Information and Computation*, 100(1):1–77, 1992.

[22] B. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[23] D. Sangiorgi and D. Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.

[24] H. T. Vieira, L. Caires, and J. C. Seco. A Model of Service Oriented Computation. TR-DI/FCT/UNL 6/07, Universidade Nova de Lisboa, Departamento de Informática, 2007.

[25] 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, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2008, Proceedings*, volume 4960 of *Lecture Notes in Computer Science*, pages 269–283. Springer-Verlag, 2008.

A Theorem 3.15 and Theorem 3.18 auxiliary results

We start by the proof of the Substitution Lemma (3.14) and the statements of the other Lemmas used in the proofs of the Theorems.

Substitution Lemma (Lemma 3.14)

Let P be a well typed process such that $P :: T \mid x : C$ and $x \notin dom(T)$. If there is T' such that $T' = T \bowtie a : C$ then $P\{x/a\} :: T'$.
Proof. Follows by induction on the structure of P. Essentially the existence of a merge $T \bowtie a : C$ ensures the existence and is the result of the new type derivation. We show the case when P is of the form $x \blacktriangleleft [Q]$. We have

$$x \blacktriangleleft [Q] :: T \mid x : C \tag{A.0.1}$$

where $x \notin dom(T)$. We have that there exist T'' such that $T \mid x : C <: T''$ and (A.0.1) is derived from

$$x \blacktriangleleft [Q] :: T'' \tag{A.0.2}$$

being that (A.0.2) is derived from

$$Q :: L \mid B \tag{A.0.3}$$

where L, B are such that

$$T'' = (L \bowtie x : [\downarrow B]) \mid loc(\uparrow B) \tag{A.0.4}$$

We have that there exist L', C'' such that

$$L \equiv L' \mid x : C'' \tag{A.0.5}$$

From (A.0.4) and (A.0.5) and $T \mid x : C <: T''$ we conclude that there exist C' such that $C <: C'$ and

$$x : C' = x : C'' \bowtie x : [\downarrow B] \tag{A.0.6}$$

and there exist T' such that $T <: T'$ such that

$$T' \equiv L' \mid loc(\uparrow B) \tag{A.0.7}$$

We now consider there is type

$$T \bowtie a : C \tag{A.0.8}$$

From (A.0.8), (A.0.7), $T <: T'$, (A.0.6) and $C <: C'$ we conclude there is

$$L' \bowtie a : C'' \tag{A.0.9}$$

From (A.0.9) we conclude

$$(L' \mid B) \bowtie a : C'' \tag{A.0.10}$$

From (A.0.3) and (A.0.5) we conclude

$$Q :: L' \mid B \mid x : C'' \tag{A.0.11}$$

By induction hypothesis on (A.0.11) and (A.0.10) we conclude

$$Q\{x/a\} :: (L' \mid B) \bowtie a : C'' \tag{A.0.12}$$

From (A.0.8), (A.0.7), $T <: T'$, $C <: C'$ and (A.0.6) we conclude

$$L' \bowtie (a : C'' \bowtie a : [\downarrow B]) \tag{A.0.13}$$

From (A.0.12) and (A.0.13) we derive

$$a \blacktriangleleft [(Q\{x/a\})] :: (L' \bowtie (a : C'' \bowtie a : [\downarrow B])) \mid loc(\uparrow B) \tag{A.0.14}$$

From (A.0.14), (A.0.7), $T <: T'$, $C <: C'$ and (A.0.6) we conclude

$$a \blacktriangleleft [(Q\{x/a\})] :: T \bowtie a : C \tag{A.0.15}$$

which completes the proof. ∎

Lemma A.1 *Let P be a well-typed process such that $P :: T$. If $P \stackrel{l^d?(a)}{\longrightarrow} Q$ then there are T', C, B such that $T \equiv T' \bowtie \oplus\{\ldots; !\, l^d(C).B; \ldots\}$. Furthermore if there is T'' such that $T'' = (T' \bowtie B) \bowtie a : C$ then $Q :: T''$.*

Proof. Follows by induction on the derivation of the label. We consider the different cases: either the transition results from a input summation, or from within a conversation piece, or from within the scope of a restriction, or from one component of a parallel composition, or from the body of a recursive process.

($Case\ \Sigma_{i\in I}\ l_i^d?(x_i).P_i' \stackrel{l_i^d?(a)}{\longrightarrow} P_j'\{x_j/a\}$)
We have that

$$\Sigma_{i\in I}\ l_i^d?(x_i).P_i' :: T \tag{A.1.1}$$

and

$$\Sigma_{i\in I}\ l_i^d?(x_i).P_i' \stackrel{l_i^d?(a)}{\longrightarrow} P_j'\{x_j/a\} \tag{A.1.2}$$

We have there is L, C_i, B_i such that

$$T <: L \mid \ \oplus\{!l_i^d(C_i).B_i\} \tag{A.1.3}$$

and

$$\Sigma_{i\in I}\ l_i^d?(x_i).P_i' :: L \mid \ \oplus\{!l_i^d(C_i).B_i\} \tag{A.1.4}$$

and

$$P_i' :: L \mid B_i \mid x_i : C_i \tag{A.1.5}$$

From (A.1.3) we have there is T' such that

$$T \equiv T' \bowtie \oplus\{\ldots; !l_i^d(C_i).B_i; \ldots\} \tag{A.1.6}$$

Let us now consider there is T'' such that

$$T'' \equiv (T' \bowtie B) \bowtie a : C_j \tag{A.1.7}$$

Considering Lemma 3.14 we then have

$$P'_j\{x_j/a\} :: T'' \tag{A.1.8}$$

which completes the proof for this case.

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

We have that

$$c \blacktriangleleft \left[P'\right] :: T \tag{A.1.9}$$

and

$$c \blacktriangleleft \left[P'\right] \overset{l^{\downarrow}?(a)}{\longrightarrow} c \blacktriangleleft \left[Q'\right] \tag{A.1.10}$$

We have there is L, B such that

$$T <: (L \bowtie c : [\downarrow B]) \mid loc(\uparrow B) \tag{A.1.11}$$

and

$$c \blacktriangleleft \left[P'\right] :: (L \bowtie c : [\downarrow B]) \mid loc(\uparrow B) \tag{A.1.12}$$

and

$$P' :: L \mid B \tag{A.1.13}$$

We also have that (A.1.10) is derived from

$$P' \overset{l^{\uparrow}?(a)}{\longrightarrow} Q' \tag{A.1.14}$$

By induction hypothesis on (A.1.14) and (A.1.13) we have there is T', C, B' such that

$$L \mid B \equiv T' \bowtie \oplus\{\ldots; !l^{\uparrow}(C).B'; \ldots\} \tag{A.1.15}$$

From (A.1.15) we have that there is B_1 such that

$$loc(\uparrow B) \equiv B_1 \bowtie \oplus\{\ldots; !l^{\downarrow}(C).B'; \ldots\} \tag{A.1.16}$$

From (A.1.11) and (A.1.16) we have there is T_1 such that

$$T \equiv T_1 \bowtie \oplus\{\ldots; !l^{\downarrow}(C).B'; \ldots\} \tag{A.1.17}$$

Let us now consider there is T_2 such that

$$T_2 \equiv (T_1 \bowtie B') \bowtie a : C \tag{A.1.18}$$

From (A.1.18), (A.1.17), (A.1.11) and (A.1.15) we conclude there is T'' such that

$$T'' \equiv (T' \bowtie B') \bowtie a : C \tag{A.1.19}$$

By induction hypothesis we then have

$$Q' :: (T' \bowtie B') \bowtie a : C \tag{A.1.20}$$

From (A.1.15) we have that there is B_2 such that

$$L \mid B_2 \equiv (T' \bowtie B') \tag{A.1.21}$$

and

$$\downarrow B \equiv \downarrow B_2 \tag{A.1.22}$$

and

$$\uparrow B_2 \equiv B_1 \bowtie B' \tag{A.1.23}$$

We then have that

$$c \blacktriangleleft \left[Q'\right] :: ((L \bowtie a : C) \bowtie c : [\downarrow B_2]) \mid loc(\uparrow B_2) \tag{A.1.24}$$

From (A.1.24), (A.1.23), (A.1.22), (A.1.11) and (A.1.17) we have that

$$c \blacktriangleleft \left[Q'\right] :: (T_1 \bowtie B') \bowtie a : C \tag{A.1.25}$$

which completes the proof for this case.

(*Case* $(\nu c)P' \overset{l^d?(a)}{\longrightarrow} (\nu c)Q'$)

We have that

$$(\nu c)P' :: T \tag{A.1.26}$$

and

$$(\nu c)P' \overset{l^d?(a)}{\longrightarrow} (\nu c)Q' \tag{A.1.27}$$

which is derived from

$$P' \overset{l^d?(a)}{\longrightarrow} Q' \tag{A.1.28}$$

We have that there is T', B such that

$$T <: T' \tag{A.1.29}$$

and

$$(\nu c)P' :: T' \tag{A.1.30}$$

and

$$P' :: T' \mid c : [B] \tag{A.1.31}$$

and $closed(B)$. By induction hypothesis on (A.1.28) and (A.1.31) we have that there is T'', C, B' such that

$$T' \mid c : [B] \equiv T'' \bowtie \oplus\{\ldots; !l^d(C).B'; \ldots\} \tag{A.1.32}$$

From (A.1.32) we have there is T_1'' such that

$$T'' \equiv T_1'' \mid c : [B] \tag{A.1.33}$$

We then have that

$$(\nu c)P' :: T_1'' \bowtie \oplus\{\ldots; !l^d(C).B'; \ldots\} \tag{A.1.34}$$

Let us now consider there is T_2 such that

$$T_2 \equiv (T_1'' \bowtie B') \bowtie a : C \tag{A.1.35}$$

We then have, since $a \neq c$, that

$$((T_1'' \mid c : [B]) \bowtie B') \bowtie a : C \tag{A.1.36}$$

which by induction hypothesis gives us

$$Q' :: ((T_1'' \mid c : [B]) \bowtie B') \bowtie a : C \tag{A.1.37}$$

From (A.1.37) we derive

$$(\nu c)Q' :: (T_1'' \bowtie B') \bowtie a : C \tag{A.1.38}$$

which completes the proof for this case.

(*Case* $P' \mid R \overset{l^d?(a)}{\longrightarrow} Q' \mid R$)

We have that

$$P' \mid R :: T \tag{A.1.39}$$

and

$$P' \mid R \overset{l^d?(a)}{\longrightarrow} Q' \mid R \tag{A.1.40}$$

which is derived from

$$P' \overset{l^d?(a)}{\longrightarrow} Q' \tag{A.1.41}$$

We have that there is T_1, T_2 such that

$$T <: T_1 \bowtie T_2 \tag{A.1.42}$$

and

$$P' \mid R :: T_1 \bowtie T_2 \tag{A.1.43}$$

and

$$P' :: T_1 \tag{A.1.44}$$

and

$$R :: T_2 \tag{A.1.45}$$

By induction hypothesis on (A.1.41) and (A.1.44) we have that there is T', C, B such that

$$T_1 \equiv T' \bowtie \oplus\{\ldots; !l^d(C).B; \ldots\} \tag{A.1.46}$$

From (A.1.46) and (A.1.42) we conclude there is T'' such that

$$T \equiv T'' \bowtie \oplus\{\ldots; !l^d(C).B; \ldots\} \tag{A.1.47}$$

Let us now consider there is T_3 such that

$$T_3 \equiv (T'' \bowtie B) \bowtie a : C \tag{A.1.48}$$

From (A.1.48), (A.1.46) and (A.1.42) we have that

$$(T' \bowtie B) \bowtie a : C \tag{A.1.49}$$

By induction hypothesis we then have

$$Q' :: (T' \bowtie B) \bowtie a : C \tag{A.1.50}$$

From (A.1.50) and (A.1.45) we derive

$$Q' \mid R :: ((T' \bowtie B) \bowtie a : C) \bowtie T_2 \tag{A.1.51}$$

From (A.1.51), (A.1.47), (A.1.46) and (A.1.42) we conclude

$$Q' \mid R :: (T'' \bowtie B) \bowtie a : C \tag{A.1.52}$$

which completes the proof for this case.

