A Theoretical Basis of Communication-Centred Concurrent Programming

Marco Carbone[1,2] Kohei Honda[1] Nobuko Yoshida[2]
Robin Milner[3] Gary Brown[4] Steve Ross-Talbot[4]

[1] Queen Mary, University of London, UK
[2] Imperial College, London, UK
[3] University of Cambridge, UK
[4] Pi4 Technologies Ltd

Abstract.

This document presents two different paradigms of description of communication behaviour, one focussing on global message flows and another on end-point behaviours, as formal calculi based on session types. The global calculus originates from Choreography Description Language, a web service description language developed by W3C WS-CDL working group. The end-point calculus is a typed π-calculus. The global calculus describes an interaction scenario from a vantage viewpoint; the endpoint calculus precisely identifies a local behaviour of each participant. After introducing the static and dynamic semantics of these two calculi, we explore a theory of endpoint projection which defines three principles for well-structured global description. The theory then defines a translation under the three principles which is sound and complete in the sense that all and only behaviours specified in the global description are realised as communications among end-point processes. Throughout the theory, underlying type structures play a fundamental role.

The document is divided in two parts: part I introduces the two descriptive frameworks using simple but non-trivial examples; the second part establishes a theory of the global and end-point formalisms.

Contents

Part 1

Introductory Examples

1. Introduction to Part 1

This paper introduces two different ways of describing communication-centred software in the form of formal calculi and discusses their relationship. Two different frameworks of description, one centring on global message flows and another centring on local (end-point) behaviours, share the common feature, *structured representation of communications*. The global calculus originates from Choreography Description Language (CDL) [**52**], a web service description language developed by W3C's WS-CDL Working Group. The local calculus is based on the π-calculus [**35**], one of the representative calculi for communicating processes. We show any well-formed description (in a technical sense we shall make clear) in the global calculus has a precise representation in the local calculus.

Both calculi are based on a common notion of structured communication, called *session*. A session binds a series of communications between two parties into one, distinguishing them from communications belonging to other sessions. This is a standard practice in business protocols (where an instance of a protocol should be distinguished from another instance of the same or other protocols) and in distributed programming (where two interacting parties use multiple TCP connections for performing a unit of conversation). As we shall explore in the present paper, the notion of session can be cleanly integrated with such notions as branching, recursion (loop) and exceptions. We show, through examples taken from simple but non-trivial business protocols, how concise structured description of non-trivial interactive behaviour is possible using sessions. From a practical viewpoint, a session gives us the following merits.

- It offers a clean way to describe a complex sequence of communications with rigorous operational semantics, allowing structured description of interactive behaviour.
- Session-based programs can use a simple, algorithmically efficient typing algorithm to check its conformance to expected interaction structures.
- Sessions offer a high-level abstraction for communication behaviour upon which further refined reasoning techniques, including type/transition/logic-based ones, can be built.

The presentation in this paper focusses the first point, and gives a formal basis for the second point. A full discussion of the second point and exploration of the third point are left to a later version of this paper and in its sequels.

An engineering background of the present work is the explosive growth of the Internet and world-wide web which has given rise to, in the shape of de facto standards, an omnipresent naming scheme (URI/URL), an omnipresent communication protocols (HTTP/TCP/IP) and an omnipresent data format (XML). These three elements arguably offer the key infra-structural bases for application-level distributed programming. This engineering background makes it feasible and advantageous to develop applications which will be engaged in complex sequences of interactions among two or more parties. Another background is maturing of theories of processes centring on the π-calculus and its types. The π-calculus and its theories of types are singular in that not only do they enable a study of diverse ways for structuring communication but also they allow fruitful and often surprising connections to existing formalisms including process algebras (e.g. CSP and CCS), functional computation (e.g. λ-calculus), logics (Linear Logic) and objects (e.g. Java). We believe a combination of strong practical needs for interactional computation and rich theoretical foundations will lead to rich dialogues between practice and theories. The present work is intended to offer some of the technical elements which may become useful in this dialogue.

This paper consists of two parts. In the first part, which are the first give sections including this Introduction, we informally introduce two paradigms of describing interactions through incrementally complex examples. These examples come from use-cases for CDL found in CDL primer [**45**] by Steve Ross-Talbot and Tony Fletcher, and those examples communicated by Gary Brown [**13**] and Nickolas Kavanztas [**29**]. In the second part, which form the remaining sections, we introduce formal semantics, type discipline, and the formal connection between the core parts of these two formalisms.

Structure of the paper. In the rest of this paper, Sections 2, 3 and 4 are devoted to informal illustration of key technical elements through description of small but non-trivial use-cases in the global and local calculi. The description starts from a simple example and reaches a fairly complex one, illustrating the essence of each construct as well as the relationship between their respective global descriptions and the corresponding local ones. Section 5 comments on the correspondence and differences between our formal calculi and CDL. The second part (from Section 6 to Section 18) formally introduces two calculi (the global calculus in Sections 8, 9 and 10; and the endpoint calculus in Sections 11 and 12), then develops the theory of end-point projections for these calculi (in Sections 13, 14, 15 and 16), and concludes the paper with related works and further topics (in Section 18). The appendix offers further technical details.

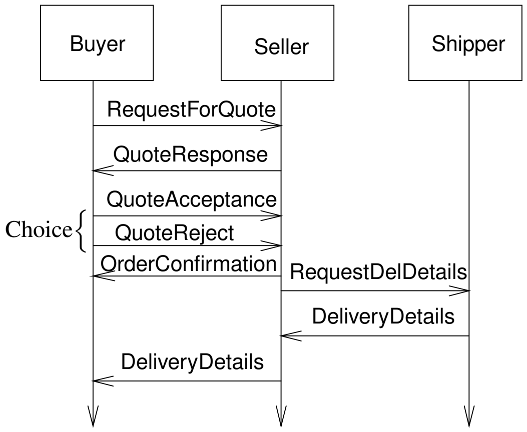

FIGURE 1. Graphical Representation of Simple Protocol

.

2. Describing Communication Behaviour (1)

2.1. A Simple Business Protocol. In this section and the next, we show how small, but increasingly complex, business protocols can be accurately and concisely described in two small programming languages, one based on global message flows and another based on local, or end-point, behaviours. Along the way we also illustrate each construct of these mini programming languages (whose formal semantics is discussed in the second part of the paper).

Our starting point is a simple business protocol for purchasing a good among a buyer, a seller and a shipper, which we call **Simple BSH Protocol**. Informally the expected interaction is described as follows.

(1) First, Buyer asks Seller, through a specified channel, to offer a quote (we assume the good to buy is fixed).
(2) Then Seller replies with a quote.
(3) Buyer then answers with either QuoteAcceptance or QuoteRejection. If the answer is QuoteAcceptance, then Seller sends a confirmation to Buyer, and sends a channel of Buyer to Shipper. Then Shipper sends the delivery details to Buyer, and the protocol terminates. If the answer is QuoteRejection, then the interaction terminates.

Figure 1 presents an UML sequence diagram of this protocol. Observe that, in Figure 1, many details are left unspecified: in real interaction, we need to specify, for example, the types of messages and the information exchanged in interaction, etc. While the protocol does not include practically important elements such as conditional and loops, its simplicity serves as a good starting point for introducing two formalisms.

2.2. Assumption on Underlying Communication Mechanisms. We first outline the basic assumptions common to both global and local formalisms. Below and henceforth we call the dramatis personae of a protocol (Buyer, Seller and Shipper in the present case), *participants*.

- We assume each participant either communicates through channels or change the content of variables local to it (two participants may have their own local variables with the same name but they are considered distinct).
- In communication:

(1) A sender participant sends a message and a receiver receives it, i.e. we only consider a point-to-point communication. A communication is always done through a *channel*. The message in a communication consists of an operator name and, when there is a value passing, a value. The value will be assigned to a local variable at the receiver's side upon the arrival of that message.
(2) Communication can be either an *in-session communication* which belongs to a session, or *session initiation channels* which establishes a session (which may be liked to establishing one or more fresh transport connections for a piece of conversation between two distributed peers). In a session initiation communication, one or more fresh session channels belonging to a session are declared, i.e. one session can use multiple channels.
(3) A channel can be either a *session channel* which belongs to a specific session or an *session-initiating channel* which is used for session-initiation. For a session-initiating channel, we assume its sender and a receiver is pre-determined.

- We may or we may not demand:
  (1) the order of messages from one participant to another through a specified channel is preserved.
  (2) each communication is synchronous, i.e. a sender immediately knows the arrival of a message at a receiver.
  (3) one party participating in a session can use a session-channel both for sending and receiving.

The last three assumptions which we leave undermined do affect a way to formalise protocols, as well as for understanding their formal properties. Nevertheless the existence or lack of these assumptions do not substantially affect the informal discussions in this and the next section.

2.3. Representing Communication (1): Initiating Session. Buyer's session-initiating communication in Simple BSH Protocol is described in the global calculus as follows.

(1) $$\mathsf{Buyer} \to \mathsf{Seller} : \mathsf{InitB2S}(\mathit{B2Sch})\,.\,I$$

which says:

> Buyer initiates a session with Seller by communication through a session-initiating channel INITB2S, declaring a fresh in-session channel *B2Sch*. Then interaction moves to *I*.

Note "." indicates sequencing, as in process calculi. A session initiation can specify more than one session channels as needed, as the following example shows.

(2) $$\mathsf{Buyer} \to \mathsf{Seller} : \mathsf{InitB2S}(\mathit{B2Sch},\ \mathit{S2Bch})\,.\,I$$

which declares two (fresh) session channels, one from Buyer to Seller and another in the reverse direction.

In local description, the behaviour is split into two, one for Buyer and another for Seller, using the familiar notation from process algebras. For example (1) becomes:

(3) $$\mathsf{Buyer}[\ \mathsf{InitB2S}\,(\mathit{B2Sch})\,.\,P_1\ ],\qquad \mathsf{Seller}[\ \overline{\mathsf{InitB2S}}\,(\mathit{B2Sch})\,.\,P_2\ ]$$

Above $\mathsf{Buyer}[\ P\ ]$ specifies a buyer's behaviour, while $\mathsf{Seller}[\ P\ ]$ specifies a seller's behaviour. The over-lined channel indicates it is used for output (this follows the tradition of CCS/π-calculus: in CSP, the same action is written $\mathsf{InitB2S}\,!\,(\mathtt{B2Sch})$.

Note the behaviour of each participant is described rather than their interaction. When these processes are combined, they engage in interaction as described in the scenario above.

2.4. Representing Communication (2): In-session Communication. An in-session communication specifies an operator and, as needed, a message content. First we present interaction without communication of values.

(4) $$\mathsf{Buyer} \to \mathsf{Seller} : \mathit{B2Sch}\,\langle \mathtt{QuoteRequest}\rangle\,.\,I'$$

where *B2Sch* is an in-session channel. It says:

> Buyer sends a *QuoteRequest*-message to Seller, then the interaction I' ensues.

The same behaviour can be written down in the local calculus as:

(5) $$\overline{B2Sch}\,\langle\texttt{QuoteRequest}\rangle\,.\,P_1, \qquad B2Sch\,\langle\texttt{QuoteRequest}\rangle\,.\,P_2$$

An in-session communication may involve value passing, as follows.

(6) $$\mathsf{Seller} \to \mathsf{Buyer} : S2Bch\,\langle\texttt{QuoteResponse}, 3{,}000, x\rangle\,.\,I'$$

which says:

> *Seller sends a* QuoteRespons-*message with value* $3{,}000$ *to Buyer; Buyer, upon reception, assigns the received value,* $3{,}000$*, to its local variable x.*

This description can be translated into end-point behaviours as follows.

(7) $$\overline{S2Bch}\,\langle\texttt{QuoteResponse}, 3{,}000\rangle\,.\,P_1, \qquad S2Bch\,\langle\texttt{QuoteResponse}, y\rangle\,.\,P_2$$

which describes precisely the same communication behaviour.

2.5. Representing Branching. In various high-level protocols, we often find the situation where a sender invokes one of the options offered by a receiver. A method invocation in object-oriented languages is a simplest such example. In a global calculus, we may write an in-session communication which involves such a branching behaviour as follows.

(8) $$\begin{array}{c}\{\mathsf{Buyer} \to \mathsf{Seller} : B2Sch\,\langle\texttt{QuoteAccept}\rangle\,.\,I_1\}\\ +\\ \{\mathsf{Buyer} \to \mathsf{Seller} : B2Sch\,\langle\texttt{QuoteReject}\rangle\,.\,I_2\}\end{array}$$

which reads:

> Through an in-session channel B2Sch, Buyer selects one of the two options offered by Seller, QuoteAccept and QuoteReject, and respectively proceeds to I_1 and I_2.

The same interaction can be written down in the local calculus as follows. First, Buyer's side (the one who selects) becomes:

(9) $$\begin{array}{c}\{\overline{B2Sch}\,\langle\texttt{QuoteAccept}\rangle\,.\,.P_1\}\\ \oplus\\ \{\overline{B2Sch}\,\langle\texttt{QuoteReject}\rangle\,.\,P_2\}\end{array}$$

Above $\oplus$ indicates this agent may either behave as $\overline{B2Sch}\,\langle\texttt{QuoteAccept}\rangle\,.\,.P_1$ or $\overline{B2Sch}\,\langle\texttt{QuoteReject}\rangle\,.\,P_2$, based on its own decision (this is so-called *internal sum*, whose nondeterminism comes from its internal behaviour).

In turn, Seller's side (which waits with two options) becomes:

(10) $$\begin{array}{c}\{B2Sch\,\langle\texttt{QuoteAccept}\rangle\,.\,Q_1\}\\ +\\ \{B2Sch\,\langle\texttt{QuoteReject}\rangle\,.\,Q_2\}\end{array}$$

Here $+$ indicates this agent may either behave as $B2Sch\,\langle\texttt{QuoteAccept}\rangle\,.\,Q_1$ or as $B2Sch\,\langle\texttt{QuoteReject}\rangle\,.\,Q_2$ depending on what the interacting party communicates through B2Sch (this is so-called *external sum*, whose nondeterminism comes from the behaviour of an external process). Note both branches start from input through the same channel B2Sch.

In the local descriptions, the original sum in the global description in (8) is decomposed into the internal choice and the external choice. Similarly, I_1 (resp. I_2) may be considered as the result of interactions between P_1 and Q_1 (resp. P_2 and Q_2).

$$\begin{array}{l}
\mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathsf{InitB2S}(\mathit{B2Sch})\,. \\
\mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathit{B2Sch}\,\langle \mathtt{QuoteRequest} \rangle\,. \\
\mathsf{Seller} \rightarrow \mathsf{Buyer} : \mathit{B2Sch}\,\langle \mathtt{QuoteResponse}, v_{\mathtt{quote}}, x_{\mathtt{quote}} \rangle\,. \\
\{\ \ \mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathit{B2Sch}\,\langle \mathtt{QuoteAccept} \rangle\,. \\
\quad \mathsf{Seller} \rightarrow \mathsf{Buyer} : \mathit{B2Sch}\,\langle \mathtt{OrderConfirmation} \rangle\,. \\
\quad \mathsf{Seller} \rightarrow \mathsf{Shipper} : \mathsf{InitS2H}(\mathit{S2Hch})\,. \\
\quad \mathsf{Seller} \rightarrow \mathsf{Shipper} : \mathit{S2Hch}\,\langle \mathtt{RequestDeliveryDetails} \rangle\,. \\
\quad \mathsf{Shipper} \rightarrow \mathsf{Seller} : \mathit{S2Hch}\,\langle \mathtt{DeliveryDetails}, v_{\mathtt{details}}, x_{\mathtt{details}} \rangle\,. \\
\quad \mathsf{Seller} \rightarrow \mathsf{Buyer} : \mathit{B2Sch}\,\langle \mathtt{DeliverDetails}, x_{\mathtt{details}}, y_{\mathtt{details}} \rangle\,.\,\mathbf{0}\ \ \} \\
\qquad\qquad\quad \boldsymbol{+} \\
\{\ \ \mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathit{B2Sch}\,\langle \mathtt{QuoteReject} \rangle\,.\,\mathbf{0}\ \ \}
\end{array}$$

FIGURE 2. Global Description of Simple Protocol

2.6. Global Description of Simple BSH Protocol. We can now present the whole of a global description of Simple BSH Protocol, in Figure 2. While its meaning should be clear from our foregoing illustration, we illustrate the key aspects of the description in the following.

- Buyer initiates a session by invoking Seller through the session-initiating channel INITB2S, declaring an in-session channel B2SCh. Next, Buyer sends another message to Seller with the operation name "QuoteRequest" and without carried values (this message may as well be combined with the first one in practice).
- Seller then sends (and Buyer receives) a reply "QuoteResponse" together with the quote value $v_{\mathtt{quote}}$. $v_{\mathtt{quote}}$ is a variable local to Seller (its exact content is irrelevant here). This received value will then be stored in $x_{\mathtt{quote}}$, local to Buyer.
- In the next step, Buyer decides whether the quote is acceptable or not. Accordingly:
  1. Buyer may send QuoteAccept-message to Seller. Then Seller confirms the purchase, and asks Shipper for details of a delivery; Shipper answers with the requested details (say a delivery date), which Buyer forwards to Seller. Upon reception of this message the protocol terminates (denoted by **0**, the inaction).
  2. Alternatively Buyer may send QuoteReject-message to Seller, in which case the protocol terminates without any further interactions.

Remark. The description could have used more than one channels: for example, the Buyer-Seller interactions can use *S2Bch* in addition for communication from Seller to Buyer. The use of only *B2Sch* may be considered as a way to describe "request-reply" mini-protocol inside a session, where an initial sender sends a request through a channel, and a receiver in turn replies leaving the involved channel implicit (which is a practice found in CDL, cf. [**45**]).

2.7. Local Description of Simple BSH Protocol. Figure 2 describes Simple BSH Protocol from a vantage viewpoint, having all participants and their interaction flows in one view. The same behaviour can be described focussing on behaviours of individual participants, as follows.

The description is now divided into (1) Buyer's interactive behaviour, (2) Seller's interactive behaviour, and (3) Shipper's interactive behaviour. We focus on Buyer's behaviour. One can intuitively see two descriptions of the same protocol, a global version in Figure 2 and a local version in Figure 3, represent the same software behaviours — we can extract the former from the latter and vice versa. We shall later establish such compatibility as a formal result. However there is a basic difference in the nature of descriptions: A global description allows us to see how messages are exchanged

$$
\begin{array}{ll}
\mathsf{Buyer}[& \overline{\mathsf{InitB2S}}\,(B2Sch)\,.\\
& \quad \overline{B2Sch}\,\langle \mathtt{QuoteRequest}\rangle\,.\\
& \quad B2Sch\,\langle \mathtt{QuoteResponse}, x_{\mathtt{quote}}\rangle\,.\\
& \quad \{\ \overline{B2Sch}\,\langle \mathtt{QuoteAccept}\rangle\,.\\
& \qquad B2Sch\,\langle \mathtt{OrderConfirmation}\rangle\,.\\
& \qquad B2Sch\,\langle \mathtt{DeliveryDetails}, y_{\mathtt{details}}\rangle\,.\,\mathbf{0}\ \}\\
& \qquad\qquad\qquad \oplus\\
& \quad \{\ \overline{B2Sch}\,\langle \mathtt{QuoteReject}\rangle\,.\,\mathbf{0}\ \}\quad]
\end{array}
$$

$$
\begin{array}{ll}
\mathsf{Seller}[& \mathsf{InitB2S}\,(B2Sch)\,.\\
& \quad B2Sch\,\langle \mathtt{QuoteRequest}\rangle\,.\\
& \quad \overline{B2Sch}\,\langle \mathtt{QuoteResponse}, v_{\mathtt{quote}}\rangle\,.\\
& \quad \{\ B2Sch\,\langle \mathtt{QuoteAccept}\rangle\,.\\
& \qquad \overline{B2Sch}\,\langle \mathtt{OrderConfirmation}\rangle\,.\\
& \qquad \overline{\mathsf{InitS2H}}\,(S2Hch)\,.\\
& \qquad\quad \overline{S2Hch}\,\langle \mathtt{DeliveryDetails}\rangle\,.\\
& \qquad\quad S2Hch\,\langle \mathtt{DeliveryDetails}, x_{\mathtt{details}}\rangle\,.\\
& \qquad \overline{B2Sch}\,\langle \mathtt{DeliveryDetails}, x_{\mathtt{details}}\rangle\,.\,\mathbf{0}\ \}\\
& \qquad\qquad\qquad +\\
& \quad \{\ B2Sch\,\langle \mathtt{QuoteReject}\rangle\,.\,\mathbf{0}\ \}\quad]
\end{array}
$$

$$
\begin{array}{ll}
\mathsf{Shipper}[& \mathsf{InitS2H}\,(S2Hch)\,.\\
& \quad S2Hch\,\langle \mathtt{DeliveryDetails}\rangle\,.\\
& \quad \overline{S2Hch}\,\langle \mathtt{DeliveryDetails}, v_{\mathtt{details}}\rangle\,.\,\mathbf{0}\quad]
\end{array}
$$

FIGURE 3. Local Description of Simple Protocol

between participants and how, as a whole, the interaction scenario proceeds; whereas, in the local description, the behaviour of each party is made explicit, as seen in distinct forms of choices used in Buyer and Seller.

```
$\textbf{if}\ x_{\texttt{quote}} \leq 1000\,@\texttt{Buyer}\ \textbf{then}$
$\{\ \ \mathsf{Buyer} \to \mathsf{Seller} : \mathit{B2Sch}\,\langle \texttt{QuoteAccept} \rangle\,.$
    $\mathsf{Seller} \to \mathsf{Buyer} : \mathit{B2Sch}\,\langle \texttt{OrderConfirmation} \rangle\,.$
    $\mathsf{Seller} \to \mathsf{Shipper} : \mathsf{InitS2H}(\mathit{S2Hch})\,.$
    $\mathsf{Seller} \to \mathsf{Shipper} : \mathit{S2Hch}\,\langle \texttt{RequestDeliveryDetails} \rangle\,.$
    $\mathsf{Shipper} \to \mathsf{Seller} : \mathit{S2Hch}\,\langle \texttt{DeliveryDetails}, v_{\texttt{details}}, x_{\texttt{details}} \rangle\,.$
    $\mathsf{Seller} \to \mathsf{Buyer} : \mathit{B2Sch}\,\langle \texttt{DeliverDetails}, x_{\texttt{details}}, y_{\texttt{details}} \rangle\,.\mathbf{0}\ \ \}$
 $\textbf{else}$
$\{\ \ \mathsf{Buyer} \to \mathsf{Seller} : \mathit{B2Sch}\,\langle \texttt{QuoteReject} \rangle\,.\mathbf{0}\ \ \}$
```

FIGURE 4. Global Description of Simple Protocol with Conditional

```
$\mathsf{Buyer}[$    $\overline{\mathsf{InitB2S}}\,(\mathit{B2Sch})\,.$
          $\overline{\mathit{B2Sch}}\,\langle \texttt{QuoteRequest} \rangle\,.$
          $\mathit{B2Sch}\,\langle \texttt{QuoteResponse}, x_{\texttt{quote}} \rangle\,.$
          $\textbf{if}\ x_{\texttt{quote}} \leq 1000\ \ \textbf{then}$
          $\{\ \ \overline{\mathit{B2Sch}}\,\langle \texttt{QuoteAccept} \rangle\,.$
              $\mathit{B2Sch}\,\langle \texttt{OrderConfirmation} \rangle\,.$
              $\mathit{B2Sch}\,\langle \texttt{DeliveryDetails}, y_{\texttt{details}} \rangle\,.\mathbf{0}\ \ \}$
           $\textbf{else}$
          $\{\ \ \overline{\mathit{B2Sch}}\,\langle \texttt{QuoteReject} \rangle\,.\mathbf{0}\ \ \}$    $]$
```

FIGURE 5. Local Description of Simple Protocol with Conditional (Buyer)

3. Describing Communication Behaviour (2)

3.1. Conditional. In Simple BSH Protocol, we only specified that Buyer may choose either `QuoteAccept` or `QuoteReject` nondeterministically. Suppose we wish to refine the description so that Buyer would choose the former when the quote is bigger than a certain amount, otherwise if else. For this purpose we can use a conditional.

The description now specifies the "reason" why each branch is taken. Notice the condition in the conditional branch, $x \leq 1000$, is explicitly *located*: the description says this judgement takes place at Buyer. The same scenario is described as follows using the end-point calculus. Other participants' behaviours remain the same.

3.2. Recursion. Assume we wish to further refine the protocol with the following specification:

> *If the quote is too high, Buyer asks another quote until it receives a satisfactory quote.*

Such behaviour is easily described using a loop or, more generally, recursion. In Figure 6, we show the global description of this enhanced protocol. There are only two additional lines: in the second line, **rec X**. indicates that, intuitively:

> We name the following block **X**. If **X** occurs inside that block, then we again recur to the top of the block.

In the last line, which is the second branch, **X** recurs again. Thus, at this point, the description recurs to a point immediately after **rec X** (i.e. the third line). The significance of recursion is its

$$
\begin{array}{l}
\mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathsf{InitB2S}(\mathit{B2Sch})\,.\\
\textbf{rec X}.\\
\{\ \ \mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathit{B2Sch}\,\langle \mathtt{QuoteRequest} \rangle\,.\\
\quad \mathsf{Seller} \rightarrow \mathsf{Buyer} : \mathit{B2Sch}\,\langle \mathtt{QuoteResponse},\, v_{\mathtt{quote}},\, x_{\mathtt{quote}} \rangle\,.\\
\quad \textbf{if } x_{\mathtt{quote}} \leq 1000\,@\mathtt{Buyer}\ \ \textbf{then}\\
\quad \{\ \ \mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathit{B2Sch}\,\langle \mathtt{QuoteAccept} \rangle\,.\\
\qquad \mathsf{Seller} \rightarrow \mathsf{Buyer} : \mathit{B2Sch}\,\langle \mathtt{OrderConfirmation} \rangle\,.\\
\qquad \mathsf{Seller} \rightarrow \mathsf{Shipper} : \mathsf{InitS2H}(\mathit{S2Hch})\,.\\
\qquad \mathsf{Seller} \rightarrow \mathsf{Shipper} : \mathit{S2Hch}\,\langle \mathtt{RequestDeliveryDetails} \rangle\,.\\
\qquad \mathsf{Shipper} \rightarrow \mathsf{Seller} : \mathit{S2Hch}\,\langle \mathtt{DeliveryDetails},\, v_{\mathtt{details}},\, x_{\mathtt{details}} \rangle\,.\\
\qquad \mathsf{Seller} \rightarrow \mathsf{Buyer} : \mathit{B2Sch}\,\langle \mathtt{DeliverDetails},\, x_{\mathtt{details}},\, y_{\mathtt{details}} \rangle\,.\,\mathbf{0}\ \ \}\\
\quad \textbf{else}\\
\quad \{\ \ \mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathit{B2Sch}\,\langle \mathtt{QuoteReject} \rangle\,.\,\mathbf{X}\ \ \}\quad \}
\end{array}
$$

FIGURE 6. Global Description of Simple Protocol with Conditional and Recursion

expressiveness (it can easily express various forms of loops) and its theoretical tractability. In the description, it is assumed that the value $v_{\mathtt{quote}}$ will be updated appropriately by Seller, which is omitted from the protocol description.

It is instructive to see how this recursion is translated into end-point behaviour. We present the local counterpart of Figure 6 in Figure 7 (we omit Shipper's behaviour which does not change). Observe both Buyer and Seller use recursion, so that they can collaboratively be engaged in recursive interactions. No change is needed in Shipper's local description, since it does not involve any recursion.

3.3. Timeout.

Let's consider refining Simple BSH protocol as follows:

> *If Buyer does not reply in 30 seconds after Seller presents a quote, then Seller will abort the transaction. Once Seller decides to do so, even if a confirmation message arrives from Buyer later, it is deemed invalid.*

For describing this refined behaviour, we first should have a means to describe a timeout. We consider this mechanism consisting of (1) creating a timer with a timeout value; (2) starting a timer; and (3) exception is thrown when a time out occurs. This exception is a *local exception*, in the sense that we consider our abstract notion of exceptions on the basis of the following infra-structural support:

> All exceptions are caught and handled within a participant locally (a participant may interact with other parties as a result).

This is the standard, low-cost mechanism employed in many run-times such as those of Java and C++.

Let us see how this can be realised in concrete syntax. We first refine the global description in Figure 2. Some comments:

- In the first line (initiating a session), two session channels, *B2Sch* (for default communications) and *S2Babort* (for aborting a transaction), are communicated through InitB2S. This generalised form of a session, where participants can use multiple channels in a single session, is useful for varied purposes.

$$
\begin{array}{ll}
\mathsf{Buyer}[& \overline{\mathsf{InitB2S}}\,(\mathit{B2Sch})\,. \\
 & \textbf{rec}\ \mathbf{X}. \\
 & \{\ \overline{\mathit{B2Sch}}\,\langle\texttt{QuoteRequest}\rangle\,. \\
 & \quad \mathit{B2Sch}\,\langle\texttt{QuoteResponse}, x_{\texttt{quote}}\rangle\,. \\
 & \quad \textbf{if}\ x_{\texttt{quote}} \leq 1000\ \ \textbf{then} \\
 & \quad \{\ \overline{\mathit{B2Sch}}\,\langle\texttt{QuoteAccept}\rangle\,. \\
 & \quad\quad \mathit{B2Sch}\,\langle\texttt{OrderConfirmation}\rangle\,. \\
 & \quad\quad \mathit{B2Sch}\,\langle\texttt{DeliveryDetails}, y_{\texttt{details}}\rangle\,.\mathbf{0}\ \ \} \\
 & \quad \textbf{else} \\
 & \quad \{\ \overline{\mathit{B2Sch}}\,\langle\texttt{QuoteReject}\rangle\,.\mathbf{X}\ \ \}\ \ \}\quad]
\end{array}
$$

$$
\begin{array}{ll}
\mathsf{Seller}[& \mathsf{InitB2S}\,(\mathit{B2Sch})\,. \\
 & \textbf{rec}\ \mathbf{X}. \\
 & \{\ \mathit{B2Sch}\,\langle\texttt{QuoteRequest}\rangle\,. \\
 & \quad \overline{\mathit{B2Sch}}\,\langle\texttt{QuoteResponse}, v_{\texttt{quote}}\rangle\,. \\
 & \quad \{\ \mathit{B2Sch}\,\langle\texttt{QuoteAccept}\rangle\,. \\
 & \quad\quad \overline{\mathit{B2Sch}}\,\langle\texttt{OrderConfirmation}\rangle\,. \\
 & \quad\quad \overline{\mathsf{InitS2H}}\,(\mathit{S2Hch})\,. \\
 & \quad\quad\quad \overline{\mathit{S2Hch}}\,\langle\texttt{DeliveryDetails}\rangle\,. \\
 & \quad\quad\quad \mathit{S2Hch}\,\langle\texttt{DeliveryDetails}, x_{\texttt{details}}\rangle\,. \\
 & \quad\quad \overline{\mathit{B2Sch}}\,\langle\texttt{DeliveryDetails}, x_{\texttt{details}}\rangle\,.\mathbf{0}\ \ \} \\
 & \qquad\qquad\qquad + \\
 & \quad \{\ \mathit{B2Sch}\,\langle\texttt{QuoteReject}\rangle\,.\mathbf{X}\ \ \}\quad]
\end{array}
$$

FIGURE 7. Local Description of Simple Protocol with Recursion (Buyer/Seller)

$$
\begin{array}{l}
\mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathsf{InitB2S}(\mathit{B2Sch, S2Babort})\,. \\
\quad \mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathit{B2Sch}\,\langle\texttt{QuoteRequest}\rangle\,. \\
\quad \mathsf{Seller} \rightarrow \mathsf{Buyer} : \mathit{B2Sch}\,\langle\texttt{QuoteResponse}, v_{\texttt{quote}}, x_{\texttt{quote}}\rangle\,. \\
\quad \textbf{let}\ t = \textbf{timer}(30)@\mathsf{Seller}\ \textbf{in}\ \ \{ \\
\quad\quad \{\ \mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathit{B2Sch}\,\langle\texttt{QuoteAccept}\rangle\ \textbf{timer}(t)\,. \\
\quad\quad\quad \mathsf{Seller} \rightarrow \mathsf{Buyer} : \mathit{B2Sch}\,\langle\texttt{OrderConfirmation}\rangle\,. \\
\quad\quad\quad \mathsf{Seller} \rightarrow \mathsf{Buyer} : \mathit{B2Sch}\,\langle\texttt{DeliverDetails}, x_{\texttt{details}}, y_{\texttt{details}}\rangle\,.\mathbf{0}\ \ \} \\
\qquad\qquad\quad \mathbf{+} \\
\quad\quad \{\ \mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathit{B2Sch}\,\langle\texttt{QuoteReject}\rangle\ \textbf{timer}(t)\,.\mathbf{0}\quad \} \\
\quad\quad \textbf{catch}\ (\textbf{timeout}(t)) \\
\quad\quad \{\ \mathsf{Seller} \rightarrow \mathsf{Buyer} : \mathit{S2Babort}\,\langle\texttt{Abort}\rangle\,.\mathbf{0}\ \ \}\ \ \}
\end{array}
$$

FIGURE 8. Global Description of Simple Protocol with Timeout

- In the fourth line, a timer t with timeout value 30 is initiated at Seller. This timer will be stopped if the input guard specifying that timer (Lines 6 and 10) receives a message (the two branches of a single choice have the same timer).

$$
\begin{array}{ll}
\mathsf{Seller}[& \mathsf{InitB2S}\,(\mathit{B2Sch},\ \mathit{S2Babort})\,.\\
 & \quad \mathit{B2Sch}\,\langle \mathtt{QuoteRequest}\rangle\,.\\
 & \quad \overline{\mathit{B2Sch}}\,\langle \mathtt{QuoteResponse},\, v_{\mathtt{quote}}\rangle\,.\\
 & \quad \mathbf{let}\; t = \mathbf{timer}(30)\;\mathbf{in} \quad \{\\
 & \qquad \{\ \mathit{B2Sch}\,\langle \mathtt{QuoteAccept}\rangle\; \mathbf{timer}(t)\,.\\
 & \qquad\quad \overline{\mathit{B2Sch}}\,\langle \mathtt{OrderConfirmation}\rangle\,.\\
 & \qquad\quad \overline{\mathit{B2Sch}}\,\langle \mathtt{DeliveryDetails},\, x_{\mathtt{details}}\rangle\,.\,\mathbf{0}\ \}\\
 & \qquad\qquad\qquad +\\
 & \qquad \{\ \mathit{B2Sch}\,\langle \mathtt{QuoteReject}\rangle\; \mathbf{timer}(t)\,.\,\mathbf{0}\ \}\\
 & \qquad \mathbf{catch}\ (\mathbf{timeout}(t))\\
 & \qquad \{\quad \overline{\mathit{S2Babort}}\,\langle \mathtt{Abort}\rangle \mathtt{abort}\,.\,\mathbf{0} \quad \} \quad \} \quad]\\
\mathsf{Buyer}[& \overline{\mathsf{InitB2S}}\,(\mathit{B2Sch},\ \mathit{S2Babort})\,.\\
 & \quad \{\\
 & \qquad \overline{\mathit{B2Sch}}\,\langle \mathtt{QuoteRequest}\rangle\,.\\
 & \qquad \mathit{B2Sch}\,\langle \mathtt{QuoteResponse},\, x_{\mathtt{quote}}\rangle\,.\\
 & \qquad\quad \{\ \mathit{B2Sch}\,\langle \mathtt{QuoteAccept}\rangle\,.\\
 & \qquad\qquad \overline{\mathit{B2Sch}}\,\langle \mathtt{OrderConfirmation}\rangle\,.\\
 & \qquad\qquad \overline{\mathit{B2Sch}}\,\langle \mathtt{DeliveryDetails},\, x_{\mathtt{details}}\rangle\,.\,\mathbf{0}\ \}\\
 & \qquad\qquad\qquad \oplus\\
 & \qquad\quad \{\ \overline{\mathit{B2Sch}}\,\langle \mathtt{QuoteReject}\rangle\,.\,\mathbf{0}\\
 & \quad \}\\
 & \quad \mathbf{par}\\
 & \quad \{\quad \mathit{S2Babort}\,\langle \mathtt{Abort}\rangle \mathtt{abort}\,.\,\mathbf{0} \quad \}\\
\quad] &
\end{array}
$$

FIGURE 9. Local Description of Simple Protocol with Timeout

- In the second line to the last, an exception handler is given, which says: when the timer fires, Seller will send an abort message to Buyer. It is omitted that, if Buyer's message arrives, Seller behaves as a sink, i.e. does nothing.

The same protocol can be described using the local formalism extended with timeout as follows. As before, in the exception branch, that Seller is assumed to behave as a sink to messages at B2Sch (i.e. $\mathit{B2Sch}\,\langle\mathtt{QuoteAccept}\rangle.\mathbf{0} + \mathit{B2Sch}\,\langle\mathtt{QuoteReject}\rangle.\mathbf{0}$ is omitted: it is possible it would behave non-trivially after it is in the abort mode). On the other hand, in Buyer's behaviour, we use **par** which indicates parallel composition. This behaviour is the same as before except the reception at the abort channel is added on parallel.

3.4. Combining Conditional, Recursion and Timeout. As a conclusion to this section, we present the combination of all constructs we have introduced so far. Figure Figure 10 gives a global description of the following behaviour:

(1) First, Buyer asks Seller, through a specified channel, to offer a quote (we assume the good to buy is fixed).
(2) Then Seller replies with a quote.
(3) Buyer then answers with either "I will buy" (if the price is cheap) or "I will not buy" (if not) to Seller. S
(4) If the answer is "I will buy", then Seller sends a confirmation to Buyer, and sends a channel of Buyer to Shipper. Then Shipper sends the delivery details to Buyer, and the protocol terminates.
(5) If the answer is "I will not buy", then the interaction recurs to (1) above.

```
Buyer → Seller : InitB2S($B2Sch$) .
rec X. {
  Buyer → Seller : $B2Sch$ ⟨QuoteRequest⟩ .
  Seller → Buyer : $B2Sch$ ⟨QuoteResponse, $v_{\texttt{quote}}$, $x_{\texttt{quote}}$⟩ .
  let $t$ = timer(30)@Seller in {
    if ($x_{\texttt{quote}} \leq 1000$ @Buyer) {
      Buyer → Seller : $B2Sch$ ⟨QuoteAccept⟩ timer($t$) .
      Seller → Buyer : $B2Sch$ ⟨OrderConfirmation⟩ .
      Seller → Shipper : InitS2H($S2Hch$) .
      Seller → Shipper : $S2Hch$ ⟨RequestDeliveryDetails⟩ .
      Shipper → Seller : $S2Hch$ ⟨DeliveryDetails, $v_{\texttt{details}}$, $x_{\texttt{details}}$⟩ .
      Seller → Buyer : $B2Sch$ ⟨DeliverDetails, $x_{\texttt{details}}$, $y_{\texttt{details}}$⟩ .
      0
    }   else {
      Buyer → Seller : $B2Sch$ ⟨QuoteReject⟩ timer($t$) . X    }
    catch (timeout($t$)) {
      Seller → Buyer : $S2Babort$ ⟨Abort⟩ . 0
    }
  }
```

FIGURE 10. Global Description of BSH Protocol with Conditional/Loop/Timeout

(6) If Buyer does not reply in time, Seller will abort the transaction.

The local description is given in Figure 11.

```
Buyer[  $\overline{\mathsf{InitB2S}}\,(B2Sch,\ S2Babort)$ .
        {
          rec X.
          { $\overline{B2Sch}\,\langle\mathtt{QuoteRequest}\rangle$ .
            $B2Sch\,\langle\mathtt{QuoteResponse}, x_{\mathtt{quote}}\rangle$ .
            if $x_{\mathtt{quote}} \leq 1000$ then
            { $\overline{B2Sch}\,\langle\mathtt{QuoteAccept}\rangle$ .
              $B2Sch\,\langle\mathtt{OrderConfirmation}\rangle$ .
              $B2Sch\,\langle\mathtt{DeliveryDetails}, y_{\mathtt{details}}\rangle\,.\,\mathbf{0}$ }
            else
            { $\overline{B2Sch}\,\langle\mathtt{QuoteReject}\rangle\,.\,\mathbf{X}$ } }
          par
          $S2Babort\,\langle\mathtt{ABORT}, x_{\mathtt{abort}}\rangle\,.\,\mathbf{0}$
        }
  ]

Seller[  $\mathsf{InitB2S}\,(B2Sch)$ .
         rec X.
         { $B2Sch\,\langle\mathtt{QuoteRequest}\rangle$ .
           $\overline{B2Sch}\,\langle\mathtt{QuoteResponse}, v_{\mathtt{quote}}\rangle$ .
           let $t = \mathbf{timer}(30)$ in    {
            { $B2Sch\,\langle\mathtt{QuoteAccept}\rangle\ \mathbf{timer}(t)$ .
              $\overline{B2Sch}\,\langle\mathtt{OrderConfirmation}\rangle$ .
              $\overline{\mathsf{InitS2H}}\,(S2Hch)$ .
                $\overline{S2Hch}\,\langle\mathtt{DeliveryDetails}\rangle$ .
                $S2Hch\,\langle\mathtt{DeliveryDetails}, x_{\mathtt{details}}\rangle$ .
              $\overline{B2Sch}\,\langle\mathtt{DeliveryDetails}, x_{\mathtt{details}}\rangle\,.\,\mathbf{0}$ }
                        +
            { $B2Sch\,\langle\mathtt{QuoteReject}\rangle\ \mathbf{timer}\,.\,\mathbf{X}$ }
           catch $(\mathbf{timeout}(t))$
           {   $\overline{S2Babort}\,\langle\mathtt{Abort}, \mathtt{abort}\rangle\,.\,\mathbf{0}$   }
         }
  ]

Shipper[  $\mathsf{InitS2H}\,(S2Hch)$ .
            $S2Hch\,\langle\mathtt{DeliveryDetails}\rangle$ .
            $\overline{S2Hch}\,\langle\mathtt{DeliveryDetails}, v_{\mathtt{details}}\rangle\,.\,\mathbf{0}$   ]
```

FIGURE 11. End-Point Description of BSH Protocol with Conditional/Loop/Timeout

4. Describing Communication Behaviour (3)

4.1. Criss-Crossing of Actions: Proactive Quoting (1). In this section we treat behaviours which involve *criss-crossing*: between two participants, say *A* and *B*, one message goes from *A* to *B* and another from *B* to *A* in parallel, one of which often having a stronger priority. We use use-cases contributed by Gary Brown [**13**] and Nickolas Kavanztas [**29**].

Brown's use-case is a (simplified form of) one of the typical interaction patterns in Investment Bank and other businesses. Its narrative description is extremely short, but the induced behaviour is non-trivial to describe. We assume two participants, *A* and *B*.

(1) Initially, *A* sends a request for quote to *B*.

```
$\mathsf{A} \to \mathsf{B} : \mathsf{InitA2B}(\mathit{A2Bch})\,.$
$\mathsf{A} \to \mathsf{B} : \mathit{A2Bch}\langle \mathtt{RequestQuote} \rangle\,.$
$\mathsf{B} \to \mathsf{A} : \mathit{A2Bch}\langle \mathtt{Quote}, y_{\mathtt{quote}}, x_{\mathtt{quote}} \rangle\,.$
$p_{\mathtt{quoteAccepted}} = \mathtt{ff}\ @\ B\,.$
$\{$
  $\tau_A\,.\ \mathsf{A} \to \mathsf{B} : \mathit{A2Bch}\langle \mathtt{AcceptQuote} \rangle\,.\ p_{\mathtt{quoteAcceppted}} := \mathtt{tt}\ @\ B\,.\ \mathbf{0}$
  $\mathbf{par}$
  $\mathbf{rec}\ X\,.\,\{$
    $\mathbf{let}\ t = \mathbf{timer}(5)@B\ \mathbf{in}$
    $\mathbf{when}\ (\mathbf{expired}(t)@B)$
      $\mathbf{if}(p_{\mathtt{quoteAccepted}} = \mathtt{ff}@B)\ \{\ \mathsf{B} \to \mathsf{A} : \mathit{A2Bch}\langle \mathtt{RefreshQuote}, y_{\mathtt{quote}}, x_{\mathtt{quote}} \rangle\,.\,\mathbf{X}\ \}$
$\}$
```

FIGURE 12. A Proactive Quoting with a Criss-Cross (global)

(2) Then *B* sends an initial quote to *A* as a response.

(3) Then *B* will enter a loop, sending pro-actively a new quote in a "RefreshQuote"-message every 5 seconds until *A*'s "AcceptQuote"-message arrives at *B*.

Thus the "AcceptQuote"-message from *A* is in a race condition with a "RefreshQuote"-message from *B*. Once the quote is accepted, *B* should terminate its loop. We leave unspecified in the use-case how a quote is calculated, how *A* decides to accept a quote, and how *A* notifies which quote *A* is agreeing on (refinements are easy).

The repeated actions at each time interval can be cleanly modelled using the predicate-based invocation mechanism [**52**], which is also useful for other purposes.

$$\mathbf{when}\ (p@A)\ \{I\}$$

where p is a predicate (an expression of a boolean type). It reads:

> The interaction I does not start until the predicate p becomes true: when it becomes so, then I will be engaged in.

Its precise semantics is either (1) whenever p becomes true, I should start; or (2) when p becomes true, I can start, but this "event" can be missed in which case I may not start. The behaviour in (1) tends to become more deterministic, while (2) is realisable through busy-waiting without additional synchronisation mechanism.

We use this construct to describe the use-case. We first informally illustrate the underlying idea (suggested by [**13**]): after the initial quote has arrived at *A*, we consider there are two independent threads of interactions, in both *A* and *B*.

- In one, *A* may decide to send the "AcceptQuote"-message; when *B* receives it, *B* will set its local variable $p_{\mathtt{quoteAccepted}}$ to "truth" (which should be initially "false").
- In another, *A* is always ready to receive "RefreshQuote"-message (with a new quote value); On the other hand, *as far as the local variable* $p_{\mathtt{quoteAccepted}}$ *is false*, *B* will repeatedly send, at each 5 seconds, a fresh quote.

Note the variable $p_{\mathtt{quoteAccepted}}$ is used for communication between two threads in *B*. When *B* ceases to send new quotes, *A* also cease to react to new quotes from *B*, thus both reaching a quiescent state. The description in the global formalism (augmented with "when"-construct) follows.

$$\begin{array}{ll}
\mathsf{A}[& \overline{\mathsf{InitA2B}}\,(A2Bch,\ B2Ach)\,. \\
 & \overline{A2Bch}\,\langle \texttt{RequestQuote}\rangle\,. \\
 & B2Ach\,\langle \texttt{Quote}, x_{quote}\rangle\,. \\
 & \{\ \tau.\overline{A2Bch}\,\langle \texttt{AcceptQuote}, x_{quote}\rangle\,.\mathbf{0} \quad \mathbf{par} \quad \mathbf{rec}\ \mathbf{X}.\{\ B2Ach\,\langle \texttt{RefreshQuote}, x_{quote}\rangle\,.\mathbf{X}\ \}\ \} \\
]
\end{array}$$

$$\begin{array}{ll}
\mathsf{B}[& \mathsf{InitA2B}\,(A2Bch,\ B2Ach)\,. \\
 & A2Bch\,\langle \texttt{RequestQuote}\rangle\,. \\
 & \overline{B2Ach}\,\langle \texttt{RefreshQuote}, y_{\texttt{quote}}\rangle\,. \\
 & p_{\texttt{quoteAcceppted}} := \texttt{ff}. \\
 & \{ \\
 & \quad A2Bch\,\langle \texttt{AcceptQuote}\rangle\,.\ p_{\texttt{quoteAcceppted}} := \texttt{tt}\ \ .\ \mathbf{0} \\
 & \quad \mathbf{par} \\
 & \quad \mathbf{rec}\ \ X.\,\{ \\
 & \quad \mathbf{let}\,t = \mathbf{timer}(5)\ \ \mathbf{in\ when}\ \ (\mathbf{expired}(t)) \\
 & \qquad \{\ \ \mathbf{if}\ \ (p_{\texttt{quoteAccepted}} = \texttt{ff})\ \ \{\ \overline{B2Ach}\,\langle \texttt{RefreshQuote}, y_{quote}\rangle\,.\mathbf{X}\ \}\ \ \} \\
 & \} \\
]
\end{array}$$

FIGURE 13. A Proactive Quoting with a Criss-Cross (local)

Above, "τ_A" is the standard τ-action local to A, indicating passage of an unspecified duration of time. Thus as a whole

$$\tau_A\,.\ \mathsf{A} \to \mathsf{B} : A2Bch\,\langle \texttt{AcceptQuote}\rangle\,.\ p_{\texttt{quoteAcceppted}} := \texttt{tt}\ @B\,.\,\mathbf{0},$$

indicates that the sending of "AcceptQuote" (with a quote value at the time) may take place after some duration of time, and when B receives this message, B will assign "truth" to its local variable $p_{\texttt{quoteAcceppted}}$. One may as well refine the above part as follows, using the "when" construct.

$$\begin{array}{l}
\mathbf{when}\,(\texttt{satisfied})@A) \\
\quad \{\ \mathsf{A} \to \mathsf{B} : A2Bch\,\langle \texttt{AcceptQuote}\rangle\,.\ p_{\texttt{quoteAcceppted}} := \texttt{tt}\ @B\,.\,\mathbf{0}\ \}
\end{array}$$

where `satisfied` is an unspecified predicate local to A, indicating the satisfaction of A w.r.t., say, the current quote value.

In the second thread, B is engaged in a loop: the timer t expires at each 5 seconds and, when $\mathsf{expires}(t)$ (which is a predicate rather than exception) becomes true, the body of "when" is executed. If $p_{\texttt{quoteAcceppted}}$ is false, it sends a quote and re-enters the loop: if $p_{\texttt{quoteAcceppted}}$ is true, it terminates the loop. The interaction

$$\mathsf{B} \to \mathsf{A} : A2Bch\,\langle \texttt{RefreshQuote}, y_{\texttt{quote}}, x_{\texttt{quote}}\rangle$$

not only indicates B sends a "RefreshQuote"-message, but also A is ready to receive it and sets the communicated quote into its variable $x_{\texttt{quote}}$.

The protocol description invites us to diverse forms of refinement. For example, we may consider the predicate `satisfied` is a boolean variable set after A receives a new quote (in the second thread). We leave exploration of such refinements to the reader.

Next we consider the local version of Figure 12, using the end-point counterpart of the "when"-construct. This is given in Figure 13. One may compare the presented behaviours with those in Figure 12. The "when" construct is used in B, with the same semantics as in the global calculus.

In the local description of A's behaviour, the projection makes clear that, in one of its two threads, A repeatedly gets ready to receive "RefreshQuote"-messages from B, while, independently, may move to the stage where it sends an "AcceptQuote"-message to B. Thus, when a criss-cross of these messages take place, A will simply receives the message from B while sending its own

$$\begin{array}{l}
\mathsf{A} \to \mathsf{B} : \mathsf{InitA2B}(\mathit{A2Bch,\ B2Ach})\,. \\
\mathsf{A} \to \mathsf{B} : \mathit{A2Bch}\,\langle \mathtt{RequestQuote}\rangle\,. \\
\mathsf{B} \to \mathsf{A} : \mathit{B2Ach}\,\langle \mathtt{Quote}, \mathtt{quote}, x_{\mathtt{Quote}}\rangle\,. \\
\mathbf{rec}\ \ X.\{ \\
\quad \mathbf{let}\, t = \mathbf{timer}(5)@B\ \ \mathbf{in} \\
\quad\quad \mathsf{A} \to \mathsf{B} : \mathit{A2Bch}\,\langle \mathtt{AcceptQuote}\rangle \mathbf{timer}(t)\,.\,\mathbf{0} \\
\quad \mathbf{catch}(\mathit{timeout}(t)) \\
\quad\quad \mathsf{B} \to \mathsf{A} : \mathit{B2Ach}\,\langle \mathtt{RefreshQuote}, \mathit{newQuote}, x_{\mathtt{Quote}}\rangle\,.\,\mathbf{X}\ \ \}
\end{array}$$

FIGURE 14. A Proactive Quoting with a Criss-Cross (global, with atomic interaction)

message. As noted before, we may as well refine A's behaviour, for example in its transition to the quote acceptance state.

In the local description of B, the first thread does not start from the τ-action (which is A's local action) but starts from the reception of "QuoteAcceptance"-message from A. The second thread is engaged with the timeout and loop using the "when" construct, using the variable $p_{\mathtt{quoteAccepted}}$.

The local descriptions of the proactive quoting protocol in Figure 13 are directly related with its global description in Figure 12 and vice versa, up to the treatment of criss-crossing. In particular, it is not hard to imagine how we can project the description in Figure 12 to the one in Figure 13 following a simple principle. A natural question is whether we can do the *reverse* translation in a general way: can we integrate the local descriptions in Figure 13 to synthesize the global description in Figure 12? What would be the general principle involved in such projection? Part of this question will be answered in Part II of the present paper.

4.2. Criss-Crossing of Actions: Proactive Quoting (2). In this subsection, we present an alternative global description of the proactive quoting protocol. It is simple and understandable, even though the description is only sound under a strong assumption about the underlying communication on mechanism. The description follows.

The description in Figure 12 is terse and understandable. However its clarity has become possible only by assuming a significant condition on the underlying messaging semantics: each interaction is atomic. This assumption becomes essential in $\mathsf{A} \to \mathsf{B} : \mathit{A2Bch}\,\langle \mathtt{AcceptQuote}\rangle$, which needs be executed atomically: if not, it is possible that A sends a $\mathsf{AcceptQuote}$-message to B, but the time-out in B is caught, B sends $\mathsf{RefreshQuote}$ to A, and A should again sends $\mathsf{AcceptQuote}$-message again, which is not the expected behaviour of A. Rather it says that $\mathsf{A} \to \mathsf{B} : \mathit{A2Bch}\,\langle \mathtt{AcceptQuote}\rangle$ either happens or not at all and moves to a timer, which is only realisable if this action is atomic. It may be costly to realise such atomicity in general. At the same time, the description may suggest atomicity of interaction can lead to terse specification of a complex behaviour.

Due to the assumption on atomicity and its interplay with timer, it is hard to devise local descriptions directly corresponding to Figure 14. Even if we stipulate the same atomicity assumption in local descriptions, it is hard to construct the projection onto A: the problem is that the 'when' loop within A does not have an activity that it can observe to indicate that B has exited the loop. A possible approach to this would be to model a guard condition for A to also include the 'quoteAccepted' variable — but this guard condition would also have to include the aspect of duration, otherwise (as a result of the 'when' blocking semantics) the guard at A would simply block until the variable 'quoteAccepted' was set to true, and it would not receive any of the quote refresh messages. Further, if both participants are required to use the same guard condition, then it also assumes they have synchronised clocks and evaluate the expressions at exactly the same time.

$$
\begin{array}{l}
\mathsf{RBC} \to \mathsf{STC} : \mathsf{InitR2S}(\mathit{R2Sch})\,. \\
\mathsf{RBC} \to \mathsf{STC} : \mathit{R2Sch}\,\langle \mathtt{CreateOrder} \rangle\,. \\
\mathsf{STC} \to \mathsf{RBC} : \mathit{R2Sch}\,\langle \mathtt{OrderAck} \rangle\,. \\
\mathbf{let}\ t = \mathbf{timer}(T)@\mathsf{RBC}\ \mathbf{in} \\
\quad \{\mathsf{STC} \to \mathsf{RBC} : \mathit{R2Sch}\,\langle \mathtt{POCompleted} \rangle\ \mathbf{timer}(t)\,.\,\mathbf{0}\} \\
\mathbf{catch\ timeout}(t)\ \{ \\
\quad \mathsf{RBC} \to \mathsf{STC} : \mathit{S2Rabort}\,\langle \mathtt{Abort} \rangle\,.\,\{ \\
\qquad \mathsf{STC} \to \mathsf{RBC} : \mathit{R2Sabort}\,\langle \mathtt{ConfirmAbort} \rangle\,.\,\mathbf{0} \\
\qquad + \\
\qquad \mathsf{STC} \to \mathsf{RBC} : \mathit{R2Sch}\,\langle \mathtt{POConfirmation} \rangle\,.\,\mathbf{0} \quad \}
\end{array}
$$

FIGURE 15. A Global Description of T-Shirts Procurement

4.3. Criss-Crossing of Actions: A T-Shirts Procurement Protocol (1). Next we treat Kavanztas's use-case [**29**], which describes a protocol for purchase orders between a really big corporation (RBC) and a small T-shirts company (STC).

(1) RBC sends a purchase order (PO) to STC.
(2) STC acknowledges the PO and initiates a business process to handle the PO.
(3) After STC's internal processes regarding the PO are completed, STC sends "PO-Completed" to RBC in order for RBC to complete its own business process.
(4) RBC can send a Cancel Order message to abort STC's business process (which can criss-cross with a PO completed message), any time before RBC receives the PO Completed message from STC
(5) If Cancel Order arrives at STC before PO Completed is sent from STC, then STC aborts its business process and acknowledges this to RBC with PO Cancelled, in order for RBC to abort its own business process. Otherwise, if STC has already sent PO Completed, it ignores the Cancel Order because RBC has agreed it will honor POs when cancellations are not sent out within an agreed-upon time-frame.
(6) If RBC has already sent the Cancel Order message and then it receives the PO Completed message, then instead of aborting, RBC completes it.

Figure 15 presents a global description of this protocol.
Above, RBC first initialises a session channel R2Sch through InitR2S, then sends an order, which STC acknowledges. RBC then starts a timer, i.e. the longest time T it is willing to wait before the PO confirmation arrives. The timer is frozen upon the PO confirmation. Alternatively if the time-out occurs, it is handled by the catch part: RBC sends an abort message to STC, and either STC acknowledges it or its PO-confirmation arrives. Note we have made a timer explicit in this description: we later show a description which does not rely on the use of a timer.

An acute reader may observe that this description again assumes atomicity of communication, as in the previous subsection, in the sense that: the execution of an interaction $\mathsf{A} \to \mathsf{B} : ch\,\langle \mathtt{Op} \rangle$ means the two things at the same time: A sends a message and B has received that message.

Next we give an end-point counterpart of the same description, in Figure 16.
In STC's description, we use the following predicate-based exception mechanism. The syntax for this exception handling is:

$$\mathbf{try}\ \{P\}\ \ \mathbf{catch}\ (p)\ \{Q\}$$

whose semantics is, informally: to execute the interaction P unless the predicate (a boolean-valued expression) p is satisfied (note p is treated as an event). In the latter case, Q would be executed. This

```
$\mathsf{RBC}[$   $\overline{\mathsf{InitR2S}}\,(R2Sch)\,.$
        $\overline{R2Sch}\,\langle\mathtt{CreateOrder}\rangle\,.$
        $S2Rch\,\langle\mathtt{OrderAck}\rangle\,.$
        $\mathbf{let}\ t = \mathbf{timer}(T)\ \mathbf{in}\quad\{$
          $S2Rch\,\langle\mathtt{POCompleted}\rangle\mathbf{timer}(t)\,.\,\mathbf{0}$
        $\}$
        $\mathbf{catch\ timeout}(t)\quad\{$
          $\overline{S2Babort}\,\langle\mathtt{Abort},\mathbf{true}\rangle\,.$
            $S2Babort\,\langle\mathtt{ConfirmAbort}\rangle\,.\,\mathbf{0}$
            $+$
            $S2Rch\,\langle\mathtt{POCompleted}\rangle\,.\,\mathbf{0}\}\quad]$

$\mathsf{STC}[$   $\mathsf{InitR2S}\,(R2Sch)\,.$
        $R2Sch\,\langle\mathtt{CreateOrder}\rangle\,.$
        $x_{\mathtt{Abort}} := \mathbf{false}\,.$
        $\overline{S2Rch}\,\langle\mathtt{OrderAck}\rangle\,.$
        $\mathbf{try}$
        $\{\quad \tau\,.\,\overline{S2Rch}\,\langle\mathtt{POCompleted}\rangle\,.\,\mathbf{0}\}$
        $\mathbf{catch}\ (\neg x_{\mathtt{Abort}})$
          $\{\overline{S2Babort}\,\langle\mathtt{ConfirmAbort}\rangle\,.\,\mathbf{0}$
          $+$
          $\overline{S2Rch}\,\langle\mathtt{POCompleted}\rangle\,.\,\mathbf{0}\}$
        $\mathbf{par}$
        $S2Babort\,\langle\mathtt{Abort},x_{\mathtt{abort}}\rangle\,.\,\mathbf{0}\quad]$
```

FIGURE 16. A Local Description of T-Shirts Procurement

construct is feasibly implemented if the "catch" part is an exception such as timeout or explicitly thrown exceptions. However its implementation becomes more involved if, as here, a predicate is used for invocation since in that case a mechanism is necessary to watch the update of relevant variables. Note this construct is similar to the "when" construct: the same underlying mechanism can realise both. As an alternative, one may realise a similar behaviour using either a busy-waiting or a "sleep" construct, though these alternatives may not be faithful to intended semantics when we use arbitrary predicates for invocation.

We illustrate the behaviour of RBC and STC in this end-point description. First, RBC's local behaviour is as follows.

- The first three actions (session init, order request and acknowledgement) are obviously implemented;
- RBC sets a timer and waits for T time-units to receive the PO confirmation from STC;
- If the time-out is triggered, RBC will send an abort to STC, and then wait for the abort confirmation or for the PO confirmation.

