Globally Governed Session Semantics

Dimitrios Kouzapas and Nobuko Yoshida

Imperial College London

Abstract. This paper proposes a new bisimulation theory based on multiparty session types where a choreography specification governs the behaviour of session typed processes and their observer. The bisimulation is defined with the observer cooperating with the observed process in order to form complete global session scenarios and usable for proving correctness of optimisations for globally coordinating threads and processes. The induced bisimulation is strictly more fine-grained than the standard session bisimulation. The difference between the governed and standard bisimulations only appears when more than two interleaved multiparty sessions exist. The framework is systematically applied to four kinds of synchronous and asynchronous semantics, and for each semantics, the soundness and completeness of the governed bisimilarity with respect to the governed reduction-based congruence is proved. Finally its usage is demonstrated by a thread transformation governed under multiple sessions which is applicable to real choreographic usecase scenarios.

1 Introduction

Modern society increasingly depends on distributed software infrastructures such as the backend of popular Web portals, global E-science cyberinfrastructure, e-healthcare and e-governments. An application in these environments is typically organised into many components which communicate through message passing. Thus an application is naturally designed as a collection of interaction scenarios, or *multiparty sessions*, each following an interaction pattern, or *choreographic protocol*. The theories of multiparty session types [10] capture these two natural abstraction units, representing the situation where two or more multiparty sessions (choreographies) can interleave for a single point application, with each message clearly identifiable as belonging to a specific session.

This paper introduces a new behavioural theory which can reason about distributed processes globally controlled by multiple choreographic sessions. Typed behavioural theory has been one of the central topics of the study of the π-calculus throughout its history, for example, in order to reason about various encodings into the typed π-calculi [16, 18]. Our behavioural theory treats the mutual effects of multiple choreographic sessions and differs from any type-based bisimulations in the literature. Since our behavioural theory is based on the regulation of conversational behaviours of distributed components by global specifications, we call our bisimulation *globally governed bisimulation*.

To illustrate the key idea, we first explain the mechanisms of multiparty session types [10]. Let us consider a simple protocol where participant 1 sends a message of type `bool` to participant 2. To develop the code for this protocol, we start by specifying the *global type* [10] as $G_1 = 1 \to 2 \colon \langle \mathtt{bool} \rangle; \mathtt{end}$ where $\to$ signifies the flow of communication and `end` denotes protocol termination. With agreement on G_1 as a specification for participant 1 and participant 2, each program can be implemented separately. Then

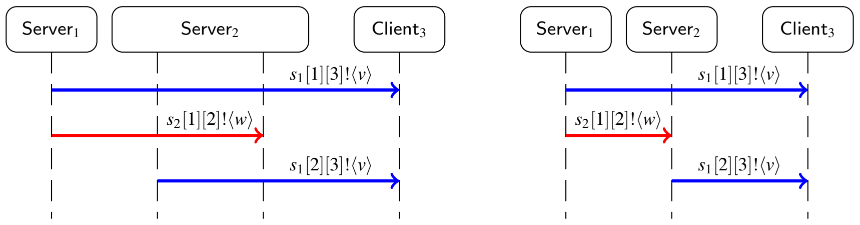

Server1
Server2
Client3
$s_1[1][3]!\langle v\rangle$
$s_2[1][2]!\langle w\rangle$
$s_1[2][3]!\langle v\rangle$
Server1
Server2
Client3
$s_1[1][3]!\langle v\rangle$
$s_2[1][2]!\langle w\rangle$
$s_1[2][3]!\langle v\rangle$

Fig. 1. Resource Managment Example: (a) before optimisation; (b) after optimisation

for type-checking, G_1 is *projected* into local session types: one local session type from 1's point of view, $[2]!\langle\texttt{bool}\rangle$ (output to 2 with `bool`-type), and another from 2's point of view, $[1]?\langle\texttt{bool}\rangle$ (input from 1 with `bool`-type), against which both processes are checked to be correct.

Now we explain how our new theory can reason about an optimisation of choreography interactions (a simplified usecase (UC.R2.13 "Acquire Data From Instrument") from [14]). Consider the two global types between three participants $(1,2,3)$:

$$G_a = 1 \to 3 : \langle ser\rangle.2 \to 3 : \langle ser\rangle.\texttt{end}, \quad G_b = 1 \to 2 : \langle sig\rangle.\texttt{end}$$

and a scenario in Figure 1(a) where Client3 (participant 3) uses two services, the first from Server1 (participant 1) and Server2 (participant 2), and Server1 sends an internal signal to Server2. The three parties belonging to these protocols are implemented as:

$$P_1 = a[1](x).b[1](y).x[3]!\langle v\rangle;y[2]!\langle w\rangle;\mathbf{0} \quad P_2 = a[2](x).\overline{b}[2](y).(y[1]?(z);\mathbf{0} \mid x[3]!\langle v\rangle;\mathbf{0})$$
$$P_3 = \overline{a}[3](x).x[1]?(z);x[2]?(y);\mathbf{0}$$

where session name a establishes the session corresponding to G_a. Client3 (P_3) initiates a session involving three processes as the third participant by $\overline{a}[3](x)$: Service1 (P_1) and Service2 (P_2) participate to the session $a[1](x)$ and $a[2](x)$, respectively. Similarly the session corresponding to G_b is established between Service1 and Service2.

Since from Client3, the internal signal is invisible, we optimise Server2 to a single thread to avoid an unnecessary thread creation as $R_2 = a[2](x).\overline{b}[2](y).y[1]?(z);x[3]!\langle v\rangle;\mathbf{0}$ in Figure 1(b). Note that both P_2 and R_2 are typable under G_a and G_b. Obviously, in the untyped setting, $P_1 \mid P_2$ and $P_1 \mid R_2$ are *not* bisimilar (by either synchronous or asynchronous semantics) since in P_2, the output action $x[3]!\langle v\rangle$ can be observed before the input action $y[1]?(z)$ happens. However, with the global constraints given by G_a and G_b, a service provided by Server2 is only available to Client3 after Server1 sends a signal to Server2, i.e. action $x[3]!\langle v\rangle$ can only happen after action $y[1]?(z)$ in P_2. Hence $P_1 \mid P_2$ and $P_1 \mid R_2$ are not distinguishable by Client3 and the thread optimisation of R_2 is correct.

On the other hand, if we change the global type G_a as:

$$G'_a = 2 \to 3 : \langle ser\rangle.1 \to 3 : \langle ser\rangle.\texttt{end}$$

then R_2 can perform both the output to Client3 and the input from Server1 concurrently since G'_a states that Client3 can receive the message from Server2 first. Hence $P_1 \mid P_2$ and $P_1 \mid R_2$ are no longer equivalent.

The key point to make this difference possible is to observe the behaviour of processes together with the information provided by the global types. The global types can

$$
\begin{array}{lllr|llr}
P & ::= & \overline{u}[\mathtt{p}](x).P & \text{Request} & \mid & \mathtt{if}\ e\ \mathtt{then}\ P\ \mathtt{else}\ Q & \text{Conditional} \\
 & \mid & u[\mathtt{p}](x).P & \text{Accept} & \mid & P \mid Q & \text{Parallel} \\
 & \mid & c[\mathtt{p}]!\langle e\rangle;P & \text{Sending} & \mid & \mathbf{0} & \text{Inaction} \\
 & \mid & c[\mathtt{p}]?(x);P & \text{Receiving} & \mid & (\nu\, n)P & \text{Hiding} \\
 & \mid & c[\mathtt{p}] \oplus l;P & \text{Selection} & \mid & \mu X.P & \text{Recursion} \\
 & \mid & c[\mathtt{p}]\&\{l_i : P_i\}_{i\in I} & \text{Branching} & \mid & X & \text{Variable} \\
u & ::= & x \mid a & \text{Identifier} & c & ::= s[\mathtt{p}] \mid x & \text{Session} \\
n & ::= & s \mid a & \text{Name} & v & ::= a \mid \mathtt{tt} \mid \mathtt{ff} \mid s[\mathtt{p}] & \text{Value} \\
e & ::= & v \mid x \mid e\ \mathtt{and}\ e' \mid e = e' \mid \ldots & & \text{Expression} & &
\end{array}
$$

Fig. 2. Syntax for synchronous multiparty session calculus

define additional knowledge about how the observer (the client in the above example) will collaborate with the observed processes so that different global types (i.e. global witnesses) can induce the different equivalences.

Contributions This paper introduces two kinds of typed bisimulations based on multiparty session types and systematically applies them to synchronous and three kinds of asynchronous session semantics. The first bisimulation is solely based on local (end-point) types defined without global information, hence it resembles the standard linearity-based bisimulation. The second one is a *globally governed session bisimilarity* which uses multiparty session types as information for a global witness. We prove that each coincides with a corresponding standard contextual equivalence [9] (Theorems 4.1, 5.1, 6.1, 6.2). The governed bisimulation gives more fine-grained equivalences than the locally typed bisimulation. We identify the condition when the two semantics exactly coincide (Theorem 5.2). Interestingly our theorem (Theorem 5.3) shows this difference appears only when processes are running under more than two interleaved global types. We show how the synchronous and the three kinds of asynchronous semantics are related with each other (Theorems 6.1, 6.2). We demonstrate the use of governed bisimulation through the running example, which is applicable to a thread optimisation of a real usecase from a large scale distributed system [14].

Due to space limitations, this paper introduces the theories for the synchronous semantics and later summarises the key elements of asynchronous semantics. The Appendix includes the full definitions for asynchronous semantics as well as the detailed proofs of the theorems, lemmas and propositions listed in the main sections. The Appendix also lists a full derivation of a usecase example from [14].

2 Synchronous Multiparty Session Calculus

This section defines a synchronous version of the multiparty session calculus. The syntax follows [3] except we eliminate queues for asynchronous communication. We chose synchrony since it allows the simplest formulations and definitions for demonstrating the key concepts of globally governed bisimulations. The summary of the asynchronous semantics is given in § 6.

Syntax Figure 2 defines the syntax for synchronous multiparty session calculus. u ranges over shared names $a, b, \ldots$ or variables $x, y, \ldots$. c ranges over session roles $s[\mathtt{p}]$

$$a[1](x).P_1 \mid \ldots \mid \overline{a}[n](x).P_n \longrightarrow (\nu\, s)(P_1\{s[1]/x\} \mid \ldots \mid P_n\{s[n]/x\}) \qquad \text{[Link]}$$

$$s[\mathtt{p}][\mathtt{q}]!\langle e\rangle;P \mid s[\mathtt{q}][\mathtt{p}]?(x);Q \longrightarrow P \mid Q\{v/x\} \quad (e \downarrow v) \qquad \text{[Comm]}$$

$$s[\mathtt{p}][\mathtt{q}] \oplus l_k;P \mid s[\mathtt{q}][\mathtt{p}]\&\{l_i : P_i\}_{i\in I} \longrightarrow P \mid P_k \quad k \in I \qquad \text{[Label]}$$

$$\mathtt{if}\ e\ \mathtt{then}\ P\ \mathtt{else}\ Q \longrightarrow P \quad (e \downarrow \mathtt{tt}) \qquad \mathtt{if}\ e\ \mathtt{then}\ P\ \mathtt{else}\ Q \longrightarrow Q \quad (e \downarrow \mathtt{ff}) \quad \text{[If]}$$

$$\frac{P \longrightarrow P'}{(\nu\, n)P \longrightarrow (\nu\, n)P'}\ \text{[Res]} \qquad \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q}\ \text{[Par]} \qquad \frac{P \equiv P' \longrightarrow Q' \equiv Q}{P \longrightarrow Q}\ \text{[Str]}$$

Fig. 3. Operational semantics for synchronous multiparty session calculus

or variables. Values $v, v', \ldots$ range over shared names, constants $\mathtt{tt}, \mathtt{ff}, \ldots$ or a channel with role $s[\mathtt{p}]$. Expressions $e, e', \ldots$ are either values, logical operations on expressions or name matching operations. We call $\mathtt{p}, \mathtt{p}', \mathtt{q}, \ldots$ (range over the natural numbers) the *participants*.

For the primitives for session initiation, $\overline{u}[\mathtt{p}](x).P$ initiates a new session through an identifier u (which represents a shared interaction point) with the other multiple participants, each of shape $u[\mathtt{p}](x)..Q_{\mathtt{q}}$ where $1 \leq \mathtt{q} \leq \mathtt{p} - 1$. The (bound) variable x is the channel used to do the communications. Session communications (communications that take place inside an established session) are performed using the next two pairs: the sending and receiving of a value and the selection and branching (where the former chooses one of the branches offered by the latter). The input/output operations specify the sender and the receiver, respectively. Hence $c[\mathtt{p}]!\langle e\rangle;P$ sends a value to $\mathtt{p}$; accordingly, $c[\mathtt{p}]?(x);P$ denotes the intention of receiving a value from the participant $\mathtt{p}$. The same holds for selection/branching. We call $s[\mathtt{p}]$ a *channel with role*: it represents the channel of the participant $\mathtt{p}$ in the session s. Process $\mathbf{0}$ is the inactive process. Other processes are standard. We say that a process is closed it does not contain free variables. We denote $\mathtt{fn}(P)/\mathtt{bn}(P)$ and $\mathtt{fv}(P)/\mathtt{bv}(P)$ for a set of free/bound names and free/bound variables, respectively. We use the standard structure rules (denoted by $\equiv$) including $\mu X.P \equiv P\{\mu X.P/X\}$.

Operational semantics Operational semantics of the calculus are defined in Figure 3. Rule [Link] defines synchronous session initiation. All session roles must be present to synchronously reduce each role $\mathtt{p}$ on a fresh session name $s[\mathtt{p}]$. Rules [Comm] is for sending a value to the corresponding receiving process where $e \downarrow v$ means expression e evaluates to value v. The interaction between selection and branching is defined via rule [Label]. Other rules are standard. We write $\twoheadrightarrow$ for $(\longrightarrow \cup \equiv)^*$.

3 Typing for Synchronous Multiparty Sessions

This section first defines types and then summarises a typing system for the synchronous multiparty session calculus.

3.1 Global and local types

Global types, ranged over by $G, G', \ldots$ describe the whole conversation scenario of a multiparty session as a type signature. Its grammar is given in Figure 4. The global type $\mathtt{p} \to \mathtt{q} : \langle U\rangle.G'$ says that participant $\mathtt{p}$ sends a message of type U to the participant $\mathtt{q}$ and then interactions described in G' take place. *Exchange types* $U, U', \ldots$ consist of *sorts*

$$
\begin{array}{rrcll}
\text{Global} & G & ::= & \mathtt{p} \to \mathtt{q} : \langle U \rangle.G' & \text{exchange} \\
 & & \mid & \mathtt{p} \to \mathtt{q} : \{l_i : G_i\}_{i\in I} & \text{branching} \\
 & & \mid & \mu\mathtt{t}.G & \text{recursion} \\
 & & \mid & \mathtt{t} & \text{variable} \\
 & & \mid & \mathtt{end} & \text{end} \\
\text{Exchange} & U & ::= & S \mid T & \\
\text{Sort} & S & ::= & \mathtt{bool} \mid \langle G \rangle &
\end{array}
\qquad
\begin{array}{rrcll}
\text{Local} & T & ::= & [\mathtt{p}]!\langle U \rangle;T & \text{send} \\
 & & \mid & [\mathtt{p}]?(U);T & \text{receive} \\
 & & \mid & [\mathtt{p}] \oplus \{l_i : T_i\}_{i\in I} & \text{selection} \\
 & & \mid & [\mathtt{p}]\&\{l_i : T_i\}_{i\in I} & \text{branching} \\
 & & \mid & \mu\mathtt{t}.T & \text{recursion} \\
 & & \mid & \mathtt{t} & \text{variable} \\
 & & \mid & \mathtt{end} & \text{end}
\end{array}
$$

Fig. 4. Global and local types

types $S, S', \ldots$ for values (either base types or global types), and *local session* types $T, T', \ldots$ for channels (defined in the next paragraph). Type $\mathtt{p} \to \mathtt{q} : \{l_i : G_i\}_{i\in I}$ says participant $\mathtt{p}$ sends one of the labels l_i to $\mathtt{q}$. If l_j is sent, interactions described in G_j take place. In both cases we assume $\mathtt{p} \neq \mathtt{q}$. Type $\mu\mathtt{t}.G$ is a recursive type, assuming type variables $(\mathtt{t}, \mathtt{t}', \ldots)$ are guarded in the standard way, i.e., type variables only appear under some prefix. We take an *equi-recursive* view of recursive types, not distinguishing between $\mu.G$ and its unfolding $G\{\mu\mathtt{t}.G/\mathtt{t}\}$.We assume that G in the grammar of sorts is closed, i.e., without free type variables. Type $\mathtt{end}$ represents the termination of the session.

Local types are defined in Figure 4 and correspond to the communication actions, representing sessions from the view-points of single participants. The *send type* $[\mathtt{p}]!\langle U \rangle;T$ expresses the sending to $\mathtt{p}$ of a value of type U, followed by the communications of T. The *selection type* $[\mathtt{p}] \oplus \{l_i : T_i\}_{i\in I}$ represents the transmission to $\mathtt{p}$ of a label l_i chosen in the set $\{l_i \mid i \in I\}$ followed by the communications described by T_i. The *receive* and *branching* are dual. Other types are the same as global types.

3.2 Two projections

The relation between global and local types is formalised by the standard projection function [10].

Definition 3.1 (Global projection and projection set). The projection of a global type G onto a participant $\mathtt{p}$ is defined by induction on G:

$$
\mathtt{p}' \to \mathtt{q} : \langle U \rangle.G\lceil\mathtt{p} = \begin{cases} [\mathtt{q}]!\langle U \rangle;G\lceil\mathtt{p} & \mathtt{p} = \mathtt{p}' \\ [\mathtt{p}']?(U);G\lceil\mathtt{p} & \mathtt{p} = \mathtt{q} \\ G\lceil\mathtt{p} & \text{otherwise} \end{cases}
\qquad
\mathtt{p}' \to \mathtt{q} : \{l_i : G_i\}_{i\in I}\lceil\mathtt{p} = \begin{cases} [\mathtt{q}] \oplus \{l_i : G_i\lceil\mathtt{p}\}_{i\in I} & \mathtt{p} = \mathtt{p}' \\ [\mathtt{p}']\&\{l_i : G_i\lceil\mathtt{p}\}_{i\in I} & \mathtt{p} = \mathtt{q} \\ G_1\lceil\mathtt{p} & \text{if } \forall j \in I.\ G_1\lceil\mathtt{p} = G_j\lceil\mathtt{p} \end{cases}
$$

$$
(\mu\mathtt{t}.G)\lceil\mathtt{p} = \begin{cases} \mu\mathtt{t}.(G\lceil\mathtt{p}) & \mathtt{p} \in G \\ \mathtt{end} & \text{otherwise} \end{cases}
\qquad \mathtt{t}\lceil\mathtt{p} = \mathtt{t} \quad \mathtt{end}\lceil\mathtt{p} = \mathtt{end}
$$

Then the *projection set* of $s : G$ is defined as $\mathtt{proj}(s : G) = \{s[\mathtt{p}] : G\lceil\mathtt{p} \mid \mathtt{p} \in \mathtt{roles}(G)\}$ where $\mathtt{roles}(G)$ denotes the set of the roles appearing in G.

We also need the following projection from a local type T to produce binary session types for defining the equivalence relations later.

Definition 3.2 (Local projection). The projection of a local type T onto a participant $\mathtt{p}$ is defined by induction on T:

$$
[\mathtt{p}]!\langle U \rangle;T\lceil\mathtt{q} = \begin{cases} !\langle U \rangle;T\lceil\mathtt{q} & \mathtt{q}{=}\,\mathtt{p} \\ T\lceil\mathtt{q} & \text{otherwise} \end{cases}
\qquad
[\mathtt{p}]?(U);T\lceil\mathtt{q} = \begin{cases} ?(U);T\lceil\mathtt{q} & \mathtt{q}{=}\,\mathtt{p} \\ T\lceil\mathtt{q} & \text{otherwise} \end{cases}
$$

$$[\mathtt{p}] \oplus \{l_i : T_i\}_{i\in I} \lceil \mathtt{q} = \begin{cases} \oplus\{l_i : T_i \lceil \mathtt{q}\}_{i\in I} & \mathtt{q} = \mathtt{p} \\ T_1 \lceil \mathtt{q} & \text{if } \forall i \in I. T_i \lceil \mathtt{q} = T_1 \lceil \mathtt{q} \end{cases}$$
$$[\mathtt{p}] \& \{l_i : T_i\}_{i\in I} \lceil \mathtt{q} = \begin{cases} \&\{l_i : T_i \lceil \mathtt{q}\}_{i\in I} & \mathtt{q} = \mathtt{p} \\ T_1 \lceil \mathtt{q} & \text{if } \forall i \in I. T_i \lceil \mathtt{q} = T_1 \lceil \mathtt{q} \end{cases}$$

The rest is similar as Definition 3.1.

The duality over the projected types are defined as: $\mathtt{end} = \overline{\mathtt{end}}$, $\mathtt{t} = \overline{\mathtt{t}}$, $\overline{\mu\mathtt{t}.T} = \mu\mathtt{t}.\overline{T}$, $\overline{!\langle U\rangle;T} = ?(U);\overline{T}$, $\overline{?(U);T} = !\langle U\rangle;\overline{T}$, $\overline{\oplus\{l_i : T_i\}_{i\in I}} = \&\{l_i : \overline{T_i}\}_{i\in I}$ and $\overline{\&\{l_i : T_i\}_{i\in I}} = \oplus\{l_i : \overline{T_i}\}_{i\in I}$. We note that if $\mathtt{p},\mathtt{q} \in \mathtt{roles}(G)$ then $(G\lceil\mathtt{p})\lceil\mathtt{q} = \overline{(G\lceil\mathtt{q})\lceil\mathtt{p}}$.

3.3 Typing system and its properties

The typing judgements for expressions and processes are of the shapes:

$$\Gamma \vdash e : S \quad \text{and} \quad \Gamma \vdash P \triangleright \Delta$$

where Γ is the standard environment which associates variables to sort types, shared names to global types and process variables to session environments; and Δ is the session environment which associates channels to session types. Formally we define:

$$\Gamma \ ::= \ \emptyset \ | \ \Gamma \cdot u : S \ | \ \Gamma \cdot X : \Delta \quad \text{and} \quad \Delta \ ::= \ \emptyset \ | \ \Delta \cdot s[\mathtt{p}] : T$$

assuming we can write $\Gamma \cdot u : S$ if $u \notin \mathtt{dom}(\Gamma)$. We extend this to a concatenation for typing environments as $\Delta \cdot \Delta' = \Delta \cup \Delta'$. We define coherency of session environments as follows:

Definition 3.3 (Coherency). Typing Δ is *coherent with respect to session s* (notation $\mathtt{co}(\Delta(s))$) if $\forall s[\mathtt{p}] : T_{\mathtt{p}}, s[\mathtt{q}] : T_{\mathtt{q}} \in \Delta$ with $\mathtt{p} \neq \mathtt{q}$ then $T_{\mathtt{p}}\lceil\mathtt{q} = \overline{T_{\mathtt{q}}\lceil\mathtt{p}}$. A typing Δ is *coherent* (notation $\mathtt{co}(\Delta)$) if it is coherent with respect to all s in its domain. We say that the typing judgement $\Gamma \vdash P \triangleright \Delta$ is *coherent* if $\mathtt{co}(\Delta)$.

The typing rules are essentially identical to the communication typing system for programs in [3] (since we do not require queues). We leave the rules in Figure 10 in Appendix A. The rest of the paper can be read without knowing the typing system.

3.4 Type soundness

Next we define the reduction semantics for local types. Since session environments represent the forthcoming communications, by reducing processes session environments can change. This can be formalised as in [3, 10] by introducing the notion of reduction of session environments, whose rules are:

1. $\{s[\mathtt{p}] : [\mathtt{q}]!\langle U\rangle;T \cdot s[\mathtt{q}] : [\mathtt{p}]?(U);T'\} \longrightarrow \{s[\mathtt{p}] : T \cdot s[\mathtt{q}] : T'\}$.
2. $\{s[\mathtt{p}] : [\mathtt{q}] \oplus \{l_i : T_i\}_{i\in I} \cdot s[\mathtt{q}] : [\mathtt{p}]\&\{l_j : T'_j\}_{j\in J}\} \longrightarrow \{s[\mathtt{p}] : T_k \cdot s[\mathtt{q}] : T'_k\}$ $I \subseteq J, k \in I$.
3. $\Delta \cup \Delta' \longrightarrow \Delta \cup \Delta''$ if $\Delta' \longrightarrow \Delta''$.

We write $\twoheadrightarrow = \longrightarrow^*$. Note that $\Delta \twoheadrightarrow \Delta'$ is non-deterministic (i.e. not always confluent) by the second rule. The following theorem is proved in [3].

Theorem 3.1 (Subject reduction). *Let $\Gamma \vdash P \triangleright \Delta$ be coherent and $P \twoheadrightarrow P'$ then $\Gamma \vdash P' \triangleright \Delta'$ is coherent with $\Delta \twoheadrightarrow \Delta'$.*

4 Synchronous Multiparty Session Semantics

This section presents a typed behavioural theory for the synchronous multiparty sessions. The typed bisimulation is defined with the labelled transition system (LTS) between a tuple of the environments (Γ, Δ), which controls behaviours of untyped processes. The constraint given by the environment LTSs, which accurately captures interactions in the presence of multiparty sessions, is at the heart of our typed semantics. In the next section, this relation is extended to more fine-grained LTSs with the additional information of a global witness. We first define a typed bisimulation theory, and then show that the typed bisimulation coincides with typed reduction-based congruency. The definitions of this section will be used for the next section.

4.1 Labelled transition system

Labels We use the following labels $(\ell, \ell', ...)$:

$$\begin{array}{rcl} \ell & ::= & \overline{a}[A](s) \mid a[A](s) \mid s[\mathtt{p}][\mathtt{q}]!\langle v \rangle \mid s[\mathtt{p}][\mathtt{q}]!(a) \\ & \mid & s[\mathtt{p}][\mathtt{q}]!(s'[\mathtt{p}']) \mid s[\mathtt{p}][\mathtt{q}]?\langle v \rangle \mid s[\mathtt{p}][\mathtt{q}] \oplus l \mid s[\mathtt{p}][\mathtt{q}]\& l \mid \tau \end{array}$$

A role set A is a set of multiparty session types roles. Labels $\overline{a}[A](s)$ and $a[A](s)$ define the accept and request of a fresh session s by roles in set A respectively. Actions on session channels are denoted with labels $s[\mathtt{p}][\mathtt{q}]!\langle v \rangle$ and $s[\mathtt{p}][\mathtt{q}]?\langle v \rangle$ for output and input of value v from $\mathtt{p}$ to $\mathtt{q}$ on session s. Bound output values can be shared channels or session roles (delegation). $s[\mathtt{p}][\mathtt{q}] \oplus l$ and $s[\mathtt{p}][\mathtt{q}]\& l$ define the selection and branching respectively. Label τ is the standard hidden transition.

Dual label definition is used to define the parallel rule in the label transition system:

$$s[\mathtt{p}][\mathtt{q}]!\langle v \rangle \asymp s[\mathtt{q}][\mathtt{p}]?\langle v \rangle \quad s[\mathtt{p}][\mathtt{q}]!(v) \asymp s[\mathtt{q}][\mathtt{p}]?\langle v \rangle \quad s[\mathtt{p}][\mathtt{q}] \oplus l \asymp s[\mathtt{q}][\mathtt{p}]\& l$$

Dual labels are input and output (resp. selection and branching) on the same session channel and on complementary roles. For example, in $s[\mathtt{p}][\mathtt{q}]!\langle v \rangle$ and $s[\mathtt{q}][\mathtt{p}]?\langle v \rangle$, role $\mathtt{p}$ sends to $\mathtt{q}$ and role $\mathtt{q}$ receives from $\mathtt{p}$. Another important definition for the session initiation is the notion of the complete role set. We say the role set A is *complete with respect to n* if $n = \max(A)$ and $A = \{1, 2, \ldots, n\}$. The complete role set means that all global protocol participants are present in the set. For example, $\{1,3,4\}$ is not complete, but $\{1,2,3,4\}$ is. We use $\mathtt{fn}(\ell)$ and $\mathtt{bn}(\ell)$ to denote a set of free and bound names in ℓ and set $\mathtt{n}(\ell) = \mathtt{bn}(\ell) \cup \mathtt{fn}(\ell)$.

Labelled transition system for processes Figure 5 gives the untyped labelled transition system. Rules $\langle \text{Req} \rangle$ and $\langle \text{Acc} \rangle$ define the accept and request actions for a fresh session s on role $\{\mathtt{p}\}$. Rules $\langle \text{Send} \rangle$ and $\langle \text{Rcv} \rangle$ give the send and receive respectively for value v from role $\mathtt{p}$ to role $\mathtt{q}$ in session s. Similarly rules $\langle \text{Sel} \rangle$ and $\langle \text{Bra} \rangle$ define selecting and branching labels.

The last three rules are for collecting and synchronising the multiparty participants together. Rule $\langle \text{AccPar} \rangle$ accumulates the accept participants and records them into role set A. Rule $\langle \text{ReqPar} \rangle$ accumulates the accept participants and the request participant into role set A. Note that the request action role set always includes the maximum role number among the participants. Finally, rule $\langle \text{TauS} \rangle$ checks that a role set is complete, thus a new session can be created under the τ-action (synchronisation). Other rules are standard. See Example 4.1. We write $\Longrightarrow$ for the reflexive and transitive closure of $\longrightarrow$, $\overset{\ell}{\Longrightarrow}$ for the transitions $\Longrightarrow\overset{\ell}{\longrightarrow}\Longrightarrow$ and $\overset{\hat{\ell}}{\Longrightarrow}$ for $\overset{\ell}{\Longrightarrow}$ if $\ell \neq \tau$ otherwise $\Longrightarrow$.

$$\langle\text{Req}\rangle\quad \overline{a}[\mathtt{p}](x).P \xrightarrow{\overline{a}[\{\mathtt{p}\}](s)} P\{s[\mathtt{p}]/x\} \qquad \langle\text{Acc}\rangle\quad a[\mathtt{p}](x).P \xrightarrow{a[\{\mathtt{p}\}](s)} P\{s[\mathtt{p}]/x\}$$

$$\langle\text{Send}\rangle\quad s[\mathtt{p}][\mathtt{q}]!\langle e\rangle;P \xrightarrow{s[\mathtt{p}][\mathtt{q}]!\langle v\rangle} P \quad (e\downarrow v) \qquad \langle\text{Rcv}\rangle\quad s[\mathtt{p}][\mathtt{q}]?(x);P \xrightarrow{s[\mathtt{p}][\mathtt{q}]?\langle v\rangle} P\{v/x\}$$

$$\langle\text{Sel}\rangle\quad s[\mathtt{p}][\mathtt{q}]\oplus l;P \xrightarrow{s[\mathtt{p}][\mathtt{q}]\oplus l} P \qquad \langle\text{Bra}\rangle\quad s[\mathtt{p}][\mathtt{q}]\&\{l_i : P_i\}_{i\in I} \xrightarrow{s[\mathtt{p}][\mathtt{q}]\& l_k} P_k$$

$$\langle\text{Tau}\rangle\ \frac{P \xrightarrow{\ell} P' \quad Q \xrightarrow{\ell'} Q' \quad \ell \asymp \ell'}{P \mid Q \xrightarrow{\tau} (\nu\ \mathtt{bn}(\ell)\cap\mathtt{bn}(\ell'))(P' \mid Q')} \qquad \langle\text{Par}\rangle\ \frac{P \xrightarrow{\ell} P' \quad \mathtt{bn}(\ell)\cap\mathtt{fn}(Q)=\emptyset}{P \mid Q \xrightarrow{\ell} P' \mid Q}$$

$$\langle\text{Res}\rangle\ \frac{P \xrightarrow{\ell} P' \quad n\notin\mathtt{fn}(\ell)}{(\nu\ n)P \xrightarrow{\ell} (\nu\ n)P'} \quad \langle\text{OpenS}\rangle\ \frac{P \xrightarrow{s[\mathtt{p}][\mathtt{q}]!\langle s'[\mathtt{p}']\rangle} P'}{(\nu\ s')P \xrightarrow{s[\mathtt{p}][\mathtt{q}]!(s'[\mathtt{p}'])} P'} \quad \langle\text{OpenN}\rangle\ \frac{P \xrightarrow{s[\mathtt{p}][\mathtt{q}]!\langle a\rangle} P'}{(\nu\ a)P \xrightarrow{s[\mathtt{p}][\mathtt{q}]!(a)} P'}$$

$$\langle\text{Alpha}\rangle\ \frac{P \equiv_\alpha P' \quad P' \xrightarrow{\ell} Q'}{P \xrightarrow{\ell} Q} \qquad \langle\text{AcPar}\rangle\ \frac{P_1 \xrightarrow{a[A](s)} P_1' \quad P_2 \xrightarrow{a[A'](s)} P_2' \quad A\cap A'=\emptyset}{P_1 \mid P_2 \xrightarrow{a[A\cup A'](s)} P_1' \mid P_2'}$$

$$\langle\text{ReqPar}\rangle\ \frac{P_1 \xrightarrow{a[A](s)} P_1' \quad P_2 \xrightarrow{\overline{a}[A'](s)} P_2' \quad A\cap A'=\emptyset,\ A\cup A' \text{ not complete w.r.t } \max(A')}{P_1 \mid P_2 \xrightarrow{\overline{a}[A\cup A'](s)} P_1' \mid P_2'}$$

$$\langle\text{TauS}\rangle\ \frac{P_1 \xrightarrow{a[A](s)} P_1' \quad P_2 \xrightarrow{\overline{a}[A'](s)} P_2' \quad A\cap A'=\emptyset,\ A\cup A' \text{ complete w.r.t } \max(A')}{P_1 \mid P_2 \xrightarrow{\tau} (\nu\ s)(P_1' \mid P_2')}$$

We omit the synmetric case of $\langle\text{Par}\rangle$ and conditonals.

Fig. 5. Labelled transition system for processes

Typed labelled transition relation We define the typed LTS on the basis of the untyped one. This is realised by introducing the definition of an environment labelled transition system, defined in Figure 6. $(\Gamma,\Delta) \xrightarrow{\ell} (\Gamma',\Delta')$ means that an environment (Γ,Δ) allows an action to take place, and the resulting environment is (Γ',Δ').

The basic intuition for this graph definition is that observables on session channels occur when the corresponding endpoint is not present in the linear typing environment Δ, and the type of an action's object respects the environment (Γ,Δ). In the case when new names are created or received the environment (Γ,Δ) is extended.

The first rule says that reception of a message via a is possible when a's type $\langle G\rangle$ is recorded into Γ and the resulting session environment records projected types from G $(\{s[i] : G\lceil i\}_{i\in A})$. The second rule is for the send of a message via a and it is dual to the first rule. The next four rules are free value output, bound name output, free value input and name input. Rest of rules are free session output, bound session output, and session input as well as selection and branching rules. The bound session output records a set of session types $s'[\mathtt{p}_i]$ at opened session s'. The final rule $(\ell=\tau)$ follows the reduction rules for linear session environment defined in § 3.4 ($\Delta=\Delta'$ is the case for the reduction at hidden sessions). Note that if Δ already contains destination $(s[\mathtt{q}])$, the environment cannot perform the visible action, but only the final τ-action.

The typed LTS requires that a process can perform an untyped action ℓ and that its typing environment (Γ,Δ) can match the action ℓ.