(*Case* $\mathbf{rec}\ \mathcal{X}.P' \overset{l^d?(a)}{\longrightarrow} Q'$)

We have that

$$\mathbf{rec}\ \mathcal{X}.P' :: T \tag{A.1.53}$$

and

$$\mathbf{rec}\ \mathcal{X}.P' \overset{l^d?(a)}{\longrightarrow} Q' \tag{A.1.54}$$

which is derived from

$$P'\{\mathcal{X}/\mathbf{rec}\ \mathcal{X}.P'\} \overset{l^d?(a)}{\longrightarrow} Q' \tag{A.1.55}$$

We have that there is L_M, B such that

$$T <: \star L_M \mid \mathtt{rec}\ \mathcal{X}.B\langle\mathcal{X}\rangle \tag{A.1.56}$$

and

$$\mathbf{rec}\ \mathcal{X}.P' :: \star L_M \mid \mathtt{rec}\ \mathcal{X}.B\langle\mathcal{X}\rangle \tag{A.1.57}$$

and

$$P' :: L_M \mid B\langle\mathcal{X}\rangle \tag{A.1.58}$$

From (A.1.58) we derive

$$P'\{\mathcal{X}/\mathbf{rec}\ \mathcal{X}.P'\} :: L_M \mid \star L_M \mid B\langle\mathtt{rec}\ \mathcal{X}.B\langle\mathcal{X}\rangle\rangle \tag{A.1.59}$$

By induction hypothesis on (A.1.55) and (A.1.59) we have that there is T', C, B' such that

$$L_M \mid \star L_M \mid B\langle\mathtt{rec}\ \mathcal{X}.B\langle\mathcal{X}\rangle\rangle \equiv T' \bowtie \oplus\{\ldots; !l^d(C).B'; \ldots\} \tag{A.1.60}$$

From (A.1.60) and (A.1.56) we conclude there is T'' such that

$$T \equiv T'' \bowtie \oplus\{\ldots; !l^d(C).B'; \ldots\} \tag{A.1.61}$$

Let us now consider there is T_2 such that

$$T_2 \equiv (T'' \bowtie B') \bowtie a : C \tag{A.1.62}$$

From (A.1.62), (A.1.60) and (A.1.61) we have that

$$(T' \bowtie B') \bowtie a : C \tag{A.1.63}$$

By induction hypothesis we then have

$$Q' :: (T' \bowtie B) \bowtie a : C \tag{A.1.64}$$

From (A.1.64), (A.1.60), (A.1.61) and (A.1.56) we derive

$$Q' :: (T'' \bowtie B) \bowtie a : C \tag{A.1.65}$$

which completes the proof. ∎

Lemma A.2 *Let P be a well-typed process such that $P :: T$. If $P \overset{c\ l^{\downarrow}?(a)}{\longrightarrow} Q$ then there are T', C, B such that $T \equiv T' \bowtie c : [\oplus\{\ldots; !\, l^{\downarrow}(C).B; \ldots\}]$. Furthermore if there is T'' such that $T'' = (T' \bowtie c : [B]) \bowtie a : C$ then $Q :: T''$.*

Proof. Follows by induction on the derivation of the label, along the lines of the proof of Lemma A.1. We show the base case of $l^{\downarrow}?(a)$ transition originating from within a context piece and the case where the derivation originates from the located transition $c\ l^{\downarrow}?(a)$ from within a context piece.

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

We have that

$$c \blacktriangleleft \left[P'\right] :: T \tag{A.2.1}$$

and

$$c \blacktriangleleft \left[P'\right] \overset{c\ l^{\downarrow}?(a)}{\longrightarrow} c \blacktriangleleft \left[Q'\right] \tag{A.2.2}$$

We have there is L, B such that

$$T <: (L \bowtie c : [\downarrow B]) \mid loc(\uparrow B) \tag{A.2.3}$$

and

$$c \blacktriangleleft \left[P'\right] :: (L \bowtie c : [\downarrow B]) \mid loc(\uparrow B) \tag{A.2.4}$$

and

$$P' :: L \mid B \tag{A.2.5}$$

We also have that (A.2.2) is derived from

$$P' \overset{l^{\downarrow}?(a)}{\longrightarrow} Q' \tag{A.2.6}$$

Considering Lemma A.1, from (A.2.6) and (A.2.5) we have there is T', C, B' such that

$$L \mid B \equiv T' \bowtie \oplus\{\ldots; !l^{\downarrow}(C).B'; \ldots\} \tag{A.2.7}$$

From (A.2.7) we have that there is B_1 such that

$$\downarrow B \equiv B_1 \bowtie \oplus\{\ldots; !l^{\downarrow}(C).B'; \ldots\} \tag{A.2.8}$$

From (A.2.3) and (A.2.8) we have there is T_1 such that

$$T \equiv T_1 \bowtie c : \left[\oplus\{\ldots; !l^{\downarrow}(C).B'; \ldots\}\right] \tag{A.2.9}$$

Let us now consider there is T_2 such that

$$T_2 \equiv (T_1 \bowtie c : \left[B'\right]) \bowtie a : C \tag{A.2.10}$$

From (A.2.10), (A.2.9), (A.2.3) and (A.2.7) we conclude there is T'' such that

$$T'' \equiv (T' \bowtie c : \left[B'\right]) \bowtie a : C \tag{A.2.11}$$

By induction hypothesis we then have

$$Q' :: (T' \bowtie c : \left[B'\right]) \bowtie a : C \tag{A.2.12}$$

From (A.2.7) we have that there is B_2 such that

$$L \mid B_2 \equiv (T' \bowtie B') \tag{A.2.13}$$

and

$$\uparrow B \equiv \uparrow B_2 \tag{A.2.14}$$

and

$$\downarrow B_2 \equiv B_1 \bowtie B' \tag{A.2.15}$$

We then have that

$$c \blacktriangleleft \left[Q'\right] :: ((L \bowtie a : C) \bowtie c : [\downarrow B_2]) \mid loc(\uparrow B_2) \tag{A.2.16}$$

From (A.2.16), (A.2.15), (A.2.14), (A.2.3) and (A.2.9) we have that

$$c \blacktriangleleft \left[Q'\right] :: (T_1 \bowtie c : \left[B'\right]) \bowtie a : C \tag{A.2.17}$$

which completes the proof for this case.

($Case\ b \blacktriangleleft [P'] \overset{c\ l^{\downarrow}?(a)}{\longrightarrow} b \blacktriangleleft [Q']$)

We have that

$$b \blacktriangleleft \left[P'\right] :: T \tag{A.2.18}$$

and

$$b \blacktriangleleft \left[P'\right] \overset{c\ l^{\downarrow}?(a)}{\longrightarrow} b \blacktriangleleft \left[Q'\right] \tag{A.2.19}$$

We have there is L, B such that

$$T <: (L \bowtie b : [\downarrow B]) \mid loc(\uparrow B) \tag{A.2.20}$$

and

$$b \blacktriangleleft \left[P'\right] :: (L \bowtie b : [\downarrow B]) \mid loc(\uparrow B) \tag{A.2.21}$$

and

$$P' :: L \mid B \tag{A.2.22}$$

We also have that (A.2.19) is derived from

$$P' \overset{c\ l^{\downarrow}?(a)}{\longrightarrow} Q' \tag{A.2.23}$$

By induction hypothesis on (A.2.23) and (A.2.22) we have there is T', C, B' such that

$$L \mid B \equiv T' \bowtie c : \left[\oplus\{\ldots; !l^{\downarrow}(C).B'; \ldots\}\right] \tag{A.2.24}$$

From (A.2.24) we have that there is L' such that

$$L \equiv L' \bowtie c : \left[\oplus\{\ldots; !l^{\downarrow}(C).B'; \ldots\}\right] \tag{A.2.25}$$

From (A.2.20) and (A.2.25) we have there is T_1 such that

$$T \equiv T_1 \bowtie c : \left[\oplus\{\ldots; !l^{\downarrow}(C).B'; \ldots\}\right] \tag{A.2.26}$$

Let us now consider there is T_2 such that

$$T_2 \equiv (T_1 \bowtie c : \left[B'\right]) \bowtie a : C \tag{A.2.27}$$

From (A.2.27), (A.2.26), (A.2.20) and (A.2.24) we conclude there is T'' such that

$$T'' \equiv (T' \bowtie c : \left[B'\right]) \bowtie a : C \tag{A.2.28}$$

By induction hypothesis we then have

$$Q' :: (T' \bowtie c : \left[B'\right]) \bowtie a : C \tag{A.2.29}$$

We then have that

$$c \blacktriangleleft \left[Q'\right] :: ((L' \bowtie c : \left[B'\right]) \bowtie a : C) \bowtie b : [\downarrow B]) \mid loc(\uparrow B) \tag{A.2.30}$$

From (A.2.30), (A.2.20), (A.2.25) and (A.2.26) we have that

$$c \blacktriangleleft \left[Q'\right] :: (T_1 \bowtie c : \left[B'\right]) \bowtie a : C \tag{A.2.31}$$

which completes the proof for this case. ∎

Lemma A.3 *Let P be a well-typed process such that $P :: T$. If $P \xrightarrow{l^d!(a)} Q$ then there are T', C, B such that $T \equiv T' \bowtie \&\{\ldots; ?\, l^d(C).B; \ldots\}$, and there is T'' such that $T' = T'' \bowtie a : C$ and $Q :: T'' \bowtie B$.*

Proof. Follows by induction on the derivation of the label. We consider the different cases: either the transition results from a output prefix, or from within a conversation piece, or from within the scope of a restriction, or from one component of a parallel composition, or from the body of a recursive process.

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

We have that

$$l^d!(a).P' :: T \tag{A.3.1}$$

and

$$l^d!(a).P' \xrightarrow{l^d!(a)} P' \tag{A.3.2}$$

We have there is L, C, B such that

$$T <: (L \bowtie a : C) \mid \&\{?l^d(C).B\} \tag{A.3.3}$$

and

$$l^d!(a).P' :: (L \bowtie a : C) \mid \&\{?l^d(C).B\} \tag{A.3.4}$$

and

$$P' :: L \mid B \tag{A.3.5}$$

From (A.3.3) we have there is T' such that

$$T \equiv T' \bowtie \&\{\ldots; ?l^d(C).B; \ldots\} \tag{A.3.6}$$

and also that there is T'' such that

$$T' \equiv T'' \bowtie a : C \tag{A.3.7}$$

and finally, considering (A.3.5), that

$$P' :: T'' \bowtie B \tag{A.3.8}$$

which completes the proof for this case.

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

We have that

$$c \blacktriangleleft \left[P'\right] :: T \tag{A.3.9}$$

and

$$c \blacktriangleleft \left[P'\right] \overset{l^{\downarrow}!(a)}{\longrightarrow} c \blacktriangleleft \left[Q'\right] \tag{A.3.10}$$

We have there is L, B such that

$$T <: (L \bowtie c : [\downarrow B]) \mid loc(\uparrow B) \tag{A.3.11}$$

and

$$c \blacktriangleleft \left[P'\right] :: (L \bowtie c : [\downarrow B]) \mid loc(\uparrow B) \tag{A.3.12}$$

and

$$P' :: L \mid B \tag{A.3.13}$$

We also have that (A.3.10) is derived from

$$P' \overset{l^{\uparrow}!(a)}{\longrightarrow} Q' \tag{A.3.14}$$

By induction hypothesis on (A.3.14) and (A.3.13) we have there is T', C, B' such that

$$L \mid B \equiv T' \bowtie \&\{\ldots; ?l^{\uparrow}(C).B'; \ldots\} \tag{A.3.15}$$

and there is T'' such that

$$T' \equiv T'' \bowtie a : C \tag{A.3.16}$$

and

$$Q' :: T'' \bowtie B' \tag{A.3.17}$$

From (A.3.15) we have that there is B_1 such that

$$loc(\uparrow B) \equiv B_1 \bowtie \&\{\ldots; ?l^{\downarrow}(C).B'; \ldots\} \tag{A.3.18}$$

From (A.3.11) and (A.3.18) we have there is T_1 such that

$$T \equiv T_1 \bowtie \&\{\ldots; ?l^{\downarrow}(C).B'; \ldots\} \tag{A.3.19}$$

From (A.3.11), (A.3.16) and (A.3.15) we conclude there is T_1'' such that

$$T_1 \equiv T_1'' \bowtie a : C \tag{A.3.20}$$

and from (A.3.17) that

$$c \blacktriangleleft \left[Q'\right] :: T_1'' \bowtie B' \tag{A.3.21}$$

which completes the proof for this case.

(*Case* $(\nu c)P' \overset{l^{d}?(a)}{\longrightarrow} (\nu c)Q'$)

We have that

$$(\nu c)P' :: T \tag{A.3.22}$$

and

$$(\nu c)P' \xrightarrow{l^{d}!(a)} (\nu c)Q' \tag{A.3.23}$$

which is derived from

$$P' \xrightarrow{l^{d}!(a)} Q' \tag{A.3.24}$$

We have that there is T', B such that

$$T <: T' \tag{A.3.25}$$

and

$$(\nu c)P' :: T' \tag{A.3.26}$$

and

$$P' :: T' \mid c : [B] \tag{A.3.27}$$

and $closed(B)$. By induction hypothesis on (A.3.24) and (A.3.27) we have that there is T'', C, B' such that

$$T' \mid c : [B] \equiv T'' \bowtie \&\{\ldots; ?l^{d}(C).B'; \ldots\} \tag{A.3.28}$$

and

$$T'' \equiv T_1'' \bowtie a : C \tag{A.3.29}$$

and

$$Q' :: T_1'' \bowtie B' \tag{A.3.30}$$

From (A.3.28) we have there is T_2'' such that

$$T'' \equiv T_2'' \mid c : [B] \tag{A.3.31}$$

and since $a \neq c$, T_3'' such that

$$T_1'' \equiv T_3'' \mid c : [B] \tag{A.3.32}$$

We then have that

$$(\nu c)P' :: T_2'' \bowtie \&\{\ldots; ?l^{d}(C).B'; \ldots\} \tag{A.3.33}$$

and

$$T_2'' \equiv T_3'' \bowtie a : C \tag{A.3.34}$$

and

$$(\nu c)Q' :: T_3'' \bowtie B' \tag{A.3.35}$$

which completes the proof for this case.

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

We have that

$$P' \mid R :: T \tag{A.3.36}$$

and

$$P' \mid R \xrightarrow{l^{d}!(a)} Q' \mid R \tag{A.3.37}$$

which is derived from

$$P' \xrightarrow{l^d!(a)} Q' \tag{A.3.38}$$

We have that there is T_1, T_2 such that

$$T <: T_1 \bowtie T_2 \tag{A.3.39}$$

and

$$P' \mid R :: T_1 \bowtie T_2 \tag{A.3.40}$$

and

$$P' :: T_1 \tag{A.3.41}$$

and

$$R :: T_2 \tag{A.3.42}$$

By induction hypothesis on (A.3.38) and (A.3.41) we have that there is T', C, B such that

$$T_1 \equiv T' \bowtie \&\{\ldots; ?l^d(C).B; \ldots\} \tag{A.3.43}$$

and T'' such that

$$T' \equiv T'' \bowtie a : C \tag{A.3.44}$$

and

$$Q' :: T'' \bowtie B \tag{A.3.45}$$

From (A.3.43) and (A.3.39) we conclude there is T_1' such that

$$T \equiv (T' \bowtie T_2) \bowtie \&\{\ldots; ?l^d(C).B; \ldots\} \tag{A.3.46}$$

and T_3 such that

$$(T' \bowtie T_2) \equiv (T'' \bowtie T_2) \bowtie a : C \tag{A.3.47}$$

and

$$Q' \mid R :: (T'' \bowtie B) \bowtie T_2 \tag{A.3.48}$$

which completes the proof for this case.