The local behaviour of STC may be illustrated thus.

- As in the RBC part, the first three actions need no description, apart the fact that STC has a variable for checking whether RBC has requested an abort or not. This variable is initialised to false;
- At this point STC checks the abort variable, and if it is not true it decides to perform a tau action and then send the PO confirmation.
- if the abort variable is true it then confirms the abort;
- in parallel with the described thread, there is another thread which just waits for an abort message from RBC.

```
$\mathsf{RBC} \to \mathsf{STC} : \mathsf{InitR2S}(\mathit{R2Sch})\,.$
$\mathsf{RBC} \to \mathsf{STC} : \mathit{R2Sch}\,\langle \mathtt{CreateOrder} \rangle\,.$
$\mathsf{STC} \to \mathsf{RBC} : \mathit{R2Sch}\,\langle \mathtt{OrderAck} \rangle\,.$
{
  $x_{\mathtt{AbortRequested}}@\mathit{STC} := \mathbf{false}\,.$
  $\tau_{\mathit{STC}}\,.$
  $\mathbf{if}\ \neg x_{\mathtt{AbortRequested}}@\mathit{STC}$ {
    $\mathsf{STC} \to \mathsf{RBC} : \mathit{R2Sch}\,\langle \mathtt{POConfirmation} \rangle\,.$
    $x_{\mathtt{ConfArrived}}@\mathit{RBC} := \mathbf{true}\,.\,\mathbf{0}$}
  else
    $\mathsf{STC} \to \mathsf{RBC} : \mathit{R2Sabort}\,\langle \mathtt{ConfirmAbort} \rangle\,.\,\mathbf{0}$
}
par
{
  $x_{\mathtt{ConfArrived}}@\mathit{RBC} := \mathbf{false}\,.$
  $\tau_{\mathit{RBC}}\,.$
  $\mathbf{if}\ \neg x_{\mathtt{ConfArrived}}@\mathit{RBC}$ {
    $\mathsf{RBC} \to \mathsf{STC} : \mathit{S2Rabort}\,\langle \mathtt{Abort} \rangle\,.$
    $x_{\mathtt{AbortRequested}} := \mathbf{true}\,.\,\mathbf{0}$    }
}
```

FIGURE 17. A Global Description of T-Shirts Procurement without Timer

Note the end-point description makes it explicit how timeout is done and how criss-crossing occurs in terms of two distributed end-point behaviours. We believe it faithfully realises the global behaviour described in Figure 15 under the assumption of atomicity of interactions: at the same time, one may observe that the given end-point description does *not* automatically get extracted from the global description. In fact, as far as the initial protocol description goes, the local description arguably realises a correct behaviour even if we do not stipulate the atomicity assumption for communication actions (it is notable that CDL [**52**] does not stipulate such atomicity).

4.4. Criss-Crossing of Actions: A T-Shirts Procurement Protocol (2). The descriptions so far depend on the explicit use of timer and exception (timeout) which a timer engenders. However the nondeterminism and criss-crossing of message exchanges themselves may not be directly related with local use of timers. Indeed, a description of the overall exchange of interactions is possible without using timers, as we shall discuss below.

The protocol uses two (local) variables, AbortRequested at STC and ConfArrived at RBC, both initialised to be false. The timing of update of these variables is the key underlying idea of this protocol. The protocol description follows.
Let us offer an informal illustration of the protocol.

- The initial three interactions remain the same as before, i.e. sending a purchase order from RBC to STC after a session initiation, then an acknowledgement from STC to RBC.

- At this stage the interactions are divided into the parallel composition of two behaviours. In one thread of interaction, we have:
  (1) STC will, at some point, check AbortRequested is true (i.e. RBC's abort request has arrived) or false (i.e. RBC's abort request has not arrived).
  (2) If AbortRequested is false, then STC will send a PO confirmation message. When RBC receives it, it will set its ConfArrived to be true, and STC moves to the completion of PO processing.
  (3) If AbortRequested is true, then STC will send a AbortConfirmed message. RBC receives it, and in both sites the PO process aborts.

  In another thread of interaction, we have:
  (1) At some point RBC will check ConfArrived.
  (2) If it is false (i.e. a PO confirmation has not arrived), then sends AbortRequest-message to STC.
  (3) If it is true (i.e. a PO confirmation has arrived), then RBC moves to the completion of PO processing.

In Figure 17, τ_{STC} (resp. τ_{RBC}) indicates a τ-action in STC (resp. in RBC), which may take an unspecified amount of time. We can check that this protocol never moves to:

- The situation where STC sends a PO confirmation but an RBC aborts (since, for an RBC to abort, it needs to obtain AbortConfirm message from STC).
- The situation where RBC receives both a PO-confirmation and AbortConfirm (for the same reason).

Note however it is possible STC may receive, in one thread, AbortRequest message at time t but, for some reason, this has not been propagated to anther thread in time, so that, at time $t+t_0$, STC sends a PO-confirmation message to RBC. However this does not contradicts the initial specification (we also believe this is consistent with the standard business convention).

The end-point projection of this example is not to hard, which we leave to the reader. We also note Kavanztas [**29**] presents a different description in CDL using the "when" construct with distributed predicates.

4.5. Further Note. In this section we have explored various ways to describe two business protocols (though the presented ones are far from the only ways to describe them). The purpose of these formal representations of business protocols in the calculi is not only to analyse the behaviours of these protocols themselves and to reason about them, but also to understand the correspondence between various constructs and their expressiveness. By having a precise operational semantics, we can discuss diverse aspects of the constructs needed to represent a large class of communication behaviours with precision. Further analyses of these and other complex business protocols in these formalisms would be an important and stimulating future research topic.

feature	CDL	formalism
session channels	located at input	no restriction
session initiation	implicit	explicit
general co-relation	yes	by adding “polyadic sync”
typing	by-name (informal)	by-structure (formal)
type checking	no	yes
local exception	none	yes
repetition	loop	recursion
sequencing	imperative	prefix
predicate-based invocation	yes	by adding “when”
EPP	implemented	proved
global variable lookup	yes	no
global completion	yes	no

TABLE 1. Correspondence and Differences

5. Correspondence with CDL

In this section, we briefly outline relationship between CDL and the global/local calculi we have used in the previous sections. The correspondence/differences are summarised in Table 1.
Some comments:

- Channels are one of the fundamental elements in communication-based languages as well as in security engineering, arising in diverse forms (such as sockets, remote object IDs, and URLs). Even though an informal global description may not mention channels (this is because the names of participants play the role of channels), they become essential when exception and channel passing are involved. In fact, in standard distributed programming, we may use multiple channels (often in the shape of transport connections) in one unit of conversation.
- CDL channels are located at the inputting side, representing the ports where the sender writes to. Formalisms are more general, using channels both for input and for output.
- Concerning session initiation, this is done implicitly in CDL. In our calculi, we place the explicit session initiation which makes the underlying operational and type structure more explicit and more amenable to analyses. This does not prevent us from using the calculus to represent practical business protocols since we may regard the session initiation and the subsequent action to be combined into a single message in implementation.
- Co-relation is one of the significant features of CDL. Co-relation can be considered as a way to collectively treat multiple sessions as one conversation unit. Though we have not been treated in this work, this feature can be cleanly represented in formal calculi. One method is to use the so-called polyadic synchronisation.
- CDL does not have a proper notion of type checking nor type inference. However it is equipped with such notions as relationship, roles and participants, whose specifications are related with each other through XML schemas. These constructs play an important role as part of documentation. These data will be usable as a basis of typing, using the so-called by-name approach (as found in Java).
- In the current CDL specification, type checking (i.e. verifying if a particular choreography is well typed) is not part of the specification. Such type checking may as well be partly complemented by type inference (i.e. elaborating untyped phrases with appropriate types). These verifications can be done formally in the calculus, i.e. we can provide an algorithm which, given an interaction I and a type t, checks whether the t is a good type for I. Transporting this facility into a CDL development tool will be one of the significant future topics.

- As we saw above, exception are indispensable for managing many interesting real application situations. One thing missing in WS-CDL would be the ability of handling exceptions locally, with a standard local scoping rule. This topic may deserve further consideration.
- Repetition of instructions is usually dealt with while loops. In the calculus we use recursion, another mechanism which can faithfully emulate the standard loop operation as well as many forms of recursive calls. They also enjoy many theoretical features. This does not mean it is better to replace loops with recursion: when a loop behaviour is intended, writing it with a loop often leads to a more understandable program.
- Sequencing of interactions can be treated in two different ways, i.e. the way it is done in CDL and the way it is done in π-calculus. In CDL, a standard imperative language construct ";" is adopted. In our formalisms, we are using the simple prefixing operator. Superficially, the latter construct is less powerful than the former, mainly because it assumes only very simple operations are allowed before the ".". On the contrary, when using ";" we can combine complex expressions such as those combined by the parallel operator. Again there is a precise embedding of ";" into the prefixing in combination with other constructs, so we lose no generality in using "." while allowing easier analysis.
- CDL is equipped with the predicate-based invocation mechanism (for which we used the construct **when**). This mechanism is powerful for various specifications, but it also demands a heavy implementation mechanism. Exploration of cases where this construct becomes indispensable would become important for understanding its status in structured concurrent programming.
- Various globalised features of CDL are incorporated because they often naturally arise in business protocols. Their semantic content however may not be precisely understood. Note globalised behaviour has to be, in effect, realised by interactions among distributed peers. Therefore, at least at the level of formalisms, the understanding of how a certain global construct may be realised by interactions is a prerequisite for their proper inclusion in formalisms. Precise appreciation of what high-level global abstraction would be suitable for describing communication-centred software behaviour, and how they relate to their local (communication-based) realisation, is an important topic for future study.

Part 2

A Formal Theory of Structured Global Programming

6. Introduction to Part 2

Part II develops a theory of *end-point projection (EPP)*, which gives an exact condition and framework by which we can relate the global description of communication-centric software to its local description. The theory is intended to offer one of the central formal underpinnings of W3C's web service choreography description language, WS-CDL [**51**]. The development of the theory has benefitted greatly from the dialogue between the invited scientists of W3C WS-CDL Working Group and WG's members: in fact, without this dialogue, this theory may not have been developed, at least at this moment and in its current shape.

The presented theory may be considered as offering a formal substrate for designing, implementing and using distributed applications written in WS-CDL and related languages, by establishing principles by which applications' global description and their local description are naturally and precisely related. In particular, the theory may be usable, in combination with results from other research threads, as a mathematical underpinning of various tools and infrastructural support for web services, including those for static and dynamic verification. One such effort is underway, using an open-source reference implementation of WS-CDL [**40**].

In the rest of Part 2, Section 7 informally motivates the idea of end-point projection and its theory, and summarises key technical results and their engineering relevance. Sections 8, 9 and 10 formally introduce the global calculus, centring on its dynamic semantics (reduction) and static semantics (type disciplines). Section 11 and 12 does the same for the local calculus. Sections 13, 14 and 15 introduces key descriptive principles for the global calculus which form a basis of the theory. Section 16 establishes the main results of the paper, the exact correspondence in type structures and dynamics between descriptions in the global calculus and those in the local one, through an end-point projection. Section 17 discusses extensions. Section 18 positions the presented ideas in a historical context, compares the present work with existing work, and concludes with further topics. Some of the auxiliary proofs and definitions are left to Appendix.

7. Informal Preview

7.1. What is end-point projection? End-point projection, or EPP for short, is a concept frequently discussed throughout the development of CDL in the W3C WS-CDL working group. Its basic idea is simple, and may be summarised as follows.

> *Let's write down a communication-centred concurrent program (in this case a business protocol) globally, then project it to each end-point so that we can obtain a local description which realises the original global description through their interaction.*

As a simple example, consider an interaction:

(11) $$\mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathit{B2Sch}\langle \mathsf{QuoteAccept},\ \mathit{100},\ x\rangle\,.\,\mathit{0}$$

which is an interaction between a Buyer and a Seller, the former communicating it accepts the quote with price 100 pounds (or dollars or whatever currency you like). This simple global description is projected onto two end-point (local) descriptions:

(12) $$\mathsf{Buyer}[\ \overline{\mathit{B2Sch}}\langle 100\rangle.0\ ]$$

and

(13) $$\mathsf{Seller}[\ \mathit{B2Sch}(x).0\ ]$$

Here description of an "interaction" in (11) (in which both sending and receiving of information are one thing) is decomposed into its local communication actions (in which a sending action of (12) and a receiving action of (13) are separate). We can see that, if Buyer does the specified sending and Seller the receiving, then precisely the interaction as specified in (11) takes place. So we can regard (12) and (13) as how local agents should behave if we wish realise the global interaction as described in (11). As such, they can be regarded as local programs implementing original global description, or alternatively as local monitors which constrain the behaviour of each agent.

7.2. Why EPP matters. Why does EPP matter? First, without EPP, we do not have a clear idea how a global description can be executed, and, therefore, its computational meaning is not clear: a central idea of web services, or in general communication-centred programs and services, is that independently running [1] concurrent agents achieve their application goals through their communication with each other. Thus a global description should be considered as describing behaviour of distributed communicating processes: the latter is the meaning of the former. In this sense, it is only when a uniform notion of EPP is given that the computational content of global descriptions is determined.

Second and relatedly, EPP is an essential basis for diverse engineering applications of global descriptions. Once we have a clear notion of EPP, it offers, for each end-point, what local behaviour a given global description specifies: if we wish to monitor whether an independently developed end-point program behaves in a way specified by a global description, then we can compare the former with the EPP of the latter. Or if we wish to develop a program referring to a given global description of their interactions, we can start from the EPP of the latter: and after developing a fully specified program, or even during the development, a programmer/designer working on each endpoint program can check whether it conforms to the original global description with respect to its communication behaviour (such validation, which we may call *conformance validation*, will be particularly useful in collaborative program development). Or we can even develop a global description language which can specify full algorithmic details specification at each-end point in which case the result of performaing an EPP on a detailed global specification onto all the end-points offers directly executable distributed programs whose behaviour is by definition conformant to the original global specification.

[1] Here "independence" indicates primarily about synchronisation boundaries, but it can also indicate protection domain (security) boundaries.

Thirdly, EPP offer a central underpinning for the theoretical understanding of the structures of global description and their use. Indeed, EPP opens the essential conduit between theories of processes and web service engineering, by relating the global description to the equivalent description in the shape of communicating processes. The established connection enables application of algebras, logics and types of theories of process calculi in the present engineering context. It also (as our subsequent inquiries make clear) leads to a deep structural analysis of global descriptions. Web service engineering demands theoretical foundations because it is about interoperability among disparate agents inhabiting distinct protection domains (or, in a more wordily term, organisations with possibly conflicting interests and complex trust relationships). In such a context, different organisations need a clear shared understanding on how they are to interact with each other in a given business protocol. As an example, consider a business protocol which is about transaction of stocks and which need be bound by regulations. We need a clear criteria as to whether each end-point (organisation) is acting conforming to the protocol. In fact, conformance of the protocol itself to a regulation should initially be clarified, for which we need clear engineering understanding preferably backed up by a theoretical basis. We expect many key elements of theories of processes will offer critical engineering tools in this context.

7.3. Criteria for EPP. An ad-hoc EPP framework may not work: in fact, it never works. This is for simple reasons. First, we wish to implement EPP as an algorithm which can once and for all map a large class of global descriptions to their local counterparts. Thus we need a general way to relate global descriptions to local ones. Second, in relating global description to local description, we wish to avoid the situation where generated local description by different notions of EPP are not compatible with each other. This is especially true when a global description serves as a *reference description* of software infrastructure in an organisation or a social domain, used as a key reference for various business decisions, interoperability and infrastructural development (this is in fact one of the stated goals of WS-CDL [**51**]).

So we need a general framework for EPP, which can uniformly map a general class of global descriptions onto their end-point counterparts. But how can we know a given EPP is correctly projecting a global description to a local description? An informal, but important, engineering criteria is that the resulting local descriptions have *intuitively* a clear and direct connection to the original global description. That is, a designer who specifies software behaviour by a global description should not have surprises when the real computation is realised by communications among projected local processes. From the viewpoint of interoperability, it is also important that we have a general and uniform scheme which can be applied to a large class of global descriptions (note that, assuming we use a public standard for global descriptions, if we have a sufficiently general and satisfactory EPP mapping, this gives us a firm basis for interoperability). Apart from these two informal criteria, the following three are natural formal criteria by which we can measure the effectiveness of an EPP scheme (which are in fact closely related to the two informal criteria we just noted).

- Mapping preserves types and other well-formedness conditions.
- The projected local description implements all behaviours expected from the original global description. Concretely, actions expected from a global description should be faithfully realised by communication among a collection of projected end-points. This property may be called *completeness of EPP*.
- In the reverse direction, locally projected communicating processes should not exhibit observable behaviour not prescribed in global description, as far as its predefined interface goes.[2] Concretely, communications among projected peers should not go beyond actions stipulated in the original global description. This may be called *soundness of EPP*.

For these criteria (especially the latter two) to make sense in practice, we should have a precise way to say, among others, what course of actions (their kinds and structures) are stipulated in a global description, and what course of interactions (their kinds and structures) are expected from

[2] Local programs may as well need to engage in actions outside of those prescribed even just for implementing those prescribed actions.

a collection from local descriptions. To do so, we can follow the standard framework in process algebras [**8, 22, 32, 35**] and programming language semantics [**20, 41, 53**], defining formal syntax, well-formedness (type disciplines) and evolution of behaviour (dynamic semantics). By mathematically defining these ideas, we can now formulate correctness criteria without ambiguity as well as a means to prove (or refute) whether a framework of an EPP satisfies the given criteria (of course the use of formal definitions of semantics of process languages go beyond its use in end-point projections, including a reference for concrete implementation, a formal basis for developing diverse forms of verification technologies, and deeper inquiries into mathematical properties of these descriptive frameworks themselves, cf. [**36**]).

The aim of the rest of the present note is nothing but carrying out the program just outlined: formalising central notions of global/local languages in their distilled form; presenting formal criteria for correctness of EPP; and study a general framework of EPP including formal arguments for its correctness, including three natural descriptive principles under which the presented EPP results in sound and complete local descriptions. In particular we shall present:

- Formal definition of static and dynamic semantics of the global and local calculi, which distills respective descriptive paradigms/languages (static semantics specifies a type discipline for description, dynamic semantics specifies how computation proceeds in a given description). Type disciplines in respective formalisms act as a basis of the whole technical development in the paper.
- A theory of end-point projection, which maps a global description to local description, as well as offering a means to examine its properties. We first present three basic principles for global descriptions which defines a notion of "well-formedness" of description. Then we introduce a simple inductive algorithm which maps each well-formed global description onto a collection of local descriptions (one for each end-point), and present formal arguments that this map is both sound and complete, with respect to static and dynamic semantics of respective formalisms.

The theoretical development focusses on key elements of global/local formalisms without such features as timeout and exception. We believe there are no unsurmountable technical obstacles to extend the present theory to these additional features.

8. Global Calculus (1): Syntax

In this section and the next, we introduce the syntax and dynamic/static semantics of the global calculus. The *dynamic semantics* specifies an abstract notion of "computation" underlying a formalism. In the case of Turing Machine, this is a move of a head and read/write of each slot of a tape. In the case of the λ-calculus, this is an application of a lambda term to an argument and a subsequent substitution (for example, $(\lambda x.fx)3 \to f3$ indicates that, when the function $(\lambda x.fx)$ is applied to an argument 3, the result is another application $f3$). In the case of a global calculus, this is represented as a transition from a global description to another global description, carrying out each step of interaction (exchange of a message). Since each participant may own its own local variables, such transition can also involve collection of local variables of the participants involved.

From an engineering viewpoint, the dynamic semantics pins down a mathematical notion which designers, implementors and users can refer to when they wish to discuss about dynamic behaviour of description with rigour. For example, this would allow us to state with precision whether an implemented program conforms to the description or not. The dynamic semantics is defined using an intuitive notation,

$$(\sigma, I) \;\to\; (\sigma' I)$$

which says a global description I in a state σ (which is the collection of all local states of the participants) will be changed into I' in a new configuration σ'. This idea comes from the small-step semantics given to imperative languages (cf.[**41, 53**]).

The description of interactions in the global calculus centres on a notion of *session*, in which two interacting parties first establish a private connection and do a series of interactions through that private connection, possibly interleaved with other sessions. More concretely, processes first exchange fresh session channels for a newly created session, then use them for interactions belonging to the session (this is equivalent to the more implicit framework where identity tokens in message content are used for signifying a session). This idea has a natural association with a simple type discipline, where we represent a structured sequence of interactions between two parties as an type. Here "types" mean syntactic annotation on descriptions of interactions: this annotation describes an abstract notion of interface of a service (or a shared service channel), and is inferred by typing rules for each description following its syntactic structure. For example, consider an interaction:

$$\begin{array}{ll}(14) & \mathsf{Buyer} \to \mathsf{Seller} : s\langle \mathsf{RequestQuote},\ \mathit{productName},\ x\rangle. \\ & \mathsf{Seller} \to \mathsf{Buyer} : s\langle \mathsf{ReplyQuote},\ \mathit{productPrice},\ y\rangle\end{array}$$

In (14), a Buyer requests a quote for a product, specifying the product name, through a session channel s: then, through the same channel, a Seller replies with the quote value (one may consider ch to be a socket connection). This interaction at s can be abstracted by the following session type:

$$(15) \qquad s \uparrow \mathsf{RequestQuote}(\mathtt{String}).\ s \downarrow \mathsf{ReplyQuote}(\mathtt{Int})$$

The session type in (15) abstracts a sequence of actions performed at ch, specifying the following abstract behaviour:

> *First sends ("↑") a string with operation name* RequestQuote*, then receives ("↓") an integer with operation name* ReplyQuote.

Note this abstraction is given from the Buyer's viewpoint: we can equally present the abstraction for the Seller's action:

$$(16) \qquad s \downarrow \mathsf{RequestQuote}(\mathtt{String}).\ s \uparrow \mathsf{ReplyQuote}(\mathtt{Int})$$

which simply reverses the direction of information flows. Note that, in this way, there is a natural notion of **duality** associated with session types.

Section 3 and Section 4 complete the presentation of the global calculus. The first introduces the formal syntax of the global calculus, with many illustrations. In Section 4 we present the dynamic semantics of the calculus, followed by its static semantics. We then show a basic relationship between the dynamic semantics and the static semantics: when a computation happens in a well-typed description, the result is always well-typed again.

8.1. Formal Syntax. The formal syntax of the global calculus is given by the standard BNF. Below symbols $I, I', \ldots$ denote *terms* of the global calculus, also called *interactions*. Terms describe a course of information exchange among two ore more parties from a global viewpoint.

$$
\begin{array}{rlr}
I ::= & A \to B : ch(\boldsymbol{\nu}\,\tilde{s})\,.\,I & \text{(init)} \\
 & \mid A \to B : s\langle \mathsf{op},\ e,\ y\rangle\,.\,I & \text{(com)} \\
 & \mid x@A := e\,.\,I & \text{(assign)} \\
 & \mid \mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2 & \text{(ifthenelse)} \\
 & \mid I_1 + I_2 & \text{(sum)} \\
 & \mid I_1 \mid I_2 & \text{(par)} \\
 & \mid (\boldsymbol{\nu}\,s)\ I & \text{(new)} \\
 & \mid X^A & \text{(recVar)} \\
 & \mid \mathbf{rec}\ X^A\,.\,I & \text{(rec)} \\
 & \mid \mathbf{0} & \text{(inaction)}
\end{array}
$$

The grammar above uses the following symbols.[3]

- $a, b, c, ch, \ldots$ range over a collection Ch of *service channels* (also called *session initiating channels*). They may be considered as shared channels of web services.
- $s, s', \ldots$ range over a collection $\mathcal{S}$ of *session channels*. Session channels designate communication channels freshly generated for each session. They can be implemented in various ways: in TCP, the same concept is realised by so-called *connection* (also called *session*). In web services, they are realised by sending freshly generated identity information as part of messages.
- $A, B, C, \ldots$ range over a collection $\mathcal{P}$ of *participants*. Participants are those who are engaged in communications with others, each equipped with its own local state. Each participant may have more than one threads of interactions using multiple channels.
- $x, y, z, \ldots$ range over a collection of *variables*, which are close to variables in the traditional programming languages such as Pascal and C, in that their content is updatable.
- $X, Y, Z, \ldots$ range over a collection of *term variables*, which are used for representing recurrence (loop) in combination with recursion $\mathbf{rec}\ X.I$. Note term variables occur annotated with participants.
- $e, e', \ldots$ range over *expressions*, given by the grammar:

  $$e \quad ::= \quad x \mid v \mid f(e_1, \ldots, e_k).$$

  where f ranges over an appropriate set of function symbols (including standard arithmetic/boolean operators). Above $v, w, \ldots$ range over atomic values such as natural numbers and booleans.

Each construct in the above grammar is illustrated in the next subsection.

8.2. Illustration of Syntax. The initial two constructs represent communications. First,

$$A \to B : b(\boldsymbol{\nu}\,\tilde{s})\,.\,I$$

indicates that A invokes a service channel b at B and initiating a new session that will use fresh session channels $\tilde{s}$, followed by interaction I. Subsequent communications in I belonging to this session are done through $\tilde{s}$ (I can have other communications belonging to different sessions). In the main part of the present study we shall assume A and B are distinct.[4] As $\tilde{s}$ should be local to the session (i.e.

[3] As is standard, we assume there is an unbounded supply of distinct symbols in each syntactic category.

[4] This is a natural constraint if we wish to describe inter-participants interactions, which are often the only focus of many business protocols. As we shall discuss at the end of the present paper (cf. §17.2), the theoretical framework of the present study works intact when we allow intra-participant interactions.

unknown outside), we set each $s_i \in \tilde{s}$ to be bound in I. Second,

$$A \rightarrow B : s\langle \mathsf{op},\ e,\ y\rangle$$

expresses the sending action by A whose message consists of a selected operator op, with the receiver B. The value of the expression e (which can only use variables located at A) is assigned to the variable y (which must be located at B). Third, another primitive operation is *assignment*, which is the typical basic operation in imperative languages.

$$x@A := e\,.\,I.$$

The assignment is a local operation at the specified participant (A above), where a variable at A is updated with the result of evaluating e, also located at A.

We can use conditional to branch the course of actions:

$$\mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2$$

which will evaluate e once and, if it evaluates to true, the branch I_1 will be executed, else branch I_2. Note the condition e is located at A. Or, instead of explicitly selecting one of the branches, we can choose one nondeterministically:

$$I_1 + I_2$$

which either behaves as I_1 or as I_2. The summation operator $+$ is commutative and associative, so that we often write $\Sigma_i I_i$ for the n-fold sum of interactions.

We can also launch two threads of interactions in parallel:

$$I_1 \mid I_2$$

denotes the parallel composition. However, unlike the standard process calculi, there is no communication between I_1 and I_2: $I_1|I_2$ just means two independent threads of interactions. Another construction:

$$(\boldsymbol{\nu}\,s)\ I$$

is the restriction (or hiding) of a session channel, where $(\boldsymbol{\nu}\,s)$ binds free occurrences of s in I. This is used for designating newly created session channels when a session is initiated. $(\boldsymbol{\nu}\,\tilde{s})\ I$ stands for a sequence of restrictions. Since restriction is only added when an outermost initialisation prefix reduces, it is natural to stipulate:

CONVENTION 1. Henceforth we only consider terms in which restrictions never occur under prefixes (initiation, communication and assignment) nor do they occur in a summand of a summation.

Interaction which can be repeated unboundedly is realised by recursion. We start from a recursion variable X^A which has an annotation of a participant name (this annotation is later used in end-point projection: in brief, it indicates the principal participant who determines whether to recur or not). Then the term

$$\mathbf{rec}\ X^A\,.\,I$$

is the standard recursion construct, where $\mathbf{rec}\ X^A$ is called *recursor*, with X binding its free occurrences in I. We assume that whenever X occurs free in I of $\mathbf{rec}\ X^A\,.\,I$, X should always be annotated with A (the type discipline we present later automatically guarantees this property). This annotation plays an essential role in our typing later. However, when they are irrelevant (especially in examples), we often omit these annotations. Finally,

$$0$$

is the inaction, representing the lack of actions (it may be considered empty parallel composition or the empty/inactive choreography).

For expressions, we assume variables, first-order atomic values such as integers, and first-order operators such as arithmetic and boolean operations. We do *not* include channels and session channels as expressions for the present inquiry (cf. Section 18).

8.3. Examples. We illustrate the syntax through simple examples. These examples will be used throughout the paper as running examples.

EXAMPLE 1. (Syntax, 1) The following example is from Part I.

$$\begin{array}{c}\{\mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathit{B2Sch}\langle \mathsf{QuoteAccept},\ \mathit{100},\ x\rangle \,.\, \mathit{I_1}\} \\ + \\ \{\mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathit{B2Sch}\langle \mathsf{QuoteReject},\ x_{\mathsf{AbortNo}},\ y\rangle \,.\, I_2\}\end{array} \tag{17}$$

This example, as others, uses easy-to-read strings for channel/operator/variable names. Buyer and Seller are participants (which we write $A, B, \ldots$ in the formal syntax); *B2Sch* is a session channel name (which we write $s, s', \ldots$ in the grammar); and QuoteAccept and QuoteReject are operation names (which are $op, op', \ldots$ in the grammar). Thus, as a whole, (17) can be read as follows:

> *Through a session channel* B2Sch*, Buyer selects one of the two options offered by Seller,* QuoteAccept *and* QuoteReject*. If the first option is selected, Buyer sends the quote "100" which will be stored in x by Seller and proceeds to* I_1*. In the other case, Seller sends the abort number stored in the variable* x_{AbortNo} *which will be stored in y by the Seller and proceeds to* I_2*.*

Note the sum $+$ is informally interpreted as *internal sum* for Buyer (i.e. Buyer initiates this choice) and as *external sum* for Seller (i.e. Seller passively waits for one of the branches (operators) to be chosen by the environment). This reading will become formalised when we consider its end-point projection.

EXAMPLE 2. (Syntax, 2) A refinement of the description above follows.

$$\begin{array}{l}\textbf{if } x_{\mathtt{quote}} \leq 1000\,@\mathtt{Buyer}\ \ \textbf{then} \\ \quad \{\ \ \mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathit{B2Sch}\langle \mathsf{QuoteAccept}, 100, \mathsf{x},\ .,\ I\rangle_{\mathit{1}} \quad \} \\ \textbf{else} \\ \quad \{\ \ \mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathit{B2Sch}\langle \mathsf{QuoteReject}, \mathsf{x}_{\mathsf{AbortNo}}, \mathsf{y},\ .,\ I\rangle_{\mathit{2}}\ \ \}\end{array} \tag{18}$$

The description now specifies the "reason" why each branch is taken. Notice the condition in the conditional branch, $x \leq 1000$, is explicitly *located*: the description says this judgement takes place at Buyer. Note also the description is still the external choice for Seller: it is Buyer who selects one of the options, which Seller waits for passively. The description becomes self-contained by adding an initial session invocation at a service channel, say ch, and a request for a quote.

$$\begin{array}{l}\mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathsf{ch}(\boldsymbol{\nu}\, \mathit{B2Sch}, \mathit{S2Bch})\,. \\ \mathsf{Seller} \rightarrow \mathsf{Buyer} : \mathit{S2Bch}\langle \mathsf{Quote},\ \mathit{100},\ y\rangle\,.. \\ \textbf{if } x_{\mathtt{quote}} \leq 1000\,@\mathtt{Buyer}\ \ \textbf{then} \\ \quad \{\ \ \mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathit{B2Sch}\langle \mathsf{QuoteAccept}, 100, \mathsf{x},\ .,\ I\rangle_{\mathit{1}} \quad \} \\ \textbf{else} \\ \quad \{\ \ \mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathit{B2Sch}\langle \mathsf{QuoteReject}, \mathsf{x}_{\mathsf{AbortNo}}, \mathsf{y},\ .,\ I\rangle_{\mathit{2}}\ \ \}\end{array} \tag{19}$$

Initially Buyer invokes Seller to initiate a session with two session channels, *B2Sch* and *S2Bch*. The rest is the same with the previous description.

EXAMPLE 3. (Syntax, 3) A session can have multiple session names for communication. This is the standard practice in business protocols and other interaction-centric applications, and is essential to have multiple parallel interactions inside a single session. As an example, suppose that the buyer wants to start a session at a channel acc in which it communicates acceptance of a quote on a session name *Op* and, in parallel, sends its address on a session name *Data*. This can be expressed as:

$$\begin{array}{l}\mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathsf{ch}(\boldsymbol{\nu}\, \mathit{Op}, \mathit{Data})\,. \\ \quad \{ \\ \qquad \mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathit{Op}\langle \mathsf{QuoteAccept},\ \mathit{100},\ x\rangle \,.\, \mathbf{0} \quad | \\ \qquad \mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathit{Data}\langle \mathsf{QuoteAccept},\ \mathit{Address},\ y\rangle \,.\, \mathbf{0} \\ \quad \}\end{array} \tag{20}$$

Here, two session channels, *Op* and *Data*, are communicated at the time of session initiation at channel *ch*. Using these two channels, we reach

Examples of other constructs, such as recursion and hiding, will be given in later sections.

8.4. Comments on Syntax. The syntactic constructs we have presented above only include the core part of the global calculus. This is to present theory of end-point projection in a simplest possible form. Below we discuss basic extensions and alternatives of the syntax.

Channel/session passing: Values may as well include channels and session channels. When session channels are passed around, we need to obey a certain linearity constraint, as discussed in [**24**].

Variables and binding: We may as well use logical variables rather than imperative variables as receptors of communicated values, so that the result of communication is instantiation of values rather than assignment. Even in this case we may include imperative variables and its assignment, so that we can represent the notion of local states cleanly.

Operators: Operators $f(\ldots)$ in expressions can become more complex. Among others, WS-CDL includes operators which query current time and other data, which can involve reference to distributed states.

Mutual exclusion and atomicity: An important role is played by a mutual exclusion operator. The introduction of a mutex operation would incur issues of deadlock which can be taken care of with a type system. Introducing mutual exclusion would also imply changing the operational semantics as we would need to take care of variables access, which may be blocked by a mutex. We may also introduce a transactional "atomic" statement which guarantees atomicity (noninterference) of a block from local read and writes by other threads inside the same participant. This may be implemented using two-phrase locking or more optimistically using software transaction (however note interactional nature of concerned descriptions makes rollbacks more subtle than simple imperative programs: this issue parallels treatment of input/output in software transactional memories studied by Harris and others).

Exception, timeout and predicate-based invocation: Part I discusses how exception, timeout and predicate-based invocation (written $\mathsf{when}\ e@A\ \mathsf{do}\ I$, which waits for the guard e to become true and executes I, unlike the conditional) can become useful for representing advanced forms of interactions.

Loop and other imperative constructs: The standard while operator can be easily encoded in the formalism. The term:

$$\mathsf{while}\ e@A\ \mathsf{do}\ I$$

is encoded into:

$$\mathbf{rec}\ X\,.\,\mathsf{if}\ e@A\ \mathsf{then}\ I \Rightarrow X\ \mathsf{else}\ \mathbf{0}$$

where $I \Rightarrow X$ is a syntactic operation which attaches the variable X to the leaves (ends) of abstract syntax trees in I. Similarly the sequencing $I;I'$ can be encoded using sequencing (there is a non-trivial interplay with the parallel construct). Considering these features as explicit syntax will be useful when we need to directly treat practical descriptive languages such as WS-CDL in an extension of the present theory.

9. Global Calculus (2): Reduction

9.1. Basic ideas of Reduction. Computation in the global calculus is represented by a step-by-step transition, each step consisting of:

(1) Execution of a primitive operation, which can be communication, assignment and conditional.
(2) Effects the execution above has on the local state of an involved participant.

To formalise this idea, we use a *configuration* which is a pair of a *state* (a collection of the local states of all participants involved) and an interaction, written $(\sigma,\ I)$. Formally a *state*, ranged over by $\sigma, \sigma', \ldots$, is a function from $Var \times \mathcal{P}$ to Val, i.e. a variable at each participant is assigned a value in a store. We shall write $\sigma@A$ to denote the portion of σ local to A, and $\sigma[y@A \mapsto v]$ to denote a new state which is identical with σ except that $\sigma'(y,A)$ is equal to v. The dynamics is then defined in the form:

$$(\sigma,\ I) \ \rightarrow\ (\sigma',\ I')$$

which says I in the configuration σ performs one-step computation (which can be assignment, interaction, etc.) and becomes I' with the new configuration σ'. The relation $\rightarrow$ is called *reduction* or *reduction relation*.[5] For example, communication action will change both the state and the term shape:

$$(\sigma,\ A \rightarrow B : s\langle \mathsf{send},\ 3,\ x\rangle.I) \quad \rightarrow \quad (\sigma[x@B \mapsto 3],\ I)$$

which indicates:

> *"A sends a message* send *and a value* 3*, which is received by B and* 3 *is assigned to a variable x at B as the result, with the residual interaction I".*

Note communication action happens automatically, without first having sending and receiving actions separately and then having their synchronisation. Assignment is treated similarly.

$$(\sigma,\ x@B := 3.I) \quad \rightarrow \quad (\sigma[x@B \mapsto 3],\ I)$$

Since an assignment is located, only x at B is updated, and the next interaction I is unfolded. Interaction can involve choice, where one of the branches is chosen nondeterministically, i.e. we can have either:

$$(\sigma,\ (A \rightarrow B : s\langle \mathsf{ok},\ 3,\ x\rangle.I_1) + (A \rightarrow B : s\langle \mathsf{no},\ 0,\ y\rangle.I_2) \quad \rightarrow \quad (\sigma[x@B \mapsto 3],\ I_1)$$

or

$$(\sigma,\ (A \rightarrow B : s\langle \mathsf{ok},\ 3,\ x\rangle.I_1) + (A \rightarrow B : s\langle \mathsf{no},\ 0,\ y\rangle.I_2) \quad \rightarrow \quad (\sigma[y@B \mapsto 0],\ I_2)$$

will take place: both are legitimate reductions.

The conditional depends on how an expression evaluates. For example, if x at A stores 0, then we have

$$(\sigma,\ \mathsf{if}\ x@A = 0\ \mathsf{then}\ A \rightarrow B : s\langle \mathsf{ok},\ 3,\ x\rangle.I_1\ \mathsf{else}\ \ldots) \quad \rightarrow \quad (\sigma[x@B \mapsto 3],\ I_1)$$

But if x at A stores say 1, then the second branch will be selected.

For recursion, we expect a recurring behaviour. For example, the following is a silly behaviour which just continues to assign 1 to a variable. In this case we may as well have:

$$(\sigma,\ \mathbf{rec}\ X^B.x@B := 1.X^B) \ \rightarrow\ (\sigma[x@B \mapsto 3],\ \mathbf{rec}\ X^B.x@B := 1.X^B)$$

We shall realise such recurrence through the use of structural rules. Other constructs such as conditionals and choice are standard.

In the following subsection, we illustrate the notion of reduction for each construct one by one.

[5] The term "reduction" originally came from the λ-calculus, where the sole purpose of computation is to reduce to a final answer of calculation. While it is not entirely suitable for interaction computation, we use the term from convention and from our respect to the basic formalism which started semantics studies.

9.2. Reduction Rules. Reduction relation is defined by having one rule for each construct, together with associated rules. First we have a rule for session-initiating communication:

$$(\textsc{Init})\ \frac{-}{(\sigma,\ A \to B : b(\boldsymbol{\nu}\,\tilde{s})\,.I) \to (\sigma, (\boldsymbol{\nu}\,\tilde{s})\ I)}$$

where $\tilde{s}$ is a vector of one or more pairwise distinct session channels. The rule says that, after A communicates with B for session initiation with fresh session channels $\tilde{s}$, A and B share $\tilde{s}$ locally (indicated by ν-binding), and the next I is unfolded. The state σ stays as it is since no communication of values takes place.

We have already seen an example of reduction representing communication through a session channel: the formal rule follows.

$$(\textsc{Comm})\frac{\sigma \vdash e@A \Downarrow v}{(\sigma,\ A \to B : s\langle \mathsf{op},\ e,\ x\rangle\,.I) \to (\sigma[x@B \mapsto v],\ I)}$$

The premise of the rule above uses the judgement (called *evaluation judgement*):

$$\sigma \vdash e@A \Downarrow v$$

which says:

Expression e is evaluated into the value v in the A-portion of the state σ.

For example, if σ says x at A stores 3, then we have $\sigma \vdash (x+x)@A \Downarrow 6$. Thus the expression to be communicated is evaluated in the *source* part of the state: and the value communicated is assigned in the *target* part of the state.

The formal rule for assignment is given as:

$$(\textsc{Assign})\frac{\sigma \vdash e@A \Downarrow v}{(\sigma, x@A := e\,.I) \to (\sigma[x@A \mapsto v],\ I)}$$

which updates the state at the participant A and unfolds the next interaction.

The rules for conditional assumes, again using the evaluation judgement, that the conditional expression evaluates to either tt (for truth) or ff (for falsity). In the former:

$$(\textsc{IfTrue})\frac{\sigma \vdash e@A \Downarrow \mathsf{tt}}{(\sigma,\ \mathsf{if}\ e@A\ \mathsf{then}\ I\ \mathsf{else}\ {}_1I_2) \to (\sigma, I_1)}$$

Symmetrically, when the condition evaluates to the falsity:

$$(\textsc{IfFalse})\frac{\sigma \vdash e@A \Downarrow \mathsf{ff}}{(\sigma,\ \mathsf{if}\ e@A\ \mathsf{then}\ I\ \mathsf{else}\ {}_1I_2) \to (\sigma,\ I_2)}$$

The rule for summation is standard:

$$(\textsc{Sum})\frac{-}{(\sigma,\ I_1 + I_2) \to (\sigma',\ I_i)} \qquad (i = 1,2)$$

For parallel composition, the rule is defined just by considering interleaving of two components. Thus we define:

$$(\textsc{Par})\frac{(\sigma,\ I_1) \to (\sigma',\ I_1')}{(\sigma, I_1 \mid I_2) \to (\sigma',\ I_1' \mid I_2)}$$

where we reduce the left-hand side. The symmetric rule is defined similarly (which is again subsumed by the use of the structural rules we stipulate later).

For restriction we have:

$$(\textsc{Res})\frac{(\sigma,\ I) \to (\sigma',\ I')}{(\sigma, (\boldsymbol{\nu}\,\tilde{s})\ I) \to (\sigma',\ (\boldsymbol{\nu}\,\tilde{s})\ I')}$$

which says restriction does not affect reduction. For recursion, we use the standard unfolding rule.

$$(\textsc{Rec})\frac{(\sigma,\ I[\mathbf{rec}\,X^A.I/X^A]) \to (\sigma',\ I')}{(\sigma,\ \mathbf{rec}\,X^A.I) \to (\sigma',\ I')}$$

The rule says that:

If the unfolding of $\mathbf{rec}\ X^A.I$*,* $I[\mathbf{rec}\ X^A.I/X^A]$ *(which substitutes* $\mathbf{rec}\ X^A.I$ *for each free* X^A *in* I*) under* σ *reduces to* I' *with the resulting state* σ'*, then* $\mathbf{rec}\ X^A.I$ *itself under* σ *will reach* (σ', I')*.*

Note the participant annotation plays no role in the rule. As we shall discuss later, we can use the structural rule instead to obtain essentially the same reduction. Finally the inaction **0** does not have any reduction. We also use the following rule, which says that when we reduce we take terms up to a certain equality, following [**9, 33**].

$$(\textsc{Struct})\frac{I \equiv I'' \qquad (\sigma,\, I) \to (\sigma,\, I') \qquad I' \equiv I'''}{(\sigma,\, I'') \to (\sigma',\, I''')}$$

where the structural equality $\equiv$ is defined by the following rules:

$$\begin{array}{rcll}
I & \equiv & I' & (I \equiv_\alpha I'\,) \\
I+I & \equiv & I & \\
I_1+I_2 & \equiv & I_2+I_1 & \\
(I_1+I_2)+I_3 & \equiv & I_1+(I_2+I_3) & \\
I|\mathbf{0} & \equiv & I & \\
I_1|I_2 & \equiv & I_2|I_1 & \\
(I_1|I_2)|I_3 & \equiv & I_1|(I_2|I_3) & \\
((\boldsymbol{\nu} s)\, I_1)|I_2 & \equiv & (\boldsymbol{\nu} s)\, (I_1|I_2) & (s \notin \mathsf{fn}(I_2))
\end{array}$$

In the last rule, $\mathsf{fn}(I)$ denotes the free names (including variables, channels and session channels) occurring in I. The relation $\equiv$ is the least congruence on terms including the above equations. While the benefit of the use of structural rules in reduction rules is limited in the present context (in comparison with standard process calculi), considering terms up to $\equiv$ is often natural and adds clarity in practice. We may also use a structural rule for recursion,

$$\mathbf{rec}\ X^A.I \equiv I[\mathbf{rec}\ X^A.I/X^A]$$

to dispense with (Rec) rule given above. Just as (Rec) does, this rule says the recursion and its unfolding have identical behaviour. The resulting reduction is identical up to $\equiv$. In Table 18 we present all the rules.

9.3. Examples of Reduction.

Example 4. (Reduction: Communication) Recall the following term from Example 1

$$I_0 \overset{\text{def}}{=} \begin{array}{l} \{\mathsf{Buyer} \to \mathsf{Seller} : \mathit{B2Sch}\langle \mathsf{QuoteAccept},\ \mathit{100},\ x\rangle\,.\,I_1\} \qquad + \\ \{\mathsf{Buyer} \to \mathsf{Seller} : \mathit{B2Sch}\langle \mathsf{QuoteReject},\ x_{\mathsf{AbortNo}},\ y\rangle\,.\,I_2\} \end{array} \tag{21}$$

We infer the reductions of I_0. There is one reduction for each branch. For the first summand, we note $\sigma \vdash 100@\mathsf{Buyer} \Downarrow 100$ and infer, using (Comm):

$$(\sigma,\, I_0) \to (\sigma[x@\mathsf{Seller} \mapsto 100],\, I_1) \tag{22}$$

Similarly we have the following reduction for the second branch. Assume x_{AbortNo} stores (say) 28 at Buyer in σ, hence $\sigma \vdash x_{\mathsf{AbortNo}}@\mathsf{Buyer} \Downarrow 28$.

$$(\sigma,\, O_0) \to (\sigma[y@\mathsf{Seller} \mapsto 28],\, I_2) \tag{23}$$

These are the all reductions I_0 has up to $\equiv$.

Example 5. (Reduction: Conditional) We deduce reduction for the conditional, using Example 2. First we reproduce the term.

$$I_0' \overset{\text{def}}{=} \begin{array}{l} \mathbf{if}\ x_{\texttt{quote}} \leq 1000\,@\texttt{Buyer}\ \mathbf{then} \\ \quad \{\ \mathsf{Buyer} \to \mathsf{Seller} : \mathit{B2Sch}\langle \mathsf{QuoteAccept},\ \mathit{100},\ x\rangle\,.\,I_1' \quad \} \\ \mathbf{else} \\ \quad \{\ \mathsf{Buyer} \to \mathsf{Seller} : \mathit{B2Sch}\langle \mathsf{QuoteReject},\ x_{\mathsf{AbortNo}},\ y\rangle\,.\,I_2' \ \} \end{array} \tag{24}$$

$$(\textsc{Init})\ \frac{-}{(\sigma,\ A \to B : b(\boldsymbol{\nu}\,\tilde{s})\,.\,I) \to (\sigma, (\boldsymbol{\nu}\,\tilde{s})\ I)}$$

$$(\textsc{Comm})\frac{\sigma \vdash e@A \Downarrow v}{(\sigma,\ A \to B : s\langle \mathsf{op},\ e,\ x\rangle\,.\,I) \to (\sigma[x@B \mapsto v],\ I)} \qquad (\textsc{Assign})\frac{\sigma \vdash e@A \Downarrow v}{(\sigma, x@A := e\,.\,I) \to (\sigma[x@A \mapsto v],\ I)}$$

$$(\textsc{IfTrue})\frac{\sigma \vdash e@A \Downarrow \mathsf{tt}}{(\sigma,\ \mathsf{if}\ e@A\ \mathsf{then}\ I\ \mathsf{else}\ {}_1I_2) \to (\sigma, I_1)} \qquad (\textsc{Par})\frac{(\sigma,\ I_1) \to (\sigma',\ I_1')}{(\sigma, I_1 \mid I_2) \to (\sigma',\ I_1' \mid I_2)}$$

$$(\textsc{IfFalse})\frac{\sigma \vdash e@A \Downarrow \mathsf{ff}}{(\sigma,\ \mathsf{if}\ e@A\ \mathsf{then}\ I\ \mathsf{else}\ {}_1I_2) \to (\sigma,\ I_2)} \qquad (\textsc{Sum})\frac{-}{(\sigma,\ I_1 + I_2) \to (\sigma',\ I_i)} \quad (i = 1,2)$$

$$(\textsc{Res})\frac{(\sigma,\ I) \to (\sigma',\ I')}{(\sigma, (\boldsymbol{\nu}\,\tilde{s})\ I) \to (\sigma',\ (\boldsymbol{\nu}\,\tilde{s})\ I')} \qquad (\textsc{Rec})\frac{(\sigma,\ I[\mathbf{rec}\ X^A.I/X^A]) \to (\sigma',\ I')}{(\sigma,\ \mathbf{rec}\ X^A.I) \to (\sigma',\ I')}$$

$$(\textsc{Struct})\frac{I \equiv I'' \qquad (\sigma,\ I) \to (\sigma,\ I') \qquad I' \equiv I'''}{(\sigma,\ I'') \to (\sigma',\ I''')}$$

FIGURE 18. Reduction Semantics of Global Calculus

If we assume $\sigma@\mathsf{Buyer}(x_{\mathtt{quote}}) = 800$ then we can infer:

$$(25) \qquad (\textsc{IfTrue})\ \frac{\sigma \vdash (800 \leq 1000)@\mathsf{Buyer} \Downarrow \mathsf{tt}}{(\sigma,\ I_0') \to (\sigma,\ \mathsf{Buyer} \to \mathsf{Seller} : \mathit{B2Sch}\langle \mathsf{QuoteAccept},\ \mathit{100},\ x\rangle\,.\,I_1')}$$

Further applying $(\textsc{Comm})$ to the resulting configuration, we conclude:

$$\begin{array}{rcl} (\sigma,\ I_0') & \to & (\sigma,\ \mathsf{Buyer} \to \mathsf{Seller} : \mathit{B2Sch}\langle \mathsf{QuoteAccept},\ \mathit{100},\ x\rangle\,.\,I_1') \\ & \to & (\sigma[x@\mathsf{Seller} \mapsto 100],\ I_1') \end{array}$$

which is the only reduction sequences from $(\sigma,\ I_0')$ in this case. Assume on the other hand $\sigma@\mathsf{Buyer}(x_{\mathtt{quote}}) =$ 1200. Then we have

$$(26) \qquad (\textsc{IfFalse})\ \frac{\sigma \vdash (1200 \leq 1000)@\mathsf{Buyer} \Downarrow \mathsf{ff}}{(\sigma,\ I_0') \to (\sigma,\ \mathsf{Buyer} \to \mathsf{Seller} : \mathit{B2Sch}\langle \mathsf{QuoteReject},\ x_{\mathsf{AbortNo}},\ y\rangle\,.\,I_2')}$$

Hence in this case we have:

$$\begin{array}{rcl} (\sigma,\ I_0') & \to & (\sigma,\ \mathsf{Buyer} \to \mathsf{Seller} : \mathit{B2Sch}\langle \mathsf{QuoteReject},\ x_{\mathsf{AbortNo}},\ y\rangle\,.\,I_2') \\ & \to & (\sigma[y@\mathsf{Seller} \mapsto 28],\ I_2) \end{array}$$

which is again the only possible reduction sequence under the assumption.

EXAMPLE 6. (Reduction: Init, Par and Struct) We next consider Example 3:

$$(27) \qquad J_0 \stackrel{\text{def}}{=} \begin{array}{l} \mathsf{Buyer} \to \mathsf{Seller} : \mathsf{acc}(\boldsymbol{\nu}\,\mathit{Op}, \mathit{Data})\,. \\ \{ \\ \quad \mathsf{Buyer} \to \mathsf{Seller} : \mathit{Op}\langle \mathsf{QuoteAccept},\ \mathit{100},\ x\rangle\,.\,\mathbf{0} \quad \mid \\ \quad \mathsf{Buyer} \to \mathsf{Seller} : \mathit{Data}\langle \mathsf{QuoteAccept},\ w_{\mathsf{Address}},\ y\rangle\,.\,\mathbf{0} \\ \} \end{array}$$

Call two components of the parallel composition, J_1 and J_2. Then by $(\textsc{Init})$ we obtain:

$$(28) \qquad (\sigma,\ J_0) \to (\sigma,\ (\boldsymbol{\nu}\,\mathit{Op}, \mathit{Data})\ (J_1 \mid J_2))$$

By $(\textsc{Comm})$ we have: $(\sigma,\ J_1) \to (\sigma[x@\mathsf{Seller} \mapsto 100],\ \mathbf{0})$, hence by $(\textsc{Par})$ we arrive at:

$$(29) \qquad (\sigma,\ J_1|J_2) \to (\sigma[x@\mathsf{Seller} \mapsto 100],\ \mathbf{0}|J_2)$$

For the symmetric case, assume $\sigma@\mathsf{Buyer}(w_{\mathsf{Address}}) = adr$ (where adr is a string standing for an address) Then by (COMM) we have $(\sigma,\, J_2) \rightarrow (\sigma[y@\mathsf{Seller} \mapsto adr],\, \mathbf{0})$, hence by (PAR) we arrive at:

$$(\sigma,\, J_2 \mid J_1) \rightarrow (\sigma[y@\mathsf{Seller} \mapsto adr],\, \mathbf{0} \mid J_1) \tag{30}$$

Noting $J_1 \mid J_2 \equiv J_2 \mid J_1$, we can now apply ((STRUCT)) to obtain:

$$(\sigma,\, J_1 \mid J_2) \rightarrow (\sigma[y@\mathsf{Seller} \mapsto adr],\, J_1) \tag{31}$$

Note we also simplified the resulting term. In summary, we have two sequences of reductions up to $\equiv$:

$$(\sigma,\, J_0) \;\rightarrow\; (\sigma,\, (\boldsymbol{\nu}\, Op, Data)(J_1|J_2)) \;\rightarrow\; (\sigma[x@\mathsf{Seller} \mapsto 100],\, (\boldsymbol{\nu}\, Data)\; J_2) \;\rightarrow\; (\sigma',\, \mathbf{0})$$

and

$$(\sigma,\, J_0) \;\rightarrow\; (\sigma,\, (\boldsymbol{\nu}\, Op, Data)\; (J_1|J_2)) \;\rightarrow\; (\sigma[y@\mathsf{Seller} \mapsto adr],\, (\boldsymbol{\nu}\, Op)\; J_1) \;\rightarrow\; (\sigma',\, \mathbf{0})$$

where we set $\sigma' \overset{\mathrm{def}}{=} \sigma[x@\mathsf{Seller} \mapsto 100][y@\mathsf{Seller} \mapsto adr]$.

EXAMPLE 7. (Reduction: Recursion) Finally we show an example of recursion, taking the "silly" example $\mathbf{rec}\, X^B.(x@B := 1.X^B)$ before. Noting:

$$(x@B := 1.X^B)[\mathbf{rec}\, X^B.x@B := 1.X^B/X^B] \quad \overset{\mathrm{def}}{=} \quad x@B := 1\,;\, \mathbf{rec}\, X^B.x@B := 1.X^B$$

hence we have:

$$\begin{array}{rcl}
(\sigma,\, \mathbf{rec}\, X^B.x@B := 1.X^B) & \rightarrow & (\sigma[x@B \mapsto 1],\, \mathbf{rec}\, X^B.x@B := 1.X^B) \\
& \rightarrow & (\sigma[x@B \mapsto 1],\, \mathbf{rec}\, X^B.x@B := 1.X^B) \\
& \rightarrow & \ldots.
\end{array}$$

as expected.

10. Global Calculus (3): Typing

10.1. Session Types. As briefly mentioned at the outset of Section 3, we use session types [**24**] as the type structures for the global calculus. In advanced web services and business protocols, the structures of interaction in which a service/participant is engaged in may not be restricted to one-way messages or RPC-like request-replies. This is why their type abstraction needs to capture a complex interaction structure of services, leading to the use of session types. The grammar of types follow.

$$\begin{array}{lll} \theta & ::= & \texttt{bool} \mid \texttt{int} \mid \ldots \\ \alpha & ::= & \Sigma_i s \downarrow op_i(\theta_i)\,.\,\alpha_i \;\mid\; \Sigma_i s \uparrow op_i(\theta_i)\,.\,\alpha_i \;\mid\; \alpha_1 \mid \alpha_2 \;\mid\; \mathbf{t} \;\mid\; \mathbf{rec}\;\mathbf{t}.\alpha \;\mid\; \texttt{end} \end{array}$$

Above $\theta, \theta', \ldots$ range over *value types*, which in the present case only includes atomic data types. $\alpha, \alpha', \ldots$ are *session types*. Note session channels $s, s', \ldots$ occur free in session types (this is necessary because of multiple session channels in a single session, cf. [**24**]). We take $\mid$ to be commutative and associative, with the identity `end`. Recursive types are regarded as regular trees in the standard way [**41**]. Brief illustration of each construct follows.

- $\Sigma_i s \downarrow op_i(\theta_i)\,.\,\alpha_i$ is a *branching input type at s*, indicating possibilities for receiving any of the operators from $\{op_i\}$ (which should be pairwise distinct) with a value of type θ_i.
- $\Sigma_i s \uparrow op_i(\theta_i)\,.\,\alpha_i$, a *branching output type at s*, is the exact dual of the above.
- $\alpha_1 \mid \alpha_2$ is a *parallel composition of* α_1 *and* α_2, abstracting parallel composition of two sessions. We demand session channels in α_1 and those in α_2 are disjoint.
- $\mathbf{t}$ is a *type variable*, while $\mathbf{rec}\;\mathbf{t}.\alpha$ is a *recursive type*, where $\mathbf{rec}\;\mathbf{t}$ binds free occurrences of $\mathbf{t}$ in α. A recursive type represents a session with a loop. We assume each recursion is guarded, i.e., in $\mathbf{rec}\;\mathbf{t}.\alpha$, the type α should be either an input/output type or *n*-ary parallel composition of input/output types.
- `end` is the *inaction type*, indicating termination of a session. `end` is often omitted.

Each time a session occurs at a shared service channel, session channels are freshly generated and exchanged. Thus the interface of a service should indicate a vector of session channels to be exchanged, in addition to how they are used. This is represented by *abstract session type*, or *service type*, in which concrete instances of session channels in a session type are abstracted, written:

$$(\tilde{s})\,\alpha$$

where $\tilde{s}$ is a vector of pairwise distinct session channels which should cover all session channels in α, and α does not contain free type variables. $(\tilde{s})$ binds occurrences of session channels in $\tilde{s}$ in α, which induces the standard alpha-equality.

Before illustrating these types with examples, we introduce a natural notion of duality. The *co-type*, or *dual*, of α, written $\overline{\alpha}$, is given as follows.

$$\begin{array}{rcl} \overline{\Sigma_i s_i \uparrow op_i(\theta_i)\,.\,\alpha_i} & = & \Sigma_i s_i \downarrow op_i(\theta_i)\,.\,\overline{\alpha_i} \\ \overline{\Sigma_i s_i \downarrow op_i(\theta_i)\,.\,\overline{\alpha_i}} & = & \Sigma_i s_i \uparrow op_i(\theta_i)\,.\,\overline{\alpha_i} \\ \overline{\mathbf{rec}\; t\,.\,\alpha} & = & \mathbf{rec}\; t\,.\,\overline{\alpha} \\ \bar{\mathbf{t}} & = & \mathbf{t} \\ \overline{\texttt{end}} & = & \texttt{end} \end{array}$$

For example, the co-type of $s \downarrow \mathsf{QuoteReq}(\texttt{string}).\texttt{end}$ is $s \uparrow \mathsf{QuoteReq}(\texttt{string}).\texttt{end}$, exchanging input and output. The duality plays an essential role in the subsequent technical development.

10.2. Examples of Session Types.

EXAMPLE 8. (Session Type: basics) Consider the following interaction (cf. Example 1), assuming *adr* and *prd* are variables of `string` type, located at both Buyer and Seller.

$$\begin{array}{ll} & \mathsf{Buyer} \rightarrow \mathsf{Seller} : s_1\langle \mathsf{QuoteReq},\, prd,\, prd\rangle. \\ (32) & \mathsf{Seller} \rightarrow \mathsf{Buyer} : s_2\langle \mathsf{QuoteRep},\, \mathit{100},\, y\rangle. \\ & \mathsf{Buyer} \rightarrow \mathsf{Seller} : s_1\langle \mathsf{Purchase},\, adr,\, adr\rangle.\mathbf{0} \end{array}$$

The interface which Seller offers (as far as this interaction goes) can be described by the following session type:

(33) $$s_1 \downarrow \mathsf{QuoteReq}(\mathtt{string}).\ s_2 \uparrow \mathsf{QuoteRep}(\mathtt{int}).\ s_1 \downarrow \mathsf{Purchase}(\mathtt{string}).\ \mathtt{end}$$

the same interaction can be type-abstracted from the viewpoint of Buyer:

(34) $$s_1 \uparrow \mathsf{QuoteReq}(\mathtt{string}).\ s_2 \downarrow \mathsf{QuoteRep}(\mathtt{int}).\ s_1 \uparrow \mathsf{Purchase}(\mathtt{string}).\ \mathtt{end}$$

which is nothing but the co-type of (33). Now let us add a session initiation to (33):

(35) $$\begin{array}{l} \mathsf{Buyer} \to \mathsf{Seller} : \mathsf{ch}(\mathsf{s_1 s_2}). \\ \mathsf{Buyer} \to \mathsf{Seller} : s_1 \langle \mathsf{QuoteReq},\ prd,\ prd \rangle. \\ \mathsf{Seller} \to \mathsf{Buyer} : s_2 \langle \mathsf{QuoteRep},\ 100,\ y \rangle. \\ \mathsf{Buyer} \to \mathsf{Seller} : s_1 \langle \mathsf{QuoteAcc},\ adr,\ adr \rangle.\mathbf{0} \end{array}$$

Then the service type of Seller at channel sh is given as:

(36) $$(s_1 s_2)\ s_1 \downarrow \mathsf{QuoteReq}(\mathtt{string}).\ s_2 \uparrow \mathsf{QuoteRep}(\mathtt{int}).\ s_1 \downarrow \mathsf{Purchase}(\mathtt{string}).\ \mathtt{end}$$

which says: firstly, two fresh session channels $s_1 s_2$ (in this order) are exchanged; then, using these two channels, communication of the represented shape takes place. Thus the service type (36) describes the whole of the behaviour starting from ch, albeit abstractly.

EXAMPLE 9. (Session Type: branching) Let us refine (32) with branching.

(37) $$\begin{array}{l} \mathsf{Buyer} \to \mathsf{Seller} : s_1 \langle \mathsf{QuoteReq},\ prd,\ prd \rangle. \\ \mathsf{Seller} \to \mathsf{Buyer} : s_2 \langle \mathsf{QuoteRep},\ 100,\ y \rangle. \\ \left(\begin{array}{l} \mathsf{Buyer} \to \mathsf{Seller} : s_1 \langle \mathsf{Purchase},\ adr,\ adr \rangle.\mathbf{0} \\ \quad + \\ \mathsf{Buyer} \to \mathsf{Seller} : s_1 \langle \mathsf{Nothanks} \rangle.\mathbf{0} \end{array} \right) \end{array}$$

This can be abstracted, from the viewpoint of Seller:

(38) $$\begin{array}{l} s_1 \downarrow \mathsf{QuoteReq}(\mathtt{string}).\ s_2 \uparrow \mathsf{QuoteRep}(\mathtt{int}). \\ \qquad (s_1 \downarrow \mathsf{Purchase}(\mathtt{string}).\mathtt{end}\ +\ s_1 \downarrow \mathsf{Nothanks}().\mathtt{end}) \end{array}$$

Note the sum $+$ in (38) means the inputting party (here Seller) waits with two options, Purchase and Nothanks: on the other hand, the co-type of (38) (seen from Buyer's side) becomes:

(39) $$\begin{array}{l} s_1 \uparrow \mathsf{QuoteReq}(\mathtt{string}).\ s_2 \downarrow \mathsf{QuoteRep}(\mathtt{int}). \\ \qquad (s_1 \uparrow \mathsf{Purchase}(\mathtt{string}).\mathtt{end}\ +\ s_1 \uparrow \mathsf{Nothanks}().\mathtt{end}) \end{array}$$

in which the sum $+$ in (38) means that the outputting party (here Buyer) may select one of Purchase and Nothanks from the two options.

EXAMPLE 10. (Session Type: recursion) Consider the following behaviour, in which B continuously greets A.

(40) $$\mathbf{rec}\ X^B.\mathsf{B} \to \mathsf{A} : s\langle \mathsf{Greeting},\ \text{“}hello''\!,\ x \rangle.\ X^B$$

We can then abstract this behaviour as, from B's viewpoint:

(41) $$\mathbf{rec}\ Y.\ s\ \uparrow\ \mathsf{Greetings}(\mathtt{string}).\ Y$$

whereas for A the same interaction is abstracted as:

(42) $$\mathbf{rec}\ Y.\ s\ \downarrow\ \mathsf{Greetings}(\mathtt{string}).\ Y$$

which states that A repeatedly receives greetings. As a more meaningful recursion, consider the following refinement of (37):

(43) $$\mathbf{rec}\ X^{\mathsf{Buyer}}.\ \left(\begin{array}{l} \mathsf{Buyer} \to \mathsf{Seller} : s_1 \langle \mathsf{QuoteReq},\ prd,\ prd \rangle. \\ \mathsf{Seller} \to \mathsf{Buyer} : s_2 \langle \mathsf{QuoteRep},\ 100,\ y \rangle. \\ \left(\begin{array}{l} \mathsf{Buyer} \to \mathsf{Seller} : s_1 \langle \mathsf{Purchase},\ adr,\ adr \rangle.\mathbf{0} \\ \quad + \\ \mathsf{Buyer} \to \mathsf{Seller} : s_1 \langle \mathsf{Nothanks} \rangle.\ X^{\mathsf{Buyer}} \end{array} \right) \end{array} \right)$$

This behaviour, seen from the viewpoint of Seller, can be abstracted as the following session type:

$$(44)\qquad \mathbf{rec}\ Y.\left(\begin{array}{l} s_1 \downarrow \mathsf{QuoteReq}(\mathtt{string}). \\ s_2 \uparrow \mathsf{QuoteRep}(\mathtt{int}). \\ \left(\begin{array}{c} s_1 \downarrow \mathsf{Purchase}(\mathtt{string}).\mathtt{end} \\ + \\ s_1 \downarrow \mathsf{Nothanks}().Y \end{array}\right) \end{array}\right)$$

It may be notable that the following conditional has the same session type as (44).

$$(45)\qquad \mathbf{rec}\ X^{\mathsf{Buyer}}.\left(\begin{array}{l} \mathsf{Buyer} \to \mathsf{Seller} : s_1\langle \mathsf{QuoteReq},\ \mathit{prd},\ \mathit{prd}\rangle. \\ \mathsf{Seller} \to \mathsf{Buyer} : s_2\langle \mathsf{QuoteRep},\ \mathit{100},\ y\rangle. \\ \mathbf{if}\ \ \mathrm{reasonable}(y)@\mathsf{Buyer}\ \ \mathbf{then} \\ \qquad \mathsf{Buyer} \to \mathsf{Seller} : s_1\langle \mathsf{Purchase},\ \mathit{adr},\ \mathit{adr}\rangle.\mathbf{0} \\ \mathbf{else} \\ \qquad \mathsf{Buyer} \to \mathsf{Seller} : s_1\langle \mathsf{Nothanks}\rangle.\ X^{\mathsf{Buyer}} \end{array}\right)$$

One can further prefix (45) with a session initiation, for example with $\mathsf{Buyer} \to \mathsf{Seller} : \mathsf{ch}(\mathsf{s}_1\mathsf{s}_2)$, in which case we obtain the service type for ch:

$$(46)\qquad (s_1 s_2)\ \ \mathbf{rec}\ Y.\left(\begin{array}{l} s_1 \downarrow \mathsf{QuoteReq}(\mathtt{string}). \\ s_2 \uparrow \mathsf{QuoteRep}(\mathtt{int}). \\ \left(\begin{array}{c} s_1 \downarrow \mathsf{Purchase}(\mathtt{string}).\mathtt{end} \\ + \\ s_1 \downarrow \mathsf{Nothanks}().Y \end{array}\right) \end{array}\right)$$

which says that, after initialisation request exchanging two fresh session channels (designated as s_1 and s_2), it first waits for a $\mathsf{QuoteReq}$ message at s_1, to which it replies with $\mathsf{QuoteRep}$ via s_2, then it waits for two options $\mathsf{Purchase}$ and $\mathsf{Nothanks}$ at s_1: in the former case it finishes this session while in the latter it recurs to the initial state, waiting for another $\mathsf{QuoteReq}$ message.

10.3. Typing Rules. A *typed term* is a term annotated with types following a set of typing rules. There are two kinds of types we use. *Session types* are assigned to session channels, while *service types* are assigned to service channels. A typed term, which we also call typing judgement, has the shape:

$$(47)\qquad \Gamma \vdash I \rhd \Delta$$

where Γ assigns service types to located service channels, and Δ session types to located session channels. The former is called *service typing*, the latter *session typing* (Γ can also include other forms of assignments). The grammar of service/session typings are given by ($\tilde{s}$ consists of pairwise distinct session channels):

$$\begin{array}{lcl} \Gamma & ::= & \emptyset \quad | \quad \Gamma,\ ch@A : (\tilde{s})\alpha \quad | \quad \Gamma,\ x@A : \mathit{Var}(\theta) \quad | \quad \Gamma,\ X^A : \Delta \\ \Delta & ::= & \emptyset \quad | \quad \Delta,\ \tilde{s}[A,B] : \alpha \quad | \quad \Delta,\ \tilde{s} : \bot \end{array}$$

In a service typing, three forms of assignments are used. First, $ch@A : (\tilde{s})\alpha$ says:

> *A service channel ch is located at A, and ch offers a service interface represented by a service type* $(\tilde{s})\alpha$.

Here "located at A" means the service is offered by A through ch. In "$(\tilde{s})\alpha$", "$(\tilde{s})$" act as a binder, binding the occurrences of "$\tilde{s}$" in α. Hence $(\tilde{s})\alpha$ is taken up to α-convertibility. The next assignment $x@A : \mathit{Var}(\theta)$ says:

> *A variable x located at A may store values of type* θ.

Unlike service channels, the same variable (say x) can be located at different participants, so that $x@A$ and $x@B$ are distinct variables. Thus we regard $x@A : \mathit{Var}(\theta)$ as mapping $x@A$ (a pair of a channel and a principal) to its type $\mathit{Var}(\theta)$. The third assignment $X^A : \Delta$ says:

> *When the interaction recurs to* X^A*, it has a session typing* Δ.

Assignment to a term variable becomes necessary when we type recursive behaviour. Each term variable is assigned a unique principal name.

CONVENTION 2. We hereafter assume a service typing defines a function following the above reading. We write $\Gamma(ch)$ for a pair of a principal and a service type assigned to ch; $\Gamma(x@A)$ for a variable type assigned to $x@A$; and $\Gamma(X)$ for a pair of a principal and a session typing assigned to X.

Next. a session typing uses the following primary form of assignment, $\tilde{s}[A,B] : \alpha$, which says:

> *A vector of session channels $\tilde{s}$, all belonging to a same session which is between A and B, has the session type α when seen from the viewpoint of A.*

In the main part of the present study, we assume $A \neq B$ in $\tilde{s}[A,B] : \alpha$ (we shall demand it in our typing). We regard $\tilde{s}[A,B] : \alpha$ as mapping $\tilde{s}$ to a direction $[A,B]$ as well as α. As we shall see later, this is the same thing as mapping $\tilde{s}$ to the reverse direction $[B,A]$ and the dual $\overline{\alpha}$ of α. We stipulate:

CONVENTION 3. We hereafter assume a session typing defines a function following the above reading. We write $\Delta(\tilde{s})$ for a pair of $[A,B]$ and a session type α. Similarly we assume that the domain of Δ is disjoint, i.e. whenever $\tilde{s_1}, \tilde{s_2} \in \mathsf{dom}(\Delta)$ s.t. $\tilde{s_1} \neq \tilde{s_2}$, we have $\{\tilde{s_1}\} \cap \{\tilde{s_2}\} = \emptyset$.

As an example of a session typing, given the following interaction:

(48) $$\mathsf{Buyer} \to \mathsf{Seller} : s_1\langle \mathsf{Req},\ prd,\ prd\rangle . \mathsf{Seller} \to \mathsf{Buyer} : s_2\langle \mathsf{Rep},\ price,\ price\rangle . \mathbf{0}$$

one possible assignment is:

(49) $$s_1 s_2\ [\mathsf{Buyer}, \mathsf{Seller}]\ :\ s_1 \downarrow \mathsf{Req}(\mathtt{string}).\ s_2 \uparrow \mathsf{Rep}(\mathtt{int}).\ \mathtt{end}$$

which states, simultaneously:

(1) s_1 and s_2 belong to a same session;
(2) that session is between Buyer and Seller: and
(3) it has the given session type when seen from Buyer's point of view.

The other form of assignment, $\tilde{s} : \perp$, is used when we know the session type at $\tilde{s}$ will never be abstracted by session initiation (this is known for sure when one or more of channels in $\tilde{s}$ are hidden, see (G-TRES1,2) later).

We are ready to introduce typing rules. We start from the typing of session initiation.

$$(\text{G-TINIT})\ \frac{\Gamma,\ ch@B : (\tilde{s})\alpha \vdash I \rhd \Delta \cdot \tilde{s}[B,A] : \alpha \quad A \neq B}{\Gamma,\ ch@B : (\tilde{s})\alpha \vdash A \to B : ch(\tilde{s})\,.\,I \rhd \Delta}$$

The conclusion (the lower-half) introduces a session initialisation prefix $ch(\boldsymbol{\nu}\,\tilde{s})$ in the term. Since $\tilde{s}$ is to be abstracted as session channels belonging to a single session, we demand that, in the premise (the upper-half), that there is a session type assignment which assigns $\tilde{s}$ to a session type. Since $\tilde{s}$ is directed from B to A, α designates a session type seen from B's viewpoint: hence we can safely have $ch@B : (\tilde{s})\alpha$ in the service typing of the conclusion ($ch@B : (\tilde{s})\alpha$ is also assumed in the premise since ch may have already been used in I, based on the assumption that a service channel can be shared, just as the standard URL). Both A and B need be mentioned in $\tilde{s}[B,A] : \alpha$ in the premise since a session is always between two parties: however the information on A should be erased in $ch@B : (\tilde{s})$ since ch can be used by multiple users (participants).

We next type communication.

$$(\text{G-TCOMM})\ \frac{\Gamma \vdash I \rhd \Delta \cdot \tilde{s}[A,B] : \alpha_j \quad \Gamma \vdash e@A : \theta_j \quad \Gamma \vdash x@B : \theta_j \quad s \in \{\tilde{s}\} \quad j \in J \quad A \neq B}{\Gamma \vdash A \to B : s\langle \mathsf{op_j},\ e,\ x\rangle\,.\,I \rhd \Delta \cdot \tilde{s}[A,B] : \Sigma_{j \in J} s \uparrow op_j(\theta_j)\,.\,\alpha_j}.$$

In (G-TCOMM), our purpose is to type the term in the conclusion,

(50) $$A \to B : s\langle \mathsf{op_j},\ e_j,\ x_j\rangle\,.\,I_j.$$

To type this, I should contain a session type between A and B such that its session channels contain s. This session type, α_j, is to be combined with the type for the communication in the conclusion. The remaining session types in Θ will remain unchanged. The communicated value e is typed in the source (A) while the variable x is typed in the target (B), with the same type θ. In the conclusion, we use a branching type which should include the operator op_i whose value type is θ_i, and possibly (and usually) adding other operator names and communicated value types. The rule uses the an auxiliary judgement: $\Gamma \vdash\ e@A : \theta$ which says e at A has type θ (the judgement is derived in the standard way, starting from $\Gamma \vdash x@A : \theta$ when $x@A : \mathit{Var}(\theta)$ is in Γ, and e.g. $\Gamma \vdash 1@A : \mathtt{int}$ regardless of Γ and A).

In $(\text{G-TC}\textsc{omm})$, the session type in focus is given with the direction from A to B, i.e. it abstracts the structure of the interaction in this session from the viewpoint of A. While this is consistent, there is no reason we should view this session from the viewpoint of A: we may as well regard it from the viewpoint of a receiver, B. Thus we have the following symmetric variant of $(\text{G-TC}\textsc{omm})$.

$$(\text{G-TC}\textsc{ommI}\textsc{nv})\ \frac{\Gamma \vdash I \rhd \Delta \cdot \tilde{s}[B,A] : \alpha_j \quad \Gamma \vdash e@A : \theta_j \quad \Gamma \vdash x@B : \theta_j \quad s \in \{\tilde{s}\} \quad j \in J \quad A \neq B}{\Gamma \vdash A \to B : s\langle \mathsf{op_j},\ e,\ x\rangle\,.I \rhd \Delta \cdot \tilde{s}[B,A] : \Sigma_{j\in J} s \downarrow op_j(\theta_j)\,.\overline{\alpha_j}}.$$

The typing of the assignment rule follows.

$$(\text{G-TA}\textsc{ssign})\ \frac{\Gamma \vdash x@A : \theta \quad \Gamma \vdash e@A : \theta \quad \Gamma \vdash I \rhd \Delta}{\Gamma \vdash x := e@A\,.I \rhd \Delta}$$

In this rule, there is no change in the session typing Δ (as well as in the service typing Γ) since assignment does not change interaction structure. Note the types of x and e are taken at location A. For conditionals:

$$(\text{G-TI}\textsc{f})\ \frac{\Gamma \vdash e@A : \mathtt{bool} \quad \Gamma \vdash I_1 \rhd \Delta \quad \Gamma \vdash I_2 \rhd \Delta}{\Gamma \vdash \mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2 \rhd \Delta}$$

In the premise of this rule, we demand I_1 and I_2 has an identical session typing and an identical service typing. Thus either branch is taken, the type abstraction remains identical. Similarly we type the summation:

$$(\text{G-TS}\textsc{um})\ \frac{\Gamma \vdash I_1 \rhd \Delta \quad \Gamma \vdash I_2 \rhd \Delta}{\Gamma \vdash I_1 + I_2 \rhd \Delta}$$

The following rule is worth presenting, which is derivable by applying $(\text{G-TC}\textsc{omm})$ and $(\text{G-TS}\textsc{um})$ repeatedly. We use the notation for the n-fold summation.

$$(\text{G-TB}\textsc{ra})\ \frac{\Gamma \vdash I_j \rhd \Delta \cdot \tilde{s}[A,B] : \alpha_j \quad \Gamma \vdash e_j@A : \theta_j \quad \Gamma \vdash x_j@B : \theta_j \quad s \in \{\tilde{s}\} \quad J \subseteq K}{\Gamma \vdash \Sigma_{j\in J} A \to B : s\langle \mathsf{op_j},\ e_j,\ x_j\rangle\,.I_j \rhd \Delta \cdot \tilde{s}[A,B] : \Sigma_{k\in K} s_k \uparrow op_k(\theta_k)\,.\alpha_k}.$$

In the premise of $(\text{G-TB}\textsc{ra})$, it is implicit from indice symbols that j ranges over J. To type $\Sigma_{j\in J} A \to B : s\langle \mathsf{op_j},\ e_j,\ x_j\rangle\,.I_j$, each I_j needs to have a session type between A and B such that its session channels contain s. Each of these session types, α_j, can be distinct, but they should have the same vector of session channels, so that we can combine them into a single type in the conclusion. Other session types (Θ) should remain common in all branches in the premise. In the conclusion, we can combine session types for different branches into a single session type, adding operator names and communicated value types. Note the value type of e_j is typed in the source (A) while the variable x_j is typed in the target (B). Note $(\text{G-TC}\textsc{omm})$ is a special case of $(\text{G-TB}\textsc{ra})$, when the n-fold branching is a singleton. The rule has its obvious dual, corresponding to $(\text{G-TC}\textsc{ommI}\textsc{nv})$, which we omit.

To type parallel composition, we use the standard linearity condition. The the operator $\bullet$ is well-defined whenever the linearity condition is satisfied and is such that $\tilde{s}[A,B] : \alpha \in \Delta_1 \bullet \Delta_2$ iff either

(1) $\tilde{s}[A,B] : \alpha_1 \in \Delta_1$, $\{\tilde{s}\}[A,B] : \alpha_2 \in \Delta_2$ and $\alpha = \alpha_1 \mid \alpha_2$;
(2) $\tilde{s}[A,B] : \alpha \in \Delta_1$ and $\{\tilde{s}\} \cap \mathsf{fsc}(\Delta_2) = \emptyset$, or its symmetric case;

The typing rule follows.

$$(\text{G-TP}\textsc{ar})\ \frac{\Gamma \vdash I_1 \rhd \Delta_1 \quad \Gamma \vdash I_2 \rhd \Delta_2}{\Gamma \vdash I_1 \mid I_2 \rhd \Delta_1 \bullet \Delta_2}$$

For restriction we have three rules. The first one is when we first hide a session channel in a session type assignment.

$$(\text{G-TR}\textsc{es}1)\ \frac{\Gamma \vdash I \rhd \Delta,\ \tilde{s}_1 s \tilde{s}_2[A,B] : \alpha}{\Gamma \vdash (\nu s)\ I \rhd \Delta,\ \tilde{s}_1 \tilde{s}_2 : \bot}$$

To understand the rule, note the hiding is introduced after the session initiation takes place (see $(\text{G-I}\textsc{nit})$ in Section 9.2). Once this is done, there is no possibility that these session channels are abstracted by (G-TINIT). Hence the session type α is no longer necessary, so that we replace it with $\perp$. After this, we take off a hidden session channel one by one, and when this is empty, takes it away (below ε denotes the empty vector):

$$(\text{G-TRES2})\ \frac{\Gamma \vdash I \rhd \Delta,\ \tilde{s}_1 s \tilde{s}_2 : \perp}{\Gamma \vdash (\boldsymbol{\nu} s)\ I \rhd \Delta,\ \tilde{s}_1\tilde{s}_2 : \perp} \qquad (\text{G-TRES3})\ \frac{\Gamma \vdash I \rhd \Delta,\ \varepsilon : \perp}{\Gamma \vdash (\boldsymbol{\nu} s)\ I \rhd \Delta}$$

Next we treat the typing rule for a term variable.

$$(\text{G-TVAR})\ \frac{}{\Gamma,\ X^A : \Delta \vdash X^A \rhd \Delta}$$

This is one of the two base cases (another is the inaction treated below), introducing a service typing on the left-hand side of the turnstile. This typing should follow Conventions above, and, moreover, it should contain the assignment for the term variable of interest (above X), with the same participant annotation (above A). Following $X^A : \Delta$ we introduce Δ as the session typing of X^A in the conclusion. The recursion rule is symmetric to (G-TVAR):

$$(\text{G-TREC})\ \frac{\Gamma \cdot X^A : \Delta \vdash I \rhd \Delta}{\Gamma \vdash \textbf{rec}\ X^A . I \rhd \Delta}$$

Here our purpose is to type **rec** $X^A . I$, with the session typing Δ. For this purpose it suffices that I has session typing Δ under the assumption X^A has that same session typing, following the standard treatment of recursion. Finally the typing rule for the inaction follows.

$$(\text{G-TZERO})\ \frac{\forall i \neq j.\ \{\tilde{s}_i\} \cap \{\tilde{s}_j\} = \emptyset}{\Gamma \vdash 0 \rhd \bigcup_i \tilde{s}_i[A_i, B_i]\texttt{end}}$$

The rule assigns pairwise disjoint vectors the empty action types. The typing rules are summarised in Figure 19.

10.4. Properties of Type Discipline. We discuss basic properties of the typing system. Below we write e.g. $\Gamma \cdot \Gamma'$ etc. to indicate a disjoint union.

PROPOSITION 1.

(1) (weakening) $\Gamma \vdash I \rhd \Delta$ *implies* $\Gamma \cdot \Gamma' \vdash I \rhd \Delta$. *With* $\tilde{s}$ *be fresh.* $\Gamma \vdash I \rhd \Delta$ *implies* $\Gamma \vdash I \rhd \Delta \cdot \tilde{s}[A,B]\texttt{end}$.
(2) (thinning) *Assume* $\mathsf{fc}(\Gamma') \cap \mathsf{fc}(I) = \emptyset$. *Then* $\Gamma \cdot \Gamma' \vdash I \rhd \Delta$ *implies* $\Gamma \vdash I \rhd \Delta$.
(3) (co-type) $\Gamma \vdash I \rhd \Delta \cdot \tilde{s}[A,B]\alpha$ *implies* $\Gamma \vdash I \rhd \Delta \cdot \tilde{s}[B,A]\overline{\alpha}$.

Proof. Standard. For (co-type), note this information is explicitly generated only in (G-TINIT) and (G-TCOM), which show (by induction hypothesis) such inference is possible. □

Next, we observe free session channels can be grouped in an arbitrary way in the typing rules.

PROPOSITION 2 (grouping of session names).

(1) *Suppose* $\Gamma \vdash I \rhd \Delta,\ \tilde{s}_1\tilde{s}_2[A,B] : \beta$. *Then there are always* $\alpha_{1,2}$ *such that* $\Gamma \vdash I \rhd \Delta,\ \tilde{s}_1[A,B] : \alpha_1,\ \tilde{s}_2[A,B] : \alpha_2$.
(2) *Suppose* $\Gamma \vdash I \rhd \Delta,\ \tilde{s}_1[A,B] : \alpha_1,\ \tilde{s}_2[A,B] : \alpha_2$. *Then there is always some* β *such that* $\Gamma \vdash I \rhd \Delta,\ \tilde{s}_1\tilde{s}_2[A,B] : \beta$.

Proof. (1) is easy by induction on typing rules (for (G-TPAR) using the disjointness in the premise; for (G-TCOM) and (G-TCOMINV) we construct the assignment for $\tilde{s}_1$ or $\tilde{s}_2$ depending on to which s belongs to). (2) is also easy by induction (for (G-TRES1/2/3) we observe $\alpha_i \neq \perp$ by the shape of typing, even though we can in fact treat the assignment of the form $\tilde{s} : \perp$). □

Proposition 2 says the grouping of free session channels in the session typing is in fact arbitrary: they become relevant only when they are abstracted by (G-TINIT).

The type discipline has a minimal typing, which is closely related with automatic type inference a la ML. To formulate minimality, we use the following ordering. Below *closed types* are those types without free type variables.

$$(\textsc{G-TInit})\ \frac{\Gamma,\ ch@B:(\tilde{s})\alpha \vdash I \rhd \Delta\cdot\tilde{s}[B,A]:\alpha \quad A \neq B}{\Gamma,\ ch@B:(\tilde{s})\alpha \vdash A \to B : ch(\boldsymbol{\nu}\,\tilde{s})\,.I \rhd \Delta}$$

$$(\textsc{G-TCom})\ \frac{\Gamma \vdash I \rhd \Delta\cdot\tilde{s}[A,B]:\alpha_j \quad \Gamma \vdash e@A:\theta_j \quad \Gamma \vdash x@B:\theta_j \quad s \in \{\tilde{s}\} \quad j \in J \quad A \neq B}{\Gamma \ \vdash\ A \to B : s\langle \mathsf{op_j},\ e,\ x\rangle\,.I\ \rhd\ \Delta\cdot\tilde{s}[A,B]:s\uparrow\Sigma_{i\in J}\mathsf{op_i}(\theta_i).\ \alpha_i}$$

$$(\textsc{G-TComInv})\ \frac{\Gamma \vdash I \rhd \Delta\cdot\tilde{s}[B,A]:\alpha_j \quad \Gamma \vdash e@A:\theta_j \quad \Gamma \vdash x@B:\theta_j \quad s \in \{\tilde{s}\} \quad j \in J \quad A \neq B}{\Gamma \ \vdash\ A \to B : s\langle \mathsf{op_j},\ e,\ x\rangle\,.I\ \rhd\ \Delta\cdot\tilde{s}[B,A]:s\downarrow\Sigma_{i\in J}\mathsf{op_i}(\theta_i).\ \alpha_i}$$

$$(\textsc{G-TAsgn})\ \frac{\Gamma \vdash x@A:\theta \quad \Gamma \vdash e@A:\theta \quad \Gamma \vdash I \rhd \Delta}{\Gamma \vdash x := e@A\,.I \rhd \Delta}$$

$$(\textsc{G-TSum})\ \frac{\Gamma \vdash I_1 \rhd \Delta \quad \Gamma \vdash I_2 \rhd \Delta}{\Gamma \vdash I_1 + I_2 \rhd \Delta} \qquad (\textsc{G-TIf})\ \frac{\Gamma \vdash e@A:\texttt{bool} \quad \Gamma \vdash I_1 \rhd \Delta \quad \Gamma \vdash I_2 \rhd \Delta}{\Gamma \vdash \mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2 \rhd \Delta}$$

$$(\textsc{G-TPar})\ \frac{\Gamma \vdash I_1 \rhd \Delta_1 \quad \Gamma \vdash I_2 \rhd \Delta_2}{\Gamma \vdash I_1 \mid I_2 \rhd \Delta_1 \bullet \Delta_2} \qquad (\textsc{G-TRes1})\ \frac{\Gamma \vdash I \rhd \Delta,\ \tilde{s}_1 s \tilde{s}_2[A,B]:\alpha}{\Gamma \vdash (\boldsymbol{\nu}\,s)\ I \rhd \Delta,\ \tilde{s}_1\tilde{s}_2:\bot}$$

$$(\textsc{G-TRes2})\ \frac{\Gamma \vdash I \rhd \Delta,\ \tilde{s}_1 s \tilde{s}_2:\bot}{\Gamma \vdash (\boldsymbol{\nu}\,s)\ I \rhd \Delta,\ \tilde{s}_1\tilde{s}_2:\bot} \qquad (\textsc{G-TRes3})\ \frac{\Gamma \vdash I \rhd \Delta,\ \varepsilon:\bot}{\Gamma \vdash (\boldsymbol{\nu}\,s)\ I \rhd \Delta}$$

$$(\textsc{G-TRec})\ \frac{\Gamma\cdot X:\Delta \vdash I \rhd \Delta}{\Gamma \vdash \mathbf{rec}\,X\,.I \rhd \Delta} \qquad (\textsc{G-TVar})\ \frac{\Gamma,\ X:\Delta\ \text{well-formed}}{\Gamma,\ X:\Delta \vdash X \rhd \Delta}$$

$$(\textsc{G-TZero})\ \frac{\Gamma\ \text{well-formed} \quad \forall i \neq j.\ \{\tilde{s}_i\} \cap \{\tilde{s}_j\} = \emptyset}{\Gamma \vdash \mathbf{0} \rhd \bigcup_i \tilde{s}_i[A_i,B_i]\texttt{end}}$$

FIGURE 19. Typing Rules for Global Calculus

DEFINITION 1 (inclusion ordering). *The* inclusion ordering $\Subset$ *is the relation over closed types given as:* $\alpha \Subset \beta$ *iff* $\alpha\mathcal{R}\beta$ *for some* $\mathcal{R}$ *satisfying the following rules, where each rule says that: for the conclusion to be satisfied, the condition(s) in the premise should be satisfied.*

$$(\textsc{Inc-In}) \qquad \frac{\beta\ \approx\ \Sigma_{i\in J'} s\downarrow op_i(\theta_i)\,.\alpha'_i J \subset J' \quad \forall i \in J.\ \alpha_i \Subset \alpha'_i}{\Sigma_{i\in J} s\downarrow op_i(\theta_i)\,.\alpha_i\ \Subset\ \beta}$$

$$(\textsc{Inc-Out}) \qquad \frac{\beta\ \approx\ \Sigma_{i\in J'} s\uparrow op_i(\theta_i)\,.\alpha'_i J \subset J' \quad \forall i \in J.\ \alpha_i \Subset \alpha'_i}{\Sigma_{i\in J} s\uparrow op_i(\theta_i)\,.\alpha_i\ \Subset\ \beta}$$

$$(\textsc{Inc-Par}) \qquad \frac{\beta\ \approx\ \alpha'_1|\alpha'_2 \quad \alpha_1 \Subset \alpha'_1 \quad \alpha_2 \Subset \alpha'_2}{\alpha_1|\alpha_2\ \Subset\ \beta}$$

$$(\textsc{Inc-End}) \qquad \frac{\beta \approx \mathbf{0}}{\texttt{end} \Subset \beta}$$

$$(\textsc{Inc-Iso}) \qquad \frac{\alpha\ \Subset\ \beta}{\alpha'\ \Subset\ \beta}(\alpha \approx \alpha')$$

where $\approx$ *is the standard tree isomorphism on recursive types. A relation* $\mathcal{R}$ *satisfying these conditions is called a* witness *of* $\Subset$*. It is easy to see that* $\Subset$ *itself is a witness of itself, defining the largest such. We then extend* $\Subset$ *to well-formed session typings by:*

$$\frac{\Delta \subset \Delta'}{\Delta \Subset \Delta'} \qquad \frac{\Delta \Subset \Delta' \quad \alpha \Subset \alpha'}{\Delta\cdot\tilde{s}[A,B]:\alpha\ \Subset\ \Delta'\cdot\tilde{s}[A,B]:\alpha'}$$

and to the well-formed service typings by:

$$\frac{\Gamma \subset \Gamma'}{\Gamma \Subset \Gamma'} \qquad \frac{\Gamma \Subset \Gamma' \quad \alpha \Subset \alpha'}{\Gamma \cdot ch@A : (\tilde{s})\alpha \;\Subset\; \Gamma' \cdot ch@A : (\tilde{s})\alpha'}$$

We also call these extended relations, inclusion ordering.

In brief, $\alpha \Subset \alpha'$ means that α is the result of cutting off some branches (including both input and output) from α' at zero or more points. This ordering is different from the standard subtyping on session types [**19**] we shall introduce in Section 12. The inclusion ordering is also used in our technical development in Section 16. We observe:

PROPOSITION 3. *The relation $\Subset$ is a partial order on types modulo $\approx$.*

Proof. Standard [**19**], noting the coinductive definition of $\Subset$ in Definition 1 means that $\Subset$ (or its witness) can be regarded as a simulation on deterministic trees. See also the corresponding proof for Proposition 6, §12.1, page 58, on subtyping on session types. □

CONVENTION 4. *Henceforth we always take $\Subset$ as the relation on types modulo $\approx$.*

DEFINITION 2. We write $\alpha \sqcup \beta$ for the least upper bound of α and β if it exists.

PROPOSITION 4. *If $\alpha_{1,2}$ has an upper-bound then they have the least upper bound.*

Proof. If the assumption holds, then we can superimpose two (possibly infinite) trees, for which we can take the union of branches from $\alpha_{1,2}$ at each node, which is also a regular tree and which is surely smaller than any such upper-bound. □

We now establish the existence of minimal typing (which is closely related with the preceding works on principal types for recursively typed session types [**24**], cf. [**16, 26**]). Below in (2) we write $\Gamma \vdash I$ for $\Gamma \vdash I \rhd \emptyset$.

PROPOSITION 5.

(1) (subsumption) *Let $\Gamma \Subset \Gamma'$ and $\Delta \Subset \Delta'$. Then $\Gamma \vdash I \rhd \Delta$ implies $\Gamma' \vdash I \rhd \Delta'$.*
(2) (existence of minimal typing) *Let $\Gamma \vdash I$ for some Γ. Then there exists Γ_0 such that $\Gamma_0 \vdash I$ and whenever $\Gamma' \vdash I$ we have $\Gamma_0 \Subset \Gamma'$. Moreover such Γ_0 can be algorithmically calculable from I. We call Γ_0 the* minimum service typing *of I.*

Proof. (1) is because the rules which use this information, (G-TCOM) and (G-TCOMINV), only demands the communication operation the term uses is included in the type. Thus making the type include more operations is always safe. For (2), we prove the following stronger result:

Claim. Assume:

(a) The free session names in I are grouped into disjoint vectors, each of which, say $\tilde{s}$, together with an ordered pair of involved participants, say (A,B), so that it can be used as e.g. $\tilde{s}[A,B] : \alpha$. Write ψ for the set of the mutually disjoint vectors corresponding to such a grouping (cf. Proposition 2).
(b) Let I be typable with a session typing which conforms to ψ (i.e. its type assignments are done using the disjoint vectors in ψ).
(c) $\Subset'$ extends $\Subset$ by the rule $\alpha \Subset' \mathbf{t}$ for each α and $\mathbf{t}$.

Then we have $\Gamma_0 \vdash I \rhd \Delta_0$ such that (1) Δ_0 conforms to ψ; and (2) Whenever $\Gamma \vdash I \rhd \Delta$ such that Δe conforms to ψ, we have $\Gamma_0 \Subset' \Gamma$ and $\Delta_0 \Subset' \Delta$. In particular, if a term variable does not occur in I, then $\Subset'$ can be replaced by $\Subset$.

The proof is by induction on the minimal typing rules given in Figure 20, in comparison with the original typing rules in Figure 19. The former inductively constructs the typings which are minimal w.r.t. the typing given by the latter. In Figure 20, it is assumed that the subject term (the term to be type checked) is already type safe. In the figure we write $\mathsf{tvar}(\alpha)$ for the set of type variables in

$$(\text{G-MTI}{\small\text{NIT}}\text{-1})\ \frac{\Gamma \vdash_{\mathsf{min}} \mathcal{A} \rhd \Delta \cdot \tilde{s}[B,A] : \beta \quad ch \notin \mathsf{fn}(\Gamma) \quad \mathsf{tvar}(\beta) = \{\mathbf{t}\}}{\Gamma,\ ch@B : (\tilde{s})(\beta[\mathsf{end}/\mathbf{t}]) \vdash_{\mathsf{min}} A^{\tau_1} \rightarrow B^{\tau_1} : ch(\boldsymbol{\nu}\,\tilde{s})\,.\,\mathcal{A} \rhd \Delta}$$

$$(\text{G-MTI}{\small\text{NIT}}\text{-2})\ \frac{\Gamma,\ ch@B : (\tilde{s})\alpha \vdash_{\mathsf{min}} \mathcal{A} \rhd \Delta \cdot \tilde{s}[B,A] : \beta \quad \mathsf{tvar}(\beta) = \{\mathbf{t}\}}{\Gamma,\ ch@B : (\tilde{s})(\alpha \sqcup \beta[\mathsf{end}/\mathbf{t}]) \vdash_{\mathsf{min}} A^{\tau_1} \rightarrow B^{\tau_1} : ch(\boldsymbol{\nu}\,\tilde{s})\,.\,\mathcal{A} \rhd \Delta}$$

$$(\text{G-MTC}{\small\text{OMM}})\ \frac{\Gamma \vdash_{\mathsf{min}} \mathcal{A} \rhd \Delta \cdot \tilde{s}[A,B] : \alpha_j \quad \Gamma \vdash e@A : \theta_j \quad \Gamma \vdash x@B : \theta_j \quad s \in \{\tilde{s}\} \quad j \in J}{\Gamma \vdash_{\mathsf{min}} A^{\tau_1} \rightarrow B^{\tau_2} : s\langle \mathsf{op_j},\ e,\ x\rangle\,.\,\mathcal{A} \rhd \Delta \cdot \tilde{s}[A,B] : \Sigma_{j \in \{j\}} s \uparrow op_j(\theta_j)\,.\,\alpha_j}$$

$$(\text{G-MTC}{\small\text{OM}}\text{I}{\small\text{NV}})\ \frac{\Gamma \vdash_{\mathsf{min}} I \rhd \Delta \cdot \tilde{s}[B,A] : \alpha_j \quad \Gamma \vdash e@A : \theta_j \quad \Gamma \vdash x@B : \theta_j \quad s \in \{\tilde{s}\} \quad j \in J}{\Gamma\ \vdash\ A \rightarrow B : s\langle \mathsf{op_j},\ e,\ x\rangle\,.\,I\ \rhd\ \Delta \cdot \tilde{s}[B,A] : s \downarrow \Sigma_{i \in \{j\}} \mathsf{op_i}(\theta_i).\ \alpha_i}$$

$$(\text{G-MTA}{\small\text{SGN}})\ \frac{\Gamma \vdash_{\mathsf{min}} x@A : \theta \quad \Gamma \vdash_{\mathsf{min}} e@A : \theta \quad \Gamma \vdash_{\mathsf{min}} I \rhd \Delta}{\Gamma \vdash_{\mathsf{min}} x := e@A\,.\,I \rhd \Delta}$$

$$(\text{G-MTS}{\small\text{UM}})\ \frac{\Gamma_1 \vdash_{\mathsf{min}} I_1 \rhd \Delta_1 \quad \Gamma_2 \vdash_{\mathsf{min}} I_2 \rhd \Delta_2}{\Gamma_1 \sqcup \Gamma_2 \vdash_{\mathsf{min}} I_1 + I_2 \rhd \Delta_1 \sqcup \Delta_2}$$

$$(\text{G-MTI}{\small\text{F}})\ \frac{\Gamma \vdash_{\mathsf{min}} e@A : \mathtt{bool} \quad \Gamma \vdash_{\mathsf{min}} I_1 \rhd \Delta_1 \quad \Gamma \vdash_{\mathsf{min}} I_2 \rhd \Delta_2}{\Gamma \vdash_{\mathsf{min}} \mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2 \rhd \Delta_1 \sqcup \Delta_2}$$

$$(\text{G-MTP}{\small\text{AR}})\ \frac{\Gamma_1 \vdash_{\mathsf{min}} I_1 \rhd \Delta_1 \quad \Gamma_2 \vdash_{\mathsf{min}} I_2 \rhd \Delta_2}{\Gamma_1 \sqcup \Gamma_2 \vdash_{\mathsf{min}} I_1 \mid^{\tau'} I_2 \rhd \Delta_1 \bullet \Delta_2}$$

$$(\text{G-MTR}{\small\text{ES}}\text{1})\ \frac{\Gamma \vdash_{\mathsf{min}} I \rhd \Delta,\ \tilde{s}_1 s \tilde{s}_2[A,B] : \alpha}{\Gamma \vdash_{\mathsf{min}} (\boldsymbol{\nu}\,s)\ I \rhd \Delta,\ \tilde{s}_1\tilde{s}_2 : \bot}$$

$$(\text{G-MTR}{\small\text{ES}}\text{2})\ \frac{\Gamma \vdash_{\mathsf{min}} I \rhd \Delta,\ \tilde{s}_1 s \tilde{s}_2 : \bot}{\Gamma \vdash_{\mathsf{min}} (\boldsymbol{\nu}\,s)\ I \rhd \Delta,\ \tilde{s}_1\tilde{s}_2 : \bot} \qquad (\text{G-MTR}{\small\text{ES}}\text{3})\ \frac{\Gamma \vdash_{\mathsf{min}} I \rhd \Delta,\ \varepsilon : \bot}{\Gamma \vdash_{\mathsf{min}} (\boldsymbol{\nu}\,s)\ I \rhd \Delta}$$

$$(\text{G-MTV}{\small\text{AR}})\ \frac{\forall i.\tilde{s}_i \in \psi' \quad \text{for appropriate } \psi' \text{ s.t. } \psi' \supset \psi \quad \mathbf{t} \text{ fresh}}{X^A : \mathbf{t} \vdash_{\mathsf{min}} X^A \rhd \bigcup_i \tilde{s}_i[A_i,B_i] : \mathbf{t}}$$

$$(\text{G-MTR}{\small\text{EC}})\ \frac{\Gamma \cdot X^A : \mathbf{t} \vdash_{\mathsf{min}} I \rhd \Delta}{\Gamma \vdash_{\mathsf{min}} \mathbf{rec}\ X^A\,.\,I \rhd \mathtt{solve}(\mathbf{t},\Delta)}$$

$$(\text{G-MTZ}{\small\text{ERO}})\ \frac{\forall i.\tilde{s}_i \in \psi' \quad \text{for appropriate } \psi' \text{ s.t. } \psi' \supset \psi}{\Gamma \vdash_{\mathsf{min}} 0 \rhd \bigcup_i \tilde{s}_i[A_i,B_i]\mathtt{end}}$$

FIGURE 20. Minimal Typing Rules for Global Calculus

α. We need some care in calculating minimal typing in the presence of recursive types, for which we follow the preceding studies [**18**] (cf. [**16, 26**]). In Figure 20, this is treated in (G-MTVAR) and (G-MTREC), as well as in (G-MTINIT-1). In (G-MTVAR), when we introduce a term variable, we also introduce a type variable. ψ' in the premise refers to the disjoint vectors which extends ψ so that, after abstracting session channels by initialisation, the result is precisely ψ mentioned in **(a)** above. In (G-MTREC), $\mathtt{solve}(\mathbf{t},\Delta)$ essentially solves the equations $\mathbf{t} = \tilde{s}_i[A_i,B_i] : \alpha_i$ for each $\tilde{s}_i[A_i,B_i] : \alpha_i \in \Delta$. When $\mathbf{t}$ does appear in $\tilde{s}_i[A_i,B_i] : \alpha_i$, the corresponding component is $\tilde{s}_i[A_i,B_i] : \mathbf{rec}\ \mathbf{t}.\alpha_i$. If not, then we get $\tilde{s}_i[A_i,B_i] : \alpha_i[\mathtt{end}/\mathbf{t}]$ where $\alpha_i[\mathtt{end}/\mathbf{t}]$ is the result of substituting $\mathtt{end}$ for each (free, by bound name convention) occurrence of $\mathbf{t}$. Note this is simpler than the constraint solving in [**16, 26**]: this is because there is only positive occurrences of type variables. Finally in (G-MTINIT-1), when type variables occur in β, we simply replace them with end, where $\beta[\mathtt{end}/\tilde{\mathbf{t}}]$ is the result of substituting end for each (free, by bound name convention) occurrence of type variables from $\tilde{\mathbf{t}}$. This substitution is sound when we already know the target term is well-typed (note that, in the original rule (G-TINIT), α and β can never include a free type variable). (G-MTCOMM) and (G-MTCOMINV) are chosen appropriately for its use in abstraction. For free session channels, we (arbitrarily) fix which direction (say from A to B) to use. In the rules (G-MTINIT), (G-MTSUM) and (G-MTIF), the merge operator $\sqcup$ is used, for which we use the following observation:

Claim. $\alpha_{1,2} \Subset \beta$ implies $\alpha_1 \sqcup \alpha_2 \Subset \beta$.

Using this Claim, the cases for these three rules are immediate. The remaining rules are also direct from the shape of each pair of the corresponding rules. □

Next we turn to subject reduction. we start from basic lemmas. Below and henceforth we write $\Gamma \vdash \sigma$ when the typing of σ conforms to Γ.

LEMMA 1.

(1) (substitution, 1) *If* $\Gamma, X^A : \Delta \vdash I \rhd \Delta'$ *and* $\Gamma \vdash I' \rhd \Delta$ *then* $\Gamma \vdash I[I'/X^A] \rhd \Delta'$.
(2) (substitution, 2) *If* $\Gamma \vdash \sigma$, $\Gamma \vdash \sigma(x@A) : \theta$ *and* $\Gamma \vdash v : \theta$, *then* $\Gamma \vdash \sigma[x@A \mapsto v]$.

Proof. See Appendix A. □

The subject reduction in the present typing does *not* preserve session typing, for the obvious reason: if I reduces to I', it may lose the initial part of interactions, which, if it is at a session channel, will demand truncation of the corresponding session typing.

THEOREM 1.

(1) (Subject Congruence) *If* $\Gamma \vdash I \rhd \Delta$ *and* $I \equiv I'$ *then* $\Gamma \vdash I' \rhd \Delta$ *(up to alpha-renaming).*
(2) (Subject Reduction, 1) *Assume* $\Gamma \vdash \sigma$. *Then* $\Gamma \vdash I \rhd \Delta$ *and* $(\sigma, I) \to (\sigma', I')$ *imply* $\Gamma \vdash \sigma'$ *and* $\Gamma \vdash I \rhd \Delta'$ *for some* Δ'.
(3) (Subject Reduction, 2) *Assume* $\Gamma \vdash \sigma$. *Then* $\Gamma \vdash I$ *and* $(\sigma, I) \to (\sigma', I')$ *imply* $\Gamma \vdash \sigma'$ *and* $\Gamma \vdash I'$.

Proof. See Appendix A. □

10.5. Examples of Typing.

EXAMPLE 11. We conclude the section, by showing how it is possible to type an example: consider the buyer-seller case with the following interaction described in the global calculus.

$$
\begin{array}{l}
\mathtt{Buyer} \to \mathtt{Seller} : \mathtt{B2SCh}(s)\,.\,\mathtt{Buyer} \to \mathtt{Seller} : s[\mathtt{RequestForQuote}]\,. \\
\;\mathtt{Seller} \to \mathtt{Buyer} : s\langle \mathtt{QuoteResponse},\, v_{\mathtt{quote}},\, x_{\mathtt{quote}} \rangle\,. \\
\;\;(\mathtt{Buyer} \to \mathtt{Seller} : s[\mathtt{QuoteReject}] + \\
\;\;\;\mathtt{Buyer} \to \mathtt{Seller} : s[\mathtt{QuoteAcceptance}]\,. \\
\;\;\;\;\mathtt{Seller} \to \mathtt{Shipper} : \mathtt{S2ShCh}(s')\,. \\
\;\;\;\;\mathtt{Seller} \to \mathtt{Shipper} : s'\langle \mathtt{RequestDelDetails},\, \mathtt{Buyer},\, x_{\mathtt{Client}} \rangle\,. \\
\;\;\;\;\;\mathtt{Shipper} \to \mathtt{Seller} : s'[\mathtt{DeliveryDetails}, \mathtt{DD}, x_{\mathtt{DD}}]\,. \\
\;\;\;\;\;\;\mathtt{Seller} \to \mathtt{Buyer} : s[\mathtt{DeliveryDetails}, x_{\mathtt{DD}}, x_{\mathtt{DD}}])
\end{array}
$$

Above there are two sessions: the one between the buyer and the seller, and the one between the seller and the shipper. Note that both are initialised by a session "init" operation and we have also included the choice. Another notable thing is that in the last two interactions, the variable $x_{\mathtt{DD}}$ is involved three times: the first two times it is indeed the same variable located at the seller and assigned with the delivery details DD, but the third one is another variable located at the buyer which just happen to have the same name, but completely distinguished by the semantics of mini-CDL. But what are the types for channels B2SCh and S2ShCh? It can be verified by the rules in Appendix that the the interactions above can be typed by $\Delta = \texttt{B2SCh}@\texttt{Seller}(s)[\texttt{Buyer},\texttt{Seller}] : \alpha \cdot \texttt{S2ShCh}@\texttt{Shipper}[\texttt{Seller},\texttt{Shipper}] : \alpha'$ where

$$\begin{aligned}\alpha = & s\uparrow \texttt{RequestForQuote}().s\downarrow \texttt{QuoteResponse}(\texttt{QuoteType}).(s\uparrow \texttt{QuoteReject}()+\\ & s\uparrow \texttt{QuoteAcceptance}().s\downarrow \texttt{OrderConfirmation}().\\ & s\downarrow \texttt{DeliveryDetails}(\texttt{DDType}))\end{aligned}$$

and $\alpha' = s'\uparrow \texttt{RequestDelDetails}(\mathit{PType}).s'\downarrow \texttt{DeliveryDetails}(\texttt{DDType})$.

EXAMPLE 12. *In the last example of this section, we give a typing for 20. We would simply have that* $\Gamma \vdash \mathsf{comm}@\mathsf{Seller}(\mathit{B2Sch},\mathit{Data}) : \mathit{B2Sch} \uparrow (\mathit{String}) \mid \mathit{Data} \uparrow (\mathit{String})$.

11. End-Point Calculus (1): Syntax and Reduction

The end-point calculus, an applied variant of the π-calculus [**34**], specifies local behaviours of end-points and their composition. For example consider the following term in the global calculus (cf. Example 1):

$$\mathsf{Buyer} \to \mathsf{Seller} : s\langle \mathsf{QuoteAccept}, 100, \mathsf{x},\ .,\ \mathbf{0}\rangle. \tag{51}$$

This global description says that Buyer sends a QuoteAccept message with value 100 to Seller, that Seller receives it, and that Seller saves this value in its local variable x. The end-point calculus describes the same situation as combination of local behaviour, located at each end-point. First there is Buyer's behaviour:

$$\mathsf{Buyer}[\, \overline{s} \lhd \mathsf{QuoteAccept}\langle 100\rangle.\mathbf{0}\,]_{\sigma_B} \tag{52}$$

where σ_B is Buyer's local state. Similarly we have Seller's local behaviour:

$$\mathsf{Seller}[\, s \rhd \mathsf{QuoteAccept}\langle x\rangle.\mathbf{0}\,]_{\sigma_S} \tag{53}$$

where σ_S is Seller's local state. Interaction takes place when (52) and (53) are concurrently composed, as follows.

$$\mathsf{Seller}[\, s \rhd \mathsf{QuoteAccept}\langle x\rangle.\mathbf{0}\,]_{\sigma_S} \mid \mathsf{Buyer}[\, \overline{s} \lhd \mathsf{QuoteAccept}\langle 100\rangle.\mathbf{0}\,]_{\sigma_B} \tag{54}$$

Let this term be written M. Then the communication event is represented using the following one-step reduction:

$$M \quad \to \quad \mathsf{Seller}[\, \mathbf{0}\,]_{\sigma_S[x \mapsto 10]} \mid \mathsf{Buyer}[\, \mathbf{0}\,]_{\sigma_B} \tag{55}$$

Note the state at Seller is updated as a result of communication. In correspondence with the global calculus, communication in the end-point calculus is organised in the unit of session, where session initiation is done by communicating fresh channels while ordinary, in-session communication is done via session channels involving operator selection and value passing, as described above. The formal syntax and reduction rules of the end-point calculus are presented in the present section.

Since an input and an output are separately described in the end-point calculus, it is possible that there is a communication mismatch between two interacting parties. For example, instead of (54), we may have:

$$\mathsf{Seller}[\, s \rhd \mathsf{QuoteAccept}\langle x\rangle.\mathbf{0}\,]_{\sigma_S} \mid \mathsf{Buyer}[\, \overline{s} \lhd \mathsf{QuoteReject}.\mathbf{0}\,]_{\sigma_B} \tag{56}$$

Here Seller is expecting a QuoteAccept message with one integer value, while Buyer is sending a nullary QuoteReject message. To avoid such a situation, we use type discipline. We use the same syntax of types as in the global calculus. For example, Seller's interface at s in (56) is represented by the following session type:

$$s@\mathsf{Seller} : s \downarrow \mathsf{QuoteAccept}(\mathtt{int}).\mathtt{end} \tag{57}$$

while that of Buyer is abstracted as:

$$s@\mathsf{Buyer} : s \uparrow \mathsf{QuoteReject}().\mathtt{end} \tag{58}$$

Since two signatures, (57) and (58), are clearly incompatible, we conclude the composition (56) is not well-typed. The session types for the end-point calculus use a notion of subtyping which plays a central role in the theory of end-point projection. The session typing for the end-point calculus and its basic properties are studied in Section 12.

11.1. Formal Syntax. The end-point calculus is an applied form of the π-calculus [**35**] augmented with the notion of participants and their local state (cf.[**3, 15, 21**]). Session initiation uses bound name passing, while in-session communication uses variables at a local store, in the spirit of [**15**]. The following grammar define *processes*, ranged over by $P,Q,R,\ldots$.

$$\begin{array}{rlll}
P & ::= & !ch(\tilde{s}).P & \text{(init-In)} \\
 & | & \overline{ch}(\boldsymbol{\nu}\,\tilde{s}).P & \text{(init-Out)} \\
 & | & s \rhd \Sigma_i \mathsf{op}_i(x_i).P_i & \text{(input)} \\
 & | & \overline{s} \lhd \mathsf{op}\langle e \rangle P & \text{(output)} \\
 & | & x := e.P & \text{(assignment)} \\
 & | & \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q & \text{(conditional)} \\
 & | & P \oplus Q & \text{(internal sum)} \\
 & | & P|Q & \text{(parallel)} \\
 & | & (\boldsymbol{\nu}\,s)\ P & \text{(res)} \\
 & | & X & \text{(variable)} \\
 & | & \mathbf{rec}\ X.P & \text{(recursion)} \\
 & | & \mathbf{0} & \text{(inaction)}
\end{array}$$

As in the global calculus (cf. Section 8), $a,b,ch,\ldots$ above denote service channels, $s,s',\ldots$ session channels, $x,y,\ldots$ variables, and $X,\ldots$ term variables. The symbol "!" in "$!ch(\tilde{s}).P$" (the first line) indicates *replication* [**33**], which says that the input channel (here ch) is available for unbounded number of invocations.

Processes are located in participants. Participants and their composition are called *networks* (written $N,M,\ldots$), whose grammar is given by:

$$\begin{array}{rll}
N ::= & A[P]_\sigma & \text{(participant)} \\
 & |\ N|M & \text{(parallel-nw)} \\
 & |\ (\boldsymbol{\nu}\,s)\ N & \text{(res-nw)} \\
 & |\ \varepsilon & \text{(inaction-nw)}
\end{array}$$

Again as in the global calculus, $A,B,\ldots$ denote *participant names*, which are often simply called *participants*. $\sigma,\ldots$ denote local states, each mapping a finite set of variables to a finite set of values.[6]

11.2. Illustration of Syntax. For session initiation, we use a pair of mutually complementary input and output:

$$!ch(\tilde{s}).P \qquad\qquad \overline{ch}(\boldsymbol{\nu}\,\tilde{s}).Q \tag{59}$$

In the context of web services, the process $!ch(\tilde{s}).P$ may be considered as embodying a repeatedly available service accessible via a certain URL (here denoted by channel ch): after invocation, it offers interaction described in P through session channels $\tilde{s}$ it has just received. The process $\overline{ch}(\boldsymbol{\nu}\,\tilde{s}).Q$ in turn may be considered as an invoker of a service located at ch, which communicate fresh session channels and use them for its subsequent interaction, described in Q. The structure of communication within a session will later be abstracted by a session type. In practice, session initiation may as well be combined with ordinary communication.

In-session communications use operator names, analogous to methods in objects.

$$s \rhd \Sigma_i \mathsf{op}_i(x_i).P_i \qquad\qquad \overline{s} \lhd \mathsf{op}\langle e \rangle.Q \tag{60}$$

The input $s \rhd \Sigma_i \mathsf{op}_i(x_i).P_i$ says that it has one or more finite branches (indexed by i) which can be invoked. Operators op_i should be pairwise distinct. When op_i is invoked, then it instantiates a communicated value in its local variable x_i, and subsequently behaves as described in P_i. Here x_i does

[6] Note the same symbol denotes a distributed state in the global calculus: σ in the end-point calculus corresponds to local projection of such a distributed state.

not bind its occurrences in P. In turn, $\overline{s} \lhd \mathsf{op}\langle e \rangle.Q$ invokes an input with operator op, communicating the result of evaluating an expression e, then behaves as Q.

Another prefix operator is assignment:

(61) $$x := e.P$$

which assigns the result of evaluating e to a local variable x (of the enclosing participant), and then behaves as P.

There are two constructs which represent internal choice. First, the standard conditional is written as:

(62) $$\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q$$

where e should evaluate to a boolean value. In this case, once e is evaluated, we can deterministically choose between P or Q. A more nondeterministic behaviour is embodied by the standard internal sum:

(63) $$P \oplus Q$$

which chooses nondeterministically P or Q and, once chosen, behaves as such. The n-fold composition by $\oplus$ is written $\oplus_i P_i$.

Combining multiple outputs at the same channel but with distinct operator names through the n-fold internal sum, we can construct an output prefix which is dual to the branching input prefix. Since such a sum is often useful, we introduce the following notation for denoting it.

(64) $$s \lhd \Sigma_i \mathsf{op}_i\langle e_i \rangle.P_i$$

Above we assume each op_i is pairwise distinct. Note neither input branching or output branching above have mixed choice, i.e. all the components have to be either outputs or inputs (via a common session channel), but never both. They offer a structured form of choice which is easily implementable, even though some form of mixed choice is useful for representing complex interaction, as we discussed in Part I.

The syntax for parallel composition is standard:

(65) $$P|Q$$

As in the standard process algebras, and unlike parallel composition in the global calculus, P and Q may as well be engaged in communication between them. The restriction also uses the standard syntax:

(66) $$(\boldsymbol{\nu} s)\ P$$

which indicates s is local to P. In the present paper we do not consider restriction of service channels, whose addition does not pose any technical problem.

For representing recursive behaviour, we start from a term variable X and, after forming a process P in which X may occur free, we introduce:

(67) $$\mathbf{rec}\ X.P$$

where, in $\mathbf{rec}\ X.P$, free occurrences of X in P (if any) are bound by $\mathbf{rec}\ X$. Behaviourally, each free X in P denotes a recurring point (i.e. it recurs to $\mathbf{rec}\ X.P$ again). In contrast to the global calculus, cf. Section 8.1, variables need no principal annotation. Finally the inaction:

(68) $$\mathbf{0}$$

denotes the lack of action, and is the unit for parallel composition.

Processes are located in a participant as follows:

(69) $$A[P]_\sigma$$

which says a participant named A is equipped with a behaviour P and a local state σ. Such participants can be combined by parallel composition:

(70) $$N|M$$

By the typing rules discussed in the next section, in one network, two participants never have the same participant names. Since a session channel s can be shared between two participants, we also need restriction:

$$(\boldsymbol{\nu} s)\, N \tag{71}$$

Finally for technical convenience we also introduce the inaction for networks, denoting the lack of network.

$$\varepsilon \tag{72}$$

which acts as the unit of parallel composition of networks.

EXAMPLE 13. Example 1 (page 34) of a global description would be represented in the end-point formalism as a network of the shape:

$$\mathsf{Buyer}[\,\mathsf{P}\,]_{\sigma(\mathsf{Buyer})} \mid \mathsf{Seller}[\,\mathsf{Q}\,]_{\sigma(\mathsf{Seller})} \tag{73}$$

where the processes P and Q together realise the behaviour that we expressed in the global calculus. As for the Buyer, its behaviour would be represented by the following

$$P \;=\; B2SCh \lhd\; (\mathsf{QuoteAccept}\langle``100''\rangle\,.\,P_1 \oplus \mathsf{QuoteReject}\langle x_{\mathsf{AbortNo}}\rangle\,.\,P_2)$$

whereas the Seller would behave as:

$$Q \;=\; B2SCh \rhd\; (\mathsf{QuoteAccept}(x)\,.\,Q_1 + \mathsf{QuoteReject}(y)\,.\,Q_2)$$

We elaborate P and Q in the next example.

EXAMPLE 14. Example 2 (page 34) presents the if-then-else construct, in which case we can elaborate P and Q above as follows.

$$\begin{aligned} P \;&= \mathsf{if}\ (x_{\mathsf{Quote}} \leq 1000)\\ &\quad \mathsf{then}\ B2SCh \lhd\ \mathsf{QuoteAccept}\langle``100''\rangle\,.\,\mathsf{P}_1,\\ &\quad \mathsf{else}\ B2SCh \lhd\ \mathsf{QuoteReject}\langle \mathsf{x}_{\mathsf{AbortNo}}\rangle\,.\,\mathsf{P}_2)\\ Q \;&= B2SCh \rhd\ [\mathsf{QuoteAccept}(x)\,.\,Q_1 + \mathsf{QuoteReject}(y)\,.\,Q_2] \end{aligned}$$

Note that Q has kept the same shape as before: choice because the conditional guard is located where at the Buyer side, i.e. it is Buyer who chooses between two branches. We can further consider the result of adding session initiation, which is given as the second description in Example 2, we can further elaborate P and Q as follows.

$$\begin{aligned} P \;&\stackrel{\text{def}}{=} \overline{ch}(\boldsymbol{\nu}\, B2SCh, S2BCh).\\ &\qquad S2BCh(x_{\mathsf{Quote}}).\\ &\qquad\quad \mathsf{if}\ (x_{\mathsf{Quote}} \leq 1000)\\ &\qquad\quad \mathsf{then}\ B2SCh \lhd\ \mathsf{QuoteAccept}\langle``100''\rangle\,.\,\mathsf{P}_1\\ &\qquad\quad \mathsf{else}\ B2SCh \lhd\ \mathsf{QuoteReject}\langle \mathsf{x}_{\mathsf{AbortNo}}\rangle\,.\,\mathsf{P}_2)\\ Q \;&\stackrel{\text{def}}{=} !ch(B2SCh, S2BCh).\\ &\qquad \overline{S2BCh}\langle 100\rangle.\\ &\qquad\quad B2SCh \rhd\ [\mathsf{QuoteAccept}(x)\,.\,Q_1 + \mathsf{QuoteReject}(y)\,.\,Q_2] \end{aligned}$$

Note an input is compensated with an output and vice versa, similarly a branching with a selection.

EXAMPLE 15. Example 3 (page 34) presents the use of parallel in-session communications inside a single session. Using the same skeleton (73) as above, we define P and Q as follows:

$$\begin{aligned} P &\overset{\text{def}}{=} \overline{ch}(\boldsymbol{\nu}\, Op, Data). \\ &\qquad (\overline{Op} \lhd \mathsf{QuoteAcc}\langle 100\rangle.P_1 \mid \overline{Data} \lhd \mathsf{QuoteAcc}\langle adr\rangle.P_2) \\ Q &\overset{\text{def}}{=} !ch(Op, Data). \\ &\qquad (Op \rhd \mathsf{QuoteAcc}(x).Q_1 \mid Data \rhd \mathsf{QuoteAcc}(y).Q_2) \end{aligned}$$

in which Buyer offers two parallel outputs while Seller receives them with their dual inputs.

11.3. Reduction Rules. Reduction indicates evolution of processes and networks via communication and other actions. It is given as a binary relation over networks, written $N \to M$. The first rule is for initiation of a session via invocation of a shared service channel.

$$(\textsc{Init})\ \frac{-}{A[\,!ch(\tilde{s}).P \mid P'\,]_\sigma \mid B[\,\overline{ch}(\boldsymbol{\nu}\,\tilde{s}).Q \mid Q'\,]_{\sigma'} \;\to\; (\boldsymbol{\nu}\,\tilde{s})\ (A[\,!ch(\tilde{s}).P \mid P \mid P'\,]_\sigma \mid B[Q \mid Q'\,]_{\sigma'})}$$

In addition to this rule (as well as the next one), we may include intra-participant communication, on which we discuss Section 17.2. For in-session communication, we have:

$$(\textsc{Com})\ \frac{\sigma \vdash e \Downarrow v}{A[\,s \rhd \Sigma_i \mathsf{op}_i(x_i).P_i \,|\, P'\,]_\sigma | B[\,\overline{s} \lhd \mathsf{op_j}\langle e\rangle Q \,|\, Q'\,]_{\sigma'} \;\to\; A[\,P_j \mid P'\,]_{\sigma[x\mapsto v_j]} \mid B[\,Q \mid Q'\,]_{\sigma'}}$$

As before, the rule has its local version, which we omit.

Assignment only affects local store:

$$(\textsc{Assign})\ \frac{\sigma \vdash e \Downarrow v}{A[x := e.P \,|\, P'\,]_\sigma \;\to\; A[\,P \mid P'\,]_{\sigma[x\mapsto v]}}$$

In conditional, we first evaluates the guard expression, then, depending on its value, decides which branch should be chosen.

$$(\textsc{IfTrue})\frac{\sigma \vdash e \Downarrow \mathsf{tt}}{A[\mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \,|\, P'\,]_\sigma \;\to\; A[\,P_1 \mid P'\,]_\sigma}$$

$$(\textsc{IfFalse})\frac{\sigma \vdash e \Downarrow \mathsf{ff}}{A[\mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \,|\, P'\,]_\sigma \;\to\; A[\,P_2 \mid P'\,]_\sigma}$$

The internal sum $P \oplus Q$ has the following standard reduction.

$$(\textsc{Sum})\frac{-}{A[P_1 \oplus P_2 | R]_\sigma \;\to\; A[P_i | R]_\sigma \qquad (i = 1,2)}$$

The rule says $P_1 \oplus P_2$ can behave as either P_1 or P_2.

For parallel composition of processes, we have:

$$(\textsc{Par})\ \frac{A[\,P_1 \,|\, R\,]_\sigma \to A[\,P_1' \,|\, R\,]_{\sigma'}}{A[\,P_1 \,|\, P_2 \,|\, R\,]_\sigma \to A[\,P_1' \,|\, P_2 \,|\, R\,]_\sigma}$$

Finally for restriction:

$$(\textsc{Res})\ \frac{A[\,P\,]_\sigma \to A[\,P'\,]_{\sigma'}}{A[\,(\boldsymbol{\nu}\,s)\ P\,]_\sigma \to A[\,(\boldsymbol{\nu}\,s)\ P'\,]_\sigma}$$

We also list the corresponding contextual rules for networks.

$$(\textsc{Par-NW})\ \frac{M \;\to\; M'}{M|N \;\to\; M'|N} \qquad\qquad (\textsc{Res-NW})\ \frac{M \;\to\; M'}{(\boldsymbol{\nu}\,s)\ M \;\to\; (\boldsymbol{\nu}\,s)\ M'}$$

For recursion, we set:

$$(\textsc{Rec})\frac{P[\mathbf{rec}\,X.P/X] \;\to\; P'}{\mathbf{rec}\,X.P \;\to\; P'}$$

Finally the following rule says we take the reduction up to the structural rules:

$$(\textsc{Struct-NW})\frac{M \equiv M' \qquad M' \to N' \qquad N' \equiv N}{M \to N}$$

$$(\textsc{Init})\ \frac{-}{A[\,!ch(\tilde{s}).P \mid P'\,]_{\sigma} \mid B[\,\overline{ch}(\boldsymbol{\nu}\,\tilde{s}).Q \mid Q'\,]_{\sigma'} \ \rightarrow\ (\boldsymbol{\nu}\,\tilde{s})\ (A[\,!ch(\tilde{s}).P \mid P \mid P'\,]_{\sigma} \mid B[Q \mid Q'\,]_{\sigma'})}$$

$$(\textsc{Com})\ \frac{\sigma \vdash e \Downarrow v}{A[\,s \rhd \Sigma_i \mathsf{op}_i(x_i).P_i \,|\, P'\,]_{\sigma} | B[\,\overline{s} \lhd \mathsf{op_j}\langle e\rangle Q \,|\, Q'\,]_{\sigma'} \ \rightarrow\ A[\,P_j \mid P'\,]_{\sigma[x \mapsto v_j]} \,|\, B[\,Q \mid Q'\,]_{\sigma'}}$$

$$(\textsc{IfTrue})\frac{\sigma \vdash e \Downarrow \mathsf{tt}}{A[\text{if } e \text{ then } P_1 \text{ else } P_2 \,|\, P'\,]_{\sigma} \ \rightarrow\ A[\,P_1 \mid P'\,]_{\sigma}} \qquad (\textsc{Par-NW})\ \frac{M \ \rightarrow\ M'}{M|N \ \rightarrow\ M'|N}$$

$$(\textsc{IfFalse})\frac{\sigma \vdash e \Downarrow \mathsf{ff}}{A[\text{if } e \text{ then } P_1 \text{ else } P_2 \,|\, P'\,]_{\sigma} \ \rightarrow\ A[\,P_2 \mid P'\,]_{\sigma}} \qquad (\textsc{Res-NW})\ \frac{M \ \rightarrow\ M'}{(\boldsymbol{\nu}\,s)\ M \ \rightarrow\ (\boldsymbol{\nu}\,s)\ M'}$$

$$(\textsc{Assign})\ \frac{\sigma \vdash e \Downarrow v}{A[\,x := e.P \,|\, P'\,]_{\sigma} \ \rightarrow\ A[\,P \mid P'\,]_{\sigma[x \mapsto v]}} \qquad (\textsc{Res})\ \frac{A[\,P\,]_{\sigma} \rightarrow A[\,P'\,]_{\sigma'}}{A[\,(\boldsymbol{\nu}\,s)\ P\,]_{\sigma} \rightarrow A[\,(\boldsymbol{\nu}\,s)\ P'\,]_{\sigma}}$$

$$(\textsc{Sum})\frac{-}{A[P_1 \oplus P_2|R]_{\sigma} \ \rightarrow\ A[P_i|R]_{\sigma}} i \in \{1,2\} \qquad (\textsc{Par})\ \frac{A[\,P_1 \,|\, R\,]_{\sigma} \rightarrow A[\,P_1' \,|\, R\,]_{\sigma'}}{A[\,P_1 \,|\, P_2 \,|\, R\,]_{\sigma} \rightarrow A[\,P_1' \,|\, P_2 \,|\, R\,]_{\sigma}}$$

$$(\textsc{Struct-NW})\frac{M \equiv M' \qquad M' \rightarrow N' \qquad N' \equiv N}{M \rightarrow N} \qquad (\textsc{Rec})\frac{P[\mathbf{rec}\, X.P/X] \ \rightarrow\ P'}{\mathbf{rec}\, X.P \ \rightarrow\ P'}$$

FIGURE 21. Reduction Rules of the End-Point Calculus

where $\equiv$ is the least congruence on networks generated from:

$$\begin{array}{rcll}
P|\mathbf{0} & \equiv & P & \\
P|Q & \equiv & Q|P & \\
(P|Q)|R & \equiv & P|(Q|R) & \\
P \oplus P & \equiv & P & \\
P \oplus Q & \equiv & Q \oplus P & \\
(P \oplus Q) \oplus R & \equiv & P \oplus (Q \oplus R) & \\
(\boldsymbol{\nu}\,s)\ \mathbf{0} & \equiv & \mathbf{0} & \\
(\boldsymbol{\nu}\,s_1)\ (\boldsymbol{\nu}\,s_2)\ P & \equiv & (\boldsymbol{\nu}\,s_2)\ (\boldsymbol{\nu}\,s_1)\ P & \\
((\boldsymbol{\nu}\,s)\ P)|Q & \equiv & (\boldsymbol{\nu}\,s)\ (P|Q) & \qquad (s \notin \mathsf{fn}(Q))
\end{array}$$

and, for networks, we stipulate:

$$\begin{array}{rcll}
A[P]_{\sigma} & \equiv & A[Q]_{\sigma} & \qquad (P \equiv Q) \\
A[\,(\boldsymbol{\nu}\,s)\ P]_{\sigma} & \equiv & (\boldsymbol{\nu}\,s)\ (A[P]_{\sigma}) & \\
M|\varepsilon & \equiv & M & \\
M|N & \equiv & N|M & \\
(L|M)|N & \equiv & L|(M|N) & \\
(\boldsymbol{\nu}\,s)\ \varepsilon & \equiv & \varepsilon & \\
(\boldsymbol{\nu}\,s_1)\ (\boldsymbol{\nu}\,s_2)\ M & \equiv & (\boldsymbol{\nu}\,s_2)\ (\boldsymbol{\nu}\,s_1)\ M & \\
((\boldsymbol{\nu}\,s)\ M)|N & \equiv & (\boldsymbol{\nu}\,s)\ (M|N) & \qquad (s \notin \mathsf{fn}(N))
\end{array}$$

This concludes the presentation of all reduction rules. For reference, in Figure 21 we list all the reduction rules in one table.

REMARK 1. *In the definition of $\equiv$, the equations for $\oplus$ allows us to write the n-fold sum $\oplus_i P_i$ which reduces as, with $C[\,\cdot\,]$ being a reduction context, $C[\oplus_i P_i] \rightarrow C[P_i]$ for each i.*

12. End-Point Calculus (2): Typing

12.1. Types and Subtyping. As we did for the global calculus, we use session types [**24**], i.e. the typing for controlling the flow of operations and data through channels. We use the same set of types as the global calculus, whose grammar is reproduced below for convenience.

$$\begin{array}{lll} \theta & ::= & \texttt{bool} \mid \texttt{int} \mid \ldots \\ \alpha & ::= & \Sigma_i s \downarrow op_i(\theta_i)\,.\,\alpha_i \mid \Sigma_i s \uparrow op_i(\theta_i)\,.\,\alpha_i \mid \alpha_1 \mid \alpha_2 \mid \mathbf{t} \mid \mathbf{rec}\ \mathbf{t}.\alpha \mid \texttt{end} \end{array}$$

Above, as before, $\alpha, \beta, \ldots$ are called *session types*. Again as before we take $\mid$ to be commutative and associative, with the identity $\texttt{end}$. Recursive types are regarded as regular trees in the standard way [**41**]. We also use *service types*, ranged over by $\gamma, \gamma', \ldots$, given by:

$$\gamma \quad ::= \quad !(\tilde{s})\,\alpha\,@A \quad \mid \quad ?(\tilde{s})\,\alpha\,@A$$

Above, $!(\tilde{s})\,\alpha\,@A$ indicates the service located at A which is invoked with fresh session channels $\tilde{s}$ and offers service of the shape α, while $?(\tilde{s})\,\alpha\,@A$ indicates the type abstraction for the dual invocation, i.e. a client of an A's service which invokes with fresh channels $\tilde{s}$ and engages in interactions abstracted as α. Note $@A$ indicates the location of a *service* in both forms.

As before, $\tilde{s}$ should be a vector of pairwise distinct session channels which should cover all session channels in α, and α does not contain free type variables. $(\tilde{s})$ binds occurrences of session channels in $\tilde{s}$ in α, which induces the standard alpha-equality. We define the duality as:

$$\overline{!(\tilde{s})\,\alpha\,@A} = ?(\tilde{s})\,\overline{\alpha}\,@A \qquad\qquad \overline{?(\tilde{s})\,\alpha\,@A} = !(\tilde{s})\,\overline{\alpha}\,@A$$

where the notion of duality $\overline{\alpha}$ of α remains the same.

In the end-point calculus, it is useful to consider a subtyping relation on session types following [**19**]. The subtyping is defined only over closed types, i.e. those types in which no free type variables occur. The relation is written $\alpha \preceq \beta$.[7] which intuitively indicates that α is more gentle, or dually β is less constrained, in behaviour.

We define the subtyping relation using coinduction. A *subtyping witness* is a relation $\mathcal{R}$ which satisfies all of the following rules, each of which indicates the condition on $\mathcal{R}$, saying: for $\mathcal{R}$ to satisfy the conclusion, it should satisfy the condition(s) given in the premise. In each rule, $\approx$ is the standard type isomorphism.

$$(\text{SUB-IN})\ \frac{\beta \approx \Sigma_{j\in J} s \downarrow op_j(\theta_j)\,.\,\beta_j \quad I \supset J \quad \alpha_j \,\mathcal{R}\, \beta_j}{\Sigma_{i\in I} s \downarrow op_i(\theta_i)\,.\,\alpha_i \,\mathcal{R}\, \beta}$$

The rule says that for the branching type to be more gentle than the other then it should be the case that subsequent behaviours of an input are more gentle, and if it offers more options. Dually we have:

$$(\text{SUB-OUT})\ \frac{\beta \approx \Sigma_{j\in J} s \uparrow op_j(\theta_j)\,.\,\beta_j \quad I \subset J \quad \alpha_i \,\mathcal{R}\, \beta_i}{\Sigma_{i\in I} s \uparrow op_i(\theta_i)\,.\,\alpha_i \,\mathcal{R}\, \beta}$$

The remaining rules close the relation under type constructors and $\approx$.

$$(\text{SUB-PAR})\ \frac{\beta \approx \beta_1 \mid \beta_2 \quad \alpha_1 \,\mathcal{R}\, \beta_1 \quad \alpha_2 \,\mathcal{R}\, \beta_2}{\alpha_1 \mid \alpha_2 \,\mathcal{R}\, \beta} \qquad (\text{SUB-END})\ \frac{\beta \approx \texttt{end}}{\texttt{end} \,\mathcal{R}\, \beta} \qquad (\text{SUB-ISO})\ \frac{\alpha \,\mathcal{R}\, \beta}{\alpha' \,\mathcal{R}\, \beta}(\alpha \approx \alpha')$$

Together with the last rule, the subtyping rules are equivalent to saying we take two infinite trees and compare their structures by simulation, cf. [**19, 42**]. We shall write $\alpha \preceq \beta$ if for some subtyping witness $\mathcal{R}$ we have $\alpha \preceq \beta$. Note the only difference from the defining rules for defining relations for $\Subset$ (in Definition 1, page 47) is in the input branching rule.

PROPOSITION 6. *$\preceq$ on types modulo $\approx$ is a partial order.*

Proof. $\approx$ is surely a witness. If $\mathcal{R}$ is a witness and $\mathcal{R}'$ is another witness then by the shape of each rule $\mathcal{R};\mathcal{R}'$ is surely a witness. For example, if $\alpha_1|\alpha_2\mathcal{R}\gamma\mathcal{R}'\beta$ then $\gamma \approx \gamma_1|\gamma_2$ such that $\alpha_i\mathcal{R}\gamma_i$ $(i = 1,2)$ by (PAR), for which $\gamma_1|\gamma_2\mathcal{R}'\beta$ by (ISO) and by assumption. Applying (PAR) again we have $\beta \approx \beta_1|\beta_2$ such that $\gamma_i\mathcal{R}\beta_i$ $(i = 1,2)$. By transitivity of $\approx$ we are done. For anti-symmetry, we show

[7] The symbol in [**19**] is used dually, with the same formal content.

$\mathcal{S} = \preceq \cap (\preceq^{-1})$ coincides with $\approx$. Clearly $\approx \subset \mathcal{S}$. For the other direction we show $\mathcal{S}$ is a bisimulation which is by definition and by noting transition is deterministic. □

DEFINITION 3. *We write $\alpha \vee \beta$ for the lub of α and β if it exists.*

PROPOSITION 7. *If the upper bound of $\alpha_{1,2}$ exists (w.r.t. $\preceq$) then their least upper bound exists.*

Proof. By superimposing the trees corresponding to $\alpha_{1,2}$ and taking the union/intersection and checking the result is again a regular tree. □

12.2. Typing Rules. The typing judgement in the local calculus has the form:

$$\Gamma \vdash_A P \rhd \Delta$$

which mentions a participant name to be inhabited by P; and

$$\Gamma \vdash M \rhd \Delta.$$

which is for a network. Γ (service typing) and Δ (session typing) above are given by the following grammar.

$$\begin{array}{lcl} \Gamma & ::= & \emptyset \mid \Gamma, ch : \gamma \mid \Gamma, x@A : Var(\theta) \mid \Gamma, X : \Delta \\ \Delta & ::= & \emptyset \quad \mid \quad \Delta, \tilde{s}@A : \alpha \quad \mid \quad \Delta, \tilde{s} : \bot \end{array}$$

As before, we stipulate that both service and session typings define appropriate functions. In particular, whenever we write e.g. Γ_1, Γ_2, there are *no* free channels/session channels/variables shared between two typings. Some observations:

(1) One basic difference in the grammar above from the one for the global calculus (see Section 10) is that the session type assignment for the local calculus is given to the vector of names at a *single* participant. This is because a session type is now assigned to end-point behaviour, so that one end of a channel should have one end of a session type, rather than two sides coming together.
(2) When two sides of a session are compatible, we compose them and leave the assignment of $\bot$ to $\tilde{s}$ in the typing. Since $\bot$ is composable with no other types, this *effectively makes $\tilde{s}$ unusable in further composition*. This is the standard linear typing in the π-calculus.
(3) In the service typing, $ch : !(\tilde{s})\alpha@A$ is the same thing as $(\tilde{s})\alpha@A$ in the global calculus (hence we often identity these two). It is called *server type assignment*. $ch : ?(\tilde{s})\alpha@A$ is called *client type assignment*. As we stipulate below, the composition of $ch : !(\tilde{s})\alpha@A$ and $ch : ?(\tilde{s})\overline{\alpha}@A$ becomes $ch : !(\tilde{s})\alpha@A$, since a service can be usable not only once but also many times. This is from the standard replicated linear type discipline.

The types control composition of processes and networks through the following partial algebras. They say, in brief, session types are treated as linearly, while service types are treated as server-client types.

DEFINITION 4. Write $\gamma^!$ or $\gamma^?$ to indicate γ is a server or client type. Then we set:

$$\begin{array}{rcl} \gamma^? \odot \gamma^? & = & \gamma^? \\ \gamma^! \odot \overline{\gamma}^? & = & \gamma^! \\ \gamma^? \odot \overline{\gamma}^! & = & \gamma^! \end{array}$$

Otherwise $\gamma_1 \odot \gamma_2$ is undefined. Then we write $\Gamma_1 \asymp \Gamma_2$ when

(1) $\Gamma_1(ch) \odot \Gamma_2(ch)$ is defined for each $ch \in \mathsf{fn}(\Gamma_{1,2})$.
(2) $\Gamma_1(x) = \Gamma_2(x)$ for each $x \in \mathsf{fn}(\Gamma_{1,2})$.
(3) $\Gamma_1(X) = \Gamma_2(X)$ for each $X \in \mathsf{fn}(\Gamma_{1,2})$.

Finally when $\Gamma_1 \asymp \Gamma_2$ we set $\Gamma_1 \odot \Gamma_2$ as the union of Γ_1 and Γ_2 except, for each channel ch such that $ch \in \mathsf{fn}(\Gamma_{1,2})$, the type newly assigned to ch is $\Gamma_1(ch) \odot \Gamma_2(ch)$. Similarly we set:

$$\begin{array}{rcll} \alpha \odot \overline{\alpha} & = & \bot & \\ \alpha \odot \beta & = & \alpha | \beta & \qquad (\mathsf{fc}(\alpha) \cap \mathsf{fc}(\beta) = \emptyset) \end{array}$$

Otherwise $\alpha_1 \odot \alpha_2$ is undefined. As above we define $\Delta_1 \odot \Delta_2$ and $\Delta_1 \asymp \Delta_2$.

We can now introduce the typing rules. The first rule is for typing the inputting side of initialisation.

$$\text{(TInit-In)}\ \frac{\Gamma \vdash_A P \triangleright \tilde{s}@A : \alpha}{\Gamma, ch : !(\tilde{s})\alpha@A \vdash_A !ch(\tilde{s}).P \triangleright \emptyset}$$

Note that, in the premise, we do not allow those session channels other than the target of initialisation to be present in the session typing, nor another server typing with *ch* as a subject in Γ. The former prevents *free* session channels to be under the replicated input, guaranteeing their linear usage: the latter prevents another service channel to be under *ch*. We can further constrain Γ to have no server (!) mode channels: this makes *ch uniform receptive* in the sense of [**46**].

The outputting side of initialisation is analogous, except the linearity constraint needs not be specified.

$$\text{(TInit-Out)}\ \frac{\Gamma, ch : ?(\tilde{s})\alpha@B \vdash_A P \triangleright \Delta \cdot \tilde{s}@A : \alpha}{\Gamma, ch : ?(\tilde{s})\alpha@B \vdash_A \overline{ch}(\boldsymbol{\nu}\tilde{s}).P \triangleright \Delta}$$

Above A and B can be identical. The fact we allow $\overline{ch}@B : (\tilde{s})\alpha$ to occur in the premise means (together with (Par) rule) an invocation to a service can be done as many times as needed (as far as it is type correct).

Next we present typing for in-session communication, starting from input (which involves branching with distinct operators).

$$\text{(TBranch)}\ \frac{K \subseteq J \quad s \in \tilde{t} \quad \Gamma \vdash x_j : \theta_j \quad \Gamma \vdash_A P_j \triangleright \Delta \cdot \tilde{s}@A : \alpha_j}{\Gamma \vdash_A s \triangleright \Sigma_j \mathsf{op}_j(x_j).P_j \triangleright \Delta \cdot \tilde{s}@A : \Sigma_{k \in K} s_k !op_k(\theta_k) \,.\, \alpha_k}$$

In the rule above, the typing can have less branches than the real process, so that the process is prepared to receive (get invoked at) any operation specified in the type. Dually we have:

$$\text{(TSel)}\ \frac{j \in J \subseteq K \quad \Gamma \vdash e : Var(\theta_i) \quad \Gamma \vdash_A P \triangleright \Delta \cdot \tilde{s}@A : \alpha_j}{\Gamma \vdash_A \overline{s} \lhd \mathsf{op_j}\langle e \rangle.P \triangleright \Delta \cdot \tilde{s}@A : \Sigma_{k \in K} s \uparrow op_k(\theta_k) \,.\, \alpha_k}$$

Here the typing can have more branches than the real process, so that the process invokes with operators at most those specified in types. Combining (TBranch) and (TSel), an output never tries to invoke a non-existent option in its matching input.

The rules for assignment is standard.

$$\text{(TAssign)}\ \frac{\Gamma \vdash_A x : \theta \quad \Gamma \vdash e : \theta \quad \Gamma \vdash_A P \triangleright \Delta}{\Gamma \vdash_A x := e.P \triangleright \Delta}$$

The conditional is also standard.

$$\text{(TIf)}\ \frac{\Gamma \vdash e : \texttt{bool} \quad \Gamma \vdash_A P \triangleright \Delta \quad \Gamma \vdash_A Q \triangleright \Delta}{\Gamma \vdash_A \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta}$$

Note the session typings are identical for P and for Q in the premise: this is essentially a linearity constraint, ensuring a linear name (session channel) is used precisely once in each branch. Practical ramifications are possible: in particular, we can easily refine the linear typing into the affine one. The typing of a sum is similar to conditional.

$$\text{(TSum)}\ \frac{\Gamma \vdash_A P \triangleright \Delta \quad \Gamma \vdash_A Q \triangleright \Delta}{\Gamma \vdash_A P \oplus Q \triangleright \Delta}$$

The rule for parallel composition reads:

$$\text{(TPar)}\ \frac{\Gamma \vdash_A P \triangleright \Delta_1 \quad \Gamma \vdash_A Q \triangleright \Delta_2 \quad \Delta_1 \asymp \Delta_2}{\Gamma \vdash_A P \mid Q \triangleright \Delta_1 \odot \Delta_2}$$

The introduced $\perp$-types are eliminated by restriction.

$$\text{(TRes,1)}\ \frac{\Gamma \vdash_A P \triangleright \Delta, \tilde{s_1} s \tilde{s_2} : \perp}{\Gamma \vdash_A (\boldsymbol{\nu} s)\, P \triangleright \Delta, \tilde{s_1}\tilde{s_2} : \perp} \qquad \text{(TRes,2)}\ \frac{\Gamma \vdash_A P \triangleright \Delta, \varepsilon : \perp}{\Gamma \vdash_A (\boldsymbol{\nu} s)\, P \triangleright \Delta}$$

In (TRes,2), ε denotes the empty vector. The next two rules are for term variables and recursion, and is standard.

$$\text{(TVar)}\ \frac{-}{\Gamma, X : \Delta \vdash_A X \triangleright \Delta} \qquad \text{(TRec)}\ \frac{\Gamma, X : \Delta \vdash_A P \triangleright \Delta}{\Gamma \vdash_A \mathbf{rec}\, X.P \triangleright \Delta}$$

The rule for inaction introduces the empty session typing.

$$\text{(TINACT)}\ \frac{}{\Gamma \vdash_A \mathbf{0} \rhd \emptyset}$$

We may further constrain Γ so that it only contains assignments to term variables, imperative variables and client channels (i.e. of the form $\overline{ch}@A : (\tilde{s})\alpha$). If we add this constraint to this rule as well as to (TVAR), we have a property that the existence of an input channel typing in Γ implies its existence in the subject process/network.

To start session typing, we need to introduce inaction types (which represent a terminal point of a session type together with a recursive variable, which is introduced in (TVar)).

$$\text{(WEAK-}\mathtt{end}\text{)}\ \frac{\Gamma \vdash_A P \rhd \Delta \quad \{\tilde{s}\} \cap \mathsf{fn}(\Delta) = \emptyset}{\Gamma \vdash_A P \rhd \Delta \cdot \tilde{s}@A : \mathtt{end}} \qquad \text{(WEAK-}\bot\text{)}\ \frac{\Gamma \vdash_A P \rhd \Delta \quad \{\tilde{s}\} \cap \mathsf{fn}(\Delta) = \emptyset}{\Gamma \vdash_A P \rhd \Delta \cdot \tilde{s} : \bot}$$

The next rule links process typing to the typing of a network.

$$\text{(TPARTICIPANT)}\ \frac{\Gamma \vdash_A P \rhd \Delta \quad \Gamma \vdash \sigma@A}{\Gamma \vdash A[P]_\sigma \rhd \Delta}$$

Composition and inaction rules for networks follow.

$$\text{(TPAR-NW)}\ \frac{\Gamma \vdash N_1 \rhd \Delta_1 \quad \Gamma \vdash N_2 \rhd \Delta_2 \quad \Delta_1 \asymp \Delta_2}{\Gamma \vdash N_1 \mid N_2 \rhd \Delta_1 \odot \Delta_2} \qquad \text{(TINACT-NW)}\ \frac{}{\Gamma \vdash \varepsilon \rhd \emptyset}$$

Restriction rules are also a precise copy of the corresponding rules for processes.

$$\text{(TRES-NW,1)}\ \frac{\Gamma \vdash M \rhd \Delta, \tilde{s_1} s \tilde{s_2} : \bot}{\Gamma \vdash (\nu s)\, M \rhd \Delta, \tilde{s_1}\tilde{s_2} : \bot} \qquad \text{(TRES-NW,2)}\ \frac{\Gamma \vdash M \rhd \Delta, \varepsilon : \bot}{\Gamma \vdash M \rhd \Delta}$$

We also have an exact copy of the two weakening rules, listed below for reference:

$$\text{(WEAK-}\mathtt{end}\text{-NW)}\ \frac{\Gamma \vdash M \rhd \Delta \quad \{\tilde{s}\} \cap \mathsf{fn}(\Delta) = \emptyset}{\Gamma \vdash M \rhd \Delta \cdot \tilde{s}@A : \mathtt{end}} \qquad \text{(WEAK-}\bot\text{-NW)}\ \frac{\Gamma \vdash M \rhd \Delta \quad \{\tilde{s}\} \cap \mathsf{fn}(\Delta) = \emptyset}{\Gamma \vdash M \rhd \Delta \cdot \tilde{s} : \bot}$$

The list of all the typing rules are given in Figure 22

The standard syntactic properties follow. Below in (4), $\mathsf{fn}(\Gamma)$ denotes all names in Γ, which include term variables, standard variables and channels.

PROPOSITION 8.

(1) (weakening) $\Gamma \vdash M \rhd \Delta$ *implies* $\Gamma \cdot \Gamma' \vdash M \rhd \Delta$. *With* $\tilde{s}$ *fresh,* $\Gamma \vdash M \rhd \Delta$ *implies* $\Gamma \vdash M \rhd \Delta \cdot \tilde{s} : \bot$.

(2) (thinning) *Assume* $\mathsf{fc}(\Gamma') \cap \mathsf{fn}(M) = \emptyset$. *Then* $\Gamma \cdot \Gamma' \vdash M \rhd \Delta$ *implies* $\Gamma \vdash M \rhd \Delta$.

(3) (subsumption) *If* $\Gamma, \overline{ch}@A : (\tilde{s})\alpha \vdash M \rhd \Delta$ *and* $\alpha \preceq \beta$ *then* $\Gamma, \overline{ch}@A : (\tilde{s})\beta \vdash M \rhd \Delta$. *Similarly, if* $\Gamma \vdash M \rhd \Delta \cdot \tilde{s}@A : \alpha$ *and* $\alpha \preceq \beta$ *then* $\Gamma \vdash M \rhd \Delta \cdot \tilde{s}@A : \beta$.

Proof. In Appendix B. □

REMARK 2 (subsumption at service channels). The subsumption for the service typing at input channels does not hold in the present system. However *semantically* such service typing is in fact sound. Thus we may as well add the following rule:

$$\text{(SUBS-SERVICE)}\ \frac{\Gamma, ch@A : (\tilde{s})\alpha \vdash M \rhd \Delta \quad \alpha \preceq \beta}{\Gamma, ch@A : (\tilde{s})\beta \vdash M \rhd \Delta}$$

A basic consequence of adding this rule is that we have not only the minimal typing but also the principal typing, see Remark 3 below.

The following result says that we can always find a representative typing for a given process, and, moreover, we can do so effectively. Such a type is minimum among all assignable typings w.r.t. the subtyping relation, so that we call it the *minimal typing* of a given term.

CONVENTION 5. *A typing* $\Gamma \vdash M \rhd \Delta$ *is* strict *if all free identifiers in* Γ *and* Δ *occur in* M*. We also write* $\Gamma \vdash M$ *for* $\Gamma \vdash M \rhd \emptyset$*, similarly* $\Gamma \vdash_A P$ *stands for* $\Gamma \vdash_A P \rhd \emptyset$*. Further we write* $\Gamma_0 \prec \Gamma$ *and* $\Delta_0 \prec \Delta$ *by extending* $\prec$ *point-wise at their service/session channels (for variables typing should coincide).*

$$(\textsc{TInit-In})\ \frac{\Gamma \vdash_A P \rhd \tilde{s}@A : \alpha}{\Gamma, ch : !(\tilde{s})\alpha@A \vdash_A !ch(\tilde{s}).P \rhd \emptyset}$$

$$(\textsc{TInit-Out})\ \frac{\Gamma, ch : (\tilde{s})\alpha@B \vdash_A P \rhd \Delta \cdot \tilde{s}@A : \alpha}{\Gamma, ch : (\tilde{s})\alpha@B \vdash_A \overline{ch}(\boldsymbol{\nu}\tilde{s}).P \rhd \Delta}$$

$$(\textsc{TBranch})\ \frac{K \subseteq J \quad s \in \tilde{t} \quad \Gamma \vdash x_j : \theta_j \quad \Gamma \vdash_A P_j \rhd \Delta \cdot \tilde{s}@A : \alpha_j}{\Gamma \vdash s \rhd \Sigma_j \mathsf{op}_j(x_j).P_j \rhd \Delta \cdot \tilde{s}@A : \Sigma_{k \in K} s_k \mathsf{op}_\mathsf{k}(\theta_k) \,.\, \alpha_k}$$

$$(\textsc{TSel})\ \frac{j \in J \subseteq K \quad \Gamma \vdash e : Var(\theta_i) \quad \Gamma \vdash_A P \rhd \Delta \cdot \tilde{s}@A : \alpha_j}{\Gamma \vdash_A \overline{s} \lhd \mathsf{op_j}\langle e \rangle.P \rhd \Delta \cdot \tilde{s}@A : \Sigma_{k \in K} s \uparrow op_k(\theta_k) \,.\, \alpha_k}$$

$$(\textsc{TAssign})\ \frac{\Gamma \vdash_A x : \theta \quad \Gamma \vdash e : \theta \quad \Gamma \vdash_A P \rhd \Delta}{\Gamma \vdash_A x := e.P \rhd \Delta}$$

$$(\textsc{TIf})\ \frac{\Gamma \vdash e : \mathtt{bool} \quad \Gamma \vdash_A P \rhd \Delta \quad \Gamma \vdash_A Q \rhd \Delta}{\Gamma \vdash_A \textsf{if } e \textsf{ then } P \textsf{ else } Q \rhd \Delta} \qquad (\textsc{TSum})\ \frac{\Gamma \vdash_A P \rhd \Delta \quad \Gamma \vdash_A Q \rhd \Delta}{\Gamma \vdash_A P \oplus Q \rhd \Delta}$$

$$(\textsc{TPar})\ \frac{\Gamma \vdash_A P \rhd \Delta_1 \quad \Gamma \vdash_A Q \rhd \Delta_2 \quad \Delta_1 \asymp \Delta_2}{\Gamma \vdash_A P \mid Q \rhd \Delta_1 \odot \Delta_2}$$

$$(\textsc{TRes},1)\ \frac{\Gamma \vdash_A P \rhd \Delta, \tilde{s_1} s \tilde{s_2} : \bot}{\Gamma \vdash_A (\boldsymbol{\nu} s)\, P \rhd \Delta, \tilde{s_1}\tilde{s_2} : \bot} \qquad (\textsc{TRes},2)\ \frac{\Gamma \vdash_A P \rhd \Delta, \varepsilon : \bot}{\Gamma \vdash_A (\boldsymbol{\nu} s)\, P \rhd \Delta}$$

$$(\textsc{TVar})\ \frac{-}{\Gamma, X : \Delta \vdash_A X \rhd \Delta} \qquad (\textsc{TRec})\ \frac{\Gamma, X : \Delta \vdash_A P \rhd \Delta}{\Gamma \vdash_A \mathbf{rec}\, X.P \rhd \Delta}$$

$$(\textsc{TInact})\ \frac{}{\Gamma \vdash_A \mathbf{0} \rhd \emptyset}$$

$$(\textsc{Weak}\text{-}\mathsf{end})\ \frac{\Gamma \vdash_A P \rhd \Delta \quad \{\tilde{s}\} \cap \mathsf{fn}(\Delta) = \emptyset}{\Gamma \vdash_A P \rhd \Delta \cdot \tilde{s}@A : \mathsf{end}} \qquad (\textsc{Weak}\text{-}\bot)\ \frac{\Gamma \vdash_A P \rhd \Delta \quad \{\tilde{s}\} \cap \mathsf{fn}(\Delta) = \emptyset}{\Gamma \vdash_A P \rhd \Delta \cdot \tilde{s} : \bot}$$

$$(\textsc{TParticipant})\ \frac{\Gamma \vdash_A P \rhd \Delta \quad \Gamma \vdash \sigma@A}{\Gamma \vdash A[P]_\sigma \rhd \Delta}$$

$$(\textsc{TPar-NW})\ \frac{\Gamma \vdash N_1 \rhd \Delta_1 \quad \Gamma \vdash N_2 \rhd \Delta_2 \quad \Delta_1 \asymp \Delta_2}{\Gamma \vdash N_1 \mid N_2 \rhd \Delta_1 \odot \Delta_2}$$

$$(\textsc{TRes-NW},1)\ \frac{\Gamma \vdash M \rhd \Delta, \tilde{s_1} s \tilde{s_2} : \bot}{\Gamma \vdash (\boldsymbol{\nu} s)\, M \rhd \Delta, \tilde{s_1}\tilde{s_2} : \bot} \qquad (\textsc{TRes-NW},2)\ \frac{\Gamma \vdash M \rhd \Delta, \varepsilon : \bot}{\Gamma \vdash M \rhd \Delta}$$

$$(\textsc{TInact-NW})\ \frac{}{\Gamma \vdash \varepsilon \rhd \emptyset}$$

$$(\textsc{Weak}\text{-}\mathsf{end}\text{-}\textsc{NW})\ \frac{\Gamma \vdash M \rhd \Delta \quad \{\tilde{s}\} \cap \mathsf{fn}(\Delta) = \emptyset}{\Gamma \vdash M \rhd \Delta \cdot \tilde{s}@A : \mathsf{end}}$$

$$(\textsc{Weak}\text{-}\bot\text{-}\textsc{NW})\ \frac{\Gamma \vdash M \rhd \Delta \quad \{\tilde{s}\} \cap \mathsf{fn}(\Delta) = \emptyset}{\Gamma \vdash M \rhd \Delta \cdot \tilde{s} : \bot}$$

FIGURE 22. Typing Rules for End-Point Calculus

DEFINITION 5 (Minimal Typing). *Assume M is typable. Then $\Gamma_0 \vdash M \triangleright \Delta_0$ is the* minimal typing of *M if, whenever $\Gamma \vdash M \triangleright \Delta$ is strict, we have $\Gamma_0 \prec \Gamma$ and $\Delta_0 \prec \Delta$.*

PROPOSITION 9. (existence of minimal typing) *Let $\Gamma_0 \vdash M \triangleright \Delta_0$ be the minimal typing of M. Then Γ_0 and Δ_0 are algorithmically calculable from M.*

Proof. This is the standard result in session typing systems [**18**]. For reference, Figure 23 gives the derivation rules. The rule uses the lub $\vee$ (cf. Definition 3) in (MTINIT-OUT), (MTIF), (MTSUM) and (MTPAR). In (MTINIT-IN), the expression $\alpha[\mathtt{end}/\tilde{\mathbf{t}}]$ denotes the result of substituting $\mathtt{end}$ for each (free, by bound name convention) occurrence of type variables from $\tilde{\mathbf{t}}$. $\vee$ denotes taking the join with respect to the subtyping ordering. $\asymp$ is taken so that an output type α and an input type β can be coherent in the following way:

$$\alpha^{\downarrow} \asymp \beta^{\uparrow} \qquad \Longleftrightarrow \qquad \alpha \preceq \overline{\beta}$$

(note this means α has more branches than β at each input point). Similarly for the service typing. Composition $\odot$ at service typing then always preserves the input side of the typing, i.e. assuming $\alpha^{\downarrow} \asymp \beta^{\uparrow}$, we have

$$!(\tilde{s})\alpha \odot ?(\tilde{s})\beta \overset{\text{def}}{=} !(\tilde{s})\alpha \qquad (\alpha \asymp \beta)$$

For (MTREC), as we discussed in the proof of Proposition 5 (page 48, Section 10.4), the expression

$$\mathtt{solve}(\mathbf{t}, \Delta)$$

denotes the result of calculating the proper recursive type (or appropriate substitution) which follows [**18**] (which is simpler than the standard constraint solving for principal types for recursively typed λ-calculus[**16, 26**] because of the lack of channel passing). The expression returns, for each component $\tilde{s}_i : \alpha_i$ of Δ, either $\tilde{s}_i : \mathbf{rec}\ \mathbf{t}.\alpha_i$ when $\mathbf{t}$ does exist in α_i or, if not, $\tilde{s}_i : \alpha_i[\mathtt{end}/\mathbf{t}]$. That the given system derives the minimal typing is by induction on the typing rules, comparing each rule with the corresponding one in Figure 22. □

REMARK 3 (principal typing). The minimal typing of a typable network/process is determined uniquely up to the standard isomorphism on recursive types. However this minimal typing may *not* be a principal typing, in the sense that even if we have $\Gamma \vdash M$ such that Γ is minimal, and if we have $\Gamma \preceq \Gamma'$, it may not be the case we have $\Gamma \vdash M$. This is because of the lack of syntactic subtyping at service (replicated) channels, as discussed in Remark 2, page 61, By adding (SUBS-SERVICE) noted in Remark 2, each typable term has a principal typing.

We next prove the central property of the typing rules, the subject reduction.

LEMMA 2. (substitution)

(1) *If $\Gamma \vdash A[P]_{\sigma} \triangleright \Delta$, $\Gamma \vdash x@A : \theta$ and $\Gamma \vdash v : \theta$, then $\Gamma \vdash A[P]_{\sigma[x \mapsto v]} \triangleright \Delta$.*
(2) *If $\Gamma, X : \Delta \vdash_A P \triangleright \Delta'$ and $\Gamma \vdash_A Q \triangleright \Delta$, then $\Gamma \vdash P[Q/X] \triangleright \Delta$.*

Proof. Standard. See Appendix B. □

LEMMA 3. (subject congruence) *If $\Gamma \vdash M \triangleright \Delta$ and $M \equiv N$ then $\Gamma \vdash N \triangleright \Delta$.*

Proof. Standard. See Appendix B. □

THEOREM 2. (Subject Reduction) *If $\Gamma \vdash N \triangleright \Delta$ and $N \rightarrow N'$ then $\Gamma \vdash N' \triangleright \Delta$.*

Proof. By Lemmas 2 and 3. See Appendix B. □

DEFINITION 6 (Communication Error). We say M has a *communication error* if either:

$$M \equiv C_r[s \triangleright \Sigma_i \mathsf{op}_i(x_i).P_i | \overline{s} \triangleleft \mathsf{op}\langle.\rangle Q] \quad \text{s.t.} \quad \mathsf{op} \notin \{\mathsf{op}_i\}$$

or

$$M \equiv C_r[A[s \triangleright \Sigma_i \mathsf{op}_i(x_i).P_i | R]_{\sigma} | B[\overline{s} \triangleleft \mathsf{op}\langle.\rangle Q | S]_{\sigma'}] \quad \text{s.t.} \quad \mathsf{op} \notin \{\mathsf{op}_i\}.$$

or

$$M \equiv C_r[A[s \triangleright \Sigma_i \mathsf{op}_i(x_i^{\alpha}).P_i | R]_{\sigma} | B[\overline{s} \triangleleft \mathsf{op}_\mathsf{i}\langle.\rangle Q | S]_{\sigma'}] \quad \text{s.t.} \quad \mathsf{op} \notin \{\mathsf{op}_i\}.$$

where $C[\,]$ is a reduction context (i.e. a context whose hole is not under a prefix).

$$(\text{MTI}\textsc{nit-}\text{I}\textsc{n})\ \frac{\Gamma \vdash^{\mathsf{min}}_{A} P \rhd \tilde{s}@A : \alpha \quad \tilde{\mathbf{t}}\text{ exhaust type variables in }\alpha.}{\Gamma, ch : !(\tilde{s})\alpha[\mathtt{end}/\tilde{\mathbf{t}}]@A \vdash^{\mathsf{min}}_{A} !ch(\tilde{s}).P \rhd \emptyset}$$

$$(\text{MTI}\textsc{nit-}\text{O}\textsc{ut}1)\ \frac{\Gamma \vdash^{\mathsf{min}}_{A} P \rhd \Delta \cdot \tilde{s}@A : \beta}{\Gamma, ch : ?(\tilde{s})(\beta)@B \vdash^{\mathsf{min}}_{A} \overline{ch}(\boldsymbol{\nu}\,\tilde{s}).P \rhd \Delta}$$

$$(\text{MTI}\textsc{nit-}\text{O}\textsc{ut}2)\ \frac{\Gamma, ch : ?(\tilde{s})\alpha@B \vdash^{\mathsf{min}}_{A} P \rhd \Delta \cdot \tilde{s}@A : \beta}{\Gamma, ch : ?(\tilde{s})(\alpha \vee \beta)@B \vdash^{\mathsf{min}}_{A} \overline{ch}(\boldsymbol{\nu}\,\tilde{s}).P \rhd \Delta}$$

$$(\text{MTB}\textsc{ranch})\ \frac{s \in \tilde{t} \quad \Gamma \vdash x_j : \theta_j \quad \Gamma \vdash^{\mathsf{min}}_{A} P_j \rhd \Delta \cdot \tilde{s}@A : \alpha_j}{\Gamma \vdash^{\mathsf{min}} s \rhd \Sigma_j \mathsf{op}_j(x_j).P_j \rhd \Delta \cdot \tilde{s}@A : \Sigma_{j \in J} s_k \mathtt{op_j}(\theta_j)\,.\,\alpha_j}$$

$$(\text{MTS}\textsc{el})\ \frac{\Gamma \vdash e : \mathit{Var}(\theta_i) \quad \Gamma \vdash^{\mathsf{min}}_{A} P \rhd \Delta \cdot \tilde{s}@A : \alpha_j}{\Gamma \vdash^{\mathsf{min}}_{A} \overline{s} \lhd \mathsf{op_j}\langle e \rangle.P \rhd \Delta \cdot \tilde{s}@A : \Sigma_{j \in J} s \uparrow \mathtt{op_j}(\theta_j)\,.\,\alpha_j}$$

$$(\text{MTA}\textsc{ssign})\ \frac{\Gamma \vdash_{A} x : \theta \quad \Gamma \vdash e : \theta \quad \Gamma \vdash^{\mathsf{min}}_{A} P \rhd \Delta}{\Gamma \vdash^{\mathsf{min}}_{A} x := e.P \rhd \Delta}$$

$$(\text{MTI}\textsc{f})\ \frac{\Gamma_i \vdash e : \mathtt{bool} \quad \Gamma_i \vdash^{\mathsf{min}}_{A} P \rhd \Delta_i\ (i = 1, 2)}{\Gamma_1 \vee \Gamma_2 \vdash^{\mathsf{min}}_{A} \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \rhd \Delta_1 \vee \Delta_2}$$

$$(\text{MTS}\textsc{um})\ \frac{\Gamma_i \vdash^{\mathsf{min}}_{A} P_i \rhd \Delta_i\ (i = 1, 2)}{\Gamma_1 \vee \Gamma_2 \vdash^{\mathsf{min}}_{A} P_1 \oplus P_2 \rhd \Delta_1 \vee \Delta_2}$$

$$(\text{MTP}\textsc{ar})\ \frac{\Gamma \vdash^{\mathsf{min}}_{A} P \rhd \Delta_1 \quad \Gamma \vdash^{\mathsf{min}}_{A} Q \rhd \Delta_2 \quad \Delta_1 \asymp \Delta_2}{\Gamma_1 \vee \Gamma_2 \vdash^{\mathsf{min}}_{A} P | Q \rhd \Delta_1 \odot \Delta_2}$$

$$(\text{MTR}\textsc{es},1)\ \frac{\Gamma \vdash^{\mathsf{min}}_{A} P \rhd \Delta, \tilde{s_1} s \tilde{s_2} : \bot}{\Gamma \vdash^{\mathsf{min}}_{A} (\boldsymbol{\nu}\,s)\, P \rhd \Delta, \tilde{s_1}\tilde{s_2} : \bot} \qquad (\text{MTR}\textsc{es},2)\ \frac{\Gamma \vdash^{\mathsf{min}}_{A} P \rhd \Delta, \varepsilon : \bot}{\Gamma \vdash^{\mathsf{min}}_{A} (\boldsymbol{\nu}\,s)\, P \rhd \Delta}$$

$$(\text{MTV}\textsc{ar})\ \frac{\psi = \{\tilde{s_i}\}}{\Gamma, X : \mathbf{t} \vdash^{\mathsf{min}}_{A} X \rhd \cup_i \tilde{s_i} : \mathbf{t}} \qquad (\text{MTR}\textsc{ec})\ \frac{\Gamma, X : \mathbf{t} \vdash^{\mathsf{min}}_{A} P \rhd \Delta}{\Gamma \vdash^{\mathsf{min}}_{A} \mathbf{rec}\, X.P \rhd \mathtt{solve}(\mathbf{t}, \Delta)}$$

$$(\text{MTI}\textsc{nact})\ \frac{\psi = \{\tilde{s_i}\}}{\emptyset \vdash^{\mathsf{min}}_{A} \mathbf{0} \rhd \cup_i \tilde{s_i} : \mathtt{end}}$$

$$(\text{MTP}\textsc{articipant})\ \frac{\Gamma \vdash^{\mathsf{min}}_{A} P \rhd \Delta \quad \Gamma \vdash \sigma@A}{\Gamma \vdash^{\mathsf{min}} A[P]_{\sigma} \rhd \Delta} \qquad (\text{MTP}\textsc{ar-}\text{NW})\ \frac{\Gamma \vdash^{\mathsf{min}} N_i \rhd \Delta_i\ (i = 1, 2) \quad \Delta_1 \asymp \Delta_2}{\Gamma \vdash^{\mathsf{min}} N_1 \mid N_2 \rhd \Delta_1 \odot \Delta_2}$$

$$(\text{MTR}\textsc{es-}\text{NW},1)\ \frac{\Gamma \vdash^{\mathsf{min}} M \rhd \Delta, \tilde{s_1} s \tilde{s_2} : \bot}{\Gamma \vdash^{\mathsf{min}} (\boldsymbol{\nu}\,s)\, M \rhd \Delta, \tilde{s_1}\tilde{s_2} : \bot} \qquad (\text{MTR}\textsc{es-}\text{NW},2)\ \frac{\Gamma \vdash^{\mathsf{min}} M \rhd \Delta, \varepsilon : \bot}{\Gamma \vdash^{\mathsf{min}} M \rhd \Delta}$$

$$(\text{MTI}\textsc{nact-}\text{NW})\ \frac{}{\Gamma \vdash^{\mathsf{min}} \varepsilon \rhd \emptyset}$$

FIGURE 23. Minimal Typing Rules for End-Point Calculus

That is, M has a communication error when it contains an input and an output at a common channel which however do not match in operator names (we can further add mismatch in types of evaluation). A basic corollary of Theorem 2 follows.

COROLLARY 1. (Lack of Communication Error) *If* $\Gamma \vdash N \rhd \Delta$ *and* $N \rightarrow^{*} M$*, then* M *never contains a communication error.*

Proof. By Lemma 3 and by noting an incompatible redex is not typable. □
Thus once a process/network is well-typed, it never go into a communication mismatch.

12.3. Examples of Typed Terms.

EXAMPLE 16. We can now give a possible end-point version of what we showed in Example 11:

$$\begin{array}{l}
\mathtt{Buyer}[\mathtt{B2SCh}\langle s\rangle\,.\,s \lhd \mathtt{RequestForQuote}\,.\,s \rhd \mathtt{QuoteResponse}(x_{\mathtt{quote}})\,. \\
\qquad s \lhd (\mathtt{QuoteReject}\, + \\
\qquad \mathtt{QuoteAccept}\,.\,s \rhd \mathtt{OrderConfirmation}\,.\,s \lhd \mathtt{DeliveryDetails})]_{\alpha} \mid \\
\mathtt{Seller}[\mathtt{B2SCh}(s)\,.\,s \rhd \mathtt{RequestForQuote}\,.\,s \lhd \mathtt{QuoteResponse}\langle v_{\mathtt{quote}}\rangle\,. \\
\qquad s \rhd (\mathtt{QuoteReject}\, + \\
\qquad \mathtt{QuoteAccept}\,.\,s \lhd \mathtt{OrderConfirmation}\,.\,\mathtt{S2ShCh}\langle s'\rangle\,. \\
\qquad s' \lhd \mathtt{RequestDelDetails}\langle \mathtt{Buyer}\rangle\,.\,s \lhd \mathtt{DeliveryDetails}(x_{\mathtt{DD}}) \\
\qquad s \rhd \mathtt{DeliveryDetails})]_{\beta} \mid \\
\mathtt{Shipper}[\mathtt{S2ShCh}(s')\,.\,s' \rhd \mathtt{RequestDelDetails}(x_{\mathtt{Client}})\,.\,s \lhd \mathtt{DeliveryDetails}(\mathtt{DD})]_{\gamma}
\end{array}$$

It is simple to verify that the typing we gave in the previous section for the global view of this protocol is just good enough for typing the network above.

13. Theory of End-Point Projection (1): Connectedness

In preceding sections, We have presented many example specifications both as a global view in the global calculus and as a local view written in the end-point calculus. In doing so, we always introduced a global description first, and from that one we recovered the corresponding end-point processes. From an engineering viewpoint, these two steps — start from a global description, then extract out of it a local description for each end-point — offer an effective methods for designing and coding communication-centric programs. It is often simply a **pain** to design, implement and validate an application that involves complex interactions among processes and which *together work correctly*, if we are to solely rely on descriptions of local behaviours. This is why such tools as message sequence charts and sequence diagrams have been used as a primary way to design communication behaviour. In fact, the primary concern of the design/requirement of communication behaviour of an application would in general be how global information exchange among processes will take place and how these interactions lead to desired effects: the local behaviour of individual components only matter to realise this global scenario. Thus, in designing and implementing communication-centric software, one may as well start from a global description of expected behaviour, then translate it into local descriptions. How this can be done generally and uniformly with a formal foundation is the theme of this section, studied in the distilled setting of the two calculi of interaction.

Translating a global description to its end-point counterpart, the process called *end-point projection*, can however be tricky, because we can easily produce a global description which does not correspond to any reasonable local counterpart. In other words, if you do not follow good principles, our global description does *not* in fact describe realisable interaction. But are there general principles for global descriptions which guarantee any global description be uniformly mapped to correct end-point behaviour as far as it follows them? Such principles should not be too restrictive, allowing projection of a large class of global descriptions onto their efficient local realisations.

In the context of the core calculi we presented in this paper, we have identified three simple descriptive principles, whose technical examination is the purpose of the present section. These are:

- *Connectedness*, which says a basic local causality principle is obeyed in a global description.
- *Well-threadedness*, which says a stronger locality principle based on session types.
- *Coherence*, which specifies, on the basis of well-threadedness, consistency of description for each "service".

All these principles are stipulated incrementally on the basis of well-typedness: well-threadedness does not make sense without an interaction being connected; and coherence can only be defined for well-threaded interactions. These three conditions not only offer natural disciplines for well-structured description, but also they offer gradually deeper analysis of operational aspects of global description. Connectedness uncovers causal relationship among actions, on whose basis well-threadedness dissects how we can extract atomic chunks of local activities (called *threads*) from a global interaction, crucially using the underlying type structure. Coherence stipulates the condition under which these threads can be formed and combined to produce a whole behaviour of each participant. The resulting participants can now realise, when combined together, all and only interactions prescribed in the original global description. Thus by way of offering a precise analysis of the conditions for local projectability of a global description, these three principles let us arrive at the construction of a formally founded end-point projection. Descriptive principles are by themselves structural analysis of the operational content of global descriptions, leading to the function which maps them to the corresponding local descriptions.

13.1. Connectedness. Connectedness dictates a local causality principle in interaction — if A initiates any action (say sending messages, assignment, ..) as a result of a previous event (e.g. reception of a message), then that preceding event should take place at A. For example, consider:

$$A \to B : s\langle \mathsf{op}_1,\, e_1,\, y_1 \rangle . C \to D : s'\langle \mathsf{op}_2,\, e_2,\, y_2 \rangle . \mathbf{0}. \tag{74}$$

Following the dynamic semantics of the global calculus, there is first an execution of the interaction between A and B and then an interaction between C and D takes place. For implementing such a sequence of interactions in a distributed setting, we need a hidden notification message from B to C. That is. (74) does not describe all of the communication sequences needed to realise the demanded sequencing. So (74) is an incomplete description of communication behaviour. This is why we wish to avoid e descriptions violating the local causality principle such as (74).[8]

To formalise the local causality principle informally discussed above, we need to say which participant initiates an action in I: this participant should be the place where the preceding event happens. This notion is defined as follows.

DEFINITION 7 (initiating participants). Given an interaction I in which hiding does not occur, its *initiating participants*, denoted $\mathsf{top}(I)$, is inductively given as follows.

$$\mathsf{top}(I) \quad \stackrel{\text{def}}{=} \quad \begin{cases} \{A\} & \text{if } I \stackrel{\text{def}}{=} A \to B : ch(\boldsymbol{\nu}\,\tilde{s})\,.\,I' \\ \{A\} & \text{if } I \stackrel{\text{def}}{=} A \to B : s\langle \mathsf{op},\ e,\ x\rangle\,.\,I \\ \{A\} & \text{if } I \stackrel{\text{def}}{=} \mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2 \\ \{A\} & \text{if } x@A := e\,.\,I' \\ \{A\} & \text{if } I \stackrel{\text{def}}{=} X^A \\ \emptyset & \text{if } I \stackrel{\text{def}}{=} \mathbf{0} \\ \mathsf{top}(I') & \text{if } I \stackrel{\text{def}}{=} \mathbf{rec}\, X^A\,.\,I' \\ \mathsf{top}(I_1) \cup \mathsf{top}(I_2) & \text{if } I \stackrel{\text{def}}{=} I_1 \mid I_2 \\ \mathsf{top}(I_1) \cup \mathsf{top}(I_2) & \text{if } I \stackrel{\text{def}}{=} I_1 + I_2 \end{cases}$$

If $A \in \mathsf{top}(I)$, we say A is an *initiating participant of I*.

REMARK 4. By Convention 1 (cf. page 33), it is natural to restrict concerned interactions to terms without restriction.

Given I, the function top generates a set of participant. The generated set contains the participants that initiates the first action of I (note we count "sending" actions, which are session initiation and sending a message, as "initiating" actions, but we don't do so for the corresponding receiving actions: as we shall analyse later in Section 13.2-13.3, this is the most robust option, though there are alternatives). The annotation for a term variable, A for X^A, has now revealed its role, as a signifier of the initiating participant of the behaviour embodied by X. We discuss how this allows validation of connectedness in the presence of recursion. We now present the inductive definition of connectedness.

CONVENTION 6 (well-typedness). Henceforth we only consider well-typed terms for both global and local calculi, unless otherwise specified.

DEFINITION 8 (Strong Connectedness). *The collection of* strongly connected interactions *are inductively generated as follows (considering only well-typed terms, cf. Convention 6).*

(1) $A \to B : ch(\boldsymbol{\nu}\,s)\,.\,I'$ *is strongly connected when* I' *is strongly connected and* $\mathsf{top}(I') = \{B\}$.

(2) $A \to B : s\langle \mathsf{op},\ e,\ x\rangle\,.\,I$ *is strongly connected when I is strongly connected and* $\mathsf{top}(I) = \{B\}$.

(3) *if* $e@A$ *then* I_1 *else* I_2 *is strongly connected when* I_1, I_2 *are strongly connected and* $\{A\} = \mathsf{top}(I_1) = \mathsf{top}(I_2)$.

(4) $I_1 + I_2$ *is strongly connected when* I_1, I_2 *are both strongly connected and* $\{A\} = \mathsf{top}(I_1) = \mathsf{top}(I_2)$.

(5) $\mathbf{rec}\, X^A\,.\,I'$ *is strongly connected when* $\{A\} = \mathsf{top}(I')$.

(6) X^A *is always strongly connected.*

(7) $x@A := e\,.\,I'$ *is strongly connected when* I' *is strongly connected and* $\{A\} = \mathsf{top}(I')$.

(8) $I_1 \mid I_2$ *is strongly connected when both* I_1 *and* I_2 *are strongly connected.*

(9) $(\boldsymbol{\nu}\,s)\ I$ *is strongly connected when I is strongly connected.*

(10) $\mathbf{0}$ *is always strongly connected.*

[8] We can of course insert additional communication missing from (74). But this is precisely we need a principle dictating when such an insertion is necessary and how this may be done.

Note strongly connected implies well-typed. Strong connectedness says that, in communication actions, only the message reception leads to activity (at the receiving participant), and that such activity should immediately follow the reception of messages. Variants of the notion of connectedness (which loosen some of the clauses of the definition above) are discussed in the next subsection. Among others the following variant allows an identical technical development as the notion presented above while useful in various examples.

As we shall discuss in the next subsection, there are more looser variants of connectedness which can be used in its place, allowing all the remaining theoretical development to go through. Strong connectedness is chosen since it allows a most transparent theoretical development. Further we can often encode descriptions following looser principles using strongly connected interactions preserving semantics.

The defining clauses of Definition 8 should be naturally understood. We only illustrate the treatment of recursion. Given a recursion $\mathbf{rec}\, X^A . I'$ and its operational semantics (cf. Section 10.3), each occurrence of the term variable X can be seen as a link back to the beginning of recursion, i.e. the recursive term $\mathbf{rec}\, X^A . I'$ itself. This view suggests that, for guaranteeing connectedness, we need to make sure that the action preceding X should be connected to the *beginning* of the recursion, i.e. the initiating participant of I. For this to happen, we first annotate X with A, by which we can statically check its preceding event happens to A; then we demand I', the body of recursion, does indeed start from A. This justifies the participant annotation on recursion variables.

Below and henceforth we assume well-typedness (i.e. we assume $I_{1,2}$ are well-typed, I_1 is typed under a base including Y^B, and that substitution$[I_2/Y^B]$ is well-typed).

LEMMA 4 (Substitution). *Let I_1 and I_2 be strongly connected and $\mathsf{top}(I_2) = \{B\}$. Then $I_1[I_2/Y^B]$ is strongly connected and $\mathsf{top}(I_1) = \mathsf{top}(I_1[I_2/Y^B])$.*

Proof. By induction on the structure of I_1.

- Induction base. *We do case analysis of I_1.*
  - **0**. Immediate.
  - X^A. Immediate if $X \neq Y$; by induction if $X^A = Y^B$.
- Inductive cases. *We again do case analysis of I_1.*
  - $A \to B : ch(\nu\, \tilde{s}) . I$. By induction hypothesis, $I[I_2/Y^B]$ is strongly connected and $\mathsf{top}(I[I_2/Y^B]) = \mathsf{top}(I)$. Hence $I_1[I_2/Y^B]$ is strongly connected. Since $\mathsf{top}(I_1[I_2/Y^B]) = \{A\} = \mathsf{top}(I_1)$ we are done.
  - $A \to B : s\langle \mathsf{op},\ e,\ y\rangle . I$, $x@A := e . I$, $I \mid I'$, $\mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2$, $I_1 + I_2$, $(\nu\, s)\ I$. All immediate from induction hypothesis.
  - $\mathbf{rec}\, X^A . I$. By the bound name convention we can assume $X \neq Y$. Hence $(\mathbf{rec}\, X^A . I)[I_2/Y^B] = \mathbf{rec}\, X^A .(I[I_2/Y^B])$. Then we use induction hypothesis. □

LEMMA 5 (Subject Congruence: Strong Connectedness). *Let I_1 and I_2 be two interactions. If $I_1 \equiv I_2$ and I_1 is strongly connected then I_2 is strongly connected.*

Proof. We can show that this holds for all cases:

- $(\nu\, s)\ I \mid I' \equiv (\nu\, s)\ (I \mid I')$ (if $s_i \notin fn(I')$). Trivial.
- $(\nu\, s)\ (\nu\, s')\ I = (\nu\, s')\ (\nu\, s)\ I$. Trivial. □

THEOREM 3 (Subject Reduction: Strong Connectedness). *Let I be strongly connected and σ be well-typed. Then $(\sigma, I) \to (\sigma', I')$ implies I' is strongly connected.*

Proof. By induction on the reduction rules.

- (INIT). In this case we have that $(\sigma, A \to B : ch(\nu\, \tilde{s}) . I') \to (\sigma, (\nu\, \tilde{s})\ I'')$ and by definition of strong connectedness we have that it is connected whenever I'' is strongly connected and $\mathsf{top}(I'') = B$. Moreover, $(\nu\, \tilde{s})\ I''$ is strongly connected whenever I'' is strongly connected which concludes this case.

- (COMM). By applying the rule, we get $(\sigma, A \rightarrow B : s\langle \mathsf{op},\ e,\ x\rangle\, .\, I) \rightarrow (\sigma', I)$ if and only if $\sigma \vdash e@A \Downarrow v$. By definition of strong connectedness I is strongly connected.
- (ASSIGN). This rule states that $(\sigma, x@A := e\, .\, I') \rightarrow (\sigma', I')$. By definition of strong connectedness we have that I' is strongly connected.
- (IFTRUE) and (IFFALSE). We have that $(\sigma, \mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2) \rightarrow (\sigma, I)$ and by definition of strong connectedness we have that $I = I_i$ is strongly connected.
- (PAR). We have $(\sigma, I_1 \mid I_2) \rightarrow (\sigma', I_1' \mid I_2)$ inferred from $(\sigma, I_1) \rightarrow (\sigma', I_1')$. By definition of strong connectedness I_1 and I_2 are strongly connected. By induction hypothesis I_1' is strongly connected. Again by the definition of strong connectedness $I' = I_1' \mid I_2$ is strongly connected.
- (RES). Immediate from induction hypothesis and definition of strong connectedness, as above.
- (REC). We have $(\sigma, \mathbf{rec}\, X.I) \rightarrow I'$ from $(\sigma, I[(\mathbf{rec}\, X.I)/X] \rightarrow I'$. By Lemma 4, $I[(\mathbf{rec}\, X.I)/X]$ is strongly connected. By induction hypothesis so is I', done.
- (STRUCT). By Lemma 5. □

Strong connectedness (as well as its variants) imposes a strong structural constraint on the shape of interactions. One such consequence is the following observation. Intuitively it says that, in each thread of interactions, there is always one single participant ready to perform any operation that is not an input; while the remaining participants are waiting for input. At any stage of a thread of activity, there is only one participant performing any operation but and input. On the contrary, the rest of the other participant are all performing an input.

DEFINITION 9 (Input-Output Form). *Let $\Gamma \vdash I \triangleright \Delta$, assume that I structurally equivalent to $\Sigma_i A \rightarrow B : s\langle \mathsf{op_i},\ e_i,\ x_i\rangle\, .\, I_i$ or $A \rightarrow B : ch(\nu\, s)\, .\, I'$ and consider the tree generated by unfolding recursion occurrences. I is in* input-output form *whenever for all $C \neq A$, C occurs in each path (towards the leaves) first as a receiver, then zero or more ifthenelse, assignments and then as an output.*

LEMMA 6. *If I is structurally equivalent to $\Sigma_i A \rightarrow B : s\langle \mathsf{op_i},\ e_i,\ x_i\rangle\, .\, I_i$ or $A \rightarrow B : ch(\nu\, s)\, .\, I'$ and, moreover, it it is strongly connected, then I is always in input-output form.*

Proof. Direct from the definition of strong connectedness.
We shall use this observation during our next analysis, in which we extract true units of activity from a global description.

13.2. Further Examination of Connectedness (1): Input and Output Asymmetry. In strong connectedness, we regard only a sending action to be "initiating". Some observations on this point follows.

First, for session initiation actions, this is a natural choice. The typing of the end-point calculus is based on the idea that a service channel should always be available: in such a setting, the only feasible choice for guaranteeing the sequencing as specified in a global description is to use only a sending party as the one who does an action.

Second, for in-session communication actions, we can indeed swap the inputting party and outputting party as an "initiator" of sequencing, at least theoretically. For example, compare the following two interactions. The first one is strongly connected:

$$\begin{array}{l} x@A := 3. \\ A \rightarrow B : s\langle \mathsf{op}\rangle. \\ \cdots \end{array} \tag{75}$$

while the second one uses the reverse sequencing.

$$\begin{array}{l} x@A := 3. \\ B \rightarrow A : s\langle \mathsf{op}\rangle. \\ \cdots \end{array} \tag{76}$$

By a close look at (75), we observe the following assumption:

> In the second action, B should already be ready to receive at s; while A will
> *just at this second step* does the sending action.

Note that, in this assumption, we are *not* demanding a strict sequencing in the inputting side: rather it is in the outputting party which takes responsibility for the timing of this communication action. It is not feasible to demand both parties should make ready their complementary actions at the same time.

If we are to allow (76) and to have local processes obey the described sequencing, the assumption would be:

> In the second action, A should already be ready to send (or have sent) at s;
> while B will *just at this second step* become ready to receive an action.

Note this argument for "sequencing by input" holds even in the context of asynchronous communication (either the pure one or the one with arrival order nondeterminism). However (76) is *not* a good discipline, simply because, when a participant is sending, it should first create a datum: and this may as well be done as the result of the preceding event at the sender side, not at the receiver's side. From this viewpoint, (76) neglects a hidden causality principle for message creation, and may not be a practical choice.

These arguments suggest our assumption that it is a sender rather than a receiver who realises a sequencing is a natural idea. We next discuss two basic variants of connectedness based on this understanding of sequencing.

13.3. Further Examination of Connectedness (2): Variants of Connectedness.

r-Strong Connectedness. *Strong connectedness* is robust with respect to asynchrony of messages, i.e.even if we assume all messages are sent asynchronously in end-point processes, the principle still guarantees strict sequencing. Strong connectedness however is often too strict. For example, consider the following description:

$$\begin{array}{l}\mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathsf{QuoteCh}(\boldsymbol{\nu}\,\mathsf{s}).\\ \mathsf{Buyer} \rightarrow \mathsf{Seller} : s\langle \mathsf{RequestQuote},\ \mathit{productName},\ x\rangle.\\ \mathsf{Seller} \rightarrow \mathsf{Buyer} : s\langle \mathsf{ReplyQuote},\ \mathit{productPrice},\ y\rangle.\mathbf{0}\end{array} \tag{77}$$

Here a Buyer requests a Seller to start a session through a service channel QuoteCh, exchanging a fresh session channel s. Through s, the Buyer request a quote with a product name. The Seller then replies with the corresponding product price.[9]

Sending multiple consecutive messages from one party to another in a session is often found in practice (in both business and security protocols). Further (77) may not violate the essential idea of strong connectedness both logically and in implementation: first, it is still a reception of a message which acts as a trigger of an event in a different participant. Second, we can always send such consecutive messages in one go, so that it still works in the infrastructure which implements each message flow by asynchronous messaging (note if we send these consecutive messages separately, we need to guarantee the order of messages in some way, for which purpose we may use a widely used transport level protocol such as TCP). We call a refinement of strong connectedness which allows such consecutive interactions from the same sender to the same receiver, **strong connectedness relative to repetition**, or **r-strong connectedness**. We give its formal definition below for reference.

DEFINITION 10. We say *I starts from an action from A to B* when I is prefixed with a session initiation from A to B or a communication from A to B.

DEFINITION 11 (r-strong connectedness). labeldef:r:strongconnectedness The set of *r-strong connected interactions* are inductively generated as follows.

(1) $A \rightarrow B : ch(\boldsymbol{\nu}\,s).I'$ is r-strongly connected when I' is r-strongly connected and either $\mathsf{top}(I') = \{B\}$ or I' starts from an action from A to B.

[9] In practice, one may as well describe the initial "session initiation" action and the first RequestQuote action as one action, as in WS-CDL. One may as well consider (77) as a representation of this idiom in a formal setting.

(2) $A \rightarrow B : s\langle \mathsf{op},\ e,\ x\rangle . I$ is r-strongly connected when I is r-strongly connected and either $\mathsf{top}(I_i) = \{B\}$ or I is prefixed by an action from A to B.

For other terms we use the same clauses as in Definition 8, replacing "strong connectedness" with "r-strong connectedness".

One may note all relative strong connected interactions can be encoded into strong connected interactions. For example, (78) can be translated into:

$$\text{(78)} \quad \begin{array}{l} \mathsf{Buyer} \rightarrow \mathsf{Seller} : \mathsf{QuoteCh}(\boldsymbol{\nu}\, \mathsf{s}). \\ \mathsf{Seller} \rightarrow \mathsf{Buyer} : s\langle \mathsf{Ack}\rangle. \\ \mathsf{Buyer} \rightarrow \mathsf{Seller} : s\langle \mathsf{RequestQuote},\ \mathit{productName},\ x\rangle. \\ \mathsf{Seller} \rightarrow \mathsf{Buyer} : s\langle \mathsf{ReplyQuote},\ \mathit{productPrice},\ y\rangle.\mathbf{0} \end{array}$$

Thus we only have to add one ack between two consecutive actions in the same directions. For this reason, in all technical developments which depend on strong connectedness, we can equally use r-strong connectedness without any change in essential arguments. In particular, the same soundness and completeness results for the endpoint projection hold.

Connectedness. We can further loosen relative strong connectedness. For one thing, one may consider the following description is a natural one.

$$\text{(79)} \quad \begin{array}{l} \mathsf{Broker} \rightarrow \mathsf{Seller} : \mathsf{SellerCh}(\boldsymbol{\nu}\, \mathsf{s}). \\ \mathsf{Broker} \rightarrow \mathsf{Buyer} : \mathsf{BuyerCh}(\boldsymbol{\nu}\, \mathsf{f}', \mathsf{s}'). \\ \mathsf{Broker} \rightarrow \mathsf{Seller} : s\langle \mathsf{RequestQuote},\ \mathit{productName},\ x\rangle. \\ \mathsf{Broker} \rightarrow \mathsf{Buyer} : s'\langle \mathsf{RequestQuote},\ \mathit{productName},\ y\rangle. \\ \mathsf{Seller} \rightarrow \mathsf{Broker} : s\langle \mathsf{ReplyQuote},\ \mathit{productPrice},\ z\rangle \ldots\ldots \end{array}$$

Here Broker does four consecutive actions which are targeted to two different participants. Further this global description specifies, in the fifth line, that a Seller replies to a Buyer even though the immediately preceding action goes to the Buyer. However, it is natural and easy to consider that Seller can send its message after the third line, and this is received by Broker in the fifth line. The description still obeys a locality principle, which is directly realisable in synchronous communication. It is also easy to realise this idea in asynchronous communication as far as message sending order for each target is preserved (if message order is not preserved even for the same participant, we may still be able to group messages and send them again in one go up to a permutation, even though this becomes complicated if there is a branching, which is somewhat similar to permutation of instructions in pipelining in modern CPUs).

This principle, which we simply call **connectedness**, can be formalised by accumulating potential initiating participants one by one. For example, in the first line, it may well be the case that Broker is the only potential initiating participant. After the first line, Seller joins. After the second line, Buyer further joins. So in the fifth line, Seller can indeed invoke an interaction. Simply connected interactions again allow the parallel technical development, even though operational correspondence needs adjustment.

This relaxed variant of connectedness has one issue in that *sequencing in a global action may show false dependency* when projected onto local behaviour. This means, among others, connected but not r-strong connected descriptions are in general not well-threaded in the sense we shall discuss later. In spite this observation, we strongly believe this relaxed version of connectedness will have a basic role as a structuring principle of global descriptions, on which we are intending to explore elsewhere.

Other Constructs. By introducing other syntactic constructs such as join operation, the notion of connectedness can further be refined. As far as such a variant imposes a reasonable constraint following a locality principle of actions, we believe the corresponding principle can be used as a sound substrate for the essentially equivalent technical development we shall discuss in the subsequent subsections.

14. Theory of End-Point Projection (2): Well-Threadedness

14.1. Service Channel Principle. With strong connectedness, each interaction is a direct consequence of the preceding local event. On this basis, a finer analysis of interaction is possible, which allows us to extract a unit of behaviour acting in a global description. This unit is called *thread*, which plays a pivotal role in the present theory of endpoint projection.

Before introducing the notion of threads, we first illustrate one subtle point in the way service channels (which act as initiating points of sessions) are represented in the end-point calculus using an example. Consider the following global description:

(80)
$$\begin{array}{l} A \rightarrow B : ch_B(\boldsymbol{\nu} s)\,. \\ \quad B \rightarrow A : ch_A(\boldsymbol{\nu} t)\,. \\ \qquad A \rightarrow B : t\langle \mathsf{op}_1,\ v_1,\ x\rangle\,. \\ \qquad\quad B \rightarrow A : s\langle \mathsf{op}_2,\ v_2,\ y\rangle\,.\mathbf{0} \end{array}$$

First we have A asking B for service (session) ch_B, then B asking A for service ch_A, then A replying to B with a value on session name t (belonging to session ch_A) and finally B sending to A a value using session name s (belonging to ch_B). Now consider the following naive implementation of the interaction above as communicating local processes, focussing on A.

(81)
$$A[\ \ \overline{ch_B}(\boldsymbol{\nu} s)\,.!ch_A(t)\,.\bar{t} \lhd \mathsf{op}_1\langle v_1\rangle\,.s \rhd \mathsf{op}_2\langle y\rangle.\mathbf{0}\ \ ]_{\sigma_A}$$

The local description (81) directly translates A's portion of (80), where A first asks B for service via ch_B, then waits for somebody (here B) to ask for its own service ch_A, then sends a value to B over t, and finally waits for a value to be sent over s. Is this a faithful way to represent the behavioural content of (81)?

Suppose another client wishes to use a service available at ch_A. The projected behaviour (81) indicates that this service at ch_A becomes available only when A finishes an interaction at ch_A, which makes availability of service at Ch_A dependent on A's action (the issue becomes worse if A waits for B's reply before offering ch_A).

Generally, in our formalism and in web-service languages such as WS-CDL, a channel used for initiating protocols (*service channels* in our formalism, initial channels for starting choreographies/subchoreographies in WS-CDL, which may as well be public URLs) are intended to be repeatedly invokable and be always available to those who know the port names. In fact, in the standard practice of web services, a service is embodied by a shared channels in the form of URLs or URIs through which many users can throw their requests at any time (such availability at shared ports is maintained as part of the standard notion of "service" in a service-based framework going beyond web service). This is why the construction of services as found in (81) looks unnatural: a service channel should always be available to clients who know its URL. This may be called **service channel principle**.

In the engineering context, a basic form of service channel principle can be found in RPC and RMI, and its web-service embodiment such as SOAP. In the context of the π-calculus, this notion is representable as a replicated input who is "receptive" (or "input ready"). We can easily enforce a more refined discipline so that we can guarantee input service channels to be never under prefix in the typing for the end-point calculus. If we do so, (81) becomes untypable.

We now present the local representation of (80) which indeed obeys the service channel principle. First we have the following local code for A:

(82)
$$A[\ \ !\,ch_A(t).\bar{t} \lhd \mathsf{op}_1\langle v_1\rangle\,.\mathbf{0}\ \mid\ \overline{ch_B}(\boldsymbol{\nu} s)\,.s \rhd \mathsf{op}_2\langle y\rangle\,.\mathbf{0}\ \ ]_{\sigma_A}$$

For B we have:

(83)
$$B[\ \ !\,ch_B(s)\,.\overline{ch_A}(\boldsymbol{\nu} t)\,.t \rhd \mathsf{op}_1\langle x\rangle\,.\bar{s} \lhd \mathsf{op}_2\langle v_2\rangle\,.\mathbf{0}\ \ ]_{\sigma_B}$$

By tracing reductions of the parallel composition of (82) and (83), we can check the interaction does proceed faithfully following (80).

14.2. Motivation: False Causality in Global Description. We are now ready to illustrate the notions of threads and well-threadedness. Consider the following global description:

$$
\begin{array}{l}
A \to B : ch_B(\nu s) . \\
\quad B \to C : ch_C(\nu t) . \\
\quad\quad C \to A : ch_A(\nu u) . \\
\quad\quad\quad A \to B : s\langle \mathsf{op},\ v,\ x\rangle . I .
\end{array} \tag{84}
$$

Note the description is strongly connected. However we claim this description is not well-structured, and is impossible to be faithfully realised as reasonable end-point processes.

Let us examine the behaviour of A described in (84). Following the service channel principle, we can observe the behaviour of A has two different chunks of code, which we (first informally) call threads. The first thread starts a fresh session by invoking ch_B in B, and sends a value to B over s. The other thread is the one which provides the service via ch_A (which may be realised in I). Thus the local behaviour may be represented as:

$$A[\ \ !ch_A(\nu t) . P_A \ \mid\ \overline{ch_B}(\nu s) . \overline{s} \lhd \mathsf{op}\langle v\rangle . Q_A\ \ ]_{\sigma_A} \tag{85}$$

In the same way, we may consider the following local implementation of B.

$$B[\ \ ch_B(s) . \overline{ch_C}(\nu t) . t \rhd \mathsf{op}\langle x\rangle . P_B\ \ ]_{\sigma_B} \tag{86}$$

Finally, let us consider C's end-point view:

$$C[\ \ ch_C(t) . \overline{ch_A}(\nu u) . P_C\ \ ]_{\sigma_C} \tag{87}$$

Let us now see how these process interact. After B asks C for service ch_C, the process $s \rhd \mathsf{op}\langle x\rangle . P_B$ is free to react with the term $\overline{s} \lhd \mathsf{op}\langle v\rangle . P_{A2}$ in A, even before C has interacted with A's other component.

Can we change the local behaviours (85, 86, 87) so that it can precisely represent the original global behaviour (84)? We reason as follows.

(1) The service channel principle says that the channel ch_A is replicated and is ready to receive an invocation.
(2) Now the session channel s is initiated by a thread at A **which is not under** ch_A (since if it *is* under ch_A, how can it be the initial move?).
(3) But for an action at s to take place *immediately after invocation at ch_A, it should be under* ch_A, a contradiction.

Thus we conclude *it is impossible to impose the global sequencing stipulated (84) by well-typed local behaviours.* This means (84) describes a **false dependency (sequentialisation) among actions** which cannot be realised by well-typed local interactions. This examples motivates the main theme of this section, the descriptive principle called **well-threadedness**, which automatically prevents such false dependency from appearing in global description. We introduce this notion formally in the next two subsections.

14.3. Annotating Interactions with Threads. Let us come back to the first global description (80), which we found to be realisable by end-point processes given in (82) and (83). Let us analyse these few lines of global description (80) informally, reproduced below.

$$
\begin{array}{ll}
(\mathbf{1}) & A \to B : ch_B(\nu s) . \\
(\mathbf{2}) & B \to A : ch_A(\nu t) . \\
(\mathbf{3}) & A \to B : t\langle \mathsf{op}_1,\ v_1,\ x\rangle . \\
(\mathbf{4}) & B \to A : s\langle \mathsf{op}_2,\ v_2,\ y\rangle . \\
(\mathbf{5}) & \mathbf{0}
\end{array}
$$

The initial interaction in Line **(1)** is initiated by A, which is an output (session initiation) at B's service channel ch_B: dually the interaction is an input (reception of a session initiation) for B at ch_B. Then in Line **(2)**, B *reacts* by an interaction, again with A, but which is now an output for B and which is a *session initiation* at A's service channel ch_A. Dually it is an *input action* for A, receiving a session initiation at its own ch_A. For A, this input is done *independently from the initial output action* in (1).

At this point we realise that, in **(2)** above, *because B's output action is a reaction to its own previous input action*, the former and the latter should be in the same "code": we call such a causally connected sequence of actions of the same participant, a **thread**. Up to **(2)**, we have the following three threads.

Thread 1: which is in A, containing its invocation at ch_B, opening a channel s.

Thread 2: which is in B, containing its reception of the invocation above (opening s) and its subsequent invocation at ch_A (opening t).

Thread 3: which is in A, containing a reception of invocation at ch_A (opening t).

Note Thread 1 and Thread 3 are *separate threads*: whenever a new invocation of a service (or a new session initiation) is done, this creates a new thread at the receiving, or service, side.

We continue our analysis. In Line **(3)**, the output of A is reaction to its previous input, so it is in the same thread as the latter, i.e. Thread 3. Since it uses the session channel t opened by A in Thread 3 (in Line **(2)**), this action should be in Thread 3. The same interaction at the third line is an input for B, which should be Thread 2, because it uses t opened in the initial action of Thread 2. In Line **(4)**, B reacts by an output action at s. Since this is opened in its Thread 2, we know this action by B should be in Thread 2. Similarly, the dual input action by A should be in Thread 1 since s is opened in Thread 1 for A. In Line **(5)**, we have no more interaction, concluding the analysis.

As a summary, there are three threads as a whole, two for A and one for B. In A, we have one thread (Thread 1) starting from an output and another (Thread 3) which is a "service" starting from input at service channel ch_A: this is precisely the processes given in (82), reproduced below:

$$A[\ \ !ch_A(t).\bar{t} \lhd \mathsf{op}_1\langle v_1\rangle\,.\,\mathbf{0}\ \mid\ \overline{ch_B}(\boldsymbol{\nu}\boldsymbol{\nu}s)\,.\,s \rhd \mathsf{op}_2\langle y\rangle\,.\,\mathbf{0}\ \ ]_{\sigma_A}$$

Similarly there is one service in B, Thread 2, as given in (83), reproduced below:

$$B[\ \ !ch_B(s)\,.\,\overline{ch_A}(\boldsymbol{\nu}t)\,.\,t \rhd \mathsf{op}_1\langle x\rangle\,.\,\bar{s} \lhd \mathsf{op}_2\langle v_2\rangle\,.\,\mathbf{0}\ \ ]_{\sigma_B}$$

Thus extracting "threads" (in an informal sense) from a global description has led to obtaining local behaviours which faithfully realise it. The analysis of local causality based on threads based on session types is the main focus of the following discussion. We first start from annotating a global interaction with a notion of threads.

DEFINITION 12 (Annotated Interaction). Thread annotated interactions*, or simply* annotated interactions*, written $\mathcal{A}, \mathcal{A}', \ldots$, are given by the following grammar.*

$$\begin{array}{rcl}
\mathcal{A} & ::= & A^{\tau_1} \to B^{\tau_2} : ch(\tilde{s})\,.\,\mathcal{A} \\
& \mid & A^{\tau_1} \to B^{\tau_2} : s\langle \mathsf{op},\ e,\ y\rangle\,.\,\mathcal{A} \\
& \mid & x@A^{\tau} := e\,.\,\mathcal{A} \\
& \mid & \textit{if}\ e@A^{\tau}\ \textit{then}\ \mathcal{A}_1\ \textit{else}\ \mathcal{A}_2 \\
& \mid & \mathcal{A}_1 +^{\tau} \mathcal{A}_2 \\
& \mid & \mathcal{A}_1 \mid^{\tau} \mathcal{A}_2 \\
& \mid & X^A_\tau \\
& \mid & \mathbf{rec}\ {}^{\tau}X^A\,.\,\mathcal{A} \\
& \mid & \mathbf{0}
\end{array}$$

where each τ_i is a natural number. We call $\tau, \tau', \ldots$ occurring in an annotated interaction, threads*. Further the grammar of* extended annotated interactions*, $\mathcal{E}, \mathcal{E}, \ldots$, are given as:*

$$\mathcal{E} \quad ::= \quad \mathcal{A} \quad \mid \quad \mathcal{E}_1 | \mathcal{E}_2 \quad \mid \quad (\boldsymbol{\nu}s)\mathcal{E}$$

Hereafter we let $\mathcal{A}, \mathcal{A}', \ldots$ range over extended annotations by abuse of notations, and simply call them annotated interactions*.*

REMARK 5. If parallel composition is under prefix, both should anyway share the same initiating participant: thus only top-level threads may have different threads, which extended annotated threads represent. For restriction, note it can only occur outside of prefixes by Convention 1.

An annotated interaction annotates each node of an abstract syntax tree of a term with threads, which are given as natural numbers. For example, (80) is annotated as, following our previous analysis:

$$\begin{array}{l} A^1 \to B^2 : ch_B(s) . \\ \quad B^2 \to A^3 : ch_A(t) . \\ \qquad A^3 \to B^2 : t\langle \mathsf{op}_1,\, v_1,\, x\rangle . \\ \qquad\quad B^2 \to A^1 : s\langle \mathsf{op}_2,\, v_2,\, y\rangle . \mathbf{0} \end{array}$$

But we can also annotate the same global interaction with an inconsistent annotation:

$$\begin{array}{l} A^1 \to B^1 : ch_B(s) . \\ \quad B^2 \to A^1 : ch_A(t) . \\ \qquad \ldots \end{array}$$

which does not make sense.

14.4. Well-Threadedness. As we have just seen, a thread annotation may or may not make sense. How can we find a consistent way to annotate an interaction with threads? By our previous analysis, we need to stipulate whether the causality specified globally can be precisely realisable locally.

We fix some terminology. Regarding each $\mathcal{A}$ as an abstract syntax tree, it has a *constructor* at its root (say prefix or parallel composition), which is annotated by either one thread or, if it is initiation or communication, an ordered pair of threads (the first for sender the second the receiver). Above the constructor, it has its *direct subtree(s)*, each of which is another such abstract syntax tree. Each (possibly indirect) subtree of $\mathcal{A}$ is *dominated* by each of its (direct and indirect) proper subtrees.

DEFINITION 13 (Basic Terminology for Threads).

(1) If the root of $\mathcal{A}$ is initialisation/communication from B to C and is annotated by (τ_1, τ_2), then τ_1 (resp. τ_2) is the *active thread of $\mathcal{A}$ by B* (resp. the *passive thread of $\mathcal{A}$ by C*). If the root of $\mathcal{A}$ is other constructors, then its annotation τ is both its active thread and its passive thread.

(2) If $\mathcal{A}'$ occurs as a proper subtree of $\mathcal{A}$, then (the root of) $\mathcal{A}$ is a *predecessor* of (the root of) $\mathcal{A}'$. Symmetrically we define *successor*. A *direct predecessor/successor* is a predecessor/successor which does not have no intermediate predecessor/successor.

Note if the root of $\mathcal{A}$ is a predecessor of that of $\mathcal{A}'$, then the former's execution should indeed temporarily precedes that of the latter.

DEFINITION 14 (Consistent Thread Annotation). An annotated strongly connected interaction $\mathcal{A}$ is *globally consistent* if the following conditions hold for each of its subtrees, say $\mathcal{A}'$.

(G1) Distinctness/Freshness Condition: If τ is by A at some node and by B at another node then A and B always coincide. Further If $\mathcal{A}'$ starts with an initialisation, then its passive thread should be fresh w.r.t. all of its predecessors (if any).

(G2) Session Consistency: If $\mathcal{A}'$ starts with a communication between B and C via (say) s and another subtree $\mathcal{A}''$ of $\mathcal{A}$ starts with a communication via s or an initialisation which opens s, then the thread by B (resp. by C) of $\mathcal{A}'$ should coincide with the thread by B (resp. by C) of $\mathcal{A}''$.

(G3) Causal Consistency: If $\mathcal{A}''$ is the direct successor of $\mathcal{A}'$, then the active thread of $\mathcal{A}''$ should coincide with the passive thread of $\mathcal{A}'$.

A thread annotated interaction is *locally consistent* if it is globally consistent and if the following conditions hold for each of its (possibly indirect) subtrees $\mathcal{A}$.

(L) Local Causal Consistency: Suppose $\mathcal{A}'$ is a supertree of $\mathcal{A}$ and $\mathcal{A}$ is an initialisation or a communication, similarly for $\mathcal{A}'$. If both contain τ and, moreover, $\mathcal{A}$ is the first such subtree of $\mathcal{A}'$, then if τ is passive by (say) B then τ is active by B and vice versa.

(G1) says that a thread should always belong to one participant, and that a fresh thread starts when a service is invoked. **(G2)** says two distinct interactions in the same session (which are, by typing, always between the same pair of participants) should be given the same threads w.r.t. each participant, note if Δ is present this gives a grouping of free session channels which will later be abstracted by an initialisation action: note also, by Proposition 2, the grouping of session channels in Δ is arbitrary). **(G3)** says if A has an input annotated as a (passive) thread then its immediately following output should be annotated by the same (but this time active) thread.[10]

(L) is a significant condition which is about local causality. It says that, within the same thread going through an interaction, a participant acts in a strictly alternating fashion in initialisation/communication actions.[11] To illustrate this condition, let us go back to our initial example:

$$\begin{array}{l} A \to B : ch_B(s)\,. \\ B \to A : ch_A(t)\,. \\ A \to B : t\langle \mathsf{op}_1,\ v_1,\ x\rangle\,. \\ B \to A : s\langle \mathsf{op}_2,\ v_2,\ y\rangle\,.\mathbf{0}. \end{array}$$

We notice that it works just because for each session, each flow of information from one participant to another is always followed, if any, by an opposite flow of information, e.g. A starts session ch_B with name s then B replies to A on s. If not, the causality depicted in the global description can never be realised locally. Thus **(L)** embodies the condition which is the key to local realisability of causality in a global description. Somewhat surprisingly, global consistency implies local consistency.[12]

PROPOSITION 10. *If $\mathcal{A}$ is globally consistent, then it is also locally consistent.*

Proof. (outline) Suppose there are two separate inputs by B annotated by the same thread τ and for which there are no intermediate actions annotated by τ (that is, we have two consecutive inputs within the same thread which are temporarily separated). But this is impossible since immediately *after* the first passive τ, this should lead to its active occurrence in the direct subtree, which contradicts our assumption. Symmetrically suppose there are two separate outputs by B annotated by the same thread τ and for which there are no intermediate actions annotated by τ (that is, we have two consecutive outputs within the same thread which are temporarily separated). But this is impossible since immediately *before* the second active τ, this should be preceded by its passive occurrence in the direct supertree. □

DEFINITION 15. We say $\mathcal{A}$ is *consistent* if it is globally consistent or, equivalently, if it is locally consistent.

We can now define well-threadedness. Below we say *$\mathcal{A}$ is an annotation of I* when the result of stripping off annotations from $\mathcal{A}$ coincides with I.

DEFINITION 16 (Well-Threaded Interactions). A strongly connected term I is *well-threaded* when there is an annotation $\mathcal{A}$ of I which is consistent.

Note well-threadedness implies strong connectedness (hence well-typedness). In the next subsection we introduce the type discipline which type all and only well-threaded interactions, via consistent global and local annotation.

REMARK 6 (Annotations following Session Typing). *It is natural to consider a consistent annotation which conforms to the grouping of session channels by the session typing. Suppose $\Gamma \vdash I \rhd \Delta$ and $\mathcal{A}$ is an annotation of I. Then we may demand, in addition to* **(G1..3)***, the following extension of* **(G2)***:* whenever two communication actions by the same participant occur via respectively s_1 and s_2 and if s_1 and s_2 belong to the same type assignment in Δ, then these two actions should be given the same thread.

[10] If we are to work with r-strong connectedness in Section 13.3, then (G3) should be refined so that if two consecutive A to B actions are given they should be annotated by the same threads.

[11] If we are to work with r-strong connectedness in Section 13.3, then **(L)** should be refined so that we treat consecutive A to B actions as a single chunk.

[12] This result is related with what is called “switching condition” in game-based semantics.

14.5. Examples of Well-Threaded Interactions. It is important to understand now what is the connection between an interaction and its annotation. In order to give a sound and deterministic correspondence, we define a function which annotates interactions. We now try to explain the rules of the typing system. Consider the following interaction

$$\begin{array}{l} A \to B : ch_B(s)\,. \\ \quad B \to A : s\langle \mathsf{op},\ e,\ x\rangle\,. \\ \quad\quad (A \to C : ch_C(t) \mid A \to B : s\langle \mathsf{op},\ e,\ x\rangle) \end{array}$$

If we now consider its implementation in the local calculus according to our discussion above, we would get for some σ

$$\begin{array}{l} A[\,ch_B\langle s\rangle\,.\,s \rhd op(x)\,.\ \ (\ ch_C\langle t\rangle\ \mid\ s \lhd op\langle e\rangle\ )\,]_{\sigma_A} \\ \qquad\qquad B[\,!\,ch_B(s)\,.\,s \lhd op\langle e\rangle\,.\,s \rhd op(x)\,]_{\sigma_B} \\ \qquad\qquad\qquad\qquad C[\,!\,ch_C(t)\,]_{\sigma_C} \end{array}$$

If we now start talking about threads we notice that going through each action we must take a choice whether to start a new thread or continue a previous one. According to the translation we gave into the end-point calculus we can think about the following annotation:

$$\begin{array}{l} A^1 \to B^2 : ch_B(s)\,. \\ \quad B^2 \to A^1 : s\langle \mathsf{op},\ e,\ x\rangle\,. \\ \quad\quad (A^1 \to C^3 : ch_C(t) \mid^1 A^1 \to B^2 : s\langle \mathsf{op},\ e,\ x\rangle) \end{array}$$

We now show another case where we also include recursion and the if then else construct. Consider the following interaction

$$\begin{array}{l} A \to B : ch_B(s)\,. \\ \quad \mathbf{rec}\ X^B\,.\,B \to A : s\langle \mathsf{op},\ e,\ x\rangle\,.\,X^B \end{array}$$

Without going into the details of a possible end-point representation, it is clear that here is a problem with the notion of well-threadedness. In fact, after A starts the session ch_B, B continuously sends e to A on session name s. This goes against our notion of well-threadedness. We can add a further interaction between A and B to remedy this issue:

$$\begin{array}{l} A \to B : ch_B(s)\,. \\ \quad \mathbf{rec}\ X^B\,.\,B \to A : s\langle \mathsf{op},\ e,\ x\rangle\,. \\ \quad\quad A \to B : s\langle \mathsf{op}',\ e',\ y\rangle\,.\,X^B \end{array}$$

The following annotated interaction, which is well-threaded, is an example of parallel composition.

$$\begin{array}{l} A^1 \to B^2 : ch_B(s_1,s_2)\,. \\ \quad (B^2 \to C^3 : ch_C(t)\,.\,C^3 \to B^2 : t\langle\ldots\rangle B^2 \to A^1 : s_1\langle\ldots\rangle \quad \mid^2 \\ \quad B^2 \to C^4 : ch'_C(t')\,.\,C^4 \to B^2 : t\langle\ldots\rangle B^2 \to A^1 : s_2\langle\ldots\rangle) \end{array}$$

Note that, after the thread 2 is passive in the initial action, this in turn becomes active in both components of parallel composition. In this way, a prefixed parallel composition should always have the common initiating thread. For a top-level parallel composition, this does not have to be:

$$\begin{array}{l} C^3 \to B^2 : t\langle\ldots\rangle B^2 \to A^1 : s_1\langle\ldots\rangle \quad \mid \\ \quad B^2 \to C^4 : ch'_C(t')\,.\,C^4 \to B^2 : t\langle\ldots\rangle B^2 \to A^1 : s_2\langle\ldots\rangle) \end{array}$$

which is the result of reducing one interaction from the previous one and which is still well-threaded.

$$(\text{WT-INIT})\ \frac{\Theta, \tau_1 : S \uplus \{\tilde{s}\}, \tau_2 : \{\tilde{s}\} \vdash \mathcal{A} \quad \mathsf{topT}(\mathcal{A}) = \tau_2}{\Theta,\ \tau_1 : S,\ \tau_2 : \emptyset \vdash A^{\tau_1} \rightarrow B^{\tau_2} : ch(\tilde{s}) . \mathcal{A}}$$

$$(\text{WT-COMM})\ \frac{\Theta, \tau_1 : S_1, \tau_2 : S_2 \vdash \mathcal{A} \quad \mathsf{topT}(\mathcal{A}) = \tau_2 \quad s \in S_1 \cap S_2}{\Theta, \tau_1 : S_1, \tau_2 : S_2\ \vdash A^{\tau_1} \rightarrow B^{\tau_2} : s\langle \mathsf{op},\ e,\ x \rangle . \mathcal{A}}$$

$$(\text{WT-ASSIGN})\ \frac{\Theta \vdash \mathcal{A} \quad \mathsf{topT}(\mathcal{A}) = \tau}{\Theta \vdash x@A^{\tau} := e . \mathcal{A}}$$

$$(\text{WT-IFTHENELSE})\ \frac{\Theta \vdash \mathcal{A}_i \quad \mathsf{topT}(\mathcal{A}_i) = \tau}{\Theta \vdash \mathsf{if}\ e@A^{\tau}\ \mathsf{then}\ \mathcal{A}_1\ \mathsf{else}\ \mathcal{A}_2}$$

$$(\text{WT-SUM})\ \frac{\Theta \vdash \mathcal{A}_i \quad \mathsf{topT}(\mathcal{A}_i) = \tau}{\Theta \vdash \mathcal{A}_1 +^{\tau} \mathcal{A}_2}$$

$$(\text{WT-PAR})\ \frac{\Theta \vdash \mathcal{A}_i \quad \mathsf{topT}(\mathcal{A}_i) = \tau \quad (i = 1, 2)}{\Theta \vdash \mathcal{A}_1 \mid^{\tau} \mathcal{A}_2}$$

$$(\text{WT-PAR-EXT})\ \frac{\Theta \vdash \mathcal{E}_i}{\Theta \vdash \mathcal{E}_1 \mid \mathcal{E}_2}$$

$$(\text{WT-RES-EXT})\ \frac{\Theta, \tau_1 : S_1 \uplus \{s\}, \tau_2 : S_2 \uplus \{s\} \vdash \mathcal{E}}{\Theta, \tau_1 : S_1, \tau_2 : S_2 \vdash ((\boldsymbol{\nu} s)\ )\mathcal{E}}$$

$$(\text{WT-VAR})\ \frac{\Theta \text{ well-formed}}{\Theta, X : \Theta \vdash X^{A}_{\tau}} \qquad (\text{WT-REC})\ \frac{\Theta, X : \Theta \vdash \mathcal{A} \quad \mathsf{topT}(\mathcal{A}) = \tau}{\Theta \vdash \mathbf{rec}\ {}^{\tau}X^{A} . \mathcal{A}}$$

$$(\text{WT-ZERO})\ \frac{\Theta \text{ well-formed}}{\Theta \vdash \mathbf{0}^{\tau}}$$

FIGURE 24. Typing Rules for checking Well-Threadedness.

14.6. Type Disciplines for Well-Threadedness (1). Given a well-typed, strongly connected annotated interaction, its well-threadedness can be compositionally checked using a typing system. We first present the typing system which checks **(G1–G3)**. Then we refine it so that it can validate **(L)**. Henceforth let $S, S', \ldots$ range over the finite sets of session channels.

$$\Theta \quad ::= \quad \Theta \cdot \tau : S \quad | \quad \Theta, X : \Theta \quad | \quad \emptyset$$

We assume Θ defines a function (with its domain the threads and term variables). Θ_1, Θ_2 indicates their union s.t. $\mathsf{dom}(\Theta_1) \cap \mathsf{dom}(\Theta_2) = \emptyset$. We say Θ is *well-formed* iff each session channel is assigned to at most two threads, i.e. iff whenever $\tau : S \in \Theta$ such that $s \in S$, there is at most one $\tau' \neq \tau$ such that $\tau' : S' \in \Theta$ such that $s \in S'$.

The typing judgement has the form $\Theta \vdash \mathcal{A}$, where Θ records free session channels used in each thread in $\mathcal{A}$ and is (inductively) well-formed.

DEFINITION 17 (Type Discipline for Well-Threadedness). *For an annotated strongly connected (hence well-typed) interaction $\mathcal{A}$, $\Theta \vdash \mathcal{A}$ is derived by the rules in Figure 24, where for convenience we annotate each inaction* $\mathbf{0}$ *with a thread, writing* $\mathbf{0}^{\tau}$*. The notation* $\mathsf{topT}(\mathcal{A})$ *returns the active thread of $\mathcal{A}$ for $\mathcal{A}$ which should be a non-extended annotated interaction in the sense of Definition 12 with* $\mathbf{0}$ *annotated as above (for which we set* $\mathsf{topT}(\mathbf{0}^{\tau}) = \tau$*).*

In the typing, well-formedness is inductively guaranteed so that, for well-threaded interactions, the required shape of the typing in the premise of (WT-INIT) and (WT-COMM) is always satisfied. In (WT-INIT), we place $\tau_2 : \emptyset$ so that no further threads can use τ_2. The two rules for parallel composition, (WT-PAR) and (WT-PAR-EXT), are in precise correspondence with the grammar of annotated interactions (cf. Definition 12: note however we shall still be writing $\mathcal{A}$ etc. from now on for extended annotated interactions). Only in (WT-PAR) we demand the initial active threads of the two components to be identical. The hiding (WT-PAR-EXT) is also only for extended annotated interactions. A basic property of the typing system follows.

THEOREM 4 (Soundness of WT-typing). *An extended annotated strongly connected interaction $\mathcal{A}$ is well-threaded if $\Theta \vdash \mathcal{A}$ for some Θ.*

Remark. Below and henceforth (IH) stands for induction hypothesis.

Proof. We prove the following stronger result, where we say a thread in $\mathcal{A}$ is *complete* if it occurs as a passive thread in session initialisation. It is *incomplete* if not.

Claim. If $\Theta \vdash \mathcal{A}$ then (1) Θ is well-formed; (2) $\mathcal{A}$ is consistent and each incomplete thread τ in $\mathcal{A}$ uses s iff $\tau : s$ is in Θ; and (3) for each complete thread, say τ, Θ contains $\tau : \emptyset$.

For (WT-INIT), (1) and (2) are direct from (IH). (3) is by (IH) and the shape of the rule, noting τ_2 got completed. For (WT-COMM), (1) and (3) are direct from (IH). (2) is ensured by (IH) and the condition in the premise. For each of (WT-ASSIGN), (WT-IFTHENELSE), (WT-SUM), (WT-PAR), (WT-EXT-PAR) and (WT-EXT-RES), all of (1..3) are direct from (IH) and, for **(G3)** of (2), by the corresponding condition in the premise. Finally (WT-VAR) and (WT-ZERO) are obvious. For (WT-REC), (1) is direct, (2) is by (IH) with **(G3)** being obvious, and (3) is again by (IH). □

Below we define $\mathcal{A} \equiv \mathcal{A}'$, $(\sigma, \mathcal{A}) \to (\sigma', \mathcal{A}')$ etc. exactly following the corresponding relations on unannotated interactions (cf. §11.3, Figure 21, page 57), except we demand top-level parallel compositions (i.e. those which are not under prefixes) are not annotated and, when a reduction creates a new top level parallel composition, its label should be taken off.

THEOREM 5 (Subject Reduction: Well-Threadedness). *Below $\mathcal{A}, \ldots$ include extended annotated interactions.*

(1) *If $\Theta \vdash \mathcal{A}$ and $\mathcal{A} \equiv \mathcal{A}'$ then $\Theta \vdash \mathcal{E}$.*
(2) *If $I \equiv I'$ and I has a consistent annotation then I' also has a consistent annotation.*
(3) *If $\Theta \vdash \mathcal{A}$ and $(\sigma, \mathcal{A}) \to (\sigma', \mathcal{A}')$ then $\Theta \vdash \mathcal{A}'$.*

Proof. (1) is easy rule induction (on the generation rules for $\equiv$). (2) is by (1), noting if $I \equiv I'$ and $\mathcal{A}$ is a consistent annotation of I then the same derivation witnessing $I \equiv I'$ leads to $\mathcal{A}'$ such that $\mathcal{A} \equiv \mathcal{A}'$. (3) is by rule induction on reduction rules. All are easy except we use, for induction for the reduction of recursion: $\Theta \vdash \textbf{rec}\ X.I$ implies $\Theta \vdash I[(\textbf{rec}\ X.I)/X]$ [because: if $\Theta \vdash \textbf{rec}\ X.I$, then $\Theta, X : \Theta \vdash I$, hence by using induction on I and thinning we are done]. □

To capture local consistency directly in the typing system, we refine the type discipline given in Figure 24. The refined system also offers a simple, inductive justification of Proposition 10. We first augment the typing Θ as follows:

$$\Theta \quad ::= \quad \Theta \cdot \tau : (\uparrow, S) \quad | \quad \Theta \cdot \tau : (\downarrow, S) \quad | \quad \Theta, X : \Theta \quad | \quad \emptyset$$

Above we add, for each thread, the direction of the latest action in that thread. The judgement has the same form except we use this refined typing. The operation $\Theta_1 \odot \Theta_2$ now combines the information on the direction, so that it is defined iff the directions coincide for each common thread: if this fails for any thread, the composition is undefined. We then replace (WT-Init) and (WT-Comm) as follows:

$$(\text{WT-Init})\ \frac{\Theta, \tau_1 : (\downarrow, S \uplus S'\}), \tau_2 : (\uparrow, S') \vdash \mathcal{A} \quad \mathsf{topT}(\mathcal{A}) = \tau_2 \quad S' \subseteq \{\tilde{s}\}}{\Theta, \tau_1 : (\uparrow, S) \vdash A^{\tau_1} \to B^{\tau_2} : ch(\tilde{s}) . \mathcal{A}}$$

$$(\text{WT-Comm})\ \frac{\Theta, \tau_1 : (\downarrow, S_1), \tau_2 : (\uparrow, S_2) \vdash \mathcal{A}_i \quad \mathsf{topT}(\mathcal{A}_i) = \tau_2 \quad J \neq \emptyset}{\Theta, \tau_1 : (\uparrow, S_1 \cup \{s\}), \tau_2 : (\downarrow, S_2 \cup \{s\}) \vdash A^{\tau_1} \to B^{\tau_2} : s\langle \mathsf{op},\ e,\ x\rangle . \mathcal{A}}$$

In both, the condition on the direction at τ_2 is non-trivial. After giving activity to another thread, when it comes back inside $\mathcal{A}$, the thread τ_2 always starts as an input. The remaining rules stay in the same shape (except we use the refined typings).

It is easy to check that the typability in this refined system directly entails (not only global consistency, by Theorem 4, but also) local consistency, by guaranteeing strict alternation in the direction of interactions for each thread. We can further show inductively that this refined system type checks precisely the same terms as the original system, establishing Proposition 10.

14.7. Type Disciplines for Well-Threadedness (2). We next consider the existence of "representative annotation" for a well-threaded interaction. We say a consistent annotation $\mathcal{A}_0$ on I is *minimal* if, up to renaming, any consistent annotation $\mathcal{A}$ of I arises by collapsing the annotations in $\mathcal{A}_0$. For example (and as a simplest possible case), consider the following well-threaded interaction.

$$\begin{array}{c}(B \to C : ch_C(t) . C \to B : t\langle \ldots \rangle B \to A : s_1\langle \ldots \rangle \quad | \\ B \to C : ch'_C(t') . C \to B : t\langle \ldots \rangle B \to A : s_2\langle \ldots \rangle)\end{array}$$

Then its minimal annotation is, up to injective renaming:

$$\begin{array}{c}(B^1 \to C^3 : ch_C(t) . C^3 \to B^1 : t\langle \ldots \rangle B^1 \to A^5 : s_1\langle \ldots \rangle \quad | \\ B^2 \to C^4 : ch'_C(t') . C^4 \to B^2 : t\langle \ldots \rangle B^2 \to A^1 : s_2\langle \ldots \rangle)\end{array}$$

By collapsing labels, we can reach:

$$\begin{array}{c}(B^2 \to C^3 : ch_C(t) . C^3 \to B^2 : t\langle \ldots \rangle B^2 \to A^5 : s_1\langle \ldots \rangle \quad | \\ B^2 \to C^4 : ch'_C(t') . C^4 \to B^2 : t\langle \ldots \rangle B^2 \to A^1 : s_2\langle \ldots \rangle)\end{array}$$

We can further collapse labels while being consistent:

$$\begin{array}{c}(B^2 \to C^3 : ch_C(t) . C^3 \to B^2 : t\langle \ldots \rangle B^2 \to A^1 : s_1\langle \ldots \rangle \quad | \\ B^2 \to C^4 : ch'_C(t') . C^4 \to B^2 : t\langle \ldots \rangle B^2 \to A^1 : s_2\langle \ldots \rangle)\end{array}$$

Formally the notion of minimal annotations is defined as follows.

DEFINITION 18 (Minimal Consistent Annotation). Let I be well-threaded. Then its consistent annotation $\mathcal{A}_0$ is *minimal* if for each consistent annotation $\mathcal{A}$ of I, there is a possibly non-injective substitution σ of thread labels such that $\mathcal{A} = \mathcal{A}_0\sigma$.

The type discipline which inductively constructs a minimal annotated interaction of I iff I is well-threaded, is given in Figure 25. We again use annotated inaction for convenience, this time with a participant name too, writing $\mathbf{0}^A_\tau$. The sequent has the shape $\Theta \vdash^{\mathsf{min}} \mathcal{A}$ where the typing Θ is a finite sequence of assignments each of the form $\tau : (A, S)$ where A is a participant and S is a set of session names. We demand Θ to be *well-formed* in the sense that:

(1) Θ induces a finite map from threads to such pairs, in the obvious sense. In this case we write e.g. $\Theta(\tau) = (A, S)$ when the function induced by Θ maps A to S.

(2) For each τ, if $\Theta(\tau) = (A, S)$ and $s \in S$, then there is at most one $\tau' \neq \tau$ such that $\Theta(\tau') = (A', S')$ and $s \in S'$. Moreover in this case $A \neq A'$.

(This well-formedness is automatically ensured inductively in the typing rules in Figure 25.)

These rules refine the rules in Figure 24 and inductively solve simple equational constraints. This constraint is distilled in the operator $\mathsf{com}(\Theta_1\, ; \Theta_2)$ used in these rules. This operator is defined as follows.

$$(\text{WT-MI}\textsc{nit})\ \frac{\begin{array}{c}\Theta,\tau_2:(B,S)\vdash^{\mathsf{min}}\mathcal{A}\quad \mathsf{topT}(\mathcal{A})=\tau_2\quad S_2\subset\{\tilde{s}\}\\ \mathsf{com}(\tau_1:(A,\{\tilde{s}\});\Theta)\ =\ \tau_1:(A,S'),\Theta\sigma\quad (\tau_1\text{ fresh})\end{array}}{\Theta\sigma,\tau_1:(A,S'\backslash\{\tilde{s}\}),\tau_2:(B,\emptyset)\vdash^{\mathsf{min}}A^{\tau_1}\rightarrow B^{\tau_2}:ch(\tilde{s}).(\mathcal{A}\sigma)}$$

$$(\text{WT-MC}\textsc{omm})\ \frac{\begin{array}{c}\Theta,\tau_2:(B,S)\vdash^{\mathsf{min}}\mathcal{A}\quad \mathsf{topT}(\mathcal{A})=\tau_2\\ \mathsf{com}(\tau_1:(A,\{s\});\Theta)\ =\ \tau_1:(A,S'),\Theta'\sigma\quad (\tau_1\text{ fresh})\end{array}}{\tau_1:(A,S'),\tau_2:(B,S),\Theta\vdash^{\mathsf{min}}A^{\tau_1}\rightarrow B^{\tau_2}:s\langle\mathsf{op},\ e,\ x\rangle.(\mathcal{A}\sigma)}$$

$$(\text{WT-MA}\textsc{ssign})\ \frac{\Theta\vdash^{\mathsf{min}}\mathcal{A}\quad \mathsf{topT}(\mathcal{A})=\tau}{\Theta\vdash^{\mathsf{min}}x@A^{\tau}:=e.\mathcal{A}}$$

$$(\text{WT-MI}\textsc{f}\text{T}\textsc{hen}\text{E}\textsc{lse})\ \frac{\Theta_i\vdash^{\mathsf{min}}\mathcal{A}_i\quad \mathsf{topT}(\mathcal{A}_i)=\tau\quad \mathsf{com}(\Theta_1;\Theta_2)\ =\ (\Theta_1\cup\Theta_2)\sigma}{(\Theta_1\cup\Theta_2)\sigma\vdash^{\mathsf{min}}\mathsf{if}\ e@A^{\tau}\ \mathsf{then}\ \mathcal{A}_1\sigma\ \mathsf{else}\ \mathcal{A}_2\sigma}$$

$$(\text{WT-MS}\textsc{um})\ \frac{\begin{array}{c}\tau:(A,S_i),\Theta_i\vdash^{\mathsf{min}}\mathcal{A}_i\quad \mathsf{topT}(\mathcal{A}_i)=\tau\quad \mathsf{dom}(\Theta_1)\cap\mathsf{dom}(\Theta_2)=\emptyset\\ \mathsf{com}(\tau:(A,S_1\cup S_2);\Theta_1\cup\Theta_2)=\tau:(A,S'),(\Theta_1\cup\Theta_2)\sigma\end{array}}{(\Theta_1\cup\Theta_2)\sigma\vdash^{\mathsf{min}}(\mathcal{A}_1+^{\tau}\mathcal{A}_2)\sigma}$$

$$(\text{WT-MP}\textsc{ar})\ \frac{\begin{array}{c}\tau:(A,S_i),\Theta_i\vdash^{\mathsf{min}}\mathcal{A}_i\quad \mathsf{topT}(\mathcal{A}_i)=\tau\quad \mathsf{dom}(\Theta_1)\cap\mathsf{dom}(\Theta_2)=\emptyset\\ \mathsf{com}(\tau:(A,S_1\cup S_2);\Theta_1\cup\Theta_2)=\tau:(A,S'),(\Theta_1\cup\Theta_2)\sigma\end{array}}{\tau:(A,S'),(\Theta_1\cup\Theta_2)\sigma\vdash^{\mathsf{min}}(\mathcal{A}_1\mid^{\tau}\mathcal{A}_2)\sigma}$$

$$(\text{WT-MP}\textsc{ar}\text{-E}\textsc{xt})\ \frac{\Theta_i\vdash^{\mathsf{min}}\mathcal{E}_i\quad \mathsf{dom}(\Theta_1)\cap\mathsf{dom}(\Theta_2)=\emptyset}{\Theta_1\cup\Theta_2\vdash^{\mathsf{min}}\mathcal{E}_1\mid\mathcal{E}_2}$$

$$(\text{WT-MR}\textsc{es}\text{-E}\textsc{xt})\ \frac{\Theta\vdash^{\mathsf{min}}\mathcal{E}}{\Theta/s\vdash^{\mathsf{min}}\mathcal{E}}\qquad (\text{WT-MZ}\textsc{ero})\ \frac{-}{\tau:(A,\emptyset)\vdash^{\mathsf{min}}\mathbf{0}^{A}_{\tau}}$$

$$(\text{WT-MV}\textsc{ar})\ \frac{-}{\tau:(A,\emptyset)\vdash^{\mathsf{min}}X^{A}_{\tau}}\qquad (\text{WT-MR}\textsc{ec})\ \frac{\Theta\vdash^{\mathsf{min}}\mathcal{A}\quad \mathsf{topT}(\mathcal{A})=\tau}{\Theta\vdash^{\mathsf{min}}\mathbf{rec}\ {}^{\tau}X^{A}.\mathcal{A}}$$

FIGURE 25. Typing Rules for Inferring Minimal Consistent Annotations.

DEFINITION 19 (Collapsing Operator). Given well-formed $\Theta_{1,2}$, a partial operator $\mathsf{com}(\Theta_1;\Theta_2)$ which, when defined, returns another well-formed typing, is given inductively as follows.

- $\mathsf{com}(\emptyset;\Theta)$ is always defined with the value Θ.
- Given $\tau:(A,S)$ and Θ, we calculate a substitution (a finite map from thread labels to its subset) σ and a finite set of session channels T as follows.
  Step 0: First we set σ to be empty (acting as the identity) and T to be S.
  Step 1: Find all $\tau'_i:(A,S'_i)$ in $\Theta\sigma$ such that $T\cap S'_i\neq\emptyset$.
  Step 2: Given the set $\{\tau_i:(A,S'_i)\}$ obtained in **Step 1**, set new σ as the union of the original σ and the map $\bigcup_i\tau'_i\mapsto\tau$; and new T as the union of T and $\bigcup_i S'_i$; If the results are the same as the original values, we are done. If not, we go back to **Step 1** with the new σ and T.

  We can check that the procedure above always terminates. If $\Theta\sigma$ in the resulting σ is well-formed, then $\mathsf{com}(\tau:(A,S);\Theta)$ is defined and has the value $\tau:(A,T),(\Theta\sigma)/\tau$ (where $(\Theta\sigma)/\tau$ takes off the τ-component from Θ). If not, it is undefined.
- $\mathsf{com}(\Theta_1,\tau:(A,S);\Theta_2)$ is defined with value Θ_3 if and only if $\mathsf{com}(\tau:(A,S);\Theta_2)$ is defined with value Θ'_2 and $\mathsf{com}(\Theta_1;\Theta'_2)$ is defined with value Θ_3.

If $\mathsf{com}(\Theta_1;\Theta_2)$ is defined and has the value Θ_3, we write $\mathsf{com}(\Theta_1;\Theta_2)=\Theta_3$.

Equivalently, in **Step 2** above, $\mathsf{com}(\tau : (A,S)\,;\Theta)$ finds the minimum $\{\tau_i\}$ from $\mathsf{dom}(\Theta)$ such that each τ_i is assigned (A,T_i) where T_i intersects non-trivially either with S or with another T_j in $\Theta(\tau_j)$ with $\tau_j \in \{\tau_i\}$; and those outside of $\{\tau_i\}$ do not intersect with S.

Thus $\mathsf{com}(\Theta_1\,;\Theta_2)$ combines Θ_1 and Θ_2 so that if the same participant uses some common session channel in two assignments then they are collapsed into one assignment by taking the union. This notion of common-ness is taken hereditarily so that all threads of a certain participant which are assigned transitively common session channels are eventually collapsed. We are using the following notation in the rules.

NOTATION 1. *Let* $\mathsf{com}(\Theta_1\,;\Theta_2) = \Theta_3$. *Since* Θ_3 *is the result of collapsing assignments in* $\Theta_{1,2}$*, we write this result as* $(\Theta_1 \cup \Theta_2)\sigma$ *by which we mean the result of collapsing thread labels and assigning to the resulting thread labels the unions of the corresponding session channel sets.*

Given a strongly connected (hence well-typed) I which we assume to conform to the standard bound name convention, we can use the rules in Figure 25 to generate $\mathcal{A}$ which annotates I by (1) first labelling each inaction $\mathbf{0}$ and term variable X^A with distinct thread labels (all occurrences of the same term variable are given the same thread label) and (2) inductively inferring the thread labels of a term from its subterm(s) following the rules in Figure 25, *assuming it is defined*. Except for choosing fresh thread labels in the base cases as well as in $(\text{WT-MI}\textsc{nit})$ and $(\text{WT-MC}\textsc{omm})$, each induction automatically generates a unique annotated term if ever, so that the resulting term (if any) is also unique. Write this partial function MinAnn and the resulting term (when defined) $\mathsf{MinAnn}(I)$.

PROPOSITION 11 (Existence of Minimal Consistent Annotation). *Let I be strongly connected.*

(1) *If* $\mathsf{MinAnn}(I)$ *is defined, then it gives a minimal consistent annotation of I in the sense of Definition 18.*
(2) *If I is well-threaded, then* $\mathsf{MinAnn}(I)$ *is always defined.*
(3) *I is well-threaded iff* $\mathsf{MinAnn}(I)$ *is defined.*

Proof. For (1), we show by induction on $\mathcal{A}$ that typability in Figure 25 means typability in Figure 24 with the corresponding typings modulo vacuous lacking paired type assignments needed for the well-formedness for the original typing system. All are easy induction. For (2), we directly check that satisfaction of the conditions **(G1/2/3)** means typability in Figure 25. (3) is immediate from (1) and (2). □

15. Theory of End-Point Projection (3): Coherence

15.1. Mergeability of Threads. By connectedness and well-threadedness, we have shown how we can analyse the structure of a global interaction as a collection of different threads that compose it. In other words, these threads will become, in the end-point calculus, as constituents of processes which interact with each others and realise the original behaviour in the global description. In the present section, which offers the last step of our ongoing analysis, we explore how we can consistently construct concrete processes based on these threads. This concern immediately leads to the final well-structuring principle for global description on the top of strong connectedness and well-threadedness.

We first observe it is often necessary to *merge* threads to obtain the endpoint behaviour which realises a global interaction. For instance, consider the following parallel composition of two interactions.

$$\begin{aligned} &A \to B : ch(\boldsymbol{\nu} s) \,.\, B \to A : s\langle \mathsf{op},\, e,\, x\rangle \,.\, A \to B : s\langle \mathsf{op}_1,\, e_1,\, x_1\rangle \quad | \\ &A \to B : ch(s') \,.\, B \to A : s'\langle \mathsf{op},\, e,\, x\rangle \,.\, A \to B : s'\langle \mathsf{op}_2,\, e_2,\, x_2\rangle \end{aligned} \tag{88}$$

If we annotate this interaction we know that "*B*" will be marked with two threads, each corresponding to one of the two *ch* invocations. When we make the end-point processes, we need to merge these two threads into one process, since we naturally demand there is only one service offered at *ch*. The merging becomes necessary because these two threads show different behaviours:

- In one, *A* chooses the option op_1 which *B* offers; while
- in the other, *B* chooses the option op_2 which *B* also offers.

We can project these two threads into two end-point processes:

(1) $!\,ch(s) \,.\, s \lhd op\langle e\rangle \,.\, s \rhd op_1(x_1)$
(2) $!\,ch(s) \,.\, s \lhd op\langle e\rangle \,.\, s \rhd op_2(x_2)$

In spite of having two behaviours for the same "service", or behaviour, we can consistently integrate these two threads into a single behaviour, using a branching input:

$$B\,[\,!\,ch(s) \,.\, s \lhd op\langle e\rangle \,.\, s \rhd (op_1(x_1) \,+\, op_2(x_2))]_{\sigma_B} \tag{89}$$

Indeed, this combined behaviour does act as prescribed in the global description, when the following two output threads at *A* invokes *B* via *ch* are given.

$$A\,[\,\overline{ch}(\boldsymbol{\nu} s) \,.\, s \rhd \mathsf{op}\langle x\rangle \,.\, \overline{s} \lhd \mathsf{op}_1\langle e_1\rangle . \mathbf{0} \;\mid\; \overline{ch}(\boldsymbol{\nu} s') \,.\, s' \rhd \mathsf{op}\langle x\rangle \,.\, \overline{s} \lhd \mathsf{op}_2\langle e_2\rangle . \mathbf{0}]_{\sigma_A} \tag{90}$$

We can easily observe the composition of *A* and *B* does indeed induce the original global behaviour. Similarly we can easily extract threads for *B and* combine them into a consistent whole.

$$\begin{aligned} &A \to B : ch(\boldsymbol{\nu} s) \,.\, B \to A : s\langle \mathsf{op},\, e,\, x\rangle \,.\, A \to B : s\langle \mathsf{op}_1,\, e_1,\, x_1\rangle \quad + \\ &A \to B : ch(s') \,.\, B \to A : s'\langle \mathsf{op},\, e,\, x\rangle \,.\, A \to B : s'\langle \mathsf{op}_2,\, e_2,\, x_2\rangle \end{aligned} \tag{91}$$

Similarly for

$$\begin{aligned} &\mathsf{if}\ e'@A\ \mathsf{then}\ A \to B : ch(\boldsymbol{\nu} s) \,.\, B \to A : s\langle \mathsf{op},\, e,\, x\rangle \,.\, A \to B : s\langle \mathsf{op}_1,\, e_1,\, x_1\rangle \\ &\qquad\qquad\ \ \mathsf{else}\ A \to B : ch(s') \,.\, B \to A : s'\langle \mathsf{op},\, e,\, x\rangle \,.\, A \to B : s'\langle \mathsf{op}_2,\, e_2,\, x_2\rangle. \end{aligned} \tag{92}$$

These three cases — parallel composition, sum, and conditional — are the central cases from which the need to merge threads arises.

However there *are* cases when we cannot merge two related threads coming from a single global description. Consider the following interaction, again focussing on *B*'s behaviour.

$$\begin{aligned} &A \to B : ch(\boldsymbol{\nu} s) \,.\, B \to A : s\langle \mathsf{op},\, e,\, x\rangle \quad | \\ &A \to B : ch(\boldsymbol{\nu} s') \,.\, B \to C : ch'(\boldsymbol{\nu} t)... \end{aligned} \tag{93}$$

How can we project this description to the end-point behaviour of *B*? When *B* is invoked for service *ch*, on one thread it replies to the invoker (*A*), while on the other one does something completely different. In fact, we obtain the following two slices (instances) of *B*'s behaviour from this description:

(1) $!\,ch(s) \,.\, \overline{s} \lhd \mathsf{op}\langle e\rangle$ and
(2) $!\,ch(s) \,.\, \overline{ch'}(\boldsymbol{\nu} t)...$

which can hardly be merged consistently.[13]

Thus we need a formal notion by which we can judge whether two or more end-point behaviours are consistently mergeable or not. In the above example, it should tells us if the descriptions of two different invocations for a service *ch*, when transformed into end-point processes, are in fact mergeable to yield a single coherent behaviour. We call this relation *mergeability*.

DEFINITION 20 (Mergeability). *Mergeability relation*, denoted $\bowtie$, is the smallest reflexive and symmetric relation on typed processes which is closed under the following rules.

$$\textsc{Mgbl-In}\ \frac{P_i \bowtie Q_i \text{ for each } i \in J \cap K \text{ and } op_j \neq op_k \text{ for each } j \in J \backslash K, k \in K \backslash J}{s \triangleright \Sigma_J op_j(x_j) \,.\, P_j \bowtie\ s \triangleright \Sigma_K op_k(x_k) \,.\, Q_k}$$

$$\textsc{Mgbl-Cong}\ \frac{P_i \bowtie Q_i\ (i = 1, 2, .., n)}{C[P_1]..[P_n] \bowtie C[Q_1]..[Q_n]}$$

$$\textsc{Mgbl-Zero}\ \frac{-}{P \bowtie \mathbf{0}} \qquad\qquad \frac{-}{\mathbf{0} \bowtie P}$$

$$\textsc{Mgbl-Struct}\ \frac{P \equiv P' \quad P' \bowtie Q' \quad Q' \equiv Q}{P \bowtie Q}$$

In the second rule we assume $C[P_1]..[P_n]$ and $C[Q_1]..[Q_n]$ are both typed. In the third and forth rules we assume P has the empty session typing (i.e. it has no free session names). When $P \bowtie Q$, we say *P and Q are mergeable*.

Observe that the only non-trivial clause is for the branching input: it says that, for each common branch, the behaviour should be essentially identical (up to $\bowtie$ on subexpressions). Intuitively, the relation $\bowtie$ checks that two given processes are more or less identical. This “more or less identical” means that, in brief, their behaviours do not contradict when they come to the same course of interactions, i.e. when the same branch is selected by the interacting party. Thus the rules above say that we can allow differences in branches which do not overlap, but we do demand each pair of behaviours with the same operation to be identical.

The last two rules say (under the condition noted below) that P and $\mathbf{0}$ can be merged, assuming they contain no open sessions. This becomes necessary when a given global description has (for example) a sum in which a session invocation on a specific channel occurs only in one branch, which is a natural behaviour. As a simplest case, consider:

$$\mathbf{0} \quad + \quad (A^1 \rightarrow B^2 : ch(\mathbf{\nu}\, s) \,.\, B^2 \rightarrow A^1 : s\langle \mathsf{op},\ e,\ y\rangle \,.\, \mathbf{0})$$

in which case we have the empty process on the right-hand side for each thread; but, on the right, the thread 1 is projected onto a behaviour in A which starts from an output, and the thread 2 is projected onto a replicated process (service) at *ch* located in B. Since we want these to be mergeable, we allow these two to be merged.

If two end-point behaviours are mergeable in the above technical sense, we can merge them and obtain a single process which simulates both of the two behaviours, by combining missing branches from the both. For instance, the process

$$s \triangleright go(x) \,.\, P$$

and the process

$$s \triangleright stop(x) \,.\, Q$$

are mergeable, and the result of merging is simply:

$$s \triangleright go(x) \,.\, P \ +\ stop(x) \,.\, Q$$

[13] Observe the result of directly combining two threads:

$$!\, ch(s) \,.\, (\overline{s} \lhd \mathsf{op}\langle e\rangle \oplus \overline{ch'}(\mathbf{\nu}\, t) ...)$$

does *not* conform to *either* of the two components of the parallel composition in the global descriptions.

. Below we present the formal definition of merge operation.

DEFINITION 21 (Merge operator). *$P \sqcup Q$ is a partial commutative binary operator on typed processes which is well-defined iff $P \bowtie Q$ satisfying:*

$$\begin{array}{rcl}
!ch(s).P \sqcup !ch(s).Q & \stackrel{def}{=} & !ch(s).(P \sqcup Q) \\[1ex]
\overline{ch}(s).P \sqcup \overline{ch}(s).Q & \stackrel{def}{=} & \overline{ch}(s).(P \sqcup Q) \\[1ex]
s \triangleright \Sigma_{i \in J} op_i(y_i).P_i \sqcup s \triangleright \Sigma_{i \in K} op_i(y_i).Q_i & \stackrel{def}{=} & s \triangleright \left(\begin{array}{l} \Sigma_{i \in J \cap K} op_i(y_i).(P_i \sqcup Q_i) + \\ \Sigma_{i \in J \setminus K} op_i(y_i).P_i + \\ \Sigma_{i \in K \setminus J} op_i(y_i).Q_i \end{array} \right) \\[1ex]
x := e.P \sqcup x := e.Q & \stackrel{def}{=} & x := e.(P \sqcup Q) \\[1ex]
\textsf{\textit{if}}\ e\ \textsf{\textit{then}}\ P_1\ \textsf{\textit{else}}\ P_2 \sqcup \textsf{\textit{if}}\ e\ \textsf{\textit{then}}\ Q_1\ \textsf{\textit{else}}\ Q_2 & \stackrel{def}{=} & \textsf{\textit{if}}\ e\ \textsf{\textit{then}}\ (P_1 \sqcup Q_1)\ \textsf{\textit{else}}\ (P_2 \sqcup Q_2) \\[1ex]
(P_1 \mid P_2) \sqcup (P_3 \mid P_4) & \stackrel{def}{=} & (P_1 \sqcup P_3) \mid (P_2 \sqcup P_4) \\[1ex]
s \triangleleft op\langle e \rangle.P \sqcup s \triangleleft \oplus op\langle e \rangle.Q & \stackrel{def}{=} & s \triangleleft \oplus op\langle e \rangle.(P \sqcup Q) \\[1ex]
(P_1 \oplus P_2) \sqcup (Q_1 \oplus Q_2) & \stackrel{def}{=} & (P_1 \sqcup Q_1) \oplus (P_2 \oplus Q_2) \\[1ex]
\mathbf{rec}\, X.P \sqcup \mathbf{rec}\, X.Q & \stackrel{def}{=} & \mathbf{rec}\, X.(P \sqcup Q) \\[1ex]
X \sqcup X & \stackrel{def}{=} & X \\[1ex]
P \sqcup \mathbf{0} & \stackrel{def}{=} & P \\[1ex]
\mathbf{0} \sqcup P & \stackrel{def}{=} & P \\[1ex]
P \sqcup Q & \stackrel{def}{=} & R \quad (P' \sqcup Q' = R, P \equiv P', Q \equiv Q')
\end{array}$$

where, in the right-hand side of each rule, we safely assume each application of the operator to, say, P and Q, we always have $P \bowtie Q$.

The merge operator merges two end-point behaviours. In order for this merging to be successful, this partial operation requires, for its definedness, that two processes are related by $\bowtie$. This guarantees the definedness of the right-hand side expression. By the last rule, $\sqcup$ is defined up to $\equiv$, which is consistent with $\bowtie$ being defined up to $\equiv$.

As before, the most significant rule in the above definition is the one for the branching input. They inspect the two operands which must start with

$$s \triangleright \Sigma_i op_i(y_i)$$

Then, if the operation op_i appears in both terms, then the terms after the prefix (T_i and T_i') are merged as well, which are ensured to be mergeable by the assumption. In the other cases, a new branch is added to the summation Σ.

15.2. Thread Projection and Coherence. Given a consistently thread annotated interaction, we can project each of its threads onto an end-point process. This *thread projection* is a partial operation again by its use of the merge operator. Before defining the thread projection, we add a further annotation to each recursion and each recursion variable. Given $\mathbf{rec}\ {}^{\tau}X^{A}.\mathcal{A}$ in an annotated interaction, let $\{\tau_i\}$ be the set of threads occurring in, but *not* initiated in, $\mathcal{A}$. Then we further annotate this recursion as $\mathbf{rec}\ {}^{\tau:\{\tau_i\}}X$ and each free X^{A}_{τ} in $\mathcal{A}$ as $X^{A}_{\tau:\{\tau_i\}}$. The added information is used for taking off unnecessary recursion from endpoint processes.e

CONVENTION 7 (annotated interactions). *Henceforth $\mathcal{A},\ldots$ range over extended annotated interactions (cf. Definition 12).* Annotated interaction *means* extended annotated interaction.

DEFINITION 22 (Thread Projection). *Let $\mathcal{A}$ be consistently annotated with additional annotation as noted above. Then the partial operation $\mathsf{TP}(\mathcal{A},\ \tau)$ is given as follows, assuming in each line τ_1, τ_2 and τ' are pairwise distinct and the l.h.s. is undefined iff $\sqcup$ on the r.h.s. (if any) is undefined.*

$$\mathsf{TP}(A^{\tau_1} \to B^{\tau_2} : b\tilde{s}\,.\,\mathcal{A},\ \tau) \stackrel{def}{=} \begin{cases} \overline{b}(\boldsymbol{\nu}\tilde{s})\,.\,\mathsf{TP}(\mathcal{A},\ \tau_1) & \textit{if } \tau = \tau_1 \\ !\,b(\tilde{s})\,.\,\mathsf{TP}(\mathcal{A},\ \tau_2) & \textit{if } \tau = \tau_2 \\ \mathsf{TP}(\mathcal{A},\ \tau) & \textit{otherwise} \end{cases}$$

$$\mathsf{TP}(A^{\tau_1} \to B^{\tau_2} : s\langle \mathsf{op_i},\ e_i,\ x_i\rangle\,.\,\mathcal{A},\ \tau) \stackrel{def}{=} \begin{cases} \overline{s} \lhd \mathsf{op}\langle e\rangle\,.\,\mathsf{TP}(\mathcal{A},\ \tau) & \textit{if } \tau = \tau_1 \\ s \rhd \mathsf{op_i}\langle x_i\rangle\,.\,\mathsf{TP}(\mathcal{A},\ \tau) & \textit{if } \tau = \tau_2 \\ \mathsf{TP}(\mathcal{A},\ \tau) & \textit{otherwise} \end{cases}$$

$$\mathsf{TP}(\mathcal{A}_1 +^{\tau'} \mathcal{A}_2,\ \tau) \stackrel{def}{=} \begin{cases} \mathsf{TP}(\mathcal{A}_1,\ \tau') \oplus \mathsf{TP}(\mathcal{A}_2,\ \tau') & \textit{if } \tau = \tau' \\ \mathsf{TP}(\mathcal{A}_1,\ \tau) \sqcup \mathsf{TP}(\mathcal{A}_2,\ \tau) & \textit{otherwise} \end{cases}$$

$$\mathsf{TP}(\textsf{\textit{if}}\ e@A^{\tau'}\ \textsf{\textit{then}}\ \mathcal{A}_1\ \textsf{\textit{else}}\ \mathcal{A}_2,\ \tau) \stackrel{def}{=} \begin{cases} \textsf{\textit{if}}\ e\ \textsf{\textit{then}}\ \mathsf{TP}(\mathcal{A}_1,\ \tau')\ \textsf{\textit{else}}\ \mathsf{TP}(\mathcal{A}_2,\ \tau') & \textit{if } \tau = \tau' \\ \mathsf{TP}(\mathcal{A}_1,\ \tau) \sqcup \mathsf{TP}(\mathcal{A}_2,\ \tau) & \textit{otherwise} \end{cases}$$

$$\mathsf{TP}(x@A^{\tau'} := e\,.\,\mathcal{A},\ \tau) \stackrel{def}{=} \begin{cases} x := e\,.\,\mathsf{TP}(\mathcal{A},\ \tau') & \textit{if } \tau = \tau' \\ \mathsf{TP}(\mathcal{A},\ \tau) & \textit{otherwise} \end{cases}$$

$$\mathsf{TP}(\mathcal{A}_1 |^{\tau'} \mathcal{A}_2,\ \tau) \stackrel{def}{=} \mathsf{TP}(\mathcal{A}_1,\ \tau') | \mathsf{TP}(\mathcal{A}_2,\ \tau')$$

$$\mathsf{TP}(\mathcal{E}_1 | \mathcal{E}_2,\ \tau) \stackrel{def}{=} \mathsf{TP}(\mathcal{E}_1,\ \tau') | \mathsf{TP}(\mathcal{E}_2,\ \tau').$$

$$\mathsf{TP}(\mathbf{rec}^{\ \tau':\{\tilde{\tau}_i\}} X^A\,.\,\mathcal{A},\ \tau) \stackrel{def}{=} \begin{cases} \mathbf{rec}\, X\,.\,\mathsf{TP}(\mathcal{A},\ \tau) & \textit{if } \tau \in \{\tilde{\tau}_i\} \\ \mathsf{TP}(\mathcal{A},\ \tau) & \textit{otherwise} \end{cases}$$

$$\mathsf{TP}(X^A_{\tau:\{\tilde{\tau}_i\}},\ \tau) \stackrel{def}{=} \begin{cases} X & \textit{if } \tau \in \{\tilde{\tau}_i\} \\ \mathbf{0} & \textit{otherwise} \end{cases}$$

$$\mathsf{TP}(\mathbf{0},\ \tau) \stackrel{def}{=} \mathbf{0}$$

If $\mathsf{TP}(\mathcal{A},\ \tau)$ *is undefined, we set* $\mathsf{TP}(\mathcal{A},\ \tau) = \perp$.

Some observation:

(1) For each of the initialisation and communication, we have three cases:
  (a) When the concerned thread coincides with its active thread, in which case we obtain the corresponding output prefix;
  (b) When the concerned thread coincides with its passive thread, in which case we obtain the corresponding input prefix; and
  (c) three, when neither applies, in which case we simply obtain the projection of the remaining body, which is, by Lemma 6, always in the input/output form.

(2) For assignment, parallel composition, conditional and ifthenelse, each of which is annotated with a single thread, we have two cases:
  (a) When the projecting thread coincides with the thread of the interaction, we simply carry over these constructors to endpoint processes;
  (b) If not, we simply merge these threads (or identity in the case of assignment).

(3) Other cases are defined compositionally.

The definition of thread projection already demands that, if we ever wish it to be well-defined, the behaviours inside a thread should be built consistently, i.e. whenever we use $\sqcup$ the operator should be defined. The notion of coherence includes this well-definedness, and extends it to inter-thread consistency.

The need to consider inter-thread consistency arises because the description of the behaviour of a service (replicated input) can be distributed over more than one places in one global description. In this case, we should combine the result of projecting multiple threads into one code, for which we use the merge operation again.

As an example, recall the projections we have seen in (93), page 83, which we reproduce below with annotations.

(94) $$A^0 \to B^1 : ch(\boldsymbol{\nu} s) . B^1 \to A^0 : s\langle \mathsf{op},\ e,\ x\rangle \quad | \quad A^0 \to B^2 : ch(\boldsymbol{\nu} s') . B^2 \to C^3 : ch'(\boldsymbol{\nu} t)...$$

Call this interaction $\mathcal{A}$. Then we have:

$$\begin{array}{lcl}
\mathsf{TP}(\mathcal{A},\ 0) & \overset{\text{def}}{=} & \overline{ch}(\boldsymbol{\nu} s)s \rhd \mathsf{op}\langle x\rangle.\mathbf{0} \ | \ \overline{ch}(\boldsymbol{\nu} s')... \\
\mathsf{TP}(\mathcal{A},\ 1) & \overset{\text{def}}{=} & !\, ch(s) . \overline{s} \lhd \mathsf{op}\langle e\rangle \\
\mathsf{TP}(\mathcal{A},\ 2) & \overset{\text{def}}{=} & !\, ch(s') . \overline{ch'}(\boldsymbol{\nu} t)... \\
\mathsf{TP}(\mathcal{A},\ 3) & \overset{\text{def}}{=} & !\, ch'(t)...
\end{array}$$

Clearly $!\, ch(s) . \overline{s} \lhd \mathsf{op}\langle e\rangle$ and $!\, ch(s') . \overline{ch'}(\boldsymbol{\nu} t)...$ are not mergeable. The point of coherence is that, if there are multiple threads which constitute parts of the behaviour of a permanent service, then they should be mergeable.

Since each channel *ch* uniquely defines a service, we can collect all threads contributing to the behaviour of this service by taking the passive thread of each session initialisation interaction via *ch*. Formally we set:

DEFINITION 23 (threads-per-channels). The map $\mathsf{tpc}(\mathcal{A},\ ch)$ is defined as follows, assuming $ch' \neq ch$.

$$\begin{array}{rcl}
\mathsf{tpc}(A^{\tau_1} \to B^{\tau_2} : ch(\boldsymbol{\nu} s) . \mathcal{A}',\ ch) & \overset{\text{def}}{=} & \{\tau_2\} \cup \mathsf{tpc}(\mathcal{A}',\ ch) \\
\mathsf{tpc}(A^{\tau_1} \to B^{\tau_2} : ch'(\boldsymbol{\nu} s) . \mathcal{A}',\ ch) & \overset{\text{def}}{=} & \mathsf{tpc}(\mathcal{A}',\ ch) \\
\mathsf{tpc}(A^{\tau_1} \to B^{\tau_2} : s\langle \mathsf{op},\ x,\ .\rangle \mathcal{A}',\ ch) & \overset{\text{def}}{=} & \mathsf{tpc}(\mathcal{A}',\ ch) \\
\mathsf{tpc}(x@A^{\tau} := e..\mathcal{A}',\ ch) & \overset{\text{def}}{=} & \mathsf{tpc}(\mathcal{A}',\ ch) \\
\mathsf{tpc}(\mathsf{if}\ e@A^{\tau}\ \mathsf{then}\ \mathcal{A}_1'\ \mathsf{else}\ \mathcal{A}_2',\ ch) & \overset{\text{def}}{=} & \mathsf{tpc}(\mathcal{A}_1',\ ch) \cup \mathsf{tpc}(\mathcal{A}_2',\ ch) \\
\mathsf{tpc}(\mathcal{A}_1' + \mathcal{A}_2',\ ch) & \overset{\text{def}}{=} & \mathsf{tpc}(\mathcal{A}_1',\ ch) \cup \mathsf{tpc}(\mathcal{A}_2',\ ch) \\
\mathsf{tpc}(\mathcal{A}_1' \mid \mathcal{A}_2',\ ch) & \overset{\text{def}}{=} & \mathsf{tpc}(\mathcal{A}_1',\ ch) \cup \mathsf{tpc}(\mathcal{A}_2',\ ch) \\
\mathsf{tpc}(\mathbf{rec}\ X^A_{\tau} . \mathcal{A}',\ ch) & \overset{\text{def}}{=} & \mathsf{tpc}(\mathcal{A}',\ ch) \\
\mathsf{tpc}(X^A_{\tau},\ ch) & \overset{\text{def}}{=} & \emptyset \\
\mathsf{tpc}(\mathbf{0},\ ch) & \overset{\text{def}}{=} & \emptyset
\end{array}$$

That is, if $\tau_1, \tau_2 \in \mathsf{tpc}(\mathcal{A},\ ch)$, then these two threads are parts of the behaviour of the same service.

If two input threads are for the same service channel, then they are equivalent. Below $\mathsf{channels}(\mathcal{A})$ indicates the set of service channels occurring in $\mathcal{A}$.

DEFINITION 24. *Given a well-threaded annotated interaction $\mathcal{A}$, for all $\tau \in \mathcal{A}$, we define the equivalence class $[\tau]^{\mathcal{A}} \subseteq \mathbb{N}$ as*

$$[\tau]^{\mathcal{A}} = \begin{cases} \mathsf{tpc}(\mathcal{A},\ ch) & \textit{if } \exists\, ch \in \mathsf{channels}(\mathcal{A}) \textit{ such that } \tau \in \mathsf{tpc}(\mathcal{A},\ ch) \\ \{\tau\} & \textit{otherwise.} \end{cases}$$

Given $\tau_{1,2}$ in $\mathcal{A}$, we write $\tau_1 \equiv_{\mathcal{A}} \tau_2$ if there exists $\tau \in \mathcal{A}$ such that $\tau_1, \tau_2 \in [\tau]^{\mathcal{A}}$.

DEFINITION 25 (Coherence). Given a well-threaded, consistently annotated interaction $\mathcal{A}$, we say that $\mathcal{A}$ is *coherent* if the following two conditions hold:

(1) For each thread τ in $\mathcal{A}$, $\mathsf{TP}(\mathcal{A},\ \tau)$ is well-defined.

(2) For each pair of threads τ_1, τ_2 in $\mathcal{A}$ with $\tau_1 \equiv_{\mathcal{A}} \tau_2$, we have $\mathsf{TP}(\mathcal{A},\ \tau_1) \bowtie \mathsf{TP}(\mathcal{A},\ \tau_2)$.

We say a well-threaded non-annotated interaction I is *coherent* if I has an annotation which is coherent in the above sense.

EXAMPLE 17. (thread projection and coherence)

(1) The interactions (88) in page 83, (91) in page 83 and (92) in page 83 are all coherent, but (93) in page 83 is not.

(2) The following annotated global description already appeared in Section 15.1 (below and henceforth we often write I etc. for annotated interactions).

$$I \overset{\text{def}}{=} \mathbf{0} \quad + \quad (A^1 \rightarrow B^2 : ch(\boldsymbol{\nu} s)\,.\,B^2 \rightarrow A^1 : s\langle \mathsf{op},\ e,\ y\rangle\,.\,\mathbf{0})$$

The description I says that there are two possible paths in interactions: in one path, nothing will happen. On the other, A and B will initiate a session, for which there is one interaction, and the session completes there. Then the projection of I onto 1 and 2 are, referring to Definition 22:

$$\begin{aligned}
\mathsf{TP}(I,\ 1) &\overset{\text{def}}{=} \mathbf{0} \oplus (\overline{c}(\boldsymbol{\nu} s).s(y).\mathbf{0}) \\
\mathsf{TP}(I,\ 2) &\overset{\text{def}}{=} \mathbf{0} \sqcup (!c(s).\overline{s}\mathsf{op}\langle e\rangle.\mathbf{0})
\end{aligned}$$

Note the r.h.s. of the second simply becomes $!c(s).\overline{s}\mathsf{op}\langle e\rangle.\mathbf{0}$ itself. Since there is a unique thread for a service at ch, we conclude I is coherent.

(3) As a slightly more complex example, consider:

$$\begin{aligned}
I' \overset{\text{def}}{=} \quad & A^1 \rightarrow B^2 : ch(\boldsymbol{\nu} s)\,.\,B^2 \rightarrow A^1 : s\langle \mathsf{op}_1,\ \text{``}hello''\text{},\ y_1\rangle\,.\,\mathbf{0} \ + \\
& A^3 \rightarrow B^4 : ch(\boldsymbol{\nu} s)\,.\,B^2 \rightarrow A^1 : s\langle \mathsf{op}_2,\ \text{``}goodbye''\text{},\ y_2\rangle\,.\,\mathbf{0} \ + \\
& A^5 \rightarrow C^6 : ch'(\boldsymbol{\nu} s')\,.\,C^6 \rightarrow A^5 : s'\langle \mathsf{op}_1,\ \text{``}hi''\text{},\ x\rangle\,.\,\mathbf{0}
\end{aligned}$$

In this case, the projections become:

$$\begin{aligned}
\mathsf{TP}(I',\ 1) &\overset{\text{def}}{=} (\overline{ch}(\boldsymbol{\nu} s).s\mathsf{op}_1(y_1).\mathbf{0}) \oplus \mathbf{0} \oplus \mathbf{0} \\
\mathsf{TP}(I',\ 3) &\overset{\text{def}}{=} \mathbf{0} \oplus (\overline{ch}(\boldsymbol{\nu} s).s\mathsf{op}_1(y_1).\mathbf{0}) \oplus \mathbf{0} \\
\mathsf{TP}(I',\ 5) &\overset{\text{def}}{=} \mathbf{0} \sqcup \mathbf{0} \sqcup \overline{ch'}(\boldsymbol{\nu} s').s'\mathsf{op}_2(y_2).\mathbf{0}) \\
\mathsf{TP}(I',\ 2) &\overset{\text{def}}{=} (!ch(s).\overline{s}\mathsf{op}_1\langle\text{``}hello''\rangle.\mathbf{0}) \sqcup \mathbf{0} \sqcup \mathbf{0} \\
\mathsf{TP}(I',\ 4) &\overset{\text{def}}{=} \mathbf{0} \sqcup (!ch(s).\overline{s}\mathsf{op}_2\langle\text{``}goodbye''\rangle.\mathbf{0}) \sqcup \mathbf{0} \\
\mathsf{TP}(I',\ 6) &\overset{\text{def}}{=} \mathbf{0} \sqcup (!ch'(s').\overline{s'}\mathsf{op}_1\langle\text{``}hi''\rangle.\mathbf{0}) \sqcup \mathbf{0}
\end{aligned}$$

Immediately each of $\mathsf{TP}(I',\ i)$ $(1 \leq i \leq 6)$ is defined. Since we have $\mathsf{tpc}(I',\ ch) = \{2,4\}$ and $\mathsf{tpc}(I',\ ch') = \{6\}$, in order for I' to be coherent, we should have $\mathsf{TP}(I',\ 2) \bowtie \mathsf{TP}(I',\ 4)$ which is possible iff $\mathsf{op}_1 \neq \mathsf{op}_2$.

A further example can be found in Section 16.7

15.3. Subject Reduction for Coherence. Below $\mathcal{A} \equiv \mathcal{A}'$ is defined precisely following $\equiv$ on unannotated interactions.

PROPOSITION 12 (subject congruence for coherence).

(1) *$\sqcup$ is partially symmetric and associative, and has the identity $\mathbf{0}$, all up to $\equiv$. I.e.: $P \sqcup Q$ is defined iff $Q \sqcup P$ is defined and when they are so, we have $P \sqcup Q \equiv Q \sqcup P$; $(P \sqcup Q) \sqcup R$ is defined iff $P \sqcup (Q \sqcup R)$ is defined and, when they are so, we have $(P \sqcup Q) \sqcup R \equiv P \sqcup (Q \sqcup R)$; and for each P we have $P \sqcup \mathbf{0} \equiv P$.*

(2) *Suppose I is coherent. Let $\mathcal{A}$ be its consistent annotation and τ be its thread. Then $\mathcal{A} \equiv \mathcal{A}'$ implies* $\mathsf{TP}(\mathcal{A},\ \tau) \equiv \mathsf{TP}(\mathcal{A}',\ \tau)$.

(3) *Suppose I is coherent. Then $I \equiv I'$ implies I' is coherent.*

Proof. (1) is immediate from the definition, noting, by the first rule in Definition 21 as well as Definition 20, if $(P \sqcup Q) \sqcup R$ is defined then they do not have any conflicting branch among the three, which is the same thing as $P \sqcup (Q \sqcup R)$ being defined. For (2), we use rule induction on the generation of $\equiv$. As one typical case, suppose $\mathcal{A} \equiv \mathcal{A}'$ from the associativity of $|$, i.e.

$$(\mathcal{A}_1|\mathcal{A}_2)|\mathcal{A}_3 \ \equiv\ \mathcal{A}_1|(\mathcal{A}_2|\mathcal{A}_3).$$

If τ is the top thread, then we observe:

$$(\mathsf{TP}(\mathcal{A}_1,\ \tau)|\mathsf{TP}(A_2,\ \tau))|\mathsf{TP}(\mathcal{A}_3,\ \tau) \ \equiv\ \mathsf{TP}(\mathcal{A}_1,\ \tau)|(\mathsf{TP}(\mathcal{A}_2,\ \tau)|\mathsf{TP}(\mathcal{A}_3,\ \tau))$$

hence done. If τ is not a top thread, then $|$ is interpreted as $\sqcup$ which is associative by (the annotated analogue of) (1), hence done. For (3), by (2) we only have to consider mergeability of threads. For this purpose we observe that, in the definition of mergeability $P \bowtie Q$ in Definition 20, we can first apply $\equiv$ to P and Q using the last rule in Definition 20, and then validate their mergeability via the rules except that last one. Thus if $I \equiv I'$ and the thread projection of I for each thread is defined, then that of I' is defined: similarly, if we have e.g. $\mathtt{tpc}(I,\ ch) = \{\tau_1, \tau_2\}$ then $\mathsf{TP}(I,\ \tau_1) \bowtie \mathsf{TP}(I,\ \tau_2)$ iff $\mathsf{TP}(I',\ \tau_1) \bowtie \mathsf{TP}(I',\ \tau_2)$. □

Next we observe:

LEMMA 7 (substitution lemma for coherence). *If* $\mathbf{rec}\, X.I$ *is coherent then so is* $I[(\mathbf{rec}\, X.I)/X]$.

Proof. (outline) We consider a consistent annotation of $\mathbf{rec}\, X.I$ which we write

$$\mathcal{A}' \stackrel{\text{def}}{=} \mathbf{rec}\, X^A_{\tau:\tilde{\tau}}.\mathcal{A}.$$

We then consider

$$\mathcal{A}'' \stackrel{\text{def}}{=} \mathcal{A}[\mathcal{A}'/X^A_{\tau':\tilde{\tau}}].$$

where, to be exact, each substituting $\mathcal{A}'$ should be given, for its passive initial thread, a fresh thread number, though we leave this point implicit since this does not affect the following argument. Consider a thread τ_i in $\tilde{\tau}$. Then either (1) it starts from a positive position in $\mathcal{A}$ or (2) from a passive position in communication. If $\mathsf{TP}(\mathcal{A}',\ \tau_i) = \mathbf{rec}\, X.P$ then by induction we can check

$$\mathsf{TP}(\mathcal{A}'',\ \tau_i) \quad = \quad P[\mathbf{rec}\, X.P/X].$$

including its definedness. In the both (1) and (2), this thread is not to be merged with any other threads, hence done. On the other hand, for τ' which is not in $\tilde{\tau}$ but which occurs in I, this should be a passive complete thread inside I (for the meaning of "complete", see the proof of Theorem 4, page 79). Note the annotation $\tau : \tilde{\tau}$ in $\mathbf{rec}\, X^A_{\tau:\tilde{\tau}}.\mathcal{A}$ makes the thread projection of $\mathcal{A}''$ to τ'' turns this substituted term to $\mathbf{0}$, just as each $X^A_{\tau:\tilde{\tau}}$ is done so in the thread projection of $\mathcal{A}'$ to τ''. Thus, letting $\mathsf{TP}(\mathcal{A}',\ \tau') = !ch(\tilde{s}).P$, we obtain:

$$\mathsf{TP}(\mathcal{A}'',\ \tau') \quad = \quad !ch(\tilde{s}).P$$

that is the τ'-projection of $\mathcal{A}'$ and the τ-projection of $\mathcal{A}''$ coincide. Note the same reasoning holds for any other thread in $\mathcal{A}'$ which should be merged with τ', hence mergeability does not change, as required. □

Below we assume, as noted just before Theorem 5, 79, that all and only top-level parallel compositions are unannotated and that reduction takes off the annotations of newly formed top-level parallel compositions if any.

THEOREM 6 (Coherence: subject reduction). *If $\mathcal{A}$ is coherent and $(\sigma, \mathcal{A}) \to (\sigma', \mathcal{A}')$, then $\mathcal{A}'$ is also coherent.*

Proof. We check how the collection of threads together with the associated thread projections is transformed from $\mathcal{A}$ to $\mathcal{A}'$. Let this collection be Φ and Φ', respectively. By abuse of designation we say that these sets are coherent. We then show the following claim by rule induction on reduction rules.

Claim. If Φ is coherent, then Φ' is also coherent. Moreover Φ' is either the result of truncating some of threads from Φ, as well as adding zero or more collections of the fresh complete threads, each collection being multiple copies of a complete thread existing in Φ.

Above the copies can be created due to recursion. Assume Φ from $\mathcal{A}$ is coherent. For (INIT), we first take off, from Φ, the head of one complete thread at *ch* (so that it is not a target of merging anymore) and truncates one active thread, otherwise using the same threads as Φ. The originally passive thread, say τ_2, is now an active thread, whereas the originally active thread, say τ_1, is now a passive thread, both of which are put in Φ'. If τ_1 demands intra-thread merging through its own thread projection, then so does it originally in Φ. Otherwise merging in Φ' remains the same as Φ, hence done. For (COM), we truncate two non-complete threads, one active and the other passive, from Φ. Again the originally active thread, say τ_1, is now passive, whereas the originally passive thread, say τ_2, is now active. There is no change in inter-thread merging, so we again know Φ' is coherent. (ASSIGN) truncates a single active thread in Φ, while both (IFTRUE/FALSE) and (SUM) simply cut off some of the threads from Φ to make Φ'. For (PAR), assume

$$\mathcal{A} \quad = \quad \mathcal{A}_1 | \mathcal{A}_2$$

Accordingly we can set

$$\Phi \quad = \quad \Phi_1 \cup \Phi_2$$

where Φ_1 and Φ_2 respectively come from $\mathcal{A}_1$ and $\mathcal{A}_2$. Now assume $\mathcal{A}_1$ reduces to $\mathcal{A}_1'$, with the new threads Φ_1'. By assumption, we know Φ_1' may truncate some threads and may add copies of existing threads with fresh thread numbers. Since neither affect mergeability we are done (note intra-thread mergeability remains the same by the definition of the projection of a top-level parallel composition). (RES) does not change threads (i.e. $\Phi = \Phi'$). (REC) is by (IH) and by Lemma 7, possibly adding collections of fresh copies of complete threads. Finally (STRUCT) is immediate from Proposition 12. □

Finally, since $\equiv$ is decidable, so is mergeability, by which we know:

PROPOSITION 13 (Coherence Validation). *Given a consistently annotated $\mathcal{A}$, there is an algorithm which returns true if $\mathcal{A}$ is coherent, false if not.*

Since typability, connectedness and well-threadedness are also calculable, this shows that we can algorithmically check whether a given untyped interaction is coherent or not. Since complexity of checking typability/connectedness/well-threadedness is each linear, the complexity of this validation is linear if validation of mergeability can be done so. We believe this is possible by taking certain normal forms.

16. Main Results: EPP Theorem and its Consequences

16.1. The Projection. In this section we introduce the formal definition of endpoint projection and establish its properties. Below and henceforth we say I is *restriction-free* whenever it contains no terms of the form $(\boldsymbol{\nu}\, s)\ I'$ as its subterm. Recall also being coherent entails being well-typed, strongly connected and well-threaded.

DEFINITION 26 (End-Point Projection). *Let I be a coherent interaction such that $I \equiv ((\boldsymbol{\nu}\, s)\ )I'$ where I' is restriction-free. Let $\mathcal{A}$ be a consistent annotation of I'. Then the* end point projection of $\mathcal{A}$ under σ*, denoted* $\mathsf{EPP}((\boldsymbol{\nu}\, \tilde{s})\ \mathcal{A},\ \sigma)$*, is given as the following network.*

$$(\boldsymbol{\nu}\, \tilde{s})\ \Pi_{A \in \mathsf{part}(I)}\ A[\Pi_{[\tau]} \bigsqcup_{\tau' \in [\tau]} \mathsf{TP}(\mathcal{A},\ \tau')]_{\sigma@A}$$

where $\mathsf{part}(I)$ *denotes the set of participants mentioned in I.*

At the end of the section, we shall illustrate how the constructions involved in Definition 26 works step by step using a mapping of a concrete, non-trivial global description.

The mapping given in Definition 26 is defined after choosing a specific annotation of an interaction. The following result shows the map in fact does not depend on a specific (consistent) annotation chosen, as far as a global description has no incomplete threads, i.e. it does no free session channels (which is what programmers/designers usually produce).

PROPOSITION 14 (Invariance under Annotations). *Given a coherent $\Gamma \vdash I \rhd \emptyset$ and $\Gamma \vdash \sigma$, let $\mathcal{A}_{1,2}$ be consistent annotations of I and $\Gamma \vdash \sigma$. Then we have* $\mathsf{EPP}(\mathcal{A}_1,\ \sigma) \equiv \mathsf{EPP}(\mathcal{A}_2,\ \sigma)$.

Proof. Let I be coherent and, without loss of generality, I be restriction free. Let $\mathcal{A}$ be a minimal consistent annotation of I (whose existence is guaranteed by Proposition 11, 82). Let $\mathcal{A}'$ be a possibly different consistent annotation of I. By the definition of minimality (cf. Definition 18) and by construction, there is a surjective map σ from the threads in $\mathcal{A}_0$ to $\mathcal{A}$. Let τ be a thread in $\mathcal{A}'$ and let $\tau_1, .., \tau_n$ with $n \geq 2$ be threads in $\mathcal{A}$ such that $\sigma(\tau_i) = \tau$. First τ cannot be a complete thread (for the notion of complete threads see the proof of Theorem 4, 79) since if so, τ in I should start from a unique service channel, hence cannot be given multiple thread labels. Hence τ is incomplete. Because I has no free session, this means each τ_i is an active thread starting from a session initialisation. Now the initial action of such a thread cannot be under a different prefix since if so that prefix (its passive part) should have the same thread label. Hence each τ_i is the initial active thread of a prime interaction (an interaction is *prime* if it cannot be decomposed into a non-trivial top-level parallel composition). That is each τ_i is in $\mathcal{A}'_i$ such that $\mathcal{A}' \equiv (\Pi_i \mathcal{A}_i) | \mathcal{A}''$. Noting $\mathcal{A}$ has precisely the same syntactic structure, we infer:

$$\begin{array}{rcl} \mathsf{TP}(\mathcal{A}',\ \tau) & \stackrel{\text{def}}{=} & \Pi_i \mathsf{TP}(\mathcal{A}'_i,\ \tau) \\ & \equiv & \Pi_{\sigma(\tau_i)=\tau} \mathsf{TP}(\mathcal{A},\ \tau_i) \end{array}$$

that is the projection of $\mathcal{A}'$ onto τ and the projections of $\mathcal{A}$ onto $\{\tau_i\}$ coincide. □

16.2. Need for Pruning. The main results of the present paper we shall establish later says that the dynamics of a given (coherent) global description is exactly preserved and reflected by its endpoint counterpart, via the EPP mapping introduced above. To state this result accurately, we need a preparation. Consider the following very simple global description.

(95) $$A \to B : b(\boldsymbol{\nu}\, s).\mathbf{0}.$$

If we EPP this interaction with environment σ, we obtain the following network:

(96) $$A[\overline{ch}(\boldsymbol{\nu}\, s).\mathbf{0}]_{\sigma@A} \ \mid\ B[ch(s).\mathbf{0}]_{\sigma@B}$$

Now (95) reduces as:

(97) $$(\sigma, A \to B : b(\boldsymbol{\nu}\, s).\mathbf{0}) \quad \to \quad (\sigma, \mathbf{0})$$

while (96) reduces as

(98) $$A[\overline{ch}(\boldsymbol{\nu}\, s).\mathbf{0}]_{\sigma@A} \ \mid\ B[ch(s).\mathbf{0}]_{\sigma@B} \quad \to \quad A[\mathbf{0}]_{\sigma_A} \ \mid\ B[ch(s).\mathbf{0}]_{\sigma@B}$$

Note (97) results in the empty configuration, while in (97) the service at *ch* still remains, because it is replicated. Note there is a discrepancy between two reductions: before reduction, the end-point behaviour is indeed the EPP of the global description, while after reduction, the former is no longer the EPP of the latter. However, as far as "active" behaviour (i.e. those who induce immediate reduction) goes, there is a precise match: that is, as far as we take off the replicated service as a garbage (since it is no longer of the use from inside this configuration), there is an exact match.

As another, and more subtle, example, consider the following interaction:

$$\begin{aligned} &A \to B : b(\boldsymbol{\nu}\, s).B \to A : s\langle \mathsf{ack}\rangle.A \to B : s\langle \mathsf{go}\rangle.\mathbf{0} \quad + \\ &A \to B : b(\boldsymbol{\nu}\, s).B \to A : s\langle \mathsf{ack}\rangle.A \to B : s\langle \mathsf{stop}\rangle.\mathbf{0} \end{aligned} \tag{99}$$

The projection of this interaction is, omitting trailing inactions:

$$\begin{aligned} &A[\overline{ch}(\boldsymbol{\nu}\, s).s \rhd \mathsf{ack}\bar{s} \lhd \mathsf{go} \oplus \overline{ch}(\boldsymbol{\nu}\, s).s \rhd \mathsf{ack}\bar{s} \lhd \mathsf{stop}]_{\sigma@A} \quad | \\ &B[ch(s).\bar{s} \lhd \mathsf{ack}.(s \rhd \mathsf{ok}s \rhd \mathsf{stop}]_{\sigma@B} \end{aligned} \tag{100}$$

After one step which takes the left branch, (99) reduces to the following configuration:

$$A \to B : b(\boldsymbol{\nu}\, s).B \to A : s\langle \mathsf{ack}\rangle.A \to B : s\langle \mathsf{go}\rangle.\mathbf{0} \tag{101}$$

The corresponding reduction for (100) leads to:

$$A[\overline{ch}(\boldsymbol{\nu}\, s).s \rhd \mathsf{ack}\bar{s} \lhd \mathsf{go}]_{\sigma@A} \mid B[ch(s).\bar{s} \lhd \mathsf{ack}.(s \rhd \mathsf{go}s \rhd \mathsf{stop}]_{\sigma@B} \tag{102}$$

Now take the EPP of (101):

$$A[\overline{ch}(\boldsymbol{\nu}\, s).s \rhd \mathsf{ack}\bar{s} \lhd \mathsf{go}]_{\sigma@A} \mid B[ch(s).\bar{s} \lhd \mathsf{ack}.s \rhd \mathsf{go}]_{\sigma@B} \tag{103}$$

There is again a discrepancy between (103) and (102): the former (or its original, (101) has *lost* one branch, while (102) keeps it. But again we realise this lost branch is inessential from the viewpoint of the internal dynamics of the resulting configuration: the branch "stop" is never used in (101).

In summary, a global interaction can lose information during reduction which is still kept in the corresponding reduction in its EPP, due to persistent behaviour at service channels. This motivates the introduction of the following asymmetric relation that we shall use to state a property of the end-point projection. Below we write $!R$ when R is a n-fold composition of replications.

DEFINITION 27 (Pruning). Assume we have $\Gamma \vdash_A P \rhd \Delta$, $\Gamma, \Gamma' \vdash_A Q \rhd \Delta$ and, moreover, $\Gamma \vdash_A P \rhd \Delta$ is a minimal typing. If further we have $Q \equiv Q_0|!R$ where $\Gamma \vdash Q_0 \rhd \Delta$, $\Gamma' \vdash_A R$ and $P \bowtie Q_0$, then we write: $\Gamma \vdash_A P \lessdot Q \rhd \Delta$ or $P \lessdot Q$ for short, and say *P prunes Q under* $\Gamma; \Delta$.

Writing simply $P \lessdot Q$ does not in fact lose any precision since we can then always reconstruct appropriate typings. $P \lessdot Q$ indicates P is the result of cutting off "unnecessary branches" of Q, in the light of P's own typing. $\lessdot$ is in fact a typed strong bisimulation in the sense that $P \lessdot Q$ means they have precisely the same observable behaviours *except for the visible input actions at pruned inputs, either branches or replicated channels.* Thus in particular it satisfies the following condition.

LEMMA 8 (pruning lemma).

(1) *$\lessdot$ is a strong reduction bisimulation in the sense that: (a) If $M \lessdot N$ and $M \to M'$ then $M \to N'$ such that $M' \lessdot N'$; and (b) If $M \lessdot N$ and $M \to N'$ then $M \to M'$ such that $M' \lessdot N'$.*

(2) *$\lessdot$ is transitive, i.e. $M \lessdot N$ and $N \lessdot R$ imply $M \lessdot R$.*

Proof. (1) is because, if $M \lessdot N$, the branches pruned from M can only be among those which are never used by M, hence do not contribute to the reduction. (2) is by noting: if we prune R to make N following the minimal typing of N, and prune N to make M following the minimum typing of M, then we can surely take off all branches and replicated inputs from R in the light of the minimal typing of M, and obtains M itself. □

As we just observed, $\lessdot$ satisfies the much stronger property of being indeed a strong bisimulation w.r.t. all typed transitions under the minimal typing of the left-hand processes.

16.3. EPP Theorem and its Consequences. We are now ready to state the main results of this paper. Below we write $\Gamma \vdash \sigma$ when the stored values in σ follow the typing in Γ in the obvious sense. In (1) below, $\bot(\Delta)$ denotes the result of replacing each occurrence of type assignment in Δ, say $\tilde{s}[A,B] : \alpha$, with $\tilde{s} : \bot$. In (2), $\equiv_{rec}$ denotes the equality induced by the unfolding/foldings of process recursion (note they do not change behaviour up to strong bisimilarity).

THEOREM 7 (End-Point Projection). *Assume $\mathcal{A}$ is well-typed, strongly connected, well-threaded and coherent. Assume further $\Gamma \vdash \mathcal{A} \rhd \Delta$ and $\Gamma \vdash \sigma$. Then the following three properties hold.*

(1) (type preservation) *If* $\Gamma \vdash \mathcal{A} \rhd \Delta$ *is the minimal typing of* $\mathcal{A}$*, then* $\Gamma \vdash \mathsf{EPP}(\mathcal{A},\ \sigma) \rhd \bot(\Delta)$.
(2) (soundness) *if* $\mathsf{EPP}(\mathcal{A},\ \sigma) \to N$ *then there exists* $\mathcal{A}'$ *such that* $(\sigma,\ \mathcal{A}) \to (\sigma',\ \mathcal{A}')$ *such that* $\mathsf{EPP}(\mathcal{A}',\ \sigma') \lessdot\ \equiv_{rec} N$.
(3) (completeness) *If* $(\sigma,\ I) \to (\sigma',\ I')$ *then* $\mathsf{EPP}(I,\ \sigma) \to N$ *such that* $\mathsf{EPP}(I',\ \sigma') \lessdot N$.

COROLLARY 2. *Assume $\mathcal{A}$ is coherent. Assume further $\Gamma \vdash \mathcal{A} \rhd \Delta$ and $\Gamma \vdash \sigma$. Then the following three properties hold. In (2) below, $\lessdot_{rec}$ is given as the transitive closure of $\lessdot \cup \equiv_{rec}$.*

(1) (error-freedom) $\mathsf{EPP}(\mathcal{A},\ \sigma)$ *does not have a communication error (cf. §12.2).*
(2) (soundness for multi-step reduction) *if* $\mathsf{EPP}(\mathcal{A},\ \sigma) \to^n N$ *then there exists* $\mathcal{A}'$ *such that* $(\sigma,\ \mathcal{A}) \to^n (\sigma',\ \mathcal{A}')$ *and* $\mathsf{EPP}(\mathcal{A}',\ \sigma') \lessdot\ \equiv_{rec} N$.
(3) (completeness for multi-step reduction) *if* $(\sigma,\ \mathcal{A}) \to^n (\sigma',\ \mathcal{A}')$ *then* $\mathsf{EPP}(\mathcal{A},\ \sigma) \to^n N$ *such that* $\mathsf{EPP}(I',\ \sigma') \lessdot N$*;*

Proof. (1) is immediate from Theorem 7 (1) and Corollary 1 (page 65). (2) and (3) are by Lemma 8 (1,2) and Theorem 7 (2, 3), combined with the standard tiling argument and induction on n. For example, for (2), the case when $n = 1$ is Theorem 7 (2). Suppose the statement holds up to n reductions and assume $\mathsf{EPP}(\mathcal{A},\ \sigma) \to^{n+1} N$. By definition this means $\mathsf{EPP}(\mathcal{A},\ \sigma) \to^n N_0 \to N$ for some N_0. Hence by (IH) there exists $\mathcal{A}'_0$ such that $(\sigma,\ \mathcal{A}) \to^n (\sigma'_0,\ \mathcal{A}'_0)$ and $\mathsf{EPP}(\mathcal{A}'_0,\ \sigma'_0) \lessdot_{rec} N_0$. By $N_0 \to N$ and since $\lessdot_{rec}$ immediately satisfies the same simulation property as $\lessdot$, $\mathsf{EPP}(\mathcal{A}'_0,\ \sigma') \to N''$ such that $N' \lessdot_{rec} N''$. By Theorem 7 (2) again we have $(\sigma'_0, \mathcal{A}'_0) \to (\sigma', \mathcal{A}')$ such that $\mathsf{EPP}(\mathcal{A}',\ \sigma') \lessdot N''$. By the transitive of $\lessdot_{REC}$ we have $\mathsf{EPP}(\mathcal{A}',\ \sigma') \lessdot N'$ as required. □

Several observations on these results follow.

REMARK 7.

(1) Corollary 2 (1) indicates, among others, once we can type check a global description and ensures it is coherent, then its endpoint projections do *not* have type errors in their mutual interactions. This gives a basic form of a guarantee of "good" properties at runtime through the static validation of global descriptions. Potential properties for validation would include deadlock freedom, livelock freedom, and various security properties.
(2) Corollary 2 (2-3) says that all and only interactions in which endpoint processes will be engaged, however many steps they would take, are precisely in correspondence with those specified by the original global description.
(3) In both Theorem 7 (2-3) and Corollary 2 (2-3), we can strengthen the reduction-based simulation in Theorem 7 and Corollary 2 by annotating reduction with associated constructors, e.g.

$$(\sigma,\ A \to B : s\langle \mathsf{op},\ v,\ x\rangle.I') \stackrel{\langle A,B,s,\mathsf{op},v,x\rangle}{\longrightarrow} (\sigma[x \mapsto v],\ I)$$

similarly for the end-point calculus. This makes explicit how each action in the original global description corresponds to the action in its end point projection.

The rest of this section is devoted to the proof of Theorem 7. We prove its three clauses one by one.

16.4. Proof of Theorem 7: (1) Type Preservation. The type preservation is proved in the following three steps, after introducing an inference system for a single thread of an annotated interaction, written $\Gamma_i \vdash^{\tau_i}_{\mathsf{min}} I \triangleright \Delta_i$.

- **(a)** We relate the original minimal global typing with this per-thread global typing, i.e. we show $\Gamma \vdash_{\mathsf{min}} I \triangleright \Delta$ iff $\Gamma = \sqcup \Gamma_i$ and $\Delta = \sqcup \Delta_i$ where $\{\tau_i\}$ exhausts the threads in I.
- **(b)** We then relate this per-thread global typing with the minimal typing of thread projections, establishing: $\Gamma_i \vdash^{\tau_i}_{\mathsf{min}} I \triangleright \Delta_i$ iff $\Gamma_i \vdash_{\mathsf{min}} \mathsf{TP}(I,\ \tau_i) \triangleright \Delta'_i$ for each τ_i, where Δ' is the result of replacing each $\tilde{s}[A,B] : \alpha$ in Δ with $\tilde{s} : \bot$.
- **(c)** Finally we relate the minimal typing of mergeable processes with the minimal typing of the result of merging them, by showing a general statement: $\sqcup_i \Gamma_i \vdash_{\mathsf{min}} \sqcup_i P_i \triangleright \sqcup_i \Delta_i$ with $P_i \bowtie P_j$ for each $i, j \in I$.

We start our inquiry from the following observation which relates the lub in the inclusion ordering (of the global typing) and the lub in the session subtyping (of the endpoint typing).

DEFINITION 28 (equi-input subtyping). *We define the relation* $\alpha \sqsubseteq \beta$ *over closed types by setting* $\alpha \sqsubseteq \beta$ *iff* $\alpha \mathcal{R} \beta$ *for a witnessing relation* $\mathcal{R}$ *satisfying the same rules as in Definition 1 for* $\Subset$ *(in §10.4, page 47) except for replacing* (INC-IN) *with the following rule:*

$$(\textsc{Eq-In})\ \frac{\beta \ \approx\ \Sigma_{i \in J} s \downarrow op_i(\theta_i) . \alpha'_i \quad \forall i \in J.\ \alpha_i \Subset \alpha'_i}{\Sigma_{i \in J} s \downarrow op_i(\theta_i) . \alpha_i \ \Subset\ \beta}$$

We call $\sqsubseteq$ equi-input subtyping.

PROPOSITION 15 (Equi-input Subtyping).

(1) $\sqsubseteq$ *is a partial order on types modulo* $\approx$.

(2) $\alpha \sqsubseteq \beta$ *implies both* $\alpha \Subset \beta$ *and* $\alpha \preceq \beta$.

(3) *If* $\alpha_{1,2}$ *have an upper bound w.r.t.* $\sqsubseteq$*, then they also have the least upper bound. Then they also have the lub w.r.t.* $\Subset$ *and* $\preceq$ *and all these three lubs coincide.*

Proof. (1) is standard and is essentially identical to the proof of Proposition 6. (2) is by noting, for both $\Subset$ and $\preceq$, the only difference is in the input rule. For (3) the existence of a lub given an upper bound is as in Propositions 4 and 7. For the latter part, by (2) we know that an upper bound of $\alpha_{1,2}$ w.r.t. $\sqsubseteq$ is, their upper bound w.r.t. $\Subset$ and $\preceq$. Now take the lub w.r.t. $\sqsubseteq$, say β, which has the same input branches as *both* $\alpha_{1,2}$ and the join of their output branches. Take an upper bound of $\alpha_{1,2}$ w.r.t. $\Subset$, say β'. Then it also adds input branches, hence surely $\beta \Subset \beta'$. Next take an upper bound of $\alpha_{1,2}$ w.r.t. $\Subset$, say β''. Then it may take off some input branches of $\alpha_{1,2}$. Hence $\beta \Subset \beta''$, as required.□

As noted, to carry out Step **(a)**, we use an additional typing system which, given an annotated global description and its thread, a minimal typing which is specific to that thread. The sequent of this system is written:

$$\Gamma \vdash^{\tau} \mathcal{A} \triangleright \Delta$$

where $\mathcal{A}$ is an (extended) annotated interaction, τ is its thread, and Γ/Δ are as given in §12.1 (for Δ we use a slightly refined types which associate participant information to session types, as in $\tilde{s}[B] : \alpha$, so that we can directly reconstruct global types from a set of these types). The typing rules are given in Figure 26, where we omit the obvious inverse communication rules. For ψ in (GT-MTZERO) and (GT-MTVAF) (ψ is the set of mutually disjoint vectors of session channels), see the proof of Proposition 5, page 48 (as noted there, when terms treated are subterms of a term without free session channels, then these groupings are uniquely determined). The typing is done by fixing (distinct) ψ at each type variable/inaction in $\mathcal{A}$ from which the typing starts. In (GT-MTINIT-S1/2-A/P) and (GT-MTINIT-O), S denotes "self" while O denotes "others", a denotes active and p denotes passive, similarly for communication rules (here "self" denotes the typing of the target thread when it is involved in interaction, the latter otherwise). The treatment of recursion and recursive variables follow the minimal typing system given in Figures 20, illustrated in §10.4, page 46.

$$\textsc{(GT-MTInit-S1a)}\ \frac{\Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta \cdot \tilde{s}[A] : \beta \quad \tau = \tau_1 \quad ch \notin \mathsf{fn}(\Gamma) \quad \mathsf{tvar}(\beta) = \{\tilde{\mathbf{t}}\}}{\Gamma,\ \overline{ch}@B : (\tilde{s})(\beta[\mathsf{end}/\tilde{\mathbf{t}}]) \vdash^{\tau} A^{\tau_1} \longrightarrow B^{\tau_2} : ch(\boldsymbol{\nu}\,\tilde{s})\,.\,\mathcal{A} \rhd \Delta}$$

$$\textsc{(GT-MTInit-S1p)}\ \frac{\Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta \cdot \tilde{s}[B] : \beta \quad \tau = \tau_2 \quad ch \notin \mathsf{fn}(\Gamma) \quad \mathsf{tvar}(\beta) = \{\tilde{\mathbf{t}}\}}{\Gamma,\ ch@B : (\tilde{s})(\beta[\mathsf{end}/\tilde{\mathbf{t}}]) \vdash^{\tau} A^{\tau_1} \longrightarrow B^{\tau_2} : ch(\boldsymbol{\nu}\,\tilde{s})\,.\,\mathcal{A} \rhd \Delta}$$

$$\textsc{(GT-MTInit-S2a)}\frac{\Gamma,\ \overline{ch}@B : (\tilde{s})\alpha \vdash^{\tau} \mathcal{A} \rhd \Delta \cdot \tilde{s}[A] : \beta \quad \tau = \tau_1 \quad ch \notin \mathsf{fn}(\Gamma) \quad \mathsf{tvar}(\beta) = \{\tilde{\mathbf{t}}\}}{\Gamma,\ \overline{ch}@B : (\tilde{s})(\alpha \sqcup \beta[\mathsf{end}/\tilde{\mathbf{t}}]) \vdash^{\tau} A^{\tau_1} \longrightarrow B^{\tau_2} : ch(\boldsymbol{\nu}\,\tilde{s})\,.\,\mathcal{A} \rhd \Delta}$$

$$\textsc{(GT-MTInit-S2p)}\frac{\Gamma,\ ch@B : (\tilde{s})\alpha \vdash^{\tau} \mathcal{A} \rhd \Delta \cdot \tilde{s}[B] : \beta \quad \tau = \tau_2 \quad ch \notin \mathsf{fn}(\Gamma) \quad \mathsf{tvar}(\beta) = \{\tilde{\mathbf{t}}\}}{\Gamma,\ ch@B : (\tilde{s})(\alpha \sqcup \beta[\mathsf{end}/\tilde{\mathbf{t}}]) \vdash^{\tau} A^{\tau_1} \longrightarrow B^{\tau_2} : ch(\boldsymbol{\nu}\,\tilde{s})\,.\,\mathcal{A} \rhd \Delta}$$

$$\textsc{(GT-MTInit-O)}\ \frac{\Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta \quad \{\tilde{s}\} \cap \mathsf{fc}(\Delta) = \emptyset \quad \tau \notin \{\tau_1, \tau_2\}}{\Gamma \vdash^{\tau} A^{\tau_1} \longrightarrow B^{\tau_1} : ch(\boldsymbol{\nu}\,\tilde{s})\,.\,\mathcal{A} \rhd \Delta}$$

$$\textsc{(GT-MTComm-Sa)}\ \frac{\Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta \cdot \tilde{s}[A] : \alpha_j \quad \Gamma \vdash e@A : \theta_j \quad \Gamma \vdash x@B : \theta_j \quad s \in \{\tilde{s}\} \quad \tau = \tau_1}{\Gamma \vdash^{\tau} A^{\tau_1} \to B^{\tau_2} : s\langle \mathsf{op_j},\ e,\ x\rangle\,.\,\mathcal{A} \rhd \Delta \cdot \tilde{s}[A] : s \uparrow op_j(\theta_j)\,.\,\alpha_j}$$

$$\textsc{(GT-MTComm-Sp)}\ \frac{\Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta \cdot \tilde{s}[B] : \alpha_j \quad \Gamma \vdash e@A : \theta_j \quad \Gamma \vdash x@B : \theta_j \quad s \in \{\tilde{s}\} \quad \tau = \tau_2}{\Gamma \vdash^{\tau} A^{\tau_1} \to B^{\tau_2} : s\langle \mathsf{op_j},\ e,\ x\rangle\,.\,\mathcal{A} \rhd \Delta \cdot \tilde{s}[B] : s \downarrow op_j(\theta_j)\,.\,\alpha_j}$$

$$\textsc{(GT-MTComm-O)}\ \frac{\Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta \quad \Gamma \vdash e@A : \theta_j \quad \Gamma \vdash x@B : \theta_j \quad s \notin \mathsf{fc}(\Delta) \quad \tau \notin \{\tau_1, \tau_2\}}{\Gamma \vdash^{\tau} A^{\tau_1} \to B^{\tau_2} : s\langle \mathsf{op_j},\ e,\ x\rangle\,.\,\mathcal{A} \rhd \Delta}$$

$$\textsc{(GT-MTAsgn)}\ \frac{\Gamma \vdash x@A : \theta \quad \Gamma \vdash e@A : \theta \quad \Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta}{\Gamma \vdash^{\tau} x^{\tau'} := e@A\,.\,\mathcal{A} \rhd \Delta}$$

$$\textsc{(GT-MTSum)}\ \frac{\Gamma_1 \vdash^{\tau} \mathcal{A}_1 \rhd \Delta_2 \quad \Gamma_1 \vdash^{\tau} \mathcal{A}_2 \rhd \Delta_2}{\Gamma_1 \sqcup \Gamma_2 \vdash^{\tau} \mathcal{A}_1 +^{\tau'} \mathcal{A}_2 \rhd \Delta_1 \sqcup \Delta_2}$$

$$\textsc{(GT-MTIf)}\ \frac{\Gamma \vdash e@A : \mathtt{bool} \quad \Gamma \vdash^{\tau} \mathcal{A}_1 \rhd \Delta_1 \quad \Gamma \vdash^{\tau} \mathcal{A}_2 \rhd \Delta_2}{\Gamma \vdash^{\tau} \mathsf{if}\ e^{\tau'}@A\ \mathsf{then}\ \mathcal{A}_1\ \mathsf{else}\ \mathcal{A}_2 \rhd \Delta_1 \sqcup \Delta_2}$$

$$\textsc{(GT-MTPar)}\ \frac{\Gamma_1 \vdash^{\tau} \mathcal{A}_1 \rhd \Delta_1 \quad \Gamma_2 \vdash^{\tau} \mathcal{A}_2 \rhd \Delta_2 \quad \mathsf{fsc}(\Delta_1) \cap \mathsf{fsc}(\Delta_2) = \emptyset}{\Gamma_1 \sqcup \Gamma_2 \vdash^{\tau} \mathcal{A}_1 \mid^{\tau'} \mathcal{A}_2 \rhd \Delta_1 \bullet \Delta_2}$$

$$\textsc{(GT-MTRes1)}\ \frac{\Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta,\ \tilde{s}_1 s \tilde{s}_2[A, B] : \alpha}{\Gamma \vdash^{\tau} (\boldsymbol{\nu} s)\ \mathcal{A} \rhd \Delta,\ \tilde{s}_1 \tilde{s}_2 : \bot} \qquad \textsc{(GT-MTRes2)}\ \frac{\Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta,\ \tilde{s}_1 s \tilde{s}_2 : \bot}{\Gamma \vdash^{\tau} (\boldsymbol{\nu} s)\ \mathcal{A} \rhd \Delta,\ \tilde{s}_1 \tilde{s}_2 : \bot}$$

$$\textsc{(GT-MTRes3)}\ \frac{\Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta,\ \varepsilon : \bot}{\Gamma \vdash^{\tau} (\boldsymbol{\nu} s)\ \mathcal{A} \rhd \Delta} \qquad \textsc{(GT-MTVar)}\ \frac{\Psi = \{\tilde{s}_i\} \quad \mathbf{t}\ \text{fresh}}{\Psi, X^A : \mathbf{t} \vdash^{\tau} X^A \rhd \bigcup_i \tilde{s}_i[A_i, B_i] : \mathbf{t}}$$

$$\textsc{(GT-MTRec)}\ \frac{\Gamma \cdot X^A : \mathbf{t} \vdash^{\tau} \mathcal{A} \rhd \Delta}{\Gamma \vdash^{\tau} \mathbf{rec}\ X^A\,.\,\mathcal{A} \rhd \mathtt{solve}(\mathbf{t}, \Delta)}$$

$$\textsc{(GT-MTZero)}\ \frac{\Psi = \{\tilde{s}_i\}}{\Gamma \vdash^{\tau} 0 \rhd \bigcup_i \tilde{s}_i[A_i, B_i]\mathtt{end}}$$

FIGURE 26. Threaded Minimal Typing Rules for Global Calculus

Below, when we write e.g. $\Gamma \vdash_{\mathsf{min}} \mathcal{A} \triangleright \Delta$, we assume the thread annotation in $\mathcal{A}$ conforms to the grouping of session names in Δ (cf. Proposition 2). The calculation $\sqcup_i \Delta_i$ is done in the obvious way, e.g. if we merge $\tilde{s}[B] : \alpha$ and $\tilde{s}[A] : \beta$ then we get $\tilde{s}[A \to B] : \overline{\alpha} \sqcup \beta$.

PROPOSITION 16 (Threaded Typing for Interaction). *Let $\{\tau_i\}$ exhausts the threads in $\mathcal{A}$. Then $\Gamma \vdash_{\mathsf{min}} \mathcal{A} \triangleright \Delta$ iff $\Gamma = \sqcup \Gamma_i$ and $\Delta = \sqcup \Delta_i$ such that, for each τ_i, we have $\Gamma_i \vdash^{\tau_i}_{\mathsf{min}} \mathcal{A} \triangleright \Delta_i$.*

Proof. Let $\Gamma \vdash_{\mathsf{min}} \mathcal{A} \triangleright \Delta$ and $\Gamma_i \vdash^{\tau_i}_{\mathsf{min}} \mathcal{A} \triangleright \Delta_i$. By structural induction of $\mathcal{A}$, we first show that $\Gamma_i \Subset \Gamma$ and $\Delta_i \Subset \Delta$, following the typing of $\mathcal{A}$ inductively in both Figures 20 and 26. For initialisation, there are two cases in the original minimal typing, while there are four cases in the threaded typing. (GT-MTINIT-S1/2, A/P) correspond to (GT-MTINIT-1/2) and (IH). The case when (GT-MTINIT-O) is inferred corresponding to (GT-MTINIT-1) is by induction and by noting $\Subset$ includes the subset relation. The case it corresponds to (GT-MTINIT-2), is immediate by induction. The case for communication is the same as the case of initialisation. The cases for assignment, restrictions and inaction are trivial since typings do not change in each case. Finally the conditional, sum and parallel composition are immediate from induction hypothesis and $\Subset$ being closed under $\sqcup$.

Next we show the types inferred simultaneously for each τ_i in $\Gamma_i \vdash^{\tau_i}_{\mathsf{min}} \mathcal{A} \triangleright \Delta_i$ fully covers those inferred in $\Gamma \vdash_{\mathsf{min}} \mathcal{A} \triangleright \Delta$, again by induction. This only matters in (GT-MTINIT-S1/2) and (GT-MTINIT-O), as well as (GT-MTCOM-S) and (GT-MTCOM-O), where the shape of typing differ. In each case, the non-threaded inference $\Gamma \vdash_{\mathsf{min}} \mathcal{A} \triangleright \Delta$ is covered in one of the (GT-MTINIT-S1/2) and (GT-MTCOM-S), hence done.

These two results together show Γ and Δ in $\Gamma \vdash_{\mathsf{min}} \mathcal{A} \triangleright \Delta$ coincide with the merging of Γ_i and Δ_i for each threaded inference $\Gamma_i \vdash^{\tau_i}_{\mathsf{min}} \mathcal{A} \triangleright \Delta_u$, □

Next we proceed to the step **(b)** outlined at the outset of §16.4, page 16.4. Our purpose is to relate each threaded typing of a global description and the minimal typing of the corresponding thread projection, bridging the global typing and the endpoint typing. For this purpose we use the following observation. As always we consider typings under a specific grouping of session channels (which is uniquely determined if a term is a subterm of a complete term).

LEMMA 9. *Assume $P_{1,2}$ are not replicated processes and $P_1 \bowtie P_2$. Let $\Gamma_i \vdash_{\mathsf{min}} P_i \triangleright \Delta_i$ $(i = 1,2)$ and $P_1 \sqcup P_2$ is typable, all under the same grouping of session names. Then $\Gamma_1 \sqcup \Gamma_2 \vdash_{\mathsf{min}} P_1 \sqcup P_2 \triangleright \Delta_1 \sqcup \Delta_2$.*

Proof. By induction on the generation rules of $\sqcup$ in Definition 21 and its corresponding rule in Definition 20. The only non-trivial case is the merging of input branch. Let $P \bowtie Q$ such that $P = s \triangleright \Sigma_{i \in J} op_i(y_i) \,.\, P_i$ and $Q = \sqcup s \triangleright \Sigma_{i \in K} op_i(y_i) \,.\, Q_i$. Then their merge is defined as:

$$s \triangleright \Sigma_{i \in J} op_i(y_i) \,.\, P_i \sqcup s \triangleright \Sigma_{i \in K} op_i(y_i) \,.\, Q_i \;\stackrel{\text{def}}{=}\; s \triangleright \left(\begin{array}{l} \Sigma_{i \in J \cap K} op_i(y_i) \,.\, (P_i \sqcup Q_i) + \\ \Sigma_{i \in J \setminus K} op_i(y_i) \,.\, P_i + \\ \Sigma_{i \in K \setminus J} op_i(y_i) \,.\, Q_i \end{array} \right)$$

which corresponds to MGBL-IN in Definition 20. Safely neglecting the invariant part (which has no effect on the argument), we set:

(1) $\Delta_1 = \tilde{s} : \Sigma_i s \downarrow op_i(\theta_i) \,.\, \alpha_i$.
(2) $\Delta_2 = \tilde{s} : \Sigma_j s \downarrow op_j(\theta_j) \,.\, \beta_j$..

By assumption $P_i \bowtie Q_i$ and, by induction hypothesis, $P_i \sqcup Q_i$ for $i \in I \cap J$ has the type $\alpha_i \sqcup \beta_i$. Hence we know $P \sqcup Q$ has the type $\Delta_1 \sqcup \Delta_2$. In $\Gamma_{1,2}$, we may be assuming interactions with services, which are treated similarly (using the rule for session invocation in Definition 21). □

PROPOSITION 17 (Thread Projection and Thread Typing). *Assume I is coherent and $\mathcal{A}$ is its consistent annotation. Then $\Gamma_i \vdash^{\tau_i}_{\mathsf{min}} \mathcal{A} \triangleright \Delta_i$ implies $\Gamma'_i \vdash_{\mathsf{min}} \mathsf{TP}(\mathcal{A},\ \tau_i) \triangleright \Delta'_i$ where Γ'_i changes the polarity of service channel typings as needed and Δ' is the result of taking off information on participants from type assignments in Δ (e.g. $\tilde{s}[B] : \alpha$ becomes $\tilde{s} : \alpha$).*

Proof. By induction on coherent $\mathcal{A}$, we relate the result of typing $\Gamma_i \vdash^{\tau_i}_{\mathsf{min}} \mathcal{A} \triangleright \Delta_i$ in Figure 26, on the one hand, and the result of taking $\mathsf{TP}(\mathcal{A},\ \tau_i)$ by Definition 22 (page 86) and typing it by the minimum typing in Figure 23 (page 64), using Lemma 9.

For (GT-MTINIT-S1A), consider the case when $\tau = \tau_1$. By assumption we have

$$\Gamma \vdash^{\tau} \mathcal{A} \triangleright \Delta \cdot \tilde{s}[A] : \beta$$

which, by induction hypothesis, implies:

$$\Gamma' \vdash_{\mathsf{min}} \mathsf{TP}(\mathcal{A},\ \tau) \triangleright \Delta' \cdot \tilde{s} : \beta$$

Hence we have, using (MTINIT-OUT1):

$$\Gamma,\ ch@B : (\tilde{s})(\beta[\mathtt{end}/\tilde{\mathbf{t}}]) \vdash_{\mathsf{min}} c \triangleright hA^{\tau_1} \to B^{\tau_1} : ch(\boldsymbol{\nu}\,\tilde{s})\,.\,\mathcal{A} \triangleright \Delta$$

as required. Similarly (GT-MTINIT-S1P) is reasoned using (MTINIT-IN).

For (GT-MTINIT-S2A), by assumption we have

$$\Gamma,\ \overline{ch}@B : (\tilde{s})\alpha \vdash^{\tau} \mathcal{A} \triangleright \Delta \cdot \tilde{s}[A] : \beta$$

which entails, by induction hypothesis,

$$\Gamma,\ \overline{ch}@B : (\tilde{s})\alpha \vdash \mathsf{TP}(\mathcal{A},\ \tau) \triangleright \Delta' \cdot \tilde{s} : \beta$$

We now consider the typing for the thread projection of

$$\mathcal{A}' = A^{\tau_1} \to B^{\tau_2} : ch(\boldsymbol{\nu}\,\tilde{s})\,.\,\mathcal{A}.$$

We first observe:

Claim. α and β has an upper bound w.r.t. $\sqsubseteq$ in Definition 28.

This is because, if not, when we put the session initialisation $A^{\tau_1} \to B^{\tau_2} : ch(\boldsymbol{\nu}\,\tilde{s})$, the thread τ_2 at ch and another passive thread at ch which contributed to $\overline{ch}@B : (\tilde{s})\alpha$ in the preceding derivation (there is at least one by the existence of $\overline{ch}$ in the base) cannot be merged, since for them to merge they should have exactly the same outputs after each common sequence of actions which come out as inputs for its dual interactions.

By the above Claim and by Proposition 15 (3) we know $\alpha \sqcup \beta = \alpha \vee \beta$, hence we have

$$\Gamma,\ \overline{ch}@B : (\tilde{s})(\alpha \sqcup \beta[\mathtt{end}/\tilde{\mathbf{t}}]) \vdash \mathsf{TP}(\mathcal{A}',\ \tau) \triangleright \Delta'$$

as required. (GT-MTINIT-S2P) is reasoned by the essentially identical argument. (GT-MTINIT-O), (GT-MTCOMM-SA), (GT-MTCOMM-SP), and (GT-MTCOM-O) are easy from (IH). For example, for (GT-MTCOM-O), the thread projection ignores the prefix, which is precisely what Definition 22 does.

For (GT-MTSUM), by assumption we have

$$\Gamma_1 \vdash^{\tau} \mathcal{A}_1 \triangleright \Delta_2 \quad \Gamma_1 \vdash^{\tau} \mathcal{A}_2 \triangleright \Delta_2$$

where the summand, hence the top threads of $\mathcal{A}_{1,2}$, have the thread τ'. We should consider two cases, one when $\tau' = \tau$ (in which case the thread projection does $\oplus$ to the thread projection of the summands) and $\tau' \neq \tau$ (in which case the thread projection does $\sqcup$). In the former case, the minimal endpoint typing has the service typing $\Gamma_1 \vee \Gamma_2$ and the session typing $\Delta_1 \vee \Delta_2$. First, for the former, since Γ has only output service typings (and assignments to term variables which are trivially identical) the same reasoning as Claim above shows $\Gamma_1 \vee \Gamma_2 = \Gamma_1 \sqcup \Gamma_2$. For session typings, consider the interacting (dual) behaviour which, since they are using the common session channels, should have the same thread. Since they should be mergeable, they should offer the same set of output branches at each common reachable point in the course of interactions, hence, dually, the corresponding types in Δ_1 and Δ_2 have the same inputs at each such point. Hence again we have $\Delta_1 \vee \Delta_2 = \Delta_1 \sqcup \Delta_2$, hence by (IH) we are done. The case when $\tau' \neq \tau$ is immediate from Lemma 9 and (IH).

For the remaining cases, (GT-MTIF) is reasoned as (GT-MTSUM), the rules (GT-MTPAR), (GT-MTRES1/2/3), and (GT-MTREC) are immediate from (IH), and both (GT-MTVAR) and (GT-MTZERO) are vacuous by identical shape of the corresponding typing rules. □

Finally we note:

PROPOSITION 18 (Typing for Merged Threads). *Let $\mathcal{A}$ be coherent and assume that, for some ch, we have $\{\tau_i\} = \mathsf{tpc}(\mathcal{A},\ ch)$. Then $\Gamma_i \vdash_{\mathsf{min}} \mathsf{TP}(\mathcal{A},\ \tau_i) \triangleright \Delta_i$ for each $\tau_i \in \{\tau_i\}$ implies $\bigsqcup_i \Gamma_i \vdash_{\mathsf{min}} \bigsqcup_i \mathsf{TP}(\mathcal{A},\ \tau_i) \triangleright \bigsqcup_i \Delta_i$.*

Proof. Immediate from Lemma 9 (note the order of merging does not matter by Proposition 12 (commutativity and associativity of $\sqcup$). □

We now prove Theorem 7 (1). Take the consistent annotation $\mathcal{A}$ and assume $\{\tau_i\}$ is the threads of $\mathcal{A}$ (which we assume does not include free term variables for simplicity). Since $\Gamma \vdash^{\mathsf{min}} \mathcal{A} \triangleright \Delta$, we have:

(104) $$\Gamma_i \vdash^{\tau_i}_{\mathsf{min}} \mathcal{A} \triangleright \Delta_i$$

for each $\tau_i \in \{\tau_i\}$ for which, by Proposition 16, we have (1) $\sqcup_i \Gamma_i = \Gamma$ and (2) $\sqcup_i \Delta_i = \Delta$. By Proposition 17, we also have, for each $\tau_i \in \{\tau_i\}$:

(105) $$\Gamma_i \vdash^{\mathsf{min}} \mathsf{TP}(\mathcal{A},\ \tau_i) \triangleright \Delta_i$$

Now consider $\mathcal{A}$ contains $\tau_{1,2,3}$ as the threads for a server at *ch* and consider

(106) $$\Gamma_i \vdash^{\tau_i}_{\mathsf{min}} \mathcal{A} \triangleright \Delta_i \quad (i = 1,2,3)$$

as well as

(107) $$\Gamma_i \vdash^{\mathsf{min}} \mathsf{TP}(\mathcal{A},\ \tau_i) \triangleright \Delta_i \quad (i = 1,2,3)$$

By Proposition 18 we have

(108) $$\bigsqcup_{i\in\{1,2,3\}} \Gamma_i \vdash^{\mathsf{min}} \bigsqcup_{i\in\{1,2,3\}} \mathsf{TP}(\mathcal{A},\ \tau_i) \triangleright \bigsqcup_{i\in\{1,2,3\}} \Delta_i$$

gives the replicated input at *ch*. Now

(109) $$\bigsqcup_{i\in\{1,2,3\}} \Gamma_i$$

gives the service typing at *ch* and zero or more client typings, in addition to assignment to variables, as required.

16.5. Proof of Theorem 7: (2) Soundness. For soundness, we observe:

LEMMA 10. *Assume $P_1 \bowtie P_2$ and let $P \overset{def}{=} P_1 \sqcup P_2$. Then $P_i \lessdot P$ $(i = 1,2)$.*

Proof. Immediate by the construction, noting $\lessdot$ is compatible. □

Let $\mathcal{A}$ be coherent and assume $\mathcal{A}$ has the threads $\mathcal{T} = \{\tau_i\}$. Let their thread projections be $P_i \overset{\text{def}}{=} \mathsf{TP}(\mathcal{A},\ \tau_i)$ for each $\tau_i \in \mathcal{T}$, which as a whole gives the indexed family of processes, $\{P_i\}$. Each P_i (which is to be strict an indice i together with the associated process P_i) is either a replicated input with a service channel as a subject; or else. If it is a replicated input, then it needs be merged with other replicated inputs at the same service channel, which should be merged. If not, it is never merged with other threads. We can now form a partition (a quotient set) of thread projections Ψ:

1. (1) Ψ partitions the family $\{P_i\}$: i.e. if $S_1, S_2 \in \Psi$ and $S_1 \neq S_2$ then $S_1 \cap S_2 = \emptyset$; and that $\bigcup \Psi = \{P_i\}$.
2. (2) If $S \in \Psi$ and $P_i, P_j \in \Psi$ such that $i \neq j$, then P_i and P_j are (replicated processes with the same service channel and hence are) to be merged in the endpoint projection.

Given Ψ, the endpoint projection of $\mathcal{A}$ is given as the result of merging processes in each $S \in \Psi$ (let the result be P_S) and placing them in each participant, i.e. with $\mathcal{P}$ being the set of participants, and $\Psi(A)$ (for each $A \in \mathcal{P}$) being the subsets of Ψ to belong to A:

(110) $$\mathsf{EPP}(\mathcal{A},\ \sigma) \overset{\text{def}}{=} (\boldsymbol{\nu}\tilde{t})(\Pi_{A\in\mathcal{P}} A[\Pi_{S\in\Psi(A)} P_S]_{\sigma_A})$$

Note reduction is never affected by participants information as far as projected processes go (since redexes are always inter-participants or involves only a single participant). Thus for legibility we neglect participant information from (110) and consider the initial configuration:

$$\text{(111)} \qquad \mathsf{EPP}(\mathcal{A},\ \sigma) \stackrel{\text{def}}{=} (\boldsymbol{\nu}\tilde{t})(\Pi_{S\in\Psi} \bigsqcup_{P_i \in S} P_i, \quad \sigma)$$

where σ aggregates all local stores, assuming all local variables are distinct, without loss of generality. Since we can easily regroup processes and state into participants, and this grouping never affects dynamics, this does not lose generality.

We show soundness using the reduction rules from Figure 21, page 57, adapted to the shape of (111). These rules are based on (Init), (Com), (Assign), (IfTrue), (IfFalse), (Sum) and (Rec), directly closing them on $\equiv$ and reduction contexts. Calling these adapted rules with the same names as the corresponding ones in Figure 21, we list these rules in the following (note that, as noted above, we are assuming variables are globally distinct, so that there is no need to local assignments). Below $C[\ ]_r$ denotes a reduction context, i.e. a context whose hole is not under any prefix (thus we can always set $C[P]_r \equiv (\boldsymbol{\nu}\tilde{s})(P|R)$ for some R and $\tilde{s}$).

$$\textsc{(Init)}\ \frac{P \equiv C_r[!ch(\tilde{s}).Q \mid \overline{ch}(\boldsymbol{\nu}\tilde{s}).R] \quad P' \equiv C_r[!ch(\tilde{s}).Q \mid (\boldsymbol{\nu}\tilde{s})(Q|R)]}{(P,\sigma) \ \to\ (P',\sigma)}$$

$$\textsc{(Com)}\ \frac{P \equiv C_r[s \rhd \Sigma_i \mathsf{op}_i(x_i).Q_i \mid \overline{s} \lhd \mathsf{op_j}\langle e\rangle R] \quad P' \equiv C_r[Q \mid R] \quad \sigma \vdash e \Downarrow V}{(P,\sigma) \ \to\ (P',\sigma[x \mapsto V])}$$

$$\textsc{(Assign)}\ \frac{P \equiv C_r[x := e.Q] \quad P' \equiv C_r[Q] \quad \sigma \vdash e \Downarrow V}{(P,\sigma) \ \to\ (P,\sigma[x \mapsto V])}$$

$$\textsc{(IfTrue)}\frac{P \equiv C_r[\mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2] \quad P' \equiv C_r[P_1] \quad \sigma \vdash e \Downarrow \mathsf{tt}}{(P,\sigma) \ \to\ (P',\sigma)}$$

$$\textsc{(Sum)}\ \frac{P \equiv C_r[P_1 \oplus P_2] \quad P' \equiv C_r[P_1]}{(P,\sigma) \ \to\ (P',\sigma)}$$

$$\textsc{(Rec)}\ \frac{P \equiv C_r[\mathbf{rec}\, X.Q] \quad (C_r[Q[(\mathbf{rec}\, X.Q)/X]],\ \sigma) \ \to\ (P',\ \sigma')}{(P,\sigma) \ \to\ (P',\sigma')}$$

We omit (IfFalse) (which follows (IfTrue)) and the symmetric case of (Sum). We can easily check these rules give the same reduction relation as those in Figure 21 when we incorporate participants. In the following we reason by induction on the height of derivation of these reduction rules, neglecting those used for inferring $\equiv$ (the induction on reduction rules are only non-trivial for recursion, since all other cases are the base cases).

In all of the following cases, we set:

$$\mathcal{A} \equiv (\boldsymbol{\nu}\tilde{t})\Pi_{0\leq i\leq n}\mathcal{A}_i$$

where each $\mathcal{A}_i$ is prime, i.e. is not itself a non-trivial parallel composition. For simplicity we safely neglect $(\boldsymbol{\nu}\tilde{t})$ from now on (since the hiding does not affect reduction) and consider only $\Pi_{0\leq i\leq n}\mathcal{A}_i$. Note in this case the endpoint projection can

We start from (Init). Then the redex is a pair, which is given by an input:

$$!ch(\tilde{s}).Q \stackrel{\text{def}}{=} \bigsqcup_{0\leq i\leq n} !ch(\tilde{s}).Q_i \ \equiv !ch(\tilde{s}).\bigsqcup_{0\leq i\leq n} Q_i \quad (\{!ch(\tilde{s}).Q_i\} \in \Psi)$$

and an output:

$$\overline{ch}(\boldsymbol{\nu}\tilde{s})R \in \{\overline{ch}(\boldsymbol{\nu}\tilde{s})R\} \qquad (\{\overline{ch}(\boldsymbol{\nu}\tilde{s})R\} \in \Psi)$$

Thus we can write down the reduction up to $\equiv$:

$$(!ch(\tilde{s}).Q \mid \overline{ch}(\boldsymbol{\nu}\,\tilde{s})R \mid S,\ \sigma) \ \rightarrow\ (!ch(\tilde{s}).Q \mid (\boldsymbol{\nu}\,\tilde{s})(Q|R) \mid S,\ \sigma) \tag{112}$$

Note we are neglecting participants for legibility, as stipulated above.

Now $\overline{ch}(\boldsymbol{\nu}\,\tilde{s})R$ come from the thread τ. Then τ should be the active top-level thread (since if not R cannot be an output, by wellthreadedness: cf. **(G3)** of Definition 14, page 75). Hence there is a complete interaction in $\mathcal{A}$, say $\mathcal{A}_0$, which starts from τ and its dual passive thread, say τ'. Since τ' is a passive thread starting from initialisation via ch, its thread projection is in $\{!ch(\tilde{s}).Q_i\}$, which we let (w.l.o.g.) $!ch(\tilde{s}).Q_0$. Hence we can write

$$\mathcal{A}_0 \equiv A^{\tau} \rightarrow B^{\tau'} : ch(\boldsymbol{\nu}\,\tilde{s})\,.\,\mathcal{A}'_0$$

which induces the following reduction:

$$(\mathcal{A},\ \sigma) \ \rightarrow\ (\boldsymbol{\nu}\,\tilde{s})(\mathcal{A}'_0|\Pi_{1\leq i\leq n}\mathcal{A}_i,\ \sigma) \tag{113}$$

Let the term on the right-hand side above, without including $(\boldsymbol{\nu}\,\tilde{s})$, be $\mathcal{A}'$. Note $\mathcal{A}'$ has the same set $\mathcal{T}$ of threads as that of $\mathcal{A}$ (it is possible τ no longer occurs in $\mathcal{A}'$, in which case we safely stipulate it exists as $\mathbf{0}^{\tau}$). Consider all the thread projections from $\mathcal{A}'$. By (113) we have:

$$\forall \tau_i \in (\mathcal{T}\backslash\{\tau,\tau'\}).\ \mathsf{TP}(\mathcal{A},\ \tau_i) \equiv \mathsf{TP}(\mathcal{A}',\ \tau_i).$$

For τ and τ' we have:

$$\mathsf{TP}(\mathcal{A}',\ \tau') \equiv Q \qquad \text{and} \qquad \mathsf{TP}(\mathcal{A}',\ \tau) \equiv R.$$

Note Q is no longer a replicated process at ch or any service channel, since it cannot be a passive input anymore. R is not a replicated process at a service channel either since if it should have given a fresh thread in $\mathcal{A}'$, not τ (by **(G1)**, freshness, Definition 14). Thus we can construct Ψ' for $\mathcal{A}'$ as we constructed Ψ for $\mathcal{A}$ as the same collection of sets of processes except

1. We replace $\{\overline{ch}(\boldsymbol{\nu}\,\tilde{s})R\} \in \Psi$ with $\{R\} \in \Psi'$.
2. We lose $!ch(\tilde{s}).Q_0$ from $\{!ch(\tilde{s}).Q_i\} \in \Psi$, obtaining $\{!ch(\tilde{s}).Q_i\}_{1\leq i\leq n} \in \Psi'$, and instead add $\{Q_0\} \in \Psi'$.
3. Otherwise Ψ' remains identical as Ψ.

We thus obtain the following endpoint projection of $((\boldsymbol{\nu}\,\tilde{s})\mathcal{A}',\sigma)$ as (neglecting participants as we stipulated):

$$\bigsqcup_{1\leq i\leq n} !ch(\tilde{s}).Q_i \mid (\boldsymbol{\nu}\,\tilde{s})(Q_0|R) \mid S \tag{114}$$

together with σ. We now compare (114) with the right-hand side of (112), the result of reducing the EPP of $\mathcal{A}$. By Lemma 10 we have $\bigsqcup_{1\leq i\leq n}!ch(\tilde{s}).Q_i \lessdot \bigsqcup_{0\leq i\leq n}!ch(\tilde{s}).Q_i$ as well as $Q_0 \lessdot \bigsqcup_{0\leq i\leq n} Q_i$, hence we are done.

For $(\textsc{Com})$, the reduction of the endpoint projection of $\mathcal{A}$ can be written as:

$$(s\Sigma_i\mathsf{op_i}(y_i).Q_i \mid \bar{s}\mathsf{op}_j(e).R \mid S,\ \sigma) \ \rightarrow\ (Q_j \mid R \mid S,\ \sigma) \tag{115}$$

Again the thread, say τ, for $\bar{s}\mathsf{op}_j(e).R \mid S$ for which we have $\{\bar{s}\mathsf{op}_j(e).R\} \in \Psi$, cannot start from an intermediate node, so it is a top-level active thread. Then the thread corresponding to $s\Sigma_i\mathsf{op_i}(y_i).Q_i$ should be its dual. Let the prime interaction starting from these two threads be $\mathcal{A}_0$. Then we can write:

$$\mathcal{A}_0 \equiv A \rightarrow B : s\langle\mathsf{op},\ e,\ y\rangle\,.\,\mathcal{A}'_0$$

by which we in fact know the sum we assumed in (115) is a singleton,[14] i.e. we can restrict our attention to:

$$(s\mathsf{op}(y).Q \mid \bar{s}\mathsf{op}_j(e).R \mid S,\ \sigma) \ \rightarrow\ (Q \mid R \mid S,\ \sigma) \tag{116}$$

[14] Only in a top-level immediate in-session communication, the direct endpoint projection of its passive thread becomes a singleton sum (note there can be no other ways). By sums and conditionals, non-trivial sums arise, as is usually the case in endpoint projections of meaningful protocols. Concrete examples are given in §16.7 and 16.8.

Now $\mathcal{A}$ has the following reduction:

$$(117) \qquad (\mathcal{A},\ \sigma)\ \rightarrow\ (\mathcal{A}'_0|\Pi_{1\le i\le n}\mathcal{A}_i,\ \sigma)$$

where, by construction, we have $\mathsf{TP}(\mathcal{A}'_0,\ \tau') = Q$ and $\mathsf{TP}(\mathcal{A}'_0,\ \tau) = R$, which are not to be merged with other threads (since again they cannot start from a passive service initialisation), otherwise remaining the same as $\mathcal{A}$. Thus in this case we have the precise correspondence, i.e. the result of reducing $\mathsf{EPP}(\mathcal{A},\ \sigma)$ is precisely $\mathsf{EPP}(\mathcal{A}',\ \sigma)$ up to $\equiv$.

For (ASSIGN), we have the reduction of the shape:

$$(118) \qquad (x := e.Q \mid R,\ \sigma)\ \rightarrow\ (Q \mid R,\ \sigma)$$

which means $x := e.Q$ comes from the initiating active (and passive) thread, say τ, of an interaction of the form:

$$\mathcal{A}_0 \quad \equiv \quad x^A_\tau := e.\mathcal{A}'_0$$

Hence (118) is precisely matched by the global reduction:

$$(119) \qquad (\mathcal{A},\ \sigma)\ \rightarrow\ (\mathcal{A}'_0|\Pi_{1\le i\le n}\mathcal{A}_i,\ \sigma)$$

By observing $\mathsf{TP}(\mathcal{A}'_0,\ \tau) \equiv Q$ and for each $\tau_i \in \mathcal{T}$ such that $\tau_i \neq \tau$ we have $\mathsf{TP}(\mathcal{A},\ \tau_i) \equiv \mathsf{TP}(\mathcal{A}',\ \tau_i)$, we are done.

For (IFTRUE), we can write the reduction as:

$$(120) \qquad (\mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2|R,\ \sigma)\ \rightarrow\ (P_1|R,\ \sigma)$$

when e evaluates to true in σ. Since $\{\mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2\} \in \Psi$ (i.e this is a singleton thread) we know there is a prime $\mathcal{A}_0$ of the following shape:

$$\mathcal{A}_0 \quad \stackrel{\text{def}}{=} \quad \mathsf{if}\ e@A\ \mathsf{then}\ \mathcal{A}'_{0t}\ \mathsf{else}\ \mathcal{A}'_{0f}$$

Thus we have the global reduction:

$$(121) \qquad (\mathcal{A},\ \sigma)\ \rightarrow\ (\mathcal{A}'_{0t}|\Pi_{1\le i\le n}\mathcal{A}_i,\ \sigma)$$

Note we have lost $\mathcal{A}'_{0f}$. If we write τ for the corresponding initial active (and passive) thread of $\mathcal{A}_0$, then we have, by $\mathsf{TP}(\mathcal{A}_0,\ \tau) \equiv \mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2$ and by the definition of thread projection for conditional:

$$(122) \qquad \mathsf{TP}(\mathcal{A}'_{0t},\ \tau) \quad \equiv \quad P_1$$

Now suppose $\tau_1,..,\tau_m$ occur in $\mathcal{A}_{0f}$. For simplicity let us just consider τ_1 and consider its thread projection at $\mathcal{A}_0$ (it can occur in other $\mathcal{A}_i$ with $i \neq 0$ but they remain invariant hence simply later added in parallel). In $\mathcal{A}_0$ we had:

$$\mathsf{TP}(\mathcal{A}_0,\ \tau_1)\ \equiv\ \mathsf{TP}(\mathcal{A}_{0t},\ \tau_1) \sqcup \mathsf{TP}(\mathcal{A}_{0f},\ \tau_1)$$

hence the thread projection of τ_1 in $\mathcal{A}'_{0t}$ is $\lessdot$ -smaller than that of τ_1 in $\mathcal{A}_0$, that is, for such τ_1:

$$\mathsf{TP}(\mathcal{A}',\ \tau_1)\ \lessdot\ \mathsf{TP}(\mathcal{A},\ \tau_1)$$

Since the projection of other threads remain identical, we know:

$$\forall \tau_i \in \mathcal{T}\backslash\{\tau\}.\ \mathsf{TP}(\mathcal{A}',\ \tau_i) \lessdot \mathsf{TP}(\mathcal{A},\ \tau_i)$$

By (122) we are done. (IFFALSE) is the same as (IFTRUE).

For (SUM), the original reduction is

$$(123) \qquad (P_1 \oplus P_2,\ \sigma)\ \rightarrow\ (P_1,\ \sigma)$$

which means we can set

$$\mathcal{A}_0 \quad \stackrel{\text{def}}{=} \quad \mathcal{A}'_{0l} + \mathcal{A}'_{0r}$$

Corresponding to e (122) in the case of (IFTRUE) above, we have

$$\mathsf{TP}(\mathcal{A}'_{0l},\ \tau) \quad \equiv \quad P_1$$

The rest is identical with the reasoning for (IFTRUE).

Finally for (Rec), we have the reduction:

$$(\mathbf{rec}\, X.P_0 \mid \Pi_{1\le i\le m}\mathbf{rec}\, X.P_i \mid R, \quad \sigma) \quad \rightarrow \quad (S,\, \sigma) \tag{124}$$

from

$$(P_0[\mathbf{rec}\, X.P_0/X] \mid \Pi_{1\le i\le m}\mathbf{rec}\, X.P_i \mid R, \quad \sigma) \,\rightarrow\, (S,\, \sigma) \tag{125}$$

where we specify all recursion terms coming from the same recursion in the global process, say:

$$\mathcal{A}_0 \quad \equiv \quad \mathbf{rec}\, X.\mathcal{A}_0'.$$

By (125) we know, writing $P_i' \overset{\text{def}}{=} P_i[\mathbf{rec}\, X.P_i/X]$:

$$(\Pi_{0\le i\le m}P_i' \mid R, \quad \sigma) \,\rightarrow\, (S,\, \sigma') \tag{126}$$

since if there is a reduction of a folded version, it is mechanical to check that its unfolded version also has the corresponding reduction. Further we can easily check that the latter is inferred by no more inference steps than the folded version.

Now we observe, assuming each P_i' comes from the projection at τ_i, we observe, for such τ_i:

$$\mathcal{E} \quad \overset{\text{def}}{=} \quad \mathsf{TP}(\mathcal{A'}_0[\mathbf{rec}\, X.\mathcal{A}_0'/X],\ \tau_i)$$

The unfolding can increase fresh passive threads if $\mathcal{A'}_0$ contains initialisation, which will be translated into replicated processes but these do not change the resulting processes (because an exact copy of such a passive thread already exists in $\mathcal{A}_0'$ and noting $P \sqcup (P \sqcup R) \equiv P \sqcup R$). By induction we know

$$(\mathcal{E},\, \sigma) \quad \rightarrow \quad (\mathcal{E}',\, \sigma') \tag{127}$$

such that

$$S \lessdot\ \equiv_{rec} \mathsf{EPP}(E',\ \sigma).$$

By (127) we also have:

$$(\mathcal{A},\, \sigma) \quad \rightarrow \quad (\mathcal{E}',\, \sigma')$$

hence we are done. This concludes all cases.

16.6. Proof of Theorem 7: (3) Completeness. Completeness is by induction on the derivation of reduction in the global calculus. As before, we consider adapted reduction rules which are taken modulo structural equality and reduction contexts, which are equivalent to the reduction rules in Figure 18 (page 39). We list these adapted rules in the following, using the same names as the original rules in Figure 18. As before, we assume all variables are distinct, including across participants, so that we can write e.g. $\sigma \vdash e \Downarrow V$ instead of $\sigma \vdash e@A \Downarrow V$. We again write $C_r[\,]$ for a reduction context in the grammar of interactions.

$$(\textsc{Init})\ \frac{I \equiv C_r[A \to B : b(\boldsymbol{\nu}\,\tilde{s})\,.\,I_0] \quad I' \equiv C_r[(\boldsymbol{\nu}\,\tilde{s})\ I_0]}{(\sigma, I)\ \to\ (\sigma, I')}$$

$$(\textsc{Comm})\ \frac{I \equiv C_r[A \to B : s\langle \mathsf{op},\ e,\ x\rangle\,.\,I_0] \quad I' \equiv C_r[I_0] \quad \sigma \vdash e \Downarrow V}{(\sigma,\ I)\ \to\ (\sigma[x@B \mapsto v],\ I')}$$

$$(\textsc{Assign})\ \frac{I \equiv C_r[x@A := e\,.\,I_0] \quad I' \equiv C_r[I_0] \quad \sigma \vdash e \Downarrow V}{(\sigma,\ I)\ \to\ (\sigma[x@B \mapsto v],\ I')}$$

$$(\textsc{IfTrue})\ \frac{I \equiv C_r[\mathsf{if}\ e\ \mathsf{then}\ I_{0t}\ \mathsf{else}\ I_{0f}] \quad I' \equiv C_r[I_{0t}] \quad \sigma \vdash e \Downarrow \mathsf{tt}}{(\sigma, I)\ \to\ (\sigma, I')}$$

$$(\textsc{Sum})\ \frac{I \equiv C_r[\mathsf{if}\ e\ \mathsf{then}\ I_{0l}\ \mathsf{else}\ I_{0r}] \quad I' \equiv C_r[I_{0l}]}{(\sigma, I)\ \to\ (\sigma, I')}$$

$$(\textsc{Rec})\ \frac{I \equiv C_r[\mathbf{rec}\ X.I_0] \quad (\sigma,\ C_r[I[(\mathbf{rec}\ X.I_0)/X]])\ \to\ (\sigma',\ I')}{(\sigma, I)\ \to\ (\sigma', I')}$$

We omit $(\textsc{IfFalse})$ and the symmetric case. Note the rules $(\textsc{Par})$ and $(\textsc{Res})$ are no longer necessary since they are absorbed in the above rules. Up to the application of the rules of $\equiv$, all rules above except $(\textsc{Rec})$ are the base cases. In the following reasoning, we use the obvious annotated version of these rules (which preserve thread labels across reduction, except when a new top-level parallel composition arises as a result of reduction, we take off its label).

In the following, by induction on the height of derivations, we show if

$$(\sigma, \mathcal{A}) \to (\sigma', \mathcal{A}')$$

then

$$\mathsf{EPP}(I,\ \sigma) \to (P', \sigma')$$

where

(128) $$\mathsf{EPP}(\mathcal{A}',\ \sigma') = (P'_0, \sigma') \text{ such that } P' \lessdot\ \equiv_{rec} P'_0.$$

Above, as in the proof of soundness, we neglect participants information in the endpoint processes, and aggregate the local states into σ, assuming all local variables are distinct. For simplicity we also abbreviate (128) to:

(129) $$(P,\ \sigma') \quad \lessdot\ \equiv_{rec} \quad \mathsf{EPP}(\mathcal{A}',\ \sigma')$$

We set

$$\mathcal{A} \quad \equiv \quad (\boldsymbol{\nu}\,\tilde{t})\Pi_{0 \le i \le n}\mathcal{A}_i$$

where each $\mathcal{A}_i$ is a prime interaction (i.e. an interaction which does not contain a non-trivial top-level parallel composition). Henceforth we safely neglect $(\boldsymbol{\nu}\,\tilde{t})$. As before, we let $\mathcal{T}$ to be the set of threads and Ψ to be the partition of the family of thread projections w.r.t these threads. We write $S, S', \ldots$ for the elements of Ψ.

For $(\textsc{Init})$, we can set:

$$\mathcal{A}_0 \stackrel{\text{def}}{=} A \to^{:} \tau_0 B^{\tau_1} ch(\boldsymbol{\nu}\,\tilde{s})\ .\,\mathcal{A}'_0$$

and consider the reduction:

(130) $$(\mathcal{A},\ \sigma) \quad \to \quad ((\boldsymbol{\nu}\,\tilde{s})\mathcal{A}'_0 | \Pi_{1 \le i \le n}\mathcal{A}_i,\ \sigma)$$

The endpoint projection of $(\mathcal{A},\ \sigma)$ contains a pair of an input and an output corresponding to the redex of this reduction:

$$!ch(\tilde{s}).Q \stackrel{\text{def}}{=} \sqcup_{0 \le i \le n} !ch(\tilde{s}).Q_i \ \equiv\ !ch(\tilde{s}).\sqcup_{0 \le i \le n} Q_i \quad (\{!ch(\tilde{s}).Q_i\}_{0 \le i \le n} \in \Psi)$$

and an output:

$$\overline{ch}(\boldsymbol{\nu}\,\tilde{s})R \in \{\overline{ch}(\boldsymbol{\nu}\,\tilde{s})R\} \qquad (\{\overline{ch}(\boldsymbol{\nu}\,\tilde{s})R\} \in \Psi)$$

Then we can write down $\mathcal{A}$ as $!ch(\tilde{s}.Q \mid \overline{ch}(\boldsymbol{\nu}\,\tilde{s}).R \mid S$. Thus we have a reduction:

(131) $$(!ch(\tilde{s}).Q \mid \overline{ch}(\boldsymbol{\nu}\,\tilde{s})R \mid S,\, \sigma) \;\rightarrow\; (!ch(\tilde{s}).Q \mid (\boldsymbol{\nu}\,\tilde{s})(Q|R) \mid S,\, \sigma)$$

By the exactly identical reasoning as in the corresponding case in the proof of soundness, the residual in (130) and that in e (131) are related in the way:

$$\mathsf{EPP}((\boldsymbol{\nu}\,\tilde{s})\mathcal{A}'_0|\Pi_{1\leq i\leq n}\mathcal{A}_i,\ \sigma) \;\lessdot\; (!ch(\tilde{s}).Q \mid (\boldsymbol{\nu}\,\tilde{s})(Q|R) \mid S,\, \sigma)$$

hence as required.

For (COMM), assume without loss of generality we have

$$\mathcal{A}_0 \stackrel{\text{def}}{=} A^{\tau_0} \to B^{\tau_1} : s\langle \mathsf{op},\ e,\ x\rangle\,.\,\mathcal{A}'_0$$

and consider the reduction:

$$(\sigma,\ \mathcal{A}) \quad\rightarrow\quad (\sigma',\ \mathcal{A}'_0|\Pi_{1\leq i\leq n}\mathcal{A}_i)$$

where $\sigma' = \sigma[x@A \mapsto V]$ with:

$$\sigma@A \vdash e \Downarrow V.$$

The thread projection of $\mathcal{A}_0$ to τ_0 has the form $\overline{s}\,\mathsf{op}_j(e).R$ (when the branching is a singleton we omit the symbol Σ, similarly henceforth) such that $\{\overline{s}\mathsf{op}_j(e).R\} \in \Psi$, while the one onto τ_1 has the form $s\mathsf{op}(y).Q$. Without loss of generality (cf. Proposition 11) we regard $\tau_{0,1}$ is used only in $\mathcal{A}_0$. Thus we can set:

$$\mathsf{EPP}(\mathcal{A},\ \sigma) \;\equiv\; (s\,\mathsf{op}(y).Q \mid \overline{s}\mathsf{op}_j(e).R \mid S,\, \sigma)$$

hence we have:

(132) $$\mathsf{EPP}(\mathcal{A},\ \sigma) \;\rightarrow\; (Q \mid R \mid S,\, \sigma')$$

(in (132), the update of the store is safely done due to our stipulation that all local variables are distinct.) By the same reasoning as in the corresponding case in the proof of soundness, we know

$$\mathsf{EPP}(\mathcal{A}'_0|\Pi_{1\leq i\leq n}\mathcal{A}_i,\ sigma') \;\lessdot\; (Q \mid R \mid S,\, \sigma')$$

as required.

For (ASSIGN), we can set

$$\mathcal{A}_0 \stackrel{\text{def}}{=} x^{\tau}@A := e.\mathcal{A}'_0.$$

We consider the reduction:

(133) $$(\sigma,\ \mathcal{A}) \quad\rightarrow\quad (\sigma',\ \mathcal{A}'_0|\Pi_{1\leq i\leq n}\mathcal{A}_i)\sigma'$$

with appropriate σ'. The thread projection onto τ has the shape $x := e.\mathsf{TP}(\mathcal{A}'_0,\ \tau)$, hence we have the reduction:

(134) $$(x@A := e.\mathsf{TP}(\mathcal{A}'_0,\ \tau) \mid R,\, \sigma) \;\rightarrow\; (\mathsf{TP}(\mathcal{A}'_0,\ \tau) \mid R,\, \sigma')$$

As in the corresponding case in the proof of soundness, (134) shows that all thread projections of $\mathcal{A}'$ except at τ remain invariant from that of $\mathcal{A}$, whose aggregate is R; and the projection onto τ precisely matches that of the residual of (133), hence as required.

For (IFTRUE), we can set

$$\mathcal{A}_0 \quad\stackrel{\text{def}}{=}\quad \mathsf{if}\ e^{\tau}@A\ \mathsf{then}\ \mathcal{A}'_{0t}\ \mathsf{else}\ \mathcal{A}'_{0f}$$

with which we have the reduction:

(135) $$(\mathcal{A},\, \sigma) \;\rightarrow\; (\mathcal{A}'_{0t}|\Pi_{1\leq i\leq n}\mathcal{A}_i,\, \sigma)$$

Observing

$$\mathsf{TP}(\mathcal{A}_0,\ \tau) \stackrel{\text{def}}{=} \mathsf{if}\ e\ \mathsf{then}\ \mathsf{TP}(\mathcal{A}'_{0t},\ \tau)\ \mathsf{else}\ \mathsf{TP}(\mathcal{A}'_{0f},\ \tau)$$

we have the reduction for the endpoint projection:

(136) $$(ifthenelsee\mathsf{TP}(\mathcal{A}'_{0t},\ \tau)\mathsf{TP}(\mathcal{A}'_{0f},\ \tau) \mid R,\, \sigma) \quad\rightarrow\quad (\mathsf{TP}(\mathcal{A}'_{0t},\ \tau) \mid R,\, \sigma)$$

where e evaluates to true in σ. By the reasoning for the corresponding case in the soundness proof, R in (136) may contain replicated inputs which are the result of merging complete threads from $\mathcal{A}'_{0f}$. Thus we obtain:

$$\mathsf{EPP}(\mathcal{A}'_{0t}|\Pi_{1\leq i\leq n}\mathcal{A}_i,\ \sigma)\ \lessdot\ (\mathsf{TP}(\mathcal{A}'_{0t},\ \tau)\mid R,\sigma)$$

as required.

(IFFALSE) and (SUM) are similarly reasoned.

For (REC), let:

$$\mathcal{A}_0 \quad \stackrel{\mathrm{def}}{=} \quad \mathbf{rec}\,X.\mathcal{A}'_0.$$

Further assume we have:

(137) $$(\mathcal{A},\ \sigma)\ \rightarrow\ (\mathcal{A}''_0|\Pi_{1\leq i\leq n}\mathcal{A}_i,\ \sigma')$$

The reduction (137) comes from, by the recursion rule above:

(138) $$(\mathcal{A}'_0[(\mathbf{rec}\,X.\mathcal{A}'_0)/X],\sigma)\ \rightarrow\ (\mathcal{A}''_0,\ \sigma')$$

Now the endpoint projections of $\mathcal{A}_0$ has the form:

(139) $$\mathsf{EPP}(\mathcal{A}_0,\ \sigma)\quad \stackrel{\mathrm{def}}{=}\quad ((\Pi P'_i)\mid R,\sigma)$$

where R is a collection of replicated processes and each P'_i is not replicated and has the shape:

$$P'_i \stackrel{\mathrm{def}}{=} \mathbf{rec}\,X.P_i.$$

We then consider the endpoint projection of the unfolding of $\mathcal{A}_0$:

(140) $$\mathsf{EPP}(\mathcal{A}'_0[(\mathbf{rec}\,X.\mathcal{A}'_0)/X],\ \sigma)\quad \stackrel{\mathrm{def}}{=}\quad ((\Pi P_i[P'_i/X])\mid R,\sigma)$$

Note the right-hand side of (140) is the n-times unfoldings of (139). Thus by induction hypothesis and applying the recursion rule in the endpoint processes n-times we obtain:

(141) $$\mathsf{EPP}(\mathcal{A},\ \sigma)\ \rightarrow\ \mathsf{EPP}(\mathcal{A}''_0|\Pi_{1\leq i\leq n}\mathcal{A}_i,\ \sigma')$$

as required. This exhausts all cases, establishing completeness.

This concludes the proof of Theorem 7.

16.7. An Example of Endpoint Projection. In the following we present an example of the endpoint projection to illustrate the formal notion introduced in the preceding sections. We use a fairly large toy example involving five participants. First, we explain the example in English; then we introduce the description in the global calculus; finally we project the description to endpoint processes.

Global Description in English. The example is an extension of the buyer-seller example introduced in section 2. The participants involved in this protocol are

(1) Buyer (B)
(2) Seller (S)
(3) Vendor (V)
(4) CreditChecker (CC)
(5) RoyalMail (RM)

The protocol proceeds as follows:

(1) Buyer requests a service ch_{CC} for company check to the credit checker CreditChecker by sending its name.
(2) At this point CreditChecker can either give a positive or negative answer.
(3) If the answer is positive:
    (a) Buyer asks Seller for a quote about product prod;
    (b) Seller then asks Vendor for service ch_V
    (c) Seller starts recursion and asks Vendor for a quote about product prod;
    (d) Vendor replies with a quote quote;
    (e) Seller forwards quote to Buyer increasing it by 10 units (quote+10);

(f) if the quote is reasonable ($\mathit{reasonable}(\mathsf{quote}+10)$) then:

  (i) Buyer sends Seller a confirmation (quoteOK) together with the credit (cred);

  (ii) Seller then contacts CreditChecker for checking the credit;

  (iii) If the credit is good then:

    (A) Seller contacts Shipper (service ch_{Sh});

    (B) Seller sends the delivery address;

    (C) Shipper sends a confirmation;

    (D) Seller forwards confirmation to Buyer;

  (iv) If the credit is bad:

    (A) CreditChecker tells Buyer;

    (B) Buyer tells Seller terminating the protocol;

(g) if the quote is not reasonable the protocol goes back to point 3c;

(4) If the answer is negative then the protocol terminates.

Global Description in the Calculus. The global description consists of several components for readability. We directly give annotated interaction. The main description is:

```
1.  $B^1 \rightarrow CC^2 : ch_{CC}s . CC^2 \rightarrow B^1 : s\langle \mathsf{ack} \rangle$ .
2.  $B^1 \rightarrow CC^2 : s\langle \mathsf{companyCheck},\ \mathsf{sellerName},\ \mathsf{compName} \rangle$ .
3.    {
4.      $CC^2 \rightarrow B^1 : s\langle \mathsf{good} \rangle . I_{\mathsf{good}}$
5.      $+$
6.      $CC^2 \rightarrow B^1 : s\langle \mathsf{bad} \rangle . \mathbf{0}$
7.    }
```

where I_{good} in Line 4 is:

```
1.  $B^1 \rightarrow S^3 : ch_S t . S^3 \rightarrow B^1 : r\langle \mathsf{ack} \rangle$ .
2.  $B^1 \rightarrow S^3 : t\langle \mathsf{quoteReq},\ \mathsf{prod},\ \mathsf{prod} \rangle$ .
3.  $S^3 \rightarrow V^4 : ch_V r$ .
4.  $V^4 \rightarrow S^3 : r\langle \mathsf{ack} \rangle$ .
5.  $\mathbf{rec}\ X^3 . \{$
6.    $S^3 \rightarrow V^4 : r\langle \mathsf{quoteReq},\ \mathsf{prod},\ \mathsf{prod} \rangle$ .
7.    $V^4 \rightarrow S^3 : r\langle \mathsf{quoteRes},\ \mathsf{quote},\ \mathsf{quote} \rangle$ .
8.    $S^3 \rightarrow B^1 : t\langle \mathsf{quoteRes},\ \mathsf{quote}+10,\ \mathsf{quote} \rangle$ .
9.    if $\mathit{reasonable}(\mathsf{quote})@B^1$ then
10.     $B^1 \rightarrow S^3 : t\langle \mathsf{quoteOK},\ \mathsf{cred},\ \mathsf{cred} \rangle$ .
11.     $S^3 \rightarrow CC^5 : ch_{CC}u$ .
12.     $CC^5 \rightarrow S^3 : u\langle \mathsf{ack} \rangle$ .
13.     $S^3 \rightarrow CC^5 : u\langle \mathsf{personalCreditCheck},\ \mathsf{cred{:}adr},\ \mathsf{cred{:}adr} \rangle$ .
14.     {
15.       $CC^5 \rightarrow S^3 : u\langle \mathsf{good} \rangle . I'_{\mathsf{good}}$
16.       $+$
17.       $CC^5 \rightarrow S^3 : u\langle \mathsf{bad} \rangle$ .
18.       $S^3 \rightarrow B^1 : t\langle \mathsf{yourCreditIsBad} \rangle . \mathbf{0}$
19.     }
20.   else $B^1 \rightarrow S^3 : t\langle \mathsf{quoteNotOK} \rangle . X^3$
21. }
```

where I'_{good} in Line 15 is:

$$\begin{array}{ll}
1. & S^3 \to R^6 : ch_R\, p\,. \\
2. & R^6 \to S^3 : p\langle \mathsf{ack}\rangle\,. \\
3. & S^3 \to R^6 : p\langle \mathsf{deliv},\ \mathsf{adr},\ \mathsf{adr}\rangle\,. \\
4. & R^6 \to S^3 : p\langle \mathsf{conf}\rangle\,. \\
5. & S^3 \to B^1 : t\langle \mathsf{conf}\rangle\,.\,\mathbf{0}
\end{array}$$

We can check these descriptions are typable, strongly connected, well-threaded and coherent. For connectedness, the descritption given above uses a lot of acks. As we discussed in the long version, many of these acks are in fact unnecessary by using a relaxed notion of connectedness.

End-Point Projection of the Global Interaction. Following the definition of EPP in the paper, we first project the global description onto each thread. The first one is Buyer's only thread.

$$\begin{array}{l}
\mathsf{TP}(I,\ 1) = \overline{ch_{CC}}(\boldsymbol{\nu} s)\,.\,s \triangleright \mathsf{ack}\langle\rangle\,.\,\overline{s} \triangleleft \mathsf{companyCheck}\langle \mathsf{sellerName}\rangle\,. \\
\qquad \{\ s \triangleright \mathsf{good}\langle\rangle\,.\,\overline{ch_S}(\boldsymbol{\nu} t)\,.\,t \triangleright \mathsf{ack}\langle\rangle\,.\,\overline{t} \triangleleft \mathsf{quoteReq}\langle \mathsf{prod}\rangle\,. \\
\qquad\quad \mathbf{rec}\, X\,.\ t \triangleright \mathsf{quoteRes}\langle \mathsf{quote}\rangle\,. \\
\qquad\qquad \mathsf{if}\ \mathit{reasonable}\{\mathsf{quote}\}\ \mathsf{then}\ \overline{t} \triangleleft \mathsf{quoteOK}\langle \mathsf{cred}\rangle\,. \\
\qquad\qquad\qquad\qquad \{t \triangleright \mathsf{yourCreditIsBad}\langle\rangle \\
\qquad\qquad\qquad\qquad + \\
\qquad\qquad\qquad\qquad\ t \triangleright \mathsf{conf}\langle\rangle\} \\
\qquad\qquad \mathsf{else}\ \overline{t} \triangleleft \mathsf{quoteNoteOK}\langle\rangle\,.\,X \\
\qquad + \\
\qquad s \triangleright \mathsf{bad}\langle\rangle\,.\,\mathbf{0}\ \ \}
\end{array}$$

Note this thread starts before the recursion and go through inside the (global) recursion. Thus the projected endpoint behaviour also contains recursion.

The next projection is onto the first thread of CreditChecker (note this participant has two threads, 2 and 5).

$$\begin{array}{l}
\mathsf{TP}(I,\ 2) = !\,ch_{CC}(s)\,.\,\overline{s} \triangleleft \mathsf{ack}\langle\rangle\,.\,s \triangleright \mathsf{companyCheck}\langle \mathsf{compName}\rangle\,. \\
\qquad \{\ \overline{s} \triangleleft \mathsf{good}\langle\rangle\,. \\
\qquad\ \oplus \\
\qquad\ \overline{s} \triangleleft \mathsf{bad}\langle\rangle\,.\,\mathbf{0}\ \ \}
\end{array}$$

Note no recursion is involved in this thread projection, simply because the thread 2 does not occur inside the recursion.

Next we jump to Thread 5, which is another component of CreditChecker.

$$\begin{array}{l}
\mathsf{TP}(I,\ 5) = !\,ch_{CC}(u)\,.\,\overline{u} \triangleleft \mathsf{ack}\langle\rangle\,.\,u \triangleright \mathsf{personalCreditCheck}\langle \mathsf{cred{:}adr}\rangle \\
\qquad (\overline{u} \triangleleft \mathsf{good}\langle\rangle \\
\qquad\ + \\
\qquad \overline{u} \triangleleft \mathsf{bad}\langle\rangle
\end{array}$$

Note the process does not include the recursion either. This is because it is inside a recursion and it initiates a new thread there. As a result the code is identical with the projection onto Thread 2.

We now move to the projection onto the unique thread of Seller, which is Thread 3.

$$\begin{aligned}
\mathsf{TP}(I,\ 3) = {} & !\,ch_S(t)\,.\,\overline{t} \lhd \mathsf{ack}\langle\rangle\,.\,t \rhd \mathsf{quoteReq}\langle\mathsf{prod}\rangle\,.\,\overline{ch_V}(\boldsymbol{\nu}\, r)\,.\,t \rhd \mathsf{ack}\langle\rangle\,.\\
& \mathbf{rec}\, X\,.\ \overline{r} \lhd \mathsf{quoteReq}\langle\mathsf{prod}\rangle\,.\,r \rhd \mathsf{quoteRes}\langle\mathsf{quote}\rangle\,.\\
& \quad \overline{t} \lhd \mathsf{quoteRes}\langle\mathsf{quote}+10\rangle\,.\\
& \quad \{\ t \rhd \mathsf{quoteOK}\langle\mathsf{cred}\rangle\,.\,\overline{ch_C C}(\boldsymbol{\nu}\, u)\,.\,u \rhd \mathsf{ack}\langle\rangle\,.\\
& \quad\ \ \overline{u} \lhd \mathsf{personalCreditCheck}\langle\mathsf{cred{:}adr}\rangle\,.\\
& \quad\quad \{u \rhd \mathsf{good}\langle\rangle\,.\,\overline{ch_R}(\boldsymbol{\nu}\, p)\,.\,p \rhd \mathsf{ack}\langle\rangle\\
& \quad\quad\ \ \overline{p} \lhd \mathsf{deliv}\langle\mathsf{adr}\rangle\,.\,p \rhd \mathsf{conf}\langle\rangle \overline{t} \lhd \mathsf{conf}\langle\rangle\\
& \quad\quad\ \ +\\
& \quad\quad\ \ u \rhd \mathsf{bad}\langle\rangle\,.\,\overline{t} \lhd \mathsf{CreditIsBad}\langle\rangle\ \}\\
& \quad +\\
& \quad t \rhd \mathsf{quoteNoteOK}\langle\rangle\,.\,X
\end{aligned}$$

As before, this thread starts outside of the recursion in the global description and is also used inside, so that both the recursion and the recursion variable are used as they are, leading to the recursive behaviour of the process. Note how the use of session functions as a way to handle recursion appropriately in EPP.

The projection onto the unique thread of Vendor follows.

$$\begin{aligned}
\mathsf{TP}(I,\ 4) = {} & !\,ch_V(r)\,.\,\overline{t} \lhd \mathsf{ack}\langle\rangle\,.\\
& \mathbf{rec}\, X\,.\,r \rhd \mathsf{quoteReq}\langle\mathsf{prod}\rangle\,.\,\overline{r} \lhd \mathsf{QuoteRes}\langle\mathsf{quote}\rangle\,.\,X
\end{aligned}$$

Finally we end with the projection onto Thread 6, giving the simple behaviour of RoyalMail.

$$\mathsf{TP}(I,\ 6) = !\,ch_R(p)\,.\,\overline{p} \lhd \mathsf{ack}\langle\rangle\,.\,p \rhd \mathsf{deliv}\langle\mathsf{adr}\rangle\,.\,\overline{p} \lhd \mathsf{conf}\langle\rangle$$

As before, Thread 6 does noon contain recursion since it is fully inside the (global) recursion, initiating a thread there.

As noted, there are two threads (2 and 5) that belong to the same class of equivalence i.e. they are part of the same service channel ch_{CC}. This means that we must merge the two threads in the final EPP. By applying the merge operator, and noting they are evidently mergeable, we get the following process:

$$\begin{aligned}
& !\,ch_{CC}(u)\,.\,\overline{u} \lhd \mathsf{ack}\langle\rangle\,.\\
& \quad u\ \rhd \left(\begin{array}{c}
\mathsf{personalCreditCheck}(\mathsf{cred{:}adr})\,.\ (\overline{u} \lhd \mathsf{good}\langle\rangle\ \oplus\ \mathsf{bad}\langle\rangle)\\
+\\
\mathsf{companyCheck}(\mathsf{compName})\,.\ (\overline{u} \lhd \mathsf{good}\langle\rangle\ \oplus\ \mathsf{bad}\langle\rangle)
\end{array}\right)
\end{aligned}$$

By which we have arrived at the endpoint behaviours of all participants realising the original global description.

The projection works because of the linear usage of channels inside each session and service channel principle, as well as the three well-structuredness conditions. We believe many business protocols conform to these conditions (modulo relaxation of connectedness we discussed in the long version). How these conditions can be extended in disciplined ways to allow more “untamed” protocols (such as those involving exceptions) to be treated in the theory, is an interesting subject of further studies.

16.8. Another EPP Example. We now consider a slight modification of the interaction considered throughout the whole paper, i.e. where we have the strong connectedness property

$$\begin{array}{l}
\mathtt{Buyer} \rightarrow \mathtt{Seller} : \mathtt{B2SCh}(s)\,.\,\mathtt{Seller} \rightarrow \mathtt{Buyer} : s[\mathtt{AckSession}]\,.\\
\quad \mathtt{Buyer} \rightarrow \mathtt{Seller} : s[\mathtt{RequestForQuote}]\,.\\
\quad\; \mathtt{Seller} \rightarrow \mathtt{Buyer} : s\langle \mathtt{QuoteResponse},\ v,\ \mathtt{quote}, x_{\mathtt{quote}}\rangle\,.\\
\quad\;\; (\mathtt{Buyer} \rightarrow \mathtt{Seller} : s[\mathtt{QuoteReject}] +\\
\quad\;\;\; \mathtt{Buyer} \rightarrow \mathtt{Seller} : s[\mathtt{QuoteAccept}]\,.\\
\quad\;\;\;\; \mathtt{Seller} \rightarrow \mathtt{Buyer} : s\langle \mathtt{OrderConfirmation}\rangle\,.\,\mathtt{Buyer} \rightarrow \mathtt{Seller} : s\langle \mathtt{AckConfirmation}\rangle\,.\\
\quad\;\;\;\;\; \mathtt{Seller} \rightarrow \mathtt{Shipper} : \mathtt{S2ShCh}(s')\,.\,\mathtt{Shipper} \rightarrow \mathtt{Seller} : s[\mathtt{AckSession}]\,.\\
\quad\;\;\;\;\;\; \mathtt{Seller} \rightarrow \mathtt{Shipper} : s'\langle \mathtt{RequestDelDetails},\ \mathtt{Buyer},\ x_{\mathtt{Client}}\rangle\,.\\
\quad\;\;\;\;\;\;\; \mathtt{Shipper} \rightarrow \mathtt{Seller} : s'[\mathtt{DeliveryDetails}, \mathtt{DD}, x_{\mathtt{DD}}]\,.\\
\quad\;\;\;\;\;\;\;\; \mathtt{Seller} \rightarrow \mathtt{Buyer} : s[\mathtt{DeliveryDetails}, x_{\mathtt{DD}}, x_{\mathtt{DD}}])
\end{array}$$

Note that we basically added few more interactions, just to keep the whole interaction strongly connected. By this we can then give the encoding

$$\begin{array}{l}
\mathtt{Buyer}\{\mathtt{B2SCh} \cdot n_1\langle s\rangle\,.\, s \rhd \mathtt{AckSession} \cdot n_2\,.\\
\qquad s \lhd \mathtt{RequestForQuote} \cdot n_3\,.\, s \rhd \mathtt{QuoteResponse} \cdot n_4(x_{\mathtt{quote}})\,.\\
\qquad s \lhd (\mathtt{QuoteReject} \cdot n_5 \mid\\
\qquad \mathtt{QuoteAccept} \cdot n_6\,.\, s \rhd \mathtt{OrderConfirmation} \cdot n_7\,.\, s \rhd \mathtt{AckConfirmation} \cdot n_8\,.\\
\qquad s \lhd \mathtt{DeliveryDetails} \cdot n_{12})\}_{\alpha} \mid\\
\mathtt{Seller}\{\mathtt{B2SCh} \cdot n_1(s)\,.\, s \rhd \mathtt{RequestForQuote} \cdot n_3\,.\, s \lhd \mathtt{AckSession} \cdot n_2\,.\\
\qquad s \lhd \mathtt{QuoteResponse} \cdot n_4\langle v_{\mathtt{quote}}\rangle\,.\\
\qquad s \rhd (\mathtt{QuoteReject} \cdot n_5 +\\
\qquad \mathtt{QuoteAccept} \cdot n_6\,.\, s \lhd \mathtt{OrderConfirmation} \cdot n_7\,.\, s \lhd \mathtt{AckConfirmation} \cdot n_8\,.\\
\qquad \mathtt{S2ShCh} \cdot n_9\langle s'\rangle\,.\, s \rhd \mathtt{AckSession} \cdot n_{13}\,.\\
\qquad s' \lhd \mathtt{RequestDelDetails} \cdot n_{10}\langle \mathtt{Buyer}\rangle\,.\, s \lhd \mathtt{DeliveryDetails} \cdot n_{11}(x_{\mathtt{DD}})\\
\qquad s \rhd \mathtt{DeliveryDetails} \cdot n_{12})\}_{\beta} \mid\\
\mathtt{Shipper}\{\mathtt{S2ShCh} \cdot n_9(s')\,.\, s \lhd \mathtt{AckSession} \cdot n_{13}\,.\\
\qquad s' \rhd \mathtt{RequestDelDetails} \cdot n_{10}(x_{\mathtt{Client}})\,.\, s \lhd \mathtt{DeliveryDetails} \cdot n_{11}(\mathtt{DD})\}_{\gamma}
\end{array}$$

17. Extension and Applications

17.1. Local variable declaration. We consider extensions and applications of the theory of EPP. First, we augment the syntax of global/local calculi with one useful construct, *local variable declaration*:

$$\texttt{newvar}\ x@A := e\ \texttt{in}\ I \qquad \texttt{newvar}\ x := e\ \texttt{in}\ P$$

This construct is indispensable especially for repeatedly invocable behaviours, i.e. those of services. Suppose a bookseller is invoked by two buyers simultaneously, each asking a quote for a different book. If these two threads share a variable, these two requests will get confused. The use of local variable declaration can avoid such confusion. The dynamics and typing of this construct are standard [**41**]. For endpoint projection, it is treated just as assignment.

17.2. Intra-Participant Interaction. In §10.1, we demanded that, in the grammar of service typing, $A \neq B$ in $\tilde{s}[A,B]$. This means well-typed global terms never have an intra-participant interaction. This is a natural assumption in a business protocol which primarily specifies inter-organisational interactions: however it can be restrictive in other contexts. Under connectedness (whose definition does not change), we can easily adapt the EPP theory to the inclusion of intra-participant interactions. First, the typing rules in Table 19, page 47, takes off (TComInv) and refines (TCom) so that the typing $\tilde{s}[A,B] : \alpha$ always reflects the direction of the interaction just inferred. This allows us to treat the case when A and B are equal (we demand interacting parties have distinct threads). The key change is in well-threadedness. When $A = B$, the condition **(G2)** (session consistency) in the definition of wellthreadedness is problematic since we do not know which of the two threads should be given to which participant. However stipulating the local causality condition, **(L)** in Definition 14 (page 75) solves this ambiguity. We reproduce this condition below.

Local Causal Consistency: *If there is a downward sequence of actions which starts from an active thread τ and ends with an action in which τ occurs for the first time (i.e. τ occurs in no intermediate actions in the sequence), then the latter τ occurs passively.*

Note that, by Proposition 10, this condition is a *consequence* of **(G1–3)** in the theory without intra-participant interaction so that we are not adding any extra constraint to inter-participant interactions.

17.3. Name Passing. An extension which is technically significant and practically useful is the introduction of *channel passing*. Channel passing is often essential in business protocols. As an example, consider the following refinement of Buyer-Seller Protocol.

> Buyer *wants to buy a hardware from* Seller*, but* Buyer *knows no* Seller*'s address on the net, i.e. it does not know* Seller*'s service channel. The only thing* Buyer *knows is a service channel* hardware *of a* DirectoryService*, which will send back the address of a* Seller *to* Buyer *which in turn interacts with that* Seller *through the obtained channel.*

In such a situation, Buyer has no prior knowledge of not only the seller's channel but also the participant itself. In a global description including its typing, participant names play a basic role. Can we leave the name of a participant and its channels unknown and still have a consistent EPP theory? This has been an open problem left in WS-CDL's current specification (which allows channel passing only for a fixed participant). Below we restrict our attention to service channel passing, excluding session name passing (which poses an additional technical issue [**24**]).

First, at the level of he endpoint calculus, it suffices to use the channel passing in the standard π-calculus.

$$\overline{\mathsf{DirectoryService}}(s).s(y).\overline{y}(t).P$$

which describes the initial behaviour of Buyer. Note y is an imperative variable, so that $y(t).P$ first *reads* the content of y then uses it for communication. The typing rules are extended accordingly.

In the global calculus, we introduce a syntactic variable Y, called a *participant placeholder*, for denoting anonymous participants. For example we can write:

$$A \to Y : x\tilde{s}.I \qquad\qquad Y \to Y' : s\langle \mathsf{op},\ e,\ y\rangle .I$$

The newly added $A \to Y : x\tilde{s}.I$ intuitively says:

> *A starts a session with session names $\tilde{s}$ on the service channel stored in x at the location A.*

The participant at which the service is offered is left unknown by placing a placeholder Y. However this will be instantiated once the variable x at A is inspected. For example, if x is evaluated to $ch@B$ in the store, the interaction takes place as in $A \to B : ch\tilde{s}.I$.

As an example, we present the buyer-seller-directory scenario discussed above:

$$\begin{array}{l}
\mathsf{Buyer} \to \mathsf{Directory} : \mathsf{hardware}s\,. \\
\quad \mathsf{Directory} \to \mathsf{Buyer} : s\langle \mathsf{sell},\ \mathsf{hware@amazon.co.uk},\ x\rangle\,. \\
\quad (\ \mathsf{Buyer} \to Y : xs'.Y \to \mathsf{Buyer} : s'\langle \mathsf{OK},\ \mathsf{data},\ y\rangle\ \mid \\
\qquad \mathsf{Buyer} \to \mathsf{Directory} : s\langle \mathsf{more},\ \text{""},\ z\rangle\,. \\
\qquad\ \mathsf{Directory} \to \mathsf{Buyer} : s\langle \mathsf{sell},\ \mathsf{hardware@pcworld.co.uk},\ x\rangle\,. \\
\qquad\quad \mathsf{Buyer} \to Y' : xs''.Y' \to \mathsf{Buyer} : s''\langle \mathsf{OK},\ \mathsf{data},\ y\rangle)
\end{array}$$

Note that, depending on the channel sent from Directory, Y and Y' are assigned to different participants.

The dynamics of the global calculus adds the rule which infers:

$$(\sigma,\ A \to Y : x\tilde{s}.I) \to (\sigma, (\boldsymbol{\nu}\,\tilde{s})\ I[B/Y])$$

whenever we have $\sigma@A(x) = ch@B$.

For types, we first extend the basic types θ with $(\tilde{s})\alpha$. We then add, with the obvious extension to the syntax of types:

$$\frac{\Gamma \vdash x@W_1 : (\tilde{s})\alpha \qquad \Gamma \vdash I \triangleright \Delta \cdot \tilde{s}[W_2, W_1] : \alpha}{\Gamma \vdash W_1 \to W_2 : x\tilde{s}.I \triangleright \Delta}$$

Other typing rules can be extended to deal with terms containing the participant variable Y in the same manner.

Finally, for the EPP theory, we need no change in the notion of connectedness. For well-threadedness, we first annotate placeholders regarding, e.g. $A \to Y : x\tilde{s}.I$ as the start of a new thread for Y, so we annotate it as $A^{\tau_1} \to Y^{\tau_2} : x\tilde{s}.I$ with τ_2 fresh. The definition of well-threadedness remains the same. Coherence however needs additional consideration. The variable $x@A$ can store different channels from different participants. For this purpose we use a typing system which records a possible set of assignment, in the shape $x@W_1 : C$ where C is a set of channels which may be instantiated into C. If some concrete channel is in C, the behaviour of that channel becomes constrained by coherence. This set C is inferred, starting from some fixed set, by adding ch (as in $x@W_1 : C \cup \{ch@B\}$) when we infer, e.g. $W_1 \to W_2 : s\langle \mathsf{op_j},\ ch@B,\ x\rangle .I$, where W_i can be either of participants or placeholders.

We give a flavour of how this extension works by the end-point projection of the example above. We first consider the annotated interaction for placeholders.

$$\mathsf{Buyer}^1 \to Y^3 : xs'.Y^3 \to \mathsf{Buyer}^1 : s'\langle \mathsf{OK},\ \mathsf{data},\ y\rangle$$

In the projection of this thread, we have placed a hole − which should be substituted with the appropriate service channels.

$$\mathsf{TP}(\mathcal{A},\ 3) = !\,\_(s').\overline{s'} \lhd \mathsf{OK}\langle \mathsf{data}\rangle$$

Thus, checking coherence consists in updating the definition of the function threads which induces the thread equivalence classes. But what equivalence classes should threads 3 and 4 belong to? We can use the prediction of all the possible values x can assume at runtime, i.e. hware@amazon.co.uk and hardware@pcworld.co.uk. We have to make sure that thread 3 belongs to both $\mathtt{tpc}(\mathcal{A},\ \mathsf{hware})$ and

$\mathtt{tpc}(\mathcal{A}, \mathsf{hardware})$. Then, if we are end-point projecting in amazon.co.uk we will substitute hware to _ in both thread projections, and if we are end-point projecting pcworld.co.uk we will substitute hardware instead.

17.4. Conformance. By relating global descriptions to their local counterpart, the presented theory allows us to make the best of the rich results from the study of process calculi. One such application is *conformance checking* (and its dynamic variant, runtime monitoring), discussed in Section 6. Our purpose is to have a formal criteria to say the communication behvaiour of a program *P* conforms to a global specification *I*.

In process algebras, conformance checking verifies whether the behaviour of a process follows that of a specification, the latter also given as (or representable as) a process. As a theoretical basis of the notion of conformance, we can use behavioural theories such as (inverse of) simulation or bisimulation. In the present context, our purpose is to verify whether an implemented system, say *P*, conforms to a well-typed description in the global calculus. Through the use of the end-point projection, we can reduce this problem to the conformance between endpoint processes, allowing us to use the standard theory. We illustrate one basic instance in the following.

The conformance checking we consider starts from the use of type information. Let *I* be a global description and *A* be a participant whose behaviour we are interested in. Let *S* be an EPP of *I* onto *A*. Let *P* be a program which is supposedly implements the specification *I* at the participant *A*. Since the end-point projection generates a process whose typing coincides with that of the original global description, we can first check whether *P* can be typed under the minimal typing associated with *S*. If we cannot, then *P* does not conform to *I*.

Once we know the typing of *P* conforms to that of *S*, we can proceed to validation of behavioural conformance. As an example, let *P* be given by:

$$\begin{array}{l}\overline{\mathsf{QuoteCh}}(\boldsymbol{\nu}\, s)\,.\, s \rhd \mathsf{Quote}\langle x\rangle\,.\\ \quad \mathsf{if}\ (x \leq 100)\ \mathsf{then}\ \overline{s} \lhd \mathsf{Accept}\langle\rangle\ \mathsf{else}\ \overline{s} \lhd \mathsf{Reject}\langle\rangle\end{array}$$

which may be used in the following configuration:

$$\begin{array}{l}\mathsf{Buyer}[P]\ \mid\ \mathsf{Seller}[\ !\mathsf{QuoteCh}(s)\,.\,\overline{s} \lhd \mathsf{Quote}\langle 300\rangle\,.\\ \qquad\qquad\qquad\qquad s \rhd (\mathsf{Accept}\langle\rangle\ +\ \mathsf{Reject}()\ +\ \mathsf{Restart}())]\end{array}$$

Suppose we wish to check whether this *P*, as the behaviour of Buyer, conforms to a global specification given as follows.

$$\begin{array}{l}\mathsf{Buyer} \longrightarrow \mathsf{Seller} : \mathsf{QuoteCh} s\,.\\ \quad \mathsf{Seller} \longrightarrow \mathsf{Buyer} : s\langle \mathsf{Quote},\ 300,\ x\rangle\,.\\ \qquad \mathsf{Buyer} \longrightarrow \mathsf{Seller} : s\langle \mathsf{Accept}\rangle + \mathsf{Buyer} \longrightarrow \mathsf{Seller} : s\langle \mathsf{Reject}\rangle\end{array}$$

We have already seen the end point projection of the specification above is given as follows.

$$\begin{array}{l}\mathsf{Buyer}[\ \overline{\mathsf{QuoteCh}}(\boldsymbol{\nu}\, s)\,.\, s \rhd \mathsf{Quote}\langle x\rangle\,.\\ \qquad (\ \overline{s} \lhd \mathsf{Accept}\langle\rangle \oplus \overline{s} \lhd \mathsf{Reject}\langle\rangle\ )\ ] \quad \mid\\ \mathsf{Seller}[\ !\mathsf{QuoteCh}(s)\,.\,\overline{s} \lhd \mathsf{Quote}\langle 300\rangle\,.\\ \qquad s \rhd (\mathsf{Accept}\langle\rangle\ +\ \mathsf{Reject}())]\end{array}$$

Let the endpoint process inhabiting Buyer above to be *S*. First, we can check the typability quite easily, in the sense that *P* is typable under the minimal typing of *S*. For example, the minimal type for the channel QuoteCh in *S* is given as

$$s\!\uparrow \mathsf{Quote}(\mathtt{int})\ s\downarrow\ (\ \mathsf{Accept}(\mathtt{null})\ +\ \mathsf{Reject}(\mathtt{null})\ )$$

Call this type α. Then α is *not* the minimal type of *P* at the same channel, but it is indeed a type assignable to *ch*.

Second, on the basis of well-typedness, we check the conformance of behaviour using a typed transition relation. The transition we consider has the sequent of the form:

$$(\Gamma \vdash (Q,\sigma) \triangleright \Delta) \xrightarrow{l} (\Gamma' \vdash (Q',\sigma') \triangleright \Delta')$$

Following the framework in [**6, 55**], $\xrightarrow{l}$ is *typed* in the sense that we only consider those actions deemed possible by Γ and Δ. In particular, if Γ or Δ specifies input branches which are less than Q, then inputs at the extra branches do not take place. We write $\overset{l}{\Longrightarrow}$ for the standard weak transition abstracting the silent transition. We can then define the following conformance relation (we include typability: as always we fix an appropriate grouping of free session channels if any).

> We say a type relation $\mathcal{R}$ over typed terms in the endpoint calculus is a *weak conformance* iff, whenever $\Gamma \vdash P_0 \mathcal{R} S_0 \triangleright \Delta$, we have: (1) Γ and Δ are the minimal typings of S; and (2) $(\Gamma \vdash P_0 \triangleright \Delta) \xrightarrow{l} (\Gamma' \vdash P_0' \triangleright \Delta')$ implies $(\Gamma \vdash S_0 \triangleright \Delta) \overset{l}{\Longrightarrow} (\Gamma' \vdash S_0' \triangleright \Delta')$ such that $\Gamma \vdash P_0' \mathcal{R} S_0' \triangleright \Delta$. If $\Gamma \vdash P_0 \mathcal{R} S_0$ for some weak conformance $\mathcal{R}$, we say P_0 *conforms to* S_0.

Thus (apart from typability) the conformance of P_0 to S_0 says that all visible behaviours of P_0 are within what is specified in S_0. We can indeed check that, for P and S above, P conforms to S in this sense. Thus this conformance is about safety: depending on the application needs, we may as well use a more stringent notion of conformance.

In summary, let I be a global description consisting of the participant A as well as other participants. Suppose P is a program which implements A's behaviour. Then we can check the conformance of P against the specification I by projecting I to A, which we call S, and check P conforms to S, using the conformance relation noted above. The conformance of P to S may be checked through either hand-calculation (coinduction), model checking, mechanical syntactic approximation, or runtime monitoring.

18. Related Work

18.1. Global Descriptions of Communication Behaviour (1). As far as we know, this work is the first to present the typed calculus based on global description of communication behaviour, integrated with the theory of endpoint projection. Global methods for describing communication behaviour have been practiced in several different engineering scenes in addition to WS-CDL (for which this work is intended to serve as its theoretical underpinning). Representative examples include the standard notation for cryptographic protocols [**37**], message sequence charts (MSC) [**28**], and UML sequence diagrams [**38**]. These notations are intended to offer a useful aid at the design/specification stage, and do not offer full-fledged programming language, lacking in e.g. standard control structures and/or value passing.

18.2. Global Descriptions of Communication Behaviour (2). DiCons (which stands for "Distributed Consensus"), which is independently conceived and predates WS-CDL, is a notation for global description and programming of Internet applications introduced and studied by Baeton and others [**4**], and would be the first fully expressive language for representing interactions based on a global method. DiCons chooses to use programming primitives close to user's experience in the web, such as web server invocation, email, and web form filing, rather than general communication primitives. Its semantics is given by either MSCs or direct operational semantics. DiCons does not use session types or other channel-based typing. An analogue of the theory of endpoint projection has not been developed in the context of DiCons.

18.3. Global Descriptions of Communication Behaviour (3). Petri-nets may also be viewed as offering a global description, though again they are more useful as a specification/analytical tool. As an example, a study by van der Aalst [**49**] presents an analysis of a business protocol showing how a description of an inter-participant business protocol can be implemented inside each participant without losing causal constraint, all represented in Petri Nets. While quite different in the formal apparatus and motivations, it shares a technical interest with our analysis in Section 14 as a causality analysis of interactions. The current lack of notions of types in Petri nets may make it hard to carry out the analogue of the full constructions as done in the present paper (which heavily rely on types).

18.4. Type Disciplines for Concurrent Programming Languages. The present work shares with many recent works its direction towards well-structured communication-centred programming using types. Pict [**43**] is the programming language based on the π-calculus, with rich type disciplines including linear and polymorphic types (which come from the studies on types for the π-calculus discussed in the next paragraph). Polyphonic C♯ [**5**] uses a type discipline for safe and sophisticated object synchronisation. Compagnoni, Dezani, Gay, Vasconcelos and others have studied interplay of session type disciplines with different programming constructs and program properties [**12, 18, 19, 24, 48, 50**]. The EPP theory offers a passage through which these studies (all based on endpoint languages and calculi) can be reflected onto global descriptions, as we have demonstrated for session types in the present work. In the context of session types, the present work extends the session structure with multiple session names which is useful for having parallel communications inside a session.

18.5. Theories of Types for the π-Calculus. Many theories of types for the π-calculus are studied. In addition to the study of session types mentioned above, these include input/output types [**34, 42**], linear types [**23, 30**], various kinds of behavioural types which among others include causality [**3, 6, 7, 25, 27, 46, 47, 55**] and combination of behavioural types and model checking for advanced behavioural analysis [**17, 44**], to name a few. Among others, behavioural types offer an advanced analyses for such phenomena as deadlock freedom. We are currently studying how these advanced type-based validation techniques on the basis of the present simple session type discipline will lead to effective validation techniques. Again these theories would become applicable to global descriptions through the link established by the EPP theory.

18.6. Process Calculus-based Analysis of Web Service. Gordon, Fournet, Bhargavan and Corin studied security-related aspects of web services in their series of works (whose origin lies in the the spi-calculus [**2**] and the applied π-calculus [**1**]). In their recent work [**10**], the authors have implemented part of WS-Security libraries using a dialect of ML, and have shown how annotated application-level usage of these security libraries in web services can be analysed with respect to their security properties by translation into the π-calculus [**11**]. The benefits of such a tool can be reflected onto the global descriptions through the theory of EPP, by applying the tool to projections.

Laneve and Padovani [**31**] give a model of orchestrations of web services using an extensions of π-calculus to join patterns. They propose a typing system for guaranteeing a notion of smoothness i.e. a constraint on input join patterns such that their subjects (channels) are co-located in order to avoid a classical global consensus problem during communication. Reflecting the centralised nature of orchestration (cf. footnote 1), neither a global calculus nor endpoint projection is considered.

A bisimulation-based correspondence between choreography and orchestration in the context of web services has been studied in [**14**] by Busi and others, where a notion of state variables is used in the semantics of the orchestration model. They operationally relate choreographies to orchestration. Neither strong type systems nor disciplines for end-point projection are studied in their work.

To provide a formal foundation of WS-CDL based on process calculi has been studied since the inception of WS-CDL. For example, Kavantzas suggested the use of fusion calculi [**39, 54**] for such a foundation (fusion calculi are formally equivalent to the π-calculus and other representative name passing process calculi). While we could have used any name passing calculi with enough expressive power, including the π-calculus and fusion calculi, as formalisms for endpoint processes, our choice of an imperative refinement of the π-calculus in the p resent paper has the merit in that we can have a direct, transparent correspondence between the presented global formalism and the endpoint formalism. A semantic preserving encoding of this applied version of the π-calculi into the pure π-calculus (or other pure name passing calculi, including fusion calculi) is an interesting topic for future study.

Bibliography

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *POPL '01: Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 104–115, New York, NY, USA, 2001. ACM Press.

[2] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, Jan. 1999.

[3] R. Amadio, G. Boudol, and C. Lhoussaine. The receptive distributed pi-calculus. In *Proc. of the FST-TCS '99*, volume 1738 of *LNCS*. Springer-Verlag, 1999.

[4] J. Baeten, H. van Beek, and S. Mauw. Specifying internet applications with DiCons. In *SAC '01*, pages 576–584, 2001.

[5] N. Benton, L. Cardelli, and C. Fournet. Modern concurrency abstractions for C#. *ACM Trans. Program. Lang. Syst.*, 26(5):769–804, 2004.

[6] M. Berger, K. Honda, and N. Yoshida. Sequentiality and the π-calculus. In *Proc. TLCA'01*, 2001.

[7] M. Berger, K. Honda, and N. Yoshida. Genericity and the pi-calculus. In *Proc. FOSSACS'03*, 2003.

[8] J. A. Bergstra and J. W. Klop. Algebra of communicating processes. *Theoretical Computer Science*, 37:77–121, 1985.

[9] G. Berry and G. Boudol. The Chemical Abstract Machine. *TCS*, 96:217–248, 1992.

[10] K. Bhargavan, C. Fournet, and A. Gordon. Verified reference implementations of WS-Security protocols. *To appear in WS-FM '06*, 2006.

[11] B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *CSFW*, pages 82–96, 2001.

[12] E. Bonelli, A. B. Compagnoni, and E. L. Gunter. Correspondence assertions for process synchronization in concurrent communications. *Journal of Functional Programming*, 15(2):219–247, 2005.

[13] G. Brown. A post at pi4soa forum. October, 2005.

[14] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro. Choreography and orchestration conformance for system design. In *COORDINATION*, volume 4038 of *LNCS*, pages 63–81, 2006.

[15] M. Carbone, M. Nielsen, and V. Sassone. A calculus for trust management. In *Proc. of the FST-TCS '04*, volume 3328 of *LNCS*, pages 161–173. Springer-Verlag, 2004.

[16] F. Cardone and M. Coppo. Type inference with recursive types: Syntax and semantics. *Inf. Comput.*, 92(1):48–80, 1991.

[17] S. Chaki, S. K. Rajamani, and J. Rehof. Types as models: model checking message-passing programs. In *POPL*, pages 45–57, 2002.

[18] M. Dezani-Ciancaglini, D. Mostrous, N. Yoshida, and S. Drossopoulou. Session Types for Object-Oriented Languages. In *Proceedings of ECOOP'06*, LNCS, 2006.

[19] S. Gay and M. Hole. Subtyping for session types in the pi calculus. *Acta Informatica*, 42(2-3):191–225, NOV 2005.

[20] C. A. Gunter. *Semantics of Programming Languages*. MIT Press, 1995.

[21] M. Hennessy and J. Riely. Resource access control in systems of mobile agents. In *Proceedings of HLCL '98*, volume 16.3 of *ENTCS*, pages 3–17. Elsevier Science Publishers, 1998.

[22] C. Hoare. *Communicating Sequential Processes*. Prentice Holl, New York, 1985.

[23] K. Honda. Composing processes. In *Proceedings of POPL'96*, pages 344–357, 1996.

[24] K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In *ESOP '98*, pages 122–138. Springer, 1998.

[25] K. Honda, N. Yoshida, and M. Berger. Control in the π-calculus. In *Proc. Fourth ACM-SIGPLAN Continuation Workshop (CW'04)*, 2004.

[26] G. Huet. *Rsolution d' quations dans les langages d'ordre* $1,2,\ldots,\omega$. Thse d' tat, Universit Paris 7, Paris, France, 1976.

[27] A. Igarashi and N. Kobayashi. A generic type system for the pi-calculus. In *POPL*, pages 128–141, 2001.

[28] International Telecommunication Union. Recommendation Z.120: Message sequence chart, 1996.

[29] N. Kavantzas. A post at petri-pi mailing list. August, 2005.

[30] N. Kobayashi, B. Pierce, and D. Turner. Linear types and π-calculus. In *Proceedings of POPL'96*, pages 358–371, 1996.

[31] C. Laneve and L. Padovani. Smooth orchestrators. In *FoSSaCS '06*, LNCS, pages 32–46, 2006.

[32] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, Berlin, 1980.

[33] R. Milner. Functions as processes. *MSCS*, 2(2):119–141, 1992.

[34] R. Milner. The polyadic π-calculus: A tutorial. In *Logic and Algebra of Specification*. Springer-Verlag, Heidelberg, 1993.

[35] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40,41–77, Sept. 1992.

[36] R. Milner, M. Tofte, and R. W. Harper. *The Definition of Standard ML*. MIT Press, 1990.

[37] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.

[38] OMG. Unified modelling language, version 2.0, 2004.

[39] J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *LICS'98*, 1998.

[40] PI4SOA. http://www.pi4soa.org.

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

[42] B. C. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–453, Oct. 1996.

[43] B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000.

[44] S. K. Rajamani and J. Rehof. Conformance checking for models of asynchronous message passing software. In *CAV*, pages 166–179, 2002.

[45] S. Ross-Talbot and T. Fletcher. Ws-cdl primer. Unpublished draft, May 2006.

[46] D. Sangiorgi. The name discipline of uniform receptiveness. In *ICALP '97*, volume 1256 of *LNCS*, pages 303–313, 1997.

[47] D. Sangiorgi. Modal theory. In *ICALP*, 2005.

[48] K. Takeuchi, K. Honda, and M. Kubo. An interaction-based language and its typing system. In *PARLE'94*, volume 817 of *LNCS*, pages 398–413, 1994.

[49] W. van der Aalst. Inheritance of interorganizational workflows: How to agree to disagree without loosing control? *Information Technology and Management Journal*, 2(3):195–231, 2002.

[50] V. T. Vasconcelos, A. Ravara, and S. J. Gay. Session types for functional multithreading. In *CONCUR '04*, LNCS, pages 497–511, 2004.

[51] W3C. Choreography description language, w3-cdl, web services choreography working group. http://www.w3.org/2002/ws/chor/.

[52] W3C WS-CDL Working Group. Web services choreography description language version 1.0. http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/.

[53] G. Winskel. *The formal semantics of programming languages*. MIT Press, 1993.

[54] L. Wischik and P. Gardner. Explicit fusions. *Theor. Comput. Sci.*, 340(3):606–630, 2005.

[55] N. Yoshida, M. Berger, and K. Honda. Strong Normalisation in the π-Calculus. In *Proc. LICS'01*, pages 311–322. IEEE, 2001. The full version to appear in *Journal of Inf. & Comp.*.

Appendix

APPENDIX A

Subject Reduction for the global calculus type system

This appendix establishes the subject reduction of the typing system for the global calculus, Theorem 1. Moreover, we will go through some intermediate results. We first note:

Lemma 1.

(1) (substitution, 1) If $\Gamma, X : \Delta \vdash I \triangleright \Delta'$ and $\Gamma \vdash I' \triangleright \Delta$ then $\Gamma \vdash I[I'/X] \triangleright \Delta'$.

(2) (substitution, 2) If $\Gamma \vdash \sigma$, $\Gamma \vdash \sigma(x@A) : \theta$ and $\Gamma \vdash v : \theta$, then $\Gamma \vdash \sigma[x@A \mapsto v]$.

Proof.

(1) The proof is direct from the typing rules.

(2) It is immediate from the typing rules.

□

We prove main theorem for this section.

Theorem 1.

(1) (Subject Congruence) If $\Gamma \vdash I \triangleright \Delta$ and $I \equiv I'$ then $\Gamma \vdash I' \triangleright =_\alpha \Delta$.

(2) (Subject Reduction, 1) Assume $\Gamma \vdash \sigma$. Then $\Gamma \vdash I \triangleright \Delta$ and $(\sigma, I) \to (\sigma', I')$ imply $\Gamma \vdash \sigma'$ and $\Gamma \vdash I \triangleright \Delta'$ for some Δ'.

(3) (Subject Reduction, 2) Assume $\Gamma \vdash \sigma$. Then $\Gamma \vdash I$ and $(\sigma, I) \to (\sigma', I')$ imply $\Gamma \vdash \sigma'$ and $\Gamma \vdash I'$.

Proof.

(1) We shall prove this by induction on the structural congruence rules.

- The proof is trivial for all cases that define $|$ and $+$ to be commutative monoids.
- When $(\mathbf{\nu} s)\ I \mid I' \equiv (\mathbf{\nu} s)\ (I \mid I')$ whenever $s \notin fn(I')$, suppose that $\Gamma \vdash (\mathbf{\nu} s)\ I \mid I' \triangleright \Delta$. Now, by rule (TPAR), we have that $\Delta = \Delta_1 \bullet \Delta_2$ such that $\Gamma \vdash (\mathbf{\nu} s)\ I \triangleright \Delta_1$ and $\Gamma \vdash I' \triangleright \Delta_2$. According to which rule we can apply for getting $\Gamma \vdash (\mathbf{\nu} s)\ I \triangleright \Delta_1$, we have three cases: (TRES-1), (TRES-2) and (TRES-3). We only analyse the first one as the other two are very similar. If we apply (TRES-1) then we have that there exists a Δ' such that $\Delta_1 = \Delta',\ \tilde{s}_1\tilde{s}_2 : \bot$ and $\Gamma \vdash I \triangleright \Delta',\ \tilde{s}_1 s \tilde{s}_2[A,B] : \alpha$. Now, applying again rule (TPAR), we have that $\Gamma \vdash I \mid I' \triangleright \Delta',\ \tilde{s}_1 s \tilde{s}_2[A,B] : \alpha\ \bullet\ \Delta_2$ if we can prove that $\Delta',\ \tilde{s}_1 s \tilde{s}_2[A,B] : \alpha\ \bullet\ \Delta_2$ is defined. But as $\Delta',\ \tilde{s}_1\tilde{s}_2 : \bot \bullet \Delta_2$ is defined then linearity is also respected when adding s because of the assumption $s \notin \mathsf{fn}(I')$ and so $s \notin \mathsf{fn}(\Delta_2)$. The proof concludes by applying the rule (TRES-1) again.

(2) In order to prove this, we shall prove a stronger result i.e. $\Gamma \vdash I \triangleright \Delta$ and $(\sigma, I) \to (\sigma', I')$ imply $\Gamma \vdash I \triangleright \Delta'$ and one of the following statements is true:

- $\Delta = \Delta'$
- $\Delta = \Delta_1, \tilde{s}[A,B] : \alpha$ and $\Delta' = \Delta_1, \tilde{s}[A,B] : \alpha'$ for $\alpha \to \alpha'$.

With $\alpha \to \alpha'$ we refer to a reduction on types, e.g. $s \uparrow op(\theta).\alpha \to \alpha$. Note that we also have $\mathsf{fsc}(\Delta) \subseteq \mathsf{fsc}(\Delta')$. The proof proceeds by induction on the depth of the derivation of $(\sigma, I) \to (\sigma', I')$.

Basic cases.

- (COMM). By hypothesis, we have $(\sigma, A \rightarrow B : s\langle \mathsf{op},\ e,\ x\rangle\,.I) \rightarrow (\sigma', I)$ and $\Gamma \vdash A \rightarrow B : s\langle \mathsf{op},\ e,\ x\rangle\,.I \rhd \Delta$. Now, the only applicable rules are (TCOMM) and (TCOMMINV). The cases are similar, so we shall inspect only the first one. We then have that $\Delta = \Delta_1 \cdot \tilde{s}[A,B] : \Sigma_{j \in J} s \uparrow op_j(\theta_j)\,.\alpha_j$ and $\Gamma \vdash I \rhd \Delta_1 \cdot \tilde{s}[A,B] : \alpha_j$.
- (INIT). We have $(\sigma, A \rightarrow B : ch(\boldsymbol{\nu}\,\tilde{s})\,.I) \rightarrow (\sigma, (\boldsymbol{\nu}\,\tilde{s})\ I_i)$. By applying the rule (TINIT), we have that $\Gamma',\ ch@B : (\tilde{s})\alpha \vdash A \rightarrow B : ch(\tilde{s})\,.I \rhd \Delta$ for $\Gamma = \Gamma',\ ch@B : (\tilde{s})\alpha$ and $\Gamma',\ ch@B : (\tilde{s})\alpha \vdash I \rhd \Delta \cdot \tilde{s}[B,A] : \alpha$. Now, by applying rule (TRES-1) repeatedly, we have $\Gamma',\ ch@B : (\tilde{s})\alpha \vdash (\boldsymbol{\nu}\,\tilde{s})\ I \rhd \Delta \cdot \varepsilon :\perp$ and by rule (TRES-3), we can get $\Gamma',\ ch@B : (\tilde{s})\alpha \vdash (\boldsymbol{\nu}\,\tilde{s})\ I \rhd \Delta$.
- (REC). We have $(\sigma, \mathbf{rec}\ X\,.I) \rightarrow (\sigma, I[\mathbf{rec}\ X\,.I/X])$ and $\Gamma \vdash \mathbf{rec}\ X\,.I \rhd \Delta$. The only applicable rule is (TREC), which implies $\Gamma \cdot X : \Delta \vdash I : \Delta$. But, by Lemma 1, we have that $\Gamma \vdash I[\mathbf{rec}\ X\,.I/X] \rhd \Delta$.
- (IFTT). From this semantics rule it follows that $(\sigma, \mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2) \rightarrow (\sigma, I_1)$ and from the hypothesis $\Gamma \vdash \mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2 \rhd \Delta$. Applying rule (TIF) we have $\Gamma \vdash I_1 \rhd \Delta$.
- (IFFF). Similar to previous case.
- (SUM). Similar to previous case.
- (ASSIGN). We have that $(\sigma, x@A := e\,.I) \rightarrow (\sigma', I)$ and $\Gamma \vdash x@A := e\,.I \rhd \Delta$. Now, applying the rule (TASSIGN) we get $\Gamma \vdash I \rhd \Delta$.

  Inductive cases.
- (PAR). By this rule, as we assume $(\sigma, I_1 \mid I_2) \rightarrow (\sigma', I_1' \mid I_2)$, we get $(\sigma, I_1) \rightarrow (\sigma', I_1')$. Moreover, there exist Δ_1 and Δ_2 such that $\Delta = \Delta_1 \bullet \Delta_2$ and $\Gamma \vdash I_1 \mid I_2 \rhd \Delta$, and such that, applying rule (TPAR), $\Gamma \vdash I_1 \rhd \Delta_1$ and $\Gamma \vdash I_2 \rhd \Delta_2$. Now, by induction hypothesis, it follows that there exists Δ_1' such that $\Gamma \vdash I_1' \rhd \Delta_1'$. If $\Delta_1 = \Delta_1'$ then the proof is trivial. Instead, if $\Delta_1 = \Delta'', \tilde{s}[A,B] : \alpha$ and $\Delta_1' = \Delta'', \tilde{s}[A,B] : \alpha'$ with $\alpha \rightarrow \alpha'$ then it is possible to prove that $\rightarrow$ preserves linearity i.e. α' respects linearity when merged with the type of Δ_2.
- (STRUCT). It follows from first point of this theorem.
- (RES). In this case we have

  $$\frac{(\sigma, I) \rightarrow (\sigma', I')}{(\sigma, (\boldsymbol{\nu}\,\tilde{s})\ I) \rightarrow (\sigma', (\boldsymbol{\nu}\,\tilde{s})\ I')}$$

  There are three possible cases for typing restriction, but we only analyze rule (TRES-1) as the other cases are similar. By applying this rule we must have $\Gamma \vdash (\boldsymbol{\nu}\,s)\ I \rhd \Delta = \Delta_1,\ \tilde{s}_1\tilde{s}_2 : \perp$ if and only if

  $$\Gamma \vdash I \rhd \Delta_1,\ \tilde{s}_1 s \tilde{s}_2[A,B] : \alpha$$

  Now, as $(\sigma, I) \rightarrow (\sigma', I')$, by induction hypothesis, we have that $\Gamma \vdash I' \rhd \Delta''$ and three possible cases:

  (a) $\Delta'' = \Delta_1, \tilde{s}_1 s \tilde{s}_2[A,B] : \alpha'$. If we now apply again rule (TRES-1), we get that $\Gamma \vdash (\boldsymbol{\nu}\,s)\ I' \rhd \Delta_1 \tilde{s}_1 \tilde{s}_2 :\perp$.

  (b) $\Delta_1 = \Delta_2, \tilde{s}'[C,D] : \alpha'$ and $\Delta'' = \Delta_2, \tilde{s}'[C,D] : \alpha'', \tilde{s}_1 s \tilde{s}_2[A,B] : \alpha$. Now, applying again rule (TRES-1), we get that $\Gamma \vdash (\boldsymbol{\nu}\,s)\ I' \rhd \Delta_2, \tilde{s}'[C,D] : \alpha'', \tilde{s}_1 \tilde{s}_2 :\perp$.

  (c) $\Delta'' = \Delta_1, \tilde{s}_1 s \tilde{s}_2[A,B] : \alpha$ and we trivially get $\Gamma \vdash (\boldsymbol{\nu}\,s)\ I' \rhd \Delta_1 \tilde{s}_1 \tilde{s}_2 :\perp$.

(3) Easy to prove from the previous point.

□

APPENDIX B

Subject Reduction for the end-point calculus type system

In here, we give the proofs for the end-point calculus type discipline. Mainly we give proofs for Theorem 2.

Proposition 8.

(1) (well-formedness) $\Gamma \vdash M \triangleright \Delta$ implies Γ and Δ are well-formed.
(2) (weakening, 1) Assume $\Gamma' \cdot \Gamma$ is well-formed. Then $\Gamma \vdash M \triangleright \Delta$ implies $\Gamma \cdot \Gamma' \vdash M \triangleright \Delta$.
(3) (weakening, 2) Let Δ be well-formed and $\tilde{s}$ are fresh. Then $\Gamma \vdash M \triangleright \Delta$ implies $\Gamma \vdash M \triangleright \Delta \cdot \tilde{s} : \bot$.
(4) (thinning) Assume $\mathsf{fc}(\Gamma') \cap \mathsf{fn}(M) = \emptyset$. Then $\Gamma \cdot \Gamma' \vdash M \triangleright \Delta$ implies $\Gamma \vdash M \triangleright \Delta$.
(5) (subsumption, 1) If $\Gamma, \overline{ch}@A : (\tilde{s})\alpha \vdash M \triangleright \Delta$ and $\alpha \preceq \beta$ then $\Gamma, \overline{ch}@A : (\tilde{s})\beta \vdash M \triangleright \Delta$
(6) (subsumption, 2) If $\Gamma \vdash M \triangleright \Delta \cdot \tilde{s}@A : \alpha$ and $\alpha \preceq \beta$ then $\Gamma \vdash M \triangleright \Delta \cdot \tilde{s}@A : \beta$.

Proof.

(1) By induction on the typing rules. The proof is similar to the global case.
(2) This proof proceeds by induction on the typing rules. Before this we need to prove that the result holds also for processes i.e. $\Gamma \vdash_A P \triangleright \Delta$ implies $\Gamma \cdot \Gamma' \vdash_A P \triangleright \Delta$. Also in this case, the proof proceeds by induction on the typing rules and is straighforward. We are now able to prove the result for any network.
   - (TPARTICIPANT). In this rule we have that $\Gamma \vdash A[P]_\sigma \triangleright \Delta$ if and only if $\Gamma \vdash_A P \triangleright \Delta$. By what we proved above, we have that $\Gamma \cdot \Gamma' \vdash_A P \triangleright \Delta$ and then, again by rule (TPARTICIPANT) we have $\Gamma \cdot \Gamma' \vdash A[P]_\sigma \triangleright \Delta$.
   - (TPAR-NW). We have that $\Gamma \vdash N_1 \mid N_2 \triangleright \Delta_1 \odot \Delta_2$ if and only if $\Gamma \vdash N_1 \triangleright \Delta_1$ and $\Gamma \vdash N_2 \triangleright \Delta_2$. By induction hypothesis we get $\Gamma \cdot \Gamma' \vdash N_1 \triangleright \Delta_1$ and $\Gamma \cdot \Gamma' \vdash N_2 \triangleright \Delta_2$ and by rule (TPAR-NW) again we get $\Gamma \cdot \Gamma' \vdash N_1 \mid N_2 \triangleright \Delta_1 \odot \Delta_2$
   - (TRES-NW,1), (TRES-NW,2), (WEAK-end-NW) and (WEAK-$\bot$-NW). Similar to previous case.
(3) The proof proceeds by induction on the typing rule. As in the previous proof, we need to prove something similar for the processes P.
(4) By induction on the typing rules.
(5) By induction on the typing rules, the results is a direct consequence of the rules typing communication.
(6) By induction on the typing rules. Similar to previous case.

□

We then give the proof of Proposition 9.

Proposition 9. (existence of minimal typing) Let Γ_0 be the *minimal service typing of M*. Then, if $\Gamma \vdash M \triangleright \Delta$ then we have $\Gamma_0 \vdash M \triangleright \Delta_0$ such that $\Gamma' \vdash M \triangleright \Delta'$ and Δ' using the same vectors of free session channels as Δ implies $\Gamma_0 \preceq \Gamma'$ and $\Delta_0 \preceq \Delta'$. **Proof.** (outline) By typing, we know M has all session channels abstracted by initialisation actions. For this reason we already know the grouping of bound session channels in M, determining uniquely vectors used in the introduction of end-types (for $\bot$ types an arbitrary grouping of session channels is enough). Starting from them, we can inductively construct minimum typings following the syntax of M. The second clause is its simple generalisation (note grouping of free session channels should be given beforehand to construct a typing). □

And then proof of Lemma 2.

Lemma 2.

(1) If $\Gamma \vdash A[P]_\sigma \rhd \Delta$, $\Gamma \vdash x@A : \theta$ and $\Gamma \vdash v : \theta$, then $\Gamma \vdash A[P]_{\sigma[x \mapsto v]} \rhd \Delta$.

(2) If $\Gamma, X : \Delta \vdash_A P \rhd \Delta'$ and $\Gamma \vdash_A Q \rhd \Delta$, then $\Gamma \vdash P[Q/X] \rhd \Delta$.

Proof.

(1) Trivial, from typing rules.

(2) This proof is similar to the global case i.e. by induction on the typing rules.

□

Below the proof for Lemma 3

Lemma 3. (subject congruence) $\Gamma \vdash M \rhd \Delta$ and $M \equiv N$ then $\Gamma \vdash N \rhd \Delta$.

Proof. By rule induction of the generation rules of $\equiv$. □

Finally the main theorem for this section.

Theorem 2. If $\Gamma \vdash N \rhd \Delta$ and $N \longrightarrow N'$ then $\Gamma \vdash N' \rhd \Delta$.

Proof. Standard, using Lemma 2. □

APPENDIX C

Alternative Inference Algorithm for Well-threadedness

In the following we show there is a simple algorithm which can inductively infer a consistent annotation of an interaction, if any: and informs a failure of doing so if it is not. We use the following notations.

(1) ℓ indicates a sequence[1] of *thread assignments*, where a session assignment is of the form $\langle \tau, A, \tilde{s} \rangle$ (which intuitively indicates communications done via any of $\tilde{s}$ by A should be in the thread τ).

(2) We write $\ell \cdot \ell'$ etc. for the concatenation of two strings, and $\langle \tau, A, \tilde{s} \rangle \in \ell$ when $\langle \tau, A, \tilde{s} \rangle$ occurs in ℓ.

DEFINITION 29 (Annotating Function). The *annotating function* $\gamma(I,\ \ell)$ is a partial function which maps a pair of (1) a thread assignment and (2) a well-typed, strongly-connected interaction which has at most one initiating participant to the corresponding annotated interaction, defined inductively as follows. In the first line we choose a fresh τ_2 by incrementing the maximum thread in ℓ.

$$\begin{array}{rcl}
\gamma(A \to B : ch(\boldsymbol{\nu}\, \tilde{s})\,.\,I,\ \ell\cdot\langle \tau_1, A, \tilde{t}\rangle) & \stackrel{\mathrm{def}}{=} & A^{\tau_1} \to B^{\tau_2} : ch(\boldsymbol{\nu}\, \tilde{s})\,.\,\gamma(I,\ \ell\cdot\langle \tau_1, A, \tilde{t}\tilde{s}\rangle\cdot\langle \tau_2, B, \tilde{s}\rangle) \\
 & & \qquad (\tau_2 \text{ fresh}) \\
\gamma(A \to B : s\langle \mathsf{op},\ e,\ x\rangle,\ \ell\cdot\langle \tau_1, A, \tilde{t}\rangle) & \stackrel{\mathrm{def}}{=} & A^{\tau_1} \to B^{\tau_2} : s\langle \mathsf{op},\ e,\ x\rangle\,.\,\gamma(I,\ \ell\cdot\langle \tau_1, A, \tilde{t}\rangle\cdot\langle \tau_2, B, \tilde{r}_1 s \tilde{r}_2\rangle) \\
 & & \qquad (\langle \tau_2, B, \tilde{r}_1 s \tilde{r}_2\rangle \in \ell, \text{ and } \ \tau_1 \neq \tau_2) \\
\gamma(x@A := e\,.\,I,\ \ell\cdot\langle \tau, A, \tilde{t}\rangle) & \stackrel{\mathrm{def}}{=} & x@A^{\tau} := e\,.\,\gamma(I,\ \ell\cdot\langle \tau, A, \tilde{t}\rangle) \\
\gamma(I_1 \mid I_2,\ \ell\cdot\langle \tau, A, \tilde{t}\rangle) & \stackrel{\mathrm{def}}{=} & (\gamma(I_1,\ \ell\cdot\langle \tau, A, \tilde{t}\rangle) \mid^{\tau} \gamma(I_2,\ \ell\cdot\langle \tau, A, \tilde{t}\rangle)) \\
\gamma(\mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2,\ \ell\cdot\langle \tau, A, \tilde{t}\rangle) & \stackrel{\mathrm{def}}{=} & \mathsf{if}\ e@A^{\tau}\ \mathsf{then}\ \gamma(I_1,\ \ell\cdot\langle \tau, A, \tilde{t}\rangle)\ \mathsf{else}\ \gamma(I_2,\ \ell\cdot\langle \tau, A, \tilde{t}\rangle) \\
\gamma(I_1 + I_2,\ \ell\cdot\langle \tau, A, \tilde{t}\rangle) & \stackrel{\mathrm{def}}{=} & \gamma(I_1,\ \ell\cdot\langle \tau, A, \tilde{t}\rangle) + \gamma(I_2,\ \ell\cdot\langle \tau, A, \tilde{t}\rangle) \\
\gamma(X^A,\ \ell\cdot\langle \tau, A, \tilde{t}\rangle) & \stackrel{\mathrm{def}}{=} & X^A_{\tau} \\
\gamma(\mathbf{rec}\, X^A\,.\,I,\ \ell\cdot\langle \tau, A, \tilde{t}\rangle) & \stackrel{\mathrm{def}}{=} & \mathbf{rec}\ {}^{\tau}X^A\,.\,\gamma(I,\ \ell\cdot\langle \tau, A, \tilde{t}\rangle)
\end{array}$$

Otherwise the function Ψ is not defined. We further set, for I which is well-typed and strongly connected, which has a unique initiating participant, and which does not contain hiding or free session channels:

$$\Psi(I) \quad \stackrel{\mathrm{def}}{=} \quad \gamma(I,\ \langle \tau, \mathsf{top}(I), \varepsilon\rangle)$$

Remark. Above in the communication case, the last condition $\tau_1 \neq \tau_2$ guarantees the choice of τ_2 is unique.

[1] A thread assignment contains redundancy, containing identical occurrences of a thread assignment, for the readability of the clauses for annotating functions.