Structured Communication-Centered Programming for Web Services

MARCO CARBONE, IT University of Copenhagen
KOHEI HONDA, Queen Mary, University of London
NOBUKO YOSHIDA, Imperial College London

8

This article relates two different paradigms of descriptions of communication behavior, one focusing on global message flows and another on end-point behaviors, using formal calculi based on session types. The global calculus, which originates from a Web service description language (W3C WS-CDL), describes an interaction scenario from a vantage viewpoint; the end-point calculus, an applied typed π-calculus, precisely identifies a local behavior of each participant. We explore a theory of end-point projection, by which we can map a global description to its end-point counterparts preserving types and dynamics. Three principles of well-structured description and the type structures play a fundamental role in the theory.

Categories and Subject Descriptors: D.3.1 [**Programming Languages**]: Formal Definitions and Theory; D.3.3 [**Programming Languages**]: Language Constructs and Features

General Terms: Theory, Design

Additional Key Words and Phrases: Communication, session types, process calculi, choreography, type system, web services, end-point projection

ACM Reference Format:
Carbone, M., Honda, K., and Yoshida, N. 2012. Structured communication-centered programming for Web services. ACM Trans. Program. Lang. Syst. 34, 2, Article 8 (June 2012), 78 pages.
DOI = 10.1145/2220365.2220367 http://doi.acm.org/10.1145/2220365.2220367

1. INTRODUCTION

Communication-Centered Programming. The explosive growth of Internet in the last decades has led to the de facto, global standards for naming schemes (URI, Domain Names), communication protocols (HTTP, TCP/IP) and message formats (XML). These elements offer a useful basis for building applications centring on communication among distributed agents through these standards. Such communication-centered applications are sometimes called *Web services*. Web services are an active area of infrastructural development, involving the major standardization bodies such as W3C and OASIS.

A concrete application area of communication-centered applications is *business protocols*. A business protocol is a series of structured and automated interactions among business entities. It is predominantly interdomain, is often required to satisfy some regulations, and demands clear shared understanding about its meaning. Some protocols such as industry standards will remain unchanged for a long time once specified; others may undergo frequent updates. Because of its inherent

Authors' addresses: M. Carbone, IT University of Copenhagen; email: carbonem@itu.dk; K. Honda, Department of Electronic Engineering and Computer Science, Queen Mary, University of London; N. Yoshida, Department of Computing, Imperial College London.

DOI 10.1145/2220365.2220367 http://doi.acm.org/10.1145/2220365.2220367

interorganizational nature, there is a strong demand for a common standard for specifying business protocols on a sound technical basis.

Global Description of Interaction. One of the standardization efforts for a language to specify business protocols is the Web Services Choreography Description Language (WS-CDL) [W3C WS-CDL Working Group 2004], developed by W3C WS-CDL Working Group in collaboration with π-calculus experts including the present authors as scientific advisors. WS-CDL is a specification language that offers a fully expressive global description language for channel-based communication, equipped with general control constructs (e.g., sequencing, conditionals and recursion), and is designed to support type-based formal validation from the outset. The underlying intuition behind the term *choreography* is:

Dancers dance following a global scenario without a single point of control

In choreography, the software designer no longer describes the behavior of the single peers (*end-point behavior*) but establishes how the various interactions between entities happen by giving a *global description* (choreography) of the system. In a traditional approach, the architect would describe the communication operations, for instance, an input, that must be performed at each peer. Unfortunately, this makes it very difficult to have a global view of how the whole system being designed works. On the other hand, choreography can picture the whole scenario of where and when a communication has to happen. The architect will now decide that, for instance, there will be a message from A to B and no longer think how this will be implemented at A (sending a message) or B (waiting to receive a message).

WS-CDL is conceived as a language for describing such a "global scenario": once specified, this scenario is to be executed by individual distributed processes without a single point of control.[1] Another significant feature of WS-CDL is its informal use of *sessions* in communication: at the outset of each execution of a business protocol, a session is the connection established between communicating parties so that messages from one session can be distinguished from messages in other sessions.

End-Point Projection. A global description of communication behavior is useful since it offers a clear view of its dynamic structure. Real execution of the description, however, is always through communication among distributed end-points that (as the notion of choreography dictates) may as well involve no centralized control. Thus we ask:

> How can we project a global description to end-point processes so that their interactions precisely realize the original global description?

Such a projection from a global scenario to end-point processes may be called *end-point projection* (EPP), following the design documents of WS-CDL. Having a universally agreed and well-founded notion of EPP is fundamental as an engineering basis of protocols, ranging from design of sound protocols to their implementations as end-point programs to run-time monitoring (see Section 8).

This Work. This article establishes a formal theory of EPP by introducing the two typed calculi for interaction, a distilled version of WS-CDL (a global calculus) and an applied π-calculus (an end-point calculus), and defining a mapping from the former to

[1]A related idea is *orchestration* where one master component, "conductor", directly controls activity of one or more slave components, which is useful when communicating parties can be placed under a common administrative domain, see [Honda et al. 2007; O'Hanlon 2006].

the latter. This mapping is highly nontrivial due to the different nature of descriptions: a global calculus directly describes interactions among multiple participants involving sequencing, branching and recursion, which differs from the end-point-based description given in the π-calculus. A central contribution of this work is the identification of three basic principles for global descriptions under which we can define a *sound* and *complete* EPP, in the sense that, through a given EPP, all and only globally described behavior is realized as communication among end-points. The three principles are: *connectedness* (a basic local causality principle), *well-threadedness* (a stronger locality principle based on session types [Bonelli et al. 2005; Dezani-Ciancaglini et al. 2006; Gay and Hole 2005; Honda et al. 1998; Takeuchi et al. 1994; Vasconcelos et al. 2004]) and *coherence* (a consistency principle for description of each participant in a global description). Schematically, the EPP mapping has the following shape:

$$I \quad \mapsto \quad A[P] \mid B[Q] \mid C[R] \mid \ldots$$

where I is a global description, A, B, and C are *participants* of the protocol and P, Q, and R are projections of I onto A, B, and C respectively. We will show that, when applied to well-typed interactions following the three principles, the EPP mapping thus defined satisfies *type preservation*, *soundness* and *completeness*. Furthermore, we will show that EPP also guarantees liveness. The EPP theory opens a conduit between global descriptions and accumulated studies on process calculi, allowing the use of the latter's rich theories for engineering aims. The EPP theory will be published as an associated document of WS-CDL 1.0 [Carbone et al. 2006b] (which contains many examples and full technical details), and will form part of its open-source implementation [PI4SOA 2008].

As an additional result, this article introduces a type inference result in session types for both global and end-point calculus.

Contributions and Outline. This article is an expanded version of Carbone et al. [2007], which differs from the conference version in several aspects. In particular, apart from the full definitions omitted from the extended abstract, for instance, Section 5, we have extended our theory by providing various algorithms for verifying the properties required by our formalism (Sections 3, 4, 5), provided formal proofs of our results and more examples (Section 6), and given an expanded discussion with the updated related work and the possible extensions that can be addressed in the future.

After the illustration of the key concepts in Section 2, the article presents the following technical contributions.

— The typed global calculus and its semantics (Section 3). We prove the central properties of the type discipline, including the existence of the minimal typing (Proposition 3.21), which entails, under the annotation of bound names, a sound and complete algorithmic type inference (Proposition 3.22); and the subject reduction 3.24 that entails type safety.
— The typed endpoint calculus and its semantics (Section 4), for which we prove the minimal typing property (Proposition 4.14); a sound and complete algorithmic type inference (Proposition 4.15); and the subject reduction 4.17, which entails communication safety (Corollary 4.19).
— The theory of endpoint projection (Section 5), where we present the key criteria for global specifications that enable consistent projection (Section 5.2, Section 5.3, Section 5.5.1, and Section 5.6).

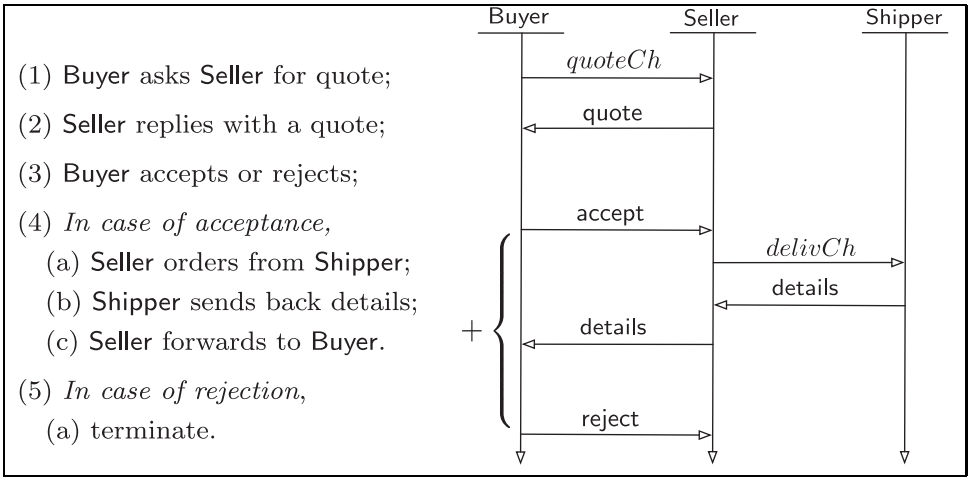
The emphasis is on illustrating, through concrete examples, how the EPP theory is defined and used for generating correct end-point processes. The present version also gives more detailed explanations of the typing systems of the global and end-point

calculi and their use for the EPP theory; and also provides comprehensive comparisons with related work.

In the remainder, Section 2 outlines the key ideas with an example, while Sections 3 and 4 introduce the global calculus and end-point calculus respectively. Section 5 develops the theory of end-point projection. In Section 6, we report an extended example of a global description and its end-point projection. Section 7 contains related work while Section 8 summarizes further results and applications of the EPP theory, and concludes with further topics. Appendix A–F give the auxiliary definitions and proofs.

2. THE BUYER-SELLER PROTOCOL

Global Descriptions. We outline the key technical ideas of this work using an example from [Ross-Talbot and Fletcher 2006], the "Buyer-Seller Protocol." The participants involved are a Buyer, a Seller and a Shipper. We describe the protocol with both text and a sequence diagram.



The diagram is ambiguous at the branching (+) actions in (4) and (5): the purpose of such diagrams is to offer an informal overview of the system omitting detailed control structures (choices, loops, etc.) and manipulation of values/states. The reason why such global descriptions (another well-known example is Message Sequence Charts [Broy 2005; Broy et al. 2007; ITU 1996]) are practiced in engineering is because they enable a clear grasp of the whole interaction structure, lessening synchronization and other errors at the design stage.

WS-CDL is intended to extend the virtues of such global notations to a full fledged description language. The global calculus, a model for WS-CDL, is built as formalization for precisely describing diagrams of this sort. Before introducing its syntax formally, we first outline its basic ideas using an example.

Figure 1(a) gives a description of the Buyer-Seller Protocol in the global calculus. In (a), Line 1 describes Action (1) in the protocol. The *quoteCh* is a *service channel*, which may be considered as a public URL for a specific service. The invocation marks the start of a session between Buyer and Seller: the ν-bound s is a *session channel*, a fresh name to be used for later communication in this session. Unlike standard process calculi, the syntax no longer describes input and output actions separately: the information exchange between two parties is directly described as one interaction.

(a) Protocol for Buyer-Seller Example

```
1.  Buyer → Seller : quoteCh(ν s).
2.  Seller → Buyer : s⟨quote, 300, x⟩. {
3.    {Buyer → Seller : s⟨accept⟩.
4.     Seller → Shipper : delivCh(ν t).
5.     Shipper → Seller : t⟨details, v, x⟩.
6.     Seller → Buyer : s⟨details, x, y⟩. 0 }
7.    +
8.    {Buyer → Seller : s⟨reject⟩. 0} }
```

(b) Protocol with Recursion

```
1.  Buyer → Seller : quoteCh(ν s).
2.  rec X. {
3.  Seller → Buyer : s⟨quote, q, x⟩.
4.  if reasonable(x)@Buyer then
5.    {Buyer → Seller : s⟨accept⟩.
6.     Seller → Shipper : delivCh(ν t).
7.     Shipper → Seller : t⟨details, v, x⟩.
8.     Seller → Buyer : s⟨details, x, y⟩. 0 }
9.  else
10.   {Buyer → Seller : s⟨reject⟩.
      q@Seller := q@Seller − 1. X } }
```

Fig. 1. Business protocols in the Global Calculus.

Line 2 describes Action (2), Seller's reply to Buyer. The session has already been started and now the two participants communicate using the session channel *s*. In addition, three factors involved: quote identifies the particular operation used in this communication, 300 is the quote sent by Seller; *x* is a variable located at Buyer where the communicated value will be stored (note that *x* is not bound).

Lines 3 and 8 describe Action (3), where Buyer communicates its choice (accept or reject) to Seller through *s*. Two series of actions that follow these choices are combined by + in Line 7. If accept is chosen, Seller invokes Shipper's service channel *delivCh*, creating a fresh session channel *t* (Line 4). Then in Line 5, Shipper sends the shipping details through *t*. Finally in Line 6, Seller forwards the details to Buyer by sending the value stored in variable *x*: here the protocol terminates. In Line 8, Buyer communicates reject, in which case the protocol immediately terminates.

Two Principles. Through our involvement in its design process, we found that WS-CDL is based on the following two engineering principles.

Principle 1 (*Service Channel Principle* (SCP)). Service channels can be shared and invoked repeatedly.

Principle 2 (*Session Principle* (SP)). A sequence of conversations belonging to a protocol should not be confused with other concurrent runs of this or other protocols by the participants, that is, each such sequence should form a logical unit of a conversation, or a *session*.

(SCP) corresponds to the repeated availability of replicated input channels in the π-calculus (called *uniformly receptive* Sangiorgi [1999] and *server channels* in Berger et al. [2001]), or, in practice, of public URLs. (SP) is a basic principle in many communication-centered programs, and can be given simple type abstraction with decidable type checking [Dezani-Ciancaglini et al. 2006; Honda et al. 1998; Vasconcelos et al. 2004].[2]

In Figure 1(a), we can observe how the distinction between service channels and session channels implements (SCP) and (SP): sessions offer logical grouping of threads of interactions, where each thread starts with a procedure-call-like service invocation at a service channel and carry out in-session communications at associated session channels. This point can be seen more clearly in Figure 1(b), a refinement of (a). In (b), if Buyer chooses reject, the protocol recurs to Line 3, after decrementing the quote. In Line 4, a unary predicate `reasonable`(x) is evaluated at Buyer's site ("@" indicates

[2]In implementations of Web services, sessions are implemented using so-called *correlation identities* (which may be considered as nonces in cryptographic protocols).

a location, similarly in Line 10). The session notation makes it clear that all quote-messages from Seller to Buyer in the recursion are done within a single session. If (SP) is violated, messages could be confused, for instance, Shipper in Line 4 may receive a message from Buyer that was sent to Seller in Line 3 since the three participants are running in parallel.

Notice that according to this choreography, the shipper is only invoked once since taking the branch between lines 5 and 8 implies termination of recursion (there is no tail X). Section 5 will show that such session information plays a crucial role in a tractable end-point projection.

3. THE GLOBAL CALCULUS

3.1. Syntax

In the sequel, $I, I', \ldots$ denote *terms* of the calculus, also called *interactions*; $ch, ch' \ldots$ range over *service channels*, intuitively denoting the shared channels of (Web) services; $s, t, r \ldots$ range over *session channels*; $\tilde{s}$ indicates a vector of session channels; $A, B, C, \ldots$ range over *participants*; $x, y, z, \ldots$ over variables local to each participant; $X, X', \ldots$ over *term variables*; and $e, e', \ldots$ over arithmetic and other first-order expressions excluding service and session channels. The syntax of the global calculus [Carbone et al. 2006a] is given by BNF in the following definition.

Definition 3.1 (*Global Calculus Syntax*).

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

(init) denotes a session initiation by A via B's service channel ch, with fresh session channels $\tilde{s}$ and continuation I. (comm) denotes an in-session communication over a session channel s, where op is an operator name, which is used as a label for selecting communication (just like a label in records and tagged unions). Note that y does not bind in I: in the operational semantics (given in Section 3.2) variables will be evaluated in a local state as in imperative languages. "|" and "+" denote respectively parallel and choice of interactions. (cond) and (assign) are standard conditional and assignment. Assignment can also be seen as an internal communication $A \to A : s\langle \mathsf{assign},\ e,\ x\rangle.\ I$. Explicit $x@A/e@A$ indicates the variable/boolean expression x/e is located at A. X^A and $\mu X^A.\ I$ (where the variable X^A is bound in I) implement recursion. Note that the participant name appearing in the recursion and the recursion variable has no importance in this section: we will therefore omit it now and reconsider it in Section 5. $\mathbf{0}$ denotes termination. We often omit $\mathbf{0}$ and empty vectors. $(\nu s)I$ is the CCS-like name restriction, binding s in I. For the sake of presentation, we will often write $(\nu s_1) \ldots (\nu s_k)$ as $(\nu \tilde{s})$. Since hiding is only generated by session initiation, that is, it is mainly used at runtime, we stipulate the following assumption.

$$\text{(G-Init)}\ \frac{}{(\sigma,\ A\to B : ch(\boldsymbol{\nu}\,\tilde{s})\boldsymbol{.}\ I) \rightsquigarrow (\sigma,\ (\boldsymbol{\nu}\tilde{s})I)} \qquad \text{(G-Asgn)}\ \frac{\sigma \vdash e@A \Downarrow v \quad \sigma' = \sigma[x@A \mapsto v]}{(\sigma,\ x@A := e\boldsymbol{.}\ I) \rightsquigarrow (\sigma',\ I)}$$

$$\text{(G-Com)}\ \frac{\sigma \vdash e@A \Downarrow v \quad \sigma' = \sigma[x@B \mapsto v]}{(\sigma,\ A\to B : s\langle \mathsf{op},\ e,\ x\rangle\boldsymbol{.}\ I) \rightsquigarrow (\sigma',\ I)} \qquad \text{(G-Sum)}\ \frac{}{(\sigma,\ I_1 + I_2) \rightsquigarrow (\sigma,\ I_1)}$$

$$\text{(G-IfT)}\ \frac{\sigma \vdash e@A \Downarrow \mathsf{tt}}{(\sigma,\ \mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2) \rightsquigarrow (\sigma, I_1)} \qquad \text{(G-Rec)}\ \frac{(\sigma,\ I[\mu X.I/X]) \rightsquigarrow (\sigma',\ I')}{(\sigma,\ \mu X.I) \rightsquigarrow (\sigma',\ I')}$$

$$\text{(G-IfF)}\ \frac{\sigma \vdash e@A \Downarrow \mathsf{ff}}{(\sigma,\ \mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2) \rightsquigarrow (\sigma,\ I_2)} \qquad \text{(G-Res)}\ \frac{(\sigma,\ I) \rightsquigarrow (\sigma',\ I')}{(\sigma, (\boldsymbol{\nu}\tilde{s})I) \rightsquigarrow (\sigma',\ (\boldsymbol{\nu}\tilde{s})I')}$$

$$\text{(G-Struct)}\ \frac{I \equiv I''\ \ (\sigma,\ I) \rightsquigarrow (\sigma',\ I')\ \ I' \equiv I'''}{(\sigma,\ I'') \rightsquigarrow (\sigma',\ I''')} \qquad \text{(G-Par)}\ \frac{(\sigma,\ I_1) \rightsquigarrow (\sigma',\ I_1')}{(\sigma, I_1 \mid I_2) \rightsquigarrow (\sigma',\ I_1' \mid I_2)}$$

Fig. 2. Reduction relation for the global calculus.

Assumption 3.2. A hiding never occurs inside a prefix, sum or conditional.

The free session channels and term variables are defined as usual and are denoted by $\mathsf{fsc}(I)$ and $\mathsf{fv}(I)$, respectively. The set $\mathsf{channels}(I)$ denotes the set of service channels occurring in I. For the formal definition see Appendix A.1.

3.2. Dynamics

Global descriptions are considered modulo *structural congruence* defined as follows.

Definition 3.3 (*Global Structural Congruence*). The structural congruence $\equiv$ is the least congruence on I such that

a) $(I, \mathbf{0}, \mid)$ and $(I, \mathbf{0}, +)$ are commutative monoids;
b) $I \equiv I'$ if I is α-equivalent to I';
c) $(\boldsymbol{\nu}s)I_1 \mid I_2 \equiv (\boldsymbol{\nu}s)(I_1 \mid I_2)$ for $s \notin \mathsf{fsc}(I_2)$.

The semantics of the global calculus is defined as a reduction relation close to that of imperative languages: it evaluates an interaction with a state resulting into a new state and a new interaction. A *state* σ assigns a value to the variables located at each participant (we assume it is a total function over the set of all possible variables). Formally, a state σ can be seen as a total function that, given a participant, returns a total function mapping variables to values. We will $\sigma@A$ to denote the portion of σ local to A, and $\sigma[y@A \mapsto v]$ to denote a new state σ', which is identical to σ except that $\sigma'@A(y)$ is equal to v. A reduction "$(\sigma,\ I) \rightsquigarrow (\sigma',\ I')$" says that I in the state σ performs one-step computation and becomes I' with the new state σ'.

Definition 3.4 (*Global Reduction Semantics*). The reduction relation $\rightsquigarrow$ is defined as the least relation satisfying the rules reported in Figure 2.

(G-Init) is for session initiation: after A initiates a session with B on service channel ch, A and B share $\tilde{s}$ locally (indicated by $(\boldsymbol{\nu}\tilde{s})$), and the next I is unfolded. The initiation channel ch will play an important role in the typing system and the endpoint projection. (G-Com) is a key rule: the expression e is evaluated into v in the A-portion of the state σ and then assigned to the variable x located at B resulting in the new state $\sigma[x@B \mapsto v]$. The same variable (say x) located at different participants denotes distinct variables (hence $\sigma@A(x)$ and $\sigma@B(x)$ may differ). We do not give a formal definition of $\Downarrow$ as we are not giving a formal syntax for expressions. In general,

$\sigma \vdash e \Downarrow v$ means that the expression obtained by substituting any variable x in e with $\sigma(x)$ evaluates to v. We assume that such operation is always computable. Note that op is not used by (G-COM): it will become meaningful in the end-point calculus where participant A will be able to select one of many branches of execution. Rule (G-ASGN) for assignment has a similar behavior to (G-COM): the expression e is evaluated into v in $\sigma@A$ and then assigned to the variable x located at A resulting in the new state $\sigma[x@A \mapsto v]$. (G-PAR) gives semantics to the parallel composition. Note that there is no synchronization between I_1 and I_2 as only parallel communications can happen. The rule for choice, (G-SUM), expresses the nondeterministic choice of one of two interactions discarding the other. Rules (G-IFT) and (G-IFF) give the semantics to the conditional construct: the expression e is evaluated in the state $\sigma@A$ and then according to its Boolean value a branch is chosen. (G-REC) and (G-RES) are the standard rules for recursion and restriction. (G-STRUCT) makes use of the structural congruence $\equiv$ defined previously.

Example 3.5. As an example of reduction, consider, for instance:

$$\mathsf{Buyer} \to \mathsf{Seller} : \mathit{quoteCh}(\boldsymbol{\nu}\, s).\ \mathsf{Seller} \to \mathsf{Buyer} : s\langle \mathsf{quote},\ 300,\ x\rangle.\ I'$$

with state σ. By (G-INIT), we get $(\sigma,\ (\boldsymbol{\nu} s)\mathsf{Seller} \to \mathsf{Buyer} : s\langle \mathsf{quote},\ 300,\ x\rangle.\ I')$. Now, by rule (G-COM), that evolves into $(\sigma[x@\mathsf{Buyer} \mapsto 300],\ (\boldsymbol{\nu} s) I')$.

The global calculus describes interactions between peers of a system. Because terms in parallel can never synchronize, the semantics given in Figure 2 enjoys the following property.

THEOREM 3.6 (PROGRESS). *For all σ and $I \not\equiv \mathbf{0}$ there exist σ' and I' such that $(\sigma, I) \to (\sigma', I')$.*

PROOF. By induction on the reduction rules in Figure 2 by noting that each rule progresses. In the case of the conditional, we just need to observe that one of the two rules is always applicable. □

3.3. Session Typing for Global Descriptions

As briefly mentioned in Section 1, we use a generalization of session types [Honda et al. 1998] as 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 follows.

Definition 3.7 (*Session Types*).

$$\theta ::= \mathsf{bool} \mid \mathsf{int} \mid \ldots$$
$$\alpha ::= s \blacktriangleright \Sigma_i\, \mathsf{op}_i(\theta_i).\ \alpha_i \mid s \blacktriangleleft \Sigma_i\, \mathsf{op}_i(\theta_i).\ \alpha_i \mid \alpha_1 \mid \alpha_2 \mid \mathtt{end} \mid \mu \mathbf{t}.\ \alpha \mid \mathbf{t}.$$

In the Definition 3.7, θ ranges over *value types* which, in the present case, only include atomic data types. $\alpha, \alpha', \ldots$ are *session types*. $s \blacktriangleright \Sigma_i\, \mathsf{op}_i(\theta_i).\ \alpha_i$ is a *branching input type* at session channel s, indicating a process is ready to receive any of the (pairwise distinct) operators $\{\mathsf{op}_i\}_i$, each with a value of type θ_i; $s \blacktriangleleft \Sigma_i\, \mathsf{op}_i(\theta_i).\ \alpha_i$, a *branching output type* at s, is its dual. The type $\alpha_1 \mid \alpha_2$ is a *parallel composition of α_1 and α_2*, abstracting parallel composition of two session channels. We take $\mid$ to be commutative and associative, with end, the *inaction type* indicating session termination, being the identity. As a syntactic restriction, we demand session channels in α_1 and α_2 to be disjoint: this guarantees a linear use of session channels. $\mathbf{t}$ is a *type variable*, while

$\mu\mathbf{t}.\alpha$ is a *recursive type*, where $\mu\mathbf{t}$ binds free occurrences of $\mathbf{t}$ in α. In recursive types, we assume each recursion is guarded, that is, in $\mu\mathbf{t}.\alpha$, α is an n-ary parallel composition of input/output types. Recursive types are regarded as regular trees in the standard way [Gay and Hole 2005].

Note that session channels occur free in session types: this is necessary to allow multiple session channels to be used in parallel in a single session; with this, we can faithfully capture use cases of Web services that exchange different data simultaneously.

We extend fsc to types, that is, $\mathsf{fsc}(\alpha)$ denotes the set of free session channels in α.

Example 3.8. Let us show a simple example:

$$s \blacktriangleleft \mathsf{quote}(\mathtt{int}).\ \mathtt{end} \mid s' \blacktriangleleft \mathsf{extra}(\mathtt{string}).\ \mathtt{end}.$$

Here a participant is sending a quote (integer) at s and extra information about the product at s' in a single session: without using distinct session channels, two communications can get confused and result in a type error. More examples of the types can be found in Carbone et al. [2006a].

We now introduce a natural notion of duality for session types.

Definition 3.9 (*Duality*). The *cotype*, or *dual*, of α, written $\overline{\alpha}$, is defined as

$$\overline{s \blacktriangleleft \Sigma_i\, \mathsf{op}_i(\theta_i).\ \alpha_i} = s \blacktriangleright \Sigma_i\, \mathsf{op}_i(\theta_i).\ \overline{\alpha}_i \qquad \overline{s \blacktriangleright \Sigma_i\, \mathsf{op}_i(\theta_i).\ \alpha_i} = s \blacktriangleleft \Sigma_i\, \mathsf{op}_i(\theta_i).\ \overline{\alpha}_i$$

$$\overline{\alpha_1 \mid \alpha_2} = \overline{\alpha_1} \mid \overline{\alpha_2} \qquad \overline{\mu t.\ \alpha} = \mu t.\ \overline{\alpha}$$

$$\overline{\mathbf{t}} = \mathbf{t} \qquad \overline{\mathtt{end}} = \mathtt{end}.$$

For example, the cotype of $s \blacktriangleright \mathsf{QuoteReq}(\mathtt{string}).\mathtt{end}$ is $s \blacktriangleleft \mathsf{QuoteReq}(\mathtt{string}).\mathtt{end}$, exchanging input and output. The duality plays an essential role in the subsequent technical development.

We now give the definition of typing judgment.

Definition 3.10 (*Global Typing Judgment*). A *global typing judgment* is a triple $\Gamma \vdash I : \Delta$ inductively defined by the typing rules given in Figure 3, where the mappings Γ and Δ are the *service typing* and the *session typing* respectively. For $A \neq B$ in $\tilde{s}[A, B]$, the grammar of typings is given as:

$$\begin{array}{lllll} \textit{(Service Typing)} & \Gamma & ::= & \emptyset \mid \Gamma, ch@A : (\tilde{s})\alpha & \mid \Gamma, x@A : \theta \mid \Gamma, X : \Delta \\ \textit{(Session Typing)} & \Delta & ::= & \emptyset \mid \Delta \cdot \tilde{s}[A, B] : \alpha & \mid \Delta \cdot \tilde{s} : \bot. \end{array}$$

Each time a session is initiated, session channels need be freshly generated. Thus, the type of a service channel indicates a vector of session channels to be initially exchanged, in addition to how they are used. This is formulated by *service type* $(\tilde{s})\alpha$ where $\tilde{s}$ is a vector of pairwise distinct session channels covering all session channels in α, and α does not contain free type variables. In a service typing, $ch@A : (\tilde{s})\alpha$ says that ch is located at A and offers a service interface $(\tilde{s})\alpha$; $x@A : \theta$ says that a variable x located at A may store values of type θ; finally, $X : \Delta$ says that when the interaction recurs to X, it should have the typing Δ. Session typing uses a primary type assignment $\tilde{s}[A, B] : \alpha$, which says that a vector of session channels $\tilde{s}$, all belonging to the same session between A and B, has the session type α when seen from the viewpoint of A. We assume that A and B are distinct in $\tilde{s}[A, B] : \alpha$. We use the other form of assignment $\tilde{s} : \bot$ whenever we know that the session type at $\tilde{s}$ will never be abstracted by session initiation (this is known for sure when one or more channels in $\tilde{s}$ are hidden by the restriction (νs), see typing rules later).

$$(\text{G-TCOM})\ \frac{\Gamma \vdash I \triangleright \Delta \cdot \tilde{s}\,[A,B]:\alpha_j \quad \Gamma \vdash e@A:\theta_j \quad \Gamma \vdash x@B:\theta_j \quad s \in \{\tilde{s}\} \quad j \in J}{\Gamma \ \vdash\ A \to B : s\langle \mathsf{op}_j,\ e,\ x\rangle.\ I \ \triangleright\ \Delta \cdot \tilde{s}\,[A,B]:s \blacktriangleleft \Sigma_{i\in J}\mathsf{op}_i(\theta_i).\ \alpha_i}$$

$$(\text{G-TCOM}_2)\ \frac{\Gamma \vdash I \triangleright \Delta \cdot \tilde{s}\,[B,A]:\alpha_j \quad \Gamma \vdash e@A:\theta_j \quad \Gamma \vdash x@B:\theta_j \quad s \in \{\tilde{s}\} \quad j \in J}{\Gamma \ \vdash\ A \to B : s\langle \mathsf{op}_j,\ e,\ x\rangle.\ I \ \triangleright\ \Delta \cdot \tilde{s}\,[B,A]:s \blacktriangleright \Sigma_{i\in J}\mathsf{op}_i(\theta_i).\ \alpha_i}$$

$$(\text{G-TASG})\ \frac{\Gamma \vdash x@A:\theta \quad \Gamma \vdash e@A:\theta \quad \Gamma \vdash I \triangleright \Delta}{\Gamma \vdash x := e@A.\ I \triangleright \Delta}$$

$$(\text{G-TPAR})\ \frac{\Gamma \vdash I_1 \triangleright \Delta_1 \quad \Gamma \vdash I_2 \triangleright \Delta_2}{\Gamma \vdash I_1 \mid I_2 \triangleright \Delta_1 \bullet \Delta_2} \qquad (\text{G-TINIT})\ \frac{\Gamma,\ ch@B:(\tilde{s})\alpha \vdash I \triangleright \Delta \cdot \tilde{s}\,[B,A]:\alpha}{\Gamma,\ ch@B:(\tilde{s})\alpha \vdash A \to B : ch(\boldsymbol{\nu}\,\tilde{s}).\ I \triangleright \Delta}$$

$$(\text{G-TSUM})\ \frac{\Gamma \vdash I_1 \triangleright \Delta \quad \Gamma \vdash I_2 \triangleright \Delta}{\Gamma \vdash I_1 + I_2 \triangleright \Delta} \quad (\text{G-TIF})\ \frac{\Gamma \vdash e@A:\mathtt{bool} \quad \Gamma \vdash I_1 \triangleright \Delta \quad \Gamma \vdash I_2 \triangleright \Delta}{\Gamma \vdash \mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2 \triangleright \Delta}$$

$$(\text{G-TRES}_1)\ \frac{\Gamma \vdash I \triangleright \Delta \cdot\ \tilde{s}_1 s \tilde{s}_2[A,B]:\alpha}{\Gamma \vdash (\boldsymbol{\nu} s) I \triangleright \Delta \ \cdot\ \tilde{s}_1\tilde{s}_2:\bot} \quad (\text{G-TRES}_2)\ \frac{\Gamma \vdash I \triangleright \Delta \ \cdot\ \tilde{s}_1 s \tilde{s}_2:\bot}{\Gamma \vdash (\boldsymbol{\nu} s) I \triangleright \Delta \ \cdot\ \tilde{s}_1\tilde{s}_2:\bot}$$

$$(\text{G-TRES}_3)\ \frac{\Gamma \vdash I \triangleright \Delta \ \cdot\ \varepsilon:\bot}{\Gamma \vdash I \triangleright \Delta} \quad (\text{G-TREC})\ \frac{\Gamma, X:\Delta \vdash I \triangleright \Delta}{\Gamma \vdash \mu X.\ I \triangleright \Delta} \quad (\text{G-TVAR})\ \frac{}{\Gamma,\ X:\Delta \vdash X \triangleright \Delta}$$

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

Fig. 3. Typing rules for global calculus.

In the sequel, we write Γ_1, Γ_2 (resp. $\Delta_1 \cdot \Delta_2$) if there is no overlap between the free variables/names in Γ_1 and Γ_2 (resp. Δ_1 and Δ_2). We will consider service and session typings as functions: $\mathsf{dom}(\Gamma)$ and $\mathsf{dom}(\Delta)$ denote the domains of Γ and Δ respectively. For the sake of notation, we often write $\Gamma \vdash I$ for $\Gamma \vdash I \triangleright \emptyset$. The notions of free session channel, free term variable and service channels are extended to both service and session typing.

We now comment the typing rules in Figure 3. Rule (G-TINIT) types session initiation. Given $A \to B : ch(\boldsymbol{\nu}\,\tilde{s}).\ I$, it checks that $ch@B : (\tilde{s})\alpha$ is a service channel in the service typing Γ. Moreover, in the subterm I, session channels $\tilde{s}$ must be used according to session type α by A and B only.

Rule (G-TCOM) states that, for typing an in-session communication of an expression e from A to B at s with choice op_j, (1) the body I should assign α_j to $\tilde{s}$ containing s; (2) the value e should be typed in the source A with θ_j; and (3) the variable (parameter) x should be typed in the target B with the same type. In the conclusion, a branching type is formed whose j-th branch consists of op_j, θ_j and α_j. In (G-TCOM), the session type in focus is considered from the viewpoint of A. We may also regard it from the receiver's viewpoint (B) in its symmetric variant (G-TCOM$_2$). This is necessary since the chosen viewpoint must be preserved when typing subterms. Moreover, such a viewpoint denotes the participant providing the service that is set by rule (G-TINIT). Note that, as we assumed that A and B are distinct in $\tilde{s}[A, B]:\alpha$, practically irrelevant self-calling interactions $A \to A \ldots$ are not typable (see Section 8 for further comments on the need of self-calling interactions).

There are three rules for typing restriction. (G-TRES$_1$) is for hiding a session channel in a session type assignment. Note that, because of Assumption 3.2, hiding never occurs inside a prefix, sum, or conditional and the semantics introduces hiding only after the session initiation takes place. 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 and can be replaced with $\bot$. After this, we can remove hidden

session channels one by one with (G-TRES$_2$), until we get the empty vector, and then completely take it away with (G-TRES$_3$).

(G-TVAR) introduces a service typing on the left-hand side of a judgment. 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), typing $\mu X^A.\ I$, 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 [Honda et al. 1998]. Note that no typing rule treats session types with explicit recursion ($\mu\mathbf{t}.\ \alpha$ and $\mathbf{t}$) because we regard recursive types as regular trees [Honda et al. 1998; Yoshida and Vasconcelos 2007]. (G-TZERO) is the typing rule for the inaction and assigns pairwise disjoint vectors the empty action types.

Given an assignment $x := e@A$, (G-TASG) checks that variable x is located at A in the service environment Γ and its type is the same as the one of the expression e. Finally, the parallel composition rule (G-TPAR) uses the linearity condition found in Honda et al. [1998], that is, any session channel s can only appear at most in one of the two interactions. Formally, the latter is done through the following.

Definition 3.11 ($\simeq$). Two session environments Δ_1 and Δ_2 are compatible (written $\Delta_1 \simeq \Delta_2$) if they satisfy the following conditions:

(1) if $\tilde{s} \in \mathsf{dom}(\Delta_1)$, $\tilde{t} \in \mathsf{dom}(\Delta_2)$ and $\tilde{s} \cap \tilde{t} \neq \emptyset$ then $\tilde{s} = \tilde{t}$;
(2) if $\tilde{s} : \bot \in \Delta_i$ then
$\tilde{s} \notin \mathsf{fsc}(\Delta_j)$ for $j \neq i$;
(3) if $\tilde{s}[A, B] : \alpha_1$ in Δ_1 and $\tilde{s}[A, B] : \alpha_2$ in Δ_2 then $\mathsf{fsc}(\alpha_1) \cap \mathsf{fsc}(\alpha_2) = \emptyset$.

In this definition, (1) requires that session names are equally grouped in both Δ_i; (2) makes sure that $\tilde{s} : \bot$ can only occur exclusively in one of the environments; and, (3) checks the standard notion of linearity. Compatible environments can then be merged by $\bullet$ as follows:

Definition 3.12 ($\bullet$). The partial operation $\Delta_1 \bullet \Delta_2$ is well-defined whenever $\Delta_1 \simeq \Delta_2$ and is the minimal set containing $\Delta_1 \uplus \Delta_2$ such that if $\tilde{s}[A, B] : \alpha_1 \in \Delta_1$ and $\tilde{s}[A, B] : \alpha_2 \in \Delta_2$ then $\tilde{s}[A, B] : \alpha_1 \mid \alpha_2 \in \Delta_1 \bullet \Delta_2$.

Here, the operation $\Delta_1 \uplus \Delta_2$ excludes all those elements in $\mathsf{dom}(\Delta_1) \cap \mathsf{dom}(\Delta_2)$.

Example 3.13. We explain these two definitions with a small example. Consider the following session environments:

$$\Delta_1 = ts[A, B] :\ s \blacktriangleleft (\mathsf{int}) \qquad \Delta_2 = ts[A, B] :\ s \blacktriangleleft (\mathsf{bool}).$$

If we compose them then there are two threads emitting an integer and a Boolean respectively so that an input at s could receive either of them, causing a communication error. However, we could compose Δ_1 with the environment $\Delta_3 = ts[A, B] :\ t \blacktriangleleft (\mathsf{bool})$ obtaining the environment $\Delta_1 \bullet \Delta_3 = ts[A, B] :\ s \blacktriangleleft (\mathsf{int}) \mid t \blacktriangleleft (\mathsf{bool})$.

Example 3.14. As a simple example, we type the Buyer-Seller interaction I in Figure 1(a). Service channel *quoteCh* can be assigned the following service type:

$$(s)\ s \blacktriangleleft \mathsf{quote}(\texttt{integer}).\ s \blacktriangleright (\ \mathsf{accept}(\texttt{null}).\ s \blacktriangleleft \mathsf{details}(\texttt{string}).\ \texttt{end} + \mathsf{reject}(\texttt{null}).\ \texttt{end}\ ).$$

Service channel *deliveryCh* has type $(t)\ t \blacktriangleleft \mathsf{details}(\texttt{string}).\ \texttt{end}$. Denoting these two types by $(s)\alpha_1$ and $(t)\alpha_2$, we have: $quoteCh : (s)\,\alpha_1$, $deliveryCh : (t)\,\alpha_2 \vdash I \triangleright \emptyset$. Similarly, we can type the interaction in Figure 1(b) where we have recursion. The typing of the service channel *quoteCh* will differ in the "rejection" branch, given as: $(s)\ \mu\mathbf{t}.\ s \blacktriangleleft \mathsf{quote}(\texttt{integer}).\ s \blacktriangleright (\ldots + \mathsf{reject}(\texttt{null}).\ \mathbf{t})$.

$$\text{(S-In)}\ \frac{\text{for all } i \in J.\ \alpha_i \Subset \alpha'_i \qquad J \subseteq J'}{s \blacktriangleright \Sigma_{i\in J}\, \mathsf{op}_i(\theta_i).\ \alpha_i \ \Subset\ s \blacktriangleright \Sigma_{i\in J'}\, \mathsf{op}_i(\theta_i).\ \alpha'_i}$$

$$\text{(S-Out)}\ \frac{\text{for all } i \in J.\ \alpha_i \Subset \alpha'_i \qquad J \subseteq J'}{s \blacktriangleleft \Sigma_{i\in J}\, \mathsf{op}_i(\theta_i).\ \alpha_i \ \Subset\ s \blacktriangleleft \Sigma_{i\in J'}\, \mathsf{op}_i(\theta_i).\ \alpha'_i}$$

$$\text{(S-Par)}\ \frac{\beta \approx \alpha'_1|\alpha'_2 \quad \alpha_1 \Subset \alpha'_1 \quad \alpha_2 \Subset \alpha'_2}{\alpha_1|\alpha_2 \ \Subset\ \beta} \qquad \text{(S-End)}\ \frac{\beta \approx \mathtt{end}}{\mathtt{end} \Subset \beta} \qquad \text{(S-Iso)}\ \frac{\alpha' \approx \alpha \quad \alpha \ \Subset\ \beta}{\alpha' \ \Subset\ \beta}$$

Fig. 4. Rules for the subtyping relation $\Subset$.

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

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

Fig. 5. Extension of $\Subset$ to well-formed session and service typings.

3.4. Properties of the Type Discipline

We discuss basic properties of the typing system. Note that Γ, Γ' indicates the disjoint union of Γ and Γ'.

Proposition 3.15.

(*1*) (weakening)
- (*a*) $\Gamma \vdash I \rhd \Delta$ *implies* $\Gamma, \Gamma' \vdash I \rhd \Delta$;
- (*b*) $\Gamma \vdash I \rhd \Delta$ *implies* $\Gamma \vdash I \rhd \Delta \cdot \tilde{s}[A, B] : \mathtt{end}$ *for* $\tilde{s}$ *fresh;*
- (*c*) $\Gamma \vdash I \rhd \Delta \cdot \tilde{s}_1\tilde{s}_2[A, B] : \alpha$ *implies* $\Gamma \vdash I \rhd \Delta \cdot \tilde{s}_1 s\tilde{s}_2[A, B] : \alpha$ *for* $s \notin \mathsf{fsc}(I)$;
- (*d*) $\Gamma \vdash I \rhd \Delta \cdot \tilde{s}_1\tilde{s}_2 :\!\perp$ *implies* $\Gamma \vdash I \rhd \Delta \cdot \tilde{s}_1 s\tilde{s}_2 :\!\perp$ *for* $s \notin \mathsf{fsc}(I)$.

(*2*) (strengthening)
- (*a*) $\Gamma, \Gamma' \vdash I \rhd \Delta$ *implies* $\Gamma \vdash I \rhd \Delta$ *whenever* $(\mathsf{fv}(I) \cup \mathsf{channels}(I)) \cap \Gamma' = \emptyset$;
- (*b*) $\Gamma \vdash I \rhd \Delta \cdot \tilde{s}[A, B] : \mathtt{end}$ *implies* $\Gamma \vdash I \rhd \Delta$ *for* $\tilde{s}$ *fresh;*
- (*c*) $\Gamma \vdash I \rhd \Delta \cdot \tilde{s}_1 s\tilde{s}_2[A, B] : \alpha$ *implies* $\Gamma \vdash I \rhd \Delta \cdot \tilde{s}_1\tilde{s}_2[A, B] : \alpha$ *for* $s \notin \mathsf{fsc}(I)$;
- (*d*) $\Gamma \vdash I \rhd \Delta \cdot \tilde{s}_1 s\tilde{s}_2 :\!\perp$ *implies* $\Gamma \vdash I \rhd \Delta \cdot \tilde{s}_1\tilde{s}_2 :\!\perp$ *for* $s \notin \mathsf{fsc}(I)$.

(*3*) (cotype) $\Gamma \vdash I \rhd \Delta \cdot \tilde{s}[A, B] : \alpha$ *implies* $\Gamma \vdash I \rhd \Delta \cdot \tilde{s}[B, A] : \overline{\alpha}$.

Proof. Standard by induction on the typing rules given in Figure 3. Note that for (co-type) the form $\Delta \cdot \tilde{s}[A, B] : \alpha$ is explicitly generated only in (G-TInit), (G-TCom) and (G-TCom$_2$), which show such inference is possible. □

The typing system also incorporates subtyping based on an inclusion ordering on each type. In the remainder of this section, we show that it is always possible to generate the minimum typing with respect to subtyping, which is the key property for the EPP theory.

In the following definition, $\approx$ is the standard tree isomorphism on recursive types.

Definition 3.16 (*Subtyping Ordering*). The *subtyping ordering* $\Subset$ is the smallest relation over closed types satisfying the rules given in Figure 4. We extend $\Subset$ to well-formed session typings and to well-formed service typings by the rules reported in Figure 5.

Note that the subtyping relation for the global calculus is covariant in both input and output cases since we type interactions rather than input or output operations. Having an input or an output in the type only describes in which direction the communication is taking place, that is, from server to client or vice versa.

Example 3.17. The type from Example 3.14 is a subtype of the following type.

$$(s)\, s \blacktriangleleft \left(\begin{array}{l} \mathsf{quote}(\mathtt{integer}).\, s \blacktriangleright \left(\begin{array}{l} \mathsf{accept}(\mathtt{null}).\, s \blacktriangleleft \mathsf{details}(\mathtt{string}).\, \mathtt{end} + \\ \mathsf{reject}(\mathtt{null}).\, \mathtt{end} + \\ \mathsf{continue}(\mathtt{null}).\, \mathtt{end} \end{array} \right) + \\ \mathsf{update}(\mathtt{integer}).\, \mathtt{end} \end{array} \right)$$

This is because the first output type has an extra branch (namely the update branch) and the first input in the quote branch has the extra branch continue.

A relation $\mathcal{R}$ satisfying the conditions in Definition 3.16 is called a *witness* of $\Subset$. It is easy to see that $\Subset$ itself is a witness of itself, defining the largest such. Note that the subtyping relation is a partial order on types modulo $\approx$. Therefore, we stipulate the following assumption.

Assumption 3.18. We consider $\Subset$ as a relation on types modulo $\approx$.

Let $\alpha \bigtriangledown \beta$ denote the least upper bound of α and β if it exists. The relation $\Subset$ enjoys the following property.

PROPOSITION 3.19. *If α_1 and α_2 have an upperbound, then $\alpha \bigtriangledown \beta$ exists.*

We then have:

PROPOSITION 3.20 (SUBSUMPTION). *Let $\Gamma \Subset \Gamma'$ and $\Delta \Subset \Delta'$. Then $\Gamma \vdash I \rhd \Delta$ implies $\Gamma' \vdash I \rhd \Delta'$.*

PROOF. The rules that use this information, (G-TCOM) and (G-TCOM$_2$), only demand that the operation used by the term is included in the type. Thus, a type including more operations is always safe. □

Finally, we establish the existence of minimal typing. In the sequel, we write $\Gamma \vdash I$ for $\Gamma \vdash I \rhd \emptyset$.

PROPOSITION 3.21 (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. By constructing the minimum typing system. See Section A.3 in Appendix A. □

The proof of Proposition 3.21 immediately gives us the following.

PROPOSITION 3.22.

(*1*) *Given I with annotation on bound channels and variables, there is a decidable algorithm that can check if it is typable and, if it is so, find a minimal typing Γ_0 such that $\Gamma_0 \vdash I$.*
(*2*) *Given I and Γ, there is a decidable algorithm to check $\Gamma \vdash I$ or not.*

PROOF. (1) is by the rules in Figure 12 in Appendix A used for deriving minimal typing. (2) is by (1) and because the subtyping relation is decidable by adapting [Gay and Hole 2005]. □

Since a global interaction is intended to be a specification a programmer will write, and because s/he may as well wish to type check her/his specification at the design time, Proposition 3.22 gives a useful basis for the application of the global description of programs. The minimal typing also plays an important role in endpoint projection, which follows the type structure.

Next we turn to subject reduction. Henceforth, we write $\Gamma \vdash \sigma$ when the typing of σ conforms to Γ.

LEMMA 3.23 (SUBSTITUTION).

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

PROOF. The proof follows from the typing rules. □

The subject reduction in the present typing does *not* fully 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 3.24.

(*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* $(\sigma, I) \rightsquigarrow (\sigma', I')$ *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* $(\sigma, I) \rightsquigarrow (\sigma', I')$ *imply* $\Gamma \vdash \sigma'$ *and* $\Gamma \vdash I'$.

PROOF. In Section A.4 in Appendix A. □

4. THE END-POINT CALCULUS

4.1. Syntax

The end-point calculus is the π-calculus [Milner et al. 1992] extended with sessions [Honda et al. 1998] as well as locations [Hennessy and Riely 1998] and store [Carbone et al. 2004]. In the following, $P, Q, \ldots$ denote *processes*, $M, N, \ldots$ *networks*.

Definition 4.1 (*End-Point Calculus Syntax*).

$P ::=$	$!ch(\tilde{s}).\ P$	(initin)	
$\mid$	$\overline{ch}(\nu\tilde{s}).\ P$	(initout)	
$\mid$	$s \rhd \Sigma_i \mathsf{op}_i(y_i).\ P_i$	(branch)	
$\mid$	$\overline{s} \lhd \mathsf{op}\langle e \rangle.\ P$	(out)	
$\mid$	$x := e.\ P$	(assign)	
$\mid$	if e then P_1 else P_2	(cond)	
$\mid$	$P_1 \oplus P_2$	(plus)	
$\mid$	$P_1 \mid P_2$	(par)	
$\mid$	$(\nu s)P$	(new)	
$\mid$	X	(recvar)	
$\mid$	$\mu X.\ P$	(rec)	
$\mid$	$\mathbf{0}$	(inact)	

$N ::=$	$A[\ P\ ]_\sigma$	(participant)	
$\mid$	$N_1 \mid N_2$	(parnet)	
$\mid$	$(\nu s)N$	(newnet)	
$\mid$	ϵ	(inactnet)	

(initin) and (initout) are dual operations for describing session initiation: $!ch(\tilde{s}).\ P$ denotes a process offering a replicated (available in many copies) service ch with session

channels $\tilde{s}$ while $\overline{ch}(\nu\tilde{s})\mathbf{.}\ P$ denotes a process requesting a service ch with session channels $\tilde{s}$. In both cases, P is the continuation. The next two processes denote standard in-session communications (where y_i in the first construct, the branching input, is *not* bound in P_i, and $\{\mathsf{op}_i\}$ should be pairwise distinct). Next, $x := e\mathbf{.}\ P$ assigns the value of e to x in the store then continues as P.[3] The term (plus) denotes internal choice. The rest is standard. Networks are parallel composition of participants, where a *participant* has the shape $A[\ P\ ]_\sigma$, with A being the name of the participant, P its behavior, and σ its local state, now interpreted as a local function from variables to values. We often omit σ when irrelevant. The free session channels, free term variables and service channels are defined as usual over processes and networks and, similarly to the global calculus, are denoted by $\mathsf{fsc}(P/N)$, $\mathsf{fv}(P/N)$ and $\mathsf{channels}(P/N)$, respectively. For the formal definition see Section B.1 in Appendix B.

4.2. Dynamics

Here, we give the find structural congruence for the end-point calculus.

Definition 4.2 (*End-Point Structural Congruence*). The structural congruence $\equiv$ is defined as the least congruence on processes such that $(\equiv, \mathbf{0}, \oplus)$, $(\equiv, \mathbf{0}, +)$ and $(\equiv, \mathbf{0}, |)$ are commutative monoids and such that it satisfies:

$$\text{a) } (\nu s)\mathbf{0} \equiv \mathbf{0} \qquad \text{b) } (\nu s_1)(\nu s_2)P \equiv (\nu s_2)(\nu s_1)P$$

$$\text{c) } (\nu s)P \mid Q \equiv (\nu s)(P \mid Q) \qquad (\text{for } s \notin \mathsf{fsc}(Q)).$$

We extend $\equiv$ to networks such that $(|, \epsilon)$ is a commutative monoid and

$$\text{d) } A[\ P\ ]_\sigma \equiv A[\ Q\ ]_\sigma \qquad (\text{for } P \equiv Q)$$

$$\text{e) } A[\ (\nu s)P\ ]_\sigma \equiv (\nu s)(A[\ P\ ]_\sigma) \qquad \text{f) } (\nu s_1)(\nu s_2)M \equiv (\nu s_2)(\nu s_1)M$$

$$\text{g) } (\nu s)\epsilon \equiv \epsilon \qquad \text{h) } (\nu s)M \mid N \equiv (\nu s)(M \mid N) \qquad (\text{for } s \notin \mathsf{fsc}(N)).$$

Similarly to the global calculus, we are now able to define the reduction semantics for the end-point calculus that follows the one of the π-calculus.

Definition 4.3 (*End-Point Reduction Semantics*). The reduction semantics for the end-point calculus is defined by the rules in Figure 6.

(E-INIT) defines the session initiation: two participants A and B will synchronize to start a session, $!ch(\tilde{s})\mathbf{.}\ P$ denoting a replicated service and $\overline{ch}(\nu\tilde{s})\mathbf{.}\ Q$ a request. It will result in sharing fresh session channels $\tilde{s}$ local to A and B. These names are then used in (E-COM) for communication. In (E-COM), communicated values are assigned to local variables in the local state σ, rather than substituted, for having the correspondence with the global calculus. (E-ASSIGN) updates the local store. The other rules are standard.

4.3. Session Typing for the End-Point Calculus

The syntax of types for the end-point calculus is the one given for the global calculus in Definition 3.7. Unlike in the global case, we have two different typing judgments, one used for typing processes and one for typing networks.

[3]For simplicity of the end-point projection, we add assignment to the calculus. It is well-known that we can easily encode the primitive into the π-calculus.

$$\text{(E-Init)}\quad \frac{s_i \notin \mathsf{fsc}(P') \cup \mathsf{fsc}(Q')}{A[\,!ch(\tilde{s}).\,P \mid P'\,]_\sigma \mid B[\,\overline{ch}(\nu\tilde{s}).\,Q \mid Q'\,]_{\sigma'} \rightsquigarrow\ (\nu\tilde{s})(A[\,!ch(\tilde{s}).\,P \mid P \mid P'\,]_\sigma \mid B[\,Q \mid Q'\,]_{\sigma'})}$$

$$\text{(E-Com)}\quad \frac{\sigma \vdash e \Downarrow v \qquad j \in I}{A[\,s \rhd \Sigma_{i\in I}\,\mathsf{op}_i(x_i).\,P_i \mid P'\,]_\sigma \mid B[\,\overline{s} \lhd \mathsf{op}_j\langle e\rangle.\,Q \mid Q'\,]_{\sigma'} \rightsquigarrow\ A[\,P_j \mid P'\,]_{\sigma[x_j \mapsto v]} \mid B[\,Q \mid Q'\,]_{\sigma'}}$$

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

$$\text{(E-IfF)}\frac{\sigma \vdash e \Downarrow \mathsf{ff}}{A[\,\mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \mid P'\,]_\sigma \rightsquigarrow\ A[\,P_2 \mid P'\,]_\sigma} \qquad \text{(E-Sum)}\frac{i \in \{1,2\}}{A[P_1 \oplus P_2|R]_\sigma \rightsquigarrow\ A[P_i|R]_\sigma}$$

$$\text{(E-Rec)}\frac{A[\,P[\mu X.P/X] \mid Q\,]_\sigma \mid N \rightsquigarrow N'}{A[\,\mu X.P \mid Q\,]_\sigma \mid N \rightsquigarrow N'} \qquad \text{(E-Assign)}\ \frac{\sigma \vdash e \Downarrow v}{A[\,x := e.\,P \mid P'\,]_\sigma \rightsquigarrow\ A[\,P \mid P'\,]_{\sigma[x \mapsto v]}}$$

$$\text{(E-Res)}\ \frac{M \rightsquigarrow M'}{(\nu s)M \rightsquigarrow (\nu s)M'} \qquad \text{(E-Struct)}\frac{M \equiv M' \qquad M' \rightsquigarrow N' \qquad N' \equiv N}{M \rightsquigarrow N}$$

Fig. 6. Reduction relation for the end-point calculus.

Definition 4.4 (*End-Point Typing Judgments*). *Typing judgments* for the end-point calculus are triples on processes or networks inductively defined by the rules given in Figure 7 and 8. Judgments have the form:

(Processes) $\Gamma \vdash_A P \rhd \Delta$ (where P is typed as a behavior for A)

(Networks) $\Gamma \vdash N \rhd \Delta$.

where the mappings Γ and Δ are the (end-point) *service* and *session typing*, respectively, and are defined as:

$$\begin{array}{rcl} \Gamma & ::= & \emptyset \mid \Gamma, ch@A : (\tilde{s})\alpha \mid \Gamma, \overline{ch}@A : (\tilde{s})\alpha \mid \Gamma, x@A : \theta \mid \Gamma, X : \Delta \\ \Delta & ::= & \emptyset \mid \Delta \cdot \tilde{s}@A : \alpha \mid \Delta \cdot \tilde{s} : \bot. \end{array}$$

Note that service typings and session typings differ from the ones for the global calculus shown in Section 3.3. In the sequel, $\Gamma(ch) = (\tilde{s})\alpha$ if either $ch@A : (\tilde{s})\alpha$ or $\overline{ch}@A : (\tilde{s})\alpha$ belongs to Γ. As before, we stipulate that both service and session typings define appropriate functions. In particular, whenever we write, for instance, Γ_1, Γ_2, there are *no* free channels/session channels/variables shared between two typings (free session channel, free term variable and service channels are similar to the global case). Also, we often write $\Gamma \vdash N$ for $\Gamma \vdash N \rhd \emptyset$ and $\Gamma \vdash_A P$ for $\Gamma \vdash_A P \rhd \emptyset$. The notions of free session channel, free term variable and service channel extend to typings accordingly.

We observe the following facts.

(1) One basic difference from the global case 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 behavior, 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

$$\text{(E-TB)}\ \frac{j \in J \quad K \subseteq J \quad s \in \tilde{s} \quad \Gamma \vdash x_j : \theta_j \quad \Gamma \vdash_A P_j \triangleright \Delta \cdot \tilde{s}@A : \alpha_j}{\Gamma \vdash_A s \triangleright \Sigma_{i \in J} \mathsf{op}_i(x_i).P_i \triangleright \Delta \cdot \tilde{s}@A : s \blacktriangleright \Sigma_{i \in K} \mathsf{op}_i(\theta_i).\ \alpha_i}$$

$$\text{(E-TS)}\ \frac{j \in K \quad \Gamma \vdash e : \theta_j \quad \Gamma \vdash_A P \triangleright \Delta \cdot \tilde{s}@A : \alpha_j}{\Gamma \vdash_A \overline{s} \triangleleft \mathsf{op}_j\langle e \rangle.P \triangleright \Delta \cdot \tilde{s}@A : s \blacktriangleleft \Sigma_{i \in K} \mathsf{op}_i(\theta_i).\ \alpha_i}$$

$$\text{(E-TSERV)}\ \frac{\Gamma \vdash_A P \triangleright \tilde{s}@A : \alpha}{\Gamma, ch@A : (\tilde{s})\alpha \vdash_A \,!ch(\tilde{s}).\ P \triangleright \emptyset} \qquad \text{(E-TREQ)}\ \frac{\Gamma, \overline{ch}@B : (\tilde{s})\alpha \vdash_A P \triangleright \Delta \cdot \tilde{s}@A : \alpha}{\Gamma, \overline{ch}@B : (\tilde{s})\alpha \vdash_A \overline{ch}(\boldsymbol{\nu}\tilde{s}).P \triangleright \Delta}$$

$$\text{(E-TRES}_1\text{)}\ \frac{\Gamma \vdash_A P \triangleright \Delta \cdot \tilde{s}_1 s \tilde{s}_2 : \bot}{\Gamma \vdash_A (\boldsymbol{\nu} s)P \triangleright \Delta \cdot \tilde{s}_1 \tilde{s}_2 : \bot} \qquad \text{(E-TRES}_2\text{)}\ \frac{\Gamma \vdash_A P \triangleright \Delta \cdot \varepsilon : \bot}{\Gamma \vdash_A P \triangleright \Delta}$$

$$\text{(E-TPAR)}\ \frac{\Gamma_i \vdash_A P_i \rhd \Delta_i \quad \Gamma_1 \asymp \Gamma_2 \quad \Delta_1 \asymp \Delta_2}{\Gamma_1 \odot \Gamma_2 \vdash_A P_1 \mid P_2 \rhd \Delta_1 \odot \Delta_2} \qquad \text{(E-TINACT)}\ \frac{}{\Gamma \vdash_A \mathbf{0} \rhd \emptyset}$$

$$\text{(E-TASG)}\ \frac{\Gamma \vdash_A x : \theta \quad \Gamma \vdash e : \theta \quad \Gamma \vdash_A P \rhd \Delta}{\Gamma \vdash_A x := e.\ P \rhd \Delta} \qquad \text{(E-TREC)}\ \frac{\Gamma, X : \Delta \vdash_A P \triangleright \Delta}{\Gamma \vdash_A \mu X.P \triangleright \Delta}$$

$$\text{(E-TIF)}\ \frac{\Gamma \vdash e : \mathtt{bool} \quad \Gamma \vdash_A P_i \rhd \Delta}{\Gamma \vdash_A \mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \rhd \Delta} \qquad \text{(E-TVAR)}\ \frac{}{\Gamma, X : \Delta \vdash_A X \rhd \Delta}$$

$$\text{(E-TBOT)}\ \frac{\Gamma \vdash_A P \rhd \Delta \quad \{\tilde{s}\} \cap \mathsf{fsc}(\Delta) = \emptyset}{\Gamma \vdash_A P \rhd \Delta \cdot \tilde{s} : \bot} \qquad \text{(E-TSUM)}\ \frac{\Gamma \vdash_A P \rhd \Delta \quad \Gamma \vdash_A Q \rhd \Delta}{\Gamma \vdash_A P \oplus Q \rhd \Delta}$$

$$\text{(E-TEND)}\ \frac{\Gamma \vdash_A P \rhd \Delta \quad \{\tilde{s}\} \cap \mathsf{fsc}(\Delta) = \emptyset}{\Gamma \vdash_A P \rhd \Delta \cdot \tilde{s}@A : \mathtt{end}}$$

Fig. 7. Session types for processes in the end-point calculus.

effectively makes $\tilde{s}$ unusable in further composition. This is the standard linear typing in the π-calculus.

(3) In the service typing, $ch@A : (\tilde{s})\alpha$ is identical to the one in the global calculus and it is called *server type assignment*. On the other hand, $\overline{ch} : (\tilde{s})\alpha@A$ is the *client type assignment*. Note that A is the participant offering service ch rather than the participant invoking such a service. As we will stipulate, the composition of $ch@A : (\tilde{s})\alpha$ and $\overline{ch}@A : (\tilde{s})\overline{\alpha}$ becomes $ch@A : (\tilde{s})\alpha$, since a service may be used not only one but many times (standard replicated linear type discipline [Berger et al. 2001; Yoshida et al. 2004]).

We now comment the typing rules defining $\vdash$ and $\vdash_A$ given in Figures 7 and 8.

(E-TB), used for typing the branching input $s \triangleright \Sigma_{i \in J} \mathsf{op}_i(x_i).P_i$, prefixes the type α_i in P_i with $\mathsf{op}_i(\theta_i)$ only for those branches $j \in K \subseteq J$ so that the process is prepared to receive any operator specified in the type. (E-TS), used for typing $\overline{s} \triangleleft \mathsf{op}_j\langle e \rangle.P$, is its dual: the typing can have more branches than the real process, so that the process invokes at most those operators specified in the typing. Combining (E-TB) and (E-TS), an output never invokes a nonexistent option in the input. (E-TSERV) is for the receiving side of initialization. In the premise, the session typing should not have session channels other than the target of initialization: this prevents *free* session channels from occurring under the replicated input, thus guaranteeing their linear usage. Additionally,

$$(\text{E-TPART})\ \frac{\Gamma \vdash_A P \triangleright \Delta \quad \Gamma \vdash \sigma@A}{\Gamma \vdash A[\,P\,]_\sigma \triangleright \Delta} \qquad (\text{E-TINACTN})\ \frac{}{\Gamma \vdash \epsilon \triangleright \emptyset}$$

$$(\text{E-TENDN})\ \frac{\Gamma \vdash M \triangleright \Delta \quad \{\tilde{s}\} \cap \mathsf{fsc}(\Delta) = \emptyset}{\Gamma \vdash M \triangleright \Delta \cdot \tilde{s}@A : \mathsf{end}} \qquad (\text{E-TRESN}_1)\ \frac{\Gamma \vdash M \triangleright \Delta \cdot \tilde{s}_1 s \tilde{s}_2 : \bot}{\Gamma \vdash (\nu s)M \triangleright \Delta \cdot \tilde{s}_1 \tilde{s}_2 : \bot}$$

$$(\text{E-TBOTN})\ \frac{\Gamma \vdash M \triangleright \Delta \quad \{\tilde{s}\} \cap \mathsf{fsc}(\Delta) = \emptyset}{\Gamma \vdash M \triangleright \Delta \cdot \tilde{s} : \bot} \qquad (\text{E-TRESN}_2)\ \frac{\Gamma \vdash M \triangleright \Delta \cdot \varepsilon : \bot}{\Gamma \vdash M \triangleright \Delta}$$

$$(\text{E-TPARN})\ \frac{\Gamma_i \vdash N_i \triangleright \Delta_i \quad \Gamma_1 \asymp \Gamma_2 \quad \Delta_1 \asymp \Delta_2}{\Gamma_1 \odot \Gamma_2 \vdash N_1 \mid N_2 \triangleright \Delta_1 \odot \Delta_2}$$

Fig. 8. Session types for networks in the end-point calculus.

the rule guarantees uniqueness of services in the subterm P by restricting the services environment as in Sangiorgi [1999]. This is essential for modeling Web services with respect to the Service Channel Principle as we can view the service channel as, for instance, an IP address or URI.

The output side of initialization (E-TREQ) is analogous, except it does not need the linearity constraint. The remaining rules are standard [Honda et al. 1998]. Note that, in (E-TPAR) and (E-TPARN), we ensure that an input of type α is composed with an output of its dual and that a service channel can occur as (initin) only once but several times as (initout). This is done by operators $\asymp$ and $\odot$.

Definition 4.5 ($\asymp$).

(1) Two service typings Γ_1 and Γ_2 are compatible (written $\Gamma_1 \asymp \Gamma_2$) if they satisfy the following conditions:
   (a) if $ch@A \in \mathsf{dom}(\Gamma_i)$ then $ch@B \notin \mathsf{dom}(\Gamma_j)$ for every B and for $i \neq j$;
   (b) if $\overline{ch}@A \in \mathsf{dom}(\Gamma_i)$ and $\overline{ch}@B \in \mathsf{dom}(\Gamma_j)$ then $A = B$ and $\Gamma_i(\overline{ch}@A) = \Gamma_j(\overline{ch}@B)$ for $i \neq j$ (up to α-renaming of bound names);
   (c) if $ch@A \in \mathsf{dom}(\Gamma_i)$ and $\overline{ch}@B \in \mathsf{dom}(\Gamma_j)$ then $A = B$ and $\Gamma_i(ch@A) = \overline{\Gamma_j(\overline{ch}@B)}$ for $i \neq j$ (up to α-renaming of bound names);
   (d) $\Gamma_1(x) = \Gamma_2(x)$ for each x in Γ_1 and Γ_2;
   (e) $\Gamma_1(X) = \Gamma_2(X)$ for each X in Γ_1 and Γ_2.
(2) Two session typings Δ_1 and Δ_2 are compatible (written $\Delta_1 \asymp \Delta_2$) if they satisfy the following conditions:
   (a) if $\tilde{s} \in \mathsf{dom}(\Delta_1)$, $\tilde{t} \in \mathsf{dom}(\Delta_2)$ and $\tilde{s} \cap \tilde{t} \neq \emptyset$ then $\tilde{s} = \tilde{t}$;
   (b) if $\tilde{s}:\bot \in \Delta_i$ then $\tilde{s} \notin \mathsf{dom}(\Delta_j)$ for $i \neq j$;
   (c) if $\tilde{s}@A : \alpha_1$ in Δ_1 and $\tilde{s}@A : \alpha_2$ in Δ_2 then $\mathsf{fsc}(\alpha_1) \cap \mathsf{fsc}(\alpha_2) = \emptyset$;
   (d) if $\tilde{s}@A : \alpha_1$ in Δ_1 and $\tilde{s}@B : \alpha_2$ in Δ_2 then $\alpha_1 = \overline{\alpha_2}$ (for $A \neq B$).

Definition 4.6 ($\odot$).

(1) $\Gamma_1 \odot \Gamma_2$, defined whenever $\Gamma_1 \asymp \Gamma_2$, is the minimum service typing such that:
   (a) if $ch@A : \alpha \in \Gamma_i$ then $ch@A : \alpha \in \Gamma_1 \odot \Gamma_2$;
   (b) if $\overline{ch}@A : \alpha \in \Gamma_i$ and $ch@A : \alpha \notin \Gamma_j$ for $i \neq j$ then $\overline{ch}@A : \alpha \in \Gamma_1 \odot \Gamma_2$;
   (c) if $x@A : \theta \in \Gamma_i$ ($X : \Delta \in \Gamma_i$) then $x@A : \theta \in \Gamma_1 \odot \Gamma_2$ ($X : \Delta \in \Gamma_1 \odot \Gamma_2$).
(2) $\Delta_1 \odot \Delta_2$, defined whenever $\Delta_1 \asymp \Delta_2$, is the minimum session typing such that:
   (a) if $\tilde{s}@A \in \mathsf{dom}(\Delta_i) \backslash \mathsf{dom}(\Delta_j)$ for $i \neq j$ then $\tilde{s}@A : \Delta_i(\tilde{s}@A) \in \Delta_1 \odot \Delta_2$;
   (b) if $\tilde{s} \in \mathsf{dom}(\Delta_i) \backslash \mathsf{dom}(\Delta_j)$ for $i \neq j$ then $\tilde{s} : \bot \in \Delta_1 \odot \Delta_2$;

(c) if $\tilde{s}@A : \alpha \in \Delta_i$ and $\tilde{s}@A : \beta \in \Delta_j$ for $i \neq j$ then $\tilde{s}@A : \alpha \mid \beta \in \Delta_1 \odot \Delta_2$;
(d) if $\tilde{s}@A \in \mathsf{dom}(\Delta_i)$ and $\tilde{s}@B \in \mathsf{dom}(\Delta_j)$ for $i \neq j$ then $\tilde{s} : \bot \in \Delta_1 \odot \Delta_2$.

Example 4.7. We provide an example explaining how $\asymp$ and $\odot$ work in the case of the session environment. Consider the following typing environments.

$$\Delta_1 = rt@B : r \blacktriangleleft (\mathsf{char}) \cdot s@A : s \blacktriangleleft (\mathsf{int})$$
$$\Delta_2 = rt@B : t \blacktriangleleft (\mathsf{bool}) \cdot s@C : s \blacktriangleright (\mathsf{int})$$

Clearly, $\Delta_1 \asymp \Delta_2$ and

$$\Delta_1 \odot \Delta_2 = rt@B : (r \blacktriangleleft (\mathsf{char}) \mid t \blacktriangleleft (\mathsf{bool})) \quad \cdot \quad s@A : \bot .$$

However, we have that $\Delta_1 \not\asymp \Delta_3$ for

$$\Delta_3 = rt@B : r \blacktriangleleft (\mathsf{bool}) \cdot s@C : s \blacktriangleright (\mathsf{int}).$$

since in session rt we cannot merge the session type $r \blacktriangleleft (\mathsf{bool})$ with $r \blacktriangleleft (\mathsf{char})$: if we use parallel we break linearity of channel r; alternatively, we cannot replace them with $\bot$ since the two types are not dual.