(*Case* $\mathbf{rec}\ \mathcal{X}.P' \xrightarrow{l^d!(a)} Q'$)

We have that

$$\mathbf{rec}\ \mathcal{X}.P' :: T \tag{A.3.49}$$

and

$$\mathbf{rec}\ \mathcal{X}.P' \xrightarrow{l^d!(a)} Q' \tag{A.3.50}$$

which is derived from

$$P'\{\mathcal{X}/\mathbf{rec}\ \mathcal{X}.P'\} \xrightarrow{l^d!(a)} Q' \tag{A.3.51}$$

We have that there is L_M, B such that

$$T <: \star L_M \mid \mathtt{rec}\ \mathcal{X}.B\langle\mathcal{X}\rangle \tag{A.3.52}$$

and

$$\mathbf{rec}\ \mathcal{X}.P' :: \star L_M \mid \mathtt{rec}\ \mathcal{X}.B\langle\mathcal{X}\rangle \tag{A.3.53}$$

and

$$P' :: L_M \mid B\langle \mathcal{X} \rangle \tag{A.3.54}$$

From (A.3.54) we derive

$$P'\{\mathcal{X}/\mathbf{rec}\ \mathcal{X}.P'\} :: L_M \mid \star L_M \mid B\langle \mathtt{rec}\ \mathcal{X}.B\langle \mathcal{X} \rangle \rangle \tag{A.3.55}$$

By induction hypothesis on (A.3.51) and (A.3.55) we have that there is T', C, B' such that

$$L_M \mid \star L_M \mid B\langle \mathtt{rec}\ \mathcal{X}.B\langle \mathcal{X} \rangle \rangle \equiv T' \bowtie \&\{\ldots; ?l^d(C).B'; \ldots\} \tag{A.3.56}$$

and T'' such that

$$T' \equiv T'' \bowtie a : C \tag{A.3.57}$$

and

$$Q' :: T'' \bowtie B' \tag{A.3.58}$$

From (A.3.56) and (A.3.52) we conclude there is T_2 such that

$$T \equiv T_2 \bowtie \oplus\{\ldots; !l^d(C).B'; \ldots\} \tag{A.3.59}$$

From (A.3.57), (A.3.56) and (A.3.52) we conclude there is T_2'' such that

$$T_2 \equiv T_2'' \bowtie a : C \tag{A.3.60}$$

and

$$Q' :: T_2'' \bowtie B' \tag{A.3.61}$$

which completes the proof. ∎

Lemma A.4 *Let P be a well-typed process such that $P :: T$. If $P \overset{c\ l^{\downarrow}!(a)}{\longrightarrow} Q$ then there are T', C, B such that $T \equiv T' \bowtie c : [\&\{\ldots; ?\, l^{\downarrow}(C).B; \ldots\}]$, and there are T'' such that $T' \equiv T'' \bowtie a : C$ and $Q :: T'' \bowtie c : [B]$.*

Proof. Follows by induction on the derivation of the label, along the lines of the proof of Lemma A.3. We show the base case of $l^{\downarrow}!(a)$ transition originating from within a context piece and the case where the derivation originates from the located transition $c\ l^{\downarrow}!(a)$ from within a context piece.

(*Case* $c \blacktriangleleft [P'] \overset{c\ l^{\downarrow}!(a)}{\longrightarrow} c \blacktriangleleft [Q']$)
We have that

$$c \blacktriangleleft \left[P'\right] :: T \tag{A.4.1}$$

and

$$c \blacktriangleleft \left[P'\right] \overset{c\ l^{\downarrow}!(a)}{\longrightarrow} c \blacktriangleleft \left[Q'\right] \tag{A.4.2}$$

We have there is L, B such that

$$T <: (L \bowtie c : [\downarrow B]) \mid loc(\uparrow B) \tag{A.4.3}$$

and

$$c \blacktriangleleft \left[P'\right] :: (L \bowtie c : [\downarrow B]) \mid loc(\uparrow B) \tag{A.4.4}$$

and

$$P' :: L \mid B \tag{A.4.5}$$

We also have that (A.4.2) is derived from

$$P' \xrightarrow{l^{\downarrow}!(a)} Q' \tag{A.4.6}$$

Considering Lemma A.3, from (A.4.6) and (A.4.5) we have there is T', C, B' such that

$$L \mid B \equiv T' \bowtie \&\{\ldots; ?l^{\downarrow}(C).B'; \ldots\} \tag{A.4.7}$$

and T'' such that

$$T' \equiv T'' \bowtie a : C \tag{A.4.8}$$

and

$$Q' :: T'' \bowtie B' \tag{A.4.9}$$

From (A.4.7) we have that there is B_1 such that

$$\downarrow B \equiv B_1 \bowtie \&\{\ldots; ?l^{\downarrow}(C).B'; \ldots\} \tag{A.4.10}$$

From (A.4.3) and (A.4.10) we have there is T_1 such that

$$T \equiv T_1 \bowtie c : \left[\&\{\ldots; ?l^{\downarrow}(C).B'; \ldots\}\right] \tag{A.4.11}$$

From (A.4.7) we have that there is B_2 such that

$$L \mid B_2 \equiv (T' \bowtie B') \tag{A.4.12}$$

and

$$\uparrow B \equiv \uparrow B_2 \tag{A.4.13}$$

and

$$\downarrow B_2 \equiv B_1 \bowtie B' \tag{A.4.14}$$

We also have that there is L' such that

$$L \equiv L' \bowtie a : C \tag{A.4.15}$$

We then have that

$$c \blacktriangleleft \left[Q'\right] :: (L' \bowtie c : [\downarrow B_2]) \mid loc(\uparrow B_2) \tag{A.4.16}$$

From (A.4.16), (A.4.14), (A.4.13), (A.4.3) and (A.4.11) we have that

$$T_1 \equiv ((L' \bowtie c : [B_1]) \mid loc(\uparrow B_2)) \bowtie a : C \tag{A.4.17}$$

and

$$c \blacktriangleleft \left[Q'\right] :: ((L' \bowtie c : [B_1]) \mid loc(\uparrow B_2)) \bowtie c : \left[B'\right] \tag{A.4.18}$$

which completes the proof for this case.

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

We have that

$$b \blacktriangleleft \left[P'\right] :: T \tag{A.4.19}$$

and

$$b \blacktriangleleft \left[P'\right] \overset{c\ l^{\downarrow}!(a)}{\longrightarrow} b \blacktriangleleft \left[Q'\right] \tag{A.4.20}$$

We have there is L, B such that

$$T <: (L \bowtie b : [\downarrow B]) \mid loc(\uparrow B) \tag{A.4.21}$$

and

$$b \blacktriangleleft \left[P'\right] :: (L \bowtie b : [\downarrow B]) \mid loc(\uparrow B) \tag{A.4.22}$$

and

$$P' :: L \mid B \tag{A.4.23}$$

We also have that (A.4.20) is derived from

$$P' \overset{c\ l^{\downarrow}!(a)}{\longrightarrow} Q' \tag{A.4.24}$$

By induction hypothesis on (A.4.24) and (A.4.23) we have there is T', C, B' such that

$$L \mid B \equiv T' \bowtie c : \left[\&\{\ldots; ?l^{\downarrow}(C).B'; \ldots\}\right] \tag{A.4.25}$$

and

$$T' \equiv T'' \bowtie a : C \tag{A.4.26}$$

and

$$Q' :: T'' \bowtie c : \left[B'\right] \tag{A.4.27}$$

From (A.4.25) we have that there is L' such that

$$L \equiv L' \bowtie c : \left[\&\{\ldots; ?l^{\downarrow}(C).B'; \ldots\}\right] \bowtie a : C \tag{A.4.28}$$

From (A.4.21) and (A.4.28) we have there is T_1 such that

$$T \equiv T_1 \bowtie c : \left[\&\{\ldots; ?l^{\downarrow}(C).B'; \ldots\}\right] \tag{A.4.29}$$

We then have that

$$c \blacktriangleleft \left[Q'\right] :: ((L' \bowtie c : \left[B'\right]) \bowtie b : [\downarrow B]) \mid loc(\uparrow B) \tag{A.4.30}$$

and

$$T_1 \equiv ((L' \bowtie b : [\downarrow B]) \mid loc(\uparrow B)) \bowtie a : C \tag{A.4.31}$$

which completes the proof for this case. ∎

Lemma A.5 *Let P be a well-typed process such that $P :: T$. If $P \overset{(\nu a) l^{d}!(a)}{\longrightarrow} Q$ then there are T', C, B such that $T \equiv T' \bowtie \&\{\ldots; ?\, l^{d}(C).B; \ldots\}$, and there are B', C' such that closed(B') and $a : [B'] = a : C' \bowtie a : C$ and $Q :: (T' \bowtie B) \mid a : C'$.*

Proof. Follows by induction on the derivation of the label. The proof follows similar lines to that of Lemma A.3. We show the case of restriction open (Fig. 3 (*opn*)).

(*Case* $(\nu a)P' \overset{(\nu a) l^{d}!(a)}{\longrightarrow} Q'$)
We have that

$$(\nu a)P' :: T \tag{A.5.1}$$

and

$$(\nu a)P' \stackrel{l^d!(a)}{\longrightarrow} Q' \tag{A.5.2}$$

which is derived from

$$P' \stackrel{l^d!(a)}{\longrightarrow} Q' \tag{A.5.3}$$

We have that there is T', B such that

$$T <: T' \tag{A.5.4}$$

and

$$(\nu a)P' :: T' \tag{A.5.5}$$

and

$$P' :: T' \mid a : [B] \tag{A.5.6}$$

and $closed(B)$. Considering Lemma A.3 and (A.5.3) and (A.5.6) we have that there are T'', C, B' such that

$$T' \mid a : [B] \equiv T'' \bowtie \&\{\ldots; ?l^d(C).B'; \ldots\} \tag{A.5.7}$$

and

$$T'' \equiv T_1'' \bowtie a : C' \tag{A.5.8}$$

and

$$Q' :: T_1'' \bowtie B' \tag{A.5.9}$$

From (A.5.7) we have there is T_2'', C' such that

$$T_1'' \equiv T_2'' \mid a : C \tag{A.5.10}$$

and

$$a : [B] = a : C' \bowtie a : C \tag{A.5.11}$$

which completes the proof for this case. ∎

Lemma A.6 *Let P be a well-typed process such that $P :: T$. If $P \stackrel{(\nu a)c\ l^{\downarrow}!(a)}{\longrightarrow} Q$ then there are T', C, B such that $T \equiv T' \bowtie c : [\&\{\ldots; ?\, l^{\downarrow}(C).B; \ldots\}]$, and 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. Follows by induction on the derivation of the label. We show the case of a $(\nu a)l^{\downarrow}!(a)$ transition originating from within a context piece.

(*Case* $c \blacktriangleleft [P'] \stackrel{(\nu a)c\ l^{\downarrow}!(a)}{\longrightarrow} c \blacktriangleleft [Q']$)
We have that

$$c \blacktriangleleft \left[P'\right] :: T \tag{A.6.1}$$

and

$$c \blacktriangleleft \left[P'\right] \stackrel{(\nu a)c\ l^{\downarrow}!(a)}{\longrightarrow} c \blacktriangleleft \left[Q'\right] \tag{A.6.2}$$

We have there is L, B such that

$$T <: (L \bowtie c : [\downarrow B]) \mid loc(\uparrow B) \tag{A.6.3}$$

and

$$c \blacktriangleleft \left[P'\right] :: (L \bowtie c : [\downarrow B]) \mid loc(\uparrow B) \tag{A.6.4}$$

and

$$P' :: L \mid B \tag{A.6.5}$$

We also have that (A.6.2) is derived from

$$P' \xrightarrow{(\nu a) l^{\downarrow} ! (a)} Q' \tag{A.6.6}$$

Considering Lemma A.5, from (A.6.6) and (A.6.5) we have there is T', C, B' such that

$$L \mid B \equiv T' \bowtie \&\{\ldots; ?l^{\downarrow}(C).B'; \ldots\} \tag{A.6.7}$$

and B'', C'' such that $closed(B'')$ and

$$a : \left[B''\right] = a : C' \bowtie a : C \tag{A.6.8}$$

and

$$Q' :: (T' \bowtie B') \bowtie a : C' \tag{A.6.9}$$

From (A.6.7) we have that there is B_1 such that

$$\downarrow B \equiv B_1 \bowtie \&\{\ldots; ?l^{\downarrow}(C).B'; \ldots\} \tag{A.6.10}$$

From (A.6.3) and (A.6.10) we have there is T_1 such that

$$T \equiv T_1 \bowtie c : \left[\&\{\ldots; ?l^{\downarrow}(C).B'; \ldots\}\right] \tag{A.6.11}$$

From (A.6.7) we have that there is B_2 such that

$$L \mid B_2 \equiv (T' \bowtie B') \tag{A.6.12}$$

and

$$\uparrow B \equiv \uparrow B_2 \tag{A.6.13}$$

and

$$\downarrow B_2 \equiv B_1 \bowtie B' \tag{A.6.14}$$

We then have that

$$c \blacktriangleleft \left[Q'\right] :: ((L \bowtie c : [\downarrow B_2]) \mid loc(\uparrow B_2)) \bowtie a : C' \tag{A.6.15}$$

From (A.6.15), (A.6.14), (A.6.13), (A.6.3) and (A.6.11) we have that

$$T_1 \equiv (L \bowtie c : [B_1]) \mid loc(\uparrow B_2) \tag{A.6.16}$$

and

$$c \blacktriangleleft \left[Q'\right] :: (((L \bowtie c : [B_1]) \mid loc(\uparrow B_2)) \bowtie c : \left[B'\right]) \bowtie a : C' \tag{A.6.17}$$

which completes the proof for this case. ∎

Lemma A.7 *Let P be a well-typed process such that $P :: T$. If $P \xrightarrow{c\ \texttt{this}} Q$ 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. Follows by induction on the derivation of the label. We show the case of a **this** prefix transition.