Definition 4.1 (Typed transition). *Typed transition* relation is defined as $\Gamma_1 \vdash P_1 \triangleright \Delta_1 \xrightarrow{\ell} \Gamma_2 \vdash P_2 \triangleright \Delta_2$ if (1) $P_1 \xrightarrow{\ell} P_2$ and (2) $(\Gamma_1,\Delta_1) \xrightarrow{\ell} (\Gamma_2,\Delta_2)$ with $\Gamma_i \vdash P_i \triangleright \Delta_i$.

$$\begin{array}{rrcl}
\Gamma(a) = \langle G \rangle, s \text{ fresh implies} & (\Gamma,\Delta) & \xrightarrow{a[A](s)} & (\Gamma,\Delta\cdot\{s[i]:G\lceil i\}_{i\in A}) \\
\Gamma(a) = \langle G \rangle, s \text{ fresh implies} & (\Gamma,\Delta) & \xrightarrow{\overline{a}[A](s)} & (\Gamma,\Delta\cdot\{s[i]:G\lceil i\}_{i\in A}) \\
\Gamma \vdash v:U, s[\mathtt{q}] \notin \mathtt{dom}(\Delta) \text{ implies} & (\Gamma,\Delta\cdot s[\mathtt{p}]:[\mathtt{q}]!\langle U\rangle;T) & \xrightarrow{s[\mathtt{p}][\mathtt{q}]!\langle v\rangle} & (\Gamma,\Delta\cdot s[\mathtt{p}]:T) \\
s[\mathtt{q}] \notin \mathtt{dom}(\Delta), a \notin \mathtt{dom}(\Gamma) \text{ implies} & (\Gamma,\Delta\cdot s[\mathtt{p}]:[q]!\langle U\rangle;T) & \xrightarrow{s[\mathtt{p}][\mathtt{q}]!(a)} & (\Gamma\cdot a:U,\Delta\cdot s[\mathtt{p}]:T) \\
\Gamma \vdash v:U, s[\mathtt{q}] \notin \mathtt{dom}(\Delta) \text{ implies} & (\Gamma,\Delta\cdot s[\mathtt{p}]:[\mathtt{q}]?(U);T) & \xrightarrow{s[\mathtt{p}][\mathtt{q}]?\langle v\rangle} & (\Gamma,\Delta\cdot s[\mathtt{p}]:T) \\
a \notin \mathtt{dom}(\Gamma), s[\mathtt{q}] \notin \mathtt{dom}(\Delta) \text{ implies} & (\Gamma,\Delta\cdot s[\mathtt{p}]:[\mathtt{q}]?(U);T) & \xrightarrow{s[\mathtt{p}][\mathtt{q}]?\langle a\rangle} & (\Gamma\cdot a:U,\Delta\cdot s[\mathtt{p}]:T) \\
s[\mathtt{q}] \notin \mathtt{dom}(\Delta) \text{ implies} & (\Gamma,\Delta\cdot s'[\mathtt{p}']:T'\cdot s[\mathtt{p}]:[\mathtt{q}]!\langle T'\rangle;T) & \xrightarrow{s[\mathtt{p}][\mathtt{q}]!\langle s'[\mathtt{p}']\rangle} & (\Gamma,\Delta\cdot s[\mathtt{p}]:T) \\
s[\mathtt{q}] \notin \mathtt{dom}(\Delta) \text{ implies} & (\Gamma,\Delta\cdot s[\mathtt{p}]:[\mathtt{q}]!\langle T'\rangle;T) & \xrightarrow{s[\mathtt{p}][\mathtt{q}]!(s'[\mathtt{p}'])} & (\Gamma,\Delta\cdot s[\mathtt{p}]:T\cdot\{s'[\mathtt{p}_i]:T_i\}) \\
s[\mathtt{q}], s'[\mathtt{p}'] \notin \mathtt{dom}(\Delta) \text{ implies} & (\Gamma,\Delta\cdot s[\mathtt{p}]:[\mathtt{q}]?(T');T) & \xrightarrow{s[\mathtt{p}][\mathtt{q}]?\langle s'[\mathtt{p}']\rangle} & (\Gamma,\Delta\cdot s'[\mathtt{p}']:T'\cdot s[\mathtt{p}]:T) \\
s[\mathtt{q}] \notin \mathtt{dom}(\Delta) \text{ implies} & (\Gamma,\Delta\cdot s[\mathtt{p}]:[\mathtt{q}]\oplus\{l_i:T_i\}_{i\in I}) & \xrightarrow{s[\mathtt{p}][\mathtt{q}]\oplus l_k} & (\Gamma,\Delta\cdot s[\mathtt{p}]:T_k) \\
s[\mathtt{q}] \notin \mathtt{dom}(\Delta) \text{ implies} & (\Gamma,\Delta\cdot s[\mathtt{p}]:[\mathtt{q}]\&\{l_i:T_i\}_{i\in I}) & \xrightarrow{s[\mathtt{p}][\mathtt{q}]\& l_k} & (\Gamma,\Delta\cdot s[\mathtt{p}]:T_k) \\
\Delta \longrightarrow \Delta' \quad \text{or} \quad \Delta = \Delta' \text{ implies} & (\Gamma,\Delta) & \xrightarrow{\tau} & (\Gamma,\Delta')
\end{array}$$

Fig. 6. Labelled Transition Relation for Environments

4.2 Synchronous multiparty behavioural theory

We define the typed relation as the binary relation over typed processes.

Definition 4.2 (Typed relation). We define a relation $\mathscr{R}$ as *typed relation* if it relates two closed, coherent typed terms $\Gamma \vdash P_1 \triangleright \Delta_1 \mathscr{R} \Gamma \vdash P_2 \triangleright \Delta_2$. We often write $\Gamma \vdash P_1 \triangleright \Delta_1 \mathscr{R} P_2 \triangleright \Delta_2$.

Next we define the *barb* [1]: we write $\Gamma \vdash P \triangleright \Delta \downarrow_{s[\mathtt{p}][\mathtt{q}]}$ if $P \equiv (\nu\ \tilde{a}\tilde{s})(s[\mathtt{p}][\mathtt{q}]!\langle v\rangle;R \mid Q)$ with $s \notin \tilde{s}$ and $s[\mathtt{q}] \notin \mathtt{dom}(\Delta)$; and $\Gamma \vdash P \triangleright \Delta \downarrow_a$ if $P \equiv (\nu\ \tilde{a}\tilde{s})(\overline{a}[n](s).R \mid Q)$ with $a \notin \tilde{a}$. Then we write m for either a or $s[\mathtt{p}][\mathtt{q}]$. We define $\Gamma \vdash P \triangleright \Delta \Downarrow_m$ if there exists Q such that $P \twoheadrightarrow Q$ and $\Gamma \vdash Q \triangleright \Delta' \downarrow_m$.

The context is defined as:

$$\begin{array}{rcl}
C & ::= & - \mid C \mid P \mid P \mid C \mid (\nu\ n)C \mid \mathtt{if}\ e\ \mathtt{then}\ C\ \mathtt{else}\ C' \mid \mu X.C \mid \\
& & s!\langle v\rangle;C \mid s?(x);C \mid s\oplus l;C \mid s\&\{l_i:C_i\}_{i\in I} \mid \overline{a}(x).C \mid a(x).C
\end{array}$$

$C[P]$ substitutes process P for each hole $(-)$ in context C.

Definition 4.3 (Linear Environment Convergence). We write $\Delta_1 \rightleftharpoons \Delta_2$ if there exists Δ such that $\Delta_1 \twoheadrightarrow \Delta$ and $\Delta_2 \twoheadrightarrow \Delta$.

We now define the contextual congruence based on the barb and [9].

Definition 4.4 (Reduction congruence). A typed relation $\mathscr{R}$ is *reduction congruence* if it satisfies the following conditions for each $\Gamma \vdash P_1 \triangleright \Delta_1 \mathscr{R} P_2 \triangleright \Delta_2$ with $\Delta_1 \rightleftharpoons \Delta_2$.

1. $\Gamma \vdash P_1 \triangleright \Delta_1 \Downarrow_m$ iff $\Gamma \vdash P_2 \triangleright \Delta_2 \Downarrow_m$
2. Whenever $\Gamma \vdash P_1 \triangleright \Delta_1 \mathscr{R} P_2 \triangleright \Delta_2$ holds, $P_1 \twoheadrightarrow P_1'$ implies $P_2 \twoheadrightarrow P_2'$ such that $\Gamma \vdash P_1' \triangleright \Delta_1' \mathscr{R} P_2' \triangleright \Delta_2'$ holds with $\Delta_1' \rightleftharpoons \Delta_2'$.
3. For all closed context C, such that $\Gamma \vdash C[P_1'] \triangleright \Delta_1'$ and $\Gamma \vdash C[P_2'] \triangleright \Delta_2'$ where $\Delta_1' \rightleftharpoons \Delta_2'$, $\Gamma \vdash C[P_1] \triangleright \Delta_1' \mathscr{R} \Gamma \vdash C[P_2] \triangleright \Delta_2'$.

The union of all reduction congruence relations is denoted as $\cong^s$.

Definition 4.5 (Synchronous multiparty session bisimulation). A typed relation $\mathscr{R}$ over closed processes is a (weak) *synchronous multiparty session bisimulation* or often a *synchronous bisimulation* if, whenever $\Gamma \vdash P_1 \triangleright \Delta_1 \mathscr{R} P_2 \triangleright \Delta_2$, it holds:

1. $\Gamma \vdash P_1 \triangleright \Delta_1 \xrightarrow{\ell} \Gamma' \vdash P_1' \triangleright \Delta_1'$ implies $\Gamma \vdash P_2 \triangleright \Delta_2 \overset{\hat{\ell}}{\Longrightarrow} \Gamma' \vdash P_2' \triangleright \Delta_2'$ such that $\Gamma' \vdash P_1' \triangleright \Delta_1' \mathscr{R} P_2' \triangleright \Delta_2'$.
2. The symmetric case.

The maximum bisimulation exists which we call *synchronous bisimilarity*, denoted by $\approx^s$. We sometimes leave environments implicit, writing e.g. $P \approx^s Q$. We also write $\approx$ for untyped synchronous bisimilarity which is defined by the untyped LTS in Figure 5 but without the environment LTS in Figure 6.

Lemma 4.1. *If $\Gamma \vdash P_1 \triangleright \Delta_1 \approx^s P_2 \triangleright \Delta_2$ then $\Delta_1 \rightleftharpoons \Delta_2$.*

Proof. The proof uses the co-induction method and can be found in Appendix C.2. □

Theorem 4.1 (Soundness and completeness). $\cong^s = \approx^s$.

Proof. The proof is a simplification of the proof of Theorem 5.1 in Appendix C.6. □

Example 4.1 (Synchronous multiparty bisimulation). We use the running example from § 1. First we explain the LTS for session initialisation from Figure 5. First by $\langle\text{Acc}\rangle$ and $\langle\text{Req}\rangle$, we have:

$$P_1 \xrightarrow{a[\{1\}](s_1)} P_1' = b[1](y).s_1[1][3]!\langle v\rangle;y[2]!\langle w\rangle;\mathbf{0}$$
$$P_2 \xrightarrow{a[\{2\}](s_1)} P_2' = \overline{b}[2](y).(y[1]?(z);\mathbf{0} \mid s_1[2][3]!\langle v\rangle;\mathbf{0}) \qquad P_3 \xrightarrow{\overline{a}[\{3\}](s_1)} P_3' = s_1[3][1]?(z);s_1[3][2]?(y);\mathbf{0}$$

with

$$\Gamma\cdot v : U \vdash P_1' \triangleright s_1[1] : [3]!\langle U\rangle;\mathtt{end}$$
$$\Gamma\cdot v : U \vdash P_2' \triangleright s_1[2] : [3]!\langle U\rangle;\mathtt{end}$$
$$\Gamma \vdash P_3' \triangleright s_1[3] : [1]?(U);[2]?(U);\mathtt{end}$$

By $\langle\text{AccPar}\rangle$, we have $P_1 \mid P_2 \xrightarrow{a[\{1,2\}](s_1)} P_1' \mid P_2'$. We have another possible initialisation: $P_1 \mid P_3 \xrightarrow{\overline{a}[\{1,3\}](s_1)} P_1' \mid P_3'$. From both of them, if we compose another process, the set $\{1,2,3\}$ becomes complete so that by synchronisation $\langle\text{TauS}\rangle$,

$$\Gamma \vdash P_1 \mid P_2 \mid P_3 \triangleright \emptyset \xrightarrow{\tau} (\nu\, s_1)(P_1' \mid P_2' \mid P_3') \triangleright \emptyset$$

Further we have:

$$\Gamma \vdash P_1' \mid P_2' \triangleright \emptyset \xrightarrow{\tau}$$
$$(\nu\, s_2)(s_1[1][3]!\langle v\rangle;s_2[1][2]!\langle w\rangle;\mathbf{0} \mid s_2[2][1]?(z);\mathbf{0} \mid s_1[2][3]!\langle v\rangle;\mathbf{0}) = Q_1 \triangleright \Delta$$

with $\Delta_0 = s_1[1] : [3]!\langle U\rangle;\mathtt{end}\cdot s_1[2] : [3]!\langle U\rangle;\mathtt{end}$. Then

$$\Gamma \vdash Q_1 \mid P_3' \triangleright \Delta_0 \cdot s_1[3] : [1]?(U);[2]?(U);\mathtt{end} \approx^s \mathbf{0} \triangleright \emptyset$$

since $(\Gamma,\Delta) \stackrel{\ell}{\not\longrightarrow}$ for any $\ell \neq \tau$. However by the untyped synchronous bisimulation, $Q_1 \mid P_3' \not\approx \mathbf{0}$ since, e.g. $Q_1 \mid P_3' \xrightarrow{s_1[1][3]!\langle v\rangle}$.

5 Globally Governed Behavioural Theory

We introduce the semantics for globally governed behavioural theory. In the previous section, the local typing (Δ) constrains the untyped LTS to give rise to a local typed LTS. In a multiparty distributed environment, communications follow the global protocol, which controls both an observed process and its observer. The local typing is not sufficient to maintain the consistency of transitions of a process with respect to a global protocol. In this section we refine the environment LTS with a *global environment* E to give a more fine-grained control over the LTS of the processes.

5.1 Global environments and configurations

We define a *global environment* $(E, E', ...)$ as a mapping from session name s to global types G.

$$E \ ::= \ E \cdot s : G \ \mid \ \emptyset$$

The projection definition is extended to include E as $\mathtt{proj}(E) = \bigcup_{s:G \in E} \mathtt{proj}(s : G)$.

We define a labelled reduction relation over global environments which corresponds to $\Delta \longrightarrow \Delta'$ defined in § 3.4. We use the labels $\lambda \in \{s : \mathtt{p} \to \mathtt{q} : U, s : \mathtt{p} \to \mathtt{q} : l\}$ to annotate reductions over global environments. We define $\mathtt{out}(\lambda)$ and $\mathtt{inp}(\lambda)$ as $\mathtt{out}(s : \mathtt{p} \to \mathtt{q} : U) = \mathtt{out}(s : \mathtt{p} \to \mathtt{q} : l) = \mathtt{p}$ and as $\mathtt{inp}(s : \mathtt{p} \to \mathtt{q} : U) = \mathtt{inp}(s : \mathtt{p} \to \mathtt{q} : l) = \mathtt{q}$ and $\mathtt{p} \in \ell$ if $\mathtt{p} \in \mathtt{out}(\ell) \cup \mathtt{inp}(\ell)$. We often omit the label λ by writing $\longrightarrow$ for $\xrightarrow{\lambda}$ and $\longrightarrow^*$ for $(\xrightarrow{\lambda})^*$. The first rule is the axiom for the input and output interaction between two parties; the second rule is for the choice; the third and forth rules formulate the case that the action λ can be performed under $\mathtt{p} \to \mathtt{q}$ if $\mathtt{p}$ and $\mathtt{q}$ are not related to the participants in λ; and the fifth rule is a congruent rule.

$$\frac{}{\{s : \mathtt{p} \to \mathtt{q} : \langle U \rangle.G\} \xrightarrow{s:\mathtt{p}\to\mathtt{q}:U} \{s : G\}} \qquad \frac{}{\{s : \mathtt{p} \to \mathtt{q} : \{l_i : G_i\}_{i \in I}\} \xrightarrow{s:\mathtt{p}\to\mathtt{q}:l_k} \{s : G_k\}}$$

$$\frac{\{s : G\} \xrightarrow{\lambda} \{s : G'\} \quad \mathtt{p}, \mathtt{q} \notin \lambda}{\{s : \mathtt{p} \to \mathtt{q} : \langle U \rangle.G\} \xrightarrow{\lambda} \{s : \mathtt{p} \to \mathtt{q} : \langle U \rangle.G'\}}$$

$$\frac{\{s : G_i\} \xrightarrow{\lambda} \{s : G_i'\} \quad i \in I, \quad \mathtt{p}, \mathtt{q} \notin \lambda}{\{s : \mathtt{p} \to \mathtt{q} : \{l_i : G_i\}_{i \in I}\} \xrightarrow{\lambda} \{s : \mathtt{p} \to \mathtt{q} : \{l_i : G_i'\}_{i \in I}\}} \qquad \frac{E \xrightarrow{\lambda} E'}{E \cdot E_0 \xrightarrow{\lambda} E' \cdot E_0}$$

As a simple example of the above LTS, consider $s : \mathtt{p} \to \mathtt{q} : \langle U_1 \rangle.\mathtt{p}' \to \mathtt{q}' : \{l_1 : \mathtt{end}, l_2 : \mathtt{p}' \to \mathtt{q}' : \langle U_2 \rangle.\mathtt{end}\}$. Since $\mathtt{p}, \mathtt{q}, \mathtt{p}', \mathtt{q}'$ are pairwise distinct, we can apply the second and third rules to obtain:

$$s : \mathtt{p} \to \mathtt{q} : \langle U_1 \rangle.\mathtt{p}' \to \mathtt{q}' : \{l_1 : \mathtt{end}, l_2 : \mathtt{p}' \to \mathtt{q}' : \langle U_2 \rangle.\mathtt{end}\} \xrightarrow{s:\mathtt{p}'\to\mathtt{q}':l_1} s : \mathtt{p} \to \mathtt{q} : \langle U_1 \rangle.\mathtt{end}$$

Next we introduce the *governance judgement* which controls the behaviour of processes by the global environment.

Definition 5.1 (Governance judgement). *Let $\Gamma \vdash P \triangleright \Delta$ be coherent. We write $E, \Gamma \vdash P \triangleright \Delta$ if $\exists E' \cdot E \longrightarrow^* E'$ and $\Delta \subseteq \mathtt{proj}(E')$.*

$$[\text{Acc}]\,\frac{\Gamma \vdash a : \langle G\rangle \quad (\Gamma,\Delta_1) \xrightarrow{a[A](s)} (\Gamma,\Delta_2)}{(E,\Gamma,\Delta_1) \xrightarrow{a[A](s)} (E\cdot s : G,\Gamma,\Delta_2)} \qquad [\text{Req}]\,\frac{\Gamma \vdash a : \langle G\rangle \quad (\Gamma,\Delta_1) \xrightarrow{\overline{a}[A](s)} (\Gamma,\Delta_2)}{(E,\Gamma,\Delta_1) \xrightarrow{\overline{a}[A](s)} (E\cdot s : G,\Gamma,\Delta_2)}$$

$$[\text{Out}]\,\frac{\Gamma \vdash v : U \quad (\Gamma,\Delta_1) \xrightarrow{s[\mathtt{p}][\mathtt{q}]!\langle v\rangle} (\Gamma,\Delta_2) \quad E_1 \xrightarrow{s:\mathtt{p}\to\mathtt{q}:U} E_2}{(E_1,\Gamma,\Delta_1) \xrightarrow{s[\mathtt{p}][\mathtt{q}]!\langle v\rangle} (E_2,\Gamma,\Delta_2)} \qquad [\text{In}]\,\frac{(\Gamma,\Delta_1) \xrightarrow{s[\mathtt{p}][\mathtt{q}]?\langle v\rangle} (\Gamma\cdot v : U,\Delta_2) \quad E_1 \xrightarrow{s:\mathtt{q}\to\mathtt{p}:U} E_2}{(E_1,\Gamma,\Delta_1) \xrightarrow{s[\mathtt{p}][\mathtt{q}]?\langle v\rangle} (E_2,\Gamma\cdot v : U,\Delta_2)}$$

$$[\text{ResN}]\,\frac{\begin{array}{c}(\Gamma,\Delta_1) \xrightarrow{s[\mathtt{p}][\mathtt{q}]!(a)} (\Gamma\cdot a : \langle G\rangle,\Delta_2)\\ E_1 \xrightarrow{s:\mathtt{q}\to\mathtt{p}:\langle G\rangle} E_2\end{array}}{(E_1,\Gamma,\Delta_1) \xrightarrow{s[\mathtt{p}][\mathtt{q}]!(a)} (E_2,\Gamma\cdot a : \langle G\rangle,\Delta_2)} \qquad [\text{ResS}]\,\frac{\begin{array}{c}(\Gamma,\Delta_1) \xrightarrow{s[\mathtt{p}][\mathtt{q}]!(s'[\mathtt{p}'])} (\Gamma,\Delta_2\cdot\{s'[\mathtt{p}_i] : T_i\})\\ E_1 \xrightarrow{s:\mathtt{q}\to\mathtt{p}:T} E_2 \quad \cdot\forall i.G\lceil \mathtt{p}_i = T_i\end{array}}{(E_1,\Gamma,\Delta_1) \xrightarrow{s[\mathtt{p}][\mathtt{q}]!(s'[\mathtt{p}'])} (E_2\cdot s' : G,\Gamma,\Delta_2\cdot\{s'[\mathtt{p}_i] : T_i\})}$$

$$[\text{Sel}]\,\frac{(\Gamma,\Delta_1) \xrightarrow{s[\mathtt{p}][\mathtt{q}]\oplus l} (\Gamma,\Delta_2) \quad E_1 \xrightarrow{s:\mathtt{p}\to\mathtt{q}:l} E_2}{(E_1,\Gamma,\Delta_1) \xrightarrow{s[\mathtt{p}][\mathtt{q}]\oplus l} (E_2,\Gamma,\Delta_2)} \qquad [\text{Bra}]\,\frac{(\Gamma,\Delta_1) \xrightarrow{s[\mathtt{p}][\mathtt{q}]\& l} (\Gamma,\Delta_2) \quad E_1 \xrightarrow{s:\mathtt{q}\to\mathtt{p}:l} E_2}{(E_1,\Gamma,\Delta_1) \xrightarrow{s[\mathtt{p}][\mathtt{q}]\& l} (E_2,\Gamma,\Delta_2)}$$

$$[\text{Tau}]\,\frac{(\Delta_1 = \Delta_2,\ E_1 = E_2) \vee (\Delta_1 \xrightarrow{\lambda} \Delta_2,\ E_1 \xrightarrow{\lambda} E_2)}{(E_1,\Gamma,\Delta_1) \xrightarrow{\tau} (E_2,\Gamma,\Delta_2)} \qquad [\text{Inv}]\,\frac{E_1 \longrightarrow^* E_1' \quad (E_1',\Gamma_1,\Delta_1) \xrightarrow{\ell} (E_2,\Gamma_2,\Delta_2)}{(E_1,\Gamma_1,\Delta_1) \xrightarrow{\ell} (E_2,\Gamma_2,\Delta_2)}$$

Fig. 7. The LTS for the environment configuations

The global environment E records the knowledge of both the environment (Δ) of the observed process P and the environment of its *observer*. The side conditions ensure that E is coherent with Δ: there exist E' reduced from E whose projection should cover the environment of P (since E should include the observer's information together with the observed process information recorded into Δ).

Next we define the LTS for well-formed environment configurations.

Definition 5.2 (Environment configuration). *We write* (E,Γ,Δ) *if* $\exists E'\cdot E \longrightarrow^* E'$ *and* $\Delta \subseteq \mathtt{proj}(E')$.

Figure 7 defines a LTS over environment configurations that refines the LTS over environments (i.e $(\Gamma,\Delta) \xrightarrow{\ell} (\Gamma',\Delta')$) in § 4.1.

Each rule requires a corresponding environment transition (Figure 6 in § 4.1) and a corresponding labelled global environment transition in order to control a transition following the global protocol. [Acc] is the rule for accepting a session initialisation so that it creates a new mapping $s : G$ which matches Γ in a governed environment E. [Req] is the rule for requesting a new session and it is dual to [Acc].

The next seven rules are the transition relations on session channels and we assume the condition $\mathtt{proj}(E_1) \supseteq \Delta$ to ensure the base action of the environment matches one in a global environment. [Out] is a rule for the output where the type of the value and the action of (Γ,Δ) meets those in E. [In] is a rule for the input and dual to [Out]. [ResN] is a scope opening rule for a name so that the environment can perform the corresponding type $\langle G\rangle$ of a. [ResS] is a scope opening rule for a session channel which creates a set of mappings for the opened session channel s' corresponding to the LTS of the environment. [Sel] and [Bra] are the rules for selection and branching, which is similar to [Out] and [In]. In [Tau] rule, we refined the reduction relation on Δ in § 3.4 as follows:

1. $\{s[\mathtt{p}] : [\mathtt{q}]!\langle U\rangle;T \cdot s[\mathtt{q}] : [\mathtt{p}]?(U);T'\} \xrightarrow{s:\mathtt{p}\to\mathtt{q}:U} \{s[\mathtt{p}] : T \cdot s[\mathtt{q}] : T'\}$.
2. $\{s[\mathtt{p}] : [\mathtt{q}]\oplus\{l_i : T_i\}_{i\in I} \cdot s[\mathtt{q}] : [\mathtt{p}]\&\{l_j : T_j'\}_{j\in J}\} \xrightarrow{s:\mathtt{p}\to\mathtt{q}:l_k} \{s[\mathtt{p}] : T_k \cdot s[\mathtt{q}] : T_k'\}\ I \subseteq J, k \in I$.
3. $\Delta \cup \Delta' \xrightarrow{\lambda} \Delta \cup \Delta''$ if $\Delta' \xrightarrow{\lambda} \Delta''$.

[Inv] is the key rule: the global environment E_1 reduces to E'_1 to perform the observer's actions, hence the observed process can perform the action w.r.t. E'_1. Hereafter we write $\longrightarrow$ for $\xrightarrow{\tau}$.

Example 5.1 (LTS for environment configuration). Let $E = s : \mathtt{p} \to \mathtt{q} : \langle U \rangle.\mathtt{p} \to \mathtt{q} : \langle U \rangle.G$, $\Gamma = v : U$ and $\Delta = s[\mathtt{p}] : [\mathtt{q}]!\langle U \rangle; T_{\mathtt{p}}$ with $G\lceil \mathtt{p} = T_{\mathtt{p}}$, $G\lceil \mathtt{q} = T_{\mathtt{q}}$ and $\mathtt{roles}(G) = \{\mathtt{p},\mathtt{q}\}$. Then (E,Γ,Δ) is an environment configuration since if $E \longrightarrow E'$ then $\mathtt{proj}(E') \supset \Delta$ because $E \overset{s:\mathtt{p}\to\mathtt{q}:U}{\longrightarrow} s : \mathtt{p} \to \mathtt{q} : \langle U \rangle.G$, $\mathtt{proj}(s : \mathtt{p} \to \mathtt{q} : \langle U \rangle.G) = s[\mathtt{p}] : [\mathtt{q}]!\langle U \rangle; T_{\mathtt{p}} \cdot s[\mathtt{q}] : [\mathtt{p}]?(U); T_{\mathtt{q}}$ and $\mathtt{proj}(s : \mathtt{p} \to \mathtt{q} : \langle U \rangle.G) \supset \Delta$. Then we can apply [Out] to $s : \mathtt{p} \to \mathtt{q} : \langle U \rangle.G \overset{s:\mathtt{p}\to\mathtt{q}:U}{\longrightarrow} s : G$ and $(\Gamma, s[\mathtt{p}] : [\mathtt{q}]!\langle U \rangle; T_{\mathtt{p}}) \overset{s[\mathtt{p}][\mathtt{q}]!\langle v \rangle}{\longrightarrow} (\Gamma, s[\mathtt{p}] : T_{\mathtt{p}})$ to obtain $(s : \mathtt{p} \to \mathtt{q} : \langle U \rangle.G, \Gamma, \Delta) \overset{s[\mathtt{p}][\mathtt{q}]!\langle v \rangle}{\longrightarrow} (s : G, \Gamma, s[\mathtt{p}] : T_{\mathtt{p}})$. By this and $E \longrightarrow s : \mathtt{p} \to \mathtt{q} : \langle U \rangle.G$, using [Inv], we can obtain $(E,\Gamma,\Delta) \overset{s[\mathtt{p}][\mathtt{q}]!\langle v \rangle}{\longrightarrow} (s : G, \Gamma, s[\mathtt{p}] : T_{\mathtt{p}})$, as required.

5.2 Governed reduction-closed congruency

To define the reduction-closed congruency, we first refine the barb, which is controlled by the global witness where observables of a configuration are defined with the global environment of the observer.

$$\begin{array}{rl}
(E,\Gamma,\Delta \cdot s[\mathtt{p}] : [\mathtt{q}]!\langle U \rangle; T) \downarrow_{s[\mathtt{p}][\mathtt{q}]} & \text{if } s[\mathtt{q}] \notin \mathtt{dom}(\Delta), \exists E' \cdot E \longrightarrow^* E' \overset{s:\mathtt{p}\to\mathtt{q}:U}{\longrightarrow}, \Delta \subseteq \mathtt{proj}(E') \\
(E,\Gamma,\Delta \cdot s[\mathtt{p}] : [\mathtt{q}] \oplus \{l_i : T_i\}_{i\in I}) \downarrow_{s[\mathtt{p}][\mathtt{q}]} & \text{if } s[\mathtt{q}] \notin \mathtt{dom}(\Delta), \exists E' \cdot E \longrightarrow^* E' \overset{s:\mathtt{p}\to\mathtt{q}:l_k}{\longrightarrow}, k \in I, \Delta \subseteq \mathtt{proj}(E'), \\
(E,\Gamma,\Delta) \downarrow_a & \text{if } a \in \mathtt{dom}(\Gamma)
\end{array}$$

We write $(\Gamma,\Delta,E) \Downarrow_m$ if $(\Gamma,\Delta,E) \longrightarrow^* (\Gamma,\Delta',E')$ and $(\Gamma,\Delta',E') \downarrow_m$.

Let us write $T_1 \sqsubseteq T_2$ if the syntax tree of T_2 includes T_1. For example, $[\mathtt{q}]?(U'); T \sqsubseteq [\mathtt{p}]!\langle U \rangle; [\mathtt{q}]?(U'); T$. Then we define: $E_1 \sqcup E_2 = \{E_i(s) \mid E_j(s) \sqsubseteq E_i(s), i,j \in \{1,2\}, i \neq j\} \cup E_1 \setminus \mathtt{dom}(E_2) \cup E_2 \setminus \mathtt{dom}(E_1)$. As an example of $E_1 \sqcup E_2$, let us define:

$$\begin{aligned}
E_1 &= s_1 : \mathtt{p} \to \mathtt{q} : \langle U_1 \rangle.\mathtt{p}' \to \mathtt{q}' : \langle U_2 \rangle.\mathtt{p} \to \mathtt{q} : \langle U_3 \rangle.\mathtt{end} \cdot s_2 : \mathtt{p} \to \mathtt{q} : \langle W_2 \rangle.\mathtt{end} \\
E_2 &= s_1 : \mathtt{p} \to \mathtt{q} : \langle U_3 \rangle.\mathtt{end} \cdot s_2 : \mathtt{p}' \to \mathtt{q}' : \langle W_1 \rangle.\mathtt{p} \to \mathtt{q} : \langle W_2 \rangle.\mathtt{end}
\end{aligned}$$

Then $E_1 \sqcup E_2 = \mathtt{p} \to \mathtt{q} : \langle U_1 \rangle.\mathtt{p}' \to \mathtt{q}' : \langle U_2 \rangle.\mathtt{p} \to \mathtt{q} : \langle U_3 \rangle.\mathtt{end} \cdot s_2 : \mathtt{p}' \to \mathtt{q}' : \langle W_1 \rangle.\mathtt{p} \to \mathtt{q} : \langle W_2 \rangle.\mathtt{end}$.

The behavioural relation with respects to a global whiteness is defined below.

Definition 5.3 (Configuration relation). The relation $\mathscr{R}$ is a *configuration relation* between two configurations $E_1, \Gamma \vdash P_1 \triangleright \Delta_1$ and $E_2, \Gamma \vdash P_2 \triangleright \Delta_2$, written $E_1 \sqcup E_2, \Gamma \vdash P \triangleright \Delta_1 \mathrel{\mathscr{R}} P_2 \triangleright \Delta_2$ if $E_1 \sqcup E_2$ is defined.

We note:

Proposition 5.1 (Decidability). *(1) Given E_1 and E_2, a problem whether $E_1 \sqcup E_2$ is defined or not is decidable and if it is defined, the calculation of $E_1 \sqcup E_2$ terminates; and (2) Given E, a set $\{E' \mid E \longrightarrow^* E'\}$ is finite.*

Proof. (1) since $T_1 \sqsubseteq T_2$ is a syntactic tree inclusion, it is reducible to a problem to check the isomorphism between two types. This problem is decidable [19]. *(2)* the global LTS has one-to-one correspondence with the LTS of global automata in [5] whose reachability set is finite. □

Now we define the governed typed transition relation for processes.

Definition 5.4 (Global configuration transition). We write $E_1,\Gamma \vdash P_1 \triangleright \Delta_1 \xrightarrow{\ell} E_2,\Gamma' \vdash P_2 \triangleright \Delta_2$ if $E_1,\Gamma \vdash P_1 \triangleright \Delta_1$, $P_1 \xrightarrow{\ell} P_2$ and $(E_1,\Gamma,\Delta_1) \xrightarrow{\ell} (E_2,\Gamma',\Delta_2)$.

Below states that the configuration LTS preserves the well-formedness.

Proposition 5.2 (Invariants).

1. *$(E_1,\Gamma,\Delta_1) \xrightarrow{\ell} (E_2,\Gamma_2,\Delta_2)$ implies that (E_2,Γ_2,Δ_2) is an environment configuration.*
2. *If $\Gamma \vdash P \triangleright \Delta$ and $P \longrightarrow P'$ with $\mathsf{co}(\Delta)$, then $E,\Gamma \vdash P \triangleright \Delta \longrightarrow E,\Gamma \vdash P' \triangleright \Delta'$ and $\mathsf{co}(\Delta')$.*

Proof. The proof for Part 1 can be found in Appendix C.4. Part 2 is verified by simple transitions using [Tau] in Figure 7. The $\mathsf{co}(\Delta')$ part follows the subject reduction Theorem 3.1. □

The definition of the reduction congruence for governance follows. Below we define $E,\Gamma \vdash P \triangleright \Delta \Downarrow_n$ if $P \Downarrow_m$ and $(E,\Gamma,\Delta) \Downarrow_m$.

Definition 5.5 (Governed reduction congruence). A configuration relation $\mathscr{R}$ is *governed reduction congruence* if $E,\Gamma \vdash P_1 \triangleright \Delta_1 \mathrel{\mathscr{R}} P_2 \triangleright \Delta_2$ then

1. $E,\Gamma \vdash P_1 \triangleright \Delta_1 \Downarrow_n$ if and only if $E,\Gamma \vdash P_2 \triangleright \Delta_2 \Downarrow_n$
2. $P_1 \twoheadrightarrow P_1'$ if and only if $P_2 \twoheadrightarrow P_2'$ and $E,\Gamma \vdash P_1' \triangleright \Delta_1' \mathrel{\mathscr{R}} P_2' \triangleright \Delta_2'$
3. For all closed context C, such that $E,\Gamma \vdash C[P_1] \triangleright \Delta_1'$ and $E,\Gamma \vdash C[P_2] \triangleright \Delta_2'$ then $E,\Gamma \vdash C[P_1] \triangleright \Delta_1' \mathrel{\mathscr{R}} C[P_2] \triangleright \Delta_2'$.

The union of all governed reduction congruence relations is denoted as $\cong^s_g$.

5.3 Globally governed bisimulation and its properties

This subsection introduces the globally governed bisimulation relation definition and studies its main properties.