4.4. Properties of the Type Discipline

As well as in the global case, the type discipline for the end-point calculus enjoys standard syntactic properties.

PROPOSITION 4.8.

(*1*) (weakening)
  (*a*) $\Gamma \vdash M \rhd \Delta$ *implies* $\Gamma, \Gamma' \vdash M \rhd \Delta$;
  (*b*) $\Gamma \vdash M \rhd \Delta \cdot \tilde{s}_1\tilde{s}_2@A : \alpha$ *implies* $\Gamma \vdash M \rhd \Delta \cdot \tilde{s}_1 s \tilde{s}_2@A : \alpha$ *for* $s \notin \mathsf{fsc}(M)$;
  (*c*) $\Gamma \vdash M \rhd \Delta \cdot \tilde{s}_1\tilde{s}_2@A : \bot$ *implies* $\Gamma \vdash M \rhd \Delta \cdot \tilde{s}_1 s \tilde{s}_2@A : \bot$ *for* $s \notin \mathsf{fsc}(M)$.
(*2*) (strengthening)
  (*a*) $\Gamma, \Gamma' \vdash M \rhd \Delta$ *implies* $\Gamma \vdash M \rhd \Delta$ *whenever* $(\mathsf{fv}(M) \cup \mathsf{channels}(M)) \cap \Gamma' = \emptyset$;
  (*b*) $\Gamma \vdash M \rhd \Delta \cdot \tilde{s}@A : \mathsf{end}$ *implies* $\Gamma \vdash M \rhd \Delta$ *for* $\tilde{s}$ *fresh*;
  (*c*) $\Gamma \vdash M \rhd \Delta \cdot \tilde{s}_1 s \tilde{s}_2@A : \alpha$ *implies* $\Gamma \vdash M \rhd \Delta \cdot \tilde{s}_1\tilde{s}_2@A : \alpha$ *for* $s \notin \mathsf{fsc}(M)$;
  (*d*) $\Gamma \vdash M \rhd \Delta \cdot \tilde{s}_1 s \tilde{s}_2@A : \bot$ *implies* $\Gamma \vdash M \rhd \Delta \cdot \tilde{s}_1\tilde{s}_2@A : \bot$ *for* $s \notin \mathsf{fsc}(M)$.

PROOF. Standard, by induction on the typing rules. □

We consider a subtyping relation on session types following [Gay and Hole 2005].[4] The subtyping is defined only over closed types, that is, those types in which no free type variables occur. The subtyping is written $\alpha \preceq \beta$. Intuitively, $\alpha_1 \preceq \alpha_2$ indicates that α_1 is more constrained, or dually α_2 is less constrained, in behavior.

Definition 4.9 (*Subtyping Relation*). The subtyping relation $\preceq$ is the largest binary relation satisfying the rules in Figure 9.

Rule (ES-IN) says that if the initial input offers more options, and if subsequent behaviors are more constrained, then it is more constrained. (ES-OUT) says that if the initial output has less emissions and if subsequent behaviors are more constrained then it is more constrained. Note that, unlike the inclusion ordering $\Subset$ in Section 3.3, the input type is covariant for this relation. It is easy to prove that the relation $\preceq$ taken modulo $\approx$ is also a partial order.

Definition 4.10. We write $\alpha \vee \beta$ for the lub of α and β if it exists.

[4]The direction of the subtyping is converse to (and consistent with) Gay and Hole [2005] The direction we adopted coincides with the subtyping of the λ-calculus as recently studied in Demangeon and Honda [2011].

$$(\text{ES-In})\ \frac{\text{for all } i \in J'.\ \alpha_i \preceq \beta_i \qquad J \supseteq J'}{s \blacktriangleright \Sigma_{i\in J} op_i(\theta_i).\ \alpha_i \ \preceq\ s \blacktriangleright \Sigma_{i\in J'} op_i(\theta_i).\ \beta_i}$$

$$(\text{ES-Out})\ \frac{\text{for all } i \in J.\ \alpha_i \preceq \beta_i \qquad J \subseteq J'}{s \blacktriangleleft \Sigma_{i\in J} op_i(\theta_i).\ \alpha_i \ \preceq\ s \blacktriangleleft \Sigma_{i\in J'} op_i(\theta_i).\ \beta_i}$$

$$(\text{ES-Par})\ \frac{\beta \approx \beta_1 \mid \beta_2 \quad \alpha_1 \ \preceq\ \beta_1 \quad \alpha_2 \ \preceq\ \beta_2}{\alpha_1 \mid \alpha_2 \ \preceq\ \beta}$$

$$(\text{ES-End})\ \frac{\beta \approx \mathtt{end}}{\mathtt{end} \ \preceq\ \beta} \qquad\qquad (\text{ES-Iso})\ \frac{\alpha \approx \alpha' \quad \alpha \ \preceq\ \beta}{\alpha' \ \preceq\ \beta}$$

Fig. 9. Inductive rules defining the subtyping relation.

PROPOSITION 4.11. *If the upper bound of $\alpha_{1,2}$ exists (w.r.t. $\preceq$) then their least upper bound exists.*

PROPOSITION 4.12 (SUBSUMPTION). *Let $\alpha \preceq \beta$. Then,*

(1) $\Gamma, \overline{ch}@A : (\tilde{s})\alpha \vdash M \rhd \Delta$ *implies* $\Gamma, \overline{ch}@A : (\tilde{s})\beta \vdash M \rhd \Delta$*;*
(2) $\Gamma \vdash M \rhd \Delta \cdot \tilde{s}@A : \alpha$ *implies* $\Gamma \vdash M \rhd \Delta \cdot \tilde{s}@A : \beta$*;*
(3) $\Gamma, ch@A : (\tilde{s})\alpha \vdash M \rhd \Delta$ *implies* $\Gamma, ch@A : (\tilde{s})\beta \vdash M \rhd \Delta$.

PROOF. In the first two cases the proof follows by induction on the typing rules. The last case follows by case (2). □

Note that in Proposition 4.12, point (1) is covariant because of the direction we chose for $\preceq$.

Similarly to the global case, we now show that we can always find a representative typing for a given process. 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. In the sequel, we say that a typing $\Gamma \vdash M \rhd \Delta$ is *strict*, if all free identifiers in Γ and Δ occur in M. Moreover, we write $\Gamma_0 \preceq \Gamma$ and $\Delta_0 \preceq \Delta$ by extending $\preceq$ point-wise at their service/session channels (for variables typing should coincide).

Definition 4.13 (*Minimal Typing*). $\Gamma_0 \vdash M \rhd \Delta_0$ is the *minimal typing of M* if, whenever $\Gamma \vdash M \rhd \Delta$ is strict, we have $\Gamma_0 \prec \Gamma$ and $\Delta_0 \prec \Delta$.

PROPOSITION 4.14 (EXISTENCE OF MINIMAL TYPING). *Let $\Gamma_0 \vdash M \rhd \Delta_0$ be the minimal typing of M. Then Γ_0 and Δ_0 are algorithmically calculable from M.*

PROOF. See Section B.2 in Appendix B. □

PROPOSITION 4.15. *In the following, we assume M is annotated on bound channels and variables in the standard way.*

(1) Given M, there is a decidable algorithm that can check if it is typable and find Γ_0 and Δ_0 such that $\Gamma_0 \vdash M \rhd \Delta_0$ is a minimal typing of M.
(2) Given M, Γ and Δ, there is a decidable algorithm to check $\Gamma \vdash I$ or not.

PROOF. (1) is by the rules in Figures 13 and 14 in Appendix B.2, where they are proved to derive the minimal typing. (2) is from (1) and by adapting Gay and Hole [2005]. □

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

LEMMA 4.16 (SUBSTITUTION).

(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. By induction on the typing rules. □

THEOREM 4.17.

(1) *(Subject Congruence) If* $\Gamma \vdash M \rhd \Delta$ *and* $M \equiv N$ *then* $\Gamma \vdash N \rhd \Delta$;
(2) *(Subject Reduction) If* $\Gamma \vdash N \rhd \Delta$ *and* $N \to N'$ *then* $\Gamma \vdash N' \rhd \Delta$.

PROOF. The proofs of subject congruence and subject reduction for the end-point calculus are standard [Honda et al. 1998; Yoshida and Vasconcelos 2007]. □

Note that the session environment Δ remains unchanged after reduction (unlike in Theorem 3.24). In fact, in the end-point case, if a session can reduce will have type $\bot$ in the session environment before and after reduction. This is due to the absence of communication error in well-typed terms as presented in the following.

Definition 4.18 (*Communication Error*). We say M has a *communication error* if $M \equiv (\nu \tilde{s})(N \mid A[s \rhd \Sigma_i \mathsf{op}_i(x_i).\ P_i | R]_\sigma \mid B[\overline{s} \lhd \mathsf{op}\langle e \rangle.\ Q | S]_{\sigma'})$ s.t. $\mathsf{op} \notin \{\mathsf{op}_i\}$.

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 4.17 follows.

COROLLARY 4.19 (LACK OF COMMUNICATION ERROR). *If* $\Gamma \vdash N \rhd \Delta$ *and* $N \rightsquigarrow^* M$, *then* M *never contains a communication error.*

PROOF. By Theorem 4.17 noting an incompatible redex is not typable. □

Thus once a process/network is well-typed, it never go into a communication mismatch.

4.5. Examples of Typed Terms

We recall our running example, Figure 1(a). An end-point representation of this example for Buyer may be written:

$$\mathsf{Buyer}[\ \overline{quoteCh}(\nu s).\ s \rhd \mathsf{quote}(x).\ (\overline{s} \lhd \mathsf{accept}.\ s \rhd \mathsf{details}(y).\ \mathbf{0} \oplus \overline{s} \lhd \mathsf{reject}.\ \mathbf{0})\ ].$$

Here Buyer[...] indicates a participant (a named agent) whose behavior is given by the process $\overline{quoteCh}(\nu s).\ s \rhd \mathsf{quote}(x).\ \ldots$ The Seller's code is given as:

$$\mathsf{Seller}[!\, quoteCh(s).\ \overline{s} \lhd \mathsf{quote}\langle 300 \rangle.\ \ s \rhd \\ (\mathsf{accept}.\ \overline{deliveryCh}(\nu t).\ t \rhd \mathsf{delivery}(x).\ \overline{s} \lhd \mathsf{delivery}\langle x \rangle.\ \mathbf{0}\ +\ \mathsf{reject}.\ \mathbf{0})],$$

The end-point representation for Shipper is given similarly. These end-point descriptions do not directly and explicitly describe how interaction proceeds globally, which may often be the central concern of communication-centered applications designers/users. However, they precisely represent local communication behaviors that give rise to global interactions. The two service channels *quoteCh* and *deliveryCh* are replicated and ready to be invoked, following (SCP).

We can type these processes using the service types $(s)\alpha_1$ and $(t)\alpha_2$ from Section 3.3. The type of the seller becomes (writing P for its process):

$$quoteCh : (s)\,\alpha_1,\ \overline{deliveryCh} : (t)\,\overline{\alpha}_2 \vdash \mathsf{Seller}[\ P\ ]_\sigma \rhd \emptyset.$$

Note that the service channel *deliveryCh* is overlined, indicating the direction: this is because the input channel is located at Shipper's. In the global calculus, a channel is

always used for both input and output, so there is no such need. Similarly we may type the end-point processes for Buyer and Seller with recursion as in Figure 1(b), as follows.

$$\begin{aligned}&\mathsf{Buyer}[\ \overline{quoteCh}(\nu s).\ \mu X.\ s \triangleright \mathsf{quote}(x).\\ &\qquad \mathsf{if\ reasonable}(x)\ \mathsf{then}\ \overline{s} \triangleleft \mathsf{accept}.\ s \triangleright \mathsf{details}(y).\ \mathbf{0}\ \mathsf{else}\ \overline{s} \triangleleft \mathsf{reject}.\ X\ ]\quad |\\ &\mathsf{Seller}[\ !quoteCh(s).\ \mu X.\ \overline{s} \triangleleft \mathsf{quote}\langle 300\rangle.\ \ s \triangleright\\ &\qquad (\mathsf{accept}.\ \overline{deliveryCh}(\nu t).\ t \triangleright \mathsf{delivery}(x).\ \overline{s} \triangleleft \mathsf{delivery}\langle x\rangle.\ \mathbf{0} + \mathsf{reject}.\ X)]\end{aligned}$$

We may also note, both in its term and in its typing, that the end-point process for Shipper in Figure 1(b) does not involve recursion, since its session is self-contained inside a recursion.

5. THE END-POINT PROJECTION

5.1. Three Principles for End-Point Projections

In preceding sections, we have presented example specifications both as a global and a local view in the global and the end-point calculus respectively. 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 method for designing and coding communication-centric programs. It is often too complicated to design, implement and validate an application that involves complex interactions among processes and which work together correctly, if we are to solely rely on descriptions of local behaviors. This is why such tools as message sequence charts and sequence diagrams have been used as a primary way to design communication behavior. Thus, in designing and implementing communication-centric software, one may as well start from a global description of expected behavior, then translate it into local descriptions. 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 that does not correspond to any reasonable local counterpart. How this can be done generally and uniformly with a formal foundation is the theme of this section.

In the context of the core calculi introduced in the previous sections, we have identified three simple descriptive principles.

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

These three conditions offer not only natural disciplines for well-structured description, but also gradually deeper analysis of operational aspects of choreography. 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 structures. Coherence stipulates a condition under which these threads can be formed and combined to produce a whole behavior of each participant. The resulting participants can realize, when combined together, all and only interactions prescribed in the original global description. Thus, by a precise analysis of 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 that maps them to the corresponding local descriptions.

5.2. Connectedness

Let us consider the following code snippet for global description.

$$\mathsf{Buyer} \to \mathsf{Seller} : ch_1(\boldsymbol{\nu}\, s).\ \ \mathsf{Shipper} \to \mathsf{Depot} : ch_2(\boldsymbol{\nu}\, t).\ \mathbf{0}.$$

Remembering that "." indicates sequencing, Shipper is supposed to contact Depot only after Buyer performed a request to Seller in this description. Implementing this behavior as distributed processes demands that Shipper be notified once the first communication is performed by message passing, that is, there should be a notification from Seller to Shipper for example as follows.

$$\mathsf{Buyer} \to \mathsf{Seller} : ch_1(\boldsymbol{\nu}\, s).\ \mathsf{Seller} \to \mathsf{Shipper} : ch(\boldsymbol{\nu}\, s').\ \mathsf{Shipper} \to \mathsf{Depot} : ch_2(\boldsymbol{\nu}\, t).\ \mathbf{0}.$$

Observe the second description is directly realizable as end-point processes while the first one is not. Even if one may informally write down the first description, it is the second one that can have a precise correspondence with end-point behavior. Thus we preclude descriptions like the first one, by demanding each participant acts only as a result of its local event. We call this principle *connectedness*.

To formalize connectedness, we need to say which participant initiates an action in I that is, the place where the preceding event happens.

Definition 5.1 (*Initiating Participants*). Given an interaction I in which hiding does not occur, its *initiating participants*, denoted by $\mathsf{top}(I)$, is inductively given as:

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

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

Given I, the function top generates a set of participants. The generated set contains the participants that initiate the first action of I. The annotation for a term variable, A for X^A, has now revealed its role, as a signifier of the initiating participant of the behavior embodied by X. We discuss how this allows validation of connectedness in the presence of recursion.

We now present the inductive definition of connectedness. Connectedness is simply defined by tracking active/passive participants of each action. In brief, for each A, A's sending action or its self-contained action (e.g., assignment and evaluation of a conditional guard) should always be immediately preceded by A's receiving action or its another self-contained action. Henceforth we only consider well-typed terms for both global and local calculi, unless otherwise specified.

Definition 5.2 (*Connectedness*). The collection of *connected interactions* Con is inductively generated as follows:

(1)	$A \to B : ch(\boldsymbol{\nu}\, s).\ I' \in \mathsf{Con}$	if $I' \in \mathsf{Con}$ and $\mathsf{top}(I') = \{B\}$
(2)	$A \to B : s\langle \mathsf{op},\ e,\ x\rangle.\ I' \in \mathsf{Con}$	if $I' \in \mathsf{Con}$ and $\mathsf{top}(I') = \{B\}$
(3)	if $e@A$ then I_1 else $I_2 \in \mathsf{Con}$	if $I_1, I_2 \in \mathsf{Con}$ and $\mathsf{top}(I_1) = \mathsf{top}(I_2) = \{A\}$
(4)	$I_1 + I_2 \in \mathsf{Con}$	if $I_1, I_2 \in \mathsf{Con}$ and $\mathsf{top}(I_1) = \mathsf{top}(I_2)$
(5)	$\mu X^A.\ I' \in \mathsf{Con}$	if $\mathsf{top}(I') = \{A\}$
(6)	$x@A := e.\ I' \in \mathsf{Con}$	if $I' \in \mathsf{Con}$ and $\mathsf{top}(I') = \{A\}$
(7)	$I_1 \mid I_2 \in \mathsf{Con}$	if $I_1, I_2 \in \mathsf{Con}$ and $I_1 \mid I_2$ not top-level implies $\mathsf{top}(I_1) = \mathsf{top}(I_2)$
(8)	$(\boldsymbol{\nu} s)I' \in \mathsf{Con}$	if $I' \in \mathsf{Con}$
(9)	$\mathbf{0}, X^A \in \mathsf{Con}$.	

An interaction I is *strongly connected* if and only if $I \in \mathsf{Con}$.

Here, *top-level* is an interaction that is not prefixed. 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. Our theory can generalize to a more relaxed notion of connectedness only requiring that the intersection of the sets of participants of subsequent actions is not empty. Section C.2 in Appendix C analyses different variants of connectedness.

In $\mu X^A.\ I'$, each occurrence of the term variable X can be seen as a link back to the beginning of recursion, that is, $\mu X^A.\ I'$ itself. Hence for guaranteeing connectedness, we need to make sure that the action preceding X should be connected to the beginning of the recursion, that is, the initiating participant of I. For this to happen, we first annotate X with A (we assume that each occurrence of X in I' is indeed annotated with the same A that appears in $\mu X^A.\ I'$), 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.

LEMMA 5.3 (SUBSTITUTION FOR CONNECTEDNESS). *If* $I_1, I_2 \in \mathsf{Con}$ *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. □

LEMMA 5.4 (SUBJECT CONGRUENCE FOR CONNECTEDNESS). *If* $I_1 \equiv I_2$ *and* $I_1 \in \mathsf{Con}$ *then* $I_2 \in \mathsf{Con}$.

PROOF. By induction on the structural rules. □

THEOREM 5.5 (SUBJECT REDUCTION FOR CONNECTEDNESS). *Let* $I \in \mathsf{Con}$ *and* σ *be well-typed. Then* $(\sigma, I) \rightsquigarrow (\sigma', I')$ *implies* $I' \in \mathsf{Con}$.

PROOF. By induction on the reduction rules. See Section C.1 in Appendix C. □

One consequence of the connectedness is 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.

5.3. Well-Threadedness

Well-threadedness is a condition on global descriptions that also concerns causality. As an example, consider the following connected global description:

$$\mathsf{Buyer} \to \mathsf{Seller} : ch_1(\nu\, s).\ \mathsf{Seller} \to \mathsf{Shipper} : ch_2(\nu\, t).$$
$$\mathsf{Shipper} \to \mathsf{Buyer} : ch_3(\nu\, u).\ \mathsf{Buyer} \to \mathsf{Seller} : s\langle \mathsf{op},\ v,\ x\rangle.\ I.$$

In this description, the first interaction tells us that there is a processes in Buyer, say P, which invokes Seller. Concurrent processes such as P are called *threads*. During the next two interaction, P, located at Buyer's, becomes inactive. Then, in the last interaction, P is resumed and communicates to Seller via session channel s, opened in the initial action. We claim that the global code (regardless of I) is unrealizable at end-points. In fact, let us consider the following end-point view:

$$\begin{array}{rlll} \mathsf{Buyer}[& \overline{ch_1}(\nu s).\ \overline{s} \lhd \mathsf{op}\langle v\rangle.\ P_1 \quad | \quad !ch_3(t).\ P_2 &]_{\sigma_1} & | \\ \mathsf{Seller}[& !ch_1(s).\ \overline{ch_2}(\nu t).\ s \rhd \mathsf{op}(x).\ Q &]_{\sigma_2} & | \\ \mathsf{Shipper}[& !ch_2(t).\ \overline{ch_3}(\nu u).\ R &]_{\sigma_2}. & \end{array}$$

The first process of Buyer invokes ch_1 and sends v with operation op in the same session, while the second is service ch_3. Note that $\overline{s} \lhd \mathsf{op}\langle v\rangle$ cannot be located under ch_3, as it belongs to session s. When the three processes interact, first, Buyer invokes ch_1, then Seller invokes ch_2 of Shipper: up to here the interaction follows the original global scenario. However, at this point, the action $s \rhd \mathsf{op}(x)$ is free to react with its dual action $\overline{s} \lhd \mathsf{op}\langle v\rangle$, before Shipper invokes Buyer's other component, the service at ch_3. Thus the sequencing in the global description gets violated.

The fundamental issue in this example is that the given global code assumes a false, or unrealizable, dependency among actions: the last action belongs to a thread that started from the invocation of ch_1, while the description says it should take place as a direct result of the third action at a distinct thread that has been opened by the invocation at ch_3. If a global description is free from false dependencies as such, we say it is *well-threaded*.

In order to give the formal definition of well-threadedness, we first annotate a global interaction with identifiers for threads.

Definition 5.6 (*Annotated Interaction*). *Thread annotated interactions*, or simply *annotated interactions*, written $\mathcal{A}, \mathcal{A}', \ldots$, are given by the following grammar.

$$\begin{array}{rcl} \mathcal{A} ::= & A^{\tau_1} \to B^{\tau_2} : ch(\nu\, \tilde{s}).\ \mathcal{A} \\ & | \ A^{\tau_1} \to B^{\tau_2} : s\langle \mathsf{op},\ e,\ y\rangle.\ \mathcal{A} \\ & | \ x@A^{\tau} := e.\ \mathcal{A} \\ & | \ \mathsf{if}\ e@A^{\tau}\ \mathsf{then}\ \mathcal{A}_1\ \mathsf{else}\ \mathcal{A}_2 \\ & | \ \mathcal{A}_1 \mid^{\tau} \mathcal{A}_2 \\ & | \ \mathcal{A}_1 +^{\tau} \mathcal{A}_2 \\ & | \ \mu^{\tau} X^{A}.\ \mathcal{A} \\ & | \ X^{A}_{\tau} \\ & | \ \mathbf{0} \end{array}$$

where each τ is a natural number. We call $\tau, \tau', \ldots$ occurring in an annotated interaction, *threads*. Further, the grammar of *extended annotated interactions*, $\mathcal{E}, \mathcal{E}', \ldots$, is given as:

$$\mathcal{E} \quad ::= \quad \mathcal{A} \quad | \quad \mathcal{E}_1 \mid \mathcal{E}_2 \quad | \quad (\nu s)\mathcal{E}.$$

Hereafter, whenever irrelevant, with an abuse of notation, we let $\mathcal{A}, \mathcal{A}', \ldots$ range over extended annotations, and simply call them *annotated interactions*.

Our task is to find a notion of "consistent annotation" for $\mathcal{A}$, so that causality specified globally can be precisely realizable locally. We stipulate the following conditions.

(1) *Freshness Condition (FC)*. When a service is invoked, the service's receiving action should be assigned a fresh thread.
(2) *Session Consistency (SC)*. Two distinct interactions in the same session (which are, by session typing, always between the same pair of participants) should be given the same thread number for respective participants.
(3) *Causal Consistency (CC)*. Any occurrence of a thread τ that is not on an output is immediately followed by an occurrence of the same thread τ.

Intuitively, (FC) substantiates SCP, (SC) does so for SP and (CC) both.

Before formally defining these conditions, we fix some terminology. Each $\mathcal{A}$ regarded as an abstract syntax tree 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 5.7 (*Threads*).

(1) If the root of $\mathcal{A}$ is initialization/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 another constructor 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) $\mathcal{A}'$. A *direct predecessor* is a predecessor that has no intermediate predecessor. Symmetrically we define *successor* and *direct successor*.

Note if the root of $\mathcal{A}$ is a predecessor of that of $\mathcal{A}'$, then the former's execution should indeed temporarily precede that of the latter.

Definition 5.8 (*Consistent Thread Annotation*). A well-typed annotated connected interaction $\mathcal{A}$ is *consistent* if the following conditions hold for each of its subtrees, say $\mathcal{A}'$.

— *Freshness Condition* (FC). 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 initialization, then its passive thread should be fresh w.r.t. all of its predecessors (if any).
— *Session Consistency* (SC). 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 initialization that opens s, then the thread by B (resp. by C) of $\mathcal{A}'$ should be equal to the thread by B (resp. by C) of $\mathcal{A}''$.
— *Causal Consistency* (CC). If $\mathcal{A}''$ is the direct successor of $\mathcal{A}'$, then the active thread of $\mathcal{A}''$ should coincide with the passive thread of $\mathcal{A}'$.

We also say I is *well-threaded* when there is a well-formed annotation $\mathcal{A}$ of I (i.e., the result of erasing the annotations from $\mathcal{A}$ coincides with I).

Example 5.9. To see how these conditions work, for some $\tau_i \in \mathbb{N}$, let us annotate the global description given in the beginning of Section 5.3 as follows:

$$\mathsf{Buyer}^{\tau_1} \to \mathsf{Seller}^{\tau_2} : ch_1(\boldsymbol{\nu}\, s).\ \mathsf{Seller}^{\tau_3} \to \mathsf{Shipper}^{\tau_4} : ch_2(\boldsymbol{\nu}\, t).$$
$$\mathsf{Shipper}^{\tau_5} \to \mathsf{Buyer}^{\tau_6} : ch_3(\boldsymbol{\nu}\, u).\ \mathsf{Buyer}^{\tau_7} \to \mathsf{Seller}^{\tau_8} : s\langle \mathsf{op},\ v,\ x\rangle.\ I.$$

In order to satisfy (CC) and (SC), it should be $\tau_6 = \tau_7$ and $\tau_1 = \tau_7$, respectively, which implies $\tau_6 = \tau_1$, violating (FC) of τ_6. Hence this description is not well-threaded. The following annotated interaction satisfies the three conditions

$$\mathsf{Buyer}^{1} \to \mathsf{Seller}^{2} : ch_1(\boldsymbol{\nu}\, s).\ \ \mathsf{Seller}^{2} \to \mathsf{Buyer}^{3} : ch_2(\boldsymbol{\nu}\, t).$$
$$\mathsf{Buyer}^{3} \to \mathsf{Seller}^{2} : t\langle \mathsf{op}_1,\ v_1,\ x\rangle.\ \mathsf{Seller}^{2} \to \mathsf{Buyer}^{1} : s\langle \mathsf{op}_2,\ v_2,\ y\rangle.\ \mathbf{0}.$$

and generates the following correct end-points.

$$\begin{array}{lllll} \mathsf{Buyer}[& \overline{ch_1}(\boldsymbol{\nu} s).\ s \rhd \mathsf{op}_2(y).\ \mathbf{0} & \mid & !ch_2(t).\ \overline{t} \lhd \mathsf{op}_1\langle v_1\rangle.\ \mathbf{0} &] \ \mid \\ \mathsf{Seller}[& !ch_1(s).\ \overline{ch_2}(\boldsymbol{\nu} t).\ t \rhd \mathsf{op}_1(x).\ \overline{s} \lhd \mathsf{op}_2\langle v_2\rangle.\ \mathbf{0} & & &]. \end{array}$$

5.4. Static Checking for Well-Threadedness

This section introduces a type discipline for checking well-threadedness. Let $S, S', \ldots$ range over the finite sets of session channels. Then we define the following typing environment:

Definition 5.10 (*Typing Environment for Well-Threadedness*).

$$\Theta \quad ::= \quad \Theta \cdot \tau : S \quad \mid \quad \Theta, X : \Theta \quad \mid \quad \emptyset.$$

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, that is, iff whenever $\tau : S \in \Theta$ such that $s \in S$, there is at most one $\tau' \neq \tau$ such that $\tau' : S' \in \Theta$ and $s \in S'$.

The typing judgment 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 5.11 (*Type Discipline for Well-Threadedness*). Given an annotated connected and well-typed interaction $\mathcal{A}$, $\Theta \vdash \mathcal{A}$ is derived by the rules in Figure 10.

Here, for convenience, we annotate each inaction $\mathbf{0}$ with a thread, writing $\mathbf{0}^{\tau}$. The notation $\mathsf{top}^{\circ}(\mathcal{A})$ returns the active thread of $\mathcal{A}$ for $\mathcal{A}$ with $\mathbf{0}$ annotated before (for which we set $\mathsf{top}^{\circ}(\mathbf{0}^{\tau}) = \tau$).

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 thread τ_2 is no longer used. The two rules for parallel composition, (WT-PAR) and (WT-PARE), are in precise correspondence with the grammar of annotated interactions defined in Definition 5.6. Note that only in (WT-PAR) we demand the initial active threads of the two components to be identical. This is necessary because (WT-PAR) is used for nested (prefixed) parallel compositions that need to be activated by the same thread. The hiding (WT-RESE) is also only for extended annotated interactions. We then have the following basic property.

THEOREM 5.12 (TYPING SOUNDNESS FOR WELL-THREADEDNESS). *A connected annotated interaction $\mathcal{E}$ is well-threaded if $\Theta \vdash \mathcal{E}$ for some Θ.*

PROOF. In Section D.1 in Appendix D. □

$$(\text{WT-INIT})\ \frac{\Theta, \tau_1 : S \uplus \{\tilde{s}\}, \tau_2 : \{\tilde{s}\} \vdash \mathcal{A} \quad \mathsf{top}^\circ(\mathcal{A}) = \tau_2}{\Theta,\ \tau_1 : S,\ \tau_2 : \emptyset \vdash A^{\tau_1} \to B^{\tau_2} : ch(\boldsymbol{\nu}\, \tilde{s}).\ \mathcal{A}}$$

$$(\text{WT-COMM})\ \frac{\Theta, \tau_1 : S_1, \tau_2 : S_2 \vdash \mathcal{A} \quad \mathsf{top}^\circ(\mathcal{A}) = \tau_2 \quad s \in S_1 \cap S_2}{\Theta, \tau_1 : S_1, \tau_2 : S_2\ \vdash A^{\tau_1} \to B^{\tau_2} : s\langle \mathsf{op},\ e,\ x\rangle.\ \mathcal{A}}$$

$$(\text{WT-ASSIGN})\ \frac{\Theta \vdash \mathcal{A} \quad \mathsf{top}^\circ(\mathcal{A}) = \tau}{\Theta \vdash x@A^\tau := e.\ \mathcal{A}} \qquad (\text{WT-IF})\ \frac{\Theta \vdash \mathcal{A}_i \quad \mathsf{top}^\circ(\mathcal{A}_i) = \tau}{\Theta \vdash \mathsf{if}\ e@A^\tau\ \mathsf{then}\ \mathcal{A}_1\ \mathsf{else}\ \mathcal{A}_2}$$

$$(\text{WT-SUM})\ \frac{\Theta \vdash \mathcal{A}_i \quad \mathsf{top}^\circ(\mathcal{A}_i) = \tau}{\Theta \vdash \mathcal{A}_1 +^\tau \mathcal{A}_2} \qquad (\text{WT-PAR})\ \frac{\Theta \vdash \mathcal{A}_i \quad \mathsf{top}^\circ(\mathcal{A}_i) = \tau}{\Theta \vdash \mathcal{A}_1 \mid^\tau \mathcal{A}_2}$$

$$(\text{WT-PARE})\ \frac{\Theta \vdash \mathcal{E}_i}{\Theta \vdash \mathcal{E}_1 \mid \mathcal{E}_2} \qquad (\text{WT-RESE})\ \frac{\Theta, \tau_1 : S_1 \uplus \{s\}, \tau_2 : S_2 \uplus \{s\} \vdash \mathcal{E}}{\Theta, \tau_1 : S_1, \tau_2 : S_2 \vdash (\boldsymbol{\nu} s)\mathcal{E}}$$

$$(\text{WT-VAR})\ \frac{\Theta \text{ well-formed}}{\Theta, X : \Theta \vdash X_\tau^A} \qquad (\text{WT-ZERO})\ \frac{\Theta \text{ well-formed}}{\Theta \vdash \mathbf{0}^\tau}$$

$$(\text{WT-REC})\ \frac{\Theta, X : \Theta \vdash \mathcal{A} \quad \mathsf{top}^\circ(\mathcal{A}) = \tau}{\Theta \vdash \mu^\tau X^A.\ \mathcal{A}}$$

Fig. 10. Typing rules for checking well-threadedness.

Here, we define $\mathcal{A} \equiv \mathcal{A}'$, $(\sigma, \mathcal{A}) \rightsquigarrow (\sigma', \mathcal{A}')$, etc. exactly following the corresponding relations on unannotated interactions, except that we demand top-level parallel compositions (i.e., those that 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 5.13 (SUBJECT REDUCTION FOR WELL-THREADEDNESS). *Let $\mathcal{A}, \ldots$ an annotated interactions, then*

(1) If $\Theta \vdash \mathcal{A}$ and $\mathcal{A} \equiv \mathcal{A}'$ then $\Theta \vdash \mathcal{A}'$.
(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 $(\sigma, \mathcal{A}) \rightsquigarrow (\sigma', \mathcal{A}')$ then $\Theta \vdash \mathcal{A}'$.

PROOF. (1) follows by induction on the generation rules for $\equiv$. (2) is by (1), noting that if $I \equiv I'$ and $\mathcal{A}$ is a well-threaded 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 induction on $\mathcal{A}$. Note that in the recursion case, we have $\Theta \vdash \mu X.\mathcal{A}$ implies $\Theta \vdash \mathcal{A}[(\mu X.\mathcal{A})/X]$ (because if $\Theta \vdash \mu X.\mathcal{A}$, then $\Theta, X : \Theta \vdash \mathcal{A}$, hence by using induction on $\mathcal{A}$ and thinning we are done). □

We conclude this section by considering the existence of "representative (minimal) annotation" for a well-threaded interaction. We say a consistent annotation $\mathcal{A}_0$ on I is *minimal* if, up to renaming, any well-threaded annotation $\mathcal{A}$ of I arises by collapsing the annotations in $\mathcal{A}_0$. For example, consider the following well-threaded interaction.

$$B \to C : ch(\boldsymbol{\nu}\, t).\ C \to B : t\langle \mathsf{op}_1,\ v_1,\ x_1\rangle.\ B \to A : s_1\langle \mathsf{op}_2,\ v_2,\ x_2\rangle \quad |$$
$$B \to C : ch'(\boldsymbol{\nu}\, t').\ C \to B : t'\langle \mathsf{op}'_1,\ v'_1,\ x'_1\rangle.\ B \to A : s_2\langle \mathsf{op}'_2,\ v'_2,\ x'_2\rangle,$$

which can be annotated (up to injective renaming) as:

$$B^1 \to C^3 : ch(\boldsymbol{\nu}\, t).\ C^3 \to B^1 : t'\langle \mathsf{op}_1,\ v_1,\ x_1\rangle.\ B^1 \to A^5 : s_1\langle \mathsf{op}_2,\ v_2,\ x_2\rangle \quad |$$
$$B^2 \to C^4 : ch'(\boldsymbol{\nu}\, t').\ C^4 \to B^2 : t'\langle \mathsf{op}'_1,\ v'_1,\ x'_1\rangle.\ B^2 \to A^6 : s_2\langle \mathsf{op}'_2,\ v'_2,\ x'_2\rangle.$$

By collapsing labels, we can reach:

$$B^2 \to C^3 : ch(\nu\, t).\ C^3 \to B^2 : t\langle \mathsf{op}_1,\ v_1,\ x_1\rangle.\ B^2 \to A^1 : s_1\langle \mathsf{op}_2,\ v_2,\ x_2\rangle \ \mid$$
$$B^2 \to C^4 : ch'(\nu\, t').\ C^4 \to B^2 : t'\langle \mathsf{op}'_1,\ v'_1,\ x'_1\rangle.\ B^2 \to A^1 : s_2\langle \mathsf{op}'_2,\ v'_2,\ x'_2\rangle.$$

Formally the notion of minimal annotations is defined as follows.

Definition 5.14 (*Minimal Consistent Annotation*). Let I be well-threaded. Its consistent annotation $\mathcal{A}_0$ is *minimal* if for each consistent annotation $\mathcal{A}$ of I, there is a possibly noninjective substitution σ of thread labels such that $\mathcal{A} = \mathcal{A}_0\sigma$.

The next result shows the existence of a minimal consistent annotation whenever an interaction I is well-threaded.

PROPOSITION 5.15 (EXISTENCE OF MINIMAL CONSISTENT ANNOTATION). *Let I be strongly connected. Then, I has a minimal consistent annotation if and only if I is well-threaded.*

PROOF. See Section D.2 in Appendix D. □

Note that this result is proved by giving a minimal typing system for well-threadedness that can also be considered as an inference mechanism to annotate an interaction consistently whenever it is possible.

5.5. Coherence

Well-threadedness not only eliminates false dependency but also allows consistent extraction of threads (i.e., sequences of actions) from a given global code. These threads become the constituents of end-point processes in EPP. The final principle concerns consistency of descriptions of a behavior belonging to the same service, distributed in multiple places in a given global description. First of all, we observe that it is often necessary to *merge* threads to obtain the final end-point behavior of a single service. Consider the following annotated parallel composition.

$$\mathsf{Buyer}^1 \to \mathsf{Seller}^2 : ch(\nu\, s).\ \mathsf{Seller}^2 \to \mathsf{Buyer}^1 : s\langle \mathsf{op}_1,\ e,\ x_1\rangle.\ \mathcal{A}_1 \ \mid$$
$$\mathsf{Buyer}^3 \to \mathsf{Seller}^4 : ch(\nu\, t).\ \mathsf{Seller}^4 \to \mathsf{Buyer}^3 : t\langle \mathsf{op}_2,\ e,\ x_2\rangle.\ \mathcal{A}_2$$

where $\mathsf{op}_1 \neq \mathsf{op}_2$. In this example, Buyer invokes Seller's service at ch twice, in parallel. Because of SCP and the end-point calculus typing (cf. rule (E-TSERV)), in order to construct the code for service ch, we need to merge threads 2 and 4 into one end-point behavior. But the global description is contradictory, since in one invocation the service reacts with op_1, while in the other one the service reacts with op_2.

As the description of a single end-point behavior can be scattered in different portions of the code, we need to guarantee, in EPP, that these descriptions are mergeable. We will call this mergeablity condition *coherence*.

5.5.1. Merging Threads. We define mergeability and merging.

Definition 5.16 (*Mergeability*). *Mergeability*, denoted by $\bowtie$, is the smallest equivalence over typed terms up to $\equiv$ closed under all typed contexts and:

$$(\text{M-IN})\ \frac{\forall i \in (J \cap K).\ (P_i \bowtie Q_i \wedge x_i = y_i) \qquad \forall j \in J \setminus K.\ \forall k \in K \setminus J.\ \mathsf{op}_j \neq \mathsf{op}_k}{s \rhd \Sigma_{j\in J}\, \mathsf{op}_j(x_j).\ P_j \ \bowtie\ s \rhd \Sigma_{k\in K}\, \mathsf{op}_k(y_k).\ Q_k}$$

$$(\text{M-ZERO})\ \frac{\mathsf{fsc}(P) = \emptyset}{P \bowtie \mathbf{0}}.$$

When $P \bowtie Q$, we say P *and* Q *are mergeable*.

Here, a context is any end-point calculus process with some holes. (M-IN) is for branching and says that we can allow differences in that do not overlap, but we do demand each pair of behaviors with the same operation to be identical. If two end-point behaviors are mergeable in the defined technical sense, we can merge them and obtain a single process that simulates both of the two behaviors, by combining missing branches from the both. For instance, the processes $s \rhd \mathsf{go}(x).\ P$ and $s \rhd \mathsf{stop}(x).\ Q$ are mergeable, and the result of merging is simply $s \rhd (\mathsf{go}(x).\ P + \mathsf{stop}(x).\ Q)$. Formally, we give the following definition.

Definition 5.17 (*Merge Operator*). $P \sqcup Q$ is a partial commutative binary operator on typed processes that is well-defined iff $P \bowtie Q$ and satisfies the following rules:

$$
\begin{array}{rcl}
!ch(\tilde{s}).\ P \sqcup !ch(\tilde{s}).\ Q & \stackrel{\text{def}}{=} & !ch(\tilde{s}).\ (P \sqcup Q) \\
\overline{ch}(\boldsymbol{vs}).\ P \sqcup \overline{ch}(\boldsymbol{vs}).\ Q & \stackrel{\text{def}}{=} & \overline{ch}(\boldsymbol{vs}).\ (P \sqcup Q) \\
s \rhd \Sigma_{i \in J} \mathsf{op}_i(x_i).\ P_i \sqcup s \rhd \Sigma_{i \in K} \mathsf{op}_i(y_i).\ Q_i & \stackrel{\text{def}}{=} & s \rhd \left(\begin{array}{l} \Sigma_{i \in J \cap K} \mathsf{op}_i(y_i).\ (P_i \sqcup Q_i) + \\ \Sigma_{i \in J \setminus K} \mathsf{op}_i(y_i).\ P_i + \\ \Sigma_{i \in K \setminus J} \mathsf{op}_i(y_i).\ Q_i \end{array} \right) \\
x := e.\ P \sqcup x := e.\ Q & \stackrel{\text{def}}{=} & x := e.\ (P \sqcup Q) \\
\mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \sqcup \mathsf{if}\ e\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2 & \stackrel{\text{def}}{=} & \mathsf{if}\ e\ \mathsf{then}\ (P_1 \sqcup Q_1)\ \mathsf{else}\ (P_2 \sqcup Q_2) \\
(P_1 \mid P_2) \sqcup (P_3 \mid P_4) & \stackrel{\text{def}}{=} & (P_1 \sqcup P_3) \mid (P_2 \sqcup P_4) \\
s \lhd \mathsf{op}\langle e \rangle.\ P \sqcup s \lhd \mathsf{op}\langle e \rangle.\ Q & \stackrel{\text{def}}{=} & s \lhd \mathsf{op}\langle e \rangle.\ (P \sqcup Q) \\
(P_1 \oplus P_2) \sqcup (Q_1 \oplus Q_2) & \stackrel{\text{def}}{=} & (P_1 \sqcup Q_1) \oplus (P_2 \sqcup Q_2) \\
\mu X.\ P \sqcup \mu X.\ Q & \stackrel{\text{def}}{=} & \mu X.\ (P \sqcup Q) \\
X \sqcup X & \stackrel{\text{def}}{=} & X \\
P \sqcup \mathbf{0} & \stackrel{\text{def}}{=} & P \\
P \sqcup Q & \stackrel{\text{def}}{=} & P' \sqcup Q' \quad (P \equiv P',\ Q \equiv Q'),
\end{array}
$$

where, in the right-hand side of each rule, we assume that each application of the operator to, say, P and Q, is such that $P \bowtie Q$.

The main branching rule says if the operation op_i appears in both terms, then the terms after the prefix (P_i and Q_i) are merged as well (which are ensured to be mergeable by the assumption) otherwise a new branch is added.

5.5.2. Thread Projection and Coherence. Given a consistently 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. We now add a further annotation to each recursion and each recursion variable. Given $\mu^{\tau} X^{A}.\mathcal{A}$ in an annotated interaction, let $\{\tau_i\}$ be the set of threads occurring in, but *not* initiated in, $\mathcal{A}$ (recall that a thread is initiated in $\mathcal{A}$ whenever it occurs passive in a session initiation). Then, we further annotate this recursion as $\mu^{\tau:\{\tau_i\}} X$ and each free X^{A}_{τ} in $\mathcal{A}$ as $X^{A}_{\tau:\{\tau_i\}}$. The added information is used for removing unnecessary recursion from endpoint processes in the following definition.

Definition 5.18 (*Thread Projection*). Given a consistently annotated interaction $\mathcal{A}$, the partial operation $\mathsf{TP}(\mathcal{A}, \tau)$ is defined as follows.

$$\mathsf{TP}(A^{\tau_1} \to B^{\tau_2} : b(\boldsymbol{\nu}\, \tilde{s}).\ \mathcal{A}, \tau) \stackrel{\text{def}}{=} \begin{cases} \overline{b}(\boldsymbol{\nu}\tilde{s}).\ \mathsf{TP}(\mathcal{A}, \tau_1) & \text{if } \tau = \tau_1 \\ !\, b(\tilde{s}).\ \mathsf{TP}(\mathcal{A}, \tau_2) & \text{if } \tau = \tau_2 \\ \mathsf{TP}(\mathcal{A}, \tau) & \text{otherwise} \end{cases}$$

$$\mathsf{TP}(A^{\tau_1} \to B^{\tau_2} : s\langle \mathsf{op},\ e,\ x\rangle.\ \mathcal{A}, \tau) \stackrel{\text{def}}{=} \begin{cases} \overline{s} \lhd \mathsf{op}\langle e\rangle.\ \mathsf{TP}(\mathcal{A}, \tau) & \text{if } \tau = \tau_1 \\ s \rhd \mathsf{op}(x).\ \mathsf{TP}(\mathcal{A}, \tau) & \text{if } \tau = \tau_2 \\ \mathsf{TP}(\mathcal{A}, \tau) & \text{otherwise} \end{cases}$$

$$\mathsf{TP}(\text{if } e@A^{\tau'} \text{ then } \mathcal{A}_1 \text{ else } \mathcal{A}_2, \tau) \stackrel{\text{def}}{=} \begin{cases} \text{if } e \text{ then } \mathsf{TP}(\mathcal{A}_1, \tau') \text{ else } \mathsf{TP}(\mathcal{A}_2, \tau') & \text{if } \tau = \tau' \\ \mathsf{TP}(\mathcal{A}_1, \tau) \sqcup \mathsf{TP}(\mathcal{A}_2, \tau) & \text{otherwise} \end{cases}$$

$$\mathsf{TP}(x@A^{\tau'} := e.\ \mathcal{A}, \tau) \stackrel{\text{def}}{=} \begin{cases} x := e.\ \mathsf{TP}(\mathcal{A}, \tau') & \text{if } \tau = \tau' \\ \mathsf{TP}(\mathcal{A}, \tau) & \text{otherwise} \end{cases}$$

$$\mathsf{TP}(\mathcal{A}_1 +^{\tau'} \mathcal{A}_2, \tau) \stackrel{\text{def}}{=} \begin{cases} \mathsf{TP}(\mathcal{A}_1, \tau') \oplus \mathsf{TP}(\mathcal{A}_2, \tau') & \text{if } \tau = \tau' \\ \mathsf{TP}(\mathcal{A}_1, \tau) \sqcup \mathsf{TP}(\mathcal{A}_2, \tau) & \text{otherwise} \end{cases}$$

$$\mathsf{TP}(\mathcal{A}_1 \mid^{\tau'} \mathcal{A}_2, \tau) \stackrel{\text{def}}{=} \mathsf{TP}(\mathcal{A}_1, \tau') \mid \mathsf{TP}(\mathcal{A}_2, \tau')$$

$$\mathsf{TP}(\mu^{\tau':\{\tilde{\tau}_i\}} X^A.\ \mathcal{A}, \tau) \stackrel{\text{def}}{=} \mu X.\ \mathsf{TP}(\mathcal{A}, \tau) \text{ if } \tau \in \{\tilde{\tau}_i\},\ \mathsf{TP}(\mathcal{A}, \tau) \text{ otherwise.}$$

$$\mathsf{TP}(X^A_{\tau:\{\tilde{\tau}_i\}}, \tau) \stackrel{\text{def}}{=} X \text{ if } \tau \in \{\tilde{\tau}_i\},\ \mathbf{0} \text{ otherwise.}$$

$$\mathsf{TP}(\mathbf{0}, \tau) \stackrel{\text{def}}{=} \mathbf{0}.$$

If $\mathsf{TP}(\mathcal{A}, \tau)$ is undefined then we set $\mathsf{TP}(\mathcal{A}, \tau) = \perp$ (or simply called undefined) .

Here, for initialization and communication, we have three cases:

(1) When the concerned thread coincides with its active thread, in which case we obtain the corresponding output prefix;
(2) When the concerned thread coincides with its passive thread, in which case we obtain the corresponding input prefix; and
(3) When neither applies, in which case we get the projection of the body.

For conditional, assignment and sum, each of which is annotated with a single thread, we have two cases:

(1) When the projecting thread coincides with the thread of the interaction, we simply carry over these constructors to endpoint processes;
(2) If not, we simply merge these threads (or identity in the case of assignment).

Other cases are defined compositionally.

$$\mathsf{threads}(A^{\tau_1} \to B^{\tau_2} : ch'(\boldsymbol{\nu}\, s).\ \mathcal{A}', ch) \stackrel{\text{def}}{=} \begin{cases} \{\tau_2\} \cup \mathsf{threads}(\mathcal{A}', ch) & \text{if } ch = ch' \\ \mathsf{threads}(\mathcal{A}', ch) & \text{otherwise} \end{cases}$$

$$\mathsf{threads}(\mathcal{A}, ch) \stackrel{\text{def}}{=} \begin{cases} \mathsf{threads}(\mathcal{A}', ch) & \text{if } \mathcal{A} \in \left\{ \begin{array}{l} A^{\tau_1} \to B^{\tau_2} : s\langle \mathsf{op},\ e,\ x\rangle.\ \mathcal{A}', \\ x@A^{\tau} := e.\ \mathcal{A}',\ \mu X^{A}_{\tau}.\mathcal{A}' \end{array} \right\} \\ \mathsf{threads}(\mathcal{A}'_1, ch) \cup \mathsf{threads}(\mathcal{A}'_2, ch) & \text{if } \mathcal{A} \in \left\{ \begin{array}{l} \text{if } e@A^{\tau} \text{ then } \mathcal{A}'_1 \text{ else } \mathcal{A}'_2, \\ \mathcal{A}'_1 + \mathcal{A}'_2,\ \mathcal{A}'_1 \mid \mathcal{A}'_2 \end{array} \right\} \end{cases}$$

$$\mathsf{threads}(X^{A}_{\tau}, ch) \stackrel{\text{def}}{=} \mathsf{threads}(\mathbf{0}, ch) \stackrel{\text{def}}{=} \emptyset$$

Fig. 11. Inductive definition of the function $\mathsf{threads}(\mathcal{A}, ch)$.

The notion of coherence assumes mergeability, extending it to interthread consistency. The need for interthread consistency arises because the description of the behavior of a service may as well be scattered over more than one place in a global description. Since each service channel ch uniquely defines a service, we can collect all threads contributing to its behavior by taking the passive thread of each session initialization at ch. For this, we define the mapping $\mathsf{threads}(\mathcal{A}, ch)$ in Figure 11. If two input threads are for the same service channel, then they are equivalent. That is, if $\tau_1, \tau_2 \in \mathsf{threads}(\mathcal{A}, ch)$, then these two threads are parts of the behavior of the same service.

Definition 5.19 (*threads*). 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

$$[\tau]^{\mathcal{A}} = \begin{cases} \mathsf{threads}(\mathcal{A}, ch) & \text{if } \exists\, ch \in \mathsf{channels}(\mathcal{A}) \text{ such that } \tau \in \mathsf{threads}(\mathcal{A}, ch) \\ \{\tau\} & \text{otherwise.} \end{cases}$$

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 5.20 (*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 $\mathcal{A}$, $\mathsf{TP}(\mathcal{A}, \tau) \neq \bot$.
(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 nonannotated interaction I is *coherent* if I has an annotation that is coherent in the given sense.

Example 5.21. Consider the following annotated interaction:

$$\begin{array}{rcl} \mathcal{A} & \stackrel{\text{def}}{=} & A^1 \to B^2 : ch(\boldsymbol{\nu}\, s).\ B^2 \to A^1 : s\langle \mathsf{op}_1,\ \text{“hello”},\ y_1\rangle.\ \mathbf{0}\ +^1 \\ & & A^1 \to B^3 : ch(\boldsymbol{\nu}\, s).\ B^3 \to A^1 : s\langle \mathsf{op}_2,\ \text{“goodbye”},\ y_2\rangle.\ \mathbf{0}\ +^1 \\ & & A^1 \to C^4 : ch'(\boldsymbol{\nu}\, s').\ C^4 \to A^1 : s'\langle \mathsf{op}_1,\ \text{“hi”},\ x\rangle.\ \mathbf{0}. \end{array}$$

In this case, the projections become:

$$\begin{array}{lcl}
\mathsf{TP}(\mathcal{A},1) & \stackrel{\text{def}}{=} & \overline{ch}(\nu s).s \triangleright \mathsf{op}_1(y_1).\,\mathbf{0} \;\oplus \\
& & \overline{ch}(\nu s).s \triangleright \mathsf{op}_2(y_2).\,\mathbf{0} \;\oplus \\
& & \overline{ch'}(\nu s').s' \triangleright \mathsf{op}_1(x).\,\mathbf{0} \\
\mathsf{TP}(\mathcal{A},2) & \stackrel{\text{def}}{=} & (!ch(s).\overline{s} \triangleleft \mathsf{op}_1\langle\text{“hello”}\rangle.\,\mathbf{0}) \sqcup \mathbf{0} \sqcup \mathbf{0} \\
\mathsf{TP}(\mathcal{A},3) & \stackrel{\text{def}}{=} & \mathbf{0} \sqcup (!ch(s).\overline{s} \triangleleft \mathsf{op}_2\langle\text{“goodbye”}\rangle.\,\mathbf{0}) \sqcup \mathbf{0} \\
\mathsf{TP}(\mathcal{A},4) & \stackrel{\text{def}}{=} & \mathbf{0} \sqcup \mathbf{0} \sqcup (!ch'(s').\overline{s'} \triangleleft \mathsf{op}_1\langle\text{“hi”}\rangle.\,\mathbf{0}).
\end{array}$$

Immediately, each of $\mathsf{TP}(\mathcal{A},i)$ $(1 \le i \le 4)$ is defined. Since $\mathsf{threads}(\mathcal{A},ch) = \{2,3\}$ and $\mathsf{threads}(\mathcal{A},ch') = \{4\}$, in order for $\mathcal{A}$ to be coherent, we should have $\mathsf{TP}(\mathcal{A},2) \bowtie \mathsf{TP}(\mathcal{A},3)$, which is not possible.

5.5.3. Subject Reduction for Coherence. Coherence enjoys the following properties.

LEMMA 5.22 (SUBSTITUTION LEMMA FOR COHERENCE). *If $\mu X.I$ is coherent then so is $I[(\mu X.I)/X]$.*

PROOF. See Section E.1 in Appendix E. □

PROPOSITION 5.23 (SUBJECT CONGRUENCE FOR COHERENCE).

(*1*) *$\sqcup$ is partially symmetric and associative, and has the identity $\mathbf{0}$, all up to $\equiv$ that is, $P \sqcup Q$ 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. See Section E.2 in Appendix E. □

In the following, we assume (as noted just before Theorem 5.13) 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 5.24 (SUBJECT REDUCTION FOR COHERENCE). *If $\mathcal{A}$ is coherent and $(\sigma,\mathcal{A}) \rightsquigarrow (\sigma',\mathcal{A}')$, then $\mathcal{A}'$ is also coherent.*

PROOF. See Section E.3 in Appendix E. □

Finally, since $\equiv$ is decidable, so is mergeability, by which we know the following.

PROPOSITION 5.25 (COHERENCE VALIDATION). *Given a consistently annotated $\mathcal{A}$, there is an algorithm that 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 if a given untyped interaction is coherent.

5.6. The End-Point Projection

In this section we introduce the formal definition of endpoint projection and establish its properties. We will say I is *restriction-free* whenever it contains no terms of the

form $(\nu s)I'$ as its subterm. Additionally, $\mathsf{part}(I)$ denotes the set of participants names occurring in I. Moreover, $[\tau] \in A$ is a thread (equivalence class) belonging to A in an interaction. Note that each thread is uniquely assigned to participant as a consequence of session typing and well-threadedness. Recall also being coherent entails being well-typed, connected and well-threaded.

Definition 5.26 (*End-Point Projection*). Let I be a coherent interaction such that $I \equiv (\nu \tilde{s})I'$ 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 $\mathsf{EPP}((\nu \tilde{s})\mathcal{A}, \sigma)$, is given as the following network.

$$(\nu \tilde{s})\Pi_{A \in \mathsf{part}(I)}\, A[\ \Pi_{[\tau] \in A} \bigsqcup_{\tau' \in [\tau]} \mathsf{TP}(\mathcal{A}, \tau')\ ]_{\sigma @ A}.$$

The mapping given here is defined after choosing a specific annotation of an interaction. Intuitively, the end-point projection works as follows. A network with all participants occurring in I is formed and each participant A is assigned the parallel composition of processes each corresponding to an equivalence class of a thread belonging to A. In case such equivalence classes has more than one element a merge operation is performed.

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, that is, it has no free session channels (which is what programmers/designers usually produce).

PROPOSITION 5.27 (INVARIANCE UNDER ANNOTATIONS). *Given a coherent interaction I and a state σ such that $\Gamma \vdash I$ and $\Gamma \vdash \sigma$, let $\mathcal{A}_1$ and $\mathcal{A}_2$ be consistent annotations of I. Then,* $\mathsf{EPP}(\mathcal{A}_1, \sigma) \equiv \mathsf{EPP}(\mathcal{A}_2, \sigma)$.

PROOF. See Section F.1 in Appendix F. □

5.6.1. Pruning. Before stating the correctness of the end-point projection, we define a relation on processes, called *pruning*, for dealing with some information that could be lost during reduction which is still kept in the corresponding reduction in its EPP, due to persistent behavior at service channels. As an example, consider the following interaction

$$\begin{array}{l} \mathsf{Buyer} \to \mathsf{Seller} : ch(\nu\, s).\ \mathsf{Seller} \to \mathsf{Buyer} : s\langle \mathsf{ack} \rangle.\ \mathsf{Buyer} \to \mathsf{Seller} : s\langle \mathsf{go} \rangle\ + \\ \mathsf{Buyer} \to \mathsf{Seller} : ch(\nu\, s).\ \mathsf{Seller} \to \mathsf{Buyer} : s\langle \mathsf{ack} \rangle.\ \mathsf{Buyer} \to \mathsf{Seller} : s\langle \mathsf{stop} \rangle, \end{array} \tag{1}$$

whose EPP is

$$\begin{array}{l} \mathsf{Buyer}[\overline{ch}(\nu s).s \rhd \mathsf{ack}.s \lhd \mathsf{go} \oplus \overline{ch}(\nu s).s \rhd \mathsf{ack}.s \lhd \mathsf{stop}]\ \mid \\ \mathsf{Seller}[!\, ch(s).\overline{s} \lhd \mathsf{ack}.s \rhd (\mathsf{go} + \mathsf{stop})]. \end{array} \tag{2}$$

Now, if apply reduction rules (G-SUM) and (G-INIT) to (1) we obtain

$$\mathsf{Seller} \to \mathsf{Buyer} : s\langle \mathsf{ack} \rangle.\ \mathsf{Buyer} \to \mathsf{Seller} : s\langle \mathsf{go} \rangle.$$

The EPP of this reductum is:

$$\mathsf{Buyer}[s \rhd \mathsf{ack}.s \lhd \mathsf{go}]\ \mid\ \mathsf{Seller}[\overline{s} \lhd \mathsf{ack}.s \rhd \mathsf{go})]. \tag{3}$$

If we compare the two EPPs (before and after reduction), we notice that Seller in (2) has a redundant "stop," which does not occur in (3). This example shows that reduction in a global description can lose information that is still kept in the corresponding reduction in its EPP, due to replication at service channels.

In the following, we write $!R$ when R is a n-fold composition of replications, that is, $R \equiv \mathbf{!}ch_1(\tilde{s}_1).\ R_1 \mid \ldots \mid \mathbf{!}ch_n(\tilde{s}_n).\ R_n$. The asymmetric relation of *pruning* $P \prec Q$, which indicates P is the result of cutting off "unnecessary branches" of Q, is formally defined as follows.

Definition 5.28 (*Pruning*). Let $\Gamma \vdash_A P \triangleright \Delta$ for Γ and Δ minimal and $\Gamma, \Gamma' \vdash_A Q \triangleright \Delta$. If further we have $Q \equiv Q_0 \mid !R$ where $\Gamma \vdash Q_0 \triangleright \Delta$, $\Gamma' \vdash_A R$ and $P \bowtie Q_0$, then we write: $\Gamma \vdash_A P \prec Q \triangleright \Delta$ or $P \prec Q$ for short, and say P prunes Q under $\Gamma; \Delta$. $\prec$ is extended to networks accordingly.

Writing simply $P \prec Q$ does not in fact lose any precision since we can then always reconstruct appropriate typings. $P \prec Q$ indicates P is the result of cutting off "unnecessary branches" of Q, in the light of P's own typing. $\prec$ is in fact a typed strong bisimulation in the sense that $P \prec Q$ means they have precisely the same observable behaviors except for the visible input actions at pruned inputs, either branches or replicated channels. Thus in particular it satisfies the following condition.

LEMMA 5.29 (PRUNING LEMMA).

(*1*) $\prec$ *is a strong reduction bisimulation in the sense that:*
  (*a*) *If* $M \prec N$ *and* $M \rightsquigarrow M'$ *then* $N \rightsquigarrow N'$ *such that* $M' \prec N'$*; and*
  (*b*) *If* $M \prec N$ *and* $N \rightsquigarrow N'$ *then* $M \rightsquigarrow M'$ *such that* $M' \prec N'$.
(*2*) $\prec$ *is transitive, that is,* $M \prec N$ *and* $N \prec R$ *imply* $M \prec R$.

PROOF SKETCH. (1) is because, if $M \prec N$, the branches pruned in M can only be among those that are never used by N, 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 minimal 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. □

5.6.2. EPP Theorem and its Consequences. We are now ready to state the main results of this article. In the following, we write $\Gamma \vdash \sigma$ when the stored values in σ follow the typing in Γ in the obvious sense. We use several notations as follow.

— In (1), $\perp(\Delta)$ denotes the result of replacing each occurrence of type assignment in Δ, say $\tilde{s}[A, B] : \alpha$, with $\tilde{s} : \perp$.
— In (2), $\equiv_{rec}$ denotes the equality induced by the unfolding/folding of process recursion (note they do not change behavior up to strong bisimilarity).
— In (4) and (5), we elaborate reduction with labels for interactions, both for a global calculus and a local calculus, writing $\overset{\ell}{\rightsquigarrow}$ where ℓ ranges over the set of labels of the form:

$$\ell \ ::= \ (A \to B, ch(\tilde{s})) \mid (A \to B, s \triangleright \mathsf{op}\langle v \rangle) \mid \tau.$$

  where $(A \to B, ch(\tilde{s}))$ denotes a session initiation, $(A \to B, s \triangleright \mathsf{op}\langle v \rangle)$ denotes an interaction inside a session s with operator op and value $\tilde{v}$, and the third denotes the remaining, noninteractional reductions (we can also add labels for them, to obtain the same result, since each action in one calculus matches precisely that of the other). The annotated reduction relations for both global and local calculi are given in Appendix F.
— Further, in (4) and (5), we consider only strict reductions in which ν-restricted channels that are active, that is, not under a prefix, are never renamed. For details, see

Appendix F. This allows us to identify the binding relationship between session initialization (say $ch(\tilde{s})$) and a subsequent associated session communication (say at one of s_i in $\tilde{s}$), while staying at the level of reduction. Note for each nonstrict reduction we have the corresponding strict reduction, so considering only the latter does not lose generality.

— In (6), we set $I \downarrow (A, B, s)$ whenever $I \equiv (\nu\tilde{s})(A \to B : s\langle \mathsf{op},\ e,\ x\rangle.\ I' \mid I'')$ for $s \notin \tilde{s}$ and $I \downarrow (A, B, ch)$ whenever $I \equiv (\nu\tilde{s})(A \to B : ch(\nu\,\tilde{s}).\ I' \mid I'')$. Also, for $M \equiv (\nu\tilde{s})(A[\ P\ ]_\sigma \mid M')$ and $s \notin \tilde{s}$, we have (i) $M \downarrow (A, \overline{s})$ whenever $P \equiv \overline{s} \lhd \mathsf{op}\langle e\rangle.\ P' \mid P''$; (ii) $M \downarrow (A, s)$ whenever $P \equiv s \rhd \mathsf{op}(x).\ P' \mid P''$; (iii) $M \downarrow (A, \overline{ch})$ whenever $P \equiv \overline{ch}(\nu\tilde{s}).\ P' \mid P''$; and (iv) $M \downarrow (A, ch)$ whenever $P \equiv !ch(\tilde{s}).\ P' \mid P''$. These definitions extend to annotated interactions accordingly.

THEOREM 5.30 (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 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 $\mathsf{EPP}(\mathcal{A}, \sigma) \rightsquigarrow N$ then there exists $\mathcal{A}'$ such that $(\sigma,\ \mathcal{A}) \rightsquigarrow (\sigma',\ \mathcal{A}')$ such that $\mathsf{EPP}(\mathcal{A}', \sigma') \prec \equiv_{rec} N$.*
(*3*) (completeness) *If $(\sigma,\ \mathcal{A}) \rightsquigarrow (\sigma',\ \mathcal{A}')$ then there exists N such that $\mathsf{EPP}(\mathcal{A}, \sigma) \rightsquigarrow N$ and $\mathsf{EPP}(\mathcal{A}', \sigma') \prec N$.*
(*4*) (soundness with action labels) *If $\mathsf{EPP}(\mathcal{A}, \sigma) \overset{\ell}{\rightsquigarrow} N$ in a strict reduction, then there exists $\mathcal{A}'$ such that $(\sigma,\ \mathcal{A}) \overset{\ell}{\rightsquigarrow} (\sigma',\ \mathcal{A}')$ in a strict reduction such that $\mathsf{EPP}(\mathcal{A}', \sigma') \prec \equiv_{rec} N$.*
(*5*) (completeness with action labels) *If $(\sigma,\ \mathcal{A}) \overset{\ell}{\rightsquigarrow} (\sigma',\ \mathcal{A}')$ in a strict reduction then there exists N such that $\mathsf{EPP}(\mathcal{A}, \sigma) \overset{\ell}{\rightsquigarrow} N$ in a strict reduction and $\mathsf{EPP}(\mathcal{A}', \sigma') \prec N$.*
(*6*) (barbs) *$\mathcal{A} \downarrow (A, B, u)$ if and only if $\mathsf{EPP}(\mathcal{A}, \sigma) \downarrow (A, \overline{u})$ and $\mathsf{EPP}(\mathcal{A}, \sigma) \downarrow (B, u)$ for u either a service or a session channel.*

PROOF. (Outline: for full proof see Appendix F: Section F.5 for (1), Section F.6 for (2, 4) and Section F.7 for (3, 5)). (1) In order to show type preservation we define an intermediate type system that focuses on single threads called *per-thread typing*. Hence, the proof consists of showing (i) global typing implies per-thread typing (and vice versa); (ii) per-thread typing is preserved by EPP (and this is easier as we only look at single threads); and (iii) per-thread typing implies end-point typing (and vice versa); (2, 4) and (3, 5) are proved by induction on the reduction rules; (6) follows from (4, 5). □

COROLLARY 5.31. *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), $\prec_{rec}$ is given as the transitive closure of $\prec \cup \equiv_{rec}$.*

(*1*) (error-freedom) *$\mathsf{EPP}(\mathcal{A}, \sigma)$ does not have a communication error (cf. Section 4.19);*
(*2*) (soundness for multistep reduction) *if $\mathsf{EPP}(\mathcal{A}, \sigma) \rightsquigarrow^n N$ then there exists $\mathcal{A}'$ such that $(\sigma,\ \mathcal{A}) \rightsquigarrow^n (\sigma',\ \mathcal{A}')$ and $\mathsf{EPP}(\mathcal{A}', \sigma') \prec_{rec} N$;*
(*3*) (completeness for multistep reduction) *if $(\sigma,\ \mathcal{A}) \rightsquigarrow^n (\sigma',\ \mathcal{A}')$ then we have $\mathsf{EPP}(\mathcal{A}, \sigma) \rightsquigarrow^n N$ such that $\mathsf{EPP}(I', \sigma') \prec N$;*
(*4*) (progress of projections) *for all $\mathcal{A} \not\equiv \mathbf{0}$ there exists N such that $\mathsf{EPP}(\mathcal{A}, \sigma) \rightsquigarrow N$.*

PROOF. (1) is immediate from Theorem 5.30 (1) and Corollary 4.19. (2) and (3) are by Lemma 5.29 (1,2) and Theorem 5.30 (2, 3), combined with the standard tiling

argument and induction on n. For example, for (2), the case when $n = 1$ is Theorem 5.30 (2). Suppose the statement holds up to n reductions and assume $\mathsf{EPP}(\mathcal{A}, \sigma) \rightsquigarrow^{n+1} N$. By definition this means $\mathsf{EPP}(\mathcal{A}, \sigma) \rightsquigarrow^{n} N_0 \rightsquigarrow N$ for some N_0. Hence by (IH) there exists $\mathcal{A}'_0$ such that $(\sigma,\ \mathcal{A}) \rightsquigarrow^{n} (\sigma'_0,\ \mathcal{A}'_0)$ and $\mathsf{EPP}(\mathcal{A}'_0, \sigma'_0) \prec_{rec} N_0$. By $N_0 \rightsquigarrow N$ and since $\prec_{rec}$ immediately satisfies the same simulation property as $\prec$, $\mathsf{EPP}(\mathcal{A}'_0, \sigma') \rightsquigarrow N''$ such that $N' \prec_{rec} N''$. By Theorem 5.30 (2) again we have $(\sigma'_0, \mathcal{A}'_0) \rightsquigarrow (\sigma', \mathcal{A}')$ such that $\mathsf{EPP}(\mathcal{A}', \sigma') \prec N''$. By transitivy of $\prec_{rec}$ we have $\mathsf{EPP}(\mathcal{A}', \sigma') \prec N'$ as required. (4) follows from Theorem 3.6 and (3). □

Remark 5.32.

(1) Corollary 5.31 (1) indicates 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 run-time through the static validation of global descriptions. Potential properties for validation would include deadlock-freedom, livelock-freedom, and various security properties.
(2) Corollary 5.31 (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.

6. A COMPLETE EXAMPLE OF END-POINT PROJECTION

In the following we illustrate the formal notion of end-point projection we have developed by using 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 end-point processes.

6.1. 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:
Buyer (B), Seller (S), Vendor (V) CreditChecker (CC), Shipper (Sh) and 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 (*reasonable*(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.

6.2. Global Description in the Calculus

We now give the formal description of this protocol using the global calculus. Since the description is long, we divide it into several parts and we also give its annotation. First, we present the basic skeleton of the protocol.

1. $B^1 \to CC^2 : ch_{CC}(\nu\, s).\ CC^2 \to B^1 : s\langle \mathsf{ack} \rangle.$
2. $B^1 \to CC^2 : s\langle \mathsf{companyCheck},\ sellerName,\ compName \rangle.$
3. { $CC^2 \to B^1 : s\langle \mathsf{good} \rangle.\ I_{\mathsf{good}}$
4. +
5. $CC^2 \to B^1 : s\langle \mathsf{bad} \rangle.\ \mathbf{0}$ }

In Line 3, I_{Good} represents the interactions that take place after CreditChecker tells Buyer that the company is good, which is given as follows.

1. $B^1 \to S^3 : ch_S(\nu\, t).\ S^3 \to B^1 : t\langle \mathsf{ack} \rangle.$
2. $B^1 \to S^3 : t\langle \mathsf{quoteReq},\ prod,\ prod \rangle.$
3. $S^3 \to V^4 : ch_V(\nu\, r).\ V^4 \to S^3 : r\langle \mathsf{ack} \rangle.$
4. $\mu^3 X^S.$ {
5. $S^3 \to V^4 : r\langle \mathsf{quoteReq},\ prod,\ prod \rangle.$
6. $V^4 \to S^3 : r\langle \mathsf{quoteRes},\ quote,\ quote \rangle.$
7. $S^3 \to B^1 : t\langle \mathsf{quoteRes},\ quote + 10,\ quote \rangle.$
8. if $reasonable(quote)@B^1$ then
9. $B^1 \to S^3 : t\langle \mathsf{quoteOK},\ cred,\ cred \rangle.$
10. $S^3 \to CC^5 : ch_{CC}(\nu\, u).$
11. $CC^5 \to S^3 : u\langle \mathsf{ack} \rangle.$
12. $S^3 \to CC^5 : u\langle \mathsf{personalCreditCheck},\ cred{:}adr,\ cred{:}adr \rangle.$
13. { $CC^5 \to S^3 : u\langle \mathsf{good} \rangle.\ I'_{\mathsf{good}}$
14. +
15. $CC^5 \to S^3 : u\langle \mathsf{bad} \rangle.$
16. $S^3 \to B^1 : t\langle \mathsf{yourCreditIsBad} \rangle.\ \mathbf{0}$ }
17. else
18. $B^1 \to S^3 : t\langle \mathsf{quoteNotOK} \rangle.\ X_3^S$ }

Finally I'_{Good} in Line 13 is given as follows.

1. $S^3 \to R^6 : ch_R(\nu\, p).$
2. $R^6 \to S^3 : p\langle \mathsf{ack} \rangle.$
3. $S^3 \to R^6 : p\langle \mathsf{deliv},\ adr,\ adr \rangle.$
4. $R^6 \to S^3 : p\langle \mathsf{conf} \rangle.$
5. $S^3 \to B^1 : t\langle \mathsf{conf} \rangle.\ \mathbf{0}$

We can check these descriptions are typable, strongly connected, well-threaded, and coherent. For connectedness, the description given here uses a lot of acks. As we discussed in Carbone et al. [2006b], many of these acks are in fact unnecessary by using a relaxed notion of connectedness.

6.3. End-Point Projection of the Global Interaction

Following the definition of EPP, we first project the global description onto each thread. The first one is Buyer's only thread.

$$
\begin{aligned}
\mathsf{TP}(I,1) \;=\;& \overline{ch_{CC}}(\nu s).\ s \triangleright \mathsf{ack}.\ \overline{s} \triangleleft \mathsf{companyCheck}\langle sellerName\rangle.\ s \triangleright \\
& \{\ \mathsf{good}.\ \overline{ch_S}(\nu t).\ t \triangleright \mathsf{ack}.\ \overline{t} \triangleleft \mathsf{quoteReq}\langle prod\rangle. \\
& \ \mu X.\ \ t \triangleright \mathsf{quoteRes}(quote). \\
& \ \ \mathsf{if}\ reasonable(quote)\mathsf{then}\ \overline{t} \triangleleft \mathsf{quoteOK}\langle cred\rangle. \\
& \ \ \ t \triangleright \{\mathsf{yourCreditIsBad}\ +\ \mathsf{conf}\} \\
& \ \ \mathsf{else}\ \overline{t} \triangleleft \mathsf{quoteNotOK}.\ X \\
& \ + \\
& \ \mathsf{bad}\ \ \}.
\end{aligned}
$$

Note this thread starts before the recursion and go through inside the (global) recursion. Thus the projected end-point behavior also contains recursion.

The next projection is onto the first thread of CreditChecker (note this participant has two threads, 2 and 5).

$$
\begin{aligned}
\mathsf{TP}(I,2) \;=\;& !ch_{CC}(s).\ \overline{s} \triangleleft \mathsf{ack}.\ s \triangleright \mathsf{companyCheck}(compName). \\
& \{\ \overline{s} \triangleleft \mathsf{good}\ \oplus\ \overline{s} \triangleleft \mathsf{bad}\ \}.
\end{aligned}
$$

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.

$$
\begin{aligned}
\mathsf{TP}(I,5) \;=\;& !ch_{CC}(u).\ \overline{u} \triangleleft \mathsf{ack}\langle\rangle.\ u \triangleright \mathsf{personalCreditCheck}(cred{:}adr). \\
& \{\ \overline{u} \triangleleft \mathsf{good}\ \oplus\ \overline{u} \triangleleft \mathsf{bad}\ \}.
\end{aligned}
$$

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.

$$
\begin{aligned}
\mathsf{TP}(I,3) \;=\;& !ch_S(t).\ \overline{t} \triangleleft \mathsf{ack}.\ t \triangleright \mathsf{quoteReq}(prod).\ \overline{ch_V}(\nu r).\ t \triangleright \mathsf{ack}. \\
& \ \mu X.\ \ \overline{r} \triangleleft \mathsf{quoteReq}\langle prod\rangle.\ r \triangleright \mathsf{quoteRes}(quote). \\
& \ \ \overline{t} \triangleleft \mathsf{quoteRes}\langle quote + 10\rangle.\ t \triangleright \\
& \ \ \{\ \mathsf{quoteOK}(cred).\ \overline{ch_{CC}}(\nu u).\ u \triangleright \mathsf{ack}. \\
& \ \ \ \overline{u} \triangleleft \mathsf{personalCreditCheck}\langle cred{:}adr\rangle.\ u \triangleright \\
& \ \ \ \ \{\mathsf{good}.\ \overline{ch_R}(\nu p).\ p \triangleright \mathsf{ack}. \\
& \ \ \ \ \ \overline{p} \triangleleft \mathsf{deliv}\langle adr\rangle.\ p \triangleright \mathsf{conf}.\ \overline{t} \triangleleft \mathsf{conf} \\
& \ \ \ \ \ + \\
& \ \ \ \ \ \mathsf{bad}.\ \overline{t} \triangleleft \mathsf{creditIsBad}\ \} \\
& \ \ \ + \\
& \ \ \ \mathsf{quoteNotOK}.\ X.
\end{aligned}
$$

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 behavior 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.

$$\mathsf{TP}(I,4) \quad = \quad !ch_V(r).\, \overline{t} \lhd \mathsf{ack}.\, \mu X.\, r \rhd \mathsf{quoteReq(prod)}.\, \overline{r} \lhd \mathsf{QuoteRes}\langle \mathsf{quote} \rangle.\, X.$$

Finally we end with the projection onto Thread 6, giving the simple behavior of RoyalMail.

$$\mathsf{TP}(I,6) \quad = \quad !ch_R(p).\, \overline{p} \lhd \mathsf{ack}.\, p \rhd \mathsf{deliv(adr)}.\, \overline{p} \lhd \mathsf{conf}.$$

As before, Thread 6 does not 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 that is, 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:

$$!ch_{CC}(u).\, \overline{u} \lhd \mathsf{ack}\langle\rangle.\, u \rhd \left(\begin{array}{c} \mathsf{personalCreditCheck}(cred{:}adr).\ (\overline{u} \lhd \mathsf{good} \oplus \overline{u} \lhd \mathsf{bad}) \\ + \\ \mathsf{companyCheck}(compName).\ (\overline{u} \lhd \mathsf{good} \oplus \overline{u} \lhd \mathsf{bad}) \end{array} \right).$$

By which we have arrived at the end-point behaviors 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.

7. RELATED WORK

Industrial Development. Since the early 1990s, among industries, there have been many attempts to describing and modeling business processes and protocols. The first attempt back in the early 1990s (such as the Fidelio project by IBM, which later became IBM WebSphere MQ-Workflow [IBM 2010]) was based on individual, proprietary description languages, which was not desirable from the viewpoint of interoperability, leading to such issues as depriving a user of choices in products and vendors once it has started to use a specific product. At the time, no de facto/de jure standards existed that could integrate existing systems in a unified manner. These proprietary technologies also suffered from high integration costs. In the latter part of the 1990s, a standardization group, the Work Flow Management Coalition (WFMC) [WFMC 2010], was formed. Its task was to standardize the description of workflows to enable its interoperability. This had solved part of the issues, that is, the lack of standards in description languages, but could not present a wholesome solution because the underlying workflow technologies were still bound to the proprietary solutions. In 2002, IBM proposed the Web Service Flow Language (WSFL) [IBM 2001] and Microsoft released Xlang [Microsoft 2001] for Biztalk. These two originally proprietary offerings were merged in the development of BPEL (later WS-BPEL, Web Services Business Process Execution Language) in 2003. BPEL solved the problem of integration by utilising Web Service standards, WSDL (Web Services Description Language) and SOAP to make legacy integration part of the standards solution, even though a fully fledged global description is not present in these languages.

Global Descriptions of Communication Behavior. Global methods for describing communication behavior 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 [Needham and Schroeder 1978], message sequence charts (MSC) [Alur et al. 2005; Henriksen et al. 2005; ITU 1996], and UML sequence diagrams [OMG 2004]. These notations are intended to offer a useful aid at the design/specification stage, but do not offer fully fledged programming language, lacking, for example, standard control structures and/or value passing.

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 Baeten et al. [2001], and, to the best of our knowledge, is the first fully expressive language for representing interactions based on a global method. DiCons uses specific programming primitives 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 end-point projection has not been developed in the context of DiCons.

Theoretical studies [Broy 2005, 2007; Broy et al. 2007] offer a formal framework centering on message sequence charts, based on stream-based formalization. End-point projectability is discussed under the name of *realizability*. Many concerns in specifications including refinements are treated in their work on a uniform semantic basis. Streams are one of the powerful semantic models for concurrent interactive behaviors, starting from such early work as Kahn's network. However, a wide range of interactive behavior may naturally be captured by explicit representation of a sequence of input and output communications as found in process algebras in general, and name passing process calculi in particular. For example, a connection between a request and a reply is clearly and precisely captured semantically by channel passing (in the present article in the context of session initiation). This leads to tractable projectability involving various control and other structures in the present work, which may not be found in [Broy 2005, 2007; Broy et al. 2007].

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 [2002] presents an analysis of a business protocol showing how a description of an interparticipant 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 5 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 our work.

As we noted, global notations are often used for representing security protocols. Strand Spaces [Guttman et al. 2001] is a structure for analysing properties of cryptographic protocols. It models protocols as causal chains of interactions, and is often presented in a global notation similar to UML sequence diagrams. Strand Spaces does not offer a fully expressive description language with general control constructs. Their methods for security analysis may be applicable to our global calculus. Briais and Nestmann [2005] present a global notation for representing protocol narrations and relate it to the π-calculus. Since their sole focus is on cryptographic aspects, their global formalism is not intended as a fully expressive language for describing interactions, lacking, for example, channels, conditional and loops, as well as type disciplines for interactions.

Fu et al. [2004, 2005] model choreography and end-point behavior in terms of finite state automata. Given the reduced complexity of their model they are able to provide EPP as well as synthesising global descriptions from end-point processes. BIP [Basu et al. 2006] and REO [Arbab 2004] are formally founded languages for real time systems, which, similarly to choreography, focus on interactions rather than input/output primitives. In the field of service oriented programming, Qiu et al. [2007] have proposed a calculus for modeling choreography that is then mapped into Hoare's CSP [Hoare 1985].

One of the significant contributions that distinguish our work from the other approaches we have listed is that we provided the formal projection algorithm and analysis for a core global, structured programming/description language built on a rigorous semantic foundation and underpinned by formal theorems, which are further illustrated by various nontrivial description examples.

In Honda et al. [2008], we use choreography as a type for multiparty session types providing a simplified end-point projection into end-point types. Recently, Corin et al. [2007] introduce Global Session Graphs, directed graphs for expressing types for multiparty conversations. Their formalism provides an algorithm for going from graphs (global) to role types (end-point) and back. They extend this work to security protected multiparty session executions in Bhargavan et al. [2009]. These works do not study EPPs direct from the global language.

Theories of Types for the π-Calculus. Many theories of types for the π-calculus are studied. In this work, we used session types, a typing discipline that provides a flexible programming style for structural interaction, and is used to statically check the safe and consistent composition of protocols in communication-centric distributed software. Session types also play a key role in EPP theory. 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. As far as we know this formalism [Carbone et al. 2007] is the first extension of session types that supports parallel interaction within the same session.

In addition to the study of session types, the types for processes studied in the literature includes input/output types [Milner 1993; Pierce and Sangiorgi 1996], linear types [Honda 1996; Kobayashi et al. 1996], various kinds of behavioral types, many of which incorporate causality [Igarashi and Kobayashi 2001; Yoshida et al. 2004; Yoshida 1996] and combination of behavioral types and model checking for advanced behavioral analysis [Chaki et al. 2002; Rajamani and Rehof 2002], to name a few. Behavioral 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.

Type Disciplines for Concurrent Programming Languages. Our work shares with many recent works its direction towards well-structured communication-centered programming using types. Pict [Pierce and Turner 2000] is the programming language based on the π-calculus, with rich type disciplines including linear and polymorphic types. Polyphonic C$\sharp$ [Benton et al. 2004] uses a type discipline for safe and sophisticated object synchronization. The interplay of session type disciplines with different programming constructs and program properties has been studied [Bonelli et al. 2005; Dezani-Ciancaglini et al. 2006; Gay and Hole 2005; Honda et al. 1998; Takeuchi et al. 1994; Vasconcelos et al. 2004]. More recently, a similar approach to the session types has been studied in the Singularity OS [Fähndrich et al. 2006]; behaviors in this system are defined in the form of a state machine that specifies desired message

exchange patterns. Messages encapsulate asynchronous method invocation, and consist of information on which method should be invoked, along with the actual arguments to use when the message received. Our end-point calculus is based on synchronous communication following the previous works on the session types; we believe all the technical results can be adapted to the asynchrony.

The EPP theory offers a passage through which these studies (all based on end-point languages and calculi) can be reflected onto global descriptions, as we have demonstrated for session types in the present work.

Process Calculus-Based Analysis of Web Service. Foster et al. [2010] studied a transition-based approach for checking the correctness of the protocols written in CDL including absence of deadlock and livelock. Their approach is limited in untyped (data and message are abstracted and represented simply as base types) and communication without name passing. Extending their model abstraction to support data variable expression and message correlation (state, types and channels) based on the syntax and semantics of the π-calculus is an interesting future topic. In general, pre-type checking would provide opportunity to reducing the size of static state machines in model checking.

There has been a series of works studying security-related aspects of Web services in their series of works (whose origin lies in the the spi-calculus [Abadi and Gordon 1999] and the applied π-calculus [Abadi and Fournet 2001]). Their initial work [Gordon and Pucella 2002] may be the first to apply the π-calculus to Web Services, focusing on security concern. In a more recent work, Bhargavan et al. [2006] 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 applied π-calculus. 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 [2006] give a model of orchestrations of Web services using an extension of π-calculus to join patterns. They propose a typing system for guaranteeing a notion of smoothness that is a constraint on input join patterns such that their subjects (channels) are colocated in order to avoid a classical global consensus problem during communication. Reflecting the centralized nature of orchestration, neither a global calculus nor end-point projection is considered.

A bisimulation-based correspondence between choreography and orchestration in the context of Web services has been studied by Busi et al. [2006], where a notion of state variables is used in the semantics of the orchestration model. They operationally relate choreography to orchestration. In Guidi et al. [2006], the same authors introduce SOCK, a calculus for Web services based on end-point descriptions. This formalism models networks of participants and their local behavior enhanced with a mechanism for controlling communication at runtime based on logical conditions on the participant's store. In our work, communication structures are represented by session types, hence interaction is statically controlled and its error freedom is guaranteed by a sound type system. Their dynamic control allows concrete reasoning on the communicated values, but requires heavy runtime mechanisms. It is an interesting topic to integrate static and dynamic methods for efficient and flexible control of interactions. Neither strong type systems for communication nor descriptive principles for EPP are studied in their work.

Castagna et al. [2008] and Bravetti and Zavattaro [2007] study theoretical aspects of contracts for Web Services. While the subtyping relation for session types is quite strong but, as seen, necessary at choreography level, a coarser relation on contracts known as compliance is important from the client viewpoint when dealing with service

discovery and run-time compatibility. The conversation calculus [Caires and Vieira 2009; Vieira et al. 2008] is an extension of the π-calculus to context-sensitive interactions with service and request primitives and local exceptions. In Caires and Vieira [2009], they also proposed a typing system for guaranteeing error-freedom based on global and local types, and its extension for ensuring a progress based on a well-founded ordering on events. Like other works, the EPP is only studied at the type level.

For a further reference on session types, see Dezani-Ciancaglini and de' Liguoro [2009], which covers a wide range of related works on sessions and session types.

8. CONCLUSIONS, EXTENSIONS AND FUTURE WORK

This article introduced a pair of typed formalisms for interaction, one based on global descriptions and another an applied π-calculus, both using a parallel extension of session types. Both calculi are based on a new extension of session types, which can handle parallel interaction in one session. We developed a theory of end-point projection, which enunciates three conditions for global descriptions and presents a sound and complete mapping from well-structured global descriptions to their end-point counterpart. We have shown this mapping is sound and complete in the sense that all and only behaviors in the original descriptions are realized as communications among the projected end-point processes. Global descriptions have been practiced in various engineering contexts for a long time. The present work is a trial to realize its potential as a mathematically well-founded programming method, centring on type structures for communication. For practical applications, the EPP theory demands further exploration, capturing all basic concurrent programming primitives, including mutual exclusion, exceptions and timeout. In the following, we list some of the possible extensions and future work.

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 invokable behaviors, that is, 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 [Pierce 2002]. For end-point projection, it is treated just as assignment.

Intraparticipant Interaction. In Section 3.3, 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 that primarily specifies interorganizational 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 intraparticipant interactions. First, the typing rules in Figure 3, takes off (G-TCOMINV) and refines (G-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. The key change is in well-threadedness. When $A = B$, the condition of session consistency in well-threadedness is problematic since we do not know which of the two threads should be given to which participant. However stipulating the following condition solves this ambiguity.

Local Causal Consistency. If there is a downward sequence of actions that 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.

We also note this condition is a consequence of the definition of well-threadedness in the theory without intraparticipant interaction so that we are not adding any extra constraint to interparticipant interactions.

Shared Channel Passing. Shared channel passing is a practically useful extension for business protocols, for example in the scenarios where participants need to send links to other participants. A typical example is when Buyer wants to buy a book from Seller, but Buyer does not know Seller's address (service channel) on the net. The only information Buyer has is a service channel book of DirectoryService, which will send back the address of Seller to Buyer, which in turn interacts with 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 name, Seller. Can we have a consistent EPP theory with unknown participants and channels? In practice, this has been an open problem left in WS-CDL's current specification (which allows channel passing only for fixed participants). A possible extension of the present EPP theory to channel passing is discussed in Carbone et al. [2006b, Section 17], using anonymous participant names as placeholders. Channel passing may also be used for a consistent representation of synchronization mechanisms. As a related point, while the three principles we have presented do allow description of many known use cases, some of these conditions can indeed be loosened with essentially the same technical results. Such an extension is discussed by Carbone et al. [2006b, Section 13.2/3].

Session Channel Passing (Delegation). Similarly to shared channel passing, session channel passing or simply *delegation* is another practically useful extension that this work does not address. The main difference with shared channel passing is that instead of passing a channel name representing a public (shared) service, delegation transmits a private session channel from another session. Such a feature may prove to be very useful in several situations. As an example, consider our buyer-seller protocol from Section 2 and suppose that we wish to optimize the notification from Shipper to Buyer. A possible solution is for Seller to pass her session channel s, used for communicating with Buyer, directly to Shipper who can use it for directly notifying Buyer with the transaction details.

Practical Use of EPP. The EPP theory has been developed with practical use in mind. There are several engineering scenes where we consider the theory and its extensions may be useful.

— *Code generation.* We can create a (perhaps multilanguage) *complete distributed application* by projecting a detailed global description to each of its end-points.
— *Prototype generation.* Projection can also be used for generating a *skeleton code* for each end-point that only contains basic communication behavior, to be elaborated to full code. This is already used in [PI4SOA 2008].
— *Use of conformance.* A team of programmers initially agree on a shared global specification for communications among end-points: during/after programming, each programmer can check if her/his code conforms to the specification by conformance checking against projection. This scheme also applies to conformance of existing services/libraries to a given global scenario.
— *Runtime monitoring, testing, and debugging.* At runtime, each end-point can check if ongoing communications at his/her site conform to the global description by checking against its projection to that end-point. The monitoring can also be used for debugging and testing existing code.

Further, various static analyses/logical validation methods would become available for a global description from their well-developed end-point counterpart. The present article is intended to be an initial trial towards a well-founded specification/implementation/validation framework for communication-centered programming, underpinned by a theory of EPP.

APPENDIXES

A. AUXILIARY DEFINITIONS AND PROOFS OF SECTION 3

A.1. Free Session Channels, Free Term Variables and Service Channels

The set of free session channels, denoted by fsc, is defined as follows.

Term	fsc
$A \to B : ch(\boldsymbol{v}\,\tilde{s})\,.\,I$	$\mathsf{fsc}(I)\backslash\tilde{s}$
$A \to B : s\langle \mathsf{op},\ e,\ y\rangle\,.\,I$	$\mathsf{fsc}(I) \cup \{s\}$
$x@A := e\,.\,I,\ \mu X\,.\,I$	$\mathsf{fsc}(I)$
if $e@A$ then I_1 else I_2, $I_1 + I_2$, $I_1 \mid I_2$	$\mathsf{fsc}(I_1) \cup \mathsf{fsc}(I_2)$
$(\boldsymbol{v}s)I$	$\mathsf{fsc}(I)\backslash\{s\}$
X, $\mathbf{0}$	$\{\}$

The set of free term variables, denoted by fv, is defined as follows.

Term	fv
$A \to B : ch(\boldsymbol{v}\,\tilde{s})\,.\,I$, $A \to B : s\langle \mathsf{op},\ e,\ y\rangle\,.\,I$, $x@A := e\,.\,I$, $(\boldsymbol{v}s)I$	$\mathsf{fv}(I)$
if $e@A$ then I_1 else I_2, $I_1 + I_2$, $I_1 \mid I_2$	$\mathsf{fv}(I_1) \cup \mathsf{fv}(I_2)$
$\mu X\,.\,I$	$\mathsf{fv}(I)\backslash\{X\}$
X	$\{X\}$
$\mathbf{0}$	$\{\}$

Finally, the set of service channels, denoted by channels, is defined as follows.

Term	channels
$A \to B : ch(\boldsymbol{v}\,\tilde{s})\,.\,I$	$\{ch\} \cup \mathsf{channels}(I)$
$A \to B : s\langle \mathsf{op},\ e,\ y\rangle\,.\,I$,$x@A := e\,.\,I$, $(\boldsymbol{v}s)I$, $\mu X\,.\,I$	$\mathsf{channels}(I)$
if $e@A$ then I_1 else I_2, $I_1 + I_2$, $I_1 \mid I_2$	$\mathsf{channels}(I_1) \cup \mathsf{channels}(I_2)$
X, $\mathbf{0}$	$\{\}$

A.2. Grouping of Session Channels

We observe free session channels can be grouped in an arbitrary way in the typing rules: the following proposition says that grouping of free session channels in the session typing is in fact arbitrary, that is, they become relevant only when they are abstracted by session initialization or session channel restriction.

PROPOSITION A.1 (GROUPING OF SESSION CHANNELS).

(1) *Suppose* $\Gamma \vdash I \triangleright \Delta \cdot \tilde{s}_1\tilde{s}_2[A, B] : \beta$. *Then there exist* α_1 *and* α_2 *such that* $\Gamma \vdash I \triangleright \Delta \cdot \tilde{s}_1[A, B] : \alpha_1 \cdot \tilde{s}_2[A, B] : \alpha_2$.
(2) *Suppose* $\Gamma \vdash I \triangleright \Delta \cdot \tilde{s}_1[A, B] : \alpha_1 \cdot \tilde{s}_2[A, B] : \alpha_2$. *Then there exists* β *such that* $\Gamma \vdash I \triangleright \Delta \cdot \tilde{s}_1\tilde{s}_2[A, B] : \beta$.

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

A.3. Proof of Proposition 3.21 (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. We will prove a stronger result. Let

(a) ψ be a the set of mutually disjoint vectors containing (only) the free session names in I (cf. Proposition A.1) (each vector in ψ, say $\tilde{s}$, can be used as, for instance, $\tilde{s}[A, B] : \alpha$);
(b) I be typable with a session typing that 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, for some Γ_0 and Δ_0, we have $\Gamma_0 \vdash I \triangleright \Delta_0$ such that (1) Δ_0 conforms to ψ; and (2) whenever $\Gamma \vdash I \triangleright \Delta$ such that Δ 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 proceeds by induction on the minimal typing rules reported in Figure 12. The new rules inductively construct the minimal typing. In Figure 12, it is assumed that the subject term (the term to be type checked) is already type safe. Note that the rules are sound, that is, $\Gamma \vdash_{\mathsf{min}} I \triangleright \Delta$ implies $\Gamma \vdash I \triangleright \Delta$ (this directly follows by induction on the minimal typing rules).

In the figure we write $\mathsf{tvar}(\alpha)$ for the set of type variables in α. We need some care in calculating minimal typing in the presence of recursive types, for which we follow Dezani-Ciancaglini et al. [2006]. This is treated by (G-MTVAR) and (G-MTREC), as well as by (G-MTINIT$_{1/2}$).

In (G-MTVAR), when we introduce a term variable, we also introduce a type variable. ψ' in the premise refers to the disjoint vectors that extend ψ so that, after abstracting session channels by initialization, the result is precisely ψ mentioned in **(a)**.

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 $\tilde{s}_i[A_i, B_i] : \mu\mathbf{t}.\alpha_i$. If not, then we get $\tilde{s}_i[A_i, B_i] : \alpha_i$.

Finally in (G-MTINIT$_1$), when type variables occur in β, we simply replace them with $\mathtt{end}$, where $\beta[\mathtt{end}/\tilde{\mathbf{t}}]$ is the result of substituting $\mathtt{end}$ 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$_2$), (G-MTSUM) and (G-MTIF), the lub operator $\bigtriangledown$ is

$$(\text{G-MTInit}_1)\ \frac{\Gamma \vdash_{\mathsf{min}} I \rhd \Delta \cdot \tilde{s}\,[B, A] : \beta \quad ch \notin dom(\Gamma) \quad \mathsf{tvar}(\beta) \subseteq \tilde{\mathbf{t}}}{\Gamma,\ ch@B : (\tilde{s})(\beta[\mathtt{end}/\tilde{\mathbf{t}}]) \vdash_{\mathsf{min}} A \to B : ch(\boldsymbol{\nu}\,\tilde{s}).\ I \rhd \Delta}$$

$$(\text{G-MTInit}_2)\ \frac{\Gamma,\ ch@B : (\tilde{s})\alpha \vdash_{\mathsf{min}} I \rhd \Delta \cdot \tilde{s}\,[B, A] : \beta \quad \mathsf{tvar}(\beta) \subseteq \tilde{\mathbf{t}}}{\Gamma,\ ch@B : (\tilde{s})(\alpha \bigtriangledown \beta[\mathtt{end}/\tilde{\mathbf{t}}]) \vdash_{\mathsf{min}} A \to B : ch(\boldsymbol{\nu}\,\tilde{s}).\ I \rhd \Delta}$$

$$(\text{G-MTComm})\ \frac{\Gamma \vdash_{\mathsf{min}} I \rhd \Delta \cdot \tilde{s}\,[A, B] : \alpha_j \quad \Gamma \vdash e@A : \theta_j \quad \Gamma \vdash x@B : \theta_j \quad s \in \{\tilde{s}\}}{\Gamma \vdash_{\mathsf{min}} A \to B : s\langle \mathsf{op}_j,\ e,\ x\rangle.\ I \rhd \Delta \cdot \tilde{s}\,[A, B] : s \blacktriangleleft \Sigma_{j\in\{j\}} op_j(\theta_j).\ \alpha_j}$$

$$(\text{G-MTComInv})\ \frac{\Gamma \vdash_{\mathsf{min}} I \rhd \Delta \cdot \tilde{s}\,[B, A] : \alpha_j \quad \Gamma \vdash e@A : \theta_j \quad \Gamma \vdash x@B : \theta_j \quad s \in \{\tilde{s}\}}{\Gamma\ \vdash_{\mathsf{min}}\ A \to B : s\langle \mathsf{op}_j,\ e,\ x\rangle.\ I\ \rhd\ \Delta \cdot \tilde{s}\,[B, A] : s \blacktriangleright \Sigma_{i\in\{j\}} \mathsf{op_i}(\theta_i).\ \alpha_i}$$

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

$$(\text{G-MTSum})\ \frac{\Gamma_1 \vdash_{\mathsf{min}} I_1 \rhd \Delta_1 \quad \Gamma_2 \vdash_{\mathsf{min}} I_2 \rhd \Delta_2}{\Gamma_1 \bigtriangledown \Gamma_2 \vdash_{\mathsf{min}} I_1 + I_2 \rhd \Delta_1 \bigtriangledown \Delta_2}$$

$$(\text{G-MTIf})\ \frac{\Gamma_1 \vdash_{\mathsf{min}} e@A : \mathtt{bool} \quad \Gamma_2 \vdash_{\mathsf{min}} e@A : \mathtt{bool} \quad \Gamma_1 \vdash_{\mathsf{min}} I_1 \rhd \Delta_1 \quad \Gamma_2 \vdash_{\mathsf{min}} I_2 \rhd \Delta_2}{\Gamma_1 \bigtriangledown \Gamma_2 \vdash_{\mathsf{min}} \mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2 \rhd \Delta_1 \bigtriangledown \Delta_2}$$

$$(\text{G-MTPar})\ \frac{\Gamma_1 \vdash_{\mathsf{min}} I_1 \rhd \Delta_1 \quad \Gamma_2 \vdash_{\mathsf{min}} I_2 \rhd \Delta_2}{\Gamma_1 \bigtriangledown \Gamma_2 \vdash_{\mathsf{min}} I_1 \mid I_2 \rhd \Delta_1 \bullet \Delta_2} \qquad (\text{G-MTRes}_1)\ \frac{\Gamma \vdash_{\mathsf{min}} I \rhd \Delta,\ \tilde{s}_1 s \tilde{s}_2[A, B] : \alpha}{\Gamma \vdash_{\mathsf{min}} (\boldsymbol{\nu} s) I \rhd \Delta\ \cdot\ \tilde{s}_1 \tilde{s}_2 : \bot}$$

$$(\text{G-MTRes}_2)\ \frac{\Gamma \vdash_{\mathsf{min}} I \rhd \Delta,\ \tilde{s}_1 s \tilde{s}_2 : \bot}{\Gamma \vdash_{\mathsf{min}} (\boldsymbol{\nu} s) I \rhd \Delta\ \cdot\ \tilde{s}_1 \tilde{s}_2 : \bot} \qquad (\text{G-MTRes}_3)\ \frac{\Gamma \vdash_{\mathsf{min}} I \rhd \Delta\ \cdot\ \varepsilon : \bot}{\Gamma \vdash_{\mathsf{min}} I \rhd \Delta}$$

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

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

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

Note. Definition of ψ is given in the proof of Proposition 3.21.

Fig. 12. Minimal typing rules for global calculus.

used, for which we observe that $\alpha_{1,2} \Subset \beta$ implies $\alpha_1 \bigtriangledown \alpha_2 \Subset \beta$. This implies that the cases for these three rules are immediate. The remaining rules are also direct from the shape of each pair of the corresponding rules. □

A.4. Proof of Theorem 3.24 (Subject Reduction)

(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 $(\sigma, I) \rightsquigarrow (\sigma', I')$ 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 $(\sigma, I) \rightsquigarrow (\sigma', I')$ imply $\Gamma \vdash \sigma'$ and $\Gamma \vdash I'$.

Proof.

(1) We will prove this by induction on the structural congruence rules.
— The proof is trivial for all cases that define | and + to be commutative monoids Honda et al. [1998].

— *Case* $(\boldsymbol{\nu s})I \mid I' \equiv (\boldsymbol{\nu s})(I \mid I')$ for $s \notin \mathsf{fsc}(I')$. Suppose that $\Gamma \vdash (\boldsymbol{\nu s})I \mid I' \rhd \Delta$. By (G-TPAR), we have that $\Delta = \Delta_1 \bullet \Delta_2$ such that $\Gamma \vdash (\boldsymbol{\nu s})I \rhd \Delta_1$ and $\Gamma \vdash I' \rhd \Delta_2$. The judgment $\Gamma \vdash (\boldsymbol{\nu s})I \rhd \Delta_1$ can be derived either by (G-TRES$_1$), (G-TRES$_2$) or (G-TRES$_3$). We only analyse the case corresponding to (G-TRES$_1$) as the other two are very similar. If we apply rule (G-TRES$_1$) then there exists Δ' such that $\Delta_1 = \Delta' \cdot \tilde{s}_1\tilde{s}_2 : \bot$ and $\Gamma \vdash I \rhd \Delta' \cdot \tilde{s}_1 s \tilde{s}_2[A, B] : \alpha$. We now have three subcases:
  (a) $\tilde{s}_1 s \tilde{s}_2 \cap \mathsf{fsc}(\Delta_2) = \emptyset$. By (G-TPAR), $\Gamma \vdash I \mid I' \rhd \Delta' \cdot \tilde{s}_1 s \tilde{s}_2[A, B] : \alpha \bullet \Delta_2$ and, by (G-TRES$_1$), $\Gamma \vdash (\boldsymbol{\nu s})(I \mid I') \rhd \Delta$;
  (b) $\tilde{t}_1 s \tilde{t}_2 \in \mathsf{fsc}(\Delta_2)$ for some $\tilde{t}_1$, $\tilde{t}_2$ different from $\tilde{s}_1$, $\tilde{s}_2$. Then, there exists Δ'_2 such that $\Delta_2 = \Delta'_2 \cdot \tilde{t}_1 s \tilde{t}_2[C, D] : \beta$ (or $\Delta_2 = \Delta'_2 \cdot \tilde{t}_1 s \tilde{t}_2 : \bot$). As $s \notin \mathsf{fsc}(I')$, applying strengthening to $\Gamma \vdash I' \rhd \Delta_2$, we get $\Gamma \vdash I' \rhd \Delta'_2 \cdot \tilde{t}_1\tilde{t}_2[C, D] : \beta$ ($\Gamma \vdash I' \rhd \Delta'_2 \cdot \tilde{t}_1\tilde{t}_2 : \bot$). Now, by (G-TPAR), we have that $\Gamma \vdash I \mid I' \rhd \Delta' \cdot \tilde{s}_1 s \tilde{s}_2[A, B] : \alpha \bullet \Delta'_2 \cdot \tilde{t}_1\tilde{t}_2[C, D] : \beta$ ($\Gamma \vdash I \mid I' \rhd \Delta' \cdot \tilde{s}_1 s \tilde{s}_2[A, B] : \alpha \bullet \Delta'_2 \cdot \tilde{t}_1\tilde{t}_2 : \bot$) noting that $\bullet$ must be well-defined because $\tilde{t}_1\tilde{t}_2$ cannot occur in Δ_1 otherwise $\Delta = \Delta_1 \bullet \Delta_2$ would not be defined as it should also contain s (cf. Definition 3.11). Finally, by (G-TRES$_1$), we get $\Gamma \vdash (\boldsymbol{\nu s})(I \mid I') \rhd \Delta' \cdot \tilde{s}_1\tilde{s}_2[A, B] : \alpha \bullet \Delta'_2 \cdot \tilde{t}_1\tilde{t}_2[C, D] : \beta$ ($\Gamma \vdash (\boldsymbol{\nu s})(I \mid I') \rhd \Delta' \cdot \tilde{s}_1\tilde{s}_2[A, B] : \alpha \bullet \Delta'_2 \cdot \tilde{t}_1\tilde{t}_2 : \bot$) and by Proposition 3.15, weakening, $\Gamma \vdash (\boldsymbol{\nu s})(I \mid I') \rhd \Delta$;
  (c) $\tilde{s}_1 s \tilde{s}_2 \in \mathsf{fsc}(\Delta_2)$. This case is not possible because it would make $\Delta = \Delta_1 \bullet \Delta_2$ undefined (cf. Definition 3.11).

  Similarly, we can prove that $\Gamma \vdash (\boldsymbol{\nu s})(I \mid I') \rhd \Delta$ implies $\Gamma \vdash (\boldsymbol{\nu s})I \mid I' \rhd \Delta$.

(2) In order to prove this, we will prove a stronger result, that is, $\Gamma \vdash I \rhd \Delta$ and $(\sigma, I) \rightsquigarrow (\sigma', I')$ imply $\Gamma \vdash I \rhd \Delta'$ with $\mathsf{fsc}(\Delta) \subseteq \mathsf{fsc}(\Delta')$ and one of the following statements is true:
— $\Delta = \Delta'$
— $\Delta = \Delta_1 \cdot \tilde{s}[A, B] : \alpha$ and $\Delta' = \Delta_1 \cdot \tilde{s}[A, B] : \alpha'$ for $\alpha \searrow \alpha'$.
With $\alpha \searrow \alpha'$ we refer to the following reduction on types:

$$s \blacktriangleright \Sigma_i \mathsf{op}_i(\theta_i).\ \alpha_i \searrow \alpha_j \qquad\qquad s \blacktriangleleft \Sigma_i \mathsf{op}_i(\theta_i).\ \alpha_i \searrow \alpha_j$$

$$\frac{\alpha_1 \searrow \alpha'_1}{\alpha_1 \,|\, \alpha_2 \searrow \alpha'_1 \,|\, \alpha_2} \qquad \frac{\alpha_2 \searrow \alpha'_2}{\alpha_1 \,|\, \alpha_2 \searrow \alpha_1 \,|\, \alpha'_2} \qquad \frac{\alpha[\mu \mathbf{t}.\ \alpha/\mathbf{t}] \searrow \alpha'}{\mu \mathbf{t}.\ \alpha \searrow \alpha'}.$$

The proof proceeds by induction on the depth of the derivation of $(\sigma, I) \rightsquigarrow (\sigma', I')$.