(*Case* $\mathbf{this}(x).P' \overset{c\ \mathtt{this}}{\longrightarrow} P'\{x/c\}$)

We have that

$$\mathbf{this}(x).P' \overset{c\ \mathtt{this}}{\longrightarrow} P'\{x/c\} \tag{A.7.1}$$

and

$$\mathbf{this}(x).P' :: T \tag{A.7.2}$$

We have there are L, B_1, B_2 such that

$$T <: L \mid (B_1 \bowtie B_2) \tag{A.7.3}$$

and

$$P' :: L \mid B_1 \mid x : [B_2] \tag{A.7.4}$$

Let us consider there is T' such that

$$T' = (L \mid B_1) \bowtie c : [B_2] \tag{A.7.5}$$

Then considering Lemma 3.14 we have

$$P'\{x/c\} :: (L \mid B_1) \bowtie c : [B_2] \tag{A.7.6}$$

which completes the proof for this case. ∎

Lemma A.8 *Let P be a well-typed process such that $P :: T$. If $P \overset{c\ \mathtt{this}}{\longrightarrow} Q$ due to a synchronization then there are $T', B', B'', B_1, B_2, C_1, C_2, l$ such that $T \equiv T' \bowtie B' \bowtie c : [B'']$ and either $B' \equiv \oplus\{\ldots; !\, l^{\downarrow}(C_1).B_1; \ldots\}$ and $B'' \equiv \&\{\ldots; ?\, l^{\downarrow}(C_2).B_2; \ldots\}$, or $B' \equiv \&\{\ldots; ?\, l^{\downarrow}(C_1).B_1; \ldots\}$ and $B'' \equiv \oplus\{\ldots; !\, l^{\downarrow}(C_2).B_2; \ldots\}$. Furthermore if $C_1 \equiv C_2$ we have that $Q :: T' \bowtie B_1 \bowtie c : [B_2]$.*

Proof. Follows by induction on the derivation of the label. We show the case of a (*tco*) synchronization.

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

We have that

$$P_1 \mid P_2 \overset{c\ \mathtt{this}}{\longrightarrow} Q_1 \mid Q_2 \tag{A.8.1}$$

and

$$P_1 \mid P_2 :: T \tag{A.8.2}$$

(A.8.1) is derived from

$$P_1 \overset{l^{\downarrow}!(a)}{\longrightarrow} Q_1 \tag{A.8.3}$$

and

$$P_2 \overset{c\ l^{\downarrow}?(a)}{\longrightarrow} Q_2 \tag{A.8.4}$$

We have there are T_1, T_2 such that

$$T <: T_1 \bowtie T_2 \tag{A.8.5}$$

and

$$P_1 :: T_1 \tag{A.8.6}$$

and

$$P_2 :: T_2 \tag{A.8.7}$$

Considering Lemma A.3 and (A.8.3) and (A.8.6) we have there are T_1', C_1, B_1 such that

$$T_1 \equiv T_1' \bowtie \&\{\ldots; ?\, l^d(C_1).B_1; \ldots\} \tag{A.8.8}$$

and there is T_1'' such that

$$T_1' = T_1'' \bowtie a : C_1 \tag{A.8.9}$$

and

$$Q_1 :: T_1'' \bowtie B_1 \tag{A.8.10}$$

Considering Lemma A.2 and (A.8.4) and (A.8.7) we have there are T_2', C_2, B_2 such that

$$T_2 \equiv T_2' \bowtie c : \left[\oplus\{\ldots; !\, l^{\downarrow}(C_2).B_2; \ldots\}\right] \tag{A.8.11}$$

Let us now consider $C_1 \equiv C_2$. We then have that there is T_2'' such that

$$T_2'' = (T_2' \bowtie c : [B_2]) \bowtie a : C_2 \tag{A.8.12}$$

then

$$Q_2 :: T_2'' \tag{A.8.13}$$

From (A.8.10) and (A.8.13) we have

$$Q_1 \mid Q_2 :: (T_1'' \bowtie B_1) \bowtie (T_2' \bowtie c : [B_2]) \bowtie a : C_2 \tag{A.8.14}$$

which then, considering (A.8.9) and $C_1 \equiv C_2$, leads to

$$Q_1 \mid Q_2 :: (T_1' \bowtie B_1) \bowtie (T_2' \bowtie c : [B_2]) \tag{A.8.15}$$

which completes the proof for this case. ∎

Lemma A.9 *Let T_1 be a type such that $T_1 \to T_2$. If there is T_1' such that $T_1' <: T_1$ then there is T_2' such that $T_2 :> T_2'$ and $T_1' \to T_2'$.*

Proof. Follows by induction on the derivation of the subtype. Essentially any reduction of a supertype can be matched by a reduction on the subtype, and the subtype derivation can be reproduced for the arrival type term. We prove subtype (14).

(*Case* $M.B_1 \mid B_2 <: M.(B_1 \mid B_2)$)

We have that

$$M.(B_1 \mid B_2) \to T_2 \tag{A.9.1}$$

and

$$M.B_1 \mid B_2 <: M.(B_1 \mid B_2) \tag{A.9.2}$$

We have that either

$$M.(B_1 \mid B_2) \to B_1 \mid B_2 \tag{A.9.3}$$

or

$$M.(B_1 \mid B_2) \to M.(B_1' \mid B_2) \tag{A.9.4}$$

derived from

$$B_1 \to B_1' \tag{A.9.5}$$

or

$$M.(B_1 \mid B_2) \to M.(B_1 \mid B_2') \tag{A.9.6}$$

derived from

$$B_2 \to B_2' \tag{A.9.7}$$

For (A.9.3) we directly have that

$$M.B_1 \mid B_2 \to B_1 \mid B_2 \tag{A.9.8}$$

For (A.9.4) we have that

$$M.B_1 \mid B_2 \to M.B_1' \mid B_2 \tag{A.9.9}$$

and

$$M.B_1' \mid B_2 <: M.(B_1' \mid B_2) \tag{A.9.10}$$

and likewise for (A.9.6), thus completing the proof for this case. ∎

B Proof of Theorem 3.15

Let P be a process such that $P :: T$. If $P \to Q$ then there is T' such that $T \xrightarrow{*} T'$ and $Q :: T'$.

Proof. We proceed by induction on the derivation of the label. We consider the possible cases for synchronization, when P is of the form $P_1 \mid P_2$ and P_1 and P_2 synchronize on a message: either (*clo*) or (*com*) (Figure 3) synchronization on a unlocated $\downarrow$ transition, or a located $\downarrow$ transition. Also we consider c `this` transitions originating either from a **this** prefix or from a located/unlocated synchronization: either (*tco*) or (*tcl*) (Figure 4). We also consider that the τ transition might originate within to the scope of a restriction, or from one component of a parallel composition, or from the body of a recursive process or from within a context piece.

We give the proofs where the output originates in the left hand side of the parallel composition and the input in the right hand side, and omit the symmetric cases.

(**Synchronization**)

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

We have that

$$P_1 \mid P_2 \xrightarrow{\tau} Q_1 \mid Q_2 \tag{B.0.1}$$

and

$$P_1 \mid P_2 :: T \tag{B.0.2}$$

(B.0.1) is derived from

$$P_1 \xrightarrow{l^d!(a)} Q_1 \tag{B.0.3}$$

and

$$P_2 \xrightarrow{l^d?(a)} Q_2 \tag{B.0.4}$$

From (B.0.2) we conclude that there exists T' such that $T <: T'$ and

$$P_1 \mid P_2 :: T' \tag{B.0.5}$$

where (B.0.5) is derived from

$$P_1 :: T_1 \tag{B.0.6}$$

and

$$P_2 :: T_2 \tag{B.0.7}$$

for some T_1, T_2 such that

$$T' = T_1 \bowtie T_2 \tag{B.0.8}$$

Considering Lemma A.3 and (B.0.3) and (B.0.6) we conclude that there exist T_1', C_1, B_1 such that

$$T_1 \equiv T_1' \bowtie \&\{\ldots; ?\, l^d(C_1); B_1; \ldots\} \tag{B.0.9}$$

and there is T_1'' such that

$$T_1' = T_1'' \bowtie a : C_1 \tag{B.0.10}$$

and

$$Q_1 :: T_1'' \bowtie B_1 \tag{B.0.11}$$

Considering Lemma A.1 and (B.0.4) and (B.0.7) we conclude that there exist T_2', C_2, B_2 such that

$$T_2 \equiv T_2' \bowtie \oplus\{\ldots; !\, l^d(C_2); B_2; \ldots\} \tag{B.0.12}$$

We consider the two possible cases: either the label l is a shared label or a plain label. If l is a plain label then, from (B.0.8) we have that it must be the case that in T' there is a $\boldsymbol{\tau}$ for this synchronization and also that $C_1 \equiv C_2$ since otherwise (B.0.8) would not be defined. We use C such that $C \equiv C_1 \equiv C_2$.

From (B.0.8), (B.0.9), (B.0.10) and (B.0.12) we have

$$((T_1'' \bowtie a : C) \bowtie \&\{\ldots; ?\, l^d(C); B_1; \ldots\}) \bowtie (T_2' \bowtie \oplus\{\ldots; !\, l^d(C); B_2; \ldots\}) \tag{B.0.13}$$

From (B.0.13) we have

$$(T_2' \bowtie B_2) \bowtie a : C \tag{B.0.14}$$

Then from Lemma A.1 and (B.0.4) and (B.0.7) and (B.0.12) and (B.0.14) we have

$$Q_2 :: (T_2' \bowtie B_2) \bowtie a : C \tag{B.0.15}$$

From (B.0.11), (B.0.15) and (B.0.13) we have

$$Q_1 \mid Q_2 :: (T_1'' \bowtie B_1) \bowtie ((T_2' \bowtie B_2) \bowtie a : C) \tag{B.0.16}$$

Since l is a plain label we have from the definition of merge $\bowtie$ (Definition 3.11) and apartness $\#$ (Definition 3.10) that there must be a $\boldsymbol{\tau}$ in the resulting type term, and hence there is a corresponding $\boldsymbol{\tau}$ type transition (Figure 13 (1)). We thus have

$$\begin{array}{l} ((T_1'' \bowtie a : C) \bowtie \&\{\ldots; ?\, l^d(C_1); B_1; \ldots\}) \bowtie (T_2' \bowtie \oplus\{\ldots; !\, l^d(C_2); B_2; \ldots\}) \\ \equiv \\ (T_1'' \bowtie \oplus\{\ldots; \boldsymbol{\tau}\, l^d(C_1); (B_1 \bowtie B_2); \ldots\} \bowtie T_2') \bowtie a : C \\ \xrightarrow{*} \\ (T_1'' \bowtie B_1 \bowtie B_2 \bowtie T_2') \bowtie a : C \end{array} \tag{B.0.17}$$

From $T <: T'$, (B.0.8), (B.0.9), (B.0.10), (B.0.12) and we have

$$T <: ((T_1'' \mid \&\{\ldots; ?\, l^d(C_1); B_1; \ldots\}) \bowtie a : C) \bowtie (T_2' \mid \oplus\{\ldots; !\, l^d(C_2); B_2; \ldots\}) \tag{B.0.18}$$

which considering Lemma A.9 and (B.0.17) gives us there is T'' such that

$$(T_1'' \bowtie B_1 \bowtie B_2 \bowtie T_2') \bowtie a : C :> T'' \tag{B.0.19}$$

and

$$T \xrightarrow{*} T'' \tag{B.0.20}$$

which completes the proof for this case.

If l is a shared label then by conformance we have that $C_1 \equiv C_2 (\equiv C)$. From (B.0.8) we conclude

$$\oplus\{\ldots;!\, l^d(C); B_2; \ldots\}) \equiv \star\, !l^d(C) \tag{B.0.21}$$

and also the merge yields a $\boldsymbol{\tau}$ message type, as follows

$$\begin{array}{l}((T_1'' \bowtie a : C) \bowtie \&\{\ldots;?\, l^d(C); B_1; \ldots\}) \bowtie (T_2' \bowtie \star\, !\, l^d(C)) \\ \equiv \\ (T_1'' \bowtie (\&\{\ldots; \boldsymbol{\tau}\, l^d(C).B_1\{?\, l^d(C)/\boldsymbol{\tau}\, l^d(C)\}; \ldots\} \mid \star\, !\, l^d(C)) \bowtie T_2') \bowtie a : C \\ \xrightarrow{*} \\ (T_1'' \bowtie B_1 \bowtie \star\, !\, l^d(C) \bowtie T_2') \bowtie a : C\end{array} \tag{B.0.22}$$

From (B.0.22) we conclude

$$(T_2' \bowtie \star\, !\, l^d(C)) \bowtie a : C \tag{B.0.23}$$

Then from Lemma A.1 and (B.0.4) and (B.0.7) and (B.0.12) and (B.0.23) we have

$$Q_2 :: (T_2' \bowtie \star\, !\, l^d(C)) \bowtie a : C \tag{B.0.24}$$

From (B.0.11), (B.0.24) and (B.0.13) we have

$$Q_1 \mid Q_2 :: (T_1'' \bowtie B_1 \bowtie \star\, !\, l^d(C) \bowtie T_2') \bowtie a : C \tag{B.0.25}$$

From $T <: T'$, (B.0.8), (B.0.9), (B.0.10), (B.0.12) and we have

$$T <: (T_1'' \bowtie (\&\{\ldots; \boldsymbol{\tau}\, l^d(C).B_1\{?\, l^d(C)/\boldsymbol{\tau}\, l^d(C)\}; \ldots\} \mid \star\, !\, l^d(C)) \bowtie T_2') \bowtie a : C \tag{B.0.26}$$

which considering Lemma A.9 and (B.0.17) gives us there is T'' such that

$$(T_1'' \bowtie B_1 \bowtie \star\, !\, l^d(C) \bowtie T_2') \bowtie a : C :> T'' \tag{B.0.27}$$

and

$$T \xrightarrow{*} T'' \tag{B.0.28}$$

which completes the proof for this case.