Definition 5.6 (Globally governed bisimulation). A configuration relation $\mathscr{R}$ is a *globally governed weak bisimulation* (or governed bisimulation) if whenever $E,\Gamma \vdash P_1 \triangleright \Delta_1 \mathrel{\mathscr{R}} P_2 \triangleright \Delta_2$, it holds:

1. $E,\Gamma \vdash P_1 \triangleright \Delta_1 \xrightarrow{\ell} E_1',\Gamma' \vdash P_1' \triangleright \Delta_1'$ implies $E,\Gamma \vdash P_2 \triangleright \Delta_2 \xRightarrow{\hat{\ell}} E_2',\Gamma' \vdash P_2' \triangleright \Delta_2'$ such that $E_1' \sqcup E_2',\Gamma' \vdash P_1' \triangleright \Delta_1' \mathrel{\mathscr{R}} P_2' \triangleright \Delta_2'$.
2. The symmetric case.

The maximum bisimulation exists which we call *governed bisimilarity*, denoted by $\approx^s_g$. We sometimes leave environments implicit, writing e.g. $P \approx^s_g Q$.

Lemma 5.1. *$\approx^s_g$ is congruent.*

Proof. The proof is by a case analysis on the context structure. The interesting case is the parallel composition, which uses Proposition 5.2. See Appendix C.5. □

Lemma 5.2. $\cong_g^s \subseteq \approx_g^s$

Proof. The proof follows the facts that bisimulation has a stratifying definition (the proof method uses the technique from [1]) and that the external actions can always be tested (the technique from [6]). The proof can be found in Appendix C.6. $\square$

By Lemmas 5.1 and 5.2, we have:

Theorem 5.1 (Sound and completeness). $\approx_g^s = \cong_g^s$.

Further the relationship between $\approx^s$ and $\approx_g^s$ is given as follows.

Theorem 5.2. *If for all E, $E,\Gamma \vdash P_1 \triangleright \Delta_1 \approx_g^s P_2 \triangleright \Delta_2$ then $\Gamma \vdash P_1 \triangleright \Delta_1 \approx^s \Gamma \vdash P_2 \triangleright \Delta_2$. Also if $\Gamma \vdash P_1 \triangleright \Delta_1 \approx^s \Gamma \vdash P_2 \triangleright \Delta_2$, then for all E, $E,\Gamma \vdash P_1 \triangleright \Delta_1 \approx_g^s P_2 \triangleright \Delta_2$.*

Proof. See Appendix C.7. $\square$

To justify the above theorem, consider the following processes:

$$P_1 = s_1[1][3]!\langle v\rangle; s_2[1][2]!\langle w\rangle; \mathbf{0} \mid s_1[2][3]!\langle v\rangle; s_2[2][1]?(x); s_2[2][3]!\langle x\rangle; \mathbf{0}$$
$$P_2 = s_1[3][v]!\langle \mathbf{0}\rangle; \mid s_2[1][2]!\langle w\rangle; \mathbf{0} \mid s_1[2][3]!\langle v\rangle; s_2[2][1]?(x); s_2[2][3]!\langle x\rangle; \mathbf{0}$$

then we have $P_1 \approx^s P_2$. By the above theorem, we expect that for all E, we have $E,\Gamma \vdash P_1 \triangleright \Delta_1$ and $E,\Gamma \vdash P_2 \triangleright \Delta_2$ then $E \vdash P_1 \approx_g^s P_2$. This is in fact true because the possible E that can type P_1 and P_2 are:

$$E_1 = s_1 : 1 \to 3 : \langle U\rangle.2 \to 3 : \langle U\rangle.\mathtt{end} \cdot s_2 : 1 \to 2 : \langle W\rangle.2 \to 3 : \langle W\rangle.\mathtt{end}$$
$$E_2 = s_1 : 2 \to 3 : \langle U\rangle.1 \to 3 : \langle U\rangle.\mathtt{end} \cdot s_2 : 1 \to 2 : \langle W\rangle.2 \to 3 : \langle W\rangle.\mathtt{end}$$

Note that all E that are instances up-to weakening (see Lemma C.2) are E_1 and E_2.

To clarify the difference between $\approx^s$ and $\approx_g^s$, we introduce the notion of a *simple multiparty process* defined in [10]. A simple process contains only a single session so that it satisfies the progress property as proved in [10]. Formally a process P is *simple* when it is typable with a type derivation where the session typing in the premise and the conclusion of each prefix rule is restricted to at most a single session (i.e. any $\Gamma \vdash P \triangleright \Delta$ which appears in a derivation, Δ contains at most one session channel in its domain, see [10]). Thus each prefixed sub-term in a simple process has only a unique session. Since there is no interleaving of sessions in simple processes, the difference between $\approx^s$ and $\approx_g^s$ disappears.

Theorem 5.3 (Coincidence). *Assume P_1 and P_2 are simple. If $\exists E \cdot E,\Gamma \vdash P_1 \triangleright \Delta_1 \approx_g^s P_2 \triangleright \Delta_2$ then $\Gamma \vdash P_1 \triangleright \Delta_1 \approx^s P_2 \triangleright \Delta_2$.*

Proof. The proof follows the fact that if P is simple and $\Gamma \vdash P \triangleright \Delta \xrightarrow{\ell} P' \triangleright \Delta'$ then $\exists E \cdot E,\Gamma \vdash P \triangleright \Delta \xrightarrow{\ell} P' \triangleright \Delta'$ to continue that if P_1, P_2 are simple and $\exists E \cdot E,\Gamma \vdash P_1 \triangleright \Delta_1 \approx_g^s P_2 \triangleright \Delta_2$ then $\forall E, E,\Gamma \vdash P_1 \triangleright \Delta_1 \approx_g^s P_2 \triangleright \Delta_2$. The result then comes by applying Lemma 5.2. The details of the proof are in the Appendix C.8. $\square$

To justify the above theorem, consider simple processes:

$$\begin{aligned} P_1 &= s[1][2]?(x); s[1][3]!\langle x\rangle; \mathbf{0} \mid s[2][1]!\langle v\rangle; \mathbf{0} \\ P_2 &= s[1][3]!\langle v\rangle; \mathbf{0} \end{aligned}$$

It holds that for $E = s : 2 \to 1 : \langle U\rangle.1 \to 3 : \langle U\rangle.\mathtt{end}$ then $E \vdash P_1 \approx^s_g P_2$. We can easily reason $P_1 \approx^s P_2$.

Example 5.2 (Governed bisimulation). Recall the example from § 1 and Example 4.1. Q_1 is the process corresponding to Example 4.1, while Q_2 has a parallel thread instead of the sequential composition (this corresponds to $P_1 \mid R_2$ in § 1).

$$\begin{aligned} Q_1 &= s_1[1][3]!\langle v\rangle; s_2[1][2]!\langle w\rangle; \mathbf{0} \mid s_2[2][1]?(x); \mathbf{0} \mid s_1[2][3]!\langle v\rangle; \mathbf{0} \\ Q_2 &= s_1[1][3]!\langle v\rangle; s_2[1][2]!\langle w\rangle; \mathbf{0} \mid s_2[2][1]?(x); s_1[2][3]!\langle v\rangle; \mathbf{0} \end{aligned}$$

Assume:

$$\begin{aligned} \Gamma &= v : S \cdot w : S \\ \Delta &= s_1[1] : [3]!\langle S\rangle; \mathtt{end} \cdot s_1[2] : [3]!\langle S\rangle; \mathtt{end} \cdot s_2[1] : [2]!\langle S\rangle; \mathtt{end} \cdot s_2[2] : [1]?(S); \mathtt{end} \end{aligned}$$

Then we have $\Gamma \vdash Q_1 \triangleright \Delta$ and $\Gamma \vdash Q_2 \triangleright \Delta$. Now assume the two global witnesses as:

$$\begin{aligned} E_1 &= s_1 : 1 \to 3 : \langle S\rangle.2 \to 3 : \langle S\rangle.\mathtt{end} \cdot s_2 : 1 \to 2 : \langle S\rangle.\mathtt{end} \\ E_2 &= s_1 : 2 \to 3 : \langle S\rangle.1 \to 3 : \langle S\rangle.\mathtt{end} \cdot s_2 : 1 \to 2 : \langle S\rangle.\mathtt{end} \end{aligned}$$

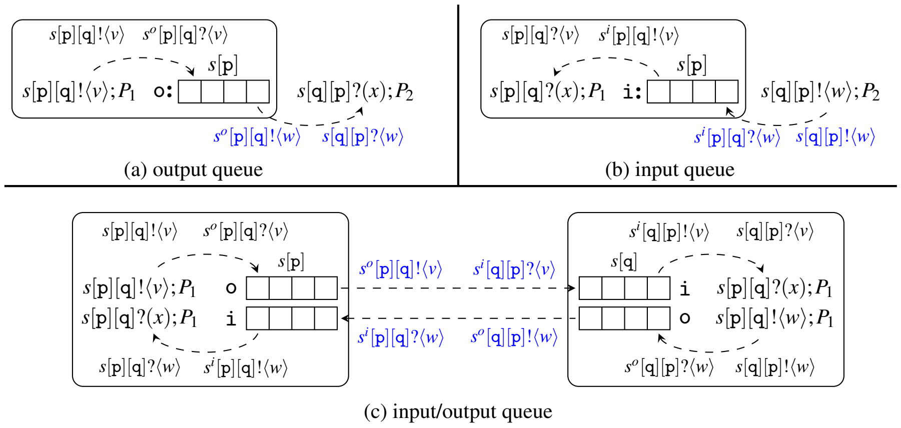
Then the projection of E_1 and E_2 are given as:

$$\begin{aligned} \mathtt{proj}(E_1) &= s_1[1] : [3]!\langle S\rangle; \mathtt{end} \cdot s_1[2] : [3]!\langle S\rangle; \mathtt{end} \cdot s_1[3] : [1]?(S); [2]?(S); \mathtt{end} \\ &\quad s_2[1] : [2]!\langle S\rangle; \mathtt{end} \cdot s_2[2] : [1]?(S); \mathtt{end} \cdot \\ \mathtt{proj}(E_2) &= s_1[1] : [3]!\langle S\rangle; \mathtt{end} \cdot s_1[2] : [3]!\langle S\rangle; \mathtt{end} \cdot s_1[3] : [2]?(S); [1]?(S); \mathtt{end} \cdot \\ &\quad s_2[1] : [2]!\langle S\rangle; \mathtt{end} \cdot s_2[2] : [1]?(S); \mathtt{end} \end{aligned}$$

with $\Delta \subset \mathtt{proj}(E_1)$ and $\Delta \subset \mathtt{proj}(E_2)$. The reader should note that the difference between E_1 and E_2 is the type of the participant 3 at s_1.

By definition, we can write: $E_i, \Gamma \vdash Q_1 \triangleright \Delta$ and $E_i, \Gamma \vdash Q_2 \triangleright \Delta$ for $i = 1, 2$. Both processes are well-formed global configurations under both witnesses. Now we can observe $\Gamma \vdash Q_1 \triangleright \Delta \xrightarrow{s[2][3]!\langle v\rangle} \Gamma \vdash Q_1' \triangleright \Delta'$ but $\Gamma \vdash Q_2 \triangleright \Delta \overset{s[2][3]!\langle v\rangle}{\not\longrightarrow}$. Hence $\Gamma \vdash Q_1 \triangleright \Delta \not\approx^s Q_2 \triangleright \Delta$. By the same argument, we have: $E_2, \Gamma \vdash Q_1 \triangleright \Delta \not\approx^s_g Q_2 \triangleright \Delta$. On the other hand, since E_1 *forces* to wait the action $s[2][3]!\langle v\rangle$, $E_1, \Gamma \vdash Q_1 \triangleright \Delta \overset{s[2][3]!\langle v\rangle}{\not\longrightarrow}$. Hence Q_1 and Q_2 are bisimilar under E_1, i.e. $E_1, \Gamma \vdash Q_1 \triangleright \Delta \approx^s_g Q_2 \triangleright \Delta$. This concludes the optimisation is correct.

The above example for the thread transformation is the *minimum* to demonstrate a difference between $\approx^s_g$ and $\approx^s$. This discipline can be applied to general situations where multiple agents need to interact following a global specification. We tested the real world usecase UC.R2.13 "Acquire Data From Instrument" from the Ocean Observatories Initiative (OOI) [14] Use Case library (Release 2). In this usecase, a user program (U) is connected to the Integrated Observatory Network (ION), which provides the infrastructure between users and remote sensing instruments. The user requests, via



$s[\mathrm{p}][\mathrm{q}]!\langle v\rangle$ $s^o[\mathrm{p}][\mathrm{q}]?\langle v\rangle$
$s[\mathrm{p}]$
$s[\mathrm{p}][\mathrm{q}]!\langle v\rangle;P_1$ o:
$s[\mathrm{q}][\mathrm{p}]?(x);P_2$
$s^o[\mathrm{p}][\mathrm{q}]!\langle w\rangle$ $s[\mathrm{q}][\mathrm{p}]?\langle w\rangle$
(a) output queue
$s[\mathrm{p}][\mathrm{q}]?\langle v\rangle$ $s^i[\mathrm{p}][\mathrm{q}]!\langle v\rangle$
$s[\mathrm{p}]$
$s[\mathrm{p}][\mathrm{q}]?(x);P_1$ i:
$s[\mathrm{q}][\mathrm{p}]!\langle w\rangle;P_2$
$s^i[\mathrm{p}][\mathrm{q}]?\langle w\rangle$ $s[\mathrm{q}][\mathrm{p}]!\langle w\rangle$
(b) input queue
$s[\mathrm{p}][\mathrm{q}]!\langle v\rangle$ $s^o[\mathrm{p}][\mathrm{q}]?\langle v\rangle$
$s[\mathrm{p}]$
$s[\mathrm{p}][\mathrm{q}]!\langle v\rangle;P_1$ o
$s[\mathrm{p}][\mathrm{q}]?(x);P_1$ i
$s[\mathrm{p}][\mathrm{q}]?\langle w\rangle$ $s^i[\mathrm{p}][\mathrm{q}]!\langle w\rangle$
$s^o[\mathrm{p}][\mathrm{q}]!\langle v\rangle$ $s^i[\mathrm{q}][\mathrm{p}]?\langle v\rangle$
$s^i[\mathrm{p}][\mathrm{q}]?\langle w\rangle$ $s^o[\mathrm{q}][\mathrm{p}]!\langle w\rangle$
$s^i[\mathrm{q}][\mathrm{p}]!\langle v\rangle$ $s[\mathrm{q}][\mathrm{p}]?\langle v\rangle$
$s[\mathrm{q}]$
i $s[\mathrm{p}][\mathrm{q}]?(x);P_1$
o $s[\mathrm{p}][\mathrm{q}]!\langle w\rangle;P_1$
$s^o[\mathrm{q}][\mathrm{p}]?\langle w\rangle$ $s[\mathrm{q}][\mathrm{p}]!\langle w\rangle$
(c) input/output queue

Fig. 8. Three asynchronous semantics

an ION agent service (`A`), the acquisition of data from an instrument (`I`). In the implementation, the ION agent (`A`) is realised by two sub ION agents (`A1` and `A2`) which internally interact and synchronise together. We are able to reason that the behaviour of `A1` and `A2` is equated by `A` by $\approx_g^s$ applying the thread transformation in Example 5.2. See Appendix I.

6 Asynchronous Globally Governed Behavioural Theory

This section summarises the three kinds of asynchronous semantics and demonstrates that the governed theory can be extended smoothly to these semantics. The formal relationship between governed synchronous and asynchronous bisimulations is also proved. Due to the space limitation, all definitions and proofs are left to Appendix.

6.1 Three asynchronous semantics

Our asynchronous formalism given below aims to model *asynchronous order-preserving communication* over session as *asynchronous session communication*, by extending the synchronous calculus in § 2 with *message queues*. The asynchronous session semantics has been used in most of the literature on multiparty session types [3, 10] since it models mechanisms of distributed systems more accurately. We consider the following three kinds of asynchronous semantics with respect to the places where the queues are located: at *each* session endpoint $s[\mathrm{p}]$, there is either (a) an output queue $s[\mathrm{p}][\mathtt{o}:\vec{h}]$ (with elements $\vec{h}$) (b) an input queue $s[\mathrm{p}][\mathtt{i}:\vec{h}]$; and (c) input and output message queues $s[\mathrm{p}][\mathtt{i}:\vec{h}]|s[\mathrm{p}][\mathtt{o}:\vec{h}']$ (with elements $\vec{h}$ and $\vec{h}'$). Formally, the syntax of processes P in Figure 2 is extended with corresponding queues with element $h ::= [\mathrm{p}](v) \mid [\mathrm{p}]l$.

These three models are illustrated in Figure 8. (a) depicts the output queue is located in the process side. A message v is first enqueued by sender process $s[\mathrm{p}][\mathrm{q}]!\langle v\rangle;P$ in the local output queue at endpoint $s[\mathrm{p}]$, which intuitively represents a communication pipe extending from the sender's locality to the receiver's. Hence the external observer can observe the output from the queue and input to the receiver, noting that enqueuing actions within a location are local to each process and are therefore invisible (τ-actions) to external observers. We use the color blue to denote the observable actions in Figure 8.

(b) is its dual. (c) depicts the two endpoints of one session connection. The communication medium (i.e. the connection) is responsible for transporting the message from the sender's locality to the receiver's, formalised as a message transfer from the sender's output queue (at $s[\mathtt{p}]$) to the receiver's input queue. For the receiver process, the message can only be received after this transfer takes place. In (c), both dequeuing and enqueuing actions within a location are local to each process and invisible to external observers.

6.2 Three asynchronous bisimulations

We extend the label ℓ in § 4.1 to include those to output (annotated by o)/input (annotated by i) queues:

$$\begin{array}{rl} \ell = \cdots & \mid\ s^o[\mathtt{p}][\mathtt{q}]!\langle v\rangle \ \mid\ s^o[\mathtt{p}][\mathtt{q}]!(v) \ \mid\ s^o[\mathtt{p}][\mathtt{q}]?\langle v\rangle \ \mid\ s^o[\mathtt{p}][\mathtt{q}]\oplus l \ \mid\ s^o[\mathtt{p}][\mathtt{q}]\& l \\ & \mid\ s^i[\mathtt{p}][\mathtt{q}]!\langle v\rangle \ \mid\ s^i[\mathtt{p}][\mathtt{q}]!(v) \ \mid\ s^i[\mathtt{p}][\mathtt{q}]?\langle v\rangle \ \mid\ s^i[\mathtt{p}][\mathtt{q}]\oplus l \ \mid\ s^i[\mathtt{p}][\mathtt{q}]\& l \end{array}$$

The duality relations which is the key to define τ-actions via the rule $\langle\text{Tau}\rangle$ in Figure 5 are extended to (a) $\asymp_o$ and (c) $\asymp_{io}$ from $\asymp$ in § 4.1 as follows:

$$\begin{array}{r|ll} s[\mathtt{p}][\mathtt{q}]!\langle v\rangle \asymp_o s^o[\mathtt{p}][\mathtt{q}]?\langle v\rangle & s[\mathtt{p}][\mathtt{q}]!\langle v\rangle \asymp_{io} s^o[\mathtt{p}][\mathtt{q}]?\langle v\rangle & s^o[\mathtt{p}][\mathtt{q}]!\langle v\rangle \asymp_{io} s^i[\mathtt{p}][\mathtt{q}]?\langle v\rangle \\ s^o[\mathtt{p}][\mathtt{q}]!\langle v\rangle \asymp_o s[\mathtt{q}][\mathtt{p}]?\langle v\rangle & s^i[\mathtt{p}][\mathtt{q}]!\langle v\rangle \asymp_{io} s[\mathtt{q}][\mathtt{p}]?\langle v\rangle & s^o[\mathtt{p}][\mathtt{q}]\oplus l \asymp_{io} s^i[\mathtt{q}][\mathtt{p}]\& l \\ s[\mathtt{p}][\mathtt{q}]\oplus l \asymp_o s^o[\mathtt{p}][\mathtt{q}]\& l & s[\mathtt{p}][\mathtt{q}]\oplus l \asymp_{io} s^o[\mathtt{p}][\mathtt{q}]\& l & \\ s^o[\mathtt{p}][\mathtt{q}]\oplus l \asymp_o s[\mathtt{q}][\mathtt{p}]\& l & s^i[\mathtt{p}][\mathtt{q}]\oplus l \asymp_{io} s[\mathtt{q}][\mathtt{p}]\& l & \end{array}$$

We omit bound labels and initialisation which are defined similarly and (b) $\asymp_i$ is defined by exchanging o by i and exchanging $\mathtt{p}$ and $\mathtt{q}$ in the queue actions in $\asymp_o$. Then the LTSs are defined by adding the following dequeue and enqueue rules in Figure 5.

$$\begin{array}{llll} \langle\text{QSendO}\rangle & s[\mathtt{p}][\mathtt{o}:h\cdot[\mathtt{q}](v)] \xrightarrow{s^o[\mathtt{p}][\mathtt{q}]!\langle v\rangle} s[\mathtt{p}][\mathtt{o}:h] & \langle\text{QRcvO}\rangle & s[\mathtt{p}][\mathtt{o}:h] \xrightarrow{s^o[\mathtt{p}][\mathtt{q}]?\langle v\rangle} s[\mathtt{p}][\mathtt{o}:[\mathtt{q}](v)\cdot h] \\ \langle\text{QSendI}\rangle & s[\mathtt{p}][\mathtt{i}:h\cdot[\mathtt{q}](v)] \xrightarrow{s^i[\mathtt{p}][\mathtt{q}]!\langle v\rangle} s[\mathtt{p}][\mathtt{i}:h] & \langle\text{QRcvI}\rangle & s[\mathtt{p}][\mathtt{i}:h] \xrightarrow{s^i[\mathtt{p}][\mathtt{q}]?\langle v\rangle} s[\mathtt{p}][\mathtt{i}:[\mathtt{q}](v)\cdot h] \end{array}$$

The rules for selection/branching are similarly defined. $\langle\text{QSendO,QRcvO}\rangle$ are dequeue and enqueue rules for (a); $\langle\text{QSendI,QRcvI}\rangle$ are for (b); and (c) uses all rules. Using the LTSs, the standard typed reduction-closed congruence and bisimulations are defined by the same clauses as in Definitions 4.4 and 4.5 by replacing $(\Gamma,\Delta)\xrightarrow{\ell}(\Gamma',\Delta')$ in Figure 6 by one based on each semantics. We write $\cong^o$, $\cong^i$, $\cong^{io}$, $\approx^o$, $\approx^i$ and $\approx^{io}$ for the counterparts of $\cong^s$ and $\approx^s$ in Definitions 4.4 and 4.5. Define the mode m as $m ::= s \mid o \mid i \mid io$ with the partial order given as $s \sqsubset o$, $s \sqsubset i$, $o \sqsubset io$ and $i \sqsubset io$. Then we have:

Theorem 6.1. (1) $\cong^m = \approx^m$ *for each* m *and* (2) $\cong^{m_1} \subsetneq \cong^{m_2}$ *if* $m_1 \sqsubset m_2$*;* (3) $\approx^{m_1} \subsetneq \approx^{m_2}$ *if* $m_1 \sqsubset m_2$*; and* (4) $\approx^i$ *and* $\approx^o$ *are incompatible.*

6.3 Three asynchronous governed bisimulations

One of the key point of our formulations for governed bisimulations is modularity: we can define various governed semantics based on the LTS semantics of global types. Recall the following key rules of $G \xrightarrow{\lambda} G'$ in § 5.1:

$$\frac{\{s:G\} \xrightarrow{\lambda} \{s:G'\} \quad (\star)}{\{s:\mathtt{p}\to\mathtt{q}:\langle U\rangle.G\} \xrightarrow{\lambda} \{s:\mathtt{p}\to\mathtt{q}:\langle U\rangle.G'\}} \qquad \frac{\{s:G_i\} \xrightarrow{\lambda} \{s:G_i'\}\ i\in I \quad (\star)}{\{s:\mathtt{p}\to\mathtt{q}:\{l_i:G_i\}_{i\in I}\} \xrightarrow{\lambda} \{s:\mathtt{p}\to\mathtt{q}:\{l_i:G_i'\}_{i\in I}\}}$$

	Synchronous	Output	Input	Input/Output
Condition $(\star)$	$\mathtt{p},\mathtt{q} \notin \lambda$	$\mathtt{q} \notin \lambda$	$\mathtt{p},\mathtt{q} \notin \mathtt{out}(\lambda)$	$\mathtt{q} \notin \lambda \vee \mathtt{p},\mathtt{q} \notin \mathtt{out}(\lambda)$
$\mathtt{p} \to \mathtt{q}; \mathtt{r} \to \mathtt{s}$	$\surd$	$\surd$	$\surd$	$\surd$
$\mathtt{p} \to \mathtt{q}; \mathtt{p} \to \mathtt{r}$	$\times$	$\surd$	$\times$	$\surd$
$\mathtt{p} \to \mathtt{q}; \mathtt{r} \to \mathtt{q}$	$\times$	$\times$	$\surd$	$\surd$
$\mathtt{p} \to \mathtt{q}; \mathtt{p} \to \mathtt{q}$	$\times$	$\times$	$\times$	$\times$
$\mathtt{p} \to \mathtt{q}; \mathtt{q} \to \mathtt{p}$	$\times$	$\times$	$\times$	$\times$

Fig. 9. Synchronous/asynchronous semantics for global types and examples of orderings

where $(\star)$ is the side condition which controls the ordering of interactions. Just by changing $(\star)$, we can define the three semantics for global types by the same rules. In the second row in Figure 6.3, we state the condition for $(\star)$ for each semantics. Then we state the examples to show whether each global type in the first column satisfies the side condition in each semantics. The first example says that if the global type is $G = \mathtt{p} \to \mathtt{q}; \mathtt{r} \to \mathtt{s}; \mathtt{end}$, then we can permute the order of two actions as $G \xrightarrow{\lambda_0}\xrightarrow{\lambda} G'$ to $G \xrightarrow{\lambda}\xrightarrow{\lambda_0} G'$ with $\lambda_0 = \mathtt{p} \to \mathtt{q}$ and $\lambda = \mathtt{r} \to \mathtt{s}$ in any four semantics. Since in (a) and (c), we cannot observe the output actions, if the receivers are distinct $(\mathtt{q},\mathtt{r})$, we can permute the two interactions, $\lambda_0 = \mathtt{p} \to \mathtt{q}$ and $\lambda = \mathtt{p} \to \mathtt{r}$. Dually, since in (b) and (c), we cannot observe the input actions, if the outputs are distinct $(\mathtt{p},\mathtt{r})$, we can permute the two interactions. Note that, because of the FIFO ordering by queues, the last two examples are not permutable in any semantics. Just by replacing $(\star)$ by an appropriate condition, we can use exactly the same rules in Figure 7 to define $(E,\Gamma,\Delta) \xrightarrow{\ell} (E',\Gamma',\Delta)$ for each semantics. Then we can reuse all definitions such as Definitions 5.3, 5.4, 5.5 and 5.6 to define governed reduction closed congruence and bisimulation. We denote $\cong^o_g$, $\cong^i_g$, $\cong^{io}_g$, $\approx^o_g$, $\approx^i_g$ and $\approx^{io}_g$ for the counterparts of $\cong^s_g$ and $\approx^s_g$. The main theorem follows:

Theorem 6.2. (1) $\cong^m_g = \approx^m_g$ *for each m and* (2) $\cong^{m_1}_g \subsetneq \cong^{m_2}_g$ *if $m_1 \sqsubset m_2$;* (3) $\approx^{m_1}_g \subsetneq \approx^{m_2}_g$ *if $m_1 \sqsubset m_2$; and* (4) $\approx^i_g$ *and* $\approx^o_g$ *are incompatible.*

Example 6.1 (asynchronous governed bisimulations). We give the examples of governed bisimulations which separate four semantics. Let:

$$\begin{aligned}
P_1 &= s_1[2][1]!\langle v_1\rangle; s_2[3][1]!\langle v_2\rangle; (s_2[1][3]!\langle v_3\rangle; \mathbf{0} \mid s_2[2][3]?(x); \mathbf{0})\\
P_2 &= s_2[3][1]!\langle v_2\rangle; s_1[2][1]!\langle v_1\rangle; (s_2[1][3]!\langle v_3\rangle; s_3[1][2]!\langle v_4\rangle; \mathbf{0} \mid s_3[2][1]?(y); s_2[2][3]?(x); \mathbf{0})
\end{aligned}$$

$$\begin{aligned}
P_3 &= s_1[2][1]?(z_1); s_2[3][1]?(z_2); (s_2[1][3]!\langle v_3\rangle; \mathbf{0} \mid s_2[2][3]?(x); \mathbf{0})\\
P_4 &= s_2[3][1]?(z_2); s_1[2][1]?(z_1); (s_2[1][3]!\langle v_3\rangle; s_3[1][2]!\langle v_4\rangle; \mathbf{0} \mid s_3[2][1]?(y); s_2[2][3]?(x); \mathbf{0})
\end{aligned}$$

and

$$\begin{aligned}
E_1 &= s_1 : 2 \to 1 : \langle U_1\rangle.\mathtt{end} \cdot s_2 : 3 \to 1 : \langle U_2\rangle.1 \to 3 : \langle U_3\rangle.3 \to 2 : \langle U_4\rangle.\mathtt{end} \cdot s_3 : 1 \to 2 : \langle U_5\rangle.\mathtt{end}\\
E_2 &= s_1 : 2 \to 1 : \langle U_1\rangle.\mathtt{end} \cdot s_2 : 3 \to 1 : \langle U_2\rangle.3 \to 2 : \langle U_4\rangle.1 \to 3 : \langle U_3\rangle.\mathtt{end} \cdot s_3 : 1 \to 2 : \langle U_5\rangle.\mathtt{end}\\
E_3 &= s_1 : 1 \to 2 : \langle U_1\rangle.\mathtt{end} \cdot s_2 : 1 \to 3 : \langle U_2\rangle.1 \to 3 : \langle U_3\rangle.3 \to 2 : \langle U_4\rangle.\mathtt{end} \cdot s_3 : 1 \to 2 : \langle U_5\rangle.\mathtt{end}\\
E_4 &= s_1 : 1 \to 2 : \langle U_1\rangle.\mathtt{end} \cdot s_2 : 1 \to 3 : \langle U_2\rangle.3 \to 2 : \langle U_4\rangle.1 \to 3 : \langle U_3\rangle.\mathtt{end} \cdot s_3 : 1 \to 2 : \langle U_5\rangle.\mathtt{end}
\end{aligned}$$

Then we have:

$$\begin{array}{llll} E_1 \vdash P_1 \not\approx^s_g P_2 & E_1 \vdash P_1 \approx^o_g P_2 & E_1 \vdash P_1 \not\approx^i_g P_2 & E_1 \vdash P_1 \approx^{io}_g P_2 \\ E_2 \vdash P_1 \not\approx^s_g P_2 & E_2 \vdash P_1 \not\approx^o_g P_2 & E_2 \vdash P_1 \not\approx^i_g P_2 & E_2 \vdash P_1 \not\approx^{io}_g P_2 \\ E_3 \vdash P_3 \not\approx^s_g P_4 & E_3 \vdash P_3 \not\approx^o_g P_4 & E_3 \vdash P_3 \approx^i_g P_4 & E_3 \vdash P_3 \approx^{io}_g P_4 \\ E_4 \vdash P_3 \not\approx^s_g P_4 & E_4 \vdash P_3 \not\approx^o_g P_4 & E_4 \vdash P_3 \not\approx^i_g P_4 & E_4 \vdash P_3 \not\approx^{io}_g P_4 \end{array}$$

Since $\approx^o_g$ cannot observe the order of output actions to the queues, we have:
$s_1[2][1]!\langle v_1\rangle; s_2[3][1]!\langle v_2\rangle; Q \approx^o_g s_1[2][1]!\langle v_1\rangle; s_2[3][1]!\langle v_2\rangle; Q$ but
$s_1[2][1]!\langle v_1\rangle; s_2[3][1]!\langle v_2\rangle; Q \not\approx^i_g s_1[2][1]!\langle v_1\rangle; s_2[3][1]!\langle v_2\rangle; Q$.

Then, under E_1,
$(s_2[1][3]!\langle v_3\rangle; \mathbf{0} \mid s_2[2][3]?(x); \mathbf{0}) \approx^o_g s_2[1][3]!\langle v_3\rangle; s_3[1][2]!\langle v_4\rangle; \mathbf{0} \mid s_3[2][1]?(y); s_2[2][3]?(x); \mathbf{0}$
since $1 \to 3 : \langle U_3\rangle.3 \to 2 : \langle U_4\rangle.G$ in E_1 guarantees $s_2[2][3]?(x); \mathbf{0}$ will not be executed before $s_2[1][3]!\langle v_3\rangle$ happens. Obviously they are not equated under E_2. The (in)equations between P_3 and P_4 can be reasoned similarly.

7 Related and Future Work

As a typed foundation for structured communications programming, session types [8, 17] have been studied over the last decade for a wide range of process calculi and programming languages. Recently several works developed multiparty session types and their extensions. While typed behavioural equivalences are one of the central topics of the π-calculus, surprisingly the typed behavioural semantics based on session types have been less explored, and the existing ones only focus on binary (two-party) sessions. Our work [12] develops an *asynchronous binary* session typed behavioural theory with event operations. An LTS is defined on session type process judgements and ensures session typed properties, such as linearity in the presence of asynchronous queues. The work [15] proves the proof conversions induced by Linear Logic interpretation coincide with an observational equivalence over a strict subset of the binary synchronous session processes. The main focus of our paper is *multiparty* session types and governed bisimulation, whose definitions and properties crucially depend on information of global types. We have shown governed bisimulations can be systematically developed under various semantics including three kinds of asynchronous semantics by modularly changing the LTS for processes, environments and global types. For governed bisimulations, we can reuse all of the definitions among four semantics by only changing the conditions of the LTS of global types to suit each semantics. Another recent work [4] gives an a fully abstract encoding of a *binary* synchronous session typed calculus into a linearly typed π-calculus [2]. We believe the same encoding method is smoothly applicable to $\approx^s$ since it is defined solely based on the projected types (i.e. local types). However a governed bisimulation requires a global witness, hence the additional global information would be required for full abstraction.

The constructions of our work are hinted by [7] which studies typed behavioural semantics for the π-calculus with IO-subtyping where a labelled transition system for pairs of typing environments and processes is used for defining typed testing equivalences and barbed congruence. Several papers have developed bisimulations for the higher-order π-calculus or its variants using the information of the environments. Among them, a recent paper [11] uses a pair of a process and an observer knowledge set for the LTS. The knowledge set contains a mapping from first order values to the higher-order processes, which allows a tractable higher-order behavioural theory using the first-order

LTS. We record a choreographic type as the witness in the environment to obtain fine-grained bisimulations of multiparty processes.

The highlight of our bisimulation construction is an effective use of the semantics of global types for LTSs of processes (cf. [Inv] in Figure 7 and Definition 5.4). Global types can give a guidance how to coordinate parallel threads giving explicit protocols, hence it is applicable to a semantic-preserving optimisation (cf. Example 5.2). While it is known that it is undecidable to check $P \approx Q$ in the full π-calculus, it is an interesting future topic to investigate automated bisimulation-checking techniques for the governed bisimulations for some subset of multiparty session processes.

References

1. R. M. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous pi-calculus. *TCS*, 195(2):291–324, 1998.
2. M. Berger, K. Honda, and N. Yoshida. Sequentiality and the π-calculus. In *Proc. TLCA'01*, volume 2044 of *LNCS*, pages 29–45, 2001.
3. L. Bettini et al. Global progress in dynamically interleaved multiparty sessions. In *CONCUR*, volume 5201 of *LNCS*, pages 418–433. Springer, 2008.
4. R. Demangeon and K. Honda. Full abstraction in a subtyped pi-calculus with linear types. In *CONCUR*, volume 6901 of *LNCS*, pages 280–296. Springer, 2011.
5. P.-M. Deniélou and N. Yoshida. Multiparty session types meet communicating automata. In *ESOP*, volume 7211 of *LNCS*, pages 194–213. Springer, 2012.
6. M. Hennessy. *A Distributed Pi-Calculus*. CUP, 2007.
7. M. Hennessy and J. Rathke. Typed behavioural equivalences for processes in the presence of subtyping. *Mathematical Structures in Computer Science*, 14(5):651–684, 2004.
8. K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP'98*, volume 1381 of *LNCS*, pages 22–138. Springer, 1998.
9. K. Honda and N. Yoshida. On reduction-based process semantics. *TCS*, 151(2):437–486, 1995.
10. K. Honda, N. Yoshida, and M. Carbone. Multiparty Asynchronous Session Types. In *POPL'08*, pages 273–284. ACM, 2008.
11. V. Koutavas and M. Hennessy. A testing theory for a higher-order cryptographic language. In *ESOP*, volume 6602 of *LNCS*, pages 358–377, 2011.
12. D. Kouzapas, N. Yoshida, and K. Honda. On asynchronous session semantics. In *FMOODS/FORTE*, volume 6722 of *Lecture Notes in Computer Science*, pages 228–243, 2011.
13. D. Kouzapas, N. Yoshida, and K. Honda. On asynchronous session semantics. In *FMOODS/FORTE*, volume 6722 of *LNCS*, pages 228–243, 2011.
14. Ocean Observatories Initiative (OOI). `http://www.oceanobservatories.org/`.
15. J. A. Pérez, L. Caires, F. Pfenning, and B. Toninho. Linear logical relations for session-based concurrency. In *ESOP*, volume 7211 of *LNCS*, pages 539–558. Springer, 2012.
16. B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *MSCS*, 6(5):409–454, 1996.
17. K. Takeuchi, K. Honda, and M. Kubo. An Interaction-based Language and its Typing System. In *PARLE'94*, volume 817 of *LNCS*, pages 398–413, 1994.
18. N. Yoshida. Graph types for monadic mobile processes. In *FSTTCS*, volume 1180 of *LNCS*, pages 371–386. Springer, 1996.
19. N. Yoshida and V. T. Vasconcelos. 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, 2007.

A Appendix for Typing Rules

We say a typing Δ is *fully coherent* (notation $\mathtt{fco}(\Delta)$) if it is coherent and if $s[\mathtt{p}] : T_{\mathtt{p}} \in \Delta$ then for all $\mathtt{q} \in \mathtt{roles}(T_{\mathtt{p}})$, $s[\mathtt{q}] : T_{\mathtt{q}} \in \Delta$.

$$\Gamma \cdot u : S \vdash u : S \ [\text{Name}] \quad \Gamma \vdash \mathtt{tt}, \mathtt{ff} : \mathtt{bool} \ [\text{Bool}] \quad \frac{\Gamma \vdash e_i : \mathtt{bool}}{\Gamma \vdash e_1 \ \mathtt{and} \ e_2 : \mathtt{bool}} \ [\text{And}]$$

$$\frac{\Gamma \vdash a : \langle G \rangle \quad \Gamma \vdash P \triangleright \Delta \cdot x[\mathtt{p}] : G\lceil \mathtt{p} \quad \max(\mathtt{roles}(G)) = \mathtt{p}}{\Gamma \vdash \overline{a}[\mathtt{p}](x).P \triangleright \Delta} \ [\text{MReq}] \qquad \frac{\Gamma \vdash a : \langle G \rangle \quad \Gamma \vdash P \triangleright \Delta \cdot x[\mathtt{p}] : G\lceil \mathtt{p}}{\Gamma \vdash a[\mathtt{p}](x).P \triangleright \Delta} \ [\text{MAcc}]$$

$$\frac{\Gamma \vdash e : S \quad \Gamma \vdash P \triangleright \Delta \cdot c : T}{\Gamma \vdash c[\mathtt{q}]!\langle e \rangle; P \triangleright \Delta \cdot c : [\mathtt{q}]!\langle S \rangle; T} \ [\text{Send}] \qquad \frac{\Gamma \cdot x : S \vdash P \triangleright \Delta \cdot c : T}{\Gamma \vdash c[\mathtt{q}]?(x); P \triangleright \Delta \cdot c : [\mathtt{q}]?(S); T} \ [\text{Recv}]$$

$$\frac{\Gamma \vdash P \triangleright \Delta \cdot c : T}{\Gamma \vdash c[\mathtt{q}]!\langle c' \rangle; P \triangleright \Delta \cdot c : [\mathtt{q}]!\langle T' \rangle; T \cdot c' : T'} \ [\text{Deleg}] \qquad \frac{\Gamma \vdash P \triangleright \Delta \cdot c : T \cdot x : T'}{\Gamma \vdash c[\mathtt{q}]?(x); P \triangleright \Delta \cdot c : [\mathtt{q}]?(T'); T} \ [\text{SRecv}]$$

$$\frac{\Gamma \vdash P \triangleright \Delta \cdot c : T}{\Gamma \vdash c[\mathtt{q}] \oplus l_i; P \triangleright \Delta \cdot c : [\mathtt{q}] \oplus \{l_i : T_i\}_{i \in I}} \ [\text{Sel}] \qquad \frac{\Gamma \vdash P_i \triangleright \Delta \cdot c : T_i \quad \forall\, i \in I}{\Gamma \vdash c[\mathtt{q}] \& \{l_i : P_i\}_{i \in I} \triangleright \Delta \cdot c : [\mathtt{q}] \& \{l_i : T_i\}_{i \in I}} \ [\text{Bra}]$$

$$\frac{\Gamma \vdash P_1 \triangleright \Delta_1 \quad \Gamma \vdash P_2 \triangleright \Delta_2 \quad \Delta_1 \cap \Delta_2 = \emptyset}{\Gamma \vdash P_1 \mid P_2 \triangleright \Delta_1 \cdot \Delta_2} \ [\text{Conc}] \qquad \frac{\Gamma \vdash e : \mathtt{bool} \quad \Gamma \vdash P \triangleright \Delta \quad \Gamma \vdash Q \triangleright \Delta}{\Gamma \vdash \mathtt{if}\ e\ \mathtt{then}\ P\ \mathtt{else}\ Q \triangleright \Delta} \ [\text{If}]$$

$$\frac{\Delta\, \mathtt{end}\ \text{only}}{\Gamma \vdash \mathbf{0} \triangleright \Delta} \ [\text{Inact}] \qquad \frac{\Gamma \cdot a : \langle G \rangle \vdash P \triangleright \Delta}{\Gamma \vdash (\nu\, a)P \triangleright \Delta} \ [\text{NRes}]$$

$$\frac{\mathtt{fco}(\{s[1] : T_1 \ldots s[n] : T_n\}) \quad \Gamma \vdash P \triangleright \Delta \cdot s[1] : T_1 \ldots s[n] : T_n}{\Gamma \vdash (\nu\, s)P \triangleright \Delta} \ [\text{SRes}] \quad \Gamma \cdot X : \Delta \vdash X \triangleright \Delta \ [\text{Var}] \quad \frac{\Gamma \cdot X : \Delta \vdash P \triangleright \Delta}{\Gamma \vdash \mu X.P \triangleright \Delta} \ [\text{Rec}]$$

Fig. 10. Typing System for Synchronous Multiparty Session Calculus

Figure 10 defines the typing system. Rule [Name] types a shared name or shared variable to type S. Boolean $\mathtt{tt}, \mathtt{ff}$ are typed with the $\mathtt{bool}$ type via rule [Bool]. Logical expressions are also typed with the $\mathtt{bool}$ type via rule [And], etc. Rules [MReq] and [MAcc] check that the local type of a session role agrees with the global type of the initiating shared name. Rules [Send] and [Recv] prefix the local type with send and receive local types respectively, after checking the type environment for the sending value type (receiving variable type resp.). Delegation is typed under rules [Deleg] and [Srecv] where we check type consistency of the delegating/receiving session role. Rules [Sel] and [Bra] type select and branch processes respectively. A select process uses the select local type. A branching process checks that all continuing process have a consistent typing environments. [Conc] types a parallel composition of processes by checking the disjointness of their typing environments. Conditional is typed with [If], where we check the expression e to be of $\mathtt{bool}$ type and the branching processes to have the same typing environment. Rule [Nres] defines the typing for shared name restriction. Rule [Sres] uses the full coherency property to restrict a session name. Recursive rules [Var] and [Rec] are standard. Finally the inactive process $\mathbf{0}$ is typed with the complete typing environment, where every session role is mapped to the inactive local type $\mathtt{end}$.

B Appendix for Sections 2 and 3

We list the omitted definitions from Section 2 and

$$P \equiv P \mid \mathbf{0} \quad P \mid Q \equiv Q \mid P \quad (P \mid Q) \mid R \equiv P \mid (Q \mid R) \quad P \equiv_\alpha P \quad \mu X.P \equiv P\{\mu X.P/X\}$$
$$(\nu\, n)(\nu\, n')P \equiv (\nu\, n')(\nu\, n)P \quad (\nu\, n)(\nu\, n')P \equiv (\nu\, nn')P \quad (\nu\, n)\mathbf{0} \equiv \mathbf{0}$$
$$(\nu\, n)(P) \mid Q \equiv (\nu\, n)(P \mid Q) \quad n \notin \mathtt{fn}(Q)$$

Fig. 11. Structural Congruence for Synchronous Multiparty Session Calculus

The structural congruence rules are defined in figure 11.

We define the roles occurring in a global type and the roles occurring in a local type.

Definition B.1 (Roles).

- *We define* $\mathtt{roles}(G)$ *as the set of roles in protocol* G*. Note that for all* $u : G \in \Gamma$, $\mathtt{roles}(G) = \{1, 2, ..., n\}$ *for some* n*.*
- *We define* $\mathtt{roles}(T)$ *on local types as:*

$$\mathtt{roles}(\mathtt{end}) = \emptyset \;\; \mathtt{roles}(\mathtt{t}) = \emptyset \;\; \mathtt{roles}(\mu\mathtt{t}.T) = \mathtt{roles}(T)$$

$$\mathtt{roles}([\mathtt{p}]!\langle U\rangle;T) = \{\mathtt{p}\} \cup \mathtt{roles}(T) \qquad \mathtt{roles}([\mathtt{p}]?(U);T) = \{\mathtt{p}\} \cup \mathtt{roles}(T)$$
$$\mathtt{roles}([\mathtt{p}] \oplus \{l_i : T_i\}_{i \in I}) = \{\mathtt{p}\} \cup \mathtt{roles}(T) \quad \mathtt{roles}([\mathtt{p}]\&\{l_i : T_i\}_{i \in I}) = \{\mathtt{p}\} \cup \mathtt{roles}(T)$$

C Proofs for Bisimulation Properties

C.1 Parallel Observer Property

Lemma C.1. *If* $\Gamma \vdash P_1 \triangleright \Delta_1, \Gamma \vdash P_2 \triangleright \Delta_2$ *and* $E, \Gamma \vdash P_1 \mid P_2 \triangleright \Delta$ *then*

1. $\Delta = \Delta_1 \cup \Delta_2$, $\Delta_1 \cap \Delta_2 = \emptyset$
2. $E, \Gamma \vdash P_1 \triangleright \Delta_1$ *and* $E, \Gamma \vdash P_2 \triangleright \Delta_2$

Proof. Part 1 is obtain from typing rule [Conc]. Part 2 is immediate from part 1, since $\Delta \subseteq \Delta_1$ (resp. $\Delta \subseteq \Delta_2$). □

C.2 Proof for Lemma 4.1

Proof. We use the coinduction method which is implied by the bisimilarity definition.

Assume that for $\Gamma \vdash P_1 \triangleright \Delta_1 \approx^s P_2 \triangleright \Delta_2$, we have $\Delta_1 \rightleftharpoons \Delta_2$. Then by the definition of $\rightleftharpoons$, there exists Δ such that

$$\Delta_1 \longrightarrow^* \Delta \text{ and } \Delta_2 \longrightarrow^* \Delta \tag{1}$$

Now assume that $\Gamma \vdash P_1 \triangleright \Delta_1 \xrightarrow{\ell} P_1' \triangleright \Delta_1'$ then, $\Gamma \vdash P_2 \triangleright \Delta_2 \xRightarrow{\ell} P_2' \triangleright \Delta_2'$ and by the typed transition definition we get $(\Gamma, \Delta_1) \xrightarrow{\ell} (\Gamma, \Delta_1')$, $(\Gamma, \Delta_2) \xRightarrow{\ell} (\Gamma, \Delta_2')$. We need to show that $\Delta_1' \rightleftharpoons \Delta_2'$.

We prove by a case analysis on the transition $\xrightarrow{\ell}$ on (Γ, Δ_1) and (Γ, Δ_2).

– **Case $\ell = \tau$:** We use the fact that $\xrightarrow{\tau}$ with $\equiv$ coincides with $\longrightarrow$. Then by Theorem 3.1, we obtain if $\Gamma \vdash P_1 \triangleright \Delta_1$ and $P_1 \longrightarrow P_1'$ then $\Gamma \vdash P_1' \triangleright \Delta_1'$ and $\Delta_1 \longrightarrow \Delta_1'$ or $\Delta_1 = \Delta_1'$.
For the reversed direction, if $\Gamma \vdash P_2 \triangleright \Delta_2$ and $P_2 \twoheadrightarrow P_2'$ then $\Gamma \vdash P_2' \triangleright \Delta_2'$ and $\Delta_2 \longrightarrow^* \Delta_2'$. From the hypothesis on $\Gamma \vdash P_1' \triangleright \Delta_1'$ and $\Gamma \vdash P_2' \triangleright \Delta_2'$, we obtain there exists Δ such that $\Delta_1' \longrightarrow^* \Delta$ and $\Delta_2' \longrightarrow^* \Delta$, as required.

Case $\ell = a[\mathtt{p}](s)$ or $\ell = \overline{a}[\mathtt{p}](s)$: Then

$$(\Gamma, \Delta_1) \xrightarrow{\ell} (\Gamma, \Delta_1 \cdot s[\mathtt{p}] : T_{\mathtt{p}} \cdot \ldots \cdot s[\mathtt{q}] : T_{\mathtt{q}})$$

and

$$(\Gamma, \Delta_2) \Longrightarrow\xrightarrow{\ell}\Longrightarrow (\Gamma, \Delta_2'' \cdot s[\mathtt{p}] : T_{\mathtt{p}} \cdot \ldots \cdot s[\mathtt{q}] : T_{\mathtt{q}})$$

We set

$$\Delta' = \Delta \cdot s[\mathtt{p}] : T_{\mathtt{p}} \cdot \ldots \cdot s[\mathtt{q}] : T_{\mathtt{q}}$$

to obtain $\Delta_1' \longrightarrow^* \Delta'$ and $\Delta_2' \longrightarrow^* \Delta'$, by the coinduction hypothesis (1).

– **Case $\ell = s[\mathtt{p}][\mathtt{q}]!\langle v \rangle$:**
For *synchronous and input asynchronous multiparty session π calculus*, we know from the definition of environment transition, that $s[\mathtt{q}] \notin \mathtt{dom}(\Delta_1)$ and $s[\mathtt{q}] \notin \mathtt{dom}(\Delta_2)$, thus $s[\mathtt{q}] \notin \mathtt{dom}(\Delta)$ for the synchronous case and $s^i[\mathtt{q}] \notin \mathtt{dom}(\Delta_1)$ and $s^i[\mathtt{q}] \notin \mathtt{dom}(\Delta_2)$, thus $s^i[\mathtt{q}] \notin \mathtt{dom}(\Delta)$ for the input asynchronous case. We set

$$\Delta_1 = s[\mathtt{p}] : [\mathtt{q}]!\langle v \rangle; T \cdot \Delta_1''$$

and

$$\Delta_2 = s[\mathtt{p}] : [\mathtt{q}]!\langle v \rangle; T \cdot \Delta_2''$$

so

$$\Delta = s[\mathtt{p}] : [\mathtt{q}]!\langle v \rangle; T \cdot \Delta''$$

by (1). We set $\Delta' = s[\mathtt{p}] : T \cdot \Delta''$ to obtain $\Delta_1' \longrightarrow^* \Delta'$ and $\Delta_2' \longrightarrow^* \Delta'$.
For *output and input/output asynchronous multiparty session π calculus*, we know from the definition of environment transition, that $s[\mathtt{q}] \notin \mathtt{dom}(\Delta_1)$ and $s[\mathtt{q}] \notin \mathtt{dom}(\Delta_2)$, thus $s[\mathtt{q}] \notin \mathtt{dom}(\Delta)$ for the output asynchrony case and $s^i[\mathtt{q}] \notin \mathtt{dom}(\Delta_1)$ and $s^i[\mathtt{q}] \notin \mathtt{dom}(\Delta_2)$, thus $s^i[\mathtt{q}] \notin \mathtt{dom}(\Delta)$ for the input/output asynchronous case. Then

$$\Delta_1 = s^o[\mathtt{p}] : M; [\mathtt{q}]!\langle v \rangle \cdot \Delta_1''$$

and

$$\Delta_2 = s^o[\mathtt{p}] : M; [\mathtt{q}]!\langle v \rangle \cdot \Delta_2''$$

so

$$\Delta = s^o[\mathtt{p}] : M; [\mathtt{q}]!\langle v \rangle \cdot \Delta''$$

by (1). We set $\Delta' = s^o[\mathtt{p}] : M \cdot \Delta''$ to obtain $\Delta_1' \longrightarrow^* \Delta'$ and $\Delta_2' \longrightarrow^* \Delta'$.

– **Case** $\ell = s[\mathtt{p}][\mathtt{q}]!(s'[\mathtt{p}'])$**:**
For *synchronous and input asynchronous multiparty session π calculus*, we know from the definition of environment transition, that for the synchronous case, $s[\mathtt{q}] \notin \mathtt{dom}(\Delta_1)$ and $s[\mathtt{q}] \notin \mathtt{dom}(\Delta_2)$, thus $s[\mathtt{q}] \notin \mathtt{dom}(\Delta)$. And for the input asynchronous case $s^i[\mathtt{q}] \notin \mathtt{dom}(\Delta_1)$ and $s^i[\mathtt{q}] \notin \mathtt{dom}(\Delta_2)$, thus $s^i[\mathtt{q}] \notin \mathtt{dom}(\Delta)$. We set

$$\Delta_1 = s[\mathtt{p}] : [\mathtt{q}]!\langle T' \rangle; T \cdot \Delta_1''$$

and

$$\Delta_2 = s[\mathtt{p}] : [\mathtt{q}]!\langle T' \rangle; T \cdot \Delta_2''$$

so

$$\Delta = s[\mathtt{p}] : [\mathtt{q}]!\langle v \rangle; T \cdot \Delta''$$

by (1). We set $\Delta' = s[\mathtt{p}] : T \cdot \Delta'' \cdot \{s[\mathtt{p}_i] : T_i\}$ to obtain $\Delta_1' \longrightarrow^* \Delta'$ and $\Delta_2' \longrightarrow^* \Delta'$.
For *output and input/output asynchronous multiparty session π calculus*, we know from the definition of environment transition, that $s[\mathtt{q}] \notin \mathtt{dom}(\Delta_1)$ and $s[\mathtt{q}] \notin \mathtt{dom}(\Delta_2)$, thus $s[\mathtt{q}] \notin \mathtt{dom}(\Delta)$ for the output asynchrony case and $s^i[\mathtt{q}] \notin \mathtt{dom}(\Delta_1)$ and $s^i[\mathtt{q}] \notin \mathtt{dom}(\Delta_2)$, thus $s^i[\mathtt{q}] \notin \mathtt{dom}(\Delta)$ for the input/output asynchronous case. Then $s[\mathtt{q}] \notin \mathtt{dom}(\Delta_1)$ and $s[\mathtt{q}] \notin \mathtt{dom}(\Delta_2)$, thus $s[\mathtt{q}] \notin \mathtt{dom}(\Delta)$ and

$$\Delta_1 = s^o[\mathtt{p}] : M; [\mathtt{q}]!\langle T' \rangle \cdot \Delta_1''$$

and

$$\Delta_2 = s^o[\mathtt{p}] : M; [\mathtt{q}]!\langle T' \rangle \cdot \Delta_2''$$

so

$$\Delta = s^o[\mathtt{p}] : M; [\mathtt{q}]!\langle v \rangle \cdot \Delta''$$

by (1). We set $\Delta' = s^o[\mathtt{p}] : M \cdot \Delta'' \cdot \{s[\mathtt{p}_i] : T_i\}$ to obtain $\Delta_1' \longrightarrow^* \Delta'$ and $\Delta_2' \longrightarrow^* \Delta'$.
– The remaining cases on session channel actions are similar.

□

C.3 Weakening - Strengthening

The following lemmas are essential for invariant properties.

Lemma C.2 (Weakening).

1. *If $E, \Gamma \vdash P \triangleright \Delta$ then*
   – $E \cdot s : G, \Gamma \vdash P \triangleright \Delta$.
   – $E = E' \cdot s : G$ *and* $\exists G' \cdot \{s : G'\} \twoheadrightarrow \{s : G\}$ *then* $E' \cdot s : G', \Gamma \vdash P \triangleright \Delta$.
2. *If* $(E, \Gamma, \Delta) \xrightarrow{\ell} (E, \Gamma', \Delta')$ *then*
   – $(E \cdot s : G, \Gamma, \Delta) \xrightarrow{\ell} (E \cdot s : G, \Gamma', \Delta')$
   – *If* $E = E' \cdot s : G$ *and* $\{s : G'\} \twoheadrightarrow \{s : G\}$ *then* $(E' \cdot s : G', \Gamma, \Delta) \xrightarrow{\ell} (E' \cdot s : G', \Gamma', \Delta')$
3. *If* $E, \Gamma \vdash P_1 \triangleright \Delta_2 \approx_g P_2 \triangleright \Delta_2$
   – $E \cdot s : G, \Gamma \vdash P_1 \triangleright \Delta_2 \approx_g P_2 \triangleright \Delta_2$
   – *If* $E = E' \cdot s : G$ *and* $\{s : G'\} \twoheadrightarrow\subseteq \{s : G\}$ *then* $E' \cdot s : G', \Gamma \vdash P_1 \triangleright \Delta_2 \approx_g P_2 \triangleright \Delta_2$

Proof. We only show Part 1. Other parts are similar.

- From the governance judgement definition we have that $E \longrightarrow^* E_1$ and $\mathtt{proj}(E_1) \supseteq *(\Delta)$.
  Let $E \cdot s : G \longrightarrow E_1 \cdot s : G$. Then $\mathtt{proj}(E_1 \cdot s : G) = \mathtt{proj}(E_1) \cup \mathtt{proj}(s : G) \supseteq \mathtt{proj}(E_1) \supseteq *(\Delta)$.
- From the governance judgement definition we have that $E \cdot s : G \longrightarrow^* E_1 \cdot s : G_1$ and $\mathtt{proj}(E_1 \cdot s : G_1) \supseteq *(\Delta)$.
  Let $E \cdot s : G' \longrightarrow^* E_1 \cdot s : G' \longrightarrow^* E_1 \cdot S : G_1$. Then the result is immediate.

□

Lemma C.3 (Strengthening).

1. *If* $E \cdot s : G, \Gamma \vdash P \triangleright \Delta, E_1 \cdot s : G_1$ *and*
   - *If* $s \notin \mathtt{fn}(P)$ *then* $E, \Gamma \vdash P \triangleright \Delta$
   - *If* $\exists G', \{s : G\} \twoheadrightarrow \{s : G'\} \twoheadrightarrow \{s : G_1\}$ *then* $E' \cdot s : G', \Gamma \vdash P \triangleright \Delta$
2. *If* $(E \cdot s : G, \Gamma, \Delta) \xrightarrow{\ell} (E' \cdot s : G, \Gamma', \Delta')$ *then*
   - $(E, \Gamma, \Delta) \xrightarrow{\ell} (E', \Gamma', \Delta')$
   - *If* $\exists G', \{s : G\} \twoheadrightarrow \{s : G'\} \twoheadrightarrow \{s : G_1\}\ S\ (E \cdot s : G', \Gamma, \Delta) \xrightarrow{\ell} (E' \cdot s : G', \Gamma', \Delta')$
3. *If* $E \cdot s : G, \Gamma \vdash P_1 \triangleright \Delta_2, E_1 \approx_g P_2 \triangleright \Delta_2, E_2$
   - *If* $s \notin \mathtt{fn}(P)$ *then* $E, \Gamma \vdash P_1 \triangleright \Delta_2 \approx_g P_2 \triangleright \Delta_2$
   - *If* $\exists G', \{s : G\} \twoheadrightarrow \{s : G'\} \twoheadrightarrow \{s : G_1\}\ E \cdot s : G', \Gamma \vdash P_1 \triangleright \Delta_2 \approx_g P_2 \triangleright \Delta_2$

Proof. We prove for part 1. Other parts are similar.

- From the governance judgement definition we have that $E \cdot s : G \longrightarrow^* E_1 \cdot s : G_1$ and $\mathtt{proj}(E_1 \cdot s : G_1) = \mathtt{proj}(E_1) \cup \mathtt{proj}(s : G_1) \supseteq *(\Delta)$. Since $s \notin \mathtt{fn}(P)$ then $s \notin \mathtt{dom}(\Delta)$, then $\mathtt{proj}(s : G_1) \cap *(\Delta) = \emptyset$. So $\mathtt{proj}(E_1) \supseteq *(\Delta)$ and $E \longrightarrow^* E_1$.
- The result is immediate from the definition of governance judgement.

□

C.4 Configuration Transition Properties

Lemma C.4.

- *If* $E \overset{s:\mathtt{p}\to\mathtt{q}:U}{\longrightarrow} E'$ *then* $\{s[\mathtt{p}] : [\mathtt{q}]!\langle U\rangle; T_{\mathtt{p}}, s[\mathtt{q}] : [\mathtt{p}]?(U); T_{\mathtt{q}}\} \subseteq \mathtt{proj}(E)$ *and* $\{s[\mathtt{p}] : T_{\mathtt{p}}, s[\mathtt{q}] : T_{\mathtt{q}}\} \subseteq \mathtt{proj}(E')$.
- *If* $E \overset{s:\mathtt{p}\to\mathtt{q}:l}{\longrightarrow} E'$ *then* $\{s[\mathtt{p}] : [\mathtt{q}] \oplus \{l_i : T_{i\mathtt{p}}\}, s[\mathtt{q}] : [\mathtt{p}]\&\{l_i : T_{i\mathtt{q}}\}\} \subseteq \mathtt{proj}(E)$ *and* $\{s[\mathtt{p}] : T_{k\mathtt{p}}, s[\mathtt{q}] : T_{k\mathtt{q}}\} \subseteq \mathtt{proj}(E')$

Proof. Part 1: We apply induction on the definition structure of $s : \mathtt{p} \to \mathtt{q} : U$. The base case

$$\{s : \mathtt{p} \to \mathtt{q} : \langle U\rangle.G\} \overset{s:\mathtt{p}\to\mathtt{q}:U}{\longrightarrow} \{s : G\}$$

is easy since

$$\{s[\mathtt{p}] : (\mathtt{p} \to \mathtt{q} : \langle U\rangle.G)\lceil\mathtt{p}, s[\mathtt{q}] : (\mathtt{p} \to \mathtt{q} : \langle U\rangle.G)\lceil\mathtt{q}\} =$$
$$\{s[\mathtt{p}] : [\mathtt{q}]!\langle U\rangle; T_{\mathtt{p}}, s[\mathtt{q}] : [\mathtt{p}]?(U); T_{\mathtt{q}}\} \subseteq \mathtt{proj}(s : \mathtt{p} \to \mathtt{q} : \langle U\rangle.G)$$

and

$$\{s[\mathtt{p}] : G\lceil \mathtt{p}, s[\mathtt{q}] : G\lceil \mathtt{q}\} = \{s[\mathtt{p}] : T_{\mathtt{p}}, s[\mathtt{q}] : T_{\mathtt{q}}\} \subseteq \mathtt{proj}(s : G)$$

The main induction rule concludes that:

$$\{s : \mathtt{p}' \to \mathtt{q}' : \langle U\rangle.G\} \xrightarrow{s:\mathtt{p}\to\mathtt{q}:U} \{s : G'\}$$

if $\mathtt{p} \neq \mathtt{p}'$ and $\mathtt{q} \neq \mathtt{q}'$ and $\{s : G\} \xrightarrow{s:\mathtt{p}\to\mathtt{q}:U} \{s : G'\}$. From the induction hypothesis we know that:

$$\begin{aligned} \{s[\mathtt{p}] : [\mathtt{q}]!\langle U\rangle; T_{\mathtt{p}}, s[\mathtt{q}] : [\mathtt{p}]?(U); T_{\mathtt{q}}\} &\subseteq \mathtt{proj}(s : G) \\ \{s[\mathtt{p}] : T_{\mathtt{p}}, s[\mathtt{q}] : T_{\mathtt{q}}\} &\subseteq \mathtt{proj}(s : G') \end{aligned}$$

to conclude that:

$$\begin{aligned} \{s[\mathtt{p}] : (\mathtt{p}' \to \mathtt{q}' : \langle U\rangle.G)\lceil \mathtt{p}, s[\mathtt{q}] : (\mathtt{p}' \to \mathtt{q}' : \langle U\rangle.G)\lceil \mathtt{q}\} &= \\ \{s[\mathtt{p}] : G\lceil \mathtt{p}, s[\mathtt{q}] : G\lceil \mathtt{q}\} &= \\ \{s[\mathtt{p}] : [\mathtt{q}]!\langle U\rangle; T_{\mathtt{p}}, s[\mathtt{q}] : [\mathtt{p}]?(U); T_{\mathtt{q}}\} &\subseteq \mathtt{proj}(s : G) \end{aligned}$$

and

$$\begin{aligned} \{s[\mathtt{p}] : (\mathtt{p}' \to \mathtt{q}' : \langle U\rangle.G')\lceil \mathtt{p}, s[\mathtt{q}] : (\mathtt{p}' \to \mathtt{q}' : \langle U\rangle.G')\lceil \mathtt{q}\} &= \\ \{s[\mathtt{p}] : G'\lceil \mathtt{p}, s[\mathtt{q}] : G'\lceil \mathtt{q}\} &= \\ \{s[\mathtt{p}] : T_{\mathtt{p}}, s[\mathtt{q}] : T_{\mathtt{q}}\} &\subseteq \mathtt{proj}(s : G) \end{aligned}$$

as required.

Part 2: Similar. □

Proof for Proposition 5.2

Proof. **(1)** We apply induction on the definition structure of $\xrightarrow{\ell}$.

Basic Step:

Case: $\ell = \overline{a}[s](A)$.
From rule [Acc] we get

$$(E_1, \Gamma_1, \Delta_1) \xrightarrow{\ell} (E_1 \cdot s : G, \Gamma_1, \Delta_1 \cdot \{s[\mathtt{p}_i] : s\lceil \mathtt{p}_i\}_{\mathtt{p}_i \in A})$$

From the environment configuration definition we get that

$$\exists E_1' \cdot E_1 \longrightarrow^* E_1', \mathtt{proj}(E_1') \supseteq *(\Delta_1)$$

We also get that $\mathtt{proj}(s : G) \supseteq \{s[\mathtt{p}_i] : s\lceil \mathtt{p}_i\}_{i \in A}$. So we can safely conclude that

$$E_1 \cdot s : G \longrightarrow^* E_1' \cdot s : G, \mathtt{proj}(E_1 \cdot s : G) \supseteq \Delta_1 \cdot \{s[\mathtt{p}_i] : s\lceil \mathtt{p}_i\}_{\mathtt{p}_i \in A}$$

Case: $\ell = \overline{a}[s](A)$. Similar as above.

Case: $\ell = s[\mathtt{p}][\mathtt{q}]!\langle v\rangle$.
From rule [Out] we get

$$(E_1, \Gamma, \Delta \cdot s[\mathtt{p}] : [\mathtt{q}]!\langle U\rangle; T) \quad \xrightarrow{\ell} \quad (E_2, \Gamma, \Delta \cdot s[\mathtt{p}] : T) \tag{2}$$
$$\mathtt{proj}(E_1) \quad \supseteq \quad \Delta \cdot s[\mathtt{p}] : [\mathtt{q}]!\langle U\rangle; T \tag{3}$$
$$E_1 \overset{s:\mathtt{p}\to\mathtt{q}:U}{\longrightarrow} E_2 \tag{4}$$

From 3, we obtain $\mathtt{proj}(E_1) \supseteq \Delta \cdot \{s[\mathtt{p}] : [\mathtt{q}]!\langle U\rangle; T \cdot s[\mathtt{q}] : [\mathtt{p}]?(U); T'\}$ and from 4 and Lemma C.4, we obtain that $\mathtt{proj}(E_2) \supseteq \Delta \cdot \{s[\mathtt{p}] : T \cdot s[\mathtt{q}] : T'\}$.
Case: $\ell = s[\mathtt{p}][\mathtt{q}]!(s'[\mathtt{p}'])$.

$$(E_1, \Gamma, \Delta \cdot s[\mathtt{p}] : [\mathtt{q}]!\langle T\mathtt{p}'\rangle; T) \quad \xrightarrow{\ell} \quad (E_2 \cdot s : G, \Gamma, \Delta \cdot s[\mathtt{p}] : T \cdot \{s[\mathtt{p}_i] : s\lceil \mathtt{p}_i\}) \tag{5}$$
$$\mathtt{proj}(E_1) \quad \supseteq \quad \Delta \cdot s[\mathtt{p}] : [\mathtt{q}]!\langle T'_{\mathtt{p}}\rangle; T \tag{6}$$
$$E_1 \overset{s:\mathtt{p}\to\mathtt{q}:T'_{\mathtt{p}}}{\longrightarrow} E_2 \tag{7}$$
$$\mathtt{proj}(s : G) \quad \supseteq \quad \{s[\mathtt{p}_i] : s\lceil \mathtt{p}_i\} \tag{8}$$

From 6 we get $\mathtt{proj}(E_1) \supseteq \Delta \cdot \{s[\mathtt{p}] : [\mathtt{q}]!\langle U\rangle; T \cdot s[\mathtt{q}] : [\mathtt{p}]?(U); T'\}$ and from 7 and lemma C.4 we get that $\mathtt{proj}(E_2) \supseteq \Delta \cdot \{s[\mathtt{p}] : T \cdot s[\mathtt{q}] : T'\} \supset \Delta \cdot s[\mathtt{p}] : T$. From 8 we get that $\mathtt{proj}(E_2 \cdot s : G) \supseteq \Delta \cdot s[\mathtt{p}] : T \cdot \{s[\mathtt{p}_i] : s\lceil \mathtt{p}_i\}$ as required.
The rest of the base cases are similar.
Inductive Step:
The inductive rule for environment configuration is [Inv]. Let $(E_1, \Gamma_1, \Delta_1) \xrightarrow{\ell} (E_2, \Gamma_2, \Delta_2)$.
From rule [Inv] we get:

$$E_1 \longrightarrow^* E'_1 \tag{9}$$
$$(E'_1, \Gamma_1, \Delta_1) \quad \xrightarrow{\ell} \quad (E_2, \Gamma_2, \Delta_2) \tag{10}$$

From the inductive hypothesis we know that for 10 $\exists E_3 \cdot E_2 \longrightarrow^* E_3$ and $\Delta_2 \subseteq \mathtt{proj}(E_3)$.
The result is then trivial from 9. □

Lemma C.5.

1. *If* $(E, \Gamma, \Delta_1) \xrightarrow{\ell} (E', \Gamma', \Delta_2)$ *then* $(\Gamma, \Delta_1) \xrightarrow{\ell} (\Gamma', \Delta_2)$
2. *If* $(E, \Gamma, \Delta_1) \xrightarrow{\ell} (E', \Gamma', \Delta'_1)$ *and* $\Delta_1 \rightleftharpoons \Delta_2$ *then* $(E, \Gamma, \Delta_2) \xRightarrow{\ell} (E', \Gamma', \Delta'_2)$
3. *If* $(\Gamma, \Delta_1) \xrightarrow{\ell} (\Gamma', \Delta_2)$ *then there exists E such that* $(E, \Gamma, \Delta) \xrightarrow{\ell} (E', \Gamma', \Delta_2)$
4. *If* $(E, \Gamma, \Delta \cdot s[\mathtt{p}] : T_{\mathtt{p}}) \xrightarrow{\ell} (E', \Gamma, \Delta' \cdot s[\mathtt{p}] : T_{\mathtt{p}})$ *then* $(E, \Gamma, \Delta) \xrightarrow{\ell} (E', \Gamma, \Delta')$
5. *If* $(E, \Gamma, \Delta_1) \xrightarrow{\ell} (E', \Gamma, \Delta_2)$ *then* $(E, \Gamma, \Delta_1 \cdot \Delta) \xrightarrow{\ell}_s (E', \Gamma, \Delta_2 \cdot \Delta)$
   provided that if $(E, \Gamma, \Delta) \xrightarrow{\ell'} (E, \Gamma, \Delta')$ *then* $\ell \not\asymp \ell'$

Proof. Part 1:
The proof for part 1 is easy to be implied by a case analysis on the configuration transition definition with respect to environment transition definition.

Part 2:

By the case analysis on ℓ.
Case $\ell = \tau$**:** The result is trivial.
Case $\ell = \overline{a}[\mathrm{p}](s)$ **or** $\ell = a[\mathrm{p}](s)$**:** The result comes from a simple transition.
Case $\ell = s[\mathrm{p}][\mathrm{q}]!\langle v\rangle$**:** $\Delta_1 \rightleftharpoons \Delta_2$ implies $\Delta_1 \longrightarrow^* \Delta$ and $\Delta_2 \longrightarrow^* \Delta$ for some Δ and $\Delta = \Delta' \cdot s[\mathrm{p}] : [\mathrm{q}]!\langle U\rangle;T$ for synchronous and input asynchronous MSP and $\Delta = \Delta' \cdot s[\mathrm{p}] : M;[\mathrm{q}]!\langle U\rangle$. for output and input/output asynchronous MSP .
$(E,\Gamma,\Delta_2) \Longrightarrow (E,\Gamma,\Delta) \xrightarrow{\ell}$ as required.
Case $\ell = s[\mathrm{p}][\mathrm{q}]!(s'[\mathrm{p}'])$**:** $\Delta_1 \rightleftharpoons \Delta_2$ implies $\Delta_1 \longrightarrow^* \Delta$ and $\Delta_2 \longrightarrow^* \Delta$ for some Δ and $\Delta = \Delta' \cdot s[\mathrm{p}] : [\mathrm{q}]!\langle T'\rangle;T$ for synchronous and input asynchronous MSP and $\Delta = \Delta' \cdot s[\mathrm{p}] : M;[\mathrm{q}]!\langle T'\rangle$. for output and input/output asynchronous MSP .
$(E,\Gamma,\Delta_2) \Longrightarrow (E,\Gamma,\Delta) \xrightarrow{\ell}$ as required.
The remaining cases are similar.

Part 3:
We do a case analysis on ℓ.
Cases $\ell = \tau, \ell = \overline{a}[\mathrm{p}](s), \ell = a[\mathrm{p}](s)$**:** The result holds for any E.
Case $\ell = s[\mathrm{p}][\mathrm{q}]!\langle v\rangle$: $\Delta_1 = \Delta_1' \cdot \Delta_1''$ with $\Delta_1'' = s[\mathrm{p}] : [\mathrm{q}]!\langle U\rangle;T_\mathrm{p} \cdot \ldots \cdot s[\mathrm{r}] : T_\mathrm{r}$ for synchronous and input asynchronous MSP. Choose $E = E' \cdot s : G$ with $*(\Delta_1'') \subseteq \mathtt{proj}(s:G)$ and $s[\mathrm{q}] : [\mathrm{p}]?(U);T_\mathrm{q} \in \mathtt{proj}(s:G)$ and $*(\Delta)_1 \subseteq \mathtt{proj}(E)$ By the definition of configuration transition relation, we obtain $(E,\Gamma,\Delta) \xrightarrow{\ell} (E,\Gamma',\Delta_2)$, as required.

$\Delta_1 = \Delta_1' \cdot \Delta_1''$ with $\Delta_1'' = s[\mathrm{p}] : T_\mathrm{p} \cdot s^o[\mathrm{p}] : M;[\mathrm{q}]!\langle U\rangle \cdot \ldots \cdot s[\mathrm{r}] : T_\mathrm{r}$ for output and input/output asynchronous MSP. Choose $E = E' \cdot s : G$ with $*(De_1'') \subseteq \mathtt{proj}(s:G)$ and $s[\mathrm{q}] : [\mathrm{p}]?(U);T_\mathrm{q} \in \mathtt{proj}(s:G)$ and $*(\Delta)_1 \subseteq \mathtt{proj}(E)$ By the definition of configuration transition relation, we obtain $(E,\Gamma,\Delta) \xrightarrow{\ell} (E,\Gamma',\Delta_2)$, as required.
Remaining cases are similar.

Part 4:
$(E,\Gamma,\Delta \cdot s[\mathrm{p}] : T_\mathrm{p}) \xrightarrow{\ell} (E',\Gamma,\Delta' \cdot s[\mathrm{p}] : T_\mathrm{p})$ implies that $s[\mathrm{p}] \notin \mathtt{subj}(\ell)$. The result then follows from the definition of configuration transition.

Part 5:
Case $\ell = \tau, \ell = \overline{a}[\mathrm{p}](s), \ell = a[\mathrm{p}](s)$**:** The result holds by definition of the configuration transition.
Case $\ell = s[\mathrm{p}][\mathrm{q}]!\langle U\rangle$**:** For synchronous and input asynchronous MSP we have that $\Delta_1 = \Delta_1' \cdot s[\mathrm{p}] : [\mathrm{q}]!\langle U\rangle;T$ and $E \xrightarrow{s:\mathrm{p}\to\mathrm{q}:U} E'$. For synchronous MSP assume $s[\mathrm{q}] \in \Delta$, then by definition of weak configuration pair we have $\Delta = \Delta'' \cdot s[\mathrm{q}] : \mathrm{q}[U][T]?()$; and $(E,\Gamma,\Delta) \xrightarrow{s[\mathrm{q}][\mathrm{p}]?\langle U\rangle}$. But this contradicts with the assumption $\ell \not\asymp \ell'$, so $s[\mathrm{q}] \notin \Delta$. By the definition of configuration pair transition we get that $(E,\Gamma,\Delta_1 \cdot \Delta) \xrightarrow{s[\mathrm{p}][\mathrm{q}]!\langle U\rangle} (E,\Gamma,\Delta_2 \cdot \Delta)$. For input asynchronous MSP assume that $s^i[\mathrm{q}] \in \Delta$, then by definition of weak configuration pair we have $\Delta = \Delta'' \cdot s^i[\mathrm{q}] : M$ and $(E,\Gamma,\Delta) \xrightarrow{s^i[\mathrm{q}][\mathrm{p}]?\langle U\rangle}$. But this contradicts with the assumption $\ell \not\asymp \ell'$, so $s^i[\mathrm{q}] \notin \Delta$. By the definition of configuration pair transition we get the result.
For output and input/output asynchronous MSP we have $\Delta_1 = \Delta_1' \cdot s^o[\mathrm{p}] : M;[\mathrm{q}]!\langle U\rangle$; and $E \xrightarrow{s:\mathrm{p}\to\mathrm{q}:U} E'$. For output asynchronous MSP assume $s[\mathrm{q}] \in \Delta$, then we would have $(E,\Gamma,\Delta) \xrightarrow{s[\mathrm{q}][\mathrm{p}]?\langle U\rangle}$ which contradicts with $\ell \not\asymp \ell'$ and $s[\mathrm{q}] \notin \Delta$ to get the required result by

the configuration pair definition. For input/output asynchronous MSP assume $s^i[\mathtt{q}] \in \Delta$, then we would have $(E, \Gamma, \Delta) \xrightarrow{s^i[\mathtt{q}][\mathtt{p}]?\langle U \rangle}$ which contradicts with $\ell \not\asymp \ell'$ and $s^i[\mathtt{q}] \notin \Delta$ to get the required result by the configuration pair definition.
Remaining cases are similar.

□

C.5 Proof for Lemma 5.1

Proof. Since we are dealing with closed processes, the interesting case is parallel composition. We need to show that if $E, \Gamma \vdash P \triangleright \Delta_1 \approx_g Q \triangleright \Delta_2$ then for all R such that $E, \Gamma \vdash P \mid R \triangleright \Delta_3, E, \Gamma \vdash Q \mid R \triangleright \Delta_4$ then $E, \Gamma \vdash P \mid R \triangleright \Delta_3 \approx_g Q \mid R \triangleright \Delta_4$.

Let

$$\begin{aligned} S = \{ & (E, \Gamma \vdash P \mid R \triangleright \Delta_3,\ E, \Gamma \vdash Q \mid R \triangleright \Delta_4) \mid \\ & E, \Gamma \vdash P \triangleright \Delta_1 \approx_g Q \triangleright \Delta_2, \\ & \forall R \cdot E, \Gamma \vdash P \mid R \triangleright \Delta_3, E, \Gamma \vdash Q \mid R \triangleright \Delta_4 \} \end{aligned}$$

Before we proceed to a case analysis, we extract general results. Let $\Gamma \vdash P \triangleright \Delta_1, \Gamma \vdash Q \triangleright \Delta_2, \Gamma \vdash R \triangleright \Delta_5, \Gamma \vdash P \mid R \triangleright \Delta_3, \Gamma \vdash Q \mid R \triangleright \Delta_4$ then from typing rule [Conc] we get

$$\Delta_3 = \Delta_1 \cup \Delta_5 \tag{11}$$

$$\Delta_4 = \Delta_2 \cup \Delta_5 \tag{12}$$

$$\Delta_1 \cap \Delta_5 = \emptyset \tag{13}$$

$$\Delta_2 \cap \Delta_5 = \emptyset \tag{14}$$

We prove that S is a bisimulation. There are three cases:
Case: $E, \Gamma \vdash P \mid R \triangleright \Delta_3 \xrightarrow{\ell} E', \Gamma \vdash P' \mid R \triangleright \Delta_3'$

From typed transition definition we have that:

$$P \mid R \xrightarrow{\ell} P' \mid R \tag{15}$$

$$(E, \Gamma, \Delta_3) \xrightarrow{\ell} (E', \Gamma, \Delta_3') \tag{16}$$

Transition (15) and rule $\langle \mathrm{Par} \rangle$ (LTS in Figure 5) imply:

$$P \xrightarrow{\ell} P' \tag{17}$$

From (11), transition (16) can be written as $(E, \Gamma, \Delta_1 \cup \Delta_5) \xrightarrow{\ell} (E', \Gamma, \Delta_1' \cup \Delta_5)$, to conclude from Lemma C.5 part 4, that:

$$(E, \Gamma, \Delta_1) \xrightarrow{\ell} (E', \Gamma, \Delta_1') \tag{18}$$

$$\mathtt{subj}(\ell) \ \notin\ \mathtt{dom}(\Delta_5) \tag{19}$$

Transitions 17 and 18 imply $E, \Gamma \vdash P \triangleright \Delta_1 \xrightarrow{\ell} E', \Gamma \vdash P' \triangleright \Delta_1'$. From the definition of set S we get that $E, \Gamma \vdash Q \triangleright \Delta_2 \xRightarrow{\ell} E', \Gamma \vdash Q' \triangleright \Delta_2'$.

From the typed transition definition we have that:

$$Q \overset{\ell}{\Longrightarrow} Q' \tag{20}$$

$$(E,\Gamma,\Delta_2) \overset{\ell}{\Longrightarrow} (E',\Gamma,\Delta_2') \tag{21}$$

From 19 and part 5 of Lemma C.5 we can write: $(E,\Gamma,\Delta_2 \cup \Delta_5) \overset{\ell}{\Longrightarrow} (E',\Gamma,\Delta_2' \cup \Delta_5)$, to imply from 20 that $E,\Gamma \vdash P \mid R \triangleright \Delta_4 \overset{\ell}{\Longrightarrow} E',\Gamma \vdash P' \mid R \triangleright \Delta_4'$ as required.
Case: 2

$$E,\Gamma \vdash P \mid R \triangleright \Delta_3 \xrightarrow{\tau} E' \vdash P' \mid R' \triangleright \Delta_3'$$

From the typed transition definition we have that:

$$P \mid R \xrightarrow{\tau} P' \mid R' \tag{22}$$

$$(E,\Gamma,\Delta_3) \xrightarrow{\tau} (E,\Gamma,\Delta_3') \tag{23}$$

From 22 and rule $\langle \text{Tau} \rangle$ we get

$$P \xrightarrow{\ell} P' \tag{24}$$

$$R \xrightarrow{\ell'} R' \tag{25}$$

From 11 transition 23 can be written $(E,\Gamma,\Delta_1 \cup \Delta_5) \xrightarrow{\tau} (E,\Gamma,\Delta_1' \cup \Delta_5')$, to conclude that

$$(E,\Gamma,\Delta_1) \xrightarrow{\ell} (E,\Gamma,\Delta_1') \tag{26}$$

$$(E,\Gamma,\Delta_5) \xrightarrow{\ell'} (E,\Gamma,\Delta_5') \tag{27}$$

From 24 and 26 we conclude that $E,\Gamma \vdash P \triangleright \Delta_1 \xrightarrow{\ell} E,\Gamma \vdash P' \triangleright \Delta_1'$ and from 25 and 27 $E,\Gamma \vdash R \triangleright \Delta_5 \xrightarrow{\ell} E,\Gamma \vdash R' \triangleright \Delta_5'$.
From the definition of set S we get that $E,\Gamma \vdash Q \triangleright \Delta_2 \overset{\ell}{\Longrightarrow} E,\Gamma \vdash Q' \triangleright \Delta_2'$, implies

$$Q \overset{\ell}{\Longrightarrow} Q' \tag{28}$$

$$(E,\Gamma,\Delta_2) \overset{\ell}{\Longrightarrow} (E,\Gamma,\Delta_2') \tag{29}$$

From 25 we get that $Q \mid R \overset{\tau}{\Longrightarrow} Q' \mid R'$ and $(E,\Gamma,\Delta_2 \cup \Delta_5) \overset{\tau}{\Longrightarrow} (E,\Gamma,\Delta_2' \cup \Delta_5')$, implies

$$E,\Gamma \vdash Q \mid R \triangleright \Delta_4 \overset{\tau}{\Longrightarrow} E' \vdash Q' \mid R' \triangleright \Delta_4'$$

Case: 3

$$E,\Gamma \vdash P \mid R \triangleright \Delta_3 \xrightarrow{\ell} E' \vdash P \mid R' \triangleright \Delta_3'$$

□

C.6 Proof for Lemma 5.2

Proof. We take into advantage the fact that bisimulation has a stratifying definition.

- $\approx_{g_0}$ is the union of all configuration relations, $E,\Gamma \vdash P \triangleright \Delta_1 \; \mathcal{R} \; Q \triangleright \Delta_2$.
- $E,\Gamma \vdash P \triangleright \Delta_1 \approx_{g_n} Q \triangleright \Delta_2$ if
  - $E,\Gamma \vdash P \triangleright \Delta_1 \xrightarrow{\ell} E',\Gamma \vdash P' \triangleright \Delta_1'$ then $E,\Gamma \vdash Q \triangleright \Delta_2 \overset{\ell}{\Longrightarrow} E',\Gamma \vdash Q \triangleright \Delta_2'$ and $E',\Gamma \vdash P' \triangleright \Delta_1' \approx_{g_{n-1}} Q' \triangleright \Delta_2'$
  - The symmetric case.
- $\approx_{g_n}^{\omega} = \bigcap_{0 \leq i \leq n} \approx_{g_i}$

From coinduction theory, we know that $(\bigcap_{\forall n} \approx_{g_n}) = \approx_g$.
To this purpose we define a set of tests $T\langle N, \vec{\ell_n}\rangle$ to inductively show that:

$$\begin{array}{ll} \text{If} & E,\Gamma \vdash P_1 \triangleright \Delta_1 \cong_g P_2 \triangleright \Delta_2 \text{ implies} \\ & E,\Gamma \vdash P_1 \mid T\langle N, \vec{\ell_n}\rangle \triangleright \Delta_1 \cong_g P_2 \mid T\langle N, \vec{\ell_n}\rangle \triangleright \Delta_2 \text{ implies} \\ \multicolumn{2}{l}{\forall n,\ E,\Gamma \vdash P_1 \triangleright \Delta_1 \approx_{g_n} P_2 \triangleright \Delta_2 \text{ implies}} \\ & E,\Gamma \vdash P_1 \triangleright \Delta_1 \approx_g P_2 \triangleright \Delta_2 \end{array}$$

We give the definition for $T\langle N, \vec{\ell_n}\rangle$:
$T\langle N, succ, \vec{\ell_n}\rangle = Q\langle N, n, \vec{\ell_i}\rangle \mid \ldots \mid Q\langle N, n, \vec{\ell_i}\rangle$
where

1. $i \in I$
2. $\bigcup_{i \in I} \vec{\ell_i} = \vec{\ell_n}$
3. $n \ ::= \ s[\mathrm{p}] \ \mid \ a.$
4. $N \ ::= \ \emptyset \ \mid \ N \cdot s[\mathrm{p}] \ \mid \ N \cdot a$ is a set of names for testing the receiving objects.

and let

- $B^s\langle s[\mathrm{p}]\rangle = \mathbf{0}$
- $B^i\langle s[\mathrm{p}]\rangle = s[\mathrm{p}][\mathtt{i} : \emptyset]$
- $B^o\langle s[\mathrm{p}]\rangle = s[\mathrm{p}][\mathtt{o} : \emptyset]$
- $B^{io}\langle s[\mathrm{p}]\rangle = s[\mathrm{p}][\mathtt{i} : \emptyset] \mid s[\mathrm{p}][\mathtt{o} : \emptyset]$

to define

- $Q\langle N, a, a[A](s) \cdot \vec{\ell_n}\rangle = \overline{a}[n](x).Q\langle N, s[n], \vec{\ell_i}\rangle \mid \ldots \mid a[\mathrm{p}](x).Q\langle N, s[\mathrm{p}], \vec{\ell_i}\rangle, i \in I.$
- $Q\langle N, s[\mathrm{q}], s[\mathrm{p}][\mathrm{q}]?\langle v\rangle \cdot \vec{\ell_n}\rangle = s[\mathrm{q}][\mathrm{p}]!\langle v\rangle; Q\langle N, s[\mathrm{q}], \vec{\ell_n}\rangle B\langle s[\mathrm{q}]\rangle.$
- $Q\langle N, s[\mathrm{q}], s[\mathrm{p}][\mathrm{q}]\& l \cdot \vec{\ell_n}\rangle = s[\mathrm{q}][\mathrm{p}] \oplus l; Q\langle N, s[\mathrm{q}], \vec{\ell_n}\rangle \mid B\langle s[\mathrm{q}]\rangle.$
- $Q\langle N, a, \overline{a}[A](s) \cdot \vec{\ell_n}\rangle = a[q](x).Q\langle N, s[q], \vec{\ell_i}\rangle \mid \ldots \mid a[\mathrm{p}](x).Q\langle N, s[\mathrm{p}], \vec{\ell_i}\rangle, i \in I.$
- $Q\langle N, s[\mathrm{q}], s[\mathrm{p}][\mathrm{q}]!\langle v\rangle \cdot \vec{\ell_n}\rangle$
  $= s[\mathrm{q}][\mathrm{p}]?(x); \mathtt{if}\ x \in N\ \mathtt{then}\ Q\langle N, s[\mathrm{q}], \vec{\ell_n}\rangle\ \mathtt{else}\ (\nu\, b)(b[1](x).Q\langle N, s[\mathrm{q}], \vec{\ell_n}\rangle) \mid B\langle s[\mathrm{q}]\rangle.$
- $Q\langle N, s[\mathrm{q}], s[\mathrm{p}][\mathrm{q}]!\langle s'[\mathrm{p}']\rangle \cdot \vec{\ell_n}\rangle$
  $= s[\mathrm{q}][\mathrm{p}]?(x); \mathtt{if}\ x \in N\ \mathtt{then}\ Q\langle N, s[\mathrm{q}], \vec{\ell_n}\rangle\ \mathtt{else}\ (\nu\, b)(b[1](x).Q\langle N, s[\mathrm{q}], \vec{\ell_n}\rangle) \mid B\langle s[\mathrm{q}]\rangle.$
- $Q\langle N, s[\mathrm{q}], s[\mathrm{p}][\mathrm{q}] \oplus l_k \cdot \vec{\ell_n}\rangle$
  $= s[\mathrm{q}][\mathrm{p}]\&\{l_k : Q\langle N, s[\mathrm{q}], \vec{\ell_n}\rangle,\ l_i : (\nu\ b)(b[1](x).Q\langle N, s[\mathrm{q}], \vec{\ell_n}\rangle)\} \mid B\langle s[\mathrm{q}]\rangle.$
- $Q\langle N, n, \emptyset\rangle = R.$

where $R = (\nu\ b)(b[1](x).R')$ or $R = \mathbf{0}$. R completes the session type on session channel n and is used to keep processes typed.

From the definition of $T\langle N, \vec{\ell_n}\rangle$ we can show that $\forall T\langle N, \ell \cdot \vec{\ell_n}\rangle, T\langle N, \ell \cdot \vec{\ell_n}\rangle \stackrel{\ell'}{\Longrightarrow} T'\langle N, \vec{\ell_n}\rangle, \ell \asymp \ell'$.

We prove the required result inductively:

$E,\Gamma \vdash P_1 \triangleright \Delta_3 \cong_g P_2 \triangleright \Delta_4$ implies
$\forall \ell \cdot \vec{\ell_n}$ choose $T\langle N, \ell \cdot \vec{\ell_n}\rangle, E,\Gamma \vdash P_1 \mid T\langle N, \ell \cdot \vec{\ell_n}\rangle \triangleright \Delta_1 \cong_g P_2 \mid T\langle N, \ell \cdot \vec{\ell_n}\rangle \triangleright \Delta_2$ implies
$E,\Gamma \vdash P_1 \mid T\langle N, \ell \cdot \vec{\ell_n}\rangle \triangleright \Delta_1 \twoheadrightarrow P_1' \mid T\langle N, \vec{\ell_n}\rangle \triangleright \Delta_1'$,
$E,\Gamma \vdash P_2 \mid T\langle N, \vec{\ell_n}\rangle \triangleright \Delta_2 \twoheadrightarrow P_2' \mid T\langle N, \vec{\ell_n}\rangle \triangleright \Delta_2'$ then by induction hypothesis
$P_1' \approx_{g_n} P_2'$ implies
$\forall n, E,\Gamma \vdash P_1 \triangleright \Delta_1 \approx_{g_n} P_2 \triangleright \Delta_2$ implies
$E,\Gamma \vdash P_1 \triangleright \Delta_1 \approx_g P_2 \triangleright \Delta_2$

We need to show that if

$$E,\Gamma \vdash P_1 \mid T\langle N, \ell \cdot \vec{\ell_n}\rangle \triangleright \Delta_1 \cong_g P_2 \mid T\langle N, \ell \cdot \vec{\ell_n}\rangle \triangleright \Delta_2$$

then

$E,\Gamma \vdash P_1 \mid T\langle N, \ell \cdot \vec{\ell_n}\rangle \triangleright \Delta_1 \twoheadrightarrow P_1' \mid T\langle N, \vec{\ell_n}\rangle \triangleright \Delta_1', E,\Gamma \vdash P_2 \mid T\langle N, \vec{\ell_n}\rangle \triangleright \Delta_2 \twoheadrightarrow P_2' \mid T\langle N, \vec{\ell_n}\rangle \triangleright \Delta_2'$

We perform a case analysis on $E,\Gamma \vdash P_1 \triangleright \Delta_3 \xrightarrow{\ell} P_1 \triangleright \Delta_3'$:

- $E,\Gamma \vdash P_1 \triangleright \Delta_3 \xrightarrow{s[\mathrm{p}][\mathrm{q}]?\langle v\rangle} P_1 \triangleright \Delta_3'$ implies, $E,\Gamma \vdash P_1 \mid T\langle N, s[\mathrm{p}][\mathrm{q}]?\langle v\rangle \cdot \vec{\ell_n}\rangle \triangleright \Delta_1 = E,\Gamma \vdash P_1 \mid Q\langle N, s[\mathrm{p}], s[\mathrm{p}][\mathrm{q}]?\langle v\rangle \cdot \vec{\ell_i}\rangle \mid \ldots \mid Q\langle N, n, \vec{\ell_i}\rangle \triangleright \Delta_1 \longrightarrow P_1' \mid Q\langle N, s[\mathrm{p}], \vec{\ell_i}\rangle \mid \ldots \mid Q\langle N, n, \vec{\ell_i}\rangle \triangleright \Delta_1$.

  $E,\Gamma \vdash P_2 \mid T\langle N, s[\mathrm{p}][\mathrm{q}]?\langle v\rangle \cdot \vec{\ell_n}\rangle \triangleright \Delta_2$ needs to match the reduction, $E,\Gamma \vdash P_2 \mid T\langle N, s[\mathrm{p}][\mathrm{q}]?\langle v\rangle \cdot \vec{\ell_n}\rangle \triangleright \Delta_2 \twoheadrightarrow E,\Gamma \vdash P_2''' \mid T\langle N, s[\mathrm{p}][\mathrm{q}]?\langle v\rangle \cdot \vec{\ell_n}\rangle \triangleright \Delta_2''' \longrightarrow E,\Gamma \vdash P_2'' \mid T'\langle N, \vec{\ell_n}\rangle \triangleright \Delta_2'' \rightarrow$
  $\rightarrow E,\Gamma \vdash P_2' \mid T'\langle N, \vec{\ell_n}\rangle \triangleright \Delta_2'$
- $E,\Gamma \vdash P_1 \triangleright \Delta_3 \xrightarrow{a[A](s)} P_1 \vdash \Delta_3' \triangleright$ implies, $E,\Gamma \vdash P_1 \mid T\langle N, a[A](s) \cdot \vec{\ell_n}\rangle \triangleright \Delta_1 = E,\Gamma \vdash P_1 \mid Q\langle N, a, a[A](s) \cdot \vec{\ell_i}\rangle \mid \ldots \mid Q\langle N, n, \vec{\ell_i}\rangle \triangleright \Delta_1 \longrightarrow P_1' \mid Q\langle N, a, \vec{\ell_i}\rangle \mid \ldots \mid Q\langle N, n, \vec{\ell_i}\rangle \vdash \Delta_1 \triangleright$.

  $E,\Gamma \vdash P_2 \mid T\langle N, a[A](s) \cdot \vec{\ell_n}\rangle \triangleright \Delta_2$ needs to match the reduction $E,\Gamma \vdash P_2 \mid T\langle N, a[A](s) \cdot \vec{\ell_n}\rangle \triangleright \Delta_2 \twoheadrightarrow E,\Gamma \vdash P_2''' \mid T\langle N, a[A](s) \cdot \vec{\ell_n}\rangle \triangleright \Delta_2''' \longrightarrow E,\Gamma \vdash P_2'' \mid T'\langle N, \vec{\ell_n}\rangle \triangleright \Delta_2'' \twoheadrightarrow$
  $E,\Gamma \vdash P_2' \mid T'\langle N, \vec{\ell_n}\rangle \triangleright \Delta_2'$
- $E,\Gamma \vdash P_1 \triangleright \Delta_3 \xrightarrow{s[\mathrm{p}][\mathrm{q}]!\langle v\rangle} P_1 \vdash \Delta_3' \triangleright$ implies, $E,\Gamma \vdash P_1 \mid T\langle N, s[\mathrm{p}][\mathrm{q}]!\langle v\rangle \cdot \vec{\ell_n}\rangle \triangleright \Delta_1 = E,\Gamma \vdash P_1 \mid Q\langle N, s[\mathrm{p}], s[p][q]!\langle v\rangle \cdot \vec{\ell_i}\rangle \mid \ldots \mid Q\langle N, n, \vec{\ell_i}\rangle \triangleright \Delta_1 \twoheadrightarrow P_1' \mid Q\langle N, s[\mathrm{p}], \vec{\ell_i}\rangle \mid \ldots \mid Q\langle N, n, \vec{\ell_i}\rangle \vdash \Delta_1 \triangleright$.

  $E,\Gamma \vdash P_2 \mid T\langle N, s[\mathrm{p}][\mathrm{q}]!\langle v\rangle \cdot \vec{\ell_n}\rangle \triangleright \Delta_2$ needs to match the reduction, $E,\Gamma \vdash P_2 \mid T\langle N, s[\mathrm{p}][\mathrm{q}]!\langle v\rangle \cdot \vec{\ell_n}\rangle \triangleright \Delta_2 \twoheadrightarrow E,\Gamma \vdash P_2''' \mid T\langle N, s[\mathrm{p}][\mathrm{q}]!\langle v\rangle \cdot \vec{\ell_n}\rangle \triangleright \Delta_2''' \twoheadrightarrow E,\Gamma \vdash P_2'' \mid T'\langle N, \vec{\ell_n}\rangle \triangleright \Delta_2'' \rightarrow$
  $\rightarrow E,\Gamma \vdash P_2' \mid T'\langle N, \vec{\ell_n}\rangle \triangleright \Delta_2'$

□

C.7 Proof for Lemma 5.2

Proof. We prove direction if $\forall E, E, \Gamma \vdash P_1 \triangleright \Delta_1 \approx_g P_2 \triangleright \Delta_2$ then $\Gamma \vdash P_1 \triangleright \Delta_1 \approx^s \Gamma \vdash P_2 \triangleright \Delta_2$.
If $\Gamma \vdash P_1 \triangleright \Delta_1 \xrightarrow{\ell} P_1' \triangleright \Delta_1'$ then $P_1 \xrightarrow{\ell} P_1'$ and $(\Gamma, \Delta_1) \xrightarrow{\ell} (\Gamma', \Delta_1')$.

From part 3 of Lemma C.5 we choose E such that $(E, \Gamma, \Delta_1) \xrightarrow{\ell} (E', \Gamma', \Delta_1')$. Since $\forall E, E, \Gamma \vdash P_1 \triangleright \Delta_1 \approx_g P_2 \triangleright \Delta_2$ it can now be implied that, $E, \Gamma \vdash P_1 \triangleright \Delta_1 \xrightarrow{\ell} E', \Gamma \vdash P_1' \triangleright \Delta_1'$ implies, $E, \Gamma \vdash P_2 \triangleright \Delta_2 \overset{\ell}{\Longrightarrow} E', \Gamma \vdash P_2' \triangleright \Delta_2'$ implies, $P_2 \overset{\ell}{\Longrightarrow} P_2'$ and $(E, \Gamma, \Delta_2) \overset{\ell}{\Longrightarrow} (E', \Gamma', \Delta_2')$.
From part 1 of Lemma C.5 we get $(\Gamma, \Delta_2) \overset{\ell}{\Longrightarrow} (\Gamma', \Delta_2')$ implies $\Gamma \vdash P_2 \triangleright \Delta_2 \overset{\ell}{\Longrightarrow} P_2' \triangleright \Delta_2'$ as required.

We prove direction if $\Gamma \vdash P_1 \triangleright \Delta_1 \approx^s \Gamma \vdash P_2 \triangleright \Delta_2$ then $\forall E, E, \Gamma \vdash P_1 \triangleright \Delta_1 \approx_g P_2 \triangleright \Delta_2$.
Let $E, \Gamma \vdash P_1 \triangleright \Delta_1 \xrightarrow{\ell} P_1' \triangleright \Delta_1'$ then

$$P_1 \xrightarrow{\ell} P_1' \tag{30}$$

$$(E, \Gamma, \Delta_1) \xrightarrow{\ell} (E', \Gamma', \Delta_1') \tag{31}$$

If $\Gamma \vdash P_1 \triangleright \Delta_1 \xrightarrow{\ell} P_1' \triangleright \Delta_1'$ then $P_1 \xrightarrow{\ell} P_1', (\Gamma, \Delta_1) \xrightarrow{\ell} (\Gamma', \Delta_1'), \Gamma \vdash P_2 \triangleright \Delta_2 \xrightarrow{\ell} P_2' \triangleright \Delta_2'$
From the last implication we get

$$P_2 \overset{\ell}{\Longrightarrow} P_2' \tag{32}$$

$$(\Gamma, \Delta_2) \overset{\ell}{\Longrightarrow} (\Gamma', \Delta_2') \tag{33}$$

$$\Delta_1 \rightleftharpoons \Delta_2 \tag{34}$$

We apply part 2 of Lemma C.5 to 31 and 34 to get $(E, \Gamma, \Delta_2) \overset{\ell}{\Longrightarrow} (E', \Gamma', \Delta_2')$. From the last result and 32 we get $E, \Gamma \vdash P_2 \triangleright \Delta_2 \overset{\ell}{\Longrightarrow} E', \Gamma \vdash P_2' \triangleright \Delta_2'$.

C.8 Proof for theorem 5.3

Proof. We follow the requirement of part 3 of lemma C.5 to show that if P is simple and $\Gamma \vdash P \triangleright \Delta \xrightarrow{\ell} \Gamma \vdash P' \triangleright \Delta'$ then $\exists E \cdot E, \Gamma \vdash P \triangleright \Delta \xrightarrow{\ell} E', \Gamma \vdash P' \triangleright \Delta'$.

From that point on we apply part 2 of lemma C.5 to get that if P_1, P_2 are simple and $\exists E \cdot E, \Gamma \vdash P_1 \triangleright \Delta_1 \approx_g P_2 \triangleright \Delta_2$ then $\forall E, E, \Gamma \vdash P_1 \triangleright \Delta_1 \approx_g P_2 \triangleright \Delta_2$. By applying lemma 5.2 we are done. □

D Output Asynchronous MSP

We extend the synchronous MSP to define the semantics for output asynchronous Multiparty Session π calculus. Semantics are based in the system developed in [3].

D.1 Syntax

$$\begin{array}{rcll} P & ::= & \vdots & \\ & & s[\mathsf{p}][\mathsf{o} : \vec{h}] & (\text{ConfigurationO}) \\ & & & \\ h & ::= & [\mathsf{p}](v) \;\mid\; [\mathsf{p}]l \;\mid\; [\mathsf{p}](s[q]) & (\text{Message}) \end{array}$$

We extend the syntax of synchronous MSP in Figure 2 with rule (ConfigurationO) to define the output configuration construct for session endpoints. Output configurations store messages $\vec{h}$ (defined in rule (Message)). Message h has the general form $[\mathsf{p}](v)$ that describes the passed value (i.e. $v, l, s[q]$) and the receiver role p. A process with no session names (i.e it does not include free session names, session endpoint configurations and session restriction), present in its syntax is called program.

Structural Congruence

$$\begin{array}{cc} \vdots & \\ s[\mathsf{p}][\mathsf{o} : \vec{h} \cdot [\mathsf{q}](v) \cdot [\mathsf{q}'](v') \cdot \vec{h}] \equiv s[\mathsf{p}][\mathsf{o} : \vec{h} \cdot [\mathsf{q}'](v') \cdot [\mathsf{q}](v) \cdot \vec{h}] & \mathsf{q} \neq \mathsf{q}' \\ (\nu\, s[\mathsf{p}])(s[\mathsf{p}][\mathsf{o} : \varepsilon]) \equiv \mathbf{0} & \end{array}$$

We extend structural congruence in Figure 11 to include message permutations inside output configurations. Messages are permuted upto structural congruence if their recipients are different. The structural rule for message permutation is at the core of the asynchronous behaviour of output asynchronous MSP. The second structural congruence rule is used for garbage collection of inactive output configurations.

D.2 Operational Semantics

$$\begin{array}{rll} a[1](x).P_1 \mid \ldots \mid \overline{a}[n](x).P_n \longrightarrow_o (\nu\, s)(P_1\{s[1]/x\} \mid \ldots \mid P_n\{s[n]/x\} \mid s[\mathsf{p}][\mathsf{o} : \varepsilon] \mid \ldots \mid s[n][\mathsf{o} : \varepsilon]) & & [\text{Link}] \\ s[\mathsf{p}][\mathsf{q}]!\langle v\rangle; P \mid s[\mathsf{p}][\mathsf{o} : \vec{h}] \longrightarrow_o P \mid s[\mathsf{p}][\mathsf{o} : [\mathsf{q}](v) \cdot \vec{h}] & & [\text{Send}] \\ s[\mathsf{p}][\mathsf{q}]?(x); P \mid s[\mathsf{q}][\mathsf{o} : \vec{h} \cdot [\mathsf{p}](v)] \longrightarrow_o P\{v/x\} \mid s[\mathsf{q}][\mathsf{o} : \vec{h}] & & [\text{Rcv}] \\ s[\mathsf{p}][\mathsf{q}]?(x); P \mid s[\mathsf{q}][\mathsf{o} : \vec{h} \cdot [\mathsf{p}](s'[\mathsf{p}'])] \longrightarrow_o P\{s'[\mathsf{p}']/x\} \mid s[\mathsf{q}][\mathsf{o} : \vec{h}] & & [\text{Rcv-S}] \\ s[\mathsf{p}][\mathsf{q}] \oplus l; P \mid s[\mathsf{p}][\mathsf{o} : \vec{h}] \longrightarrow_o P \mid s[\mathsf{p}][\mathsf{o} : [\mathsf{q}]l \cdot \vec{h}] & & [\text{Sel}] \\ s[\mathsf{p}][\mathsf{q}]\&\{l_i : P_i\}_{i\in I} \mid s[\mathsf{q}][\mathsf{o} : [\mathsf{p}]l_k \cdot \vec{h}] \longrightarrow_o P_k \mid s[\mathsf{q}][\mathsf{o} : \vec{h}] & & [\text{Bra}] \end{array}$$

Rule [Link] describes session initiation. All session participants should be present before each participant p synchronously reduces to create a fresh role $s[\mathsf{p}]$ and the corresponding output configurations $s[\mathsf{p}][\mathsf{o} : \varepsilon]$. Session communication is described as session configuration interactions. Rule [Send] describes an enqueue operation from role $s[\mathsf{p}]$ of a value v as a message $[\mathsf{q}](v)$ in session configuration $s[\mathsf{p}][\mathsf{o} : \vec{h}]$. Dually rule [Rcv] describes the dequeue operation and reception of a value v from role $s[\mathsf{q}]$ out of session configuration $s[\mathsf{q}][\mathsf{o} : \vec{h} \cdot [\mathsf{p}](va)]$. The reception happens on the substitution of value v on variable x on the continuation process of the receive action. Rule [Sel] and [Bra] send and receive labels l interacting with the session endpoints (in a similar way with rules [Send] and [Rcv] respectively) to perform select and branch operations respectively. A branch operation upon the reception of a label, decides the continuation of the process with respect to the label received. Operational semantics are completed with the standard π calculus rules (cf. §2).

D.3 Typing System for Programs

We use the syntax and definitions for global types and local types as defined for the synchronous MSP in §3.

We use global types described in Figure 4 (§3.1). We use the global and local projection definitions and duality defined in §3.2.

As in §3.3 use the definition of the typing judgement:

$$\Gamma \vdash e : S \quad \text{and} \quad \Gamma \vdash P \triangleright \Delta$$

We use the notion of coherency in Definition 3.3.

The typing rules for programs is the set of rules found in Figure 10 with the exclusion of rule [SRes]. The typing rules description is found in §A. Note that the typing system for output asynchronous MSP programs is identical to the typing system for synchronous MSP processes without session restriction, since session restriction is not included in program syntax.

D.4 Typing System for Runtime Processes

A runtime process is a closed output asynchronous multiparty session π calculus term. We extend the typing system for programs to type output session configurations We start by extending the linear session environment Δ with the message type M:

$$\Delta \ ::= \ \Delta \cdot c[\mathrm{p}] : T \ \mid \ s^o[\mathrm{p}] : M \ \mid \ \emptyset$$

where

$$M \ ::= \ \emptyset \ \mid \ [\mathrm{q}]!\langle U \rangle; M \ \mid \ [\mathrm{q}] \oplus l; M$$

Δ is extended to include configuration endpoints $s^o[\mathrm{p}]$ mapped to the message type M. A message type is defined as a sequence of output message types $[\mathrm{q}]!\langle U \rangle$ and select message types $[\mathrm{q}] \oplus l$.

We define a permutation relation over message types.

Definition D.1 (Message Type Permutation). *Let* $\mathrm{p} \neq \mathrm{q}$. *We then define:*

$$M; [\mathrm{p}]!\langle U \rangle; [\mathrm{q}]!\langle U' \rangle; M' \sim M; [\mathrm{q}]!\langle U' \rangle; [\mathrm{p}]!\langle U \rangle; M'$$
$$M; [\mathrm{p}] \oplus l_i; [\mathrm{q}] \oplus l_j; M' \sim M; [\mathrm{q}] \oplus l_j; [\mathrm{p}] \oplus l_i; M'$$
$$M; [\mathrm{p}]!\langle U \rangle; [\mathrm{q}] \oplus l; M' \sim M; [\mathrm{q}] \oplus l; [\mathrm{p}]!\langle U \rangle; M'$$

with $\approx = \sim *$.

A message type sequence can be permuted on two message types if they have different recipients. Message permutation is defined following the structural congruence rule for message permutation.

We define a concatenation operator $*$ between message types M and local types T. The result of the concatenation operator is a local type T. We use the concatenation operator to reconstruct a complete local type $T = M * T'$out of the local type $s[\mathrm{p}] : T'$ of a process and the message type $s^o[\mathrm{p}] : M$.

Definition D.2 (Message Type Concatenation).

$$\emptyset * T = T \quad [\mathrm{q}]!\langle U \rangle; M * T = [\mathrm{q}]!\langle U \rangle; (M * T) \quad [\mathrm{q}] \oplus l; M * T = [\mathrm{q}] \oplus \{l_i : M * T\}_{i \in I}$$

The $*$ operates as follows: The message type prefix is used as a prefix for the resulting local type. The operation proceeds inductively on the sequence of the message type to reconstruct the session role local type.

We proceed with the definition of the runtime typing system:

$$\Gamma \vdash s[\mathtt{p}][\mathtt{o} : \varepsilon] \triangleright s^o[\mathtt{p}] : \emptyset \qquad (\text{QEmpty})$$

$$\frac{\Gamma \vdash P \triangleright \Delta \cdot s^o[\mathtt{p}] : M \quad M \approx M'}{\Gamma \vdash P \triangleright \Delta \cdot s^o[\mathtt{p}] : M'} \; (\text{Equiv})$$

$$\frac{\Gamma \vdash s[\mathtt{p}][\mathtt{o} : \vec{h}] \triangleright \Delta \cdot s^o[\mathtt{p}] : M}{\Gamma \vdash s[\mathtt{p}][\mathtt{o} : [\mathtt{q}](v) \cdot \vec{h}] \triangleright \Delta \cdot s^o[\mathtt{p}] : [\mathtt{q}]!\langle S \rangle; M} \; (\text{QVal})$$

$$\frac{\Gamma \vdash s[\mathtt{p}][\mathtt{o} : \vec{h}] \triangleright \Delta \cdot s^o[\mathtt{p}] : M}{\Gamma \vdash s[\mathtt{p}][\mathtt{o} : [\mathtt{q}]l \cdot \vec{h}] \triangleright \Delta \cdot s^o[\mathtt{p}] : [\mathtt{q}] \oplus l; M} \; (\text{QSel})$$

$$\frac{\Gamma \vdash s[\mathtt{p}][\mathtt{o} : \vec{h}] \triangleright \Delta \cdot s^o[\mathtt{p}] : M}{\Gamma \vdash s[\mathtt{p}][\mathtt{o} : [\mathtt{q}](s'[\mathtt{q}']) \cdot \vec{h}] \triangleright \Delta \cdot s^o[\mathtt{p}] : [\mathtt{q}]!\langle T \rangle; M} \; (\text{QDel})$$

$$\frac{\Gamma \vdash P_1 \triangleright \Delta_1 \quad \Gamma \vdash P_2 \triangleright \Delta_2 \quad \mathtt{dom}(\Delta_1) \cup \mathtt{dom}(\Delta_2) = \emptyset}{\Gamma \vdash P_1 \mid P_1 \triangleright \Delta_1 \cdot \Delta_2} \; (\text{QConc})$$

$$\frac{\begin{array}{c}\Gamma \vdash P \triangleright \Delta \cdot s[1] : T_1 \cdot s^o[1] : M_1 \ldots s[n] : T_n \cdot s^o[n] : M_n \\ \mathtt{co}(\{s[1] : M_1 * T_1 \ldots s[n] : M_n * T_n\})\end{array}}{\Gamma \vdash (\nu\, s)P \triangleright \Delta} \; (\text{SRes})$$

In rule (QEmpty) we map empty session configurations to the empty message type $\emptyset$. Rule (QVal) (and rules (QSel), (QDel)) requires an inductive typing of the session configuration $s^o[\mathtt{p}]$ without its message prefix $[\mathtt{q}](v)$ ($[\mathtt{q}]l, [\mathtt{q}](s'[\mathtt{p}'])$ respectively) to get the type mapping $s^o[\mathtt{p}] : M$. The resulting message type for $s^o[\mathtt{p}]$ is prefixed with the message type of value v together with the receiver $\mathtt{q}$ to get $s^o[\mathtt{p}] : [\mathtt{q}]!\langle U \rangle; M$. For rule (QSel) we prefix with the select message type $[\mathtt{q}] \oplus l$ and for rule (QDel) we prefix with $[\mathtt{q}]!\langle s'[\mathtt{p}'] \rangle$. The parallel operator in rule (Conc) is identical to the parallel operator for programs and requires for disjoint linear session environments of the two operands. The result typing is the union of the two linear session environments. Rule (Equiv) requires that a runtime process can be typed upto message permutation. We use rule (SRes) for restricting a session name. We first require to construct the local types for all the session roles $s[\mathtt{p}]$ using the $*$ operator to concatenate the message type $s^o[\mathtt{p}]M$ and $s[\mathtt{p}]T$. Finally before we restrict we check the resulting local types to be coherent.

D.5 Type Soundness

We define the reduction semantics for local types. Since session environments represent the forthcoming communications, by reducing processes session environments can change. This can be formalised as in [3, 10] by introducing the notion of reduction of session environments, whose rules are:

Definition D.3 (Session Environment Reduction).

$$\begin{array}{rl}
\{s[\mathrm{p}] : [\mathrm{q}]!\langle U\rangle;T \cdot s^o[\mathrm{p}] : M\} & \longrightarrow_o \{s[\mathrm{p}] : T \cdot s^o[\mathrm{p}] : [\mathrm{q}]!\langle U\rangle;M\} \\
\{s[\mathrm{q}] : [\mathrm{p}]?(U);T \cdot s^o[\mathrm{p}] : M;[\mathrm{p}]!\langle U\rangle\} & \longrightarrow_o \{s[\mathrm{q}] : T \cdot s^o[\mathrm{p}] : M\} \\
\{s[\mathrm{p}] : [\mathrm{q}] \oplus \{l_i : T_i\}_{i\in I} \cdot s^o[p] : M\} & \longrightarrow_o \{s[\mathrm{p}] : T_k \cdot s^o[\mathrm{p}] : [\mathrm{q}] \oplus l_k;M\} \\
\{s[\mathrm{q}] : [\mathrm{p}]\&\{l_i : T_i\}_{i\in I} \cdot s^o[\mathrm{p}] : M;[\mathrm{p}] \oplus l_k\} & \longrightarrow_o \{s[\mathrm{q}] : T_k \cdot s^o[\mathrm{p}] : M\}
\end{array}$$

We write $\twoheadrightarrow = \longrightarrow^*$.

Definition D.4. *Let session typing environment Δ. We define*

$$\begin{array}{rl}
*(\Delta) = & \{s[\mathrm{p}] : T \mid s[\mathrm{p}] : T \in \Delta, s^o[\mathrm{p}] \notin \mathtt{dom}(\Delta)\} \\
& \cup \{s^o[\mathrm{p}] : M \mid s^o[\mathrm{p}] : M \in \Delta, s[\mathrm{p}] \notin \mathtt{dom}(\Delta)\} \\
& \cup \{s[\mathrm{p}] : M^o * T \mid s[\mathrm{p}] : T, s^o[\mathrm{p}] : M^o \in \Delta, s^i[\mathrm{p}] \notin \mathtt{dom}(\Delta)\}
\end{array}$$

The $*(\Delta)$ operator reconstructs the local types for the session roles inside a linear session environment. It uses the $*$ operator to concatenate roles $s^o[\mathrm{p}]M$ with $s[\mathrm{p}]T$. The resulting linear session environment is used for coherency checking in the subject reduction theorem.

Lemma D.1. *If* $\mathtt{co}(\Delta \cdot s[\mathrm{p}] : [\mathrm{q}]!\langle U\rangle;T)$ *and* $s[\mathrm{q}] \notin \mathtt{dom}(\Delta)$ *then* $\mathtt{co}(\Delta \cdot s[\mathrm{p}] : T)$.

Proof. Let $\mathtt{q}' \neq \mathtt{q}$ and $s[\mathrm{q}'] : T_{\mathtt{q}'} \in \Delta$. Then $[\mathrm{q}]!\langle U\rangle;T\lceil \mathrm{q}' = T\lceil \mathrm{q}'$. Since $s[\mathrm{q}] \notin \mathtt{dom}(\Delta)$ then for all $s[\mathrm{q}'] \in \mathtt{dom}(\Delta)$ $[\mathrm{q}]!\langle U\rangle;T\lceil \mathrm{q}' = T\lceil \mathrm{q}'$, so $\mathtt{co}(\Delta \cdot s[\mathrm{p}] : T)$.

Theorem D.1 (Subject Reduction). *Let* $\Gamma \vdash P \triangleright \Delta$ *with* $\mathtt{co}(*(\Delta))$ *and if* $P \twoheadrightarrow P'$ *then* $\Gamma \vdash P' \triangleright \Delta'$ *with* $\Delta \twoheadrightarrow \Delta'$ *and* $\mathtt{co}(*(\Delta'))$.

Proof. We apply induction on the length of the reduction $\twoheadrightarrow$. Induction is done by a case analysis on the reduction rules.
Case: [Link]

$P = a[\mathrm{p}](x_1).P_1 \mid \ldots \mid \overline{a}[n](x).P_n$. We apply typing rules [Accept], [Request], [Conc] to get $\Gamma \vdash P \triangleright \Delta$ with $\mathtt{co}(*(\Delta))$.

$P \longrightarrow P' = (\nu\ s)(P_1\{s[1]/x_1\} \mid \ldots \mid P_n\{s[n]/x_n\}) \mid s[1][\mathtt{o} : \varepsilon] \mid \ldots \mid s[n][\mathtt{o} : \varepsilon]$. We apply typing rules [Accept], [Request], (Qemp), [Conc] to get $\Gamma \vdash P \triangleright \Delta \cdot s[1] : T_1 \ldots s[n] : T_n \cdot s^o[1] : \emptyset \cdot \ldots \cdot s^o[n] : \emptyset$. We apply (SRes) to get $\Gamma \vdash P \triangleright \Delta$ as required.
Case: [Send]

$P = s[\mathrm{p}][\mathrm{q}]!\langle v\rangle;P_1 \mid s[\mathrm{p}][\mathtt{o} : h]$. We type to get $\Gamma \vdash P \triangleright \Delta$ with $\Delta = \Delta_1 \cdot s[\mathrm{p}] : [\mathrm{q}]!\langle U\rangle;T \cdot s^o[\mathrm{p}] : M$. $*(\Delta) = *(\Delta_1) \cup \{s[\mathrm{p}] : M * [\mathrm{q}]!\langle U\rangle;T\}$.

$P \longrightarrow P' = P_1 \mid s[\mathrm{p}][\mathtt{o} : [\mathrm{q}](v)h]$ with $\Gamma \vdash P' \triangleright \Delta'$ with $\Delta' = \Delta_1 \cdot s[\mathrm{p}] : T \cdot s^o[\mathrm{p}] : [\mathrm{q}]!\langle U\rangle;M$. $*(\Delta') = *(\Delta_1) \cup \{s[\mathrm{p}] : M * [\mathrm{q}]!\langle U\rangle;T\} = *(\Delta)$ as required.

D.6 Behavioural Semantics for Output Asynchronous MSP

We extend the label definition ℓ in §4.1 to include action labels on output configurations:

$$\begin{array}{rl}
\ell = & \vdots \\
& \mid\ s^o[\mathrm{p}][\mathrm{q}]!\langle v\rangle \ \mid\ s^o[\mathrm{p}][\mathrm{q}]!(v) \ \mid\ s^o[\mathrm{p}][\mathrm{q}]?\langle v\rangle \ \mid\ s^o[\mathrm{p}][\mathrm{q}] \oplus l \ \mid\ s^o[\mathrm{p}][\mathrm{q}]\&l
\end{array}$$

Labels $s^o[\mathtt{p}][\mathtt{q}]!\langle v\rangle$ and $s^o[\mathtt{p}][\mathtt{q}]!(v)$ denote the output of value v (output of bound value v) from session configuration $s^o[\mathtt{p}]$ to participant q. Dually action $s^o[\mathtt{p}][\mathtt{q}]?\langle v\rangle$ denotes the reception of value v by session configuration $s^o[\mathtt{p}]$ send by participant q. Actions $s^o[\mathtt{p}][\mathtt{q}]\oplus l$ and $s^o[\mathtt{p}][\mathtt{q}]\& l$ respectively describe the send (select) and receive (branch) of label l from participant p to participant q.

We use the definitions for *role set A* and $\max(A)$ from §4.1.

We define the duality relation $\asymp$ between labels:

$$\begin{array}{ll}
a[A](s) \asymp \overline{a}[A'](s) & \\
s[\mathtt{p}][\mathtt{q}]!\langle v\rangle \asymp_o s^o[\mathtt{p}][\mathtt{q}]?\langle v\rangle & s^o[\mathtt{p}][\mathtt{q}]!\langle v\rangle \asymp_o s[\mathtt{q}][\mathtt{p}]?\langle v\rangle \\
s[\mathtt{p}][\mathtt{q}]!(v) \asymp_o s^o[\mathtt{p}][\mathtt{q}]?\langle v\rangle & s^o[\mathtt{p}][\mathtt{q}]!(v) \asymp_o s[\mathtt{q}][\mathtt{p}]?\langle v\rangle \\
s[\mathtt{p}][\mathtt{q}]\oplus l \asymp_o s^o[\mathtt{p}][\mathtt{q}]\& l & s^o[\mathtt{p}][\mathtt{q}]\oplus l \asymp_o s[\mathtt{q}][\mathtt{p}]\& l
\end{array}$$

Accept $a[A](s)$ and request $\overline{a}[A'](s)$ labels are defined as dual. Process output actions $s[\mathtt{p}][\mathtt{q}]!\langle v\rangle$ interact with the corresponding session configuration input actions $s^o[\mathtt{p}][\mathtt{q}]?\langle v\rangle$. Similarly for process bound output. Process input actions $s[\mathtt{q}][\mathtt{p}]?\langle v\rangle$ interact with the corresponding session configuration output action ($s^o[\mathtt{p}][\mathtt{q}]!\langle v\rangle$) of the receiver participant q. Similarly when the session configuration output action is bound. Select and branching label duality follows the value send and receive semantics.

The labelled transition system is extended from the synchronous MSP LTS in Figure 5 (description in §4.1) to define actions on output session configurations.

$$\begin{array}{llll}
\langle\text{QSendO}\rangle & s[\mathtt{p}][\mathtt{o}: h\cdot[\mathtt{q}](v)] \xrightarrow{s^o[\mathtt{p}][\mathtt{q}]!\langle v\rangle} s[\mathtt{p}][\mathtt{o}: h] & \langle\text{QRcvO}\rangle & s[\mathtt{p}][\mathtt{o}: h] \xrightarrow{s^o[\mathtt{p}][\mathtt{q}]?\langle v\rangle} s[\mathtt{p}][\mathtt{o}: [\mathtt{q}](v)\cdot h] \\
\langle\text{QSelO}\rangle & s[\mathtt{p}][\mathtt{o}: h\cdot[\mathtt{q}]l] \xrightarrow{s^o[\mathtt{p}][\mathtt{q}]\oplus l} s[\mathtt{p}][\mathtt{o}: h] & \langle\text{QBraO}\rangle & s[\mathtt{p}][\mathtt{o}: h] \xrightarrow{s^o[\mathtt{p}][\mathtt{q}]\& l} s[\mathtt{p}][\mathtt{o}: [\mathtt{q}]l\cdot h] \\
\langle\text{QOpenS}\rangle & \dfrac{P \xrightarrow{s^o[\mathtt{p}][\mathtt{q}]!\langle s'[\mathtt{p}']\rangle} P'}{(\nu\, s[\mathtt{p}])P \xrightarrow{s^o[\mathtt{p}][\mathtt{q}]!(s'[\mathtt{p}'])} P'} & \langle\text{QOpenN}\rangle & \dfrac{P \xrightarrow{s^o[\mathtt{p}][\mathtt{q}]!\langle a\rangle} P'}{(\nu\, a)P \xrightarrow{s^o[\mathtt{p}][\mathtt{q}]!(a)} P'}
\end{array}$$

Rule ⟨QSendO⟩ defines that a non-empty output configuration queue $s^o[\mathtt{p}]$ can perform an output action $s^o[\mathtt{p}][\mathtt{q}]!\langle v\rangle$ and send (dequeue) a value v towards an observer role q. Dually, using rule ⟨QRcvO⟩, it can perform an input action $s^o[\mathtt{p}][\mathtt{q}]?\langle v\rangle$ to receive (enqueue) a value v from role p. Similarly rules ⟨QSelO⟩ and ⟨QBraA⟩, describe the select and branch interactions on labels. Rules ⟨QOpenS⟩ and ⟨QOpenN⟩ respectively extend scope opening using bound actions $s^o[\mathtt{p}][\mathtt{q}]!(s'[\mathtt{p}'])$ and $s^o[\mathtt{p}][\mathtt{q}]!(a)$ for output actions $s^o[\mathtt{p}][\mathtt{q}]!\langle s'[\mathtt{p}']\rangle$ and $s^o[\mathtt{p}][\mathtt{q}]!\langle a\rangle$ on session configurations respectively.

Localisation - Environment Transition Relation A localised process is a typed process that includes the corresponding output configurations for each free session role in the process.

Definition D.5 (Localisation).

1. *An environment Δ is* localised *if* $\forall s[\mathtt{p}] \in \mathtt{dom}(\Delta), s^o[\mathtt{p}] \in \mathtt{dom}(\Delta)$.
2. *P is* localised *if $\Gamma \vdash P \triangleright \Delta$ and Δ is localised.*

We define a labelled transition system over localised environments (Γ, Δ) which we use for the definition of a typed transition system over typed processes.

$$\begin{array}{llrcl}
\Gamma(a) = \langle G \rangle, s \text{ fresh} & \text{implies} & (\Gamma, \Delta) & \xrightarrow{a[A](s)}_o & (\Gamma, \Delta \cdot \{s[\mathtt{p}] : G\lceil \mathtt{p} \cdot s^o[\mathtt{p}] : \emptyset\}_{\mathtt{p} \in A}) \\
\Gamma(a) = \langle G \rangle, s \text{ fresh} & \text{implies} & (\Gamma, \Delta) & \xrightarrow{\overline{a}[A](s)}_o & (\Gamma, \Delta \cdot \{s[\mathtt{p}] : G\lceil \mathtt{p} \cdot s^o[\mathtt{p}] : \emptyset\}_{\mathtt{p} \in A}) \\
\Gamma \vdash v : U, s[\mathtt{q}] \notin \mathtt{dom}(\Delta) & \text{implies} & (\Gamma, \Delta \cdot s^o[\mathtt{p}] : M; [q]!\langle U \rangle) & \xrightarrow{s^o[\mathtt{p}][\mathtt{q}]!\langle v \rangle}_o & (\Gamma, \Delta \cdot s^o[\mathtt{p}] : M) \\
s^o[\mathtt{q}] \notin \mathtt{dom}(\Delta) & \text{implies} & (\Gamma, \Delta \cdot s[\mathtt{p}] : [\mathtt{q}]?(U); T) & \xrightarrow{s[\mathtt{p}][\mathtt{q}]?\langle v \rangle}_o & (\Gamma \cdot v : U, \Delta \cdot s[\mathtt{p}] : T) \\
s[\mathtt{q}] \notin \mathtt{dom}(\Delta) & \text{implies} & (\Gamma, \Delta \cdot s^o[\mathtt{p}] : M; [\mathtt{q}] \oplus l_k) & \xrightarrow{s^o[\mathtt{p}][\mathtt{q}] \oplus l_k}_o & (\Gamma, \Delta \cdot s^o[\mathtt{p}] : M) \\
s^o[\mathtt{q}] :\notin \mathtt{dom}(\Delta) & \text{implies} & (\Gamma, \Delta \cdot s[\mathtt{p}] : [\mathtt{q}]\&\{l_i : T_i\}) & \xrightarrow{s[\mathtt{p}][\mathtt{q}]\& l_k}_o & (\Gamma, \Delta \cdot s[\mathtt{p}] : T_k) \\
a \notin \mathtt{dom}(\Gamma), s[\mathtt{q}] \notin \mathtt{dom}(\Delta) & \text{implies} & (\Gamma, \Delta \cdot s^o[\mathtt{p}] : M; [\mathtt{q}]!\langle\langle G \rangle\rangle) & \xrightarrow{s^o[\mathtt{p}][\mathtt{q}]!(a)}_o & (\Gamma \cdot a : \langle G \rangle, \Delta \cdot s^o[\mathtt{p}] : M) \\
\Delta \longrightarrow \Delta' \vee \Delta = \Delta' & \text{implies} & (\Gamma, \Delta) & \xrightarrow{\tau} & (\Gamma, \Delta')
\end{array}$$

Actions $a[A](s)$ and $\overline{a}[A](s)$ extend a session environment to include type mapping of the session roles included in A. Types for each role derive from the typing of a in the shared environment Γ. Action $s^o[\mathtt{p}][\mathtt{q}]!\langle v \rangle$ happens on a message type. The rule checks for the type of v in the shared environment Γ to agree with the message type for $s^o[\mathtt{p}]$. Input action is $s[\mathtt{p}][\mathtt{q}]?\langle v \rangle$ observed on local types. After the action, the shared environment Γ is extended to include the received value. Similar with the send and receive actions are the select and branch actions respectively. Output actions on bounded shared names $s^o[\mathtt{p}][\mathtt{q}]!(a)$ check that a the shared name is not included in the shared environment Γ. Hidden actions τ follow the session environment reduction in Definition D.3. Finally we can always observe a τ action on any environment without changing its state. Environment LTS does not allow observable delegation actions. This is because a delegation action may result in a non-localised linear session environment Δ. However internal delegation can happen using the τ action. Internal delegation always results in a localised session environment. Finally we can always observe a τ action on any environment without changing its state.

D.7 Governed Behavioural Semantics for Output Asynchronous MSP

We use the global environment E definition from §5.1.

Recall that $\mathtt{out}(\lambda)$ and $\mathtt{inp}(\lambda)$ as $\mathtt{out}(s : \mathtt{p} \to \mathtt{q} : U) = \mathtt{out}(s : \mathtt{p} \to \mathtt{q} : l) = \mathtt{p}$ and as $\mathtt{inp}(s : \mathtt{p} \to \mathtt{q} : U) = \mathtt{inp}(s : \mathtt{p} \to \mathtt{q} : l) = \mathtt{q}$ and $\mathtt{p} \in \ell$ if $\mathtt{p} \in \mathtt{out}(\ell) \cup \mathtt{inp}(\ell)$.

We define the labelled reduction for global environments with respect to the definition of labelled reduction for global environments in §5.1

$$\frac{\{s : G\} \xrightarrow{\ell}_o \{s : G'\} \quad \mathtt{q} \notin \ell}{\{s : \mathtt{p} \to \mathtt{q} : \langle U \rangle.G\} \xrightarrow{\ell}_o \{s : \mathtt{p} \to \mathtt{q} : \langle U \rangle.G'\}} \qquad \frac{\{s : G_i\} \xrightarrow{\ell}_o \{s : G_i'\}\ i \in I\ \mathtt{q} \notin \ell}{\{s : \mathtt{p} \to \mathtt{q} : \{l_i : G_i\}_{i \in I}\} \xrightarrow{\ell}_o \{s : \mathtt{p} \to \mathtt{q} : \{l_i : G'\}_{i \in I}\}}$$

We change the rules for permutation to subsume both output asynchrony and role permutation with the two rules requiring that two sequenced actions can be permuted if the receiver of the fist action is not the sender or the receiver of the second action.

We use Definition 5.2 for global configurations, the global environment LTS in Figure 7 (description in § 5.1) and Definition 5.4 for configuration transition as defined using the global reduction for the output asynchronous MSP. We also prove that proposition 5.2 holds for the output asynchronous MSP.

We define global configuration barbs:

$$\begin{array}{ll}
(\Gamma, \Delta \cdot s^o[\mathtt{p}] : M; [\mathtt{q}]!\langle U\rangle, E) \downarrow_{s[\mathtt{p}][\mathtt{q}]} & \text{if } s[\mathtt{q}] \notin \mathtt{dom}(\Delta), E \overset{s:\mathtt{p}\to\mathtt{q}:U}{\longrightarrow} \\
(\Gamma, \Delta \cdot s^o[\mathtt{p}] : [\mathtt{q}] \oplus l_k; M, E) \downarrow_{s[\mathtt{p}][\mathtt{q}]} & \text{if } s[\mathtt{q}] \notin \mathtt{dom}(\Delta), E \overset{s:\mathtt{p}\to\mathtt{q}:l_k}{\longrightarrow}, k \in I \\
(\Gamma, \Delta, E) \downarrow_a & \text{always}
\end{array}$$

E Input Asynchronous MSP

We extend the synchronous MSP to define the semantics for input asynchronous Multiparty Session π calculus. Semantics are based in the system developed in [3].

E.1 Syntax

$$\begin{array}{rcll}
P & ::= & \vdots & \\
 & & s[\mathtt{p}][\mathtt{i} : \vec{h}] & (\text{ConfigurationI}) \\
 & & & \\
h & ::= & [\mathtt{p}](v) \mid [\mathtt{p}]l \mid [\mathtt{p}](s[q]) & (\text{Message})
\end{array}$$

We extend the syntax of synchronous MSP in Figure 2 with rule (ConfigurationI) to define the input configuration construct for session endpoints. Input configurations store messages $\vec{h}$ (defined in rule (Message)). Message h has the general form $[\mathtt{p}](v)$ that describe the passed value (i.e. $v, l, s[q]$) and the sender role $\mathtt{p}$. A process with no session names (i.e it does not include free session names, session endpoint configurations and session restriction), present in its syntax is called program.

Structural Congruence

$$\begin{array}{c}
\vdots \\
s[\mathtt{p}][\mathtt{i} : \vec{h} \cdot [\mathtt{q}](v) \cdot [\mathtt{q}'](v') \cdot \vec{h}] \equiv s[\mathtt{p}][\mathtt{i} : \vec{h} \cdot [\mathtt{q}'](v') \cdot [\mathtt{q}](v) \cdot \vec{h}] \qquad \mathtt{q} \neq \mathtt{q}' \\
(\nu\, s[\mathtt{p}])(s[\mathtt{p}][\mathtt{i} : \varepsilon]) \equiv \mathbf{0}
\end{array}$$

We extend structural congruence in Figure 11 to include message permutations inside input configurations. Messages are permuted upto structural congruence if their senders are different. The structural rule for message permutation is at the core of the asynchronous behaviour of output asynchronous MSP. The second structural congruence rule is used for garbage collection of inactive output configurations.

E.2 Operational Semantics

$$\begin{array}{rcll}
a[1](x).P_1 \mid \ldots \mid \overline{a}[n](x).P_n & \longrightarrow_i & (\nu\, s)(P_1\{s[1]/x\} \mid \ldots \mid P_n\{s[n]/x\} \mid s[\mathtt{p}][\mathtt{i} : \varepsilon] \mid \ldots \mid s[n][\mathtt{i} : \varepsilon]) & [\text{Link}] \\
s[\mathtt{p}][\mathtt{q}]!\langle v\rangle; P \mid s[\mathtt{q}][\mathtt{i} : \vec{h}] & \longrightarrow_i & P \mid s[\mathtt{q}][\mathtt{i} : [\mathtt{p}](v) \cdot \vec{h}] & [\text{Send}] \\
s[\mathtt{p}][\mathtt{q}]?(x); P \mid s[\mathtt{p}][\mathtt{i} : \vec{h} \cdot [\mathtt{q}](v)] & \longrightarrow_i & P\{v/x\} \mid s[\mathtt{p}][\mathtt{i} : \vec{h}] & [\text{Rcv}] \\
s[\mathtt{p}][\mathtt{q}]?(x); P \mid s[\mathtt{p}][\mathtt{i} : \vec{h} \cdot [\mathtt{q}](s'[\mathtt{p}'])] & \longrightarrow_i & P\{s'[\mathtt{p}']/x\} \mid s[\mathtt{p}][\mathtt{i} : \vec{h}] & [\text{Rcv-S}] \\
s[\mathtt{p}][\mathtt{q}] \oplus l; P \mid s[\mathtt{q}][\mathtt{i} : \vec{h}] & \longrightarrow_i & P \mid s[\mathtt{q}][\mathtt{i} : [\mathtt{p}]l \cdot \vec{h}] & [\text{Sel}] \\
s[\mathtt{p}][\mathtt{q}]\&\{l_i : P_i\}_{i \in I} \mid s[\mathtt{p}][\mathtt{i} : [\mathtt{q}]l_k \cdot \vec{h}] & \longrightarrow_i & P_k \mid s[\mathtt{p}][\mathtt{i} : \vec{h}] & [\text{Bra}]
\end{array}$$

The key difference from the operational semantics definition in Appendix D.2 is the use of input session configurations. Rule [Link] apart from session initiation it creates the corresponding input configurations $s[\mathtt{p}][\mathtt{i} : \varepsilon]$. Session communication is done on the input session configuration basis. A participant p receives values from the corresponding configuration $s^i[\mathtt{p}]$ (in contrast with output asynchronous MSP), while it sends a value to the corresponding receiving $s^i[\mathtt{q}]$ configurations. Rule [Send] describes an enqueue operation from role $s[\mathtt{p}]$ of a value v as a message $[\mathtt{p}](v)$ in session configuration $s[\mathtt{q}][\mathtt{i} : \vec{h}]$. Dually rule [Rcv] describes the dequeue operation and reception of a value v from role $s[\mathtt{q}]$ out of session configuration $s[\mathtt{p}][\mathtt{i} : \vec{h} \cdot [\mathtt{q}](va)]$. The reception happens on the substitution of value v on variable x on the continuation process of the receive action. There is a similar use for labels for rules [Sel] and [Bra]. The rest of the rules are standard π-calculus rules (similar to the synchronous MSP in §2).

E.3 Typing System for Programs

The typing System for Programs is identical to the typing system for programs defined for output asynchronous MSP in Appendix D.3

E.4 Typing System for Runtime Processes

A runtime process is a closed input asynchronous multiparty session π calculus term. We extend the typing system for programs to type output session configurations (cf. [13]). We start by extending the linear session environment Δ with the message type M:

$$\Delta \ ::= \ \Delta \cdot c[\mathtt{p}] : T \ \mid \ s^i[\mathtt{p}] : M \ \mid \ \emptyset$$

where

$$M \ ::= \ \emptyset \ \mid \ [\mathtt{q}]?(U);M \ \mid \ [\mathtt{q}]\&l;M$$

Δ is extended to include configuration endpoints $s^i[\mathtt{p}]$ mapped to the message type M. A message type is defined as a sequence of input message types $[\mathtt{q}]?(U)$ and branch message types $[\mathtt{q}]\&l$.

We define a permutation relation over message types.

Definition E.1 (Message Type Permutation). *Let* $\mathtt{p} \neq \mathtt{q}$. *We then define:*

$$\begin{gathered} M;[\mathtt{p}]?(U);[\mathtt{q}]?(U');M' \sim M;[\mathtt{q}]?(U');[\mathtt{p}]?(U);M' \\ M;[\mathtt{p}]\&l_i;[\mathtt{q}]\&l_j;M' \sim M;[\mathtt{q}]\&l_j;[\mathtt{p}]\&l_i;M' \textit{ if} \\ M;[\mathtt{p}]?(U);[\mathtt{q}]\&l;M' \sim M;[\mathtt{q}]\&l;[\mathtt{p}]?(U);M' \end{gathered}$$

with $\approx = \sim *$.

A message type sequence can be permuted on two message types if they have different senders. Message permutation is defined following the structural congruence rule for message permutation.

We define a concatenation operator $*$ between message types M and local types T. The result of the concatenation operator is a local type T. We use the concatenation operator to reconstruct a complete local type $T = M * T'$out of the local type $s[\mathtt{p}] : T'$ of a process and the message type $s^i[\mathtt{p}] : M$ of an input session configuration.

Definition E.2 (Message Type Concatenation). *Let* $\mathtt{p} \neq \mathtt{q}$. *We define message concatenation rules:*

$$\begin{aligned}
\emptyset * T &= T \\
[\mathtt{q}]?(U);M * [\mathtt{q}]?(U);T &= M * T \\
[\mathtt{q}]\&l_k;M * [\mathtt{q}]\&\{l_i : T_i\}_{i\in M} &= M * T_k \\
[\mathtt{p}]?(U');M * [\mathtt{q}]!\langle U\rangle;T &= [\mathtt{q}]!\langle U\rangle;([\mathtt{p}]?(U');M * T) \\
[\mathtt{p}]\&l;M * [\mathtt{q}]!\langle U\rangle;T &= [\mathtt{q}]!\langle U\rangle;([\mathtt{p}]\&l;M * T) \\
[\mathtt{p}]?(U);M * [\mathtt{q}]\oplus\{l_i : T_i\}_{i\in I} &= [\mathtt{q}]\oplus\{l_i : [\mathtt{p}]?(U);M * T_i\}_{i\in I} \\
[\mathtt{p}]\&l;M * [\mathtt{q}]\oplus\{l_i : T_i\}_{i\in I} &= [\mathtt{q}]\&\{l_i : [\mathtt{p}]\oplus l;M * T_i\}_{i\in I} \\
[\mathtt{p}]?(U');M * [\mathtt{q}]?(U);T &= [\mathtt{q}]?(U);([\mathtt{p}]?(U');M * T) \\
[\mathtt{p}]\&l;M * [\mathtt{q}]?(U);T &= [\mathtt{q}]?(U);([\mathtt{p}]\&l;M * T) \\
[\mathtt{p}]?(U);M * [\mathtt{q}]\&\{l_i : T_i\}_{i\in M} &= [\mathtt{q}]\&\{l_i : [\mathtt{p}]?(U);M * T_i\}_{i\in M} \\
[\mathtt{p}]\&l;M * [\mathtt{q}]\&\{l_i : T_i\}_{i\in M} &= [\mathtt{q}]\&\{l_i : [\mathtt{p}]\&l;M * T_i\}_{i\in M}
\end{aligned}$$

The message type prefix is used as a prefix for the resulting local type. The operation proceeds inductively on the sequence of the message type to reconstruct the session role local type.

We proceed with the definition of the runtime typing system:

$$\Gamma \vdash s[\mathtt{p}][\mathtt{i} : \varepsilon] \triangleright s^i[\mathtt{p}] : \emptyset \qquad \text{(QEmpty)}$$

$$\frac{\Gamma \vdash s[\mathtt{p}][\mathtt{i} : \vec{h}] \triangleright \Delta \cdot s^i[\mathtt{p}] : M}{\Gamma \vdash s[\mathtt{p}][\mathtt{i} : [\mathtt{q}](v) \cdot \vec{h}] \triangleright \Delta \cdot s^i[\mathtt{p}] : [\mathtt{q}]?(S);M} \qquad \text{(QRcv)}$$

$$\frac{\Gamma \vdash s[\mathtt{p}][\mathtt{i} : \vec{h}] \triangleright \Delta \cdot s^i[\mathtt{p}] : M}{\Gamma \vdash s[\mathtt{p}][\mathtt{i} : [\mathtt{q}]l \cdot \vec{h}] \triangleright \Delta \cdot s^i[\mathtt{p}] : [\mathtt{q}]\&l;M} \qquad \text{(QBra)}$$

$$\frac{\Gamma \vdash s[\mathtt{p}][\mathtt{i} : \vec{h}] \triangleright \Delta \cdot s^i[\mathtt{p}] : M}{\Gamma \vdash s[\mathtt{p}][\mathtt{i} : [\mathtt{q}](s'[\mathtt{q}']) \cdot \vec{h}] \triangleright \Delta \cdot s^i[\mathtt{p}] : [\mathtt{q}]?(T);M} \qquad \text{(QRcvS)}$$

$$\frac{\Gamma \vdash P \triangleright \Delta \cdot s^i[\mathtt{p}] : M \quad M \approx M'}{\Gamma \vdash P \triangleright \Delta \cdot s^i[\mathtt{p}] : M'} \qquad \text{(Equiv)}$$

$$\frac{\begin{array}{c}\Gamma \vdash P \triangleright \Delta \cdot s[1] : T_1 \cdot s^i[1] : M_1 \ldots s[n] : T_n \cdot s^i[n] : M_n \\ \mathtt{co}(\{s[1] : M_1 * T_1 \ldots s[n] : M_n * T_n\})\end{array}}{\Gamma \vdash (\nu\ s)P \triangleright \Delta} \qquad \text{(SRes)}$$

We ignore the (QCong) which is identical to the rule (QCong) for the output asynchronous message type system in Appendix E.4. Overall the message typing system for input asynchronous MSP is very similar the message typing system for output asynchronous MSP. The key difference is the sequence of input (resp. branch) message types rather than the use of output (resp. select) message types.

E.5 Type Soundness

We define the reduction semantics for local types. Since session environments represent the forthcoming communications, by reducing processes session environments can change. This can be formalised as in [3, 10] by introducing the notion of reduction of session environments, whose rules are:

Definition E.3 (Session Environment Reduction).

$$\begin{array}{rcl}
\{s[\mathtt{p}] : [\mathtt{q}]!\langle U\rangle;T \cdot s^i[\mathtt{q}] : M\} & \longrightarrow_i & \{s[\mathtt{p}] : T \cdot s^i[\mathtt{q}] : [\mathtt{p}]?(U);M\} \\
\{s[\mathtt{q}] : [\mathtt{p}]?(U);T \cdot s^i[\mathtt{q}] : M;[\mathtt{p}]?(U)\} & \longrightarrow_i & \{s[\mathtt{q}] : T \cdot s^i[\mathtt{q}] : M\} \\
\{s[\mathtt{p}] : [\mathtt{q}] \oplus \{l_i : T_i\}_{i\in I} \cdot s^i[\mathtt{q}] : M\} & \longrightarrow_i & \{s[\mathtt{p}] : T_k \cdot s^i[\mathtt{q}] : [\mathtt{p}] \oplus l_k;M\} \\
\{s[\mathtt{q}] : [\mathtt{p}]\&\{l_i : T_i\}_{i\in I} \cdot s^i[\mathtt{q}] : M;[\mathtt{p}] \oplus l_k\} & \longrightarrow_i & \{s[\mathtt{q}] : T_k \cdot s^i[\mathtt{q}] : M\}
\end{array}$$

We write $\twoheadrightarrow = \longrightarrow^*$.

Definition E.4. *Let session typing environment Δ. We define*

$$\begin{array}{rl}
*(\Delta) = & \{s[\mathtt{p}] : T \mid s[\mathtt{p}] : T \in \Delta, s^i[\mathtt{p}] \notin \mathtt{dom}(\Delta)\} \\
& \cup\ \{s^i[\mathtt{p}] : M \mid s^i[\mathtt{p}] : M \in \Delta, s[\mathtt{p}] \notin \mathtt{dom}(\Delta)\} \\
& \cup\ \{s[\mathtt{p}] : M^i * T \mid s[\mathtt{p}] : T, s^i[\mathtt{p}] : M^i \in \Delta, s^i[\mathtt{p}] \notin \mathtt{dom}(\Delta)\}
\end{array}$$

The $*(\Delta)$ operator reconstructs the local types for the session roles inside a linear session environment. It uses the $*$ operator to concatenate roles $s^i[\mathtt{p}]M$ with $s[\mathtt{p}]T$. The resulting linear session environment is used for coherency checking in the subject reduction theorem.

Lemma E.1. *If* $\mathtt{co}(\Delta \cdot s[\mathtt{p}] : [\mathtt{q}]!\langle U\rangle;T)$ *and* $s[\mathtt{q}] \notin \mathtt{dom}(\Delta)$ *then* $\mathtt{co}(\Delta \cdot s[\mathtt{p}] : T)$.

Proof. Let $\mathtt{q}' \neq \mathtt{q}$ and $s[\mathtt{q}'] : T_{\mathtt{q}'} \in \Delta$. Then $[\mathtt{q}]!\langle U\rangle;T\lceil \mathtt{q}' = T\lceil \mathtt{q}'$. Since $s[\mathtt{q}] \notin \mathtt{dom}(\Delta)$ then for all $s[\mathtt{q}'] \in \mathtt{dom}(\Delta)$ $[\mathtt{q}]!\langle U\rangle;T\lceil \mathtt{q}' = T\lceil \mathtt{q}'$, so $\mathtt{co}(\Delta \cdot s[\mathtt{p}] : T)$.

Theorem E.1 (Subject Reduction). *Let* $\Gamma \vdash P \triangleright \Delta$ *with* $\mathtt{co}(*(\Delta))$ *and if* $P \twoheadrightarrow P'$ *then* $\Gamma \vdash P' \triangleright \Delta'$ *with* $\Delta \twoheadrightarrow \Delta'$ *and* $\mathtt{co}(*(\Delta'))$.

E.6 Behavioural Semantics for Input Asynchronous MSP

We extend the label definition ℓ in §4.1 to include action labels on input configurations:

$$\begin{array}{rl}
\ell = & \vdots \\
& \mid\ s^i[\mathtt{p}][\mathtt{q}]!\langle v\rangle \ \mid\ s^i[\mathtt{p}][\mathtt{q}]!(v) \ \mid\ s^i[\mathtt{p}][\mathtt{q}]?\langle v\rangle \ \mid\ s^i[\mathtt{p}][\mathtt{q}] \oplus l \ \mid\ s^i[\mathtt{p}][\mathtt{q}]\&l
\end{array}$$

Labels $s^i[\mathtt{p}][\mathtt{q}]!\langle v\rangle$ and $s^i[\mathtt{p}][\mathtt{q}]!(v)$ denote the output of value v (output of bound value v) from session configuration $s^i[\mathtt{p}]$ to participant $\mathtt{q}$. Dually action $s^i[\mathtt{p}][\mathtt{q}]?\langle v\rangle$ denotes the reception of value v by session configurations $s^i[\mathtt{p}]$ sent by participant $\mathtt{q}$. Actions $s^i[\mathtt{p}][\mathtt{q}] \oplus l$ and $s^i[\mathtt{p}][\mathtt{q}]\&l$ respectively describe the send (select) and receive (branch) of label l from participant $\mathtt{p}$ to participant $\mathtt{q}$.

We use the definitions for *role set* A and $\max(A)$ from §4.1.

We define the duality relation $\asymp$ between labels:

$$\begin{array}{rcll}
a[A](s) & \asymp & \overline{a}[A'](s) & \\
s[\mathtt{p}][\mathtt{q}]!\langle v\rangle & \asymp_i & s^i[\mathtt{q}][\mathtt{p}]?\langle v\rangle & s^i[\mathtt{p}][\mathtt{q}]!\langle v\rangle \asymp_i s[\mathtt{p}][\mathtt{q}]?\langle v\rangle \\
s[\mathtt{p}][\mathtt{q}]!(v) & \asymp_i & s^i[\mathtt{q}][\mathtt{p}]?\langle v\rangle & s^i[\mathtt{p}][\mathtt{q}]!(v) \asymp_i s[\mathtt{p}][\mathtt{q}]?\langle v\rangle \\
s[\mathtt{p}][\mathtt{q}] \oplus l & \asymp_i & s^i[\mathtt{q}][\mathtt{p}]\&l & s^i[\mathtt{p}][\mathtt{q}] \oplus l \asymp_i s[\mathtt{p}][\mathtt{q}]\&l
\end{array}$$

Accept $a[A](s)$ and request $\overline{a}[A'](s)$ labels are defined as dual. Process output actions $s[\mathtt{p}][\mathtt{q}]!\langle v\rangle$ interact with the corresponding session configuration input actions

$s^i[\mathtt{q}][\mathtt{p}]?\langle v\rangle$. Similarly for process bound output. Process input actions $s[\mathtt{q}][\mathtt{p}]?\langle v\rangle$ interact with the corresponding session configuration output action ($s^i[\mathtt{q}][\mathtt{p}]!\langle v\rangle$) of the receiver participant q. Similarly when the session configuration output action is bound. Select and branching label duality follows the value send and receive semantics.

The labelled transition system is extended from the synchronous MSP LTS in Figure 5 (description in §4.1) to define actions on output session configurations.

$$\begin{array}{llll}
\langle\text{QSendI}\rangle & s[\mathtt{p}][\mathtt{i}:h\cdot[\mathtt{q}](v)] \xrightarrow{s^i[\mathtt{p}][\mathtt{q}]!\langle v\rangle} s[\mathtt{p}][\mathtt{i}:h] & \langle\text{QRcvI}\rangle & s[\mathtt{p}][\mathtt{i}:h] \xrightarrow{s^i[\mathtt{p}][\mathtt{q}]?\langle v\rangle} s[\mathtt{p}][\mathtt{i}:[\mathtt{q}](v)\cdot h] \\
\langle\text{QSelI}\rangle & s[\mathtt{p}][\mathtt{i}:h\cdot[\mathtt{q}]l] \xrightarrow{s^i[\mathtt{p}][\mathtt{q}]\oplus l} s[\mathtt{p}][\mathtt{i}:h] & \langle\text{QBraI}\rangle & s[\mathtt{p}][\mathtt{i}:h] \xrightarrow{s^i[\mathtt{p}][\mathtt{q}]\& l} s[\mathtt{p}][\mathtt{i}:[\mathtt{q}]l\cdot h] \\
\langle\text{QOpenS}\rangle & \dfrac{P \xrightarrow{s^i[\mathtt{p}][\mathtt{q}]!\langle s'[\mathtt{p}']\rangle} P'}{(\nu\, s[\mathtt{p}])P \xrightarrow{s^i[\mathtt{p}][\mathtt{q}]!(s'[\mathtt{p}'])} P'} & \langle\text{QOpenN}\rangle & \dfrac{P \xrightarrow{s^i[\mathtt{p}][\mathtt{q}]!\langle a\rangle} P'}{(\nu\, a)P \xrightarrow{s^i[\mathtt{p}][\mathtt{q}]!(a)} P'}
\end{array}$$

Rule $\langle\text{QSendO}\rangle$ defines that a non-empty input configuration queue $s^i[\mathtt{p}]$ can perform an output action $s^i[\mathtt{p}][\mathtt{q}]!\langle v\rangle$ and send (dequeue) a value v towards an observer role q. Dually, using rule $\langle\text{QRcvO}\rangle$, it can perform an input action $s^i[\mathtt{p}][\mathtt{q}]?\langle v\rangle$ to receive (enqueue) a value v from role p. Similarly rules $\langle\text{QSelO}\rangle$ and $\langle\text{QBraA}\rangle$, describe the select and branch interactions on labels. Rules $\langle\text{QOpenS}\rangle$ and $\langle\text{QOpenN}\rangle$ respectively extend scope opening using bound actions $s^i[\mathtt{p}][\mathtt{q}]!(s'[\mathtt{p}'])$ and $s^o[\mathtt{p}][\mathtt{q}]!(a)$ for output actions $s^i[\mathtt{p}][\mathtt{q}]!\langle s'[\mathtt{p}']\rangle$ and $s^i[\mathtt{p}][\mathtt{q}]!\langle a\rangle$ on session configurations respectively.

We define a context for global types:

Definition E.5 (Global Type Context).

$$\begin{array}{rcl}
\mathscr{T} & ::= & - \mid [\mathtt{p}]!\langle U\rangle;\mathscr{T} \mid [\mathtt{p}]?(U);\mathscr{T} \mid \\
& & [\mathtt{p}]\oplus\{l_i:\mathscr{T}_i\}_{i\in I} \mid [\mathtt{p}]\&\{l_i:\mathscr{T}_i\}_{i\in I} \mid \\
& & \mu\mathsf{X}.\mathscr{T}
\end{array}$$

Localisation - Environment Transition Relation A localised process is a typed process that includes the corresponding output configurations for each free session role in the process.

Definition E.6 (Localisation).

1. *An environment Δ is* localised *if* $\forall s[\mathtt{p}] \in \mathtt{dom}(\Delta), s^i[\mathtt{p}] \in \mathtt{dom}(\Delta)$.
2. *P is* localised *if $\Gamma \vdash P \triangleright \Delta$ and Δ is localised.*

We define a labelled transition system over localised environments (Γ,Δ) which we use for the definition of a typed transition system over typed processes.

$$
\begin{array}{rcl}
\Gamma(a) = \langle G \rangle, s \text{ fresh implies} & (\Gamma, \Delta) \xrightarrow{\overline{a}[A](s)}_i & (\Gamma, \Delta \cdot \{s[\mathtt{p}] : G\lceil \mathtt{p} \cdot s^i[\mathtt{p}] : \emptyset\}_{\mathtt{p} \in A}) \\
\Gamma(a) = \langle G \rangle, s \text{ fresh implies} & (\Gamma, \Delta) \xrightarrow{a[A](s)}_i & (\Gamma, \Delta \cdot \{s[\mathtt{p}] : G\lceil \mathtt{p} \cdot s^i[\mathtt{p}] : \emptyset\}_{\mathtt{p} \in A}) \\
\Gamma \vdash v : U, s^i[\mathtt{q}] \notin \mathtt{dom}(\Delta) \text{ implies} & (\Gamma, \Delta \cdot s[\mathtt{p}] : [q]!\langle U \rangle; T) \xrightarrow{s[\mathtt{p}][\mathtt{q}]!\langle v \rangle}_i & (\Gamma, \Delta \cdot s[\mathtt{p}] : T) \\
a \notin \mathtt{dom}(\Gamma), s^i[\mathtt{q}] \notin \mathtt{dom}(\Delta) \text{ implies} & (\Gamma, \Delta \cdot s[\mathtt{p}] : [\mathtt{q}]!\langle U \rangle; T) \xrightarrow{s[\mathtt{p}][\mathtt{q}]!(a)}_i & (\Gamma \cdot a : U, \Delta \cdot s[\mathtt{p}] : T)
\end{array}
$$

$$
\frac{T \ * \ M = \mathscr{T}[[\mathtt{q}]?(S); T'] \quad \mathscr{T} \text{ not contain prefix on role } \mathtt{q} \quad s[\mathtt{q}] \notin \mathtt{dom}(\Delta)}{(\Gamma, \Delta \cdot s[\mathtt{p}] : T \cdot s^i[\mathtt{p}] : M) \xrightarrow{s^i[\mathtt{p}][\mathtt{q}]?\langle v \rangle}_i (\Gamma \cdot v : U, \Delta \cdot s[\mathtt{p}] : T \cdot s^i[\mathtt{p}] : [\mathtt{q}]?(S); M)}
$$

$$
s^i[\mathtt{q}] \notin \mathtt{dom}(\Delta) \text{ implies } (\Gamma, \Delta \cdot s[\mathtt{p}] : [\mathtt{q}]!\langle l_i : T_i \rangle;) \xrightarrow{s[\mathtt{p}][\mathtt{q}] \oplus l_k}_i (\Gamma \cdot v : U, \Delta \cdot s[\mathtt{p}] : T_k)
$$

$$
\frac{T \ * \ M = \mathscr{T}[[\mathtt{q}]\&\{l_i : T_i\}] \quad \mathscr{T} \text{ not contain prefix on role } \mathtt{q} \quad s[\mathtt{q}] \notin \mathtt{dom}(\Delta)}{(\Gamma, \Delta \cdot s[\mathtt{p}] : T \cdot s^i[\mathtt{p}] : M) \xrightarrow{s^i[\mathtt{p}][\mathtt{q}]\& l_k}_i (\Gamma, \Delta \cdot s[\mathtt{p}] : T \cdot s^i[\mathtt{p}] : [\mathtt{q}] \oplus l_k; M)}
$$

$$
\Delta \longrightarrow \Delta' \vee \Delta = \Delta' \text{ implies} \qquad (\Gamma, \Delta) \quad \xrightarrow{\tau} \quad (\Gamma, \Delta')
$$

Actions $a[A](s)$ and $\overline{a}[A](s)$ extend a session environment to include type mapping of the session roles included in A. Types for each role derive from the typing of a in the shared environment Γ. Action $s[\mathtt{p}][\mathtt{q}]!\langle v \rangle$ happens on a local type. The rule checks for the type of v in the shared environment Γ to agree with the local type for $s[\mathtt{p}]$. Input action $s^i[\mathtt{p}][\mathtt{q}]?\langle v \rangle$ is observed on message types. We expect the sending local type not to be present in the session typing. After the action, the shared environment Γ is extended to include the received value. Similar with the send and receive actions are the select and branch actions respectively. Output actions on bounded shared names $s[\mathtt{p}][\mathtt{q}]!(a)$ check that a the shared name is not included in the shared environment Γ. Hidden actions τ are defined with respect to to the session environment reduction in Definition E.3. Environment LTS does not allow observable delegation actions. This is because delegation a delegation action may result in a non-localised linear session environment Δ. However internal delegation can happen using the τ action. Internal delegation always results in a localised session environment. Finally we can always observe a τ action on any environment without changing its state.

E.7 Governed Behavioural Semantics for Input Asynchronous MSP

We use the global environment E definition from §5.1.

Recall that $\mathtt{out}(\lambda)$ and $\mathtt{inp}(\lambda)$ as $\mathtt{out}(s : \mathtt{p} \to \mathtt{q} : U) = \mathtt{out}(s : \mathtt{p} \to \mathtt{q} : l) = \mathtt{p}$ and as $\mathtt{inp}(s : \mathtt{p} \to \mathtt{q} : U) = \mathtt{inp}(s : \mathtt{p} \to \mathtt{q} : l) = \mathtt{q}$ and $\mathtt{p} \in \ell$ if $\mathtt{p} \in \mathtt{out}(\ell) \cup \mathtt{inp}(\ell)$.

We define the labelled reduction for global environments with respect to the definition of labelled reduction for global environments in §5.1

$$
\frac{\{s : G\} \xrightarrow{\ell}_i \{s : G'\} \quad \mathtt{p}, \mathtt{q} \notin \mathtt{out}(\ell)}{\{s : \mathtt{p} \to \mathtt{q} : \langle U \rangle.G\} \xrightarrow{\ell}_o \{s : \mathtt{p} \to \mathtt{q} : \langle U \rangle.G'\}}
\qquad
\frac{\{s : G_i\} \xrightarrow{\ell}_i \{s : G_i'\} \ i \in I \ \mathtt{p}, \mathtt{q} \notin \mathtt{out}(\ell)}{\{s : \mathtt{p} \to \mathtt{q} : \{l_i : G_i\}_{i \in I}\} \xrightarrow{\ell}_o \{s : \mathtt{p} \to \mathtt{q} : \{l_i : G_i'\}_{i \in I}\}}
$$

We change the rules for permutation to subsume both input asynchrony and role permutation with the two rules requiring that two sequenced actions can be permuted if the sender or the receiver of the first action is different than the sender of the second action.

We use Definition 5.2 for global configurations, the global environment LTS in Figure 7 (description in § 5.1) and Definition 5.4 for configuration transition as defined using the global reduction for the input asynchronous MSP. We also prove that proposition 5.2 holds for the input asynchronous MSP.

We define global configuration barbs:

$$\begin{array}{rl}
(\Gamma, \Delta \cdot s[\mathtt{p}] : [\mathtt{q}]!\langle U\rangle; T, E) \downarrow_{s[\mathtt{p}][\mathtt{q}]} & \text{if } s^i[\mathtt{q}] \notin \mathtt{dom}(\Delta), E \overset{s:\mathtt{p}\to\mathtt{q}:U}{\longrightarrow} \\
(\Gamma, \Delta \cdot s[\mathtt{p}] : [\mathtt{q}] \oplus \{l_i : T_i\}_{i\in I}, E) \downarrow_{s[\mathtt{p}][\mathtt{q}]} & \text{if } s^i[\mathtt{q}] \notin \mathtt{dom}(\Delta), E \overset{s:\mathtt{p}\to\mathtt{q}:l_k}{\longrightarrow}, k \in I \\
(\Gamma, \Delta, E) \downarrow_a & \text{always}
\end{array}$$

F Input/Output Asynchronous MSP

Input/Output Asynchronous Multiparty Session π-calculus is defined as a mixture of the semantics for output asynchronous MSP (§D) and input asynchronous MSP (§E). Semantics are based in the system developed in [3].

F.1 Syntax

$$\begin{array}{rll}
P ::= & \vdots & \\
& s[\mathtt{p}][\mathtt{o} : \vec{h}] & (\text{ConfigurationO}) \\
& s[\mathtt{p}][\mathtt{i} : \vec{h}] & (\text{ConfigurationI})
\end{array}$$

The syntax is extended with both the output session configuration $s[\mathtt{p}][\mathtt{o} : \vec{h}]$ (description in §D.1) and the input session configuration $s[\mathtt{p}][\mathtt{i} : \vec{h}]$ (description §E.1). We usually write $s[\mathtt{p}][\mathtt{i} : \vec{h}, \mathtt{o} : \vec{h'}]$ for the concatenation of the process $s[\mathtt{p}][\mathtt{i} : \vec{h}] \mid s[\mathtt{p}][\mathtt{o} : \vec{h'}]$. A process with no session names (i.e it does not include free session names, session endpoint configurations and session restriction), present in its syntax is called program.

Structural Congruence

$$\begin{array}{cr}
\vdots & \\
s[\mathtt{p}][\mathtt{o} : \vec{h} \cdot [\mathtt{q}](v) \cdot [\mathtt{q}'](v') \cdot \vec{h}] \equiv s[\mathtt{p}][\mathtt{o} : \vec{h} \cdot [\mathtt{q}'](v') \cdot [\mathtt{q}](v) \cdot \vec{h}] & \mathtt{q} \neq \mathtt{q}' \\
s[\mathtt{p}][\mathtt{i} : \vec{h} \cdot [\mathtt{q}](v) \cdot [\mathtt{q}'](v') \cdot \vec{h}] \equiv s[\mathtt{p}][\mathtt{i} : \vec{h} \cdot [\mathtt{q}'](v') \cdot [\mathtt{q}](v) \cdot \vec{h}] & \mathtt{q} \neq \mathtt{q}' \\
(\nu\, s[\mathtt{p}])(s[\mathtt{p}][\mathtt{o} : \varepsilon]) \equiv \mathbf{0} & \\
(\nu\, s[\mathtt{p}])(s[\mathtt{p}][\mathtt{i} : \varepsilon]) \equiv \mathbf{0} &
\end{array}$$

Structural congruence extends the structural congruence in figure 11 to include the union of the rules for output asynchronous MSP (description in §D.1) and input asynchronous MSP (description in §E.1).

F.2 Operational Semantics

$$a[1](x).P_1 \mid \ldots \mid \overline{a}[n](x).P_n \longrightarrow_{io} (\nu\, s)(P_1\{s[1]/x\} \mid \ldots \mid P_n\{s[n]/x\} \mid s[\mathtt{p}][\mathtt{i}:\varepsilon, \mathtt{o}:\varepsilon] \mid \ldots \mid s[n][\mathtt{i}:\varepsilon, \mathtt{o}:\varepsilon]) \quad \text{[Link]}$$

$$s[\mathtt{p}][\mathtt{q}]!\langle v\rangle;P \mid s[\mathtt{p}][\mathtt{o}:\vec{h}] \longrightarrow_{io} P \mid s[\mathtt{p}][\mathtt{o}:[\mathtt{q}](v)\cdot\vec{h}] \quad \text{[Send]}$$

$$s[\mathtt{p}][\mathtt{q}]?(x);P \mid s[\mathtt{p}][\mathtt{i}:\vec{h}\cdot[\mathtt{q}](v)] \longrightarrow_{io} P\{v/x\} \mid s[\mathtt{p}][\mathtt{i}:\vec{h}] \quad \text{[Rcv]}$$

$$s[\mathtt{p}][\mathtt{q}]?(x);P \mid s[\mathtt{p}][\mathtt{i}:\vec{h}\cdot[\mathtt{q}](s'[\mathtt{p}'])] \longrightarrow_{io} P\{s'[\mathtt{p}']/x\} \mid s[\mathtt{p}][\mathtt{i}:\vec{h}] \quad \text{[Rcv-S]}$$

$$s[\mathtt{p}][\mathtt{q}]\oplus l;P \mid s[\mathtt{p}][\mathtt{o}:\vec{h}] \longrightarrow_{io} P \mid s[\mathtt{p}][\mathtt{o}:[\mathtt{q}]l\cdot\vec{h}] \quad \text{[Sel]}$$

$$s[\mathtt{p}][\mathtt{q}]\&\{l_i:P_i\}_{i\in I} \mid s[\mathtt{p}][\mathtt{i}:\vec{h}\cdot[\mathtt{q}]l_k] \longrightarrow_{io} P_k \mid s[\mathtt{p}][\mathtt{i}:\vec{h}] \quad \text{[Bra]}$$

$$s[p][\mathtt{o}:[q](v)\cdot\vec{h}] \mid s[q][\mathtt{i}:\vec{h}'] \longrightarrow_{io} s[p][\mathtt{o}:\vec{h}] \mid s[q][\mathtt{i}:[p](v)\cdot\vec{h}'] \quad \text{[Comm]}$$

F.3 Typing System for Programs

The session type system programs is identical with the session type system for output asynchronous MSP (§D.3) and input asynchronous MSP (§E.3).

F.4 Typing System For Runtime processes

A runtime process is a closed input/output asynchronous multiparty session π calculus term. We extend the typing system for programs to type output session configurations (cf. [13]). The extension is a combination of the runtime typing definitions for output asynchronous MSP (§D.4) and input asynchronous MSP (§E.4).

We start by extending the linear session environment Δ with the message type M:

$$\Delta \ ::= \ \Delta\cdot c[\mathtt{p}]:T \ \mid \ s^o[\mathtt{p}]:M \ \mid \ s^i[\mathtt{p}]:M \ \mid \ \emptyset$$

where

$$\begin{aligned} M &::= O \mid I \\ O &::= \emptyset \mid [\mathtt{q}]!\langle U\rangle;O \mid [\mathtt{q}]\oplus l;O \\ I &::= \emptyset \mid [\mathtt{q}]?(U);I \mid [\mathtt{q}]\&l;I \end{aligned}$$

Δ is extended to include configuration endpoints $s^o[\mathtt{p}], s^i[\mathtt{q}]$ types, which is notation $s^o[\mathtt{p}]$ mapped to the message type M. A message type is defined as a sequence of output message types $[\mathtt{q}]!\langle U\rangle$ and select message types $[\mathtt{q}]\oplus l$ (§D.4). Or a sequence of input message type $[\mathtt{q}]?(U)$ and branch message types $[\mathtt{q}]\&l$ (§E.4).

We define a permutation relation over message types.

Definition F.1 (Message Type Permutation). *Let $\sim_o$ for the Definition D.1 and $\sim_i$ for the Definition E.1. We then define $\sim = \sim_o \cup \sim_i$. We write $\approx = \sim^*$*

An input/output message permutation is the union of output message permutation and input message permutation. Message permutation is defined following the structural congruence rule for message permutation.

We define a concatenation operator $*$ between message types M and local types T. The result of the concatenation operator is a local type T. We use the concatenation operator to reconstruct a complete local type $T = M * T'$out of the local type $s[\mathtt{p}]:T'$ of a process and the message type $s^o[\mathtt{p}]:M$ of an output session configuration.

Definition F.2 (Message Type Concatanation).

$$\begin{array}{rcl} \emptyset * T = T & & \\ [\mathtt{q}]!\langle U\rangle;O * T = [\mathtt{q}]!\langle U\rangle;(O * T) & & [\mathtt{q}]\oplus l;O * T = [\mathtt{q}]\oplus\{l_i : O * T\}_{i\in I} \\ [\mathtt{q}]?(U);I * [\mathtt{q}]?(U);T = I * T & & [\mathtt{q}]\&l_k;I * [\mathtt{q}]\&\{l_i : T_i\}_{i\in I} = I * T \end{array}$$

$$\begin{array}{rcll} I * [\mathtt{q}]!\langle U\rangle;T &=& [\mathtt{q}]!\langle U\rangle;(I * T) & I \neq [\mathtt{q}]?(U);I', I \neq [\mathtt{q}]\&l;I' \\ I * [\mathtt{q}]\oplus\{l_i : T_i\}_{i\in I} &=& [\mathtt{q}]\oplus\{l_i : I * T_i\}_{i\in I} & I \neq [\mathtt{q}]?(U);I', I \neq [\mathtt{q}]\&l;I' \\ I * [\mathtt{q}]?(U);T &=& [\mathtt{q}]?(U);(I * T) & I \neq [\mathtt{q}]?(U');I' \\ I * [\mathtt{q}]\&\{l_i : T_i\}_{i\in I} &=& [\mathtt{q}]\&\{l_i : I * T_i\}_{i\in I} & I \neq [\mathtt{q}]\&l;I' \end{array}$$

We proceed with the definition of the runtime typing system. The definition is the union of the runtime typing rules for the output asynchronous MSP (§D.4) and input asynchronous MSP (§E.4) , with the exception of the (SRes) which is redefined as:

$$\frac{\begin{array}{c}\Gamma \vdash P \triangleright \Delta \cdot s[1] : T_1 \cdot s^o[1] : M_1^o \cdot s^i[1] : M_1^i \ldots s[n] : T_n \cdot s^i[n] : M_n^o \cdot s^i[1] : M_n^i \\ \mathtt{co}(\{s[1] : M_1^o * (M_1^i * T_1) \ldots s[n] : M_n^o * (M_n^i * T_n)\})\end{array}}{\Gamma \vdash (\nu\, s)P \triangleright \Delta}\ [\mathrm{SRes}]$$

We define the reduction semantics for local types. Since session environments represent the forthcoming communications, by reducing processes session environments can change. This can be formalised as in [3, 10] by introducing the notion of reduction of session environments, whose rules are:

Definition F.3 (Session Environment Reduction).

$$\begin{array}{rcl} \{s[\mathtt{p}] : [\mathtt{q}]!\langle U\rangle;T \cdot s^o[\mathtt{p}] : M\} &\longrightarrow_{io}& \{s[\mathtt{p}] : T \cdot s^o[\mathtt{p}] : [\mathtt{q}]!\langle U\rangle;M\} \\ \{s[\mathtt{p}] : [\mathtt{q}]?(U);T \cdot s^i[\mathtt{p}] : M;[\mathtt{q}]?(U)\} &\longrightarrow_{io}& \{s[\mathtt{p}] : T \cdot s^i[\mathtt{p}] : M\} \\ \{s[\mathtt{p}] : [\mathtt{q}]\oplus\{l_i : T_i\}_{i\in I} \cdot s^o[\mathtt{p}] : M\} &\longrightarrow_{io}& \{s[\mathtt{p}] : T_k \cdot s^i[\mathtt{p}] : [\mathtt{q}]\oplus l;M\} \\ \{s[\mathtt{p}] : [\mathtt{q}]\&\{l_i : T_i\}_{i\in I} \cdot s^i[\mathtt{p}] : M;[\mathtt{q}]\&l_k\} &\longrightarrow_{io}& \{s[\mathtt{p}] : T_k \cdot s^i[\mathtt{p}] : M\} \\ \Delta \cup \Delta' &\longrightarrow& \Delta \cup \Delta'' \quad \textit{if} \quad \Delta' \longrightarrow \Delta'' \end{array}$$

We write $\twoheadrightarrow = \longrightarrow^*$.

Definition F.4. *Let session typing environment Δ. We define*

$$\begin{array}{rl} *(\Delta) = & \{s[\mathtt{p}] : T \mid s[\mathtt{p}] : T \in \Delta, s^i[\mathtt{p}], s^o[\mathtt{p}] \notin \mathtt{dom}(\Delta)\} \\ & \cup\ \{s^m[\mathtt{p}] : M \in \Delta, s[\mathtt{p}] \notin \mathtt{dom}(\Delta)\} \\ & \cup\ \{s[\mathtt{p}] : M^o * T \mid s[\mathtt{p}] : T, s^o[\mathtt{p}] : M^o \in \Delta, s^i[\mathtt{p}] \notin \mathtt{dom}(\Delta)\} \\ & \cup\ \{s[\mathtt{p}] : M^i * T \mid s[\mathtt{p}] : T, s^i[\mathtt{p}] : M^i \in \Delta, s^o[\mathtt{p}] \notin \mathtt{dom}(\Delta)\} \\ & \cup\ \{s[\mathtt{p}] : M^o * M^i * T \mid s[\mathtt{p}] : T, s^i[\mathtt{p}] : M^i, s^o[\mathtt{p}] : M^o \in \Delta\} \end{array}$$

The $*(\Delta)$ operator reconstructs the local types for the session roles inside a linear session environment. It uses the $*$ operator to concatenate roles $s^o[\mathtt{p}]M_o, s^i[\mathtt{p}]M_i$ with $s[\mathtt{p}]T$. The resulting linear session environment is used for coherency checking in the subject reduction theorem.

Theorem F.1 (Subject Reduction). *Let $\Gamma \vdash P \triangleright \Delta$ with $\mathtt{co}(*(\Delta))$ and if $P \twoheadrightarrow P'$ then $\Gamma \vdash P' \triangleright \Delta'$ with $\Delta \twoheadrightarrow \Delta'$ and $\mathtt{co}(*(\Delta'))$.*

F.5 Behavioural Semantics for Input/Output Asynchronous MSP

We use the union of the semantic theories for output asynchronous MSP (in Appendix D.6) and input asynchronous MSP (in Appendix E.6) to define the behavioural theory for input/output asynchronous MSP.

We extend the label definition ℓ in §4.1 with the union of the labels on input and output configurations. to include action labels on input configurations.

We use the definitions for *role set* A and $\max(A)$ from §4.1.

We define the duality relation $\asymp$ between labels:

$$\begin{array}{rcl} a[A](s) & \asymp & \overline{a}[A'](s) \\ s[\mathtt{p}][\mathtt{q}]!\langle v\rangle \asymp_{io} s^o[\mathtt{p}][\mathtt{q}]?\langle v\rangle & & s^i[\mathtt{p}][\mathtt{q}]!\langle v\rangle \asymp_{io} s[\mathtt{p}][\mathtt{q}]?\langle v\rangle \\ s[\mathtt{p}][\mathtt{q}]!(v) \asymp_{o} s^o[\mathtt{p}][\mathtt{q}]?\langle v\rangle & & s^i[\mathtt{p}][\mathtt{q}]!(v) \asymp_{io} s[\mathtt{p}][\mathtt{q}]?\langle v\rangle \\ s[\mathtt{p}][\mathtt{q}] \oplus l \asymp_{o} s^o[\mathtt{p}][\mathtt{q}]\& l & & s^i[\mathtt{p}][\mathtt{q}] \oplus l \asymp_{io} s[\mathtt{p}][\mathtt{q}]\& l \\ s^o[\mathtt{p}][\mathtt{q}]!\langle v\rangle \asymp_{io} s^i[\mathtt{q}][\mathtt{p}]?\langle v\rangle & & s^o[\mathtt{p}][\mathtt{q}] \oplus l \asymp_{io} s^i[\mathtt{q}][\mathtt{p}]\& l \end{array}$$

The duality relation on labels is a combination of the duality between process output labels and output configuration input labels from the output asynchronous MSP definition and the duality between process input labels and input configuration output labels from the input asynchronous MSP definition. We extend with the duality between input and output configuration labels.

The labelled transition system is extended from the synchronous MSP LTS in Figure 5 (description in §4.1) with the union of the labelled transition semantics from the input and output asynchronous MSP LTSs (Appendix D.6, E.6).

We use the Definition E.5 for global type context.

Localisation - Environment Transition Relation A localised process is a typed process that includes the corresponding output configurations for each free session role in the process.

Definition F.5 (Localisation).

1. *An environment Δ is* localised *if* $\forall s[\mathtt{p}] \in \mathtt{dom}(\Delta), s^i[\mathtt{p}], s^o[\mathtt{p}] \in \mathtt{dom}(\Delta)$.
2. *P is* localised *if $\Gamma \vdash P \triangleright \Delta$ and Δ is localised.*

We define a labelled transition system over localised environments (Γ, Δ) which we use for the definition of a typed transition system over typed processes.

$$\begin{array}{rl} \Gamma(a) = \langle G\rangle, s \text{ fresh implies} & (\Gamma,\Delta) \xrightarrow{a[A](s)}_{io} (\Gamma, \Delta\cdot\{s[\mathtt{p}] : G\lceil \mathtt{p} \cdot s^o[\mathtt{p}] : \emptyset \cdot s^i[\mathtt{p}] : \emptyset\}_{\mathtt{p}\in A}) \\ \Gamma(a) = \langle G\rangle, s \text{ fresh implies} & (\Gamma,\Delta) \xrightarrow{\overline{a}[A](s)}_{io} (\Gamma, \Delta\cdot\{s[\mathtt{p}] : G\lceil \mathtt{p} \cdot s^o[\mathtt{p}] : \emptyset \cdot s^i[\mathtt{p}] : \emptyset\}_{\mathtt{p}\in A}) \\ \Gamma \vdash v : U, s^i[\mathtt{q}] \notin \mathtt{dom}(\Delta) \text{ implies} & (\Gamma, \Delta\cdot s^o[\mathtt{p}] : M; [q]!\langle U\rangle) \xrightarrow{s^o[\mathtt{p}][\mathtt{q}]!\langle v\rangle}_{io} (\Gamma, \Delta\cdot s^o[\mathtt{p}] : M) \\ a \notin \mathtt{dom}(\Gamma), s^i[\mathtt{q}] \notin \mathtt{dom}(\Delta) \text{ implies} & (\Gamma, \Delta\cdot s^o[\mathtt{p}] : M; [\mathtt{q}]!\langle U\rangle) \xrightarrow{s^o[\mathtt{p}][\mathtt{q}]!(a)}_{io} (\Gamma\cdot a : U, \Delta\cdot s^o[\mathtt{p}] : M) \end{array}$$

$$\frac{T \ast M = \mathscr{T}[[\mathtt{q}]?(S);T'] \quad \mathscr{T} \text{ not contain prefix on role } \mathtt{q} \quad s^o[q] \notin \mathtt{dom}(\Delta)}{(\Gamma, \Delta\cdot s[\mathtt{p}] : T\cdot s^i[\mathtt{p}] : M) \xrightarrow{s^i[\mathtt{p}][\mathtt{q}]?\langle v\rangle}_{io} (\Gamma\cdot v : U, \Delta\cdot s[\mathtt{p}] : T \cdot s^i[\mathtt{p}] : [\mathtt{q}]?(U);M)}$$

$$s^i[\mathtt{q}] \notin \mathtt{dom}(\Delta) \text{ implies } (\Gamma, \Delta\cdot s^o[\mathtt{p}] : M; [\mathtt{q}] \oplus l_k) \xrightarrow{s^o[\mathtt{p}][\mathtt{q}]\oplus l_k}_{io} (\Gamma, \Delta\cdot s^o[\mathtt{p}] : M)$$

$$\frac{T \ast M = \mathscr{T}[[\mathtt{q}]\&\{l_i : T_i\}] \quad \mathscr{T} \text{ not contain prefix on role } \mathtt{q} \quad s^o[\mathtt{q}] \notin \mathtt{dom}(\Delta)}{(\Gamma, \Delta\cdot s[\mathtt{p}] : T\cdot s^i[\mathtt{p}] : M) \xrightarrow{s^i[\mathtt{p}][q]\oplus l_k}_{io} (\Gamma, \Delta\cdot s[\mathtt{p}] : T \cdot s^i[\mathtt{p}] : [\mathtt{q}] \oplus l_k;M)}$$

$$\Delta \longrightarrow \Delta' \vee \Delta = \Delta' \text{ implies} \qquad (\Gamma,\Delta) \xrightarrow{\tau} (\Gamma,\Delta')$$

F.6 Governed Behavioural Semantics for Output Asynchronous MSP

We use the global environment E definition from section 5.1.

We define $\mathtt{out}(\lambda)$ and $\mathtt{inp}(\lambda)$ as $\mathtt{out}(s : \mathtt{p} \to \mathtt{q} : U) = \mathtt{out}(s : \mathtt{p} \to \mathtt{q} : l) = \mathtt{p}$ and as $\mathtt{inp}(s : \mathtt{p} \to \mathtt{q} : U) = \mathtt{inp}(s : \mathtt{p} \to \mathtt{q} : l) = \mathtt{q}$ and $\mathtt{p} \in \ell$ if $\mathtt{p} \in \mathtt{out}(\ell) \cup \mathtt{inp}(\ell)$.

We define the labelled reduction for global environments with respect to the definition of labelled reduction for global environments in §5.1

$$\frac{\{s : G\} \xrightarrow{\ell}_{io} \{s : G'\} \quad \mathtt{q} \notin \ell \vee \mathtt{p}, \mathtt{q} \notin \mathtt{out}(\ell)}{\{s : \mathtt{p} \to \mathtt{q} : \langle U \rangle.G\} \xrightarrow{\ell}_{io} \{s : \mathtt{p} \to \mathtt{q} : \langle U \rangle.G'\}}$$

$$\frac{\forall i \in I, \{s : G_i\} \xrightarrow{\ell}_{io} \{s : G'_i\} \quad \mathtt{q} \notin \ell \vee \mathtt{p}, \mathtt{q} \notin \mathtt{out}(\ell)}{\{s : \mathtt{p} \to \mathtt{q} : \{l_i : G_i\}\} \xrightarrow{\ell}_{o} \{s : \mathtt{p} \to \mathtt{q} : \{l_i : G'_i\}_{i \in I}\}}$$

We change the rules for permutation to subsume both Input/output asynchrony and role permutation with the two rules requiring that two sequenced actions can be permuted if either the condition from output asynchronous semantics holds, or the condition from input asynchronous semantics holds.

We use Definition 5.2 for global configurations, the global environment LTS in figure 7 (description in § 5.1) and Definition 5.4 for configuration transition as defined using the global reduction for the input/output asynchronous MSP. We also prove that proposition 5.2 holds for the input/output asynchronous MSP.

We define global configuration barbs:

$$\begin{array}{rl}
(\Gamma, \Delta \cdot s^o[\mathtt{p}] : M; [\mathtt{q}]!\langle U \rangle, E) \downarrow_{s[\mathtt{p}][\mathtt{q}]} & \text{if } s^i[\mathtt{q}] \notin \mathtt{dom}(\Delta), E \xrightarrow{s:\mathtt{p} \to \mathtt{q}:U} \\
(\Gamma, \Delta \cdot s^o[\mathtt{p}] : [\mathtt{q}] \oplus l_k; M, E) \downarrow_{s[\mathtt{p}][\mathtt{q}]} & \text{if } s^i[\mathtt{q}] \notin \mathtt{dom}(\Delta), E \xrightarrow{s:\mathtt{p} \to \mathtt{q}:l_k}, k \in I \\
(\Gamma, \Delta, E) \downarrow_a & \text{always}
\end{array}$$

G Properties of MSP Behavioural Theory

Definition G.1 (α Multiparty Session π-calculus).

- We define $\mathscr{P}^s, \mathscr{P}^i, \mathscr{P}^o$ and $\mathscr{P}^{io}$ as the set of processes for synchronous MSP, input asynchronous MSP, output asynchronous MSP, and input/output asynchronous MSP.
- Define $\alpha = s \mid i \mid o \mid io$.
- Define a partial order $\sqsubseteq$ as: $s \sqsubset i, s \sqsubset o, i \sqsubset io, o \sqsubset io$.

We can map processes following the $\sqsubset$ ordering by allowing the uniform syntax for input/output asyncronous MSP to be defined for all four calculi. Furhtermore we need to use the uniform [Link] from the input/output asynchronous MSP definition. This definition allows for the presence of both input and output session endpoints in all four calculi. This is done without loss of generality since the rest of the semantic theory for each calculus restricts the use of configuration endpoints accordingly.

Definition G.2. We define the reduction between a session name and a session endpoint as $\longrightarrow_b$.

$$P \longrightarrow_b P'$$

if $\longrightarrow_b$ derives from the reduction rules [Send], [Rcv], [Rcv-S], [Sel], [Bra], [Comm]. We call $\longrightarrow_b$ the session endpoint reduction.

Lemma G.1 (Session endpoint linearity). *If* $\Gamma \vdash P_1 \triangleright \Delta_1 \Longrightarrow_b \Gamma \vdash P_2 \triangleright \Delta_2$ *then* $\Gamma \vdash P_1 \triangleright \Delta_1 \approx \Gamma \vdash P_2 \triangleright \Delta_2$.

Proof. We are based on the fact that $\longrightarrow_b$ is a linear transition. We define the relation $S = \{(\Gamma \vdash P_1 \triangleright \Delta_1, \Gamma \vdash P_2 \triangleright \Delta_2)\} \cup R$ where R is the reflexive relation on the derivatives of $\Gamma \vdash P_2 \triangleright \Delta_2$.

Because $\Longrightarrow_b$ is linear we can easily show that S is a bisimulation. □

Lemma G.2. *Let* $\Gamma \vdash P_1 \triangleright \Delta_1 \xrightarrow{\ell} \Gamma \vdash P_2' \triangleright \Delta_2'$ *and* $\Gamma \vdash P_1 \triangleright \Delta_1 \Longrightarrow_b \Gamma \vdash P_1' \triangleright \Delta_1'$. *Then* $\Gamma \vdash P_2' \triangleright \Delta_2' \Longrightarrow_b \Gamma \vdash P_2 \triangleright \Delta_2$ *and* $\Gamma \vdash P_1' \triangleright \Delta_1' \xrightarrow{\ell} \Gamma \vdash P_2 \triangleright \Delta_2$.
Proof. □

Theorem G.1. Let $P_1, P_2 \in \mathscr{P}^{\alpha}$ and $\alpha \sqsubseteq \alpha'$

– If $\Gamma \vdash_{\alpha} P_1 \triangleright \Delta$ then $\Gamma \vdash_{\alpha'} P_1 \triangleright \Delta'$ and $*(\Delta) = *(\Delta')$
– If $\Gamma \vdash_{\alpha} P_1 \triangleright \Delta_1 \approx^{\alpha} P_2 \triangleright \Delta_2$ then $\Gamma \vdash_{\alpha'} P_1 \triangleright \Delta_1' \approx^{\alpha'} P_2 \triangleright \Delta_2'$.

Proof (Sketch). We use lemma G.1 to achieve the required result.
Case: $\alpha = s, \alpha' = i$
Let

$$S = \{(P_1, P_2), \ \mid \ P_1 \approx_s P_2\}$$

We can show that if $P_1 \ S \ P_2$

1. $\Gamma \vdash P_1 \triangleright \Delta_1 \xrightarrow{\ell} \Gamma \vdash P_1'' \triangleright \Delta_1''$ then $\Gamma \vdash P_2 \triangleright \Delta_2 \overset{\ell}{\Longrightarrow} \Gamma \vdash P_2' \triangleright \Delta_2'$ and $\Gamma \vdash P_1'' \triangleright \Delta_1'' \Longrightarrow_b \Gamma \vdash P_1' \triangleright \Delta_1' \ S \ \Gamma \vdash P_2' \triangleright \Delta_2'$.
2. The symmetric case.

From lemma G.1 we get that relation S is a bisimulation up-to $\Longrightarrow_b$.
Case: $\alpha = s, \alpha' = o$
Let

$$S = \{(P_1', P_2), (P_1, P_2') \ \mid \ P_1 \approx_s P_2, P_1 \Longrightarrow_b P_1', P_2 \Longrightarrow_b P_2'\}$$

We can show that if $P_1 \ S \ P_2$
Case: $\ell = \longrightarrow_b$

$\Gamma \vdash P_1 \triangleright \Delta_1 \xrightarrow{\ell} \Gamma \vdash P_1' \triangleright \Delta_1'$ then $\Gamma \vdash P_1' \triangleright \Delta_1' \ S \ \Gamma \vdash P_2 \triangleright \Delta_2$.
Case: otherwise

From lemma G.1 we can get that if $\Gamma \vdash P_1 \triangleright \Delta_1 \xrightarrow{\ell} \Gamma \vdash P_1' \triangleright \Delta_1'$ then $\Gamma \vdash P_2 \triangleright \Delta_1 \xrightarrow{\ell} \Gamma \vdash P_2' \triangleright \Delta_2'$ and $\Gamma \vdash P_1' \triangleright \Delta_1' \ S \ \Gamma \vdash P_2 \triangleright \Delta_2$
Case: $\alpha' = io$
We rely on similar arguments to prove that $\Gamma \vdash P_1 \triangleright \Delta_1 \Longrightarrow_b \xrightarrow{\ell} \Longrightarrow_b \Gamma \vdash P_1' \triangleright \Delta_1'$ results in a closed up-to $\Longrightarrow_b$ bisimulation relation.

□

H Comparing Behaviour

We discuss asynchrony inside a global session protocol. Consider the global protocols:

$$E_1 = s_1 : 1 \to 3 : \langle U \rangle.2 \to 3 : \langle U \rangle.\mathtt{end} \cdot s_2 : 1 \to 2 : \langle\rangle.\mathtt{end}$$
$$E_2 = s_1 : 2 \to 3 : \langle U \rangle.1 \to 3 : \langle U \rangle.\mathtt{end} \cdot s_2 : 1 \to 2 : \langle\rangle.\mathtt{end}$$

$$E_3 = s_1 : 3 \to 1 : \langle U \rangle.3 \to 2 : \langle U \rangle.\mathtt{end} \cdot s_2 : 1 \to 2 : \langle\rangle.\mathtt{end}$$
$$E_4 = s_1 : 3 \to 1 : \langle U \rangle.3 \to 2 : \langle U \rangle.\mathtt{end} \cdot s_2 : 1 \to 2 : \langle\rangle.\mathtt{end}$$

$$E_5 = s_1 : 1 \to 3 : \langle U \rangle.3 \to 2 : \langle U \rangle.\mathtt{end} \cdot s_2 : 1 \to 2 : \langle\rangle.\mathtt{end}$$
$$E_6 = s_1 : 3 \to 2 : \langle U \rangle.1 \to 3 : \langle U \rangle.\mathtt{end} \cdot s_2 : 1 \to 2 : \langle\rangle.\mathtt{end}$$

Protocols $E_1, _3, E_5$ implement a sequential behaviour on role 3 from roles 1 and 2 and respectively protocols E_2, E_4, E_6 implement the permutation of protocols $E_1, _3, E_5$. As a result roles 1 and 2 project to the same local type (i.e can have the same implementation) for a protocol and its permutation. We say that protocols E_1, E_2 implement an input permutation E_3, E_4 implement an output permutation and E_5, E_6 implement an input/output permutation. We can consider the following partial implementations:

$$P_1 = s_1[1][3]!\langle v \rangle;\mathbf{0} \mid s_1[2][3]!\langle v \rangle;\mathbf{0}$$
$$P_3 = s_1[1][3]?(v);\mathbf{0} \mid s_1[2][3]?(v);\mathbf{0}$$
$$P_5 = s_1[1][3]!\langle v \rangle;\mathbf{0} \mid s_1[2][3]?(v);\mathbf{0}$$

Considering, respectively, minimal localisation on processes P_1, P_2, P_3, we can observe either of the two actions on the parallel composition with respect to non-govern typed transition semantics for all calculi defined, since their is no sequence on the parallel composition. Intuitively we would like to impose a sequence on the two actions of each of the processes. We use a global sequencing protocol with the type $E = 1 \to 2 : \langle\rangle.\mathtt{end}$ to perform sequencing.

$$P_2 = s_1[1][3]!\langle v \rangle;s_2[1][2]!\langle\rangle;\mathbf{0} \mid s_2[2][1]?();s_1[2][3]!\langle v \rangle;\mathbf{0}$$
$$P_4 = s_1[1][3]?(v);s_2[1][2]!\langle\rangle;\mathbf{0} \mid s_2[2][1]?();s_1[2][3]?(v);\mathbf{0}$$
$$P_6 = s_1[1][3]!\langle v \rangle;s_2[1][2]!\langle\rangle;\mathbf{0} \mid s_2[2][1]?();s_1[2][3]?(v);\mathbf{0}$$

Note that processes P_2, P_4, P_6 are consistent with the protocols E_1 and E_2, E_3 and E_4, E_5 and E_6 respectively. We want to study the behaviour of sequenced and not sequenced process with respect to synchronous and asynchronous semantics.

$$\begin{array}{lll} P_1 \not\approx_s P_2 & P_3 \not\approx_s P_4 & P_5 \not\approx_s P_6 \\ P_1 \approx_o P_2 & P_3 \approx_o P_4 & P_5 \not\approx_o P_6 \\ P_1 \approx_i P_2 & P_3 \approx_i P_4 & P_5 \not\approx_i P_6 \\ P_1 \approx_{io} P_2 & P_3 \approx_{io} P_4 & P_5 \not\approx_{io} P_6 \end{array}$$

As expected in the synchronous multiparty session π-calculus, their is a behavioural distinction between the pairs of processes in the synchronous case. In the asynchronous case we observe the same results for input and output permutations for all three asynchronous calculi. There is though behavioural distinction for pairs P_5 and P_6 (i.e. for input/output permutation). Note that P_6 can simulate the behaviour of P_5.

Results for asynchronous input and output permutations inside a global protocol affect the governed asynchronous theory, following theorem 5.2. Surprisingly protocol permutations in either of the governed asynchronous calculi makes no difference in the equivalence.

In the case of behavioural distinction we can govern the behaviour of processes to achieve behavioural equivalence:

$$\begin{array}{lll} E_2 \vdash P_1 \not\approx^s_g P_2 & E_4 \vdash P_3 \not\approx^s_g P_4 & E_6 \vdash P_5 \not\approx^s_g P_6 \\ E_1 \vdash P_1 \approx^s_g P_2 & E_3 \vdash P_3 \approx^s_g P_4 & E_4 \vdash P_5 \approx^s_g P_6 \\ E_5 \vdash P_5 \approx^o_g P_6 & E_5 \vdash P_5 \approx^i_g P_6 & E_5 \vdash P_5 \approx^{io}_g P_6 \\ E_6 \vdash P_5 \not\approx^o_g P_6 & E_6 \vdash P_5 \not\approx^i_g P_6 & E_6 \vdash P_5 \not\approx^{io}_g P_6 \end{array}$$

So far we have seen the asynchronous multiparty π-calculi to exhibit the same behaviour on permutation patterns inside a global protocol. To demonstrate a distinction let:

$$\begin{array}{ll} P_1 = s[3][1]?(v); s[3][2]?(v); \mathbf{0} & P_2 = s[3][2]?(v); s[3][1]?(v); \mathbf{0} \\ P_3 = s[3][1]!\langle v\rangle; s[3][2]!\langle v\rangle; \mathbf{0} & P_4 = s[3][2]!\langle v\rangle; s[3][1]!\langle v\rangle; \mathbf{0} \end{array}$$

with

$$\begin{array}{llll} P_1 \not\approx^s P_2 & P_1 \not\approx^o P_2 & P_1 \approx^i P_2 & P_1 \approx^{io} P_2 \\ P_3 \not\approx^s P_4 & P_3 \approx^o P_4 & P_3 \not\approx^i P_4 & P_3 \approx^{io} P_4 \end{array}$$

to distinct between the output, input and input/output asynchronous multiparty π calculus. We cannot govern the behaviour of each pair with the same protocol. The governed behavioural results follow the same pattern as the non-governed equivalences:

$$\begin{array}{llll} E_1 \vdash P_1 \not\approx^s E_2 \vdash P_2 & E_1 \vdash P_1 \approx^o E_2 \vdash P_2 & E_1 \vdash P_1 \not\approx^i E_2 \vdash P_2 & E_1 \vdash P_1 \approx^{io} E_2 \vdash P_2 \\ E_3 \vdash P_3 \not\approx^s E_4 \vdash P_4 & E_3 \vdash P_3 \not\approx^o E_4 \vdash P_4 & E_3 \vdash P_3 \approx^i E_4 \vdash P_4 & E_3 \vdash P_3 \approx^{io} E_4 \vdash P_4 \end{array}$$

The above distinction between asynchronous multiparty session π-calculus, also holds by permutation between different session channels. Consider:

$$\begin{array}{ll} P_1 = s_1[1][2]!\langle v\rangle; s_2[1][2]!\langle v\rangle; \mathbf{0} & P_2 = s_2[1][2]!\langle v\rangle; s_1[1][2]!\langle v\rangle; \mathbf{0} \\ P_3 = s_1[1][2]?(v); s_2[1][2]?(v); \mathbf{0} & P_4 = s_2[1][2]?(v); s_1[1][2]?(v); \mathbf{0} \end{array}$$

and

$$\begin{array}{llll} P_1 \not\approx^s P_2 & P_1 \approx^o P_2 & P_1 \not\approx^i P_2 & P_1 \approx^{io} P_2 \\ P_3 \not\approx^s P_4 & P_3 \not\approx^o P_4 & P_3 \approx^i P_4 & P_3 \approx^{io} P_4 \end{array}$$

Input and output asynchronous semantics do not distinguish between input and output permutations, respectively, between different global protocols, while input/output asynchronous semantics allow both input and output permutations.

An example that shows the govern equivalence distinction between the different asynchronous calculi:

$$\begin{aligned}
P_1 &= s_1[2][1]!\langle v_1\rangle; s_2[3][1]!\langle v_2\rangle; (s_2[1][3]!\langle v_3\rangle; \mathbf{0} \mid s_2[2][3]?(x); \mathbf{0})\\
P_2 &= s_2[3][1]!\langle v_2\rangle; s_1[2][1]!\langle v_1\rangle; (s_2[1][3]!\langle v_3\rangle; s_3[1][2]!\langle v_4\rangle; \mathbf{0} \mid s_3[2][1]?(y); s_2[2][3]?(x); \mathbf{0})
\end{aligned}$$

$$\begin{aligned}
P_3 &= s_1[2][1]?(z_1); s_2[3][1]?(z_2); (s_2[1][3]!\langle v_3\rangle; \mathbf{0} \mid s_2[2][3]?(x); \mathbf{0})\\
P_4 &= s_2[3][1]?(z_2); s_1[2][1]?(z_1); (s_2[1][3]!\langle v_3\rangle; s_3[1][2]!\langle v_4\rangle; \mathbf{0} \mid s_3[2][1]?(y); s_2[2][3]?(x); \mathbf{0})
\end{aligned}$$

and

$$\begin{aligned}
E_1 &= s_1 : 2 \to 1 : \langle U_1\rangle.\mathtt{end} \cdot s_2 : 3 \to 1 : \langle U_2\rangle.1 \to 3 : \langle U_3\rangle.3 \to 2 : \langle U_4\rangle.\mathtt{end} \cdot s_3 : 1 \to 2 : \langle U_5\rangle.\mathtt{end}\\
E_2 &= s_1 : 2 \to 1 : \langle U_1\rangle.\mathtt{end} \cdot s_2 : 3 \to 1 : \langle U_2\rangle.3 \to 2 : \langle U_4\rangle.1 \to 3 : \langle U_3\rangle.\mathtt{end} \cdot s_3 : 1 \to 2 : \langle U_5\rangle.\mathtt{end}
\end{aligned}$$

$$\begin{aligned}
E_3 &= s_1 : 1 \to 2 : \langle U_1\rangle.\mathtt{end} \cdot s_2 : 1 \to 3 : \langle U_2\rangle.1 \to 3 : \langle U_3\rangle.3 \to 2 : \langle U_4\rangle.\mathtt{end} \cdot s_3 : 1 \to 2 : \langle U_5\rangle.\mathtt{end}\\
E_4 &= s_1 : 1 \to 2 : \langle U_1\rangle.\mathtt{end} \cdot s_2 : 1 \to 3 : \langle U_2\rangle.3 \to 2 : \langle U_4\rangle.1 \to 3 : \langle U_3\rangle.\mathtt{end} \cdot s_3 : 1 \to 2 : \langle U_5\rangle.\mathtt{end}
\end{aligned}$$

with

$$\begin{array}{lll}
E_1 \vdash P_1 \approx_g^o P_2 & E_1 \vdash P_1 \not\approx_g^i P_2 & E_1 \vdash P_1 \approx_g^{io} P_2\\
E_2 \vdash P_1 \not\approx_g^o P_2 & E_2 \vdash P_1 \not\approx_g^i P_2 & E_2 \vdash P_1 \not\approx_g^{io} P_2
\end{array}$$

$$\begin{array}{lll}
E_3 \vdash P_3 \not\approx_g^o P_4 & E_3 \vdash P_3 \approx_g^i P_4 & E_3 \vdash P_3 \approx_g^{io} P_4\\
E_4 \vdash P_3 \not\approx_g^o P_4 & E_4 \vdash P_3 \not\approx_g^i P_4 & E_4 \vdash P_3 \not\approx_g^{io} P_4
\end{array}$$

I Usecase from OOI

We tested the real world usecase UC.R2.13 "Acquire Data From Instrument" from the Ocean Observatories Initiative (OOI) [14] Use Case library (Release 2). In this usecase, a user program (`U`) is connected to the Integrated Observatory Network (ION), which provides the infrastructure between users and remote sensing instruments. The user requests, via an ION agent service (`A`), the acquisition of data from an instrument (`I`). In the implementation, the ION agent (`A`) is realised by two sub ION agents (`A1` and `A2`) which internally interact and synchronise together.

We are able to reason that the behaviour of `A1` and `A2` is equated by `A` by $\approx_g^s$ applying the thread transformation in Example 5.2.

We explain here with more detailed examples.

I.1 Usecase scenario 1

Suppose the scenario where a user program (`U`) wants to acquire data from the instrument (`I`) and at the same time acuire processed data from a agent service ($\mathtt{A}_1$). The communication between an agent (`A`) and an instrument happens on a separate private session.

– A new session connection s_1 is established between (`U`), (`I`) and (`A`).
– A new session connection s_2 is established between (`A`) and (`I`).
– (`I`) sends raw data through s_2 to (`A`).
– (`A`) sends processed data (format 1) through s_1 to (`U`).

– (A) sends acknowledgement through s_2 to (I).
– (I) sends processed data (format 2) through s_1 to (U).

The above scenario is implemented as follows:

$$I \mid A \mid U$$

where

$$\begin{aligned}
I &= a[\mathtt{i}](s_1).\overline{b}[\mathtt{i}](s_2).s_2[\mathtt{i}][\mathtt{a_1}]!\langle \mathtt{rd}\rangle;s_2[\mathtt{i}][\mathtt{a_1}]?(x);s_1[\mathtt{i}][\mathtt{u}]!\langle \mathtt{pd}\rangle;\mathbf{0}\\
A &= a[\mathtt{a_1}](s_1).b[\mathtt{a_1}](s_2).s_2[\mathtt{a_1}][\mathtt{i}]?(x);s_1[\mathtt{a_1}][\mathtt{u}]!\langle \mathtt{pd}\rangle;s_2[\mathtt{a_1}][\mathtt{i}]!\langle \mathtt{ack}\rangle;\mathbf{0}\\
U &= \overline{a}[\mathtt{u}](s_1).s_1[\mathtt{u}][\mathtt{a_1}]?(x);s_1[\mathtt{u}][\mathtt{i}]?(y);\mathbf{0}
\end{aligned}$$

and $\mathtt{i}$ is the instrument role, $\mathtt{a_1}$ is the agent role and $\mathtt{u}$ is the user role.

I.2 Usecase scenario 2

Use case scenario 1 implementation requires from the instrument program to process raw data in a particular format (format 2) before sending them to the user program. On a more fine-grain level the instrument program invokes an agent service to process the raw data into the desired data format.

To capture this in a more modular and fine-tuned implementation we assume a scenario with the user program (U), the instrument (I) and agents ($\mathtt{A}_1$) and ($\mathtt{A}_2$)

– A new session connection s_1 is established between (U), ($\mathtt{A}_1$) and ($\mathtt{A}_2$).
– A new session connection s_2 is established between ($\mathtt{A}_1$), ($\mathtt{A}_2$) and (I).
– (I) sends raw data through s_2 to ($\mathtt{A}_1$).
– (I) sends raw data through s_2 to ($\mathtt{A}_2$).
– ($\mathtt{A}_1$) sends processed data (format 1) through s_1 to (U).
– ($\mathtt{A}_1$) sends acknowledgement through s_2 to (I).
– ($\mathtt{A}_2$) sends processed data (format 2) through s_1 to (U).
– ($\mathtt{A}_2$) sends acknowledgement through s_2 to (I).

The process is now refined as

$$I' \mid A_1 \mid A_2 \mid U$$

where

$$\begin{aligned}
I' &= a[\mathtt{i'}](s_1).\overline{b}[\mathtt{i'}](s_2).s_2[\mathtt{i'}][\mathtt{a_1}]!\langle \mathtt{rd}\rangle;s_2[\mathtt{i'}][\mathtt{a_2}]!\langle \mathtt{rd}\rangle;s_2[\mathtt{i'}][\mathtt{a_1}]?(x);s_2[\mathtt{i'}][\mathtt{a_2}]?(y);\mathbf{0}\\
A_1 &= a[\mathtt{a_1}](s_1).b[\mathtt{a_2}](s_2).s_2[\mathtt{a_1}][\mathtt{i'}]?(x);s_1[\mathtt{a_1}][\mathtt{u}]!\langle \mathtt{pd}\rangle;s_2[\mathtt{a_1}][\mathtt{i'}]!\langle \mathtt{ack}\rangle;\mathbf{0}\\
A_2 &= a[\mathtt{a_2}](s_1).b[\mathtt{a_2}](s_2).s_2[\mathtt{a_2}][\mathtt{i'}]?(x);s_1[\mathtt{a_2}][\mathtt{u}]!\langle \mathtt{pd}\rangle;s_2[\mathtt{a_2}][\mathtt{i'}]!\langle \mathtt{ack}\rangle;\mathbf{0}\\
U &= \overline{a}[\mathtt{u}](s_1).s_1[\mathtt{u}][\mathtt{a_1}]?(x);s_1[\mathtt{u}][\mathtt{a_2}]?(y);\mathbf{0}
\end{aligned}$$

and $\mathtt{i'}$ is the instrument role, $\mathtt{a_1}$ and $\mathtt{a_2}$ are the agent roles and $\mathtt{u}$ is the user role. Furthermore for session s_1 we have that $\mathtt{i}$ (from scenario 1) $= \mathtt{a_2}$, since we want to maintain the session s_1 as it is defined in the scenario 1.

I.3 Bisimulations

The two scenarios of the "acquire data from instrument" protocol were implemented with respect to the user process. Under the assumption that $\mathtt{i} = \mathtt{a}_2$ we have the same user process for both scenarios.

Having the user process as the observer we can see that processes

$$A \mid U$$
$$A_1 \mid A_2 \mid U$$

are not bisimilar since (recall that $\mathtt{i} = \mathtt{a}_2$)

$A_1 \mid A_2 \mid U \Longrightarrow \xrightarrow{s_1[\mathtt{a}_2][\mathtt{u}]!\langle \mathtt{pd} \rangle}$ and $A \mid U \Longrightarrow \overset{s_1[\mathtt{i}][\mathtt{u}]\langle \mathtt{pd} \rangle}{\not\longrightarrow}$.

The two processes though are equivalent and interchangable under governed semantics.

We begin with the definition of the global environment.

$$= s_1 : \mathtt{a}_1 \to \mathtt{u} : \langle \mathtt{PD} \rangle.\mathtt{a}_2 \to \mathtt{u} : \langle \mathtt{PD} \rangle.$$

The global protocol governs processes $A \mid U$ and $A_1 \mid A_2 \mid U$ to always observe action $\xrightarrow{s_1[\mathtt{a}_2][\mathtt{u}]!\langle \mathtt{pd} \rangle}$ after action $\xrightarrow{s_1[\mathtt{a}_1][\mathtt{u}]!\langle \mathtt{pd} \rangle}$ for both processes.

Note that the global protocol for s_2 is not present in the global environment since s_2 after its creation is restricted.