Basic cases.
— *Case* (COMM). By hypothesis, we have $(\sigma, A \to B : s\langle \mathsf{op},\ e,\ x\rangle.\ I) \rightsquigarrow (\sigma', I)$ and $\Gamma \vdash A \to B : s\langle \mathsf{op},\ e,\ x\rangle.\ I \rhd \Delta$. Now, the only applicable rules are (G-TCOMM) and (G-TCOMMINV). The cases are similar, so we will inspect only the first one. We then have that $\Delta = \Delta_1 \cdot \tilde{s}\,[A, B] : \Sigma_{j \in J} s \blacktriangleleft op_j(\theta_j).\ \alpha_j$ and $\Gamma \vdash I \rhd \Delta_1 \cdot \tilde{s}\,[A, B] : \alpha_j$ with $\Gamma \vdash \sigma'$.
— *Case* (INIT). We have $(\sigma, A \to B : ch(\boldsymbol{\nu}\,\tilde{s}).\ I) \rightsquigarrow (\sigma, (\boldsymbol{\nu}\tilde{s})I_i)$. By applying (G-TINIT), we have that $\Gamma',\ ch@B : (\tilde{s})\alpha \vdash A \to B : ch(\boldsymbol{\nu}\,\tilde{s}).\ I \rhd \Delta$ 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 (G-TRES$_1$) once and (G-TRES$_2$) repeatedly, we have $\Gamma',\ ch@B : (\tilde{s})\alpha \vdash (\boldsymbol{\nu}\tilde{s})I \rhd \Delta \cdot \epsilon : \bot$ and by rule (G-TRES$_3$), we can get $\Gamma',\ ch@B : (\tilde{s})\alpha \vdash (\boldsymbol{\nu}\tilde{s})I \rhd \Delta$.
— *Case* (REC). We have $(\sigma, \mu X.\ I) \rightsquigarrow (\sigma, I[\mu X.\ I/X])$ and $\Gamma \vdash \mu X.\ I \rhd \Delta$. The only applicable rule is (G-TREC), which implies $\Gamma \cdot X : \Delta \vdash I : \Delta$. But, by Lemma 3.23, we have that $\Gamma \vdash I[\mu X.\ I/X] \rhd \Delta$.
— *Case* (IFTT). We have that $(\sigma, \mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2) \rightsquigarrow (\sigma, I_1)$ and $\Gamma \vdash \mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2 \rhd \Delta$. Applying rule (G-TIF) we have $\Gamma \vdash I_1 \rhd \Delta$.
— *Case* (IFFF). Similar to the previous case.
— *Case* (SUM). Similar to the previous case.

— *Case* (ASSIGN). We have that $(\sigma, x@A := e.\ I) \rightsquigarrow (\sigma', I)$ and $\Gamma \vdash x@A := e.\ I \rhd \Delta$. Now, applying the rule (G-TASSIGN) we get $\Gamma \vdash I \rhd \Delta$ and $\Gamma \vdash \sigma'$.

Inductive cases.

— *Case* (PAR). From $(\sigma, I_1 \mid I_2) \rightsquigarrow (\sigma', I_1' \mid I_2)$, we get $(\sigma, I_1) \rightsquigarrow (\sigma', I_1')$. 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 (G-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 $\Gamma \vdash I_1' \rhd \Delta_1'$ and $\Gamma \vdash \sigma'$. If $\Delta_1 = \Delta_1'$ then the proof is trivial. Instead, if $\Delta_1 = \Delta'' \cdot \tilde{s}[A, B] : \alpha$ and $\Delta_1' = \Delta'' \cdot \tilde{s}[A, B] : \alpha'$ with $\alpha \searrow \alpha'$ then, noting that $\searrow$ preserves linearity (it introduces no fresh names), $\Delta_1' \bullet \Delta_2$ is well-defined. Therefore, if $\tilde{s}[A, B] : \beta \in \Delta_2$ (the other case is trivial) then $\tilde{s}[A, B] : \alpha \mid \beta \in \Delta_1 \bullet \Delta_2$ and $\tilde{s}[A, B] : \alpha' \mid \beta \in \Delta_1' \bullet \Delta_2$. Finally, by definition of $\searrow$, we have that $\Delta \searrow \Delta_1' \bullet \Delta_2$.

— *Case* (STRUCT). It follows from point (1) of this theorem.

— *Case* (RES). In this case we have

$$\frac{(\sigma, I) \rightsquigarrow (\sigma', I')}{(\sigma, (\nu\tilde{s})I) \rightsquigarrow (\sigma', (\nu\tilde{s})I').}$$

There are three possible cases for typing restriction, but we only analyze rule (G-TRES$_1$) as the other cases are similar. We have $\Gamma \vdash (\nu s)I \rhd \Delta = \Delta_1 \cdot \tilde{s}_1\tilde{s}_2 : \bot$ whenever $\Gamma \vdash I \rhd \Delta_1 \cdot \tilde{s}_1 s\tilde{s}_2[A, B] : \alpha$. Now, as $(\sigma, I) \rightsquigarrow (\sigma', I')$, by induction hypothesis, we have that $\Gamma \vdash I' \rhd \Delta''$ and three possible cases:

(a) $\Delta'' = \Delta_1 \cdot \tilde{s}_1 s\tilde{s}_2[A, B] : \alpha'$ with $\alpha \searrow \alpha'$. If we now apply again rule (G-TRES$_1$), we get that $\Gamma \vdash (\nu s)I' \rhd \Delta_1, \tilde{s}_1\tilde{s}_2 : \bot$.

(b) $\Delta_1 = \Delta_2 \cdot \tilde{s}'[C, D] : \alpha'$ and $\Delta'' = \Delta_2 \cdot \tilde{s}'[C, D] : \alpha'' \cdot \tilde{s}_1 s\tilde{s}_2[A, B] : \alpha$ with $\alpha' \searrow \alpha''$. Now, applying again rule (G-TRES$_1$), we get that $\Gamma \vdash (\nu s)I' \rhd \Delta_2 \cdot \tilde{s}'[C, D] : \alpha'' \cdot \tilde{s}_1\tilde{s}_2 : \bot$.

(c) $\Delta'' = \Delta_1 \cdot \tilde{s}_1 s\tilde{s}_2[A, B] : \alpha$ and we trivially get $\Gamma \vdash (\nu s)I' \rhd \Delta_1 \cdot \tilde{s}_1\tilde{s}_2 : \bot$.

(3) It follows as a corollary of point (2). □

B. AUXILIARY DEFINITIONS AND PROOFS OF SECTION 4

B.1. Free Session Channels, Free Term Variables and Service Channels

The set of free session channels, denoted by fsc, is defined as follows.

Term	fsc
$!ch(\tilde{s}).\ P$, $\overline{ch}(\nu\tilde{s}).\ P$	$\mathsf{fsc}(P)\backslash\tilde{s}$
$s \rhd \Sigma_i \mathsf{op}_i(y_i).\ P_i$	$\{s\} \cup \left(\bigcup_i \mathsf{fsc}(P_i)\right)$
$\overline{s} \lhd \mathsf{op}\langle e\rangle.\ P$	$\{s\} \cup \mathsf{fsc}(P)$
$x := e.\ P$, $\mu X.\ P$	$\mathsf{fsc}(P)$
if e then P_1 else P_2, $P_1 \oplus P_2$, $P_1 \mid P_2$	$\mathsf{fsc}(P_1) \cup \mathsf{fsc}(P_2)$
$(\nu s)P$	$\mathsf{fsc}(P)\backslash\{s\}$
X, $\mathbf{0}$	$\emptyset$
$A[\ P\ ]_\sigma$	$\mathsf{fsc}(P)$
$N_1 \mid N_2$	$\mathsf{fsc}(N_1) \cup \mathsf{fsc}(N_2)$
$(\nu s)N$	$\mathsf{fsc}(N)\backslash\{s\}$
ϵ	$\emptyset$

The set of free term variables, denoted by fv, is defined as follows.

Term	fv
$!ch(\tilde{s}).\ P$, $\overline{ch}(\nu\tilde{s}).\ P$	$\mathsf{fv}(P)$
$\overline{s} \lhd \mathsf{op}\langle e\rangle.\ P$, $x := e.\ P$	$\mathsf{fv}(P)$
$(\nu s)P$	$\mathsf{fv}(P)$
$s \rhd \Sigma_i \mathsf{op}_i(y_i).\ P_i$	$\bigcup_i \mathsf{fv}(P_i)$
if e then P_1 else P_2, $P_1 \oplus P_2$, $P_1 \mid P_2$	$\mathsf{fv}(P_1) \cup \mathsf{fv}(P_2)$
$\mathbf{0}$	$\{\}$
X	$\{X\}$
$\mu X.\ P$	$\mathsf{fv}(P) \backslash \{X\}$
$A[\ P\ ]_\sigma$	$\mathsf{fv}(P)$
$N_1 \mid N_2$	$\mathsf{fv}(N_1) \cup \mathsf{fv}(N_2)$
$(\nu s)N$	$\mathsf{fv}(N)$
ϵ	$\{\}$

Finally, we define the set of service channels channels.

Term	channels
$!ch(\tilde{s}).\ P$, $\overline{ch}(\nu\tilde{s}).\ P$	$\{ch\} \cup \mathsf{channels}(P)$
$\overline{s} \lhd \mathsf{op}\langle e\rangle.\ P$, $x := e.\ P$, $(\nu s)P$, $\mu X.\ P$	$\mathsf{channels}(P)$
$s \rhd \Sigma_i \mathsf{op}_i(y_i).\ P_i$, if e then P_1 else P_2, $P_1 \oplus P_2$, $P_1 \mid P_2$	$\bigcup_i \mathsf{channels}(P_i)$
$\mathbf{0}$, X	$\{\}$
$A[\ P\ ]_\sigma$, $(\nu s)N$	$\mathsf{channels}(N)$
$N_1 \mid N_2$	$\mathsf{channels}(N_1) \cup \mathsf{channels}(N_2)$
ϵ	$\{\}$

B.2. Proof of Proposition 4.14 (Existence of Minimal Typing)

Let $\Gamma_0 \vdash M \rhd \Delta_0$ be the minimal typing of M. Then Γ_0 and Δ_0 are algorithmically calculable from M.

PROOF. Similarly to the global case, we provide a minimum typing system for constructing the minimal type. The rules, defining $\vdash^\star_A$ (for processes) and $\vdash^\star$ (for networks), are reported in Figures 13 and 14.

We briefly comment the various rules. In (E-MTINITIN), the expression $\alpha[\mathtt{end}/\tilde{\mathbf{t}}]$ denotes the result of substituting end for each (free, by bound name convention) occurrence of type variables from $\tilde{\mathbf{t}}$. In (E-MTPAR), $\asymp$ extends $\asymp$ to service environments and is such that an output type α and an input type β can be coherent in the following way:

$$\alpha \asymp \beta \qquad \Longleftrightarrow \qquad \alpha \preceq \overline{\beta}$$

$$(\text{E-MTInitIn})\ \frac{\Gamma \vdash^{\star}_{A} P \triangleright \tilde{s}@A{:}\alpha \quad \tilde{\mathbf{t}} \text{ exhaust type variables in } \alpha \quad \overline{ch} \notin \mathsf{dom}(\Gamma) \quad \mathsf{client}(\Gamma)}{\Gamma, ch{:}(\tilde{s})\alpha[\mathtt{end}/\tilde{\mathbf{t}}]@A \vdash^{\star}_{A} !ch(\tilde{s}).\ P \triangleright \emptyset}$$

$$(\text{E-MTInitOut1})\ \frac{\Gamma \vdash^{\star}_{A} P \triangleright \Delta \cdot \tilde{s}@A : \beta}{\Gamma, \overline{ch}{:}(\tilde{s})(\beta)@B \vdash^{\star}_{A} \overline{ch}(\boldsymbol{\nu}\tilde{s}).P \triangleright \Delta}$$

$$(\text{E-MTInitOut2})\ \frac{\Gamma, \overline{ch}{:}(\tilde{s})\alpha@B \vdash^{\star}_{A} P \triangleright \Delta \cdot \tilde{s}@A : \beta}{\Gamma, \overline{ch}{:}(\tilde{s})(\alpha \vee \beta)@B \vdash^{\star}_{A} \overline{ch}(\boldsymbol{\nu}\tilde{s}).P \triangleright \Delta}$$

$$(\text{E-MTBranch})\ \frac{s \in \tilde{s} \quad \Gamma \vdash x_j : \theta_j \quad \Gamma \vdash^{\star}_{A} P_j \triangleright \Delta \cdot \tilde{s}@A : \alpha_j \quad \text{for each } j \in I}{\Gamma \vdash^{\star}_{A} s \rhd \Sigma_{i\in I}\mathsf{op}_i(x_i).\ P_i \triangleright \Delta \cdot \tilde{s}@A : s \blacktriangleright \Sigma_{i\in I}\,\mathsf{op}_i(\theta_i).\ \alpha_i}$$

$$(\text{E-MTSel})\ \frac{\Gamma \vdash e : \theta_i \quad \Gamma \vdash^{\star}_{A} P \triangleright \Delta \cdot \tilde{s}@A : \alpha_j \quad j \in \{j\}}{\Gamma \vdash^{\star}_{A} \overline{s} \lhd \mathsf{op}_j\langle e\rangle.\ P \triangleright \Delta \cdot \tilde{s}@A : s \blacktriangleleft \Sigma_{i\in\{j\}}\,\mathsf{op}_i(\theta_i).\ \alpha_i}$$

$$(\text{E-MTAssign})\ \frac{\Gamma \vdash_{A} x : \theta \quad \Gamma \vdash e : \theta \quad \Gamma \vdash^{\star}_{A} P \triangleright \Delta}{\Gamma \vdash^{\star}_{A} x ::= e.P \triangleright \Delta}$$

$$(\text{E-MTIf})\ \frac{\Gamma_i \vdash e : \mathtt{bool} \quad \Gamma_i \vdash^{\star}_{A} P \triangleright \Delta_i\ (i = 1,2)}{\Gamma_1 \vee \Gamma_2 \vdash^{\star}_{A} \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta_1 \vee \Delta_2}$$

$$(\text{E-MTSum})\ \frac{\Gamma_i \vdash^{\star}_{A} P_i \triangleright \Delta_i\ (i = 1,2)}{\Gamma_1 \vee \Gamma_2 \vdash^{\star}_{A} P_1 \oplus P_2 \triangleright \Delta_1 \vee \Delta_2}$$

$$(\text{E-MTPar})\ \frac{\Gamma_1 \vdash^{\star}_{A} P \triangleright \Delta_1 \quad \Gamma_2 \vdash^{\star}_{A} Q \triangleright \Delta_2 \quad \Gamma_1 \eqslantless \Gamma_2 \quad \Delta_1 \eqslantless \Delta_2}{\Gamma_1 \circledcirc \Gamma_2 \vdash^{\star}_{A} P \mid Q \triangleright \Delta_1 \circledcirc \Delta_2}$$

$$(\text{E-MTRes1})\ \frac{\Gamma \vdash^{\star}_{A} P \triangleright \Delta \cdot \tilde{s}_1 s \tilde{s}_2 : \bot}{\Gamma \vdash^{\star}_{A} (\boldsymbol{\nu}s)P \triangleright \Delta \cdot \tilde{s}_1\tilde{s}_2 : \bot} \qquad (\text{E-MTRes2})\ \frac{\Gamma \vdash^{\star}_{A} P \triangleright \Delta \cdot \varepsilon : \bot}{\Gamma \vdash^{\star}_{A} P \triangleright \Delta}$$

$$(\text{E-MTVar})\ \frac{\psi = \{\tilde{s}_i\}}{X : \mathbf{t} \vdash^{\star}_{A} X \triangleright \cup_i \tilde{s}_i : \mathbf{t}} \qquad (\text{E-MTInact})\ \frac{\psi = \{\tilde{s}_i\}}{\emptyset \vdash^{\star}_{A} \mathbf{0} \triangleright \cup_i \tilde{s}_i : \mathtt{end}}$$

$$(\text{E-MTRec})\ \frac{\Gamma, X : \mathbf{t} \vdash^{\star}_{A} P \triangleright \Delta}{\Gamma \vdash^{\star}_{A} \mu X.P \triangleright \mathtt{solve}(\mathbf{t}, \Delta)}$$

Fig. 13. Minimal typing rules for end-point calculus: Processes.

(note this means α has more branches than β at each input point). Similarly for the service typing. Composition $\circledcirc$ at service typing always preserves the input side of the typing, that is, assuming $\alpha \eqslantless \beta$, we have

$$ch@A : (\tilde{s})\alpha \circledcirc \overline{ch}@A : (\tilde{s})\beta \stackrel{\text{def}}{=} ch@A : (\tilde{s})\alpha \qquad (\alpha \eqslantless \beta).$$

$$(\text{E-MTPARTICIPANT})\ \frac{\Gamma \vdash^{\star}_{A} P \triangleright \Delta \quad \Gamma \vdash \sigma@A}{\Gamma \vdash^{\star} A[\,P\,]_{\sigma} \triangleright \Delta} \qquad (\text{E-MTINACTNW})\ \frac{}{\Gamma \vdash^{\star} \epsilon \triangleright \emptyset}$$

$$(\text{E-MTRESNW1})\ \frac{\Gamma \vdash^{\star} M \triangleright \Delta \cdot \tilde{s}_1 s \tilde{s}_2 : \bot}{\Gamma \vdash^{\star} (\nu s)M \triangleright \Delta \cdot \tilde{s}_1 \tilde{s}_2 : \bot} \qquad (\text{E-MTRESNW2})\ \frac{\Gamma \vdash^{\star} M \triangleright \Delta \cdot \varepsilon : \bot}{\Gamma \vdash^{\star} M \triangleright \Delta}$$

$$(\text{E-MTPARNW})\ \frac{\Gamma_i \vdash^{\star} N_i \triangleright \Delta_i \quad \Gamma_1 \Cup \Gamma_2 \quad \Delta_1 \Cup \Delta_2}{\Gamma_1 \circledcirc \Gamma_2 \vdash^{\star} N_1 \mid N_2 \triangleright \Delta_1 \circledcirc \Delta_2}$$

Fig. 14. Minimal typing rules for end-point calculus: Networks.

In (E-MTREC), as we discussed for the global case, the expression $\texttt{solve}(\mathbf{t}, \Delta)$ denotes the result of calculating the proper recursive type (or appropriate substitution), which follows Dezani-Ciancaglini et al. [2006]. The expression returns, for each component $\tilde{s}_i@A : \alpha_i$ of Δ, either $\tilde{s}_i@A : \mu\mathbf{t}.\alpha_i$ when $\mathbf{t}$ does exist in α_i or, if not, $\tilde{s}_i@A : \alpha_i$. We can prove by induction on the typing rules that the given system implies minimal typing. In particular, we can compare each rule with the corresponding one in Figure 7 and Figure 8. □

C. AUXILIARY DEFINITIONS AND PROOFS FOR CONNECTEDNESS

C.1. Proof of Theorem 5.5 (Subject Reduction for Connectednes s)

Let I be strongly connected and σ be well-typed. Then $(\sigma, I) \rightsquigarrow (\sigma', I')$ implies I' is strongly connected.

PROOF. By induction on the reduction rules.

— (INIT). In this case we have that $(\sigma, A \to B : ch(\nu\,\tilde{s})\mathbf{.}\ I') \rightsquigarrow (\sigma, (\nu\tilde{s})I'')$ 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 $(\sigma, A \to B : s\langle \mathsf{op},\ e,\ x\rangle\mathbf{.}\ I) \rightsquigarrow (\sigma', I)$ if and only if $\sigma \vdash e@A \Downarrow v$. By definition of strong connectedness I is strongly connected.
— (ASSIGN). This rule states that $(\sigma, x@A := e\mathbf{.}\ I') \rightsquigarrow (\sigma', I')$. By definition of strong connectedness we have that I' is strongly connected.
— (IFTRUE) and (IFFALSE). We have that $(\sigma, \mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2) \rightsquigarrow (\sigma, I)$ and by definition of strong connectedness we have that $I = I_i$ is strongly connected.
— (PAR). We have $(\sigma, I_1 \mid I_2) \rightsquigarrow (\sigma', I_1' \mid I_2)$ inferred from $(\sigma, I_1) \rightsquigarrow (\sigma', I_1')$. 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 since it is top-level.
— (RES). Immediate from induction hypothesis and definition of strong connectedness, as before.
— (REC). We have $(\sigma, \mu X.I) \rightsquigarrow I'$ from $(\sigma, I[(\mu X.I)/X] \rightsquigarrow I'$. By Lemma 5.3, $I[(\mu X.I)/X]$ is strongly connected. By induction hypothesis so is I', done.
— (STRUCT). By Lemma 5.4. □

C.2. Variants of Connectedness.

C.2.1. r-Strong Connectedness. The strong connectedness (or simply connectedness) seen in Section 5.2 is robust with respect to asynchrony of messages, that is, 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:

$$\begin{array}{l}\mathsf{Buyer} \to \mathsf{Seller} : \mathsf{QuoteCh}(\nu\, \mathsf{s}).\\ \mathsf{Buyer} \to \mathsf{Seller} : s\langle \mathsf{RequestQuote},\ \mathit{productName},\ x\rangle.\\ \mathsf{Seller} \to \mathsf{Buyer} : s\langle \mathsf{ReplyQuote},\ \mathit{productPrice},\ y\rangle.\mathbf{0}.\end{array} \tag{4}$$

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 requests a quote with a product name. The Seller then replies with the corresponding product price.[5]

Sending multiple consecutive messages from one party to another in a session is often found in practice (in both business and security protocols). Further (4) may not violate the essential idea of strong connectedness both logically and in implementation: first, it is still a reception of a message that 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 that 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 that 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 for reference.

Definition C.1. 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 C.2 (*r-strong connectedness*). The set of *r-strong connected interactions* are inductively generated as follows.

(1) $A \to B : ch(\nu\, \tilde{s})\mathbf{.}\ I'$ 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.
(2) $A \to B : s\langle \mathsf{op},\ e,\ x\rangle\mathbf{.}\ I$ 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 5.2, replacing "strong connectedness" with "r-strong connectedness."

One may note all relative strong connected interactions can be encoded into strong connected interactions. For example, (5) can be translated into:

$$\begin{array}{l}\mathsf{Buyer} \to \mathsf{Seller} : \mathsf{QuoteCh}(\nu\, \mathsf{s}).\\ \mathsf{Seller} \to \mathsf{Buyer} : s\langle \mathsf{Ack}\rangle.\\ \mathsf{Buyer} \to \mathsf{Seller} : s\langle \mathsf{RequestQuote},\ \mathit{productName},\ x\rangle.\\ \mathsf{Seller} \to \mathsf{Buyer} : s\langle \mathsf{ReplyQuote},\ \mathit{productPrice},\ y\rangle.\mathbf{0}.\end{array} \tag{5}$$

Thus we only have to add one ack between two consecutive actions in the same directions. For this reason, in all technical developments that 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.

[5]In practice, one may also describe the initial "session initiation" action and the first RequestQuote action as one action, as in WS-CDL and consider (4) as a representation of this idiom in a formal setting.

C.2.2. Weak Connectedness. We can further loosen relative strong connectedness. For one thing, one may consider the following description is a natural one.

$$\begin{array}{l}
\mathsf{Broker} \to \mathsf{Seller} : \mathsf{SellerCh}(\boldsymbol{\nu}\, \mathsf{s}).\\
\mathsf{Broker} \to \mathsf{Buyer} : \mathsf{BuyerCh}(\boldsymbol{\nu}\, (\mathsf{f}', \mathsf{s}')).\\
\mathsf{Broker} \to \mathsf{Seller} : s\langle \mathsf{RequestQuote},\ \mathit{productName},\ x\rangle.\\
\mathsf{Broker} \to \mathsf{Buyer} : s'\langle \mathsf{RequestQuote},\ \mathit{productName},\ y\rangle.\\
\mathsf{Seller} \to \mathsf{Broker} : s\langle \mathsf{ReplyQuote},\ \mathit{productPrice},\ z\rangle.....
\end{array} \tag{6}$$

Here Broker does four consecutive actions that are targeted to two different participants. Further this global description specifies, in the fifth line, that the Seller replies to the Broker 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 realizable in synchronous communication. It is also easy to realize 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 call *weak connectedness*, can be formalized 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. Weakly 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 behavior. This means, among others, weakly connected but not r-strong connected descriptions are in general not well-threaded. 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.

D. AUXILIARY DEFINITIONS AND PROOFS FOR WELL-THREADEDNESS

D.1. Proof of Theorem 5.12 (Typing Soundness for Well-Threadedness)

A connected annotated interaction $\mathcal{E}$ is well-threaded if $\Theta \vdash \mathcal{E}$ for some Θ.

PROOF. Henceforth, (IH) stands for induction hypothesis.

We prove the stronger result, where we say a thread in $\mathcal{A}$ is *complete* if it occurs as a passive thread in session initialization. It is *incomplete* if not.

Now, 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 *(CC)* (see Definition 5.8) 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 *(CC)* being obvious, and (3) is again by (IH). □

$$(\text{T-MINIT})\ \frac{\Theta, \tau_2 : (B, S) \vdash^\star \mathcal{A} \quad \mathsf{top}^\circ(\mathcal{A}) = \tau_2 \quad S \subset \{\tilde{s}\} \quad \mathsf{com}(\tau_1 : (A, \{\tilde{s}\}),\ \Theta) \ =\ \tau_1 : (A, S'), \Theta\sigma \quad (\tau_1 \text{ fresh})}{\Theta\sigma, \tau_1 : (A, S' \setminus \{\tilde{s}\}), \tau_2 : (B, \emptyset) \vdash^\star A^{\tau_1} \to B^{\tau_2} : ch(\boldsymbol{\nu}\, \tilde{s}).\ (\mathcal{A}\sigma)}$$

$$(\text{T-MCOMM})\ \frac{\Theta, \tau_2 : (B, S) \vdash^\star \mathcal{A} \quad \mathsf{top}^\circ(\mathcal{A}) = \tau_2 \quad \mathsf{com}(\tau_1 : (A, \{s\}),\ \Theta) \ =\ \tau_1 : (A, S'), \Theta'\sigma \quad (\tau_1 \text{ fresh})}{\tau_1 : (A, S'), \tau_2 : (B, S), \Theta'\sigma \vdash^\star A^{\tau_1} \to B^{\tau_2} : s\langle \mathsf{op},\ e,\ x \rangle.\ (\mathcal{A}\sigma)}$$

$$(\text{T-MASSIGN})\ \frac{\Theta \vdash^\star \mathcal{A} \quad \mathsf{top}^\circ(\mathcal{A}) = \tau}{\Theta \vdash^\star x@A^\tau := e.\ \mathcal{A}}$$

$$(\text{T-MIF})\ \frac{\Theta_i \vdash^\star \mathcal{A}_i \quad \mathsf{top}^\circ(\mathcal{A}_i) = \tau \quad \mathsf{com}(\Theta_1,\ \Theta_2) \ =\ (\Theta_1 \cup \Theta_2)\sigma}{(\Theta_1 \cup \Theta_2)\sigma \vdash^\star \mathsf{if}\ e@A^\tau\ \mathsf{then}\ \mathcal{A}_1\sigma\ \mathsf{else}\ \mathcal{A}_2\sigma}$$

$$(\text{T-MSUM})\ \frac{\tau : (A, S_i), \Theta_i \vdash^\star \mathcal{A}_i \quad \mathsf{top}^\circ(\mathcal{A}_i) = \tau \quad \mathsf{dom}(\Theta_1) \cap \mathsf{dom}(\Theta_2) = \emptyset \quad \mathsf{com}(\tau : (A, S_1 \cup S_2),\ \Theta_1 \cup \Theta_2) = \tau : (A, S'), (\Theta_1 \cup \Theta_2)\sigma}{\tau : (A, S'),\ (\Theta_1 \cup \Theta_2)\sigma\ \vdash^\star\ (\mathcal{A}_1 +^\tau \mathcal{A}_2)\sigma}$$

$$(\text{T-MPAR})\ \frac{\tau : (A, S_i), \Theta_i \vdash^\star \mathcal{A}_i \quad \mathsf{top}^\circ(\mathcal{A}_i) = \tau \quad \mathsf{dom}(\Theta_1) \cap \mathsf{dom}(\Theta_2) = \emptyset \quad \mathsf{com}(\tau : (A, S_1 \cup S_2),\ \Theta_1 \cup \Theta_2) = \tau : (A, S'), (\Theta_1 \cup \Theta_2)\sigma}{\tau : (A, S'), (\Theta_1 \cup \Theta_2)\sigma \vdash^\star (\mathcal{A}_1 \mid^\tau \mathcal{A}_2)\sigma}$$

$$(\text{T-MPARE})\ \frac{\Theta_i \vdash^\star \mathcal{E}_i \quad \mathsf{dom}(\Theta_1) \cap \mathsf{dom}(\Theta_2) = \emptyset}{\Theta_1 \cup \Theta_2 \vdash^\star \mathcal{E}_1 \mid \mathcal{E}_2}$$

$$(\text{T-MRESE})\ \frac{\Theta \vdash^\star \mathcal{E}}{\Theta/s \vdash^\star \mathcal{E}} \qquad (\text{T-MZERO})\ \frac{-}{\tau : (A, \emptyset) \vdash^\star \mathbf{0}^A_\tau}$$

$$(\text{T-MVAR})\ \frac{-}{\tau : (A, \emptyset) \vdash^\star X^A_\tau} \qquad (\text{T-MREC})\ \frac{\Theta \vdash^\star \mathcal{A} \quad \mathsf{top}^\circ(\mathcal{A}) = \tau}{\Theta \vdash^\star \mu^\tau X^A.\mathcal{A}}$$

Fig. 15. Typing rules for inferring minimal consistent annotations.

D.2. Proof of Proposition 5.15 (Existence of Minimal Consistent Annotation)

Let I be strongly connected. Then, I has a minimal consistent annotation if and only if I is well-threaded.

PROOF. We will define the typing system for inferring minimal consistent annotations. We use annotated inaction for convenience, this time with a participant name too, writing $\mathbf{0}^A_\tau$. The sequent has the shape $\Theta \vdash^\star \mathcal{A}$ 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 following sense.

(1) Θ induces a finite map from threads to such pairs, in the obvious sense. In this case we write, for instance, $\Theta(\tau) = (A, S)$ 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 $A \neq A'$.

The typing rules, given in Figure 15, automatically ensure the well-formedness property. These rules refine the rules in Figure 10 and inductively solve simple equational constraints. These constraints are distilled in the operator $\mathsf{com}(\Theta_1,\ \Theta_2)$ used in these rules. This operator is defined as follows.

Definition D.1 (*Collapsing Operator*). Given well-formed Θ_1 and Θ_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.
— $\mathsf{com}(\tau : (A, S), \Theta)$; 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 this procedure 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*, $\mathsf{com}(\tau : (A, S), \Theta)$ finds the minimum $\{\tau_i\}$ from $\mathsf{dom}(\Theta)$ such that each τ_i is assigned (A, T_i) where T_i intersects nontrivially 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 commonness is taken hereditarily so that all threads of a certain participant that are assigned transitively common session channels are eventually collapsed. We are using the following notation in the rules.

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 15 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 15, *assuming it is defined*. Except for choosing fresh thread labels in the base cases as well as in (T-MINIT) and (T-MCOMM), each induction automatically generates a unique annotated term if ever, so that the resulting term (if any) is also unique.

We can now prove the two directions of the proposition. For ($\Rightarrow$), we need to show by induction on $\mathcal{A}$ that typability in Figure 15 means typability in Figure 10 with the corresponding typings modulo vacuous lacking paired type assignments needed for the well-formedness for the original typing system. And for ($\Leftarrow$) we can just check that satisfaction of the conditions for the consistency means typability in Figure 15. □

E. PROOFS FOR COHERENCE

E.1. Proofs of Lemma 5.22 (Substitution Lemma for Coherence)

If $\mu X.I$ is coherent then so is $I[(\mu X.I)/X]$.

PROOF. We consider a consistent annotation of $\mu X.I$, which we write

$$\mathcal{A}' \stackrel{\text{def}}{=} \mu X^{A}_{\tau:\tilde{\tau}}.\mathcal{A}.$$

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 $\mathsf{TP}(\mathcal{A}', \tau_i) = \mu X.P$ then by induction we can check

$$\mathsf{TP}(\mathcal{A}'', \tau_i) \quad = \quad P[\mu X.P/X].$$

including its definedness. In both (1) and (2), this thread is not to be merged with any other threads, hence done. On the other hand, for τ' that is not in $\tilde{\tau}$ but occurs in I, this should be a passive complete thread inside I. Note the annotation $\tau : \tilde{\tau}$ in $\mu X^{A}_{\tau:\tilde{\tau}}.\mathcal{A}$ 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 $\mathcal{A}'$ that should be merged with τ', hence mergeability does not change, as required. □

E.2. Proofs of Proposition 5.23 (Subject Congruence for Coherence)

(1) $\sqcup$ is partially symmetric and associative, and has the identity $\mathbf{0}$, all up to $\equiv$, that is, $P \sqcup Q$ 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 5.17 as well as Definition 5.16, 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 $|$, that is,

$$(\mathcal{A}_1|\mathcal{A}_2)|\mathcal{A}_3 \;\equiv\; \mathcal{A}_1|(\mathcal{A}_2|\mathcal{A}_3).$$

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 5.16, we can first apply $\equiv$ to P and Q using the last rule in Definition 5.16, 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, for instance, $\mathsf{threads}(I, ch) = \{\tau_1, \tau_2\}$ then $\mathsf{TP}(I, \tau_1) \bowtie \mathsf{TP}(I, \tau_2)$ iff $\mathsf{TP}(I', \tau_1) \bowtie \mathsf{TP}(I', \tau_2)$. □

E.3. Proofs of Theorem 5.24 (Subject Reduction for Coherence)

If $\mathcal{A}$ is coherent and $(\sigma, \mathcal{A}) \rightsquigarrow (\sigma', \mathcal{A}')$, 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 $\mathscr{A}$ and $\mathscr{A}'$, respectively. By abuse of designation we say that these sets are coherent. We then show the following claim by rule induction on reduction rules.

If $\mathscr{A}$ is coherent, then $\mathscr{A}'$ is also coherent. Moreover $\mathscr{A}'$ is either the result of truncating some of threads from $\mathscr{A}$, as well as adding zero or more collections of the fresh complete threads, each collection being multiple copies of a complete thread existing in $\mathscr{A}$.

So far, the copies can be created due to recursion. Assume $\mathscr{A}$ from $\mathcal{A}$ is coherent. For (G-INIT), we first take off, from $\mathscr{A}$, 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 $\mathscr{A}$. 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 $\mathscr{A}'$. If τ_1 demands intrathread merging through its own thread projection, then so does it originally in $\mathscr{A}$. Otherwise merging in $\mathscr{A}'$ remains the same as $\mathscr{A}$, hence done. For (G-COM), we truncate two noncomplete threads, one active and the other passive, from $\mathscr{A}$. 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 interthread merging, so we again know $\mathscr{A}'$ is coherent. (G-ASSIGN) truncates a single active thread in $\mathscr{A}$, while both (G-IFT/G-IFF) and (G-SUM) simply cut off some of the threads from $\mathscr{A}$ to make $\mathscr{A}'$. For (G-PAR), assume $\mathcal{A} = \mathcal{A}_1|\mathcal{A}_2$. Accordingly we can set $\mathscr{A} = \mathscr{A}_1 \cup \mathscr{A}_2$ where $\mathscr{A}_1$ and $\mathscr{A}_2$ respectively come from $\mathcal{A}_1$ and $\mathcal{A}_2$. Now assume $\mathcal{A}_1$ reduces to $\mathcal{A}_1'$, with the new threads $\mathscr{A}_1'$. By assumption, we know $\mathscr{A}_1'$ may truncate some threads and may add copies of existing threads with fresh thread numbers. Since neither affect mergeability we are done (note intrathread mergeability remains the same by the definition of the projection of a top-level parallel composition). (G-RES) does not change threads (i.e., $\mathscr{A} = \mathscr{A}'$). (G-REC) is by (IH) and by Lemma 5.22, possibly adding collections of fresh copies of complete threads. Finally (G-STRUCT) is immediate from Proposition 5.23. □

F. AUXILIARY DEFINITIONS AND PROOFS FOR THE END-POINT PROJECTION

F.1. Proofs of Proposition 5.27 (Invariance under Annotations)

Given a coherent interaction I and a state σ such that $\Gamma \vdash I \triangleright \emptyset$ and $\Gamma \vdash \sigma$, let $\mathcal{A}_1$ and $\mathcal{A}_2$ be consistent annotations of I. Then, $\mathsf{EPP}(\mathcal{A}_1, \sigma) \equiv \mathsf{EPP}(\mathcal{A}_2, \sigma)$.

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 5.15. Let $\mathcal{A}'$ be a possibly different consistent annotation of I. By the definition of minimality (cf. Definition 5.14) and by construction, there is a surjective map σ from the threads in $\mathcal{A}$ 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 $\sigma(\tau_i) = \tau$. Note that τ cannot be a complete thread (from Appendix D.1, a thread is *complete* if it occurs as a passive thread in session initialization while it is *incomplete* if not): if that was the case, 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, each τ_i is an active thread starting from a session initialization. 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

$$\text{(G-Init)}\ \frac{}{(\sigma,\ A\to B : ch(\boldsymbol{\nu}\,\tilde{s}).\ I) \overset{A\to B:ch(\tilde{s})}{\rightsquigarrow} (\sigma,\ (\boldsymbol{\nu}\tilde{s})I)} \qquad \text{(G-Assign)}\ \frac{\sigma \vdash e@A \Downarrow v \quad \sigma' = \sigma[x@A \mapsto v]}{(\sigma,\ x@A := e.\ I) \overset{\tau}{\rightsquigarrow} (\sigma',\ I)}$$

$$\text{(G-Com)}\ \frac{\sigma \vdash e@A \Downarrow v \quad \sigma' = \sigma[x@B \mapsto v]}{(\sigma,\ A\to B : s\langle \mathsf{op},\ e,\ x\rangle.\ I) \overset{A\to B:\mathsf{op}\langle v\rangle}{\rightsquigarrow} (\sigma',\ I)} \qquad \text{(G-Sum)}\ \frac{}{(\sigma,\ I_1 + I_2) \overset{\tau}{\rightsquigarrow} (\sigma,\ I_1)}$$

$$\text{(G-IfT)}\ \frac{\sigma \vdash e@A \Downarrow \mathsf{tt}}{(\sigma,\ \mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2) \overset{\tau}{\rightsquigarrow} (\sigma, I_1)} \qquad \text{(G-Rec)}\ \frac{(\sigma,\ I[\mu X.I/X]) \overset{\ell}{\rightsquigarrow} (\sigma',\ I')}{(\sigma,\ \mu X.I) \overset{\ell}{\rightsquigarrow} (\sigma',\ I')}$$

$$\text{(G-IfF)}\ \frac{\sigma \vdash e@A \Downarrow \mathsf{ff}}{(\sigma,\ \mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2) \overset{\tau}{\rightsquigarrow} (\sigma,\ I_2)} \qquad \text{(G-Res)}\ \frac{(\sigma,\ I) \overset{\ell}{\rightsquigarrow} (\sigma',\ I')}{(\sigma, (\boldsymbol{\nu}\tilde{s})I) \overset{\ell}{\rightsquigarrow} (\sigma',\ (\boldsymbol{\nu}\tilde{s})I')}$$

$$\text{(G-Struct)}\ \frac{I \equiv I''\ \ (\sigma,\ I) \overset{\ell}{\rightsquigarrow} (\sigma',\ I')\ \ I' \equiv I'''}{(\sigma,\ I'') \overset{\ell}{\rightsquigarrow} (\sigma',\ I''')} \qquad \text{(G-Par)}\ \frac{(\sigma,\ I_1) \overset{\ell}{\rightsquigarrow} (\sigma',\ I_1')}{(\sigma, I_1 \mid I_2) \overset{\ell}{\rightsquigarrow} (\sigma',\ I_1' \mid I_2)}$$

Fig. 16. Annotated reduction relation for the global calculus.

initial active thread of a prime interaction (an interaction is *prime* if it cannot be decomposed into a nontrivial 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:

$$\mathsf{TP}(\mathcal{A}', \tau) \overset{\text{def}}{=} \Pi_i \mathsf{TP}(\mathcal{A}'_i, \tau) \quad \equiv \quad \Pi_{\sigma(\tau_i)=\tau} \mathsf{TP}(\mathcal{A}, \tau_i);$$

that is, the projection of $\mathcal{A}'$ onto τ and the projections of $\mathcal{A}$ onto $\{\tau_i\}$ coincide. □

F.2. Annotated Reduction (1) Global Calculus

The annotated reduction relation for the global calculus $\overset{\ell}{\rightsquigarrow}$ is given in Figure 16. Rules are identical to the ones reported in Figure 2 apart from the label ℓ annotating the relation $\rightsquigarrow$. Labels are used for *observing* actions performed in a global descriptions. In particular, we observe session initialization and communication. All other operations, for instance, if-then-else or assignment, are just observed as τ (not to be confused with thread names). Note that, unlike the standard π-calculus semantics, restriction is not hiding session channels from labels in our strict reduction (cf. Section 5.6.2).

F.3. Annotated Reduction (2) Endpoint Calculus

The annotated reduction relation for the endpoint calculus is given in Figure 17. Similarly to the global case, we provide a strict annotated reduction for the end-point calculus. Labels are identical to the ones in Figure 16.

F.4. Strict Reduction

In a strict reduction, we only α-convert names when they are under a prefix. In particular, when a term has the form $(\nu\tilde{s})P$, in reduction, $\tilde{s}$ is never renamed. For example, the following is not strict.

$$\begin{aligned}&(\boldsymbol{\nu}s)(A[\![\ s \rhd \Sigma_i \mathsf{op}_i(x_i).\ P_i\ ]\!]_\sigma \mid B[\![\ \bar{s} \lhd \mathsf{op}_j\langle e\rangle.\ Q \mid Q'\ ]\!]_{\sigma'})\\ &\rightsquigarrow\\ &(\boldsymbol{\nu}s')(A[\![\ P_j[s'/s]\ ]\!]_{\sigma[x_j \mapsto v]} \mid B[\![\ Q[s'/s]\ ]\!]_{\sigma'}),\end{aligned}$$

$$(\text{E-Init})\quad \frac{s_i \notin \mathsf{fsc}(P') \cup \mathsf{fsc}(Q')}{A[\,!\,ch(\tilde{s}).\ P \mid P'\,]_\sigma \mid B[\,\overline{ch}(\boldsymbol{\nu}\tilde{s}).\ Q \mid Q'\,]_{\sigma'} \overset{A\to B:ch(\tilde{s})}{\rightsquigarrow} (\boldsymbol{\nu}\tilde{s})(A[\,!\,ch(\tilde{s}).\ P \mid P \mid P'\,]_\sigma \mid B[\,Q \mid Q'\,]_{\sigma'})}$$

$$(\text{E-Com})\quad \frac{\sigma \vdash e \Downarrow v}{A[\,s \rhd \Sigma_i \mathsf{op}_i(x_i).\ P_i \mid P'\,]_\sigma \mid B[\,\overline{s} \lhd \mathsf{op}_j\langle e\rangle.\ Q \mid Q'\,]_{\sigma'} \overset{A\to B:s\rhd \mathsf{op}\langle v\rangle}{\rightsquigarrow} A[\,P_j \mid P'\,]_{\sigma[x_j\mapsto v]} \mid B[\,Q \mid Q'\,]_{\sigma'}}$$

$$(\text{E-IfT})\ \frac{\sigma \vdash e \Downarrow \mathsf{tt}}{A[\,\mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \mid P'\,]_\sigma \overset{\tau}{\rightsquigarrow} A[\,P_1 \mid P'\,]_\sigma} \qquad (\text{E-Par})\ \frac{M \overset{\ell}{\rightsquigarrow} M'}{M|N \overset{\ell}{\rightsquigarrow} M'|N}$$

$$(\text{E-IfF})\ \frac{\sigma \vdash e \Downarrow \mathsf{ff}}{A[\,\mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \mid P'\,]_\sigma \overset{\tau}{\rightsquigarrow} A[\,P_2 \mid P'\,]_\sigma} \qquad (\text{E-Sum})\ \frac{i \in \{1,2\}}{A[P_1 \oplus P_2|R]_\sigma \overset{\tau}{\rightsquigarrow} A[P_i|R]_\sigma}$$

$$(\text{E-Rec})\ \frac{A[\,P[\mu X.P/X] \mid Q\,]_\sigma \mid N \overset{\ell}{\rightsquigarrow} N'}{A[\,\mu X.P \mid Q\,]_\sigma \mid N \overset{\ell}{\rightsquigarrow} N'} \qquad (\text{E-Assign})\ \frac{\sigma \vdash e \Downarrow v}{A[\,x := e.\ P \mid P'\,]_\sigma \overset{\tau}{\rightsquigarrow} A[\,P \mid P'\,]_{\sigma[x\mapsto v]}}$$

$$(\text{E-Res})\ \frac{M \overset{\ell}{\rightsquigarrow} M'}{(\boldsymbol{\nu}s)M \overset{\ell}{\rightsquigarrow} (\boldsymbol{\nu}s)M'} \qquad (\text{E-Struct})\ \frac{M \equiv M' \qquad M' \overset{\ell}{\rightsquigarrow} N' \qquad N' \equiv N}{M \overset{\ell}{\rightsquigarrow} N}$$

Fig. 17. Annotated reduction relation for the end-point calculus.

but the following is:

$$(\boldsymbol{\nu}s)(A[\,s \rhd \Sigma_i \mathsf{op}_i(x_i).\ P_i\,]_\sigma \mid B[\,\overline{s} \lhd \mathsf{op}_j\langle e\rangle.\ Q \mid Q'\,]_{\sigma'}) \rightsquigarrow (\boldsymbol{\nu}s)(A[\,P_j\,]_{\sigma[x_j\mapsto v]} \mid B[\,Q\,]_{\sigma'}).$$

As noted in the main section, we can assume all reductions to be strict without loss of generality because the only case when we do need alpha conversion of ν-bound names is when we generate a new ν-hidden channel under a recursion.

F.5. Proof of Theorem 5.30: (1) Type Preservation

If $\Gamma \vdash \mathcal{A} \rhd \Delta$ is the minimal typing of $\mathcal{A}$, then $\Gamma \vdash \mathsf{EPP}(\mathcal{A}, \sigma) \rhd \bot(\Delta)$.

PROOF. The type preservation is proved in the following three steps:

(a) We introduce an inference system for a single thread of an annotated interaction, written $\Gamma_i \vdash^{\tau_i} \mathcal{A} \rhd \Delta_i$ and relate it to the original minimal global typing; that is, we show that $\Gamma \vdash_{\mathsf{min}} I \rhd \Delta$ (where $\mathcal{A}$ is the annotation of I) if and only if $\Gamma_i \vdash^{\tau_i} \mathcal{A} \rhd \Delta_i$ for $\Gamma = \sqcup\Gamma_i$ and $\Delta = \sqcup\Delta_i$ where $\{\tau_i\}$ exhausts the threads in $\mathcal{A}$.
(b) We then relate this per-thread global typing with the minimal typing of thread projections, establishing that $\Gamma_i \vdash^{\tau_i} I \rhd \Delta_i$ if and only if $\Gamma_i \vdash^\star \mathsf{TP}(I, \tau_i) \rhd \Delta_i'$ for each τ_i, where Δ' (composition of Δ_i') 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: $\sqcup_i \Gamma_i \vdash^\star \sqcup_i P_i \rhd \sqcup_i \Delta_i$ 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 F.1 (*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 3.16 for $\Subset$ (see Figure 4, page 12) except for replacing (INC-IN) with the following rule:

$$(\text{EQ-IN})\ \frac{\beta \ \approx\ \Sigma_{i\in J} s \blacktriangleright \mathsf{op}_i(\theta_i).\ \alpha_i' \quad \forall i \in J.\ \alpha_i \Subset \alpha_i'}{\Sigma_{i\in J} s \blacktriangleright \mathsf{op}_i(\theta_i).\ \alpha_i \ \Subset\ \beta}.$$

We call $\sqsubseteq$ *equi-input subtyping*.

PROPOSITION F.2 (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* α_1 *and* α_2 *have an upper bound w.r.t.* $\sqsubseteq$, *then their lub w.r.t* $\sqsubseteq$, $\Subset$ *and* $\preceq$ *exists and all these three lubs coincide.*

PROOF. (1) is standard and is essentially identical to the proof of Proposition 4.11. (2) is by noting, for both $\Subset$ and $\preceq$, that the only difference is in the input rule. For (3) the existence of a lub given an upper bound is as in Propositions 3.19 and 4.11. 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* α_1 and α_2 and the join of their output branches. Take an upper bound of α_1 and α_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 one of its threads, gives a minimal typing that is specific to that thread. The sequent of this system is written as:

$$\Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta,$$

where $\mathcal{A}$ is an (extended) annotated interaction, τ is one of its threads, Γ is as given for global session types and Δ is defined as

$$\Delta \ ::=\ \emptyset \ \mid\ \Delta,\ \tilde{s}[B]:\alpha \ \mid\ \Delta,\ \tilde{s}:\bot.$$

Here, $\tilde{s}[B] : \alpha$ associates participant information to session types so that we can directly reconstruct global types from a set of these types.

The rules for the typing system are given in Figure 18, where we omit the obvious inverse communication rules. In (GT-MTZERO) and (GT-MTVAR), ψ is the set of mutually disjoint vectors of session channels (as noted in Proposition 3.21, 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 the rule names (GT-MTINIT-S1/2-A/P) and (GT-MTINIT-O), S stands for "self" while O for "others", a for active and p for 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 Figure 12.

$$(\text{GT-MTInit-S1a})\ \frac{\Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta \cdot \tilde{s}[A] : \beta \quad \tau = \tau_1 \quad ch \notin dom(\Gamma) \quad \mathsf{tvar}(\beta) \subseteq \{\tilde{\mathbf{t}}\}}{\Gamma,\ \overline{ch}@B : (\tilde{s})(\beta[\mathsf{end}/\tilde{\mathbf{t}}]) \vdash^{\tau} A^{\tau_1} \to B^{\tau_2} : ch(\boldsymbol{\nu}\, \tilde{s}).\ \mathcal{A} \rhd \Delta}$$

$$(\text{GT-MTInit-S1p})\ \frac{\Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta \cdot \tilde{s}[B] : \beta \quad \tau = \tau_2 \quad ch \notin dom(\Gamma) \quad \mathsf{tvar}(\beta) \subseteq \{\tilde{\mathbf{t}}\}}{\Gamma,\ ch@B : (\tilde{s})(\beta[\mathsf{end}/\tilde{\mathbf{t}}]) \vdash^{\tau} A^{\tau_1} \to B^{\tau_2} : ch(\boldsymbol{\nu}\, \tilde{s}).\ \mathcal{A} \rhd \Delta}$$

$$(\text{GT-MTInit-S2a})\ \frac{\Gamma,\ \overline{ch}@B : (\tilde{s})\alpha \vdash^{\tau} \mathcal{A} \rhd \Delta \cdot \tilde{s}[A] : \beta \quad \tau = \tau_1 \quad ch \notin dom(\Gamma) \quad \mathsf{tvar}(\beta) \subseteq \{\tilde{\mathbf{t}}\}}{\Gamma,\ \overline{ch}@B : (\tilde{s})(\alpha \bigtriangledown \beta[\mathsf{end}/\tilde{\mathbf{t}}]) \vdash^{\tau} A^{\tau_1} \to B^{\tau_2} : ch(\boldsymbol{\nu}\, \tilde{s}).\ \mathcal{A} \rhd \Delta}$$

$$(\text{GT-MTInit-S2p})\ \frac{\Gamma,\ ch@B : (\tilde{s})\alpha \vdash^{\tau} \mathcal{A} \rhd \Delta \cdot \tilde{s}[B] : \beta \quad \tau = \tau_2 \quad ch \notin dom(\Gamma) \quad \mathsf{tvar}(\beta) \subseteq \{\tilde{\mathbf{t}}\}}{\Gamma,\ ch@B : (\tilde{s})(\alpha \bigtriangledown \beta[\mathsf{end}/\tilde{\mathbf{t}}]) \vdash^{\tau} A^{\tau_1} \to B^{\tau_2} : ch(\boldsymbol{\nu}\, \tilde{s}).\ \mathcal{A} \rhd \Delta}$$

$$(\text{GT-MTInit-O})\ \frac{\Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta \quad \{\tilde{s}\} \cap \mathsf{fsc}(\Delta) = \emptyset \quad \tau \notin \{\tau_1, \tau_2\}}{\Gamma \vdash^{\tau} A^{\tau_1} \to B^{\tau_1} : ch(\boldsymbol{\nu}\, \tilde{s}).\ \mathcal{A} \rhd \Delta}$$

$$(\text{GT-MTComm-Sa})\ \frac{\Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta \cdot \tilde{s}\,[A] : \alpha_j \quad \Gamma \vdash e@A : \theta_j \quad \Gamma \vdash x@B : \theta_j \quad s \in \{\tilde{s}\} \quad \tau = \tau_1}{\Gamma \vdash^{\tau} A^{\tau_1} \to B^{\tau_2} : s\langle \mathsf{op}_j,\ e,\ x\rangle.\ \mathcal{A} \rhd \Delta \cdot \tilde{s}\,[A] : s \blacktriangleleft \mathsf{op}_j(\theta_j).\ \alpha_j}$$

$$(\text{GT-MTComm-Sp})\ \frac{\Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta \cdot \tilde{s}\,[B] : \alpha_j \quad \Gamma \vdash e@A : \theta_j \quad \Gamma \vdash x@B : \theta_j \quad s \in \{\tilde{s}\} \quad \tau = \tau_2}{\Gamma \vdash^{\tau} A^{\tau_1} \to B^{\tau_2} : s\langle \mathsf{op}_j,\ e,\ x\rangle.\ \mathcal{A} \rhd \Delta \cdot \tilde{s}\,[B] : s \blacktriangleright \mathsf{op}_j(\theta_j).\ \alpha_j}$$

$$(\text{GT-MTComm-O})\ \frac{\Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta \quad \Gamma \vdash e@A : \theta_j \quad \Gamma \vdash x@B : \theta_j \quad s \notin \mathsf{fsc}(\Delta) \quad \tau \notin \{\tau_1, \tau_2\}}{\Gamma \vdash^{\tau} A^{\tau_1} \to B^{\tau_2} : s\langle \mathsf{op}_j,\ e,\ x\rangle.\ \mathcal{A} \rhd \Delta}$$

$$(\text{GT-MTAssign})\ \frac{\Gamma \vdash x@A : \theta \quad \Gamma \vdash e@A : \theta \quad \Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta}{\Gamma \vdash^{\tau} x^{\tau'} := e@A.\ \mathcal{A} \rhd \Delta}$$

$$(\text{GT-MTSum})\ \frac{\Gamma_1 \vdash^{\tau} \mathcal{A}_1 \rhd \Delta_1 \quad \Gamma_2 \vdash^{\tau} \mathcal{A}_2 \rhd \Delta_2}{\Gamma_1 \bigtriangledown \Gamma_2 \vdash^{\tau} \mathcal{A}_1 +^{\tau'} \mathcal{A}_2 \rhd \Delta_1 \bigtriangledown \Delta_2}$$

$$(\text{GT-MTIf})\ \frac{\Gamma \vdash e@A : \mathtt{bool} \quad \Gamma_1 \vdash^{\tau} \mathcal{A}_1 \rhd \Delta_1 \quad \Gamma_2 \vdash^{\tau} \mathcal{A}_2 \rhd \Delta_2}{\Gamma_1 \bigtriangledown \Gamma_2 \vdash^{\tau} \mathsf{if}\ e^{\tau'}@A\ \mathsf{then}\ \mathcal{A}_1\ \mathsf{else}\ \mathcal{A}_2 \rhd \Delta_1 \bigtriangledown \Delta_2}$$

$$(\text{GT-MTPar})\ \frac{\Gamma_1 \vdash^{\tau} \mathcal{A}_1 \rhd \Delta_1 \quad \Gamma_2 \vdash^{\tau} \mathcal{A}_2 \rhd \Delta_2 \quad \mathsf{fsc}(\Delta_1) \cap \mathsf{fsc}(\Delta_2) = \emptyset}{\Gamma_1 \bigtriangledown \Gamma_2 \vdash^{\tau} \mathcal{A}_1 \mid^{\tau'} \mathcal{A}_2 \rhd \Delta_1 \bullet \Delta_2}$$

$$(\text{GT-MTRes1})\ \frac{\Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta \cdot \tilde{s}_1 s \tilde{s}_2[A, B] : \alpha}{\Gamma \vdash^{\tau} (\boldsymbol{\nu} s)\mathcal{A} \rhd \Delta \cdot \tilde{s}_1 \tilde{s}_2 : \bot} \qquad (\text{GT-MTRes2})\ \frac{\Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta \cdot \tilde{s}_1 s \tilde{s}_2 : \bot}{\Gamma \vdash^{\tau} (\boldsymbol{\nu} s)\mathcal{A} \rhd \Delta \cdot \tilde{s}_1 \tilde{s}_2 : \bot}$$

$$(\text{GT-MTRes3})\ \frac{\Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta \cdot \varepsilon : \bot}{\Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta} \qquad (\text{GT-MTVar})\ \frac{\Psi = \{\tilde{s}_i\} \quad \mathbf{t}\ \mathsf{fresh}}{X^A : \mathbf{t} \vdash^{\tau} X^A \rhd \bigcup_i \tilde{s}_i[A_i, B_i] : \mathbf{t}}$$

$$(\text{GT-MTRec})\ \frac{\Gamma \cdot X^A : \mathbf{t} \vdash^{\tau} \mathcal{A} \rhd \Delta}{\Gamma \vdash^{\tau} \mu X^A.\ \mathcal{A} \rhd \mathtt{solve}(\mathbf{t}, \Delta)}$$

$$(\text{GT-MTZero})\ \frac{\Psi = \{\tilde{s}_i\}}{\emptyset \vdash^{\tau} \mathbf{0} \rhd \bigcup_i \tilde{s}_i[A_i, B_i]\mathtt{end}}$$

Fig. 18. Threaded minimal typing rules for global calculus.

In the following, when we write for instance, $\Gamma \vdash_{\mathsf{min}} \mathcal{A} \rhd \Delta$, we assume that the thread annotation in $\mathcal{A}$ conforms to the grouping of session names in Δ (cf. Proposition A.1). The calculation $\sqcup_i \Delta_i$ is done in the obvious way, for instance, if we merge $\tilde{s}[B] : \alpha$ and $\tilde{s}[A] : \beta$ then we get $\tilde{s}[A, B] : \overline{\alpha} \sqcup \beta$.

PROPOSITION F.3 (THREADED TYPING FOR INTERACTION). *Let* $\{\tau_i\}$ *exhaust the threads in* $\mathcal{A}$. *Then,* $\Gamma \vdash_{\mathsf{min}} \mathcal{A} \triangleright \Delta$ *implies that there exist* $\{\Gamma_i\}$ *and* $\{\Delta_i\}$ *such that* $\Gamma = \sqcup\Gamma_i$, $\Delta = \sqcup\Delta_i$ *and* $\Gamma_i \vdash^{\tau_i} \mathcal{A} \triangleright \Delta_i$ *for all i.*

PROOF. Let $\Gamma \vdash_{\mathsf{min}} \mathcal{A} \triangleright \Delta$. First, by structural induction on $\mathcal{A}$, we show that there exist $\{\Gamma_i\}$ and $\{\Delta_i\}$ such that $\Gamma_i \vdash^{\tau_i} \mathcal{A} \triangleright \Delta_i$ for all i and, additionally, $\Gamma_i \Subset \Gamma$ and $\Delta_i \Subset \Delta$. This can be done by following the typing of $\mathcal{A}$ inductively in both Figures 12 and 18. We discuss some of the cases.

— For initialization, 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 (G-MTINIT-1/2).
  — (GT-MTINIT-1). Note that in this case, there are no subterms of $\mathcal{A}$ using service channel ch. From the rule, we know:
    (1) $\mathcal{A} = A^{\tau_1} \to B^{\tau_2} : ch(\boldsymbol{\nu}\, \tilde{s})\mathbf{.}\ \mathcal{A}'$
    (2) $\Gamma = \Gamma',\ ch@B : (\tilde{s})(\beta[\mathtt{end}/\tilde{\mathbf{t}}])$
    (3) $\Gamma' \vdash_{\mathsf{min}} \mathcal{A} \triangleright \Delta \cdot \tilde{s}\,[B, A] : \beta$
    (4) $ch \notin dom(\Gamma')$
    (5) $\mathsf{tvar}(\beta) \subseteq \tilde{\mathbf{t}}$

    We wish to prove $\Gamma_\tau \vdash^{\tau} \mathcal{A} \triangleright \Delta_\tau$ for any τ. We need to proceed by cases, on τ.
    — $\tau = \tau_1$. In this case, we can apply rule (GT-MTINIT-S1A) by using induction hypothesis and these five points.
    — $\tau = \tau_1$. Similar to previous case, but applying (GT-MTINIT-S1P) instead.
    — $\tau_1 \neq \tau \neq \tau_2$. It follows by induction hypothesis and by applying (GT-MTINIT-O) noting that $\Subset$ includes the subset relation.
  — (GT-MTINIT-2). Similar to the previous case.
— The case for communication is the same as the case of initialization. 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 that the types inferred simultaneously for each τ_i in $\Gamma_i \vdash^{\tau_i} \mathcal{A} \triangleright \Delta_i$ fully cover 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 differs. In each case, the nonthreaded 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 $\Gamma_i \vdash^{\tau_i} \mathcal{A} \triangleright \Delta_i$. □

Next we proceed to the step **(b)** for proving type preservation of EPP. 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 F.4. *Assume* P_1 *and* P_2 *are not replicated processes and* $P_1 \bowtie P_2$. *Let* $\Gamma_i \vdash^{\star} P_i \triangleright \Delta_i$ *(*$i = 1, 2$*) and* $P_1 \sqcup P_2$ *is typable, all under the same grouping of session names. Then* $\Gamma_1 \sqcup \Gamma_2 \vdash^{\star} P_1 \sqcup P_2 \triangleright \Delta_1 \sqcup \Delta_2$.

PROOF. By induction on the generation rules of $\sqcup$ in Definition 5.17 and its corresponding rule in Definition 5.16. The only nontrivial case is the merging of input branch. Let $P \bowtie Q$ such that $P = s \triangleright \Sigma_{i\in J} op_i(y_i)\mathbf{.}\ P_i$ and $Q = \sqcup s \triangleright \Sigma_{i\in K} op_i(y_i)\mathbf{.}\ Q_i$. Then

their merge is defined as:

$$s \rhd \Sigma_{i\in J}\mathsf{op}_i(y_i).\ P_i \sqcup s \rhd \Sigma_{i\in K}\mathsf{op}_i(y_i).\ Q_i \stackrel{\text{def}}{=} s \rhd \left(\begin{array}{l}\Sigma_{i\in J\cap K}\mathsf{op}_i(y_i).\ (P_i \sqcup Q_i) + \\ \Sigma_{i\in J\setminus K}\mathsf{op}_i(y_i).\ P_i + \\ \Sigma_{i\in K\setminus J}\mathsf{op}_i(y_i).\ Q_I\end{array}\right),$$

which corresponds to (M-IN) in Definition 5.16. Safely neglecting the invariant part (which has no effect on the argument), we set:

(1) $\Delta_1 = \tilde{s} : \Sigma_i s \blacktriangleright \mathsf{op}_i(\theta_i).\ \alpha_i$.
(2) $\Delta_2 = \tilde{s} : \Sigma_j s \blacktriangleright \mathsf{op}_j(\theta_j).\ \beta_j$.

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 Γ_1 and Γ_2, we may be assuming interactions with services, which are treated similarly (using the rule for session invocation in Definition 5.17). □

PROPOSITION F.5 (THREAD PROJECTION AND THREAD TYPING). *Assume I is coherent and $\mathcal{A}$ is its consistent annotation. Then $\Gamma_i \vdash^{\tau_i} \mathcal{A} \rhd \Delta_i$ implies $\Gamma_i' \vdash^{\star} \mathsf{TP}(\mathcal{A}, \tau_i) \rhd \Delta_i'$ 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 $\Gamma_i \vdash^{\tau_i} \mathcal{A} \rhd \Delta_i$ in Figure 18, on the one hand, and the result of taking $\mathsf{TP}(\mathcal{A}, \tau_i)$ and typing it by the minimum typing using Lemma F.4.

— (GT-MTINIT-S1A). Consider the case when $\tau = \tau_1$. By assumption we have

$$\Gamma \vdash^{\tau} \mathcal{A} \rhd \Delta \cdot \tilde{s}\,[A] : \beta,$$

which, by induction hypothesis, implies:

$$\Gamma' \vdash^{\star} \mathsf{TP}(\mathcal{A}, \tau) \rhd \Delta' \cdot \tilde{s} : \beta.$$

Hence, using (E-MTINITOUT1), we have:

$$\Gamma,\ ch@B : (\tilde{s})(\beta[\mathtt{end}/\tilde{\mathbf{t}}]) \vdash^{\star} ch \rhd A^{\tau_1} \to B^{\tau_1} : ch(\boldsymbol{\nu}\, s).\ \mathcal{A} \rhd \Delta,$$

as required. Similarly (GT-MTINIT-S1P) is reasoned using (E-MTINITIN).
— (GT-MTINIT-S2A). By assumption we have

$$\Gamma,\ \overline{ch}@B : (\tilde{s})\alpha \vdash^{\tau} \mathcal{A} \rhd \Delta \cdot \tilde{s}[A] : \beta,$$

which entails, by induction hypothesis,

$$\Gamma,\ \overline{ch}@B : (\tilde{s})\alpha \vdash^{\star} \mathsf{TP}(\mathcal{A}, \tau) \rhd \Delta' \cdot \tilde{s} : \beta.$$

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:

α and β has an upper bound w.r.t. $\sqsubseteq$ in Definition F.1.

This is because, if not, when we put the session initialization $A^{\tau_1} \to B^{\tau_2} : ch(\boldsymbol{v}\,\tilde{s})$, the thread τ_2 at ch and another passive thread at ch that 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 that come out as inputs for its dual interactions.

By this claim and by Proposition F.2 (3) we know $\alpha \sqcup \beta = \alpha \vee \beta$, hence we have

$$\Gamma,\ \overline{ch}@B : (\tilde{s})(\alpha \sqcup \beta[\mathtt{end}/\tilde{\mathbf{t}}]) \vdash \mathsf{TP}(\mathcal{A}', \tau) \rhd \Delta',$$

as required. (GT-MTINIT-S2P) is reasoned by the essentially identical argument.

— Cases (GT-MTINIT-O), (GT-MTCOMM-SA), (GT-MTCOMM-SP) as well as GT-MTCOM-O follow directly by (IH). For example, in (GT-MTCOM-O), the thread projection ignores the prefix, which is what Definition 5.18 does.

— (GT-MTSUM), by assumption we have

$$\Gamma_1 \vdash^{\tau} \mathcal{A}_1 \rhd \Delta_1 \quad \Gamma_2 \vdash^{\tau} \mathcal{A}_2 \rhd \Delta_2,$$

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 that are trivially identical) the same reasoning as the Claim shows $\Gamma_1 \vee \Gamma_2 = \Gamma_1 \sqcup \Gamma_2$. For session typings, consider the interacting (dual) behavior, 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 F.4 and (IH).

— (GT-MTIF) is reasoned as (GT-MTSUM), 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 the following.

PROPOSITION F.6 (TYPING FOR MERGED THREADS). *Let $\mathcal{A}$ be coherent and assume that, for some ch, we have $\{\tau_i\} = \mathsf{threads}(\mathcal{A}, ch)$. Then $\Gamma_i \vdash^{\star} \mathsf{TP}(\mathcal{A}, \tau_i) \rhd \Delta_i$ for each $\tau_i \in \{\tau_i\}$ implies $\bigsqcup_i \Gamma_i \vdash^{\star} \bigsqcup_i \mathsf{TP}(\mathcal{A}, \tau_i) \rhd \bigsqcup_i \Delta_i$.*

PROOF. Immediate from Lemma F.4 (note the order of merging does not matter by Proposition 5.23 (commutativity and associativity of $\sqcup$). □

We now prove Theorem 5.30 (1). Take the consistent annotation $\mathcal{A}$ and assume $\{\tau_i\}$ are the threads of $\mathcal{A}$ (which we assume does not include free term variables for simplicity). Since $\Gamma \vdash_{\min} \mathcal{A} \rhd \Delta$, we have:

$$\Gamma_i \vdash^{\tau_i} \mathcal{A} \rhd \Delta_i$$

for each $\tau_i \in \{\tau_i\}$ for which, by Proposition F.3, we have (1) $\sqcup_i \Gamma_i = \Gamma$ and (2) $\sqcup_i \Delta_i = \Delta$. By Proposition F.5, we also have, for each $\tau_i \in \{\tau_i\}$:

$$\Gamma_i \vdash^{\star} \mathsf{TP}(\mathcal{A}, \tau_i) \rhd \Delta_i.$$

Now, suppose $\mathcal{A}$ contains $\{\tau'_j\}$ as the threads for a server at ch and consider

$$\Gamma_j \vdash^{\tau'_j} \mathcal{A} \rhd \Delta_j,$$

as well as

$$\Gamma_j \vdash^{\star} \mathsf{TP}(\mathcal{A}, \tau'_j) \rhd \Delta_j.$$

By Proposition F.6, we have that

$$\bigsqcup_j \Gamma_j \vdash^{\star} \bigsqcup_j \mathsf{TP}(\mathcal{A}, \tau_j) \rhd \bigsqcup_j \Delta_j$$

gives the replicated input at ch. Now, $\bigsqcup_j \Gamma_j$ is the service typing at ch and zero or more client typings, in addition to assignment to variables, as required.

F.6. Proof of Theorem 5.30: (2, 4) Soundness

We will show the proof for (2), to avoid the cluttering of the notation. The proof also literally serves as that of (4).

THEOREM 5.30 (2). If $\mathsf{EPP}(\mathcal{A}, \sigma) \rightsquigarrow N$ then there exists $\mathcal{A}'$ such that $(\sigma,\ \mathcal{A}) \rightsquigarrow (\sigma',\ \mathcal{A}')$ such that $\mathsf{EPP}(\mathcal{A}', \sigma') \prec \equiv_{rec} N$.

PROOF. For soundness, we have the following lemma.

LEMMA F.7. *Assume* $P_1 \bowtie P_2$ *and let* $P \stackrel{def}{=} P_1 \sqcup P_2$. *Then* $P_i \prec P$ $(i = 1, 2)$.

PROOF. Immediate by the construction, noting $\prec$ is compatible. □

Let $\mathcal{A}$ be coherent and assume $\mathcal{A}$ has the threads $\mathcal{T} = \{\tau_i\}$. Let their thread projections be $P_i \stackrel{\text{def}}{=} \mathsf{TP}(\mathcal{A}, \tau_i)$ for each $\tau_i \in \mathcal{T}$, which as a whole gives the indexed family of processes, $\{P_i\}$. If a P_i (which is to be strict an index i together with the associated process P_i) is a replicated input then it needs to be merged with other replicated inputs at the same service channel. If a P_i is not a replicated input then it is never merged with other threads. We can now form a partition (a quotient set) of thread projections Ψ:

(1) Ψ partitions the family $\{P_i\}$: that is, 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) 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, that is, 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:

$$\mathsf{EPP}(\mathcal{A}, \sigma) \stackrel{\text{def}}{=} (\nu \tilde{t})(\Pi_{A \in \mathcal{P}} A\,[\Pi_{S \in \Psi(A)} P_S]_{\sigma_A}). \tag{7}$$

Note reduction is never affected by participants information as far as projected processes go (since redexes are always interparticipants or involves only a single participant). Thus for legibility we neglect participant information from (7) and consider the initial configuration:

$$\mathsf{EPP}(\mathcal{A}, \sigma) \stackrel{\text{def}}{=} (\nu \tilde{t}) \left(\Pi_{S \in \Psi} \bigsqcup_{P_i \in S} P_i, \quad \sigma \right), \tag{8}$$

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 adapted to the shape of (8). 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 pag. 16, we list these rules in the following (note that, as noted before, we are assuming variables are globally distinct, so that there is no need to local assignments). In the following, $C[\ ]_r$ denotes a reduction context, that is, a context whose hole is not under any prefix (thus we can always set $C[P]_r \equiv (\nu\tilde{s})(P|R)$ for some R and $\tilde{s}$).

$$(\text{INIT})\ \frac{P \equiv C_r[!\,ch(\tilde{s}).Q \mid \overline{ch}(\nu\tilde{s}).R] \quad P' \equiv C_r[!\,ch(\tilde{s}).Q \mid (\nu\tilde{s})(Q|R)]}{(P,\sigma) \rightsquigarrow (P',\sigma)}$$

$$(\text{COM})\ \frac{P \equiv C_r[s \rhd \Sigma_i \mathsf{op}_i(x_i).\ Q_i \mid \overline{s} \lhd \mathsf{op}_j\langle e\rangle.\ R] \quad P' \equiv C_r[Q_i \mid R] \quad \sigma \vdash e \Downarrow V}{(P,\sigma) \rightsquigarrow (P',\sigma[x \mapsto V])}$$

$$(\text{ASSIGN})\ \frac{P \equiv C_r[x := e.Q] \quad P' \equiv C_r[Q] \quad \sigma \vdash e \Downarrow V}{(P,\sigma) \rightsquigarrow (P,\sigma[x \mapsto V])}$$

$$(\text{IFTRUE})\ \frac{P \equiv C_r[\mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2] \quad P' \equiv C_r[P_1] \quad \sigma \vdash e \Downarrow \mathsf{tt}}{(P,\sigma) \rightsquigarrow (P',\sigma)}$$

$$(\text{SUM})\ \frac{P \equiv C_r[P_1 \oplus P_2] \quad P' \equiv C_r[P_1]}{(P,\sigma) \rightsquigarrow (P',\sigma)}$$

$$(\text{REC})\ \frac{P \equiv C_r[\mu X.Q] \quad (C_r[Q[(\mu X.Q)/X]],\ \sigma) \rightsquigarrow (P',\ \sigma')}{(P,\sigma) \rightsquigarrow (P',\sigma')}.$$

We omit (IFFALSE) (which follows (IFTRUE)) and the symmetric case of (SUM). We can easily check these rules give the reduction relation 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 nontrivial for recursion, since all other cases are the base cases). Also, we set:

$$\mathcal{A} \equiv (\nu\tilde{t})\Pi_{0\le i\le n}\mathcal{A}_I,$$

where each $\mathcal{A}_i$ is prime, that is, is not itself a nontrivial parallel composition. For simplicity we safely neglect $(\nu\tilde{t})$ from now on (since the hiding does not affect reduction) and consider only $\Pi_{0\le i\le n}\mathcal{A}_i$.

We start from (INIT). Then the redex is a pair, which is given by an input:

$$!ch(\tilde{s}).Q \stackrel{\text{def}}{=} \bigsqcup_{0\le i\le n} !ch(\tilde{s}).Q_i \equiv !ch(\tilde{s}).\bigsqcup_{0\le i\le n} Q_i \quad (\{!ch(\tilde{s}).Q_i\} \in \Psi)$$

and an output:

$$\overline{ch}(\nu\tilde{s})R \in \{\overline{ch}(\nu\tilde{s})R\} \qquad (\{\overline{ch}(\nu\tilde{s})R\} \in \Psi).$$

Thus we can write down the reduction up to $\equiv$:

$$(!ch(\tilde{s}).Q \mid \overline{ch}(\nu\tilde{s})R \mid S,\ \sigma) \rightsquigarrow (!ch(\tilde{s}).Q \mid (\nu\tilde{s})(Q|R) \mid S,\ \sigma). \tag{9}$$

Note we are neglecting participants for legibility, as stipulated before.

Now $\overline{ch}(\nu\tilde{s})R$ come from the thread τ. Then τ should be the active top-level thread (since if not R cannot be an output, by well-threaded-ness: cf. *(CC)* of Definition 5.8). 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 initialization 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

$$\mathcal{A}_0 \equiv A^{\tau} \to B^{\tau'} : ch(\boldsymbol{\nu}\, \tilde{s}).\ \mathcal{A}'_0,$$

which induces the following reduction:

$$(\mathcal{A},\ \sigma).\ \rightsquigarrow\ (\nu\tilde{s})(\mathcal{A}'_0 | \Pi_{1\leq i\leq n}\mathcal{A}_i,\ \sigma). \tag{10}$$

Let the term on the right-hand side, without including $(\nu\tilde{s})$, 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 (10) we have:

$$\forall \tau_i \in (\mathcal{T}\setminus\{\tau,\tau'\}).\ \mathsf{TP}(\mathcal{A},\tau_i) \equiv \mathsf{TP}(\mathcal{A}',\tau_i).$$

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 *(FC)*, freshness, Definition 5.8). Thus we can construct Ψ' for $\mathcal{A}'$ as we constructed Ψ for $\mathcal{A}$ as the same collection of sets of processes except for the following.

(1) We replace $\{\overline{ch}(\nu\tilde{s}).\ R\} \in \Psi$ 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 $((\nu\tilde{s})\mathcal{A}', \sigma)$ as (neglecting participants as we stipulated):

$$\bigsqcup_{1\leq i\leq n} !ch(\tilde{s}).Q_i \mid (\nu\tilde{s})(Q_0|R) \mid S \tag{11}$$

together with σ. We now compare (11) with the right-hand side of (9), the result of reducing the EPP of $\mathcal{A}$. By Lemma F.7 we have $\bigsqcup_{1\leq i\leq n}!ch(\tilde{s}).Q_i \prec \bigsqcup_{0\leq i\leq n}!ch(\tilde{s}).Q_i$ as well as $Q_0 \prec \bigsqcup_{0\leq i\leq n} Q_i$, hence we are done.

For (COM), the reduction of the endpoint projection of $\mathcal{A}$ can be written as:

$$(s \rhd \Sigma_i \mathsf{op}_i(y_i).\ Q_i \mid \overline{s} \lhd \mathsf{op}_j\langle e\rangle.\ R \mid S,\ \sigma)\ \rightsquigarrow\ (Q_j \mid R \mid S,\ \sigma). \tag{12}$$

Again the thread, say τ, for $\overline{s} \lhd \mathsf{op}_j\langle e\rangle.\ R \mid S$ for which we have $\{\overline{s} \lhd \mathsf{op}\langle e\rangle.\ R\} \in \Psi$, cannot start from an intermediate node, so it is a top-level active thread. Then the thread corresponding to $s \rhd \Sigma_i \mathsf{op}_i(y_i).\ Q_i$ should be its dual. Let the prime interaction starting from these two threads be $\mathcal{A}_0$. Then we can write:

$$\mathcal{A}_0 \equiv A \to B : s\langle \mathsf{op},\ e,\ y\rangle.\ \mathcal{A}'_0$$

by which we in fact know the sum we assumed in (12) is a singleton,[6] that is, we can restrict our attention to:

$$(s \triangleright \mathsf{op}(y).\ Q \mid \overline{s} \triangleleft \mathsf{op}_j\langle e\rangle.\ R \mid S,\ \sigma) \rightsquigarrow (Q \mid R \mid S,\ \sigma). \tag{13}$$

Now $\mathcal{A}$ has the following reduction:

$$(\mathcal{A},\ \sigma).\ \rightsquigarrow (\mathcal{A}'_0 \mid \Pi_{1\leq i\leq n}\mathcal{A}_i,\ \sigma), \tag{14}$$

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 initialization), otherwise remaining the same as $\mathcal{A}$. Thus in this case we have the precise correspondence, that is, 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:

$$(x := e.Q \mid R,\ \sigma) \rightsquigarrow (Q \mid R,\ \sigma), \tag{15}$$

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, (15) is precisely matched by the global reduction:

$$(\mathcal{A},\ \sigma) \rightsquigarrow (\mathcal{A}'_0|\Pi_{1\leq i\leq n}\mathcal{A}_i,\ \sigma'). \tag{16}$$

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:

$$(\mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2|R,\ \sigma) \rightsquigarrow (P_1|R,\ \sigma) \tag{17}$$

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:

$$(\mathcal{A},\ \sigma) \rightsquigarrow (\mathcal{A}'_{0t}|\Pi_{1\leq i\leq n}\mathcal{A}_i,\ \sigma). \tag{18}$$

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:

$$\mathsf{TP}(\mathcal{A}'_{0t}, \tau) \quad \equiv \quad P_1. \tag{19}$$

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 $\prec$-smaller than that of τ_1 in $\mathcal{A}_0$, that is, for such τ_1:

$$\mathsf{TP}(\mathcal{A}', \tau_1) \ \prec\ \mathsf{TP}(\mathcal{A}, \tau_1).$$

[6]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, nontrivial sums arise, as is usually the case in endpoint projections of meaningful protocols.

Since the projection of other threads remain identical, we know:

$$\forall \tau_i \in \mathcal{T} \backslash \{\tau\}.\ \mathsf{TP}(\mathcal{A}', \tau_i) \prec \mathsf{TP}(\mathcal{A}, \tau_i).$$

By (19) we are done. (IFFALSE) is the same as (IFTRUE).

For (SUM), the original reduction is

$$(P_1 \oplus P_2,\ \sigma) \rightsquigarrow (P_1,\ \sigma), \tag{20}$$

which means we can set

$$\mathcal{A}_0 \quad \stackrel{\text{def}}{=} \quad \mathcal{A}'_{0l} + \mathcal{A}'_{0r}.$$

Corresponding to e (19) in the case of (IFTRUE), 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:

$$(\mu X.P_0 \mid \Pi_{1 \le i \le m} \mu X.P_i \mid R, \quad \sigma) \quad \rightsquigarrow \quad (S,\ \sigma') \tag{21}$$

from

$$(P_0[\mu X.P_0/X] \mid \Pi_{1 \le i \le m} \mu X.P_i \mid R, \quad \sigma) \rightsquigarrow (S,\ \sigma'), \tag{22}$$

where we specify all recursion terms coming from the same recursion in the global process, say:

$$\mathcal{A}_0 \quad \equiv \quad \mu X.\mathcal{A}'_0.$$

By (22) we know, writing $P'_i \stackrel{\text{def}}{=} P_i[\mu X.P_i/X]$:

$$(\Pi_{0 \le i \le m} P'_i \mid R, \quad \sigma) \rightsquigarrow (S,\ \sigma'), \tag{23}$$

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:

$$\mathcal{E} \quad \stackrel{\text{def}}{=} \quad \mathsf{TP}(\mathcal{A}'_0[\mu X.\mathcal{A}'_0/X], \tau_i).$$

The unfolding can increase fresh passive threads if $\mathcal{A}'_0$ contains initialization, 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

$$(\mathcal{E},\ \sigma) \quad \rightsquigarrow \quad (\mathcal{E}',\ \sigma') \tag{24}$$

such that

$$S \prec \equiv_{rec} \mathsf{EPP}(E', \sigma).$$

By (24) we also have:

$$(\sigma,\ \mathcal{A}) \quad \rightsquigarrow \quad (\sigma',\ \mathcal{E}')$$

hence we are done. This concludes all cases. □

F.7. Proof of Theorem 5.30: (3, 5) Completeness

As before we focus on the proof of (3), to avoid the cluttered notation. The proof also serves as that of (5).

THEOREM 5.30 (3). If $\mathsf{EPP}(\mathcal{A}, \sigma) \rightsquigarrow N$ then there exists $\mathcal{A}'$ such that $(\sigma,\ \mathcal{A}) \rightsquigarrow (\sigma',\ \mathcal{A}')$ such that $\mathsf{EPP}(\mathcal{A}', \sigma') \prec \equiv_{rec} N$.

PROOF. Completeness is by induction on the derivation of reduction in the global calculus. As before, we consider adapted reduction rules that are taken modulo structural equality and reduction contexts, which are equivalent to the reduction rules given initially. We list these adapted rules in the following, using the same names as the original ones. As before, we assume all variables are distinct, including across participants, so that we can write, for instance, $\sigma \vdash e \Downarrow v$ instead of $\sigma \vdash e@A \Downarrow v$. We again write $C_r[\,]$ for a reduction context in the grammar of interactions.

$$(\textsc{Init})\ \frac{I \equiv C_r[A \to B : ch(\boldsymbol{\nu}\, \tilde{s}).\ I_0] \quad I' \equiv C_r[(\boldsymbol{\nu} \tilde{s})I_0]}{(\sigma, I) \rightsquigarrow (\sigma, I')}$$

$$(\textsc{Comm})\ \frac{I \equiv C_r[A \to B : s\langle \mathsf{op},\ e,\ x\rangle.\ I_0] \quad I' \equiv C_r[I_0] \quad \sigma \vdash e \Downarrow v}{(\sigma,\ I) \rightsquigarrow (\sigma[x@B \mapsto v],\ I')}$$

$$(\textsc{Assign})\ \frac{I \equiv C_r[x@A := e.\ I_0] \quad I' \equiv C_r[I_0] \quad \sigma \vdash e \Downarrow v}{(\sigma,\ I) \rightsquigarrow (\sigma[x@B \mapsto v],\ I')}$$

$$(\textsc{IfTrue})\ \frac{I \equiv C_r[\mathsf{if}\ e\ \mathsf{then}\ I_{0t}\ \mathsf{else}\ I_{0f}] \quad I' \equiv C_r[I_{0t}] \quad \sigma \vdash e \Downarrow \mathsf{tt}}{(\sigma, I) \rightsquigarrow (\sigma, I')}$$

$$(\textsc{Sum})\ \frac{I \equiv C_r[I_{0l} + I_{0r}] \quad I' \equiv C_r[I_{0l}]}{(\sigma, I) \rightsquigarrow (\sigma, I')}$$

$$(\textsc{Rec})\ \frac{I \equiv C_r[\mu X.I_0] \quad (\sigma,\ C_r[I[(\mu X.I_0)/X]]) \rightsquigarrow (\sigma',\ I')}{(\sigma, I) \rightsquigarrow (\sigma', I')}.$$

We omit (IFFALSE) and the symmetric case. Note the rules (PAR) and (RES) are no longer necessary since they are absorbed in these rules. Up to the application of the rules of $\equiv$, all the rules except (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

$$(\sigma, \mathcal{A}) \rightsquigarrow (\sigma', \mathcal{A}'),$$

then

$$\mathsf{EPP}(I, \sigma) \rightsquigarrow (P', \sigma'),$$

where

$$\mathsf{EPP}(\mathcal{A}', \sigma') = (P'_0, \sigma') \text{ such that } P' \prec \equiv_{rec} P'_0. \tag{25}$$

Here, 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 (25) to:

$$(P,\ \sigma') \quad \prec \equiv_{rec} \quad \mathsf{EPP}(\mathcal{A}', \sigma'). \tag{26}$$

We set

$$\mathcal{A} \quad \equiv \quad (\nu \tilde{t})\Pi_{0 \le i \le n} \mathcal{A}_I,$$

where each $\mathcal{A}_i$ is a prime interaction (i.e., an interaction that does not contain a non-trivial top-level parallel composition). Henceforth we safely neglect $(\nu\tilde{t})$. 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 (INIT), we can set:

$$\mathcal{A}_0 \stackrel{\text{def}}{=} A^{\tau_0} \to B^{\tau_1} : ch(\boldsymbol{\nu}\,\tilde{s}).\ \mathcal{A}'_0$$

and consider the reduction:

$$(\sigma, \mathcal{A}) \quad \rightsquigarrow \quad (\sigma, (\nu\tilde{s})\mathcal{A}'_0 \mid \Pi_{1\le i\le n}\mathcal{A}_i). \tag{27}$$

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:

$$\overline{ch}(\nu\tilde{s})R \in \{\overline{ch}(\nu\tilde{s})R\} \qquad (\{\overline{ch}(\nu\tilde{s})R\} \in \Psi).$$

Then we can write down $\mathcal{A}$ as $!ch(\tilde{s}.Q \mid \overline{ch}(\nu\tilde{s}).R \mid S$. Thus we have a reduction:

$$(!ch(\tilde{s}).Q \mid \overline{ch}(\nu\tilde{s})R \mid S,\ \sigma) \rightsquigarrow (!ch(\tilde{s}).Q \mid (\nu\tilde{s})(Q|R) \mid S,\ \sigma). \tag{28}$$

By the exactly identical reasoning as in the corresponding case in the proof of soundness, the residual in (27) and that in e (28) are related in the way:

$$\mathsf{EPP}((\nu\tilde{s})\mathcal{A}'_0|\Pi_{1\le i\le n}\mathcal{A}_i, \sigma) \prec (!ch(\tilde{s}).Q \mid (\nu\tilde{s})(Q|R) \mid S,\ \sigma).$$

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:

$$(\sigma,\ \mathcal{A}) \quad \rightsquigarrow \quad (\sigma',\ \mathcal{A}'_0|\Pi_{1\le i\le n}\mathcal{A}_i),$$

where $\sigma' = \sigma[x@B \mapsto v]$ with $\sigma \vdash e \Downarrow v$. The thread projection of $\mathcal{A}_0$ to τ_0 has the form $\overline{s} \lhd \mathsf{op}\langle e\rangle.\ R$ such that $\{\overline{s} \lhd \mathsf{op}\langle e\rangle.\ R\} \in \Psi$, while the one onto τ_1 has the form $s \rhd \mathsf{op}(x).\ Q$ (when the branching is a singleton we omit the symbol Σ, similarly henceforth). Without loss of generality (cf. Proposition 5.15) we regard τ_0 and τ_1 are used only in $\mathcal{A}_0$. Thus, we can set:

$$\mathsf{EPP}(\mathcal{A}, \sigma) \equiv (s \rhd \mathsf{op}(x).\ Q \mid \overline{s} \lhd \mathsf{op}_j\langle e\rangle.\ R \mid S,\ \sigma);$$

hence, we have:

$$\mathsf{EPP}(\mathcal{A}, \sigma) \rightsquigarrow (Q \mid R \mid S,\ \sigma'). \tag{29}$$

(in (29), 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

$$\mathsf{EPP}(\mathcal{A}'_0 \mid \Pi_{1\le i\le n}\mathcal{A}_i, \sigma') \prec (Q \mid R \mid S,\ \sigma')$$

as required.

For (ASSIGN), we can set

$$\mathcal{A}_0 \stackrel{\text{def}}{=} x^{\tau}@A := e.\mathcal{A}'_0.$$

We consider the reduction:

$$(\sigma,\ \mathcal{A}) \quad \rightsquigarrow \quad (\sigma',\ \mathcal{A}'_0|\Pi_{1\leq i\leq n}\mathcal{A}_i) \tag{30}$$

with appropriate σ'. The thread projection onto τ has the shape $x := e.\mathsf{TP}(\mathcal{A}'_0, \tau)$, hence we have the reduction:

$$(x@A := e.\mathsf{TP}(\mathcal{A}'_0, \tau) \mid R,\ \sigma) \rightsquigarrow (\mathsf{TP}(\mathcal{A}'_0, \tau) \mid R,\ \sigma'). \tag{31}$$

As in the corresponding case in the proof of soundness, (31) 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 (30), hence as required.

For (IFTRUE), we can set

$$\mathcal{A}_0 \quad \stackrel{\text{def}}{=} \quad \text{if } e^{\tau}@A \text{ then } \mathcal{A}'_{0t} \text{ else } \mathcal{A}'_{0f}$$

with which we have the reduction:

$$(\mathcal{A},\ \sigma).\ \rightsquigarrow\ (\mathcal{A}'_{0t}|\Pi_{1\leq i\leq n}\mathcal{A}_i,\ \sigma). \tag{32}$$

Observing

$$\mathsf{TP}(\mathcal{A}_0, \tau) \stackrel{\text{def}}{=} \text{if } e \text{ then } \mathsf{TP}(\mathcal{A}'_{0t}, \tau) \text{ else } \mathsf{TP}(\mathcal{A}'_{0f}, \tau),$$

we have the reduction for the endpoint projection:

$$(\text{if } e \text{ then } \mathsf{TP}(\mathcal{A}'_{0t}, \tau) \text{ else } \mathsf{TP}(\mathcal{A}'_{0f}, \tau) \mid R,\ \sigma) \quad \rightsquigarrow \quad (\mathsf{TP}(\mathcal{A}'_{0t}, \tau) \mid R,\ \sigma), \tag{33}$$

where e evaluates to true in σ. By the reasoning for the corresponding case in the soundness proof, R in (33) may contain replicated inputs that are the result of merging complete threads from $\mathcal{A}'_{0f}$. Thus we obtain:

$$\mathsf{EPP}(\mathcal{A}'_{0t}|\Pi_{1\leq i\leq n}\mathcal{A}_i, \sigma) \prec (\mathsf{TP}(\mathcal{A}'_{0t}, \tau) \mid R,\ \sigma),$$

as required.

(IFFALSE) and (SUM) are similarly reasoned.

For (REC), let:

$$\mathcal{A}_0 \quad \stackrel{\text{def}}{=} \quad \mu X.\mathcal{A}'_0.$$

Further assume we have:

$$(\mathcal{A},\ \sigma) \rightsquigarrow (\mathcal{A}''_0|\Pi_{1\leq i\leq n}\mathcal{A}_i,\ \sigma'). \tag{34}$$

The reduction (34) comes from, by the recursion rule:

$$(\mathcal{A}'_0[(\mu X.\mathcal{A}'_0)/X], \sigma) \rightsquigarrow (\mathcal{A}''_0,\ \sigma'). \tag{35}$$

Now the endpoint projections of $\mathcal{A}_0$ has the form:

$$\mathsf{EPP}(\mathcal{A}_0, \sigma) \quad \stackrel{\text{def}}{=} \quad ((\Pi P'_i) \mid R,\ \sigma). \tag{36}$$

where R is a collection of replicated processes and each P'_i is not replicated and has the shape:

$$P'_i \stackrel{\text{def}}{=} \mu X.P_i.$$

We then consider the endpoint projection of the unfolding of $\mathcal{A}_0$:

$$\mathsf{EPP}(\mathcal{A}'_0[(\mu X.\mathcal{A}'_0)/X], \sigma) \quad \stackrel{\text{def}}{=} \quad ((\Pi P_i[P'_i/X]) \mid R,\ \sigma). \tag{37}$$

Note the right-hand side of (37) is the n-times unfoldings of (36). Thus by induction hypothesis and applying the recursion rule in the endpoint processes n-times we obtain:

$$\mathsf{EPP}(\mathcal{A}, \sigma) \rightsquigarrow \mathsf{EPP}(\mathcal{A}_0'' | \Pi_{1 \leq i \leq n} \mathcal{A}_i, \sigma') \tag{38}$$

as required. This exhausts all cases, establishing completeness.

This concludes the proof of Theorem 5.30 (3). □

ACKNOWLEDGMENTS

We deeply thank Robin Milner for setting up directions of ongoing collaboration with W3C WS-CDL WG as a scientific advisor. We thank WS-CDL WG members, in particular Gary Brown, Steve Ross-Talbot and Nickolas Kavantzas for collaborations; and Joshua Guttman for his comments on an early version of the article. Thanks to Søren Debois for several technical suggestions.

REFERENCES

ABADI, M. AND FOURNET, C. 2001. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'01)*. ACM Press, New York, NY, 104–115.

ABADI, M. AND GORDON, A. D. 1999. A calculus for cryptographic protocols: The SPI calculus. *Inf. Comput. 148*, 1, 1–70.

ALUR, R., ETESSAMI, K., AND YANNAKAKIS, M. 2005. Realizability and verification of MSC graphs. *Theor. Comput. Sci. 331*, 1, 97–114.

ARBAB, F. 2004. Reo: A channel-based coordination model for component composition. *Math. Struct. Comput. Sci. 14*, 3, 329–366.

BAETEN, J., VAN BEEK, H., AND MAUW, S. 2001. Specifying internet applications with DiCons. In *Proceedings of the ACM Symposium on Applied Computing (SAC'01)*. ACM Press, New York, NY, 576–584.

BASU, A., BOZGA, M., AND SIFAKIS, J. 2006. Modeling heterogeneous real-time components in bip. In *Proceedings of the International Conference on Software Engineering and Formal Methods (SEFM'06)*. IEEE Computer Society Press, 3–12.

BENTON, N., CARDELLI, L., AND FOURNET, C. 2004. Modern concurrency abstractions for C#. *ACM Trans. Program. Lang. Syst. 26*, 5, 769–804.

BERGER, M., HONDA, K., AND YOSHIDA, N. 2001. Sequentiality and the π-calculus. In *Proceedings of the 5th International Conference on Typed Lambda Calculi and Applications (TLCA'01)*. Lecture Notes in Computer Science, vol. 2044, Springer, 29–45.

BHARGAVAN, K., FOURNET, C., AND GORDON, A. D. 2006. Verified reference implementations of WS-security protocols. In *Proceedings of the 3rd International Workshop on Web Services and Formal Methods (WS-FM'06)*. Lecture Notes in Computer Science, vol. 4184, Springer, 88–106.

BHARGAVAN, K., CORIN, R., MALO DENILOU, P., FOURNET, C., AND LEIFER, J. J. 2009. Cryptographic protocol synthesis and verification for multiparty sessions. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium (CSF'09)*. IEEE Computer Society, 124–140.

BONELLI, E., COMPAGNONI, A. B., AND GUNTER, E. L. 2005. Correspondence assertions for process synchronization in concurrent communications. *J. Funct. Program. 15*, 2, 219–247.

BRAVETTI, M. AND ZAVATTARO, G. 2007. A theory for strong service compliance. In *Proceedings of the 8th International Conference on Coordination Models and Languages (COORDINATION'07)*. Lecture Notes in Computer Science, vol. 4467, Springer, 96–112.

BRIAIS, S. AND NESTMANN, U. 2005. A formal semantics for protocol narrations. In *Proceedings of the 1st Symposium on Trustworthy Global Computing (TGC'05)*. Lecture Notes in Computer Science, vol. 3705, Springer, 163–181.

BROY, M. 2005. A semantic and methodological essence of message sequence charts. *Sci. Comput. Program. 54*, 2–3, 213–256.

BROY, M. 2007. Interaction and realizability. In *Proceedings of the 33rd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'07)*. Springer, 29–50.

BROY, M., KRÜGER, I. H., AND MEISINGER, M. 2007. A formal model of services. *ACM Trans. Softw. Engin. Method. 16*, 1, 5.

BUSI, N., GORRIERI, R., GUIDI, C., LUCCHI, R., AND ZAVATTARO, G. 2006. Choreography and orchestration conformance for system design. In *Proceedings of the 7th International Conference on Coordination Models and Languages (COORDINATION'06)*. Lecture Notes in Computer Science, vol. 4038, Springer, 63–81.

CAIRES, L. AND VIEIRA, H. T. 2009. Conversation types. In *Proceedings of the 18th European Symposium on Programming (ESOP'09)*. Lecture Notes in Computer Science, vol. 5502, Springer, 285–300.

CARBONE, M., NIELSEN, M., AND SASSONE, V. 2004. A calculus for trust management. In *Proceedings of the 24th International Conference on Foundations of Software Technology and Theoretical Computer Science (FST-TCS'04)*. Lecture Notes in Computer Science, vol. 3328, Springer, 161–173.

CARBONE, M., HONDA, K., AND YOSHIDA, N. 2006a. A calculus of global interaction based on session types. In *Proceedings of the 2nd Workshop on Developments in Computational Models (DCM'06)*. Elsevier Science Publishers, Amsterdam, The Netherlands.

CARBONE, M., HONDA, K., YOSHIDA, N., MILNER, R., BROWN, G., AND ROSS-TALBOT, S. 2006b. A theoretical basis of communication-centred concurrent programming. W3C working note. http://www.dcs.qmul.ac.uk/~carbonem/cdlpaper/workingnote.pdf.

CARBONE, M., HONDA, K., AND YOSHIDA, N. 2007. Structured communication-centred programming for web services. In *Proceedings of the 16th European Symposium on Programming (ESOP'07)*. Lecture Notes in Computer Science, vol. 4421, Springer, 2–17.

CASTAGNA, G., GESBERT, N., AND PADOVANI, L. 2008. A theory of contracts for web services. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'08)*. ACM Press, New York, NY, 261–272.

CHAKI, S., RAJAMANI, S. K., AND REHOF, J. 2002. Types as models: model checking message-passing programs. In *Proceedings of the 29th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'02)*. ACM Press, New York, NY, 45–57.

CORIN, R., DENIÈLOU, P.-M., FOURNET, C., BHARGAVAN, K., AND LEIFER, J. 2007. Secure implementations for typed session abstractions. In *Proceedings of the 20th IEEE Computer Security Foundations Symposium (CSF'07)*. IEEE Computer Society Press, 170–186.

DEMANGEON, R. AND HONDA, K. 2011. Full abstraction in a subtyped pi-calculus with linear types. In *Proceedings of the International Conference on Concurrency Theory (CONCUR'11)*. J.-P. Katoen and B. König Eds., Lecture Notes in Computer Science Series, vol. 6901, Springer, 280–296.

DEZANI-CIANCAGLINI, M. AND DE' LIGUORO, U. 2009. Sessions and session types: An overview. In *Proceedings of the 6th International Workshop on Web Services and Formal Methods (WS-FM'09)*.

DEZANI-CIANCAGLINI, M., MOSTROUS, D., YOSHIDA, N., AND DROSSOPOULOU, S. 2006. Session types for object-oriented languages. In *Proceedings of the 20th European Conference on Object-Oriented Programming (ECOOP'06)*. Lecture Notes in Computer Science, vol. 4067, Springer, 328–352.

FÄHNDRICH, M., AIKEN, M., HAWBLITZEL, C., HODSON, O., HUNT, G. C., LARUS, J. R., AND LEVI, S. 2006. Language support for fast and reliable message-based communication in singularity OS. In *Proceedings of EuroSys'06*. W. Zwaenepoel Ed., ACM SIGOPS. ACM Press, 177–190.

FOSTER, H., MAGEE, J., KRAMER, J., AND UCHITEL, S. 2010. LTSA WS-engineer home page. http://www.doc.ic.ac.uk/ltsa/bpel4ws.

FU, X., BULTAN, T., AND SU, J. 2004. Conversation protocols: A formalism for specification and verification of reactive electronic services. *Theor. Comput. Sci. 328*, 1–2, 19–37.

FU, X., BULTAN, T., AND SU, J. 2005. Realizability of conversation protocols with message contents. *Int. J. Web Service Resear. 2*, 4, 68–93.

GAY, S. AND HOLE, M. 2005. Subtyping for session types in the pi calculus. *Acta Informatica 42*, 2–3, 191–225.

GORDON, A. D. AND PUCELLA, R. 2002. Validating a web service security abstraction by typing. In *Proceedings of the ACM Workshop on XML Security (XMLSEC'02)*. ACM Press, New York, NY, 18–29.

GUIDI, C., LUCCHI, R., ZAVATTARO, G., BUSI, N., AND GORRIERI, R. 2006. SOCK: A calculus for service oriented computing. In *Proceedings of the 4th International Conference on Service Oriented Computing (ICSOC'06)*. Lecture Notes in Computer Science, vol. 4294, Springer, 327–338.

GUTTMAN, J. D., THAYER, F. J., AND ZUCK, L. D. 2001. The faithfulness of abstract protocol analysis: message authentication. In *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS'01)*. ACM Press, New York, NY, 186–195.

HENNESSY, M. AND RIELY, J. 1998. Resource access control in systems of mobile agents. In *Proceedings of the International Workshop on High-Level Concurrent Languages (HLCL'98)*. ENTCS Series, vol. 16.3. Elsevier Science Publishers, Amsterdam, The Netherlands. 3–17.

HENRIKSEN, J. G., MUKUND, M., KUMAR, K. N., SOHONI, M. A., AND THIAGARAJAN, P. S. 2005. A theory of regular MSC languages. *Inf. Comput. 202*, 1, 1–38.

HOARE, T. 1985. *Communicating Sequential Processes*. Prentice Hall, UK.

HONDA, K. 1996. Composing processes. In *Proceedings of the 23th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'96)*. ACM Press, New York, NY, 344–357.

HONDA, K., VASCONCELOS, V., AND KUBO, M. 1998. Language primitives and type disciplines for structured communication-based programming. In *Proceedings of the 7th European Symposium on Programming (ESOP'98)*. Lecture Notes in Computer Science, vol. 1381, Springer, 22–138.

HONDA, K., YOSHIDA, N., AND CARBONE, M. 2007. Web Services, mobile processes and types. *Bull. Eur. Assoc. Theor. Comput. Sci. 91*, 165–185.

HONDA, K., YOSHIDA, N., AND CARBONE, M. 2008. Multiparty asynchronous session types. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'08)*. ACM Press, New York, NY, 273–284.

IBM. 2001. Web services flow language (WSFL 1.0). http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.

IBM. 2010. WebSphere MQ workflow. www-306.ibm.com/software/integration/wmqwf/.

IGARASHI, A. AND KOBAYASHI, N. 2001. A generic type system for the pi-calculus. In *Proceedings of the 28th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'01)*. ACM Press, New York, NY, 128–141.

INTERNATIONAL TELECOMMUNICATION UNION. 1996. Recommendation Z.120: Message sequence chart.

KOBAYASHI, N., PIERCE, B., AND TURNER, D. 1996. Linear types and π-calculus. In *Proceedings of the 23th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'96)*. ACM Press, New York, NY, 358–371.

LANEVE, C. AND PADOVANI, L. 2006. Smooth orchestrators. In *Proceedings of the 9th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'06)*. Lecture Notes in Computer Science, vol. 3921, Springer, 32–46.

MICROSOFT. 2001. XLANG: Web services for business process design. http://www.gotdotnet.com/team/xml wsspecs/xlang-c/default.htm.

MILNER, R. 1993. The polyadic π-calculus: A tutorial. In *Logic and Algebra of Specification*, Springer.

MILNER, R., PARROW, J., AND WALKER, D. 1992. A calculus of mobile processes, I and II. *Inf. Comput. 100*, 1, 1–40, 41–77.

NEEDHAM, R. M. AND SCHROEDER, M. D. 1978. Using encryption for authentication in large networks of computers. *Com. ACM 21*, 12, 993–999.

O'HANLON, C. 2006. Conversation with Steve Ross-Talbot.

OMG. 2004. Unified modelling language, version 2.0.

PI4SOA. 2008. http://www.pi4soa.org.

PIERCE, B. C. 2002. *Types and Programming Languages*. MIT Press, Cambridge, MA.

PIERCE, B. C. AND SANGIORGI, D. 1996. Typing and subtyping for mobile processes. *Math. Struct. Comput. Sci. 6*, 5, 409–453.

PIERCE, B. C. AND TURNER, D. N. 2000. Pict: A programming language based on the pi-calculus. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*, MIT Press, Cambridge, MA.

QIU, Z., ZHAO, X., CAI, C., AND YANG, H. 2007. Towards the theoretical foundation of choreography. In *Proceedings of the International World Wide Web Conference*. IEEE Computer Society Press, 973–982.

RAJAMANI, S. K. AND REHOF, J. 2002. Conformance checking for models of asynchronous message passing software. In *Proceedings of the 14th Conference on Computer-Aided Verification (CAV'02)*. Lecture Notes in Computer Science, vol. 2404, Springer, 166–179.

ROSS-TALBOT, S. AND FLETCHER, T. 2006. *WS-CDL Primer*. Published by W3C.

SANGIORGI, D. 1999. The name discipline of uniform receptiveness. *Theor. Comput. Sci. 221*, 1–2, 457–493.

TAKEUCHI, K., HONDA, K., AND KUBO, M. 1994. An interaction-based language and its typing system. In *Proceedings of the 6th International Conference on Parallel Architectures and Languages (PARLE'94)*. Lecture Notes in Computer Science Series, vol. 817, Springer, 398–413.

VAN DER AALST, W. 2002. Inheritance of interorganizational workflows: How to agree to disagree without loosing control? *Info. Tech. Manage. J. 2*, 3, 195–231.

VASCONCELOS, V., RAVARA, A., AND GAY, S. J. 2004. Session types for functional multithreading. In *Proceedings of the 15th International Conference on Concurrency Theory (CONCUR'04)*. Lecture Notes in Computer Science, vol. 3170, Springer, 497–511.

VIEIRA, H. T., CAIRES, L., AND SECO, J. C. 2008. The conversation calculus: A model of service-oriented computation. In *Proceedings of the 17th European Symposium on Programming (ESOP'08)*. Vol. 4960, Springer, 269–283.

W3C WS-CDL WORKING GROUP. 2004. Web services choreography description language version 1.0. http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/.

WFMC. 2010. Workflow management coalition. http://www.wfmc.org/.

YOSHIDA, N. 1996. Graph types for monadic mobile processes. In *Proceedings of the 16th Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer, 371–386.

YOSHIDA, N. AND VASCONCELOS, V. 2007. Language primitives and type discipline for structured communication-based programming revisited: Two systems for higher-order session communication. *Electr. Notes Theor. Comput. Sci. 171*, 4, 73–93.

YOSHIDA, N., BERGER, M., AND HONDA, K. 2004. Strong normalisation in the π-calculus. *Inf. Comput. 191*, 145–202.

Received April 2010; revised October 2011; accepted March 2012