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

Analogous to the previous case. The difference is the transition is located and so are the types given by Lemmas A.2 and A.4, being the structure of the proof identical.

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

Analogous. The difference is the synchronization takes place in rule (*clo*), so we have that the name is fresh to P_2 and hence the condition on the existence of the merge of Lemma A.1 is directly satisfied. We recover the closed under internal

interaction type from the merge of $a : C'$ and $a : C$, given in the conclusion of Lemma A.5, where $a : C'$ is in the type of Q_1 and $a : C$ is in the type of Q_2.

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

Analogous.

(*Case* $P \xrightarrow{c\ \texttt{this}} Q$)

We have that

$$c \blacktriangleleft [P] \xrightarrow{\tau} c \blacktriangleleft [Q] \tag{B.0.29}$$

and

$$c \blacktriangleleft [P] :: T \tag{B.0.30}$$

(B.0.29) is derived from

$$P \xrightarrow{c\ \texttt{this}} Q \tag{B.0.31}$$

From (B.0.30) we conclude that there exists T' such that $T <: T'$ and

$$c \blacktriangleleft [P] :: T' \tag{B.0.32}$$

where (B.0.32) is derived from

$$P :: L \mid B \tag{B.0.33}$$

for L, B such that

$$T' = (L \bowtie c : [\downarrow B]) \mid loc(\uparrow B) \tag{B.0.34}$$

We must consider the two distinct cases: either the transition originates from a **this** prefix or from a located/unlocated synchronization on a message.

(**this** *prefix*) For the case where the transition originates in a **this** prefix we consider Lemma A.7 and (B.0.33) and (B.0.31) from which we conclude there are L', B_1, B_2 such that

$$L \mid B \equiv (L' \mid B_1) \bowtie (\downarrow B_2) \tag{B.0.35}$$

and, considering (B.0.34), we have

$$Q :: (L' \mid B_1) \bowtie (c : [\downarrow B_2]) \tag{B.0.36}$$

From (B.0.36) we derive

$$c \blacktriangleleft [Q] :: ((L' \bowtie c : [\downarrow B_2]) \bowtie c : [\downarrow B_1]) \mid loc(\uparrow B_1) \tag{B.0.37}$$

From (B.0.35) we conclude there exists L', B' such that $L \equiv L'$ and $B \equiv B'$ and

$$\downarrow B' \ = \downarrow B_1 \bowtie \downarrow B_2 \tag{B.0.38}$$

and

$$\uparrow B' \ \equiv \uparrow B_1 \tag{B.0.39}$$

From (B.0.37), (B.0.38), (B.0.39), $L \equiv L'$ and $B \equiv B'$ we have

$$c \blacktriangleleft [Q] :: (L \bowtie c : [\downarrow B]) \mid loc(\uparrow B) \tag{B.0.40}$$

From (B.0.40) and (B.0.34) and $T <: T'$ we have that

$$c \blacktriangleleft [Q] :: T \tag{B.0.41}$$

which completes the proof for this case.

(*located/unlocated message synchronization*) For the case where the transition originates in a located/unlocated message synchronization we consider Lemma A.8 and (B.0.33) and (B.0.31) and conclude there exist $T', B', B'', B_1, B_2, C_1, C_2, l$ such that

$$L \mid B \equiv T' \bowtie B' \bowtie c : \left[B''\right] \tag{B.0.42}$$

and either

$$B' \equiv \oplus\{\ldots; !\, l^{\downarrow}(C_1).B_1; \ldots\} \tag{B.0.43}$$

and

$$B'' \equiv \&\{\ldots; ?\, l^{\downarrow}(C_2).B_2; \ldots\} \tag{B.0.44}$$

or

$$B' \equiv \&\{\ldots; ?\, l^{\downarrow}(C_1).B_1; \ldots\} \tag{B.0.45}$$

and

$$B'' \equiv \oplus\{\ldots; !\, l^{\downarrow}(C_2).B_2; \ldots\} \tag{B.0.46}$$

We prove the case when (B.0.43) and (B.0.44), being the proof analogous for (B.0.45) and (B.0.46). From (B.0.42) we conclude there exist L', B''' such that $T' \equiv L' \mid B'''$ and $L \equiv L' \bowtie c : [B'']$ and $B \equiv B''' \bowtie B'$. From (B.0.34) and $T <: T'$ we then have

$$T' <: ((L' \mid c : \left[B''\right]) \bowtie c : \left[\downarrow (B''' \mid B')\right]) \mid loc(\uparrow (B''' \mid B')) \tag{B.0.47}$$

We consider the two possible cases: either l is a plain label or l is a shared label. If l is a plain label, from (B.0.43) and (B.0.44) we have that the merge of (B.0.47) is only defined if $C_1 \equiv C_2$ ($\equiv C$) and yields a type such that

$$\begin{array}{l}((L' \mid c : [B'']) \bowtie c : [\downarrow (B''' \mid B')]) \mid loc(\uparrow (B''' \mid B')) \\ \equiv \\ c : [\oplus\{\ldots; \boldsymbol{\tau}\ l^{\downarrow}(C).((\downarrow B_1) \bowtie B_2); \ldots\}] \bowtie (L' \bowtie c : [\downarrow (B''')]) \mid loc(\uparrow (B''' \mid B_1))\end{array} \tag{B.0.48}$$

where $\uparrow B' \equiv \uparrow B_1$, for which we have that

$$\begin{array}{l}c : [\oplus\{\ldots; \boldsymbol{\tau}\ l^{\downarrow}(C).((\downarrow B_1) \bowtie B_2); \ldots\}] \bowtie (L' \bowtie c : [\downarrow (B''')]) \mid loc(\uparrow (B''' \mid B_1)) \\ \rightarrow \\ c : [(\downarrow B_1) \bowtie B_2] \bowtie (L' \bowtie c : [\downarrow (B''')]) \mid loc(\uparrow (B''' \mid B_1))\end{array} \tag{B.0.49}$$

From $C_1 \equiv C_2$ and Lemma A.8 we then have

$$Q :: T' \bowtie B_1 \bowtie c : [B_2] \tag{B.0.50}$$

From (B.0.50) and $T' \equiv L' \mid B'''$ we have

$$Q :: (L' \mid B''') \bowtie B_1 \bowtie c : [B_2] \tag{B.0.51}$$

From (B.0.51) we derive

$$c \blacktriangleleft [Q] :: ((L' \mid c : [B_2]) \bowtie\ c : \left[\downarrow (B''' \mid B_1)\right]) \mid loc(\uparrow (B''' \mid B_1)) \tag{B.0.52}$$

From (B.0.52) we have

$$c \blacktriangleleft [Q] :: c : [(\downarrow B_1) \bowtie B_2] \bowtie (L' \bowtie c : \left[\downarrow (B''')\right]) \mid loc(\uparrow (B''' \mid B_1)) \tag{B.0.53}$$

which completes the proof for this case. If l is a shared label, from (B.0.43) and (B.0.44) we have that the merge of (B.0.47) is only defined if $C_1 \equiv C_2$ ($\equiv C$) and $\downarrow B' \equiv \star \,!\, l^{\downarrow}(C)$ and yields a type such that

$$\begin{array}{l}((L' \mid c : [B'']) \bowtie c : [\downarrow (B''' \mid B')]) \mid loc(\uparrow (B''' \mid B')) \\ \equiv \\ (c : [\star \,!\, l^{\downarrow}(C)] \mid c : \left[\&\{\ldots; \boldsymbol{\tau}\ l^{\downarrow}(C).B_2\{?\ l^d(C)/\boldsymbol{\tau}\ l^d(C)\}; \ldots\}\right]) \\ \bowtie (L' \bowtie c : [\downarrow (B''')]) \mid loc(\uparrow (B''' \mid B_1))\end{array} \tag{B.0.54}$$

where $\uparrow B' \equiv \uparrow B_1$, for which we have that

$$\begin{array}{l}(c : [\star \,!\, l^{\downarrow}(C)] \mid c : \left[\&\{\ldots; \boldsymbol{\tau}\ l^{\downarrow}(C).B_2\{?\ l^d(C)/\boldsymbol{\tau}\ l^d(C)\}; \ldots\}\right]) \\ \bowtie (L' \bowtie c : [\downarrow (B''')]) \mid loc(\uparrow (B''' \mid B_1)) \\ \xrightarrow{*} \\ c : [\star \,!\, l^{\downarrow}(C)] \bowtie c : [B_2] \bowtie (L' \bowtie c : [\downarrow (B''')]) \mid loc(\uparrow (B''' \mid B_1))\end{array} \tag{B.0.55}$$

From $C_1 \equiv C_2$ and Lemma A.8 we then have

$$Q :: T' \bowtie B_1 \bowtie c : [B_2] \tag{B.0.56}$$

From (B.0.56) and $T' \equiv L' \mid B'''$ we have

$$Q :: (L' \mid B''') \bowtie B_1 \bowtie c : [B_2] \tag{B.0.57}$$

From (B.0.57) we derive

$$c \blacktriangleleft [Q] :: ((L' \mid c : [B_2]) \bowtie\ c : \left[\downarrow (B''' \mid B_1)\right]) \mid loc(\uparrow (B''' \mid B_1)) \tag{B.0.58}$$

From (B.0.58) and $\downarrow B' \equiv \star \,!\, l^{\downarrow}(C)$ we have

$$c \blacktriangleleft [Q] :: c : \left[\star \,!\, l^{\downarrow}(C) \bowtie B_2\right] \bowtie (L' \bowtie c : \left[\downarrow (B''')\right]) \mid loc(\uparrow (B''' \mid B_1)) \tag{B.0.59}$$

which completes the proof.

(τ **transition**)

(*Case* $(\nu a)P \rightarrow (\nu a)Q$)

We have that

$$(\nu a)P \rightarrow (\nu a)Q \tag{B.0.60}$$

and

$$(\nu a)P :: T \tag{B.0.61}$$

(B.0.60) is derived from

$$P \rightarrow Q \tag{B.0.62}$$

We have there is T', B such that

$$T <: T' \mid a : [B] \tag{B.0.63}$$

and $closed(B)$ and

$$P :: T' \mid a : [B] \tag{B.0.64}$$

By induction hypothesis on (B.0.62) and (B.0.64) we have there is T'' such that

$$T' \mid a : [B] \xrightarrow{*} T'' \tag{B.0.65}$$

and

$$Q :: T'' \tag{B.0.66}$$

We consider the two possible cases: either (B.0.65) is derived from $T' \xrightarrow{*} T_1$ and $T'' \equiv T_1 \mid a : [B]$ or there is B' such that $B \xrightarrow{*} B'$ and $T'' \equiv T' \mid a : [B']$. In the first case we have that

$$(\nu a)Q :: T_1 \tag{B.0.67}$$

which completes the proof. In the second case we have that it must be the case that $closed(B')$ thus we derive

$$(\nu a)Q :: T' \tag{B.0.68}$$

and, since $T <: T'$ we have

$$(\nu a)Q :: T \tag{B.0.69}$$

thus completing the proof for this case.

($Case\ P \mid R \rightarrow Q \mid R$)

We have that

$$P \mid R \rightarrow Q \mid R \tag{B.0.70}$$

and

$$P \mid R :: T \tag{B.0.71}$$

(B.0.70) is derived from

$$P \rightarrow Q \tag{B.0.72}$$

We have there is T_1, T_2 such that

$$T <: T_1 \bowtie T_2 \tag{B.0.73}$$

and

$$P \mid R :: T_1 \bowtie T_2 \tag{B.0.74}$$

and

$$P :: T_1 \tag{B.0.75}$$

and

$$R :: T_2 \tag{B.0.76}$$

By induction hypothesis on (B.0.72) and (B.0.75) we have there is T_1' such that

$$T_1 \xrightarrow{*} T_1' \tag{B.0.77}$$

and

$$Q :: T_1' \tag{B.0.78}$$

From (B.0.78) and (B.0.76) we derive

$$Q \mid R :: T_1' \bowtie T_2 \tag{B.0.79}$$

and from (B.0.77) we have

$$T_1 \bowtie T_2 \xrightarrow{*} T_1' \bowtie T_2 \tag{B.0.80}$$

which considering Lemma A.9 completes the proof for this case.

(*Case* $\mathbf{rec}\ \mathcal{X}.P \to Q$)

We have that

$$\mathbf{rec}\ \mathcal{X}.P \to Q \tag{B.0.81}$$

and

$$\mathbf{rec}\ \mathcal{X}.P :: T \tag{B.0.82}$$

(B.0.81) is derived from

$$P\{\mathcal{X}/\mathbf{rec}\ \mathcal{X}.P\} \to Q \tag{B.0.83}$$

We have there is $L_M, B\langle\mathcal{X}\rangle$ such that

$$T <: \star L_M \mid \mathtt{rec}\ \mathcal{X}.B\langle\mathcal{X}\rangle \tag{B.0.84}$$

and

$$P :: L_M \mid B\langle\mathcal{X}\rangle \tag{B.0.85}$$

From (B.0.85) we have

$$P\{\mathcal{X}/\mathbf{rec}\ \mathcal{X}.P\} :: \star L_M \mid L_M \mid B\langle\mathtt{rec}\ \mathcal{X}.B\langle\mathcal{X}\rangle\rangle \tag{B.0.86}$$

and also

$$P\{\mathcal{X}/\mathbf{rec}\ \mathcal{X}.P\} :: \star L_M \mid B\langle\mathtt{rec}\ \mathcal{X}.B\langle\mathcal{X}\rangle\rangle \tag{B.0.87}$$

By induction hypothesis on (B.0.83) and (B.0.87) we have there is T' such that

$$\star L_M \mid B\langle\mathtt{rec}\ \mathcal{X}.B\langle\mathcal{X}\rangle\rangle \xrightarrow{*} T' \tag{B.0.88}$$

and

$$Q :: T' \tag{B.0.89}$$

From (B.0.88) and (B.0.84) we derive

$$T \xrightarrow{*} T' \tag{B.0.90}$$

thus completing the proof for this case.

(*Case* $c \blacktriangleleft [P] \to c \blacktriangleleft [Q]$)

We have that

$$c \blacktriangleleft [P] \to c \blacktriangleleft [Q] \tag{B.0.91}$$

and

$$c \blacktriangleleft [P] :: T \tag{B.0.92}$$

(B.0.91) is derived from

$$P \to Q \tag{B.0.93}$$

We have there is L, B such that

$$T <: (L \bowtie c : [\downarrow B]) \mid loc(\uparrow B) \tag{B.0.94}$$

and

$$c \blacktriangleleft [P] :: (L \bowtie c : [\downarrow B]) \mid loc(\uparrow B) \tag{B.0.95}$$

and

$$P :: L \mid B \tag{B.0.96}$$

By induction hypothesis on (B.0.93) and (B.0.96) we have there is T' such that

$$L \mid B \xrightarrow{*} T' \tag{B.0.97}$$

We then have there is L', B' such that

$$T' \equiv L' \mid B' \tag{B.0.98}$$

and

$$c \blacktriangleleft [Q] :: (L' \bowtie c : \left[\downarrow B'\right]) \mid loc(\uparrow B') \tag{B.0.99}$$

which along with

$$T \xrightarrow{*} (L' \bowtie c : \left[\downarrow B'\right]) \mid loc(\uparrow B') \tag{B.0.100}$$

completes the proof. ∎

C Proof of Theorem 3.18

Let P be a process such that $P :: T$. Then P is not an error process.

Proof. (*Sketch*) Aiming at a contradiction let us assume that P is an error process (Definition 3.17) which then gives us there are $\mathcal{C}, Q, R, Q', R', \lambda, \lambda'$ such that $P = \mathcal{C}\,[Q \mid R]$ and

$$Q \xrightarrow{\lambda} Q' \tag{C.0.1}$$

and

$$R \xrightarrow{\lambda'} R' \tag{C.0.2}$$

and $w(\lambda) = w(\lambda')$, and the label in $w(\lambda)$ is not shared. We have that there are T_1, T_2 such that

$$Q \mid R :: T_1 \bowtie T_2 \tag{C.0.3}$$

and

$$Q :: T_1 \tag{C.0.4}$$

and

$$R :: T_2 \tag{C.0.5}$$

We consider the case when $\lambda = l^{\downarrow}?(a)$ and $\lambda' = l^{\downarrow}?(b)$, being the proof analogous for any other such labels λ and λ' such that $w(\lambda) = w(\lambda')$ and the label in $w(\lambda)$ is not shared. From (C.0.1) and (C.0.4) and Lemma A.1 we conclude there is T_1', B_1, C_1 such that

$$T_1 \equiv T_1' \bowtie \oplus\{\ldots; !\; l^{\downarrow}(C_1).B_1; \ldots\} \tag{C.0.6}$$

and from (C.0.2) and (C.0.5) and Lemma A.1 we conclude there is T_2', B_2, C_2 such that

$$T_2 \equiv T_2' \bowtie \oplus\{\ldots; !\; l^{\downarrow}(C_2).B_2; \ldots\} \tag{C.0.7}$$

From (C.0.3) and (C.0.6) and (C.0.7) we have there must be a type B such that

$$B = \oplus\{\ldots; p_1\; l^{\downarrow}(C_1).B_1; \ldots\} \bowtie \oplus\{\ldots; p_2\; l^{\downarrow}(C_2).B_2; \ldots\} \tag{C.0.8}$$

which gives us our intended contradiction since it is not possible to synchronize (the polarities are not dual) or interleave (since they are not apart #) these two types, and hence it is not possible to merge them. ∎

D Theorem 4.4 auxiliary results

We start by the proof of the Substitution Lemma (4.3) and the statements of the other Lemmas used in the proofs of the Theorems.

Proof of Lemma 4.3

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

Proof. Follows by induction on the structure of P. Essentially the condition $\Gamma'\{x/n\} \subseteq \Gamma$ ensures the ordering already prescribed for n in Γ copes with the ordering required for conversation x. We show the case when P is a context piece and is an output prefix.

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

We have that

$$\Gamma \cup \Gamma' \vdash_\ell x \blacktriangleleft [P] \tag{D.0.1}$$

derived from

$$\Gamma \cup \Gamma' \vdash_{(\ell(\downarrow),x)} P \tag{D.0.2}$$

and we have that

$$\Gamma'\{x/n\} \subseteq \Gamma \tag{D.0.3}$$

By induction hypothesis on (D.0.2) and (D.0.3) we have

$$\Gamma \vdash_{(\ell(\downarrow)\{x/n\},n)} P\{x/n\} \tag{D.0.4}$$

From (D.0.4) we derive

$$\Gamma \vdash_{\ell\{x/n\}} n \blacktriangleleft [P\{x/n\}] \tag{D.0.5}$$

which completes the proof for this case.

(*Case* $l^d!(o).P$)

We have that

$$\Gamma \cup \Gamma' \vdash_\ell l^d!(o).P \tag{D.0.6}$$

derived from

$$(\ell(d).l.(y)\Gamma''\bot(\Gamma \cup \Gamma')) \vdash_\ell P \tag{D.0.7}$$

and

$$\Gamma''\{y/o\} \subseteq (\ell(d).l.(y)\Gamma'\bot(\Gamma \cup \Gamma')) \tag{D.0.8}$$

and we have that

$$\Gamma'\{x/n\} \subseteq \Gamma \tag{D.0.9}$$

From (D.0.9) we conclude there is Γ_1, Γ_2 such that $\Gamma_2 \subseteq \Gamma$ and $\Gamma_1 \subseteq \Gamma'$ and

$$(\ell(d).l.(y)\Gamma''\bot(\Gamma \cup \Gamma')) = \Gamma_1 \cup \Gamma_2 \tag{D.0.10}$$

and

$$\Gamma_1\{x/n\} \subseteq \Gamma_2 \tag{D.0.11}$$

By induction hypothesis on (D.0.7) and (D.0.11), considering (D.0.10) we have

$$\Gamma_2 \vdash_{\ell\{x/n\}} P\{x/n\} \tag{D.0.12}$$

We have that either

$$(\ell(d).l.(y)\Gamma'' \perp \Gamma_1) = \Gamma_1 \tag{D.0.13}$$

or

$$(\ell(d).l.(y)\Gamma'' \perp \Gamma_2) = \Gamma_2 \tag{D.0.14}$$

If (D.0.14) we directly have that

$$\Gamma''\{y/o\} \subseteq \Gamma_2 \tag{D.0.15}$$

From (D.0.14) and (D.0.15) we conclude

$$\Gamma_2 \vdash_{\ell\{x/n\}} l^d!(o).P\{x/n\} \tag{D.0.16}$$

From $\Gamma_2 \subseteq \Gamma$ and considering Lemma D.1 we conclude

$$\Gamma \vdash_{\ell\{x/n\}} l^d!(o).P\{x/n\} \tag{D.0.17}$$

which completes the proof for (D.0.14).

If (D.0.13) it must be the case that $\ell(d) = x$. From (D.0.11) we have

$$((\ell(d).l.(y)\Gamma'')\{x/n\} \perp \Gamma_1\{x/n\}) = \Gamma_1\{x/n\} \tag{D.0.18}$$

From (D.0.10) we have

$$(\ell(d).l.(y)\Gamma'') \perp (\Gamma_1 \cup \Gamma_2) = \Gamma_1 \cup \Gamma_2 \tag{D.0.19}$$

Since substitution is order preserving we have

$$(\ell(d).l.(y)\Gamma'')\{x/n\} \perp (\Gamma_1 \cup \Gamma_2)\{x/n\} = \Gamma_1 \cup \Gamma_2\{x/n\} \tag{D.0.20}$$

From (D.0.20) and (D.0.11) we conclude

$$(\ell(d).l.(y)\Gamma'')\{x/n\} \perp \Gamma_2 = \Gamma_2 \tag{D.0.21}$$

From (D.0.8) and (D.0.10) we conclude

$$\Gamma''\{y/o\}\{x/n\} \subseteq \Gamma_2 \tag{D.0.22}$$

We then have from (D.0.21) and (D.0.22) that

$$\Gamma_2 \vdash_{\ell\{x/n\}} l^d!(o).P\{x/n\} \tag{D.0.23}$$

From $\Gamma_2 \subseteq \Gamma$ and considering Lemma D.1 we conclude

$$\Gamma \vdash_{\ell\{x/n\}} l^d!(o).P\{x/n\} \tag{D.0.24}$$

which completes the proof. ∎

Lemma D.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. Follows by induction on the structure of P. 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. We show the case when P is an output prefixed process.

(*Case* $l^d!(n).P$)

We have that

$$\Gamma \vdash_\ell l^d!(o).P \tag{D.1.1}$$

derived from

$$(\ell(d).l.(x)\Gamma'\bot\Gamma) \vdash_\ell P \tag{D.1.2}$$

and

$$\Gamma'\{x/n\} \subseteq (\ell(d).l.(y)\Gamma'\bot\Gamma) \tag{D.1.3}$$

Let us consider Γ'' such that $\Gamma \cup \Gamma''$ is an event ordering. We then have that

$$(\ell(d).l.(x)\Gamma'\bot(\Gamma \cup \Gamma'')) \tag{D.1.4}$$

is an event ordering. By induction hypothesis we conclude

$$(\ell(d).l.(x)\Gamma'\bot(\Gamma \cup \Gamma'')) \vdash_\ell P \tag{D.1.5}$$

We have that

$$(\ell(d).l.(x)\Gamma'\bot\Gamma) \subseteq (\ell(d).l.(x)\Gamma'\bot(\Gamma \cup \Gamma'')) \tag{D.1.6}$$

From (D.1.3) and (D.1.6) we conclude

$$\Gamma'\{x/n\} \subseteq (\ell(d).l.(x)\Gamma'\bot(\Gamma \cup \Gamma'')) \tag{D.1.7}$$

From (D.1.5) and (D.1.7) we conclude

$$\Gamma \cup \Gamma'' \vdash_\ell l^d!(o).P \tag{D.1.8}$$

which completes the proof for this case. ∎

Lemma D.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 $(\ell(d).l.(x)\Gamma')\bot\Gamma$ and $\Gamma'\{x/a\} \subseteq (\ell(d).l.(x)\Gamma')\bot\Gamma$ then $\Gamma \vdash_\ell Q$.*

Proof. Follows by induction on the derivation of the label. We show the case when P is an input summation.

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

We have that

$$\Gamma \vdash_\ell \Sigma_{i\in I}\, l_i^d?(x_i).P_i \tag{D.2.1}$$

Let us consider

$$\Sigma_{i\in I}\, l_i^d?(x_i).P_i \xrightarrow{l_j^d?(a)} P_j\{x_j/a\} \tag{D.2.2}$$

and

$$(\ell(d).l_i.(y)\Gamma_j'\bot\Gamma) \tag{D.2.3}$$

and

$$\Gamma_j'\{y/a\} \subseteq (\ell(d).l_i.(y)\Gamma_j'\bot\Gamma) \tag{D.2.4}$$

We have that (D.2.1) is derived from

$$(\ell(d).l_i.(y)\Gamma'_i \bot \Gamma) \cup \Gamma'_i\{y/x_i\} \vdash_\ell P_i \tag{D.2.5}$$

in particular for j we have

$$(\ell(d).l_i.(y)\Gamma'_j \bot \Gamma) \cup \Gamma'_j\{y/x_j\} \vdash_\ell P_j \tag{D.2.6}$$

From Lemma 4.3 considering (D.2.6) and (D.2.4) we then have

$$(\ell(d).l_i.(y)\Gamma'_j \bot \Gamma) \vdash_\ell P_j\{x_j/a\} \tag{D.2.7}$$

where $\ell\{x_j/a\} = \ell$ since a reduction can not take place under a conversation which has as identifier a variable (transitions do not originate in processes that are prefixed by an input). From (D.2.7) and considering Lemma D.1 we have

$$\Gamma \vdash_\ell P_j\{x_j/a\} \tag{D.2.8}$$

which completes the proof for this case. ∎

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

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

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

We have that

$$\Gamma \vdash_\ell c \blacktriangleleft \left[P'\right] \tag{D.3.1}$$

Let us consider

$$c \blacktriangleleft \left[P'\right] \overset{c\ l^{\downarrow}?(a)}{\longrightarrow} c \blacktriangleleft \left[Q'\right] \tag{D.3.2}$$

and

$$(c.l.(x)\Gamma' \bot \Gamma) \tag{D.3.3}$$

and

$$\Gamma'\{x/a\} \subseteq (c.l.(x)\Gamma' \bot \Gamma) \tag{D.3.4}$$

We have that (D.3.1) is derived from

$$\Gamma \vdash_{(\ell(\downarrow),c)} P' \tag{D.3.5}$$

and (D.3.2) is derived from

$$P' \overset{l^{\downarrow}?(a)}{\longrightarrow} Q' \tag{D.3.6}$$

From Lemma D.2 considering (D.2.6), (D.2.5), (D.2.4) and (D.2.3) we have

$$\Gamma \vdash_{(\ell(\downarrow),c)} Q' \tag{D.3.7}$$

From (D.3.7) we derive

$$\Gamma \vdash_\ell c \blacktriangleleft \left[Q'\right] \tag{D.3.8}$$

which completes the proof for this case. ∎

Lemma D.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 $(\ell(d).l.(x)\Gamma')\bot\Gamma$ and $\Gamma'\{x/a\} \subseteq (\ell(d).l.(x)\Gamma')\bot\Gamma$.*

Proof. Follows by induction on the derivation of the label. We show the case when P is an output prefix.

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

We have that

$$\Gamma \vdash_\ell l^d!(a).P' \tag{D.4.1}$$

Let us consider

$$l^d!(a).P' \xrightarrow{l^d!(a)} P' \tag{D.4.2}$$

We have that (D.4.1) is derived from

$$(\ell(d).l.(x)\Gamma'\bot\Gamma) \vdash_\ell P' \tag{D.4.3}$$

and

$$\Gamma'\{x/a\} \subseteq (\ell(d).l.(x)\Gamma'\bot\Gamma) \tag{D.4.4}$$

From (D.4.3) and considering Lemma D.1 we have

$$\Gamma \vdash_\ell P' \tag{D.4.5}$$

which completes the proof for this case. ∎

Lemma D.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 $(c.l.(x)\Gamma')\bot\Gamma$ and $\Gamma'\{x/a\} \subseteq (c.l.(x)\Gamma')\bot\Gamma$.*

Proof. Follows 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

$$\Gamma \vdash_\ell c \blacktriangleleft \left[P'\right] \tag{D.5.1}$$

Let us consider

$$c \blacktriangleleft \left[P'\right] \xrightarrow{c\ l^\downarrow!(a)} c \blacktriangleleft \left[Q'\right] \tag{D.5.2}$$

We have that (D.5.1) is derived from

$$\Gamma \vdash_{(\ell(\downarrow),c)} P' \tag{D.5.3}$$

and (D.5.2) is derived from

$$P' \xrightarrow{l^\downarrow!(a)} Q' \tag{D.5.4}$$

From (D.5.3) and (D.5.4), considering Lemma D.4 we have

$$(c.l.(x)\Gamma'\bot\Gamma) \vdash_\ell P' \tag{D.5.5}$$

and

$$\Gamma'\{x/a\} \subseteq (c.l.(x)\Gamma'\bot\Gamma) \tag{D.5.6}$$

and

$$\Gamma \vdash_{(\ell(\downarrow),c)} Q' \tag{D.5.7}$$

From (D.5.7) we conclude

$$\Gamma \vdash_\ell c \blacktriangleleft [Q'] \tag{D.5.8}$$

which completes the proof for this case. ∎

Lemma D.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 $(\ell(d).l.(x)\Gamma'')\bot\Gamma$ and $\Gamma''\{x/a\} \subseteq ((\ell(d).l.(x)\Gamma'')\bot(\Gamma \cup \Gamma'))$.*

Proof. Follows by induction on the derivation of the label. We show the base case of restriction open (Fig. 3 (*opn*)).

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

We have that

$$\Gamma \vdash_\ell (\nu a)P' \tag{D.6.1}$$

Let us consider

$$(\nu a)P' \xrightarrow{(\nu a) l^d!(a)} Q' \tag{D.6.2}$$

We have that (D.6.1) is derived from

$$\Gamma' \vdash_\ell P' \tag{D.6.3}$$

where $\Gamma = \Gamma' \setminus a$. (D.6.2) is derived from

$$P' \xrightarrow{l^d!(a)} Q' \tag{D.6.4}$$

From (D.6.3) and (D.6.4), considering Lemma D.4 we have

$$(\ell(d).l.(x)\Gamma''\bot\Gamma') \vdash_\ell P' \tag{D.6.5}$$

and

$$\Gamma''\{x/a\} \subseteq (\ell(d).l.(x)\Gamma''\bot\Gamma') \tag{D.6.6}$$

and

$$\Gamma' \vdash_\ell Q' \tag{D.6.7}$$

Since $\ell(d) \neq a$ from (D.6.5) we conclude

$$(\ell(d).l.(x)\Gamma''\bot\Gamma) \tag{D.6.8}$$

which completes the proof for this case. ∎

Lemma D.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 $(c.l.(x)\Gamma'')\bot\Gamma$ and $\Gamma''\{x/a\} \subseteq ((c.l.(x)\Gamma'')\bot(\Gamma \cup \Gamma'))$.*

Proof. Follows by induction on the derivation of the label. We show the base cases of restriction open (Fig. 3 (*opn*)) 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

$$\Gamma \vdash_\ell c \blacktriangleleft \left[P'\right] \tag{D.7.1}$$

Let us consider

$$c \blacktriangleleft \left[P'\right] \overset{(\nu a)c\ l^{\downarrow}!(a)}{\longrightarrow} c \blacktriangleleft \left[Q'\right] \tag{D.7.2}$$

We have that (D.7.1) is derived from

$$\Gamma \vdash_{(\ell(\downarrow),c)} P' \tag{D.7.3}$$

and (D.7.2) is derived from

$$P' \overset{(\nu a)l^{\downarrow}!(a)}{\longrightarrow} Q' \tag{D.7.4}$$

From (D.7.3) and (D.7.4), considering Lemma D.6 we have there is Γ' such that

$$\Gamma \cup \Gamma' \vdash_{(\ell(\downarrow),c)} Q' \tag{D.7.5}$$

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

$$(c.l.(x)\Gamma''\bot\Gamma) \tag{D.7.6}$$

and

$$\Gamma''\{x/a\} \subseteq (c.l.(x)\Gamma''\bot(\Gamma \cup \Gamma')) \tag{D.7.7}$$

From (D.7.5) we conclude

$$\Gamma \cup \Gamma' \vdash_\ell c \blacktriangleleft \left[Q'\right] \tag{D.7.8}$$

which completes the proof for this case.

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

We have that

$$\Gamma \vdash_\ell (\nu a)P' \tag{D.7.9}$$

Let us consider

$$(\nu a)P' \overset{(\nu a)c\ l^{\downarrow}!(a)}{\longrightarrow} Q' \tag{D.7.10}$$

We have that (D.7.9) is derived from

$$\Gamma' \vdash_\ell P' \tag{D.7.11}$$

where $\Gamma = \Gamma' \setminus a$. (D.7.10) is derived from

$$P' \overset{c\ l^{\downarrow}!(a)}{\longrightarrow} Q' \tag{D.7.12}$$

From (D.7.11) and (D.7.12), considering Lemma D.5 we have

$$(c.l.(x)\Gamma''\bot\Gamma') \vdash_\ell P' \tag{D.7.13}$$

and

$$\Gamma''\{x/a\} \subseteq (c.l.(x)\Gamma''\bot\Gamma') \tag{D.7.14}$$

and

$$\Gamma' \vdash_\ell Q' \tag{D.7.15}$$

Since $c \neq a$ from (D.7.13) we conclude

$$(c.l.(x)\Gamma''\bot\Gamma) \tag{D.7.16}$$

which completes the proof for this case. ∎

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

Proof. Follows by induction on the derivation of the label. We show the base case when P is a **this** prefixed process.

(*Case* $\mathbf{this}(x).P' \xrightarrow{c\ \mathtt{this}} P'\{x/c\}$)

We have that

$$\Gamma \vdash_\ell \mathbf{this}(x).P' \tag{D.8.1}$$

Let us consider

$$\mathbf{this}(x).P' \xrightarrow{c\ \mathtt{this}} P'\{x/c\} \tag{D.8.2}$$

and $\ell(\downarrow) = c$. We have that (D.8.1) is derived from

$$\Gamma \cup \Gamma' \vdash_\ell P' \tag{D.8.3}$$

where $\Gamma'\{x/\ell(\downarrow)\} \subseteq \Gamma$, hence $\Gamma'\{x/c\} \subseteq \Gamma$. From Lemma 4.3 we then have

$$\Gamma \vdash_\ell P'\{x/c\} \tag{D.8.4}$$

which completes the proof for this case. ∎

E Proof of Theorem 4.4

Let P be a well typed process $P :: T$ and Γ an event ordering such that $\Gamma \vdash_\ell P$. If there is Q such that $P \to Q$ then $\Gamma \vdash_\ell Q$.

Proof. Follows by induction on the derivation of the label. We consider the possible cases for synchronization, `this` label, and τ transitions.

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

We have that

$$P_1 \mid P_2 \to Q_1 \mid Q_2 \tag{E.0.1}$$

and

$$\Gamma \vdash_\ell P_1 \mid P_2 \tag{E.0.2}$$

From (E.0.2) we have that

$$\Gamma \vdash_\ell P_1 \tag{E.0.3}$$

and

$$\Gamma \vdash_\ell P_2 \tag{E.0.4}$$

(E.0.1) is derived from

$$P_1 \xrightarrow{l^d!(a)} Q_1 \tag{E.0.5}$$

and

$$P_2 \xrightarrow{l^d?(a)} Q_2 \tag{E.0.6}$$

From Lemma D.4 and (E.0.3) and (E.0.5) we have

$$(\ell(d).l.(x)\Gamma')\bot\Gamma \tag{E.0.7}$$

and

$$\Gamma'\{x/a\} \subseteq ((\ell(d).l.(x)\Gamma')\bot\Gamma) \tag{E.0.8}$$

and

$$\Gamma \vdash_\ell Q_1 \tag{E.0.9}$$

From Lemma D.2 and (E.0.4) and (E.0.6) and (E.0.7) and (E.0.8) we have

$$\Gamma \vdash_\ell Q_2 \tag{E.0.10}$$

From (E.0.9) and (E.0.10) we have

$$\Gamma \vdash_\ell Q_1 \mid Q_2 \tag{E.0.11}$$

which completes the proof for this case.

($Case\ P_1 \overset{c\ l^{\downarrow}!(a)}{\longrightarrow} Q_1\ and\ P_2 \overset{c\ l^{\downarrow}?(a)}{\longrightarrow} Q_2$)

Analogous to the previous case, considering instead Lemmas D.3 and D.5.

($Case\ P_1 \overset{(\nu a)l^d!(a)}{\longrightarrow} Q_1\ and\ P_2 \overset{l^d?(a)}{\longrightarrow} Q_2$)

We have that

$$P_1 \mid P_2 \rightarrow (\nu a)(Q_1 \mid Q_2) \tag{E.0.12}$$

and

$$\Gamma \vdash_\ell P_1 \mid P_2 \tag{E.0.13}$$

From (E.0.13) we have that

$$\Gamma \vdash_\ell P_1 \tag{E.0.14}$$

and

$$\Gamma \vdash_\ell P_2 \tag{E.0.15}$$

(E.0.12) is derived from

$$P_1 \overset{(\nu a)l^d!(a)}{\longrightarrow} Q_1 \tag{E.0.16}$$

and

$$P_2 \overset{l^d?(a)}{\longrightarrow} Q_2 \tag{E.0.17}$$

From Lemma D.6 and (E.0.14) and (E.0.16) we have there is Γ' such that

$$\Gamma \cup \Gamma' \vdash_\ell Q_1 \tag{E.0.18}$$

and

$$(\Gamma \cup \Gamma') \setminus a \subseteq \Gamma \tag{E.0.19}$$

and

$$(\Gamma(\ell(d).l).(x)\Gamma'')\bot\Gamma \tag{E.0.20}$$

and

$$\Gamma''\{x/a\} \subseteq (\Gamma(\ell(d).l).(x)\Gamma'')\bot(\Gamma \cup \Gamma') \tag{E.0.21}$$

From Lemma D.1 and (E.0.15), considering that (E.0.18) gives us that $\Gamma \cup \Gamma'$ is a well founded order, we have

$$\Gamma \cup \Gamma' \vdash_\ell P_2 \tag{E.0.22}$$

From Lemma D.2 and (E.0.22) and (E.0.17) and (E.0.20) and (E.0.21) we have

$$\Gamma \cup \Gamma' \vdash_\ell Q_2 \tag{E.0.23}$$

From (E.0.18) and (E.0.23) we have

$$\Gamma \cup \Gamma' \vdash_\ell Q_1 \mid Q_2 \tag{E.0.24}$$

From (E.0.24) and (E.0.19) we conclude

$$\Gamma \vdash_\ell (\nu a)(Q_1 \mid Q_2) \tag{E.0.25}$$

which completes the proof for this case.

($Case\ P_1 \stackrel{(\nu a)c\ l^{\downarrow}!(a)}{\longrightarrow} Q_1\ and\ P_2 \stackrel{c\ l^{\downarrow}?(a)}{\longrightarrow} Q_2$)

Analogous.

($Case\ P_1 \stackrel{c\ \texttt{this}}{\longrightarrow} Q_1$)

We have that

$$c \blacktriangleleft [P_1] \to c \blacktriangleleft [Q_1] \tag{E.0.26}$$

and

$$\Gamma \vdash_\ell c \blacktriangleleft [P_1] \tag{E.0.27}$$

From (E.0.27) we have that

$$\Gamma \vdash_{(\ell(\downarrow),c)} P_1 \tag{E.0.28}$$

(E.0.26) is derived from

$$P_1 \stackrel{c\ \texttt{this}}{\longrightarrow} Q_1 \tag{E.0.29}$$

From Lemma D.8 and (E.0.28) and (E.0.29) we have

$$\Gamma \vdash_{(\ell(\downarrow),c)} Q_1 \tag{E.0.30}$$

From (E.0.30) we conclude

$$\Gamma \vdash_\ell c \blacktriangleleft [Q_1] \tag{E.0.31}$$

which completes the proof for this case.

(τ)

Proofs for the cases where a τ transition originates from within the scope of a restriction, or from one component of a parallel composition or from within a context piece follow directly from induction hypothesis. ∎

F Theorem 4.6 auxiliary results

Lemma F.1 *Let P be a process such that $P :: T$ and closed(T). Then for every $\mathcal{C}, Q$ such that $P = \mathcal{C}[Q]$, $Q \stackrel{\lambda}{\longrightarrow} Q'$, then either there are $\mathcal{C}_1, Q_i, \lambda_i$ such that $P = \mathcal{C}_1[Q_1]$ and*

$$Q_1 \stackrel{\lambda_1}{\longrightarrow} \ldots \stackrel{\lambda_k}{\longrightarrow} Q_k \stackrel{\overline{\lambda}}{\longrightarrow} Q'_k$$

or there is λ' such that $Q \stackrel{\lambda'}{\longrightarrow} Q'$, and there are $\mathcal{C}_1, Q_i, \lambda_i$ such that $P = \mathcal{C}_1[Q_1]$ and

$$Q_1 \stackrel{\lambda_1}{\longrightarrow} \ldots \stackrel{\lambda_k}{\longrightarrow} Q_k \stackrel{\overline{\lambda}}{\longrightarrow} Q'_k$$

Proof. Follows by induction on the type derivation. A type can only be closed when all outputs have matching inputs and all input summations have a matching output (at least one of the inputs in the summation has a matching output). It is however necessary to consider that the matching input might not be an immediate transition of the process, in the sense there might be a number of transitions leading to a process where the transition is observable.

(*Sketch*) From $closed(T)$ we conclude that

$$Q :: T' \tag{F.1.1}$$

and

$$closed(T') \tag{F.1.2}$$

since conversation types for restricted names are *closed* - rule (Res) of Figure 12. Considering Lemmas A.1 to A.6 we conclude that T' is decomposable in the branch or choice type corresponding to λ, hence

$$T \equiv T' \bowtie \oplus\{\ldots; M.B; \ldots\} \tag{F.1.3}$$

or

$$T \equiv T' \bowtie \&\{\ldots; M.B; \ldots\} \tag{F.1.4}$$

We also have, from the definition of closed process (Definition 3.13) that in T' there is a $\boldsymbol{\tau}$ for all message types either from persistent outputs defined on shared labels. When such a $\boldsymbol{\tau}$ is present in the type term, by definition of merge (Definition 3.11), we conclude that the $\boldsymbol{\tau}$ originates from the merge of the referred branch/choice type associated to λ and a dual (with respect to merge) counterpart.

$$T \equiv T'' \bowtie \oplus_{i\in I}\{\overline{M}_i.B_i; \ldots\} \bowtie \&_{i\in I}\{M_i.B'_i\} \tag{F.1.5}$$

If λ is an input defined on a plain label and derived from an input summation process, we have that the corresponding choice type may be merged with a choice type obtained by weakening (Subtype (16) Figure 10) in which case $\overline{\lambda}$ may not be ever a transition of the process. On the other hand there is a transition

$$Q \xrightarrow{\lambda'} Q'' \tag{F.1.6}$$

derived from the same input summation for which the process has a dual transition - corresponding to the output for which the branch type is initially introduced. If λ is an output defined on a plain label there must be a matching input, since otherwise there would be no associated $\boldsymbol{\tau}$ in the type.

If λ is defined on a shared label we have, by definition of merge (Definition 3.11) that a $\boldsymbol{\tau}$ is introduced in the merged type when the decomposition is in a type where the output type is persistent $\star$, and in the type where the matching input must occur. Since the process typed with the persistent output is a process able to handle infinitely many outputs, we are sure that any such output has a matching input.

Proof that a process eventually (after a number of transitions) has a transition corresponding to a message type present in the process' type follows by induction on the type derivation in lines similar to the proofs of Lemmas A.1 to A.6. ∎

Lemma F.2 *Let P be a well typed process. (1) If there is an event ordering Γ such that $\Gamma \vdash_\ell P$ then there is an event ordering Γ_1 which has a minimal event e_1 such that for every $\mathcal{C}, Q$ for $P = \mathcal{C}[Q]$ we have that Q is well ordered, hence $\Gamma' \vdash_{\ell'} Q$ for some Γ', ℓ', and $\Gamma' \subseteq (e_1 \bot \Gamma_1)$ and Γ_2 such that $\Gamma_2 \vdash_\ell P$, and $\Gamma_2 \subseteq \Gamma$ and $\Gamma_2 \subseteq (n.l \bot \Gamma_1)$.*

(2) Furthermore if P is not a finished process (Definition 4.5) and has no active **this** *then there are $\mathcal{C}, Q$ and Γ', ℓ' such that $\Gamma' \vdash_{\ell'} Q$ and $\ell'(d) = n$ and $Q \xrightarrow{\lambda} Q'$ for $\lambda = l^d!(c)$ or $\lambda = l^d?(c)$.*

Proof. Follows by induction on the structure of P. We prove (1), being the proof of (2) a direct extension.

(*Case* $l^d!(n).Q$)

We have that

$$\Gamma \vdash_\ell l^d!(n).Q \tag{F.2.1}$$

from which we directly have that

$$(\ell(d).l.(x)\Gamma' \bot \Gamma) \vdash_\ell l^d!(n).Q \tag{F.2.2}$$

thus completing the proof for this case.

(*Case* $\Sigma_{i \in I} l_i^d?(x_i).Q_i$)

We have that

$$\Gamma \vdash_\ell \Sigma_{i \in I} l_i^d?(x_i).Q_i \tag{F.2.3}$$

from which we directly have that

$$(\ell(d).l_i.(x)\Gamma_i' \bot \Gamma) \vdash_\ell \Sigma_{i \in I} l_i^d?(x_i).Q_i \tag{F.2.4}$$

thus completing the proof for this case.

(*Case* $Q_1 \mid Q_2$)

We have that

$$\Gamma \vdash_\ell Q_1 \mid Q_2 \tag{F.2.5}$$

from which we have

$$\Gamma \vdash_\ell Q_1 \tag{F.2.6}$$

and

$$\Gamma \vdash_\ell Q_2 \tag{F.2.7}$$

By induction hypothesis on (F.2.6) we have there is Γ_1, Γ_1', e_1 such that

$$\Gamma_1' \vdash_\ell Q_1 \tag{F.2.8}$$

and $\Gamma_1' \subseteq \Gamma$ and $\Gamma_1' \subseteq (e_1 \bot \Gamma_1)$. By induction hypothesis on (F.2.7) we have there is Γ_2, Γ_2', e_2 such that

$$\Gamma_2' \vdash_\ell Q_2 \tag{F.2.9}$$

and $\Gamma_2' \subseteq \Gamma$ and $\Gamma_2' \subseteq (e_2 \bot \Gamma_2)$.

From $\Gamma_1' \subseteq \Gamma$ and $\Gamma_2' \subseteq \Gamma$ we conclude $\Gamma_1' \cup \Gamma_2' \subseteq \Gamma$ and hence $\Gamma_1' \cup \Gamma_2'$ is well founded. Thus considering Lemma D.1 we have

$$\Gamma_1' \cup \Gamma_2' \vdash_\ell Q_1 \tag{F.2.10}$$

and

$$\Gamma_1' \cup \Gamma_2' \vdash_\ell Q_2 \tag{F.2.11}$$

from which we conclude

$$\Gamma_1' \cup \Gamma_2' \vdash_\ell Q_1 \mid Q_2 \tag{F.2.12}$$

Given that $(e_1 \bot \Gamma_1)$ and $(e_2 \bot \Gamma_2)$ are constructed minimally we conclude that $(e_1 \bot \Gamma_1) \cap (e_2 \bot \Gamma_2) \subseteq \Gamma$, which ensures $(e_1 \bot \Gamma_1) \cup (e_2 \bot \Gamma_2)$ is well founded and has a minimal element which is either e_1 or e_2. From $\Gamma_1' \subseteq (e_1 \bot \Gamma_1)$ and $\Gamma_2' \subseteq (e_2 \bot \Gamma_2)$ we have that

$$\Gamma_1' \cup \Gamma_2' \subseteq (e_1 \bot \Gamma_1) \cup (e_2 \bot \Gamma_2) \tag{F.2.13}$$

thus completing the proof for this case.

(*Case* $(\nu a)Q$)

We have that

$$\Gamma \setminus a \vdash_\ell (\nu a)Q \tag{F.2.14}$$

derived from

$$\Gamma \vdash_\ell Q \tag{F.2.15}$$

By induction hypothesis we have that there is Γ_1, Γ_2, e such that

$$\Gamma_2 \vdash_\ell Q \tag{F.2.16}$$

and $\Gamma_2 \subseteq \Gamma$ and $\Gamma_2 \subseteq (e \bot \Gamma_1)$. We then have

$$\Gamma_2 \setminus a \vdash_\ell (\nu a)Q \tag{F.2.17}$$

and $\Gamma_2 \setminus a \subseteq \Gamma \setminus a$ and $\Gamma_2 \setminus a \subseteq (e \bot \Gamma_1)$ thus completing the proof for this case.

(*Case* $\mathsf{this}(x).Q$)

We have that

$$\Gamma \vdash_\ell \mathsf{this}(x).Q \tag{F.2.18}$$

derived from

$$\Gamma \cup \Gamma' \vdash_\ell Q \tag{F.2.19}$$

where $\Gamma'\{x/\ell(\downarrow)\} \subseteq \Gamma$. By induction hypothesis we have that there is Γ_1, Γ_2, e such that

$$\Gamma_2 \vdash_\ell Q \tag{F.2.20}$$

and $\Gamma_2 \subseteq \Gamma \cup \Gamma'$ and $\Gamma_2 \subseteq (e \bot \Gamma_1)$. We then have $\Gamma_2\{x/\ell(\downarrow)\} \subseteq \Gamma$ and

$$\Gamma_2\{x/\ell(\downarrow)\} \vdash_\ell \mathsf{this}(x).Q \tag{F.2.21}$$

and $\Gamma_2\{x/\ell(\downarrow)\} \subseteq (e \bot \Gamma_1)\{x/\ell(\downarrow)\}$, which completes the proof for this case.

(*Remaining cases*)

For inaction $\mathbf{0}$ and recursion variable $\mathcal{X}$ rules (Stop) and (RecVar) direct give the result. Recursive process **rec** and conversation access $n \blacktriangleleft [\ldots]$ follow directly from induction hypothesis. ∎

G Proof of Theorem 4.6

Let P be a well typed process such that $P :: T$ and $\mathit{closed}(T)$ and Γ an event ordering and a, b names ($a, b \notin \mathit{fn}(P)$) such that $\Gamma \vdash_{(a,b)} P$. If P is not a finished process (Definition 4.5) then there is Q such that $P \to Q$.

Proof. (*Sketch*)

We have that

$$P :: T \tag{G.0.1}$$

and $\mathit{closed}(T)$ and

$$\Gamma \vdash_{(a,b)} P \tag{G.0.2}$$

for $a, b \notin \mathit{fn}(P)$. If P has an active **this** then, since they are required to occur inside context pieces, we have that $P \to P'$. Otherwise, since P is not a finished process, we have from Lemma F.2 that there is a minimum element in the overall ordering e such that there is $\mathcal{C}, Q$ such that $P = \mathcal{C}[Q]$ and

$$Q \xrightarrow{\lambda} Q' \tag{G.0.3}$$

for $\lambda = l^d!(c)$ or $\lambda = l^d?(c)$.

We have that there is λ' obtained from λ by changing direction, locating and/or adding (νa) such that either $P \xrightarrow{\lambda'} P'$ or λ' is located at a and there is $\mathcal{C}_1, P_1$ such that $P = \mathcal{C}_1[(\nu a)P_1]$ and $P_1 \xrightarrow{\lambda'} P_1'$.

Then considering $P :: T$ and $\mathit{closed}(T)$ and Lemma F.1 we have that there must be a matching transition in P to λ' (or to another transition observable from P_1' for which the proof is analogous). Furthermore such transition must be an immediate transition of the process, otherwise the associated event would not be the minimal event in the event ordering, hence there is $\mathcal{C}', R$ such that $P = \mathcal{C}'[R]$ and

$$R \xrightarrow{\overline{\lambda'}} R' \tag{G.0.4}$$

If $P \xrightarrow{\lambda'} P'$ then we have that $P \xrightarrow{\overline{\lambda'}} P''$, and if λ' is located at a and $P = \mathcal{C}_1[(\nu a)P_1]$ then $P_1 \xrightarrow{\overline{\lambda'}} P_1''$. We focus on the first case when the label carries no restricted name, being the proof for the other cases analogous.

We have that either

$$P = \mathcal{C}_1[P_1 \mid P_2] \tag{G.0.5}$$

and $P_1 \xrightarrow{\lambda'} P_1'$ and $P_2 \xrightarrow{\lambda'} P_2'$ or

$$P = \mathcal{C}_1[c \blacktriangleleft [\mathcal{C}_2[P_1 \mid P_2]]] \tag{G.0.6}$$

and $\mathcal{C}_2[P_1] \xrightarrow{\lambda_1} P_1'$ and $\mathcal{C}_2[P_2] \xrightarrow{\lambda_2} P_2'$ and such that $\lambda_i = c \cdot \lambda_j$ ($i, j \in \{1, 2\}$ and $i \neq j$) and either $\lambda' = \lambda_i$ or $\overline{\lambda'} = \lambda_i$.

For the case of (G.0.5) we directly have a reduction from a synchronization, while for the case of (G.0.6) we derive

$$\mathcal{C}_2[P_1 \mid P_2] \xrightarrow{c\ \mathtt{this}} P_1' \mid P_2' \tag{G.0.7}$$

from which we derive

$$c \blacktriangleleft [\mathcal{C}_2[P_1 \mid P_2]] \xrightarrow{\tau} c \blacktriangleleft \left[P_1' \mid P_2'\right] \tag{G.0.8}$$

thus completing the proof. ∎