Observational Equivalence for Multiparty Sessions

Dedicated to Paweł Urzyczyn on the occasion of his 65th birthday

Paula Severi
Department of Computer Science, University of Leicester, Leicester UK
ps396@leicester.ac.uk

Mariangiola Dezani-Ciancaglini*
Dipartimento di Informatica, Università di Torino, Torino Italy
dezani@di.unito.it

Abstract. Multiparty sessions are concurrent processes, which allow several participants to communicate by sending and receiving messages. We consider an observational preorder of processes, that captures the idea that the whole session remains correct after replacing one process by another one. This preorder is characterised by means of a structural preorder between processes, which mimics the subtyping relation between session types from the literature.

1. Introduction

Multiparty sessions are concurrent processes which allow several participants to communicate by sending and receiving messages. A type system of multiparty sessions guarantees certain correctness properties such as safety and deadlock-freedom [1,2,3]. If we already know that the whole session is “correct” (possible because it was type checked) and only one participant needs to be modified, how can we guarantee that the ‘local’ change is sound without having to check again the whole session? For this, we consider an observational preorder of processes that captures the idea that a process could be replaced by another one preserving the correctness of the whole session.

*Partially supported by Ateneo/Compagnia di San Paolo 2016/2018 project “MnemoComputing - Components for Processing In Memory”.

Address for correspondence: Paula Severi, Department of Computer Science, University of Leicester, Leicester UK.
The main contribution of this paper is a characterisation of the observational preorder of processes inside sessions. This allows the programmer to know which process replacements in the session are safe and which ones are not.

We illustrate our motivation with an example. The Viva is a meeting between a student, a project supervisor, and a second supervisor to examine the student work. The first supervisor may need to exchange several emails with the second supervisor and the student to agree on time slots to conduct the Viva. The communication protocol VIVA prescribes the following message sequence:

1. the first supervisor sends a date in which she is available to the second supervisor and the student,
2. the second supervisor either
   (a) sends \textit{ok} to the first supervisor and then the first supervisor sends \textit{ok} to the student or
   (b) sends \textit{notok} to the first supervisor and after the first supervisor informs the student, and
   the same protocol repeats until they find an agreement.

Following \cite{1, 2} this protocol can be specified by a global type as follows:

\[
G = \text{fst} \rightarrow \text{snd} : \text{date}. \\
\text{fst} \rightarrow \text{std} : \text{date}. \\
\text{snd} \rightarrow \text{fst} : \{\text{ok} : \text{fst} \rightarrow \text{std} : \text{ok}, \text{notok} : \text{fst} \rightarrow \text{std} : \text{notok} : G\}
\]

where the alternatives are put in brackets and separated by commas.

The VIVA protocol can be implemented by the lock-free multiparty session:

\[
M = \text{fst} \triangleright P \triangleright \text{snd} \triangleright Q \triangleright \text{std} \triangleright R
\]

where using ! for outputs and ? for inputs we can put:

\[
P = \text{snd}!\text{date}.\text{std}!\text{date}.\text{std}?!\{\text{ok}.\text{std}!\text{ok}, \text{notok}.\text{std}!\text{notok}.P\}
\]
\[
Q = \text{fst}?\text{date}!\text{fst}?!\{\text{ok}, \text{notok}.Q\}
\]
\[
R = \text{fst}?\text{date}!\text{fst}?!\{\text{ok}, \text{notok}.R\}
\]

On one hand, soundness of the structural preorder allows us to make safe replacements in the above implementation without losing the property of lock-freedom. For example, the first supervisor could decide to let the second supervisor send him another date also, as an alternative to \textit{ok} and \textit{notok}.

\[
P' = \text{snd}!\text{date}.\text{std}!\text{date}.\text{std}?!\{\text{ok}.\text{std}!\text{ok}, \text{notok}.\text{std}!\text{notok}.P'\}
\]

The new session \[
M' = \text{fst} \triangleright P' \triangleright \text{snd} \triangleright Q \triangleright \text{std} \triangleright R
\]

inherits the property of lock-freedom from the old version.

On the other hand, completeness of the structural preorder tells us that the first supervisor cannot cancel the \textit{notok} message from the second supervisor without risking a deadlocked session. We know that if

\[
P'' = \text{snd}!\text{date}.\text{std}!\text{date}.\text{std}?!\{\text{ok}.\text{std}!\text{ok}\}
\]
then the new session \( M'' = \text{fst} \triangleright P'' | \text{snd} \triangleright Q | \text{std} \triangleright R \) can reduce to a stuck session.

The characterisation of the observational preorder is given by means of a structural preorder between processes which mimics the standard subtyping between session types \([4, 5]\). Both preorders are defined independently from the notion of types. This has several advantages. First, there are many different notions of projections and type systems in the literature and the new type independent notions of preorders could be applied in combination with any variant of type system that guarantees lock-freedom. The programmer can first compile using any available type checker and then she can refine and adapt the processes locally using the structural preorder. Second it simplifies the task at design-time for the programmer, since she can refine the process locally without running the type checker on the whole protocol every time it needs to be adapted for maintenance. The correctness of the refinement is guaranteed by running an algorithm that checks the structural preorder: it only amounts to do a syntactic comparison between the old and new processes. Though types do not play a role in the definitions of the preorders, we use a type system as an auxiliary tool to prove the equivalence of the preorders.

Since we consider only one multiparty session, typability assure lock-freedom as expected \([1, 6]\). We introduce a notion of lock-freedom, dubbed \textit{strong lock-freedom}, which is more demanding than the standard one \([7, 6]\). The observational preorder actually guarantees that a process can be replaced by another process in any session without losing the correctness property of strong lock-freedom (not just lock-freedom), while the structural preorder compares them by their choices of messages. We show that:

- strong lock-freedom is decidable for multiparty sessions;
- well-typed multiparty sessions are strongly lock-free;
- the observation preorder is equivalent to the structural preorder.

**Outline.** The multiparty session calculus, the strong lock-freedom and the observational preorder are given in Section 2 together with the proof of decidability of strong lock-freedom. The typing system and the structural preorder are defined in Section 3. The properties of the typing system such as progress and strong lock-freedom are proved in Section 4. The equivalence between the observational and the structural preorder is shown in Section 5. Sections 6 and 7 discuss related and future works, respectively. The Appendix contains some technical proofs and the algorithm for checking the structural preorder.

### 2. Multiparty Session Calculus

This section introduces the syntax and the semantics of a synchronous multiparty session calculus similar to the one defined in \([8]\), the main difference being that definitions are coinductive instead of inductive.
2.1. Processes

Processes implement the behaviours of single participants.

We use the following base sets and notation: messages, ranged over by \( \ell, \ell', \ldots \); session participants, ranged over by \( p, q, \ldots \); processes, ranged over by \( P, Q, \ldots \); multiparty sessions, ranged over by \( M, M', \ldots \); integers, ranged over by \( n, m, i, j, \ldots \).

The input process \( p?\{\ell_i.P_i \mid 1\leq i\leq n}\) waits for one of the messages \( \ell_i \) from participant \( p \) and the output process \( p!\{\ell_i.P_i \mid 1\leq i\leq n\} \) chooses one message \( \ell_i \) and sends it to participant \( p \). After sending or receiving \( \ell_i \) the process reduces to \( P_i \) \( (1 \leq i \leq n) \). We use \( \Lambda \) as shorthand for \( \{\ell_i.P_i \mid 1\leq i\leq n\} \).

The set \( \Lambda \) in \( p?\Lambda \) acts as an external choice, while the same set in \( p!\Lambda \) acts as an internal choice. In a full-fledged calculus, messages would carry values, namely they would be of the form \( \ell(v) \). Here for simplicity we consider only pure messages.

It is handy to first define pre-processes, since the processes must satisfy conditions which can be easily defined using the tree representation of pre-processes.

**Definition 2.1. (Pre-Process)**

We say that \( P \) is a pre-process and \( \Lambda \) is a pre-choice of messages if they are generated by the grammar:

\[
\begin{align*}
P & ::= \text{coinductive} & 0 & | & p?\Lambda & | & p!\Lambda \\
\Lambda & ::= \{\ell_i.P_i \mid 1\leq i\leq n\}
\end{align*}
\]

and \( \Lambda \) is a set in the sense that:

1. the order of the components in \( \Lambda \) does not matter and
2. all messages in \( \Lambda \) are pairwise different, i.e. \( \{\ell.P,\ell.Q\} \subseteq \Lambda \), then \( P \) and \( Q \) are equal.

The tree representation of a pre-process is a directed rooted tree, where

- each internal node is labelled by \( p? \) or \( p! \) and has as many children as the number of messages,
- the edge from \( p? \) or \( p! \) to the child \( P_i \) is labelled by \( \ell_i \) and

- the leaves of the tree (if any) are labelled by \( 0 \).

We identify pre-processes with their tree representations and we will sometimes refer to the trees as the processes themselves. The tree representations of \( p?\{\ell_i.P_i \mid 1\leq i\leq n\} \) and \( p!\{\ell_i.P_i \mid 1\leq i\leq n\} \) are illustrated in the following picture:

Since pre-processes are defined coinductively, their tree representation could be infinite.
Example 2.2. The tree representation of $P = p?\{\ell.P, \ell'\}$ is illustrated by the following picture:

![Tree Representation](image)

Definition 2.3. (Process)

We say that a pre–process $P$ is a process if the tree representation of $P$ is regular (namely, it has finitely many distinct subtrees). We say that $\Lambda$ is a choice of messages if all pre–processes in $\Lambda$ are processes.

The regularity condition implies that we only consider processes admitting a finite description. Example 2.2 can be represented by a finite graph with a cycle:

![Finite Graph](image)

This is equivalent to writing processes with $\mu$-notation and an equality which allows for an infinite number of unfoldings. This is also called the equirecursive approach, since it views processes as the unique solutions of recursive equations [9, Section 20.2]. The existence and uniqueness of a solution follow from known results (see [10] and also Theorem 7.5.34 of [11]).

Example 2.4. The regularity condition also allows us to define processes using (mutually) recursive equations:

\[
\begin{align*}
P_1 &= p_1?\{\ell_1. P_1, \ell_2. P_2, \ell_3. P_3\} \\
P_2 &= p_2!\{\ell_4. P_1, \ell_5. P_2, \ell_6. P_3\} \\
P_3 &= p_3?\{\ell_7. P_1, \ell_8. P_2, \ell_9. P_3\}
\end{align*}
\]

Note that the recursive equations are all guarded.

We will write $\ell.P \uplus \Lambda$ for $\{\ell. P\} \cup \Lambda$ if $\ell. P \not\in \Lambda$ and $\Lambda_1 \uplus \Lambda_2$ for $\Lambda_1 \cup \Lambda_2$ if $\Lambda_1 \cap \Lambda_2 = \emptyset$. We will omit curly brackets in choices with only one branch and trailing 0 processes.

We define the set of messages in a choice of messages as

\[
msg(\{\ell_i. P_i \mid 1 \leq i \leq n\}) = \{\ell_i \mid 1 \leq i \leq n\}
\]
and the set $\text{ppts}(P)$ of participants of process $P$ as:

$$
\text{ppts}(0) = \emptyset \\
\text{ppts}(p\{\ell_i.P_i \mid 1 \leq i \leq n\}) = \text{ppts}(p!\{\ell_i.P_i \mid 1 \leq i \leq n\}) \\
= \{p\} \cup \text{ppts}(P_1) \cup \ldots \cup \text{ppts}(P_n)
$$

The regularity of processes assures that the set of participant is finite.

Since all messages in choices are pairwise different, the set of paths of processes are determined by all labels of nodes and edges found on the way, omitting the leaf label 0. Formally the set of paths of a process can be defined as a set of sequences as follows.

$$
\text{paths}(0) = \{\epsilon\} \\
\text{paths}(p\{\ell_i.P_i \mid 1 \leq i \leq n\}) = \bigcup_{1 \leq i \leq n} \{p?\ell_i \rho \mid \rho \in \text{paths}(P_i)\} \\
\text{paths}(p!\{\ell_i.P_i \mid 1 \leq i \leq n\}) = \bigcup_{1 \leq i \leq n} \{p!\ell_i \rho \mid \rho \in \text{paths}(P_i)\}
$$

where $\epsilon$ is the empty sequence.

Note that every infinite path of $P$ has infinitely many occurrences of ? or !.

The set of paths of the process in Example 2.2 consists of $p?\ell...p?\ell_p?\ell'$ for all $n$ and the infinite path $p?\ell_p?\ell_p?...$

### 2.2. Multiparty Sessions

Intuitively, a multiparty session is a (possibly infinite) series of interactions between a fixed number of participants, and serves as a unit of abstraction for describing communication protocols. Formally, a multiparty session is a function from the set of participants into processes. It is written as the parallel composition of pairs participants/processes.

**Definition 2.5. (Multiparty Sessions)**

A multiparty session $\mathcal{M}$ is defined by the following grammar:

$$
\mathcal{M} ::= \text{inductive} \quad p \triangleright P \quad | \quad \mathcal{M} \triangleright \mathcal{M}
$$

and it should satisfy the following conditions:

1. All participants in $\mathcal{M} = p_1 \triangleright P_1 \mid \ldots \mid p_n \triangleright P_n$, are different, i.e. $\mathcal{M}$ is a function from $\{p_1, \ldots, p_n\}$ into $\{P_1, \ldots, P_n\}$.

2. In $p \triangleright P$ the participant $p$ does not occur in the corresponding process $P$ (we do not allow self-communication).

We will use $\prod_{1 \leq i \leq n} p_i \triangleright P_i$ as shorthand for $p_1 \triangleright P_1 \mid \ldots \mid p_n \triangleright P_n$. 

2.3. Operational Semantics

The structural congruence between two multiparty sessions establishes that parallel composition is commutative, associative and has neutral elements \( p \triangleright 0 \) for any \( p \).

\[
\begin{align*}
&[S-\text{PAR 1}] && [S-\text{PAR 2}] && [S-\text{PAR 3}] \\
& p \triangleright 0 \mid M \equiv M & & M \mid M' \equiv M' \mid M & & (M \mid M') \land (M' \mid M'') \equiv M \land (M' \mid M'')
\end{align*}
\]

The reduction for multiparty session allows participants to choose and communicate messages. It may contract several redexes simultaneously, which is handy for the definition of strong lock-freedom (Definition 2.11). We call \( p \triangleright \ell q \) a redex and use \( \Delta \) to range over sets of redexes.

**Definition 2.6.** The labelled transition system (LTS) for multiparty sessions (notation \( \overset{\Delta}{\rightarrow} \)) is the smallest relation closed under the following rules:

\[
\begin{align*}
&[R-\text{COMM}] \\
& \text{msg}(\Lambda) \subseteq \text{msg}(\Lambda') \\
& p \triangleright q!(\ell.P \cup \Lambda) \mid q \triangleright p?\ell.Q \cup \Lambda' \overset{\{p\triangleright q\}}{\rightarrow} p \triangleright P \mid q \triangleright Q
\end{align*}
\]

\[
\begin{align*}
&R-\text{COMP}] \\
& M \equiv M_1 \mid M_2 \mid M_3 \quad M_1 \overset{\Delta_1}{\rightarrow} M'_1 \quad M_2 \overset{\Delta_2}{\rightarrow} M'_2 \quad M'_1 \mid M'_2 \mid M_3 \equiv M' \\
&M \overset{\Delta_1 \cup \Delta_2}{\rightarrow} M'
\end{align*}
\]

Rule [R-COMM] makes the communication possible: participant \( p \) sends message \( \ell \) to participant \( q \). This rule is non-deterministic in the choice of messages. The condition \( \text{msg}(\Lambda) \subseteq \text{msg}(\Lambda') \) assures that the sender can freely choose the message, since the receiver must offer all sender messages and maybe more.

We use \( M \overset{p\triangleright q}{\rightarrow} M' \) as shorthand for \( M \overset{\{p\triangleright q\}}{\rightarrow} M' \). We sometimes write \( \rightarrow \) instead of \( \overset{\Delta}{\rightarrow} \). As usual, \( \rightarrow^* \) denotes the reflexive and transitive closure of \( \rightarrow \). Note that \( M \rightarrow^* M' \) iff \( M = M_0 \overset{\Delta_1}{\rightarrow} M_1 \overset{\Delta_2}{\rightarrow} \cdots M_{n-1} \overset{\Delta_n}{\rightarrow} M_n = M' \) for some \( \Delta_1, \ldots, \Delta_n \).

**Definition 2.7.** (Redexes of Sessions and Sets of Redexes)

1. If \( M \equiv p \triangleright q!(\ell.P \cup \Lambda) \mid q \triangleright p?\ell.Q \cup \Lambda' \mid M' \) and \( \text{msg}(\Lambda) \subseteq \text{msg}(\Lambda') \), then \( p\triangleright q \) is a redex of \( M \). If \( p\triangleright q \) is a redex of \( M \), then the participants \( p, q \) are active in \( M \).

2. A set \( \Delta \) is coherent if no participant occurs twice in \( \Delta \).

3. A set \( \Delta \) is complete for \( M \) if \( \Delta \) contains (i) only redexes that occur in \( M \) and (ii) all the active participants of \( M \).

It is easy to verify that if \( M \overset{\Delta}{\rightarrow} M' \), then \( \Delta \) is coherent.
Example 2.8. Let $\Delta_1 = \{\ell_1.P_1, \ell_2.P_2\}$ and $\Delta_2 = \{\ell_1.Q_1, \ell_2.Q_2, \ell_3.Q_3\}$. The multiparty session $\mathcal{M}_1 = p \triangleright q!\Lambda_1 \mid q \triangleright p?\Lambda_2$ has two possible transitions:

![Diagram](image)

while the multiparty session $\mathcal{M}_2 = p \triangleright q!\Lambda_2 \mid q \triangleright p?\Lambda_1$ does not reduce because $msg(\Lambda_2) \not\subseteq msg(\Lambda_1)$. The redexes of $\mathcal{M}_1$ are $p\ell_1q$ and $p\ell_2q$, while $\mathcal{M}_2$ has no redexes. The sets $\Delta_1 = \{p\ell_1q\}$ and $\Delta_2 = \{p\ell_2q\}$ and $\Delta_3 = \{p\ell_1q, p\ell_2q\}$ are all complete for $\mathcal{M}_1$. Both $\Delta_1$ and $\Delta_2$ are coherent, but $\Delta_3$ is not because both $p$ and $q$ appear twice in $\Delta_3$.

Example 2.9. A session can reduce in different ways. Let $\mathcal{M} = p \triangleright P \mid q \triangleright Q \mid \mathcal{M}'$, where $\mathcal{M}' = r \triangleright R \mid \triangleright S$ and $P = q!\ell_1$ and $Q = p?\ell_1$ and $R = s!\{\ell_2, \ell_3.R\}$ and $S = r?\{\ell_2, \ell_3.S\}$. Then $\mathcal{M}$ can reduce in 5 different ways:

![Diagram](image)

A first lemma relates an arbitrary transition with a sequence of transitions where only one redex is reduced.

**Lemma 2.10.** Let $\Delta = \{p_i\ell_i q_i \mid 1 \leq i \leq n\}$. Then $\mathcal{M} \xrightarrow{\Delta} \mathcal{M}'$ if and only if the following two conditions hold:

1. $\Delta$ is coherent and
2. $\mathcal{M} = \mathcal{M}_1$ and $\mathcal{M}_n = \mathcal{M}'$ and $\mathcal{M}_i \xrightarrow{p_i \ell_i q_i} \mathcal{M}_{i+1}$ for $1 \leq i \leq n-1$.

This lemma can be easily proved by induction on the cardinality of $\Delta$. The direction from right to left in Lemma 2.10 does not hold without condition (I). For example, if $\mathcal{M} = p \triangleright q!\ell_1.q?\ell_2 \mid q \triangleright p?\ell_1.p!\ell_2$, then

$\mathcal{M} \xrightarrow{p\ell_1q} p \triangleright q?\ell_2 \mid q \triangleright p!\ell_2 \xrightarrow{q\ell_2p} p \triangleright 0$

but there is no $\mathcal{M}'$ such that $\mathcal{M} \xrightarrow{p\ell_1q,q\ell_2p} \mathcal{M}'$. Condition (I) is not satisfied, since $\{p\ell_1q, q\ell_2p\}$ is not coherent.

Another example is $\mathcal{M} = p \triangleright P \mid q \triangleright Q$ where $P = q!\{\ell_1.P, \ell_2.P\}$ and $Q = p?\{\ell_1.Q, \ell_2.Q\}$. We have that

$\mathcal{M} \xrightarrow{p\ell_1q} \mathcal{M} \xrightarrow{p\ell_2q} \mathcal{M}$

but there is no $\mathcal{M}'$ such that $\mathcal{M} \xrightarrow{p\ell_1q,p\ell_2q} \mathcal{M}'$. Both $p\ell_1q$ and $p\ell_2q$ are redexes of $\mathcal{M}$, but condition (I) is not satisfied since $\{p\ell_1q, p\ell_2q\}$ is not coherent.
2.4. Strong Lock-Freedom and Observational Preorder

Our flexible notion of reduction allows to easily define strong lock-freedom, which is the basis for observing the behaviour of processes. Strong lock-freedom is more demanding than deadlock-freedom. Deadlock-freedom ensures only the system-wide progress, while strong lock-freedom ensures that all participants continue to do useful work, no computation can ever be blocked by another computation forever. Strong lock-freedom is also more demanding than lock-freedom, because it does not assume fairness in the choice of redexes \[7, 6\].

**Definition 2.11. (Strong Lock-Freedom)**

- A multiparty session \( M \) is stuck (or deadlocked), notation \( \text{stuck}(M) \), if \( M \not\equiv \rho \triangleright 0 \) and there is no multiparty session \( M' \) such that \( M \triangleright M' \).

- We say that \( p \) waits forever (or starves) in \( M \), notation \( \text{wait}_\infty(p, M) \), if \( M \equiv p \triangleright P \mid M_0 \) with \( P \not= 0 \) and there is an infinite reduction sequence

\[
p \triangleright P \mid M_0 \xrightarrow{\Delta_0} p \triangleright P \mid M_1 \xrightarrow{\Delta_1} p \triangleright P \mid M_2 \xrightarrow{\Delta_2} \ldots
\]

such that \( \Delta_i \) is complete for \( M_i \) and \( p \) is not in \( \Delta_i \) for all \( i \geq 0 \).

- We say that \( M \) is a strongly lock-free session, notation \( \text{lockfree}(M) \), if \( M \triangleright^* M' \) implies \( \neg \text{stuck}(M') \) and there is no \( p \) such that \( \text{wait}_\infty(p, M') \). Otherwise we say that \( M \) is locked, notation \( \text{locked}(M) \).

It is easy to verify that deadlock-freedom and strong lock-freedom coincide for binary sessions.

**Example 2.12.** The following multiparty sessions are strongly lock-free.

1. \( M_1 = p \triangleright P \mid q \triangleright Q \), where \( P = q!\ell.P \) and \( Q = p?\ell.Q \), has an infinite reduction

\[
M_1 \xrightarrow{p!q} M_1 \xrightarrow{p!q} M_1 \xrightarrow{p!q} \ldots
\]

but both \( p \) and \( q \) belong to \( \Delta = \{p\ell q\} \).

2. \( M_2 = M_1 \mid M_3 \), where \( M_3 = r \triangleright R \mid s \triangleright S \) and \( R = s!\ell'.R \) and \( S = r?\ell'.S \), has the same reduction

\[
M_2 \xrightarrow{p!q} M_2 \xrightarrow{p!q} M_2 \xrightarrow{p!q} \ldots
\]

Neither \( r \) nor \( s \) belong to \( \{p\ell q\} \), but none of the two have to wait forever in \( M_2 \) because \( \{p\ell q\} \) is not complete for \( M_2 \).

**Example 2.13.** The following two multiparty sessions are not strongly lock-free because they reduce to stuck sessions.

1. \( M_1 = p \triangleright q!\{\ell_1, \ell_2\} \mid q \triangleright p?\{\ell_1, r?\ell_3, \ell_2\} \mid r \triangleright q!\ell_3 \) reduces to \( r \triangleright q!\ell_3 \), which is stuck.
Example 2.14. The following two multiparty sessions are not strongly lock-free because one participant waits forever.

1. $\mathcal{M}_1 = p \triangleright q!\{\ell_1, \ell_2\} | q \triangleright p?\{\ell_1, r?\ell_3, \ell_2, r?\ell_4\} | r \triangleright q!\{\ell_3, \ell_4\}$ reduces to $q \triangleright r?\ell_3 | r \triangleright q!\{\ell_3, \ell_4\}$, which is stuck.

2. $\mathcal{M}_2 = p \triangleright q!\{\ell_1, \ell_2\} | q \triangleright p?\{\ell_1, r?\ell_3, \ell_2, r?\ell_4\} | r \triangleright q!\{\ell_3, \ell_4\}$ reduces to $q \triangleright r?\ell_3 | r \triangleright q!\{\ell_3, \ell_4\}$, which is stuck.

Example 2.14. The following two multiparty sessions are not strongly lock-free because one participant waits forever.

1. $\mathcal{M}_1 = p \triangleright q!\{\ell_1, \ell_2\} | q \triangleright q!\ell_3$, where $P = q!\{\ell_1, \ell_2, P\}$ and $Q = p?\{\ell_1, r!\ell_3, \ell_2, Q\}$, is a multiparty session in which $r$ starves since there is the infinite reduction

$$\mathcal{M}_1 \xrightarrow{p!q} \mathcal{M}_1 \xrightarrow{p!q} \mathcal{M}_1 \xrightarrow{p!q} \ldots$$

where $\{p!q\}$ is complete for $\mathcal{M}_1$ and $r$ is not in $\{p!q\}$.

2. $\mathcal{M}_2 = p \triangleright q!\{\ell_1, \ell_2\} | q \triangleright q!\ell_3$, where $P = q!\{\ell_1, r?\ell_3, \ell_2, P\}$ and $Q = p?\{\ell_1, \ell_2, Q\}$, is a multiparty session in which $r$ starves since there is the infinite reduction

$$\mathcal{M}_2 \xrightarrow{p!q} \mathcal{M}_2 \xrightarrow{p!q} \mathcal{M}_2 \xrightarrow{p!q} \ldots$$

where $\{p!q\}$ is complete for $\mathcal{M}_2$ and $r$ is not in $\{p!q\}$.

Strong lock-freedom is not equivalent to the notion of lock-freedom from [7][6]. If a multiparty session is strongly lock-free, it is also lock-free as defined in [7][6]. However, the converse is not true. For example, $\mathcal{M}_1$ and $\mathcal{M}_2$ of Example 2.14 are lock-free according to [7][6]. Instead $\mathcal{M}_1$ and $\mathcal{M}_2$ of Example 2.13 are not lock-free also according to [7][6].

We show that it is decidable to check whether a multiparty session is strongly lock-free or not. Intuitively, this holds because the LTS only removes symbols, it never adds or duplicates. As a consequence, there are no infinite transition paths where all the sessions are different, i.e. an infinite execution is obtained by performing a cycle. This property is undecidable in $\lambda$-calculus (or $\pi$-calculus), because we can create infinite reduction sequences, where all terms are different [12][13]. This result reflects the simplicity of the multiparty session calculus considered in this paper.

**Theorem 2.15. (Decidability of Strong Lock-Freedom)**

The property of being strongly lock-free is decidable.

**Proof:**

Let $\mathcal{M}$ be a multiparty session. We consider the directed graph whose vertices are the sessions reachable from $\mathcal{M}$ in any number of transitions. Formally, the set of (labelled) vertices is

$$\text{reach}(\mathcal{M}) = \{\mathcal{M}' \mid \mathcal{M} \rightarrow^* \mathcal{M}'\}$$

and there is an edge from $\mathcal{M}_1$ to $\mathcal{M}_2$ with label $\Delta$ if $\mathcal{M}_1 \xrightarrow{\Delta} \mathcal{M}_2$. This graph is finitely branching, since the number of redexes in each $\mathcal{M}' \in \text{reach}(\mathcal{M})$ is finite. The set $\mathcal{S}(P)$ of subtrees of $P$ is given by:

$$\mathcal{S}(P) = \begin{cases} \{P\} & \text{if } P = 0 \\ \{P\} \cup \bigcup_{1 \leq i \leq n} \mathcal{S}(P_i) & \text{if } P = p?\{\ell_i, P_i \mid 1 \leq i \leq n\} \text{ or } P = p!\{\ell_i, P_i \mid 1 \leq i \leq n\} \end{cases}$$
We define the set of subsessions of $\mathcal{M}$, notation $\text{sub}(\mathcal{M})$, as the set of sessions that can be built from $\mathcal{M}$ by replacing processes by their subtrees:

$$\text{sub}(\prod_{1 \leq i \leq n} p \triangleright P_i) = \{ \prod_{1 \leq i \leq n} p \triangleright P_i' \mid P_i' \in S(P_i) \}$$

It is easy to show that if $\mathcal{M} \xrightarrow{\ast} \mathcal{M}'$, then $\mathcal{M}'$ belongs to $\text{sub}(\mathcal{M})$, i.e.

$$\text{reach}(\mathcal{M}) \subseteq \text{sub}(\mathcal{M})$$

The cardinality of $\text{sub}(\prod_{1 \leq i \leq n} p \triangleright P_i)$ is equal to the product of the cardinalities of $S(P_i)$ for all $i$, $1 \leq i \leq n$. Since the processes are regular, this cardinality is finite. Hence, the cardinality of $\text{reach}(\mathcal{M})$ is finite. In order to decide whether $\mathcal{M}$ reduces to a stuck session or not, we just have to check whether there are vertices not labelled by a session equivalent to $p \triangleright 0$ and without outcoming edges in the graph and this can be done in finite time. In order to know whether there is some participant $p$ that waits forever in $\mathcal{M}$, we consider the subgraph whose edges have labels which (i) do not contain $p$ (ii) have complete sets of redexes, and check if that subgraph has a cycle.

If $n$ is the number of subsessions of $\mathcal{M}$ and $m$ is the number of message occurrences in $\mathcal{M}$, then the complexity of the algorithm described in the previous proof is polynomial in $n + m$. Building the graph can take $n + n^2 m$ steps since there are at most $n$ vertices and $n^2 m$ edges (there are at most $m$ edges from one vertex to another one). For each participant $p$, building the subgraph obtained by selecting the edges labelled by complete sets of redexes not containing $p$ and detecting a cycle can be done in $O(n + n^2 m)$. The latter has to be done for all participants and can take $O(n^2 + n^3 m)$, since the number of participants is less than or equal to $n$.

**Example 2.16.** For $1 \leq i \leq 3$, we consider the following:

$$\mathcal{M}_i = p \triangleright P_i \mid q \triangleright Q_i \mid r \triangleright R \quad P_i = q\{\ell_1.P_1, \ell_2.P_2, \ell_3.P_3\} \quad Q_i = p\{\ell_1.Q_1, \ell_2.Q_2, \ell_3.Q_3\}$$

where $R \neq 0$. Then $\text{reach}(\mathcal{M}_1) = \{\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3\}$ and the directed graph with labels is as follows.

The graph has exactly one edge from $i$ to $j$ for $1 \leq i \leq 3$ and $1 \leq j \leq 3$. In order to find out whether $r$ waits forever in $\mathcal{M}_1$, we consider the subgraph whose edges are labelled by complete set of redexes not containing $r$ which in this case coincides with the graph itself. In spite of the fact that there are $2^3$ simple cycles (cycles with no repetition of vertices), it is enough to detect the existence of just one cycle in the graph to conclude that $r$ waits forever.
If we have already checked that a session is strongly lock-free and we modify only some processes, then it would be convenient to have a way of guaranteeing that this ‘local’ change preserves strong lock-freedom, without having to check strong lock-freedom on the whole session again. This is specially important if the number of participants is large or the size of the new processes is small with respect to the size of the whole session.

This motivates the introduction of the observational preorder on processes induced by strong lock-freedom. This preorder compares the ability of a participant associated with the process to have a successful interactions with its execution context.

**Definition 2.17. (Observational Preorder)**

Define the observational preorder on processes, \( P \sqsubseteq Q \), if, for all \( \mathcal{M} \) and \( p \), \( \text{locked}(p \triangleright P | \mathcal{M}) \) implies \( \text{locked}(p \triangleright Q | \mathcal{M}) \). The observational equivalence, \( P \simeq Q \), is given by \( P \sqsubseteq Q \) and \( Q \sqsubseteq P \).

### 3. Typing System

The behaviour of multiparty sessions can be disciplined by means of types, as usual. Our typing system directly assigns global types to multiparty sessions without the usual detour around session types and subtyping [1][2].

#### 3.1. Global Types

Global types describe the whole conversation scenarios of multiparty sessions. The type \( p \rightarrow q : \{\ell_i. G_i | 1 \leq i \leq n\} \) formalises a protocol where participant \( p \) must send to \( q \) a message \( \ell_i \) for some \( 1 \leq i \leq n \) and then depending which \( \ell_i \) was chosen by \( p \) the protocol continues according to \( G_i \). We use \( \Gamma \) as shorthand for \( \{\ell_i. G_i | 1 \leq i \leq n\} \). As for processes, we define first pre-global types and then global types.

**Definition 3.1. (Pre-Global Types)**

We say that \( G \) is a pre-global type and \( \Gamma \) is a pre-choice of communications if they are generated by the grammar:

\[
G ::= \text{coinductive end} \quad | \quad p \rightarrow q : \Gamma \\
\Gamma ::= \{\ell_i. G_i | 1 \leq i \leq n\}
\]

and \( \Gamma \) is a set is the sense that:

1. the order of the components in \( \Gamma \) does not matter and
2. all messages in \( \Gamma \) are pairwise different.

The tree representation of a pre-global type is built as follows:

- each internal node is labelled by \( p \rightarrow q \) and has as many children as the number of messages,
- the edge from \( p \rightarrow q \) to the child \( G_i \) is labelled by \( \ell_i \) and
- the leaves of the tree (if any) are labelled by \text{end}. 

We identify pre-global types with their tree representations and we will sometimes refer to the tree representation as the global types themselves. The tree representation of $p \rightarrow q : \{\ell_i, G_i \mid 1 \leq i \leq n\}$ is illustrated in the picture below:

![Tree representation of $p \rightarrow q$]

**Definition 3.2. (Global Type)**

We say that a pre-global type $G$ is a **global type** if the tree representation of $G$ is **regular**, namely it has finitely many distinct subtrees. We say that a pre-choice of communications $\Gamma$ is a **choice of communications** if all pre-global types in $\Gamma$ are global types.

As for processes, the regularity condition implies that we only consider global types admitting a finite representation.

**Example 3.3.** The regularity condition also allows us to define global types using (mutually) recursive equations:

$$
G_1 = p_1 \rightarrow p_4 : \{\ell_1, G_1, \ell_2, G_2, \ell_3, G_3\}
$$

$$
G_2 = p_4 \rightarrow p_2 : \{\ell_4, G_1, \ell_5, G_2, \ell_6, G_3\}
$$

$$
G_3 = p_3 \rightarrow p_4 : \{\ell_7, G_1, \ell_8, G_2, \ell_9, G_3\}
$$

Note that the recursive equations are all guarded.

We will write $\ell.G \cup \Gamma$ for $\{\ell.G\} \cup \Gamma$ if $\ell.G \notin \Gamma$ and $\Gamma_1 \cup \Gamma_2$ for $\Gamma_1 \cap \Gamma_2 = \emptyset$. We will omit curly brackets in choices with only one branch and trailing end. We define the set of messages in a choice of communications as

$$
\text{msg}(\{\ell_i, G_i \mid 1 \leq i \leq n\}) = \{\ell_i \mid 1 \leq i \leq n\}
$$

and we define the set $\text{ppts}(G)$ of participants of global type $G$ as follows:

$$
\text{ppts}(\text{end}) = \emptyset
$$

$$
\text{ppts}(p \rightarrow q : \{\ell_i, G_i \mid 1 \leq i \leq n\}) = \{p, q\} \cup \bigcup_{1 \leq i \leq n} \text{ppts}(G_i)
$$

The regularity of global types assures that the set of participants is finite.

Since all messages in communication choices are pairwise different, the set of paths of global types are determined by all labels of nodes and edges found on the way, omitting the leaf label end. Formally the set of paths of a global type can be defined as a set of sequences as follows:

$$
\text{paths}(\text{end}) = \{\epsilon\}
$$

$$
\text{paths}(p \rightarrow q : \{\ell_i, G_i \mid 1 \leq i \leq n\}) = \bigcup_{1 \leq i \leq n} \{p \rightarrow q \ell_i \rho \mid \rho \in \text{paths}(G_i)\}
$$
where $\epsilon$ is the empty sequence.
Note that every infinite path of a global type has infinitely many occurrences of $\to$.

### 3.2. Projection

The standard projection of global types onto participants produces session types and session types are assigned to processes by a type system \([1, 2]\). The present simplified shape of messages allows us to project global types onto participants producing processes.

The projection of a global type onto a participant $r$ gives the process that $r$ should run to follow the protocol specified by the global type. If the global type begins by establishing a communication from $p$ to $q$, then the projection onto $p$ should send one message to $q$, and the projection onto $q$ should receive one message from $p$. The projection onto a third participant $r$ skips the initial communication, that does not involve her. If there is only one choice, then it returns the process obtained from projecting the continuation onto $r$. Otherwise it merges the processes obtained from projecting the continuations onto $r$ only if these are all inputs of different messages from the same sender.

**Definition 3.4. (Projection)**

Given a global type $G$ and a participant $p$, we define the partial function $\mid_p$ coinductively as follows:

$$G \mid_p = 0 \text{ if } p \not\in \text{ppts}(G)$$

$$(p \to q : \{\ell_i.G_i \mid 1 \leq i \leq n\}) \mid_p = q!\{\ell_i.G_i \mid p \mid p \mid 1 \leq i \leq n\}$$

$$(q \to p : \{\ell_i.G_i \mid 1 \leq i \leq n\}) \mid_p = q?\{\ell_i.G_i \mid p \mid p \mid 1 \leq i \leq n\}$$

$$(q \to r : \{\ell_i.G_i \mid 1 \leq i \leq n\}) \mid_p = \begin{cases} G_1 \mid_p & \text{if } p \not\in \{q, r\} \text{ and } n = 1 \\ s?(\Lambda_1 \uplus \ldots \uplus \Lambda_n) & \text{if } p \not\in \{q, r\} \forall 1 \leq i \leq n : G_i \mid_p = s?\Lambda_i \\ & \text{and all messages in } \Lambda_1 \uplus \ldots \uplus \Lambda_n \\ & \text{are pairwise different} \end{cases}$$

We say that $G \mid_p$ is the projection of $G$ onto $p$ if $G \mid_p$ is defined. We say that $G$ is projectable if $G \mid_p$ is defined for all participants $p$.

If $p$ is not involved in the first communication of $G$ and $G$ starts with a choice between different messages, then in all branches the process of participant $p$ must be an input, so that $p$ can understand which branch was chosen.

It is important to define the last clause using disjoint unions instead of unions to guarantee that the projection is a (partial) function (see Example 3.8). The proof that the projection is actually a (partial) function is in Appendix A. Since the global types are regular, their projection is also regular.

Note that the processes $P_1$, $P_2$ and $P_3$ of Example 2.4 are the respective projections of $G_1$, $G_2$ and $G_3$ of Example 3.3 onto the participant $p_4$.

In the following examples, we assume that $p$, $q$, $r$ and $s$ are all different. Examples 3.6, 3.7, 3.8 and 3.9 disallow badly behaved multiparty sessions.

**Example 3.5.** Let $G = p \to q : \{\ell_1.G_1, \ell_2.G_2\}$, where $G_1 = q \to r : \ell_3$ and $G_2 = q \to r : \ell_4$. 
1. $G|_p = q!\{\ell_1, \ell_2\}$

2. $G|_q = p?\{\ell_1.G_1|_q, \ell_2.G_2|_q\} = p?\{\ell_1.r!\ell_3, \ell_2.r!\ell_4\}$

3. $G|_r = q?\{\ell_3, \ell_4\}$ because $G_1|_r = q?\ell_3$ and $G_2|_r = q?\ell_4$.

Notice that $G|_r$ is defined thanks to the fact that $r$ receives $\ell_3$ in one branch and $\ell_4$ in the other. Then, $G$ is projectable.

**Example 3.6.** If $G = p \rightarrow q : \{\ell_1.G_1, \ell_2\}$, where $G_1 = r \rightarrow q : \ell_3$, then $G$ is not projectable since

1. $G|_p = q!\{\ell_1, \ell_2\}$

2. $G|_q = p?\{\ell_1.G_1|_q, \ell_2\} = p?\{\ell_1.r?\ell_3, \ell_2\}$

3. $G|_r$ is not defined because $G_1|_r = q!\ell_3$ and $G_2|_r = q!\ell_4$.

If we allowed $G$ to be projectable by defining $G|_r = q!\ell_3$, then we would be able to type the session $M_1$ of Example 2.13 which is not deadlock-free.

**Example 3.7.** If $G = p \rightarrow q : \{\ell_1.G_1, \ell_2.G_2\}$, where $G_1 = r \rightarrow q : \ell_3$ and $G_2 = r \rightarrow q : \ell_4$, then $G$ is not projectable.

1. $G|_p = q!\{\ell_1, \ell_2\}$

2. $G|_q = p?\{\ell_1.G_1|_q, \ell_2.G_2|_q\} = p?\{\ell_1.r?\ell_3, \ell_2.r?\ell_4\}$

3. $G|_r$ is not defined because $G_1|_r = q!\ell_3$ and $G_2|_r = q!\ell_4$.

If we allowed $G$ to be projectable by defining $G|_r = q!\ell_3$, then we would be able to type the session $M_2$ of Example 2.13 which is not deadlock-free.

**Example 3.8.** If $G = p \rightarrow q : \{\ell_1.q \rightarrow r : \ell_3, \ell_2.G\}$, then $G$ is not projectable because

1. $G|_p = P$ where $P = q!\{\ell_1, \ell_2.P\}$

2. $G|_q = Q$ where $Q = p?\{\ell_1.r!\ell_3, \ell_2.Q\}$

3. $G|_r$ is not defined because $(q \rightarrow r : \ell_3)|_r = q?\ell_3$ and $G|_r = q?\ell_3$ and the last clause of Definition 3.4 makes use of disjoint unions and not unions. Using unions instead of disjoint unions, the projection of $G$ onto $r$ would not be unique. Indeed, $q?\{\ell_3 \uplus \Lambda\}$ would be a projection for any $\Lambda$. If we allowed $G$ to be projectable, then we would be able to type the multiparty session $M_1$ of Example 2.14 which is not strongly lock-free.

**Example 3.9.** If $G = p \rightarrow q : \{\ell_1.r \rightarrow p : \ell_3, \ell_2.G\}$, then $G$ is not projectable since

1. $G|_p = P$ where $P = q!\{\ell_1.r?\ell_3, \ell_2.P\}$
2. \( G|_q = Q \) where \( Q = p?\{\ell_1, \ell_2, Q\} \)

3. \( G|_r \) is not defined because \( r \to p : \ell_3|_r = p!\ell_3 \) and \( G|_r = p!\ell_3 \).

If we allowed \( G \) to be projectable, then we would be able to type the multiparty session \( M_2 \) of Example 2.14 which is not strongly lock-free.

An interesting property of projectable global types is that if \( p \in \text{ppts}(G) \), then \( p \) occurs in all paths of the tree representing \( G \). This is proved in Appendix A and allows to define the weight of a participant in a global type as follows.

**Definition 3.10. (Weight)**

Let \( \text{weight}(\rho_1 (q \to r) \ell \rho_2, p) = \text{length}(\rho_1) \) if \( p \notin \rho_1 \) and \( p \in \{q, r\} \), then

\[
\text{weight}(G, p) = \begin{cases} 
\max\{\text{weight}(\rho, p) \mid \rho \in \text{paths}(G)\} & \text{if } p \in \text{ppts}(G), \\
0 & \text{otherwise} 
\end{cases}
\]

As shown in Appendix A projectability assures finiteness of weight.

**Lemma 3.11. (Finite Weight)**

For all projectable global types \( G \) and all participants \( p \) we get that \( \text{weight}(G, p) \) is finite.

Note that \( \text{weight}(G, r) \) is not finite for the global types of Examples 3.8 and 3.9.

In the following

we will always assume that all global types are projectable

unless explicitly stated.

### 3.3. Structural Preorder and Typing Rule

The only typing rule for multiparty sessions uses a structural preorder on processes, which compares the shapes of processes. A process is smaller if it offers more messages as inputs and fewer messages as outputs.

**Definition 3.12. (Structural Preorder)**

We define the **structural preorder on processes**, \( P \leq Q \), by coinduction:

\[
\begin{align*}
\text{[SUB-0]} & \quad 0 \leq 0 \\
\text{[SUB-IN]} & \quad P_i \leq Q_i \quad \forall 1 \leq i \leq n \quad \Rightarrow \quad p?\{\ell_i, P_i \mid 1 \leq i \leq n\} \leq p?\{\ell_i, Q_i \mid 1 \leq i \leq n\} \\
\text{[SUB-OUT]} & \quad P_i \leq Q_i \quad \forall 1 \leq i \leq n \quad \Rightarrow \quad p!\{\ell_i, P_i \mid 1 \leq i \leq n\} \leq p!\{\ell_i, Q_i \mid 1 \leq i \leq n\} \cup \Lambda 
\end{align*}
\]
The double line in rules indicates that the rules are interpreted coinductively \cite{9} (Chapter 21). Rule [SUB-IN] allows bigger processes to offer fewer inputs while rule [SUB-OUT] allows them to select more outputs. The regularity condition on processes is crucial to guarantee the termination of the algorithm for checking structural preorder (given in Appendix B).

We are now able to formulate the typing system for multiparty sessions which contains only one rule. The typing judgments associate global types to sessions: they are of the shape $\vdash \mathcal{M} : G$.

**Definition 3.13. (Typing system)**

$$\forall i \in I \quad P_i \leq G|_{p_i} \quad \text{ppts}(G) \subseteq \{p_i \mid i \in I\}$$

\[ \vdash \prod_{i \in I} p_i \triangleright P_i : G \]  

\[ \text{[T-SESS]} \]

This rule requires that the processes in parallel can play as participants of a whole communication protocol or they are the terminated process, i.e. they are smaller or equal (according to the structural preorder) to the projections of a unique global type. The condition $\text{ppts}(G) \subseteq \{p_i \mid i \in I\}$ allows to type also sessions containing $p \triangleright 0$, a property needed to assure invariance of types under structural congruence. Notice that this typing rule allows to type multiparty session only with global types which can be projected on all their participants.

We say that the session $\mathcal{M}$ is well typed if there exists $G$ such that $\vdash \mathcal{M} : G$.

### 3.4. LTS of Global Types

Global types are behavioural types and therefore it is meaningful to define a LTS for them which mimics the LTS of sessions \cite{1,2}. We use the following notion of evaluation context, which only allows message exchanges without choices:

\[ \mathcal{E}[\cdot] ::= \text{inductive} \ [\cdot] \mid p \rightarrow q : \ell.\mathcal{E}[\cdot] \]

**Definition 3.14.** The labelled transition system (LTS) for global types (notation by $\xrightarrow{\Delta}$) is the smallest relation closed under the following rules:

\[ \begin{align*}
\text{[R-GCOMM]} & \quad \mathcal{E}[p \rightarrow q : (\ell.G' \uplus \Gamma)] \xrightarrow{\{p\ell\}\cup\Delta} \mathcal{E}[G'] \\
\text{[R-GCOMP]} & \quad \mathcal{E}[p \rightarrow q : \ell.\mathcal{E}'[G]] \xrightarrow{\Delta} \mathcal{E}[p \rightarrow q : \ell.\mathcal{E}'[G']] \\
\end{align*} \]

These rules allow disjoint participants to communicate simultaneously if only the innermost ones make a choice.

We write $G \xrightarrow{p\ell\ell'} G'$ as shorthand for $G \xrightarrow{\{p\ell\}} G'$. We sometimes omit the label. Then, $\rightarrow$ denotes $\xrightarrow{\Delta}$ for some $\Delta$ and $\rightarrow^*$ denotes the reflexive and transitive closure of $\rightarrow$.

It is easy to verify that if $G \xrightarrow{\Delta} G'$, then $\Delta$ is coherent (see Definition \cite{2,7}).

**Definition 3.15. (Redexes of Global Types and Sets of Redexes)**

1. If $G = \mathcal{E}[p \rightarrow q : (\ell.G' \uplus \Gamma)]$ and $p, q \notin \mathcal{E}[\cdot]$, then $p\ell q$ is a redex of $G$. If $p\ell q$ is a redex of $G$, then the participants $p, q$ are active in $G$. 

2. The set $\Delta$ is complete for $G$ if $\Delta$ contains (i) only redexes that occur in $G$ and (ii) all the active participants of $G$.

In Example 3.5 the set of redexes of $G$ is $\{p\ell_1q, p\ell_2q\}$ and $p, q$ are active participants, while $r$ is not active.

**Example 3.16.** A global type can reduce in different ways.

Let $G = p \rightarrow q : \ell_1, G'$ where $G' = r \rightarrow s : \{\ell_2, \ell_3, G'\}$. Then $G$ can reduce in 5 different ways:

$$G \xrightarrow{p\ell_1q} \xrightarrow{r\ell_2s} \xrightarrow{r\ell_3s} \xrightarrow{\{p\ell_1q,r\ell_2s\}} \xrightarrow{\{p\ell_1q,\ell_3\}} \xrightarrow{\text{end}} G'$$

The session $M$ of Example 2.9 has type $G$ and each transition on $M$ has a corresponding transition on $G$ and vice versa.

The following lemma is for global types the analogous of Lemma 2.10 for sessions and it can be proved by induction on the cardinality of $\Delta$.

**Lemma 3.17.** Let $\Delta = \{p_i\ell_iq_i \mid 1 \leq i \leq n\}$. Then $G \xrightarrow{\Delta} G'$ if and only if the following two conditions hold:

1. $\Delta$ is coherent and
2. $G = G_1$ and $G_n = G'$ and $G_i \xrightarrow{p_i\ell_iq_i} G_{i+1}$ for $1 \leq i \leq n-1$.

Similarly to Lemma 2.10, Lemma 3.17 does not hold without condition (1). For example, consider $G = p \rightarrow q : \ell_1, q \rightarrow p : \ell_2$. Then,

$$G \xrightarrow{p\ell_1q} q \rightarrow p : \ell_2 \xrightarrow{q\ell_2p} \text{end}$$

but there is no $G'$ such that $G \xrightarrow{\{p\ell_1q, q\ell_2p\}} G'$. Condition (1) is not satisfied, since $\{p\ell_1q, q\ell_2p\}$ is not coherent.

Now, consider $G = p \rightarrow q : \{\ell_1, G, \ell_2, G\}$. We have that

$$G \xrightarrow{p\ell_1q} G \xrightarrow{p\ell_2q} G$$

but there is no $G'$ such that $G \xrightarrow{\{p\ell_1q, p\ell_2q\}} G'$. Both $p\ell_1q$ and $p\ell_2q$ are redexes of $G$, but condition (1) is not satisfied, since $\{p\ell_1q, p\ell_2q\}$ is not coherent.
4. Properties of Well-Typed Sessions

In this section, we prove that the proposed typing system for multiparty sessions (Definition 3.13) enjoys good properties: subject reduction, session fidelity, progress, and strong lock-freedom.

4.1. Subject Reduction and Session Fidelity

We start with the classical lemmas of inversion and canonical form, which easily follow from the typing rule \[T\text{-SESS}\].

**Lemma 4.1. (Inversion Lemma)**
If \(\vdash \prod_{i \in I} p_i \triangleright P_i : G\), then for all \(i \in I\): \(P_i \triangleq G|_{p_i}\) and \(pptns(G) \subseteq \{p_i | i \in I\}\).

**Lemma 4.2. (Canonical Form Lemma)**
If \(\vdash M : G\) and \(pptns(G) = \{p_i | i \in I\}\), then \(M \equiv \prod_{i \in I} p_i \triangleright P_i\) and for all \(i \in I\): \(P_i \triangleq G|_{p_i}\).

The following lemma allows us to guess the shape of a global type from the shapes of its projections.

**Lemma 4.3. (Key Lemma)**
If \(G|_p = q!\Lambda\) and \(G|_q = p?\Lambda'\), then \(G = E[p \rightarrow q : \Gamma]\) and \(msg(\Gamma) = msg(\Lambda) = msg(\Lambda')\) and \(p, q\) do not occur in \(E[\cdot]\).

**Proof:**
The proof is by induction on \(n = weight(G, p)\).

- If \(n = 0\), then \(G = p \rightarrow q : \Gamma\) by definition of projection.
- If \(n > 0\), then \(G = r \rightarrow s : \ell.G'\) and \(\{r, s\} \cap \{p, q\} = \emptyset\) by definition of projection. By induction hypothesis, \(G' = E[p \rightarrow q : \Gamma]\), and \(p, q\) do not occur in \(E[\cdot]\) and \(msg(\Gamma) = msg(\Lambda) = msg(\Lambda')\). □

Subject Reduction says that the transitions of well-typed sessions are mimicked by the transitions of global types.

**Theorem 4.4. (Subject Reduction)**
If \(\vdash \mathcal{M} : G\) and \(\mathcal{M} \overset{\ell}{\rightarrow} \mathcal{M}'\), then \(G \overset{\ell}{\rightarrow} G'\) and \(\vdash \mathcal{M}' : G'\).

**Proof:**
By Lemmas \ref{lem:weight} and \ref{lem:typing}, it is enough to prove that if \(\mathcal{M} \overset{p}{\rightarrow} \mathcal{M}'\), then \(G \overset{p}{\rightarrow} G'\) and \(\vdash \mathcal{M}' : G'\). If \(\mathcal{M} \overset{p}{\rightarrow} \mathcal{M}'\), then

\[
\mathcal{M} \equiv p \triangleright q!(\ell. P \uplus \Lambda) \mid q \triangleright p?(\ell. Q \uplus \Lambda') \mid \prod_{1 \leq j \leq m} r_j \triangleright R_j \quad \mathcal{M}' \equiv p \triangleright P \mid q \triangleright Q \mid \prod_{1 \leq j \leq m} r_j \triangleright R_j
\]

and \(msg(\Lambda) \subseteq msg(\Lambda')\). Since \(\vdash \mathcal{M} : G\), by Lemma \ref{lem:typing} we have that...
By definition of $\leq$ from $q!(\ell.P \uplus \Lambda) \leq G|_p$ we get $G|_p = q!(\ell.P_0 \uplus \Lambda_0)$ and $P \leq P_0$. Similarly from $p?(\ell.Q \uplus \Lambda') \leq G|_q$ we get $G|_q = p?(\ell.Q_0 \uplus \Lambda'_0)$ and $Q \leq Q_0$. Lemma 4.3 implies
\[ G = \mathcal{E}[p \to q : (\ell.G_0 \uplus \Gamma)] \]
and $p$, $q$ do not occur in $\mathcal{E}[:].$ We can take $G' = \mathcal{E}[G_0]$. By definition $G \xrightarrow{p|q} G'$. Moreover we get $G'|_p = P_0$, $G'|_q = Q_0$ and $G'|_r = G|_r$ for $1 \leq j \leq m$. By applying rule [T-SESS], we deduce $\vdash \mathcal{M}': G'$.

Session fidelity assures that the communications in a session typed by a global type are done as prescribed by the global type, but for the choices of messages. To formalise this we say that a set of redexes $\Delta$ is a relabelling of a set of redexes $\Delta'$ if the redexes in $\Delta$ and $\Delta'$ only differ by the messages.

**Theorem 4.5. (Session Fidelity)**

Let $\vdash \mathcal{M} : G$.

1. If $\mathcal{M} \xrightarrow{\Delta} \mathcal{M}'$, then $G \xrightarrow{\Delta} G'$ and $\vdash \mathcal{M}' : G'$.

2. If $G \xrightarrow{\Delta} G'$, then there is a relabelling $\Delta'$ of $\Delta$ such that $\mathcal{M} \xrightarrow{\Delta'} \mathcal{M}'$.

**Proof:**

(Item 1). It is the Subject Reduction Theorem.

(Item 2). By Lemmas 3.17 and 2.10, it is enough to prove that if $G \xrightarrow{p|q} G'$, then $\mathcal{M} \xrightarrow{p'|q'} \mathcal{M}'$ for some $p'$. If $G \xrightarrow{p|q} G'$, then $G = \mathcal{E}[p \to q : (\ell.G_0 \uplus \Gamma)]$ and $G' = \mathcal{E}[G_0]$, where $p$, $q$ do not occur in $\mathcal{E}[:].$

By Lemma 4.2, $\mathcal{M} \equiv p \triangleright P \mid q \triangleright Q \mid \mathcal{M}_0$ and $P \leq G|_p$ and $Q \leq G|_q$. By definition of projection $G|_p = q!(\ell.P_0 \uplus \Lambda)$ and $G|_q = p?(\ell.Q_0 \uplus \Lambda')$ with $\text{msg}(\Lambda) = \text{msg}(\Lambda') = \text{msg}(\Gamma)$. By definition of $\leq$ we get $P = q!\Lambda_1$ with $\text{msg}(\Lambda_1) \subseteq \{\ell\} \cup \text{msg}(\Lambda)$ and $Q_0 = p?\Lambda_2$ with $\text{msg}(\Lambda_2) \supseteq \{\ell\} \cup \text{msg}(\Lambda)$. If $\ell \in \text{msg}(\Lambda_1)$, then we take $\ell' = \ell$, otherwise we choose some $\ell' \in \text{msg}(\Lambda_1)$. In both cases

\[ \mathcal{M} \equiv p \triangleright P \mid q \triangleright Q \mid \mathcal{M}_0 \xrightarrow{p'|q'} p \triangleright P' \mid q \triangleright Q' \mid \mathcal{M}_0 \]

for some $P'$, $Q'$.

4.2. **Strong Lock-Freedom**

In this section, we show that the type system of Section 3 enjoys the property of strong lock-freedom. By Subject Reduction it is enough to prove that all well-typed sessions are deadlock-free and no participant waits forever. The former follows from Session Fidelity, while the latter follows from the following lemma that says that:

1. the weight of the participants not involved in the transition does not increase, and
2. the weight of the participants not involved in the transition strictly decreases when the reduction involves all possible active participants.

Lemma 4.6. (Weight Decrease by Complete Reduction)
Let \( G \rightarrow G' \) and \( p \notin \Delta \). We get

1. \( \text{weight}(G, p) \geq \text{weight}(G', p) \).

2. If \( \Delta \) is complete for \( G \), then \( \text{weight}(G, p) > \text{weight}(G', p) \).

Proof:
(Item 1). By Lemma 3.17 it is enough to consider \( \Delta = \{ q \ell r \} \) with \( p \notin \{ q, r \} \). In this case

\[ G = \mathcal{E}[q \rightarrow r : \{ \ell_i.G_0 \cup \Gamma \}] \text{ and } G' = \mathcal{E}[G_0] \]

Let \( \rho(q \rightarrow r) \ell \rho' \) be a path in \( G \). The first occurrence of \( p \) in that path is either in \( \rho \) or \( \rho' \). In the first case \( \text{weight}(G, p) = \text{weight}(\mathcal{E}[G_0], p) = \text{weight}(G', p) \). In the second case \( \text{weight}(G, p) > \text{weight}(\mathcal{E}[G_0], p) = \text{weight}(G', p) \).

(Item 2). Let \( G = q \rightarrow r : \{ \ell_i.G_i \mid 1 \leq i \leq n \} \). Since \( \Delta \) is complete for \( G \), we have that \( \Delta = \{ q \ell_i r \} \cup \Delta' \) for some \( i \) (1 \( \leq i \leq n \)) and some \( \Delta' \). By definition of reduction \( G \rightarrow G' \) implies

\[ G \xrightarrow{q \ell_i r} G_i \xrightarrow{\Delta'} G' \]

By definition each path of \( G \) is of the form \( (q \rightarrow r) \ell_i \rho_i \), where \( \rho_i \) is a path of \( G_i \) for 1 \( \leq i \leq n \).

This gives \( \text{weight}(G, p) > \text{weight}(G_i, p) \) for 1 \( \leq i \leq n \). Moreover by Item 1 \( \text{weight}(G_i, p) \geq \text{weight}(G', p) \) for 1 \( \leq i \leq n \).

\[ \Box \]

Theorem 4.7. (Strong Lock-Freedom)
If session \( \mathcal{M} \) is well typed, then \( \mathcal{M} \) is strongly lock-free.

Proof:
Applying Theorem 4.4 it is enough to prove that there is no well-typed \( \mathcal{M} \) such that \( \text{stuck}(\mathcal{M}, p) \) or \( \text{wait}_\infty(\mathcal{M}, p) \) for some \( p \in \text{ppts}(\mathcal{M}) \). Assume towards a contradiction that \( \vdash \mathcal{M} : G \) and either \( \text{stuck}(\mathcal{M}) \) or \( \text{wait}_\infty(\mathcal{M}, p) \) for some \( p \in \text{ppts}(\mathcal{M}) \).

Suppose first that \( \text{stuck}(\mathcal{M}) \). Since by definition \( \mathcal{M} \neq p \rightarrow 0 \), then \( \mathcal{M} \neq \text{end} \). Let \( G = q \rightarrow r : \Gamma \).

By definition of reduction \( G \rightarrow G' \) for some \( \ell \), and this implies \( \vdash \mathcal{M} \rightarrow \mathcal{M}' \) for some \( \ell' \) by Theorem 4.5.2.

Suppose now that \( \text{wait}_\infty(\mathcal{M}, p) \) for some \( p \in \text{ppts}(\mathcal{M}) \). Hence, there is an infinite reduction sequence

\[ \mathcal{M} = \vdash p \rightarrow P | \mathcal{M}_0 \xrightarrow{\Delta_0} p \rightarrow P | \mathcal{M}_1 \xrightarrow{\Delta_1} p \rightarrow P | \mathcal{M}_2 \xrightarrow{\Delta_2} \ldots \]

such that \( p \) is not in \( \Delta_i \) and \( \Delta_i \) is complete for \( p \rightarrow P \) for all \( i \geq 0 \). Theorem 4.4 implies \( \vdash p \rightarrow P | \mathcal{M}_i : G_i \) with

\[ G_0 \xrightarrow{\Delta_0} G_1 \xrightarrow{\Delta_1} G_2 \xrightarrow{\Delta_2} \ldots \]
By Lemma 4.6, there is an infinite decreasing sequence

\[ \text{weight}(G_0, p) > \text{weight}(G_1, p) > \text{weight}(G_2, p) > \ldots \]

This contradicts the fact that \( \text{weight}(G_0, p) \) is finite by Lemma 3.11.

The converse of Theorem 4.7 does not hold. For example

\[ p \triangleright q\{\ell_1, r?\ell_3, \ell_2, r?\ell_3\} \mid q \triangleright p\{\ell_1, \ell_2\} \mid r \triangleright p!\ell_3 \]

reduces in 2 ways to \( p \triangleright 0 \), then it is strongly lock-free, but it is not typable, since the global type \( p \rightarrow q : {\ell_1, r \rightarrow \ell_3, \ell_2, r \rightarrow \ell_3} \) cannot be projected onto \( r \).

The notion of progress defined in [14] is stronger than the notion of deadlock-freedom. In our setting progress can be formulated as the following corollary and it is a consequence of strong lock-freedom.

**Corollary 4.8. (Progress)**

Let \( p \triangleright P \mid M \) be a well typed session.

1. If \( P = q\{\ell_i.P_i \mid 1 \leq i \leq n\} \), then \( p \triangleright P \mid M \rightarrow^* p \triangleright P_i \mid M' \) for all \( i (1 \leq i \leq n) \).

2. If \( P = q\{\ell_i.P_i \mid 1 \leq i \leq n\} \), then \( p \triangleright P \mid M \rightarrow^* p \triangleright P_i \mid M' \) for some \( i (1 \leq i \leq n) \).

## 5. Characterisation of Observational Equivalence

This section gives the main result of the paper: the characterisation of observational equivalence. This is done by showing that the observational and the structural preorders on processes coincide. We split the proof showing first soundness, i.e. \( P \equiv Q \) implies \( P \sqsubseteq Q \), and then completeness, i.e. \( P \sqsubseteq Q \) implies \( P \equiv Q \).

### 5.1. Soundness

To show that \( \sqsubseteq \) is included in \( \sqsubseteq \) we use two lemmas which relate \( \sqsubseteq \) with session transitions, deadlock and starvation.

**Lemma 5.1.** Let \( P \sqsubseteq Q \).

1. If \( p \triangleright P \mid M \xrightarrow{\mathcal{A}} p \triangleright P' \mid M' \) then
   
   (a) if \( p \notin \Delta \), then \( P' = P \) and \( p \triangleright Q \mid M \xrightarrow{\mathcal{A}} p \triangleright Q \mid M' \).

   (b) if \( p \in \Delta \), then either \( p \triangleright Q \mid M \xrightarrow{\mathcal{A}} p \triangleright Q' \mid M' \) with \( P' \equiv Q' \) or \( \text{stuck}(p \triangleright Q \mid M) \) or \( \text{wait}_{\infty}(p, p \triangleright Q \mid M) \).

2. If \( p \triangleright Q \mid M \xrightarrow{\mathcal{A}} p \triangleright Q' \mid M' \), then \( p \triangleright P \mid M \xrightarrow{\mathcal{A}} p \triangleright P' \mid M' \) with \( P' \equiv Q' \).
3. If $\Delta$ is complete for $p \triangleright P \mid M$ and $p \not\in \Delta$, then $\Delta$ is complete for $p \triangleright Q \mid M$.

**Proof:**

By Lemma 2.10 it is enough to prove the first two items for $\Delta = \{ r, s \}$.

*(Item 1a)* Suppose $p \not\in \{ r, s \}$. Then the conclusion follows since $M \xrightarrow{\Delta} M'$ and $P' = P$.

*(Item 1b)* Suppose $\{ r, s \} = \{ p, q \}$ and $P = q!(\ell.P' \uplus \Lambda)$. Then $\Delta = \{ p\ell q \}$ and

$$M = q \triangleright p?(\ell.R \uplus \Lambda') \mid M_0 \quad \quad M' = q \triangleright R \mid M_0 \quad \quad \text{msg}(\Lambda) \subseteq \text{msg}(\Lambda')$$

From $P \leq Q$, there are two possibilities:

1. $Q = q!(\ell.Q' \uplus \Lambda''')$ with $P' \leq Q'$ and $\text{msg}(\Lambda''') \subseteq \text{msg}(\Lambda')$. Then $p \triangleright Q \mid M \xrightarrow{p\ell q} p \triangleright Q' \mid M'$.

2. If $Q = q!\Lambda''$ and $\text{msg}(\Lambda'') \not\subseteq \{ \ell \} \cup \text{msg}(\Lambda')$. There are two possibilities, either $\text{stuck}(p \triangleright Q \mid M)$ or $\text{wait}_\infty(p, p \triangleright Q \mid M)$.

If $P$ is an input process the proof is similar.

*(Item 2)* If $p \not\in \{ r, s \}$, then the conclusion is immediate, since $M \xrightarrow{\Delta} M'$ and $P' = P$. Otherwise $\{ r, s \} \neq \{ p, q \}$. Suppose $Q = q!(\ell.Q' \uplus \Lambda)$. Then $\Delta = \{ p\ell q \}$ and

$$M = q \triangleright p?(\ell.R \uplus \Lambda') \mid M_0 \quad \quad M' = q \triangleright R \mid M_0 \quad \quad \text{msg}(\Lambda) \subseteq \text{msg}(\Lambda')$$

Since $P \leq Q$, $P = q!(\ell.P' \uplus \Lambda''')$ with $P' \leq Q'$ and $\text{msg}(\Lambda''') \subseteq \text{msg}(\Lambda)$. Hence we get

$p \triangleright P \mid M \xrightarrow{p\ell q} p \triangleright P' \mid M'$.

*(Item 3)* If $r, s \in \Delta$, then $r, s$ is a redex of $p \triangleright Q \mid M$, because $r, s$ is a redex of $p \triangleright P \mid M$ and $p \not\in \{ r, s \}$. It follows from Item 2 that if the participant $q$ is active in $p \triangleright Q \mid M$, then $q$ is also active in $p \triangleright P \mid M$, and since $\Delta$ is complete for $p \triangleright P \mid M$, it contains $q$.

$\square$

**Lemma 5.2.** Let $P \leq Q$.

1. If $\text{wait}_\infty(p, p \triangleright P \mid M)$, then $\text{wait}_\infty(p, p \triangleright Q \mid M)$.

2. If $\text{wait}_\infty(r, p \triangleright P \mid M)$, then $p \triangleright Q \mid M \xrightarrow{*} M'$ and either $\text{stuck}(M')$ or $\text{wait}_\infty(r, M')$ or $\text{wait}_\infty(p, M')$.

**Proof:**

*(Item 1)* We have that

$$p \triangleright P \mid M_0 \xrightarrow{\Delta_0} p \triangleright P \mid M_1 \xrightarrow{\Delta_1} p \triangleright P \mid M_2 \xrightarrow{\Delta_2} \ldots$$

where $M_0 = M$ and $\Delta_j$ is complete for $p \triangleright P \mid M_j$ and $p \not\in \Delta_j$ for all $j \geq 0$. It follows from Lemma 5.1.1a that

$$p \triangleright Q \mid M_0 \xrightarrow{\Delta_0} p \triangleright Q \mid M_1 \xrightarrow{\Delta_1} p \triangleright Q \mid M_2 \xrightarrow{\Delta_2} \ldots$$

By Lemma 5.1.1b $\Delta_j$ is complete for $p \triangleright Q \mid M_j$ for all $j \geq 0$. Hence $\text{wait}_\infty(p, p \triangleright Q \mid M)$. 

(Item 2) We have that
\[ p \triangleright P \mid r \triangleright R \mid M_0 \xrightarrow{\Delta_0} p \triangleright P_1 \mid r \triangleright R \mid M_1 \xrightarrow{\Delta_1} p \triangleright P_2 \mid r \triangleright R \mid M_2 \xrightarrow{\Delta_2} \ldots \]
where \( P_0 = P \) and \( M_0 = M \) and \( \Delta_j \) is complete for \( p \triangleright P_j \mid r \triangleright R \mid M_j \) and \( r \notin \Delta_j \) for all \( j \geq 0 \).

By Lemma 5.1(1b) there are two possibilities:

1. we can replicate the same infinite reduction for \( p \triangleright Q \mid r \triangleright R \mid M_0 \)

\[ p \triangleright Q_0 \mid r \triangleright R \mid M_0 \xrightarrow{\Delta_0} p \triangleright Q_1 \mid r \triangleright R \mid M_1 \xrightarrow{\Delta_1} p \triangleright Q_2 \mid r \triangleright R \mid M_2 \xrightarrow{\Delta_2} \ldots \]

where \( Q_0 = Q \) and \( r \notin \Delta_j \) for all \( j \geq 0 \). Lemma 5.1(3) implies that \( \Delta_j \) is complete for \( p \triangleright Q_j \mid r \triangleright R \mid M_j \) for all \( j \geq 0 \). Hence \( \text{wait}_\infty(r, p \triangleright Q \mid M) \).

2. there exists \( k \) such that

\[ p \triangleright Q_0 \mid r \triangleright R \mid M_0 \xrightarrow{\Delta_0} p \triangleright Q_1 \mid r \triangleright R \mid M_1 \xrightarrow{\Delta_1} \ldots \xrightarrow{\Delta_{k-1}} p \triangleright Q_k \mid r \triangleright R \mid M_k \]

and either \( \text{stuck}(p \triangleright Q_k \mid r \triangleright R \mid M_k) \) or \( \text{wait}_\infty(p, p \triangleright Q_k \mid r \triangleright R \mid M_k) \).

\[ \Box \]

Theorem 5.3. (Inclusion of Structural Preorder in Observational Preorder)

If \( P \preceq Q \), then \( P \subseteq Q \).

Proof:

Let \( P \preceq Q \) and \( \text{locked}(p \triangleright P \mid M) \). By Definition 2.11 we have \( p \triangleright P \mid M \rightarrow^* p \triangleright P' \mid M' \) and either \( \text{stuck}(p \triangleright P' \mid M') \) or \( \text{wait}_\infty(q, p \triangleright P' \mid M') \) for some \( q \). We get either \( p \triangleright Q \mid M \rightarrow^* p \triangleright Q' \mid M' \) with \( P' \preceq Q' \) or \( \text{locked}(p \triangleright Q \mid M) \) by Lemma 5.1(1). In the first case the contrapositive of Lemma 5.1(2) implies \( \text{stuck}(p \triangleright Q' \mid M') \) and Lemma 5.2 implies one of the following: \( \text{stuck}(p \triangleright Q' \mid M') \) or \( \text{wait}_\infty(p, p \triangleright Q' \mid M') \) or \( \text{wait}_\infty(q, p \triangleright Q' \mid M') \) for some \( q \). So in both cases we conclude \( \text{locked}(p \triangleright Q \mid M) \).

\[ \Box \]

5.2. Completeness

To show the reverse direction of the equivalence we proceed in three steps:

- we define the negation \( \not\preceq \) of the structural preorder;
- we build the characteristic global type \( G(P, p) \) of process \( P \) for a participant \( p \) that does not occur in \( P \);
- we show that if \( Q \not\preceq P \), then \( \text{locked}(C_P[p \triangleright Q]) \) and \( \text{lockfree}(C_P[p \triangleright P]) \) for

\[ C_P[\cdot] = [\cdot] \prod_{1 \leq i \leq n} p_i \triangleright G(P, p)|_{p_i} \]

where \( \{p_i \mid 1 \leq i \leq n\} = \text{ppts}(P) \).
The following definition gives the negation of the structural preorder. These rules say that a process different from 0 cannot be compared to 0, two input or output processes with different participants, or different messages, or continuations which do not match, cannot be compared. The rules using the mapping \( \text{msg} \) take into account the preorder on external and internal choices. Note that the rules are given inductively and not coinductively. The coinductive version of \( \not\preceq \) would include the equality between processes which is not in the negation of \( \preceq \). One can show that either \( P \preceq Q \) or \( P \not\preceq Q \) holds for two arbitrary processes \( P, Q \). Appendix C contains this proof.

**Definition 5.4. (Negation of structural preorder)**

We define \( \not\preceq \) by induction as follows.

\[
\begin{align*}
\text{[NSUB-ENDL]} & \quad P \not\preceq 0 \\
\text{[NSUB-ENDR]} & \quad P \not\preceq 0 \\
\text{[NSUB-IN-OUT]} & \quad p \not\preceq q \\
\text{[NSUB-DIFF-IN]} & \quad \text{msg}(\Lambda') \not\subseteq \text{msg}(\Lambda) \\
\text{[NSUB-LAB-IN]} & \quad p \not\preceq p' \not\preceq \Lambda \\
\text{[NSUB-OUT-IN]} & \quad p \not\preceq q \\
\text{[NSUB-DIFF-OUT]} & \quad \text{msg}(\Lambda) \not\subseteq \text{msg}(\Lambda') \\
\text{[NSUB-LAB-OUT]} & \quad p \not\preceq p' \not\preceq \Lambda' \\
\text{[NSUB-CONT-IN]} & \quad P \not\preceq Q \\
\text{[NSUB-CONT-OUT]} & \quad P \not\preceq Q \\
\end{align*}
\]

The characteristic global type \( \mathcal{G}(P, p) \) of the type \( P \) for the participant \( p \) describes the communications between \( p \) and all participants in \( \text{ppts}(P) \) following \( P \). Moreover after each communication involving \( p \) and some \( q \in \text{ppts}(P) \), participant \( q \) spreads the exchanged message to all other participants in \( \text{ppts}(P) \). This is needed for getting both a projectable global type and a stuck session, see the proof of Theorem 5.8 and Examples 5.7 and 5.9.

**Definition 5.5. (Characteristic Global Type)**

Let \( p \not\in \text{ppts}(P) \). We define the characteristic global type of \( P \) for \( p \) as

\[
\mathcal{G}(P, p) = \mathcal{G}_{\text{ppts}(P)}(P, p)
\]

where \( \mathcal{G}_X(P, p) \) is given by:

\[
\begin{align*}
\mathcal{G}_X(0, p) & = \text{end} \\
\mathcal{G}_X(q?\{\ell_i, P_i \mid 1 \leq i \leq n\}, p) & = \begin{cases} 
q \to p.\{\ell_i, \mathcal{G}_X(P_i, p) \mid 1 \leq i \leq n\} & \text{if } X = \{q\}, \\
q \to p.\{\ell_i, \text{spread}_X(q, \ell_i).\mathcal{G}_X(P_i, p) \mid 1 \leq i \leq n\} & \text{otherwise}
\end{cases} \\
\mathcal{G}_X(q!\{\ell_i, P_i \mid 1 \leq i \leq n\}, p) & = \begin{cases} 
p \to q.\{\ell_i, \mathcal{G}_X(P_i, p) \mid 1 \leq i \leq n\} & \text{if } X = \{q\}, \\
p \to q.\{\ell_i, \text{spread}_X(q, \ell_i).\mathcal{G}_X(P_i, p) \mid 1 \leq i \leq n\} & \text{otherwise}
\end{cases}
\end{align*}
\]

and \( \text{spread}_X(q, \ell) \), a prefix for sending \( \ell \) to all participants in \( X \setminus \{q\} = \{p_1, \ldots, p_m\} \), is:

\[
\text{spread}_X(q, \ell) = q \to p_1 : \ell. \cdots q \to p_m : \ell
\]
The message spread has two purposes. The first one is to guarantee that the projections are defined on all participants (see Example 5.7). The second one is to force an order on the communications in which p and q first exchange a message and then the remaining participants receive this message from q (see Example 5.9).

The following lemma shows the soundness of previous definition.

**Lemma 5.6.** \(G(P, p)\) is projectable and \(G(P, p)|_p = P\).

**Proof:**
The proof is by coinduction. Let \(X \supseteq \text{ppts}(P)\). We only consider the case in which \(X\) contains more that one participant and \(P = q?!\{\ell_i.P_i \mid 1 \leq i \leq n\}\). Then

\[
G_X(P, p) = q \rightarrow p.\{\ell_i.\text{spread}_X(q, \ell_i).G_X(P_i, p) \mid 1 \leq i \leq n\}
\]

Since \(p\) does not occur in \(\text{spread}_X(q, \ell_i)\), we have that

\[
G_X(P, p)|_p = q?!\{\ell_i.G_X(P_i, p)|_p \mid 1 \leq i \leq n\}
\]

By coinduction hypothesis \(G_X(P_i, p)|_p = P_i\).

Let \(X \setminus \{q\} = \{p_1, \ldots, p_m\}\). Then,

\[
G_X(P, p)|_q = p?!\{\ell_i.p_1!\ell_i \ldots p_m!\ell_i.G_X(P_i, p)|_q \mid 1 \leq i \leq n\}
\]  (1)

By coinduction hypothesis \(G_X(P_i, p)|_q\) is defined.

Lastly let \(r = p_i\) for some \(i\) (1 \(\leq i \leq m\)), then

\[
G_X(P, p)|_r = q?!\{\ell_i.G_X(P_i, p)|_r \mid 1 \leq i \leq n\}
\]  (2)

By coinduction hypothesis \(G_X(P_i, p)|_r\) is defined.

\(\Box\)

**Example 5.7.** The characteristic global types are projectable thanks to the message spread (see Lemma 5.6). Take for example \(P = q?!\{\ell_1.r?\ell_2, \ell_3\}\). Without the message spread we would get the global type

\[
G = p \rightarrow q : \{\ell_1.r \rightarrow p : \ell_2, \ell_3\}
\]

and \(G|_r\) is undefined. Instead

\[
G(P, p) = p \rightarrow q : \{\ell_1.q \rightarrow r : \ell_1.r \rightarrow p : \ell_2.r \rightarrow q : \ell_2, \ell_3.q \rightarrow r : \ell_3\}
\]

\[
G(P, p)|_r = q?!\{\ell_1.p!\ell_2.q!\ell_2, \ell_3\}
\]

**Theorem 5.8. (Inclusion of Observational Preorder in Structural Preorder)**
If \(P \sqsubseteq Q\), then \(P \leq Q\).
Proof:
Being \( \not\prec \) the negation of \( \leq \), it is enough to prove that \( P \not\prec Q \) implies \( P \not\subseteq Q \). Let \( P \not\prec Q \) and \( p \) be fresh and \( ppts(Q) = \{ p_i \mid 1 \leq i \leq n \} \). Lemma 5.6 allows us to define the context:

\[
C_Q[\cdot] = [\cdot] | \prod_{1 \leq i \leq n} p_i \triangleright G(Q, p)|_{p_i}
\]

By Lemma 5.6, \( C_Q[p \triangleright Q] : G(Q, p) \). It follows from Theorem 4.7 that locked \( (C_Q[p \triangleright Q]) \).

We will now show that locked \( (C_Q[p \triangleright P]) \) by induction on \( \not\prec \) and by cases on the last rule applied. Note that any reduction sequence starting from \( C_Q[p \triangleright P] \) will be of the form:

\[
C_Q[p \triangleright P] \xrightarrow{\Delta_0} M_0 \xrightarrow{\Delta_1} M_1 \xrightarrow{\Delta_2} \ldots \xrightarrow{\Delta_k} M_k
\]

when finite, or

\[
C_Q[p \triangleright P] \xrightarrow{\Delta_0} M_0 \xrightarrow{\Delta_1} M_1 \xrightarrow{\ldots} \xrightarrow{\Delta_k} M_k
\]

when infinite, where all \( \Delta_j \) for \( j \geq 0 \) contains only one redex by construction. The fresh participant \( p \) should be in \( \Delta_0 \) because there is only one participant \( q \) in \( C_Q[p \triangleright P] \) that can communicate with \( p \). In fact all other participants are waiting for a message from \( q \). We can assume \( q = p_1 \). All other participants \( p_i \neq q \) for \( 2 \leq i \leq n \) will be able to reduce only after a message between \( p \) and \( q \) has been exchanged, and then they will receive that message from \( q \), see the projections (1) and (2) in the proof of Lemma 5.6.

We will only consider some interesting cases.

Case [NSUB-IN-OUT]. Let \( P = r?\Lambda \) and \( Q = q!\Lambda' \). By construction \( G(Q, p)|_q = p?\Lambda'' \), then

\[
C_Q[p \triangleright P] \equiv p \triangleright r?\Lambda \mid q \triangleright p?\Lambda'' \mid \prod_{2 \leq i \leq n} p_i \triangleright G(Q, p)|_{p_i}
\]

is stuck because \( p \) and \( q \) cannot communicate.

Case [NSUB-DIFF-IN]. Let \( P = r?\Lambda \) and \( Q = q?\Lambda' \) with \( r \neq q \). By construction \( G(Q, p)|_q = p!\Lambda'' \), then

\[
C_Q[p \triangleright P] \equiv p \triangleright r?\Lambda \mid q \triangleright p!\Lambda'' \mid \prod_{2 \leq i \leq n} p_i \triangleright G(Q, p)|_{p_i}
\]

is stuck because \( p \) and \( q \) cannot communicate.

Case [NSUB-LAB-IN]. Let \( P = q?\Lambda \) and \( Q = q?\Lambda' \) with \( msg(\Lambda') \not\subseteq msg(\Lambda) \). By construction \( G(Q, p)|_q = p!\Lambda'' \) with \( msg(\Lambda'') \neq msg(\Lambda') \). We get:

\[
C_Q[p \triangleright P] \equiv p \triangleright q?\Lambda \mid q \triangleright p!\Lambda'' \mid \prod_{2 \leq i \leq n} p_i \triangleright G(Q, p)|_{p_i}
\]

which is stuck because \( msg(\Lambda'') \not\subseteq msg(\Lambda) \) and hence \( p \) and \( q \) cannot communicate.
Case [\textsc{nsub-cont-out}]. Let \( P = q!((\ell.P' \uplus \Lambda)) \) and \( Q = q!((\ell.Q' \uplus \Lambda')) \) with \( P' \neq Q' \). By construction \( G(Q,p)|_q = p?\ell.p_2!\ldots.p_n!\ell.Q_1 \) and \( G(Q,p)|_{p_i} = q?\ell.Q_i \), where \( Q_i = G(Q',p)|_{p_i} \) for \( 2 \leq i \leq n \). We get:

\[
C_Q[p \triangleright P] = p \triangleright q!((\ell.P' \uplus \Lambda)) | q \triangleright p?\ell.p_2!\ldots.p_n!\ell.Q_1 | \prod_{2 \leq i \leq n} p_i \triangleright q?\ell.Q_i
\]

\[
\frac{p/q}{p \triangleright P' | q \triangleright p_2!\ell \ldots p_n!\ell.Q_1 | \prod_{2 \leq i \leq n} p_i \triangleright q?\ell.Q_i}
\]

\[
\frac{q/p_2}{p \triangleright P' | q \triangleright p_3!\ell \ldots p_n!\ell.Q_1 | p_2 \triangleright Q_2 | \prod_{3 \leq i \leq n} p_i \triangleright q?\ell.Q_i}
\]

\[
\vdots
\]

\[
\frac{q/p_n}{p \triangleright P' | q \triangleright Q_1 | \prod_{2 \leq i \leq n} p_i \triangleright Q_i}
\]

By induction hypothesis we get \( \text{locked}(C_{Q'}[p \triangleright P']) \). \( \square \)

\( \textbf{Example 5.9.} \) An example showing the utility of the message spread in the definition of characteristic global types is \( P = p_1!\ell_1.p_2!\ell_2 \) and \( Q = p_2!\ell_2.p_1!\ell_1 \). In fact without the message spread the characteristic global type of \( Q \) would be

\[
G = p \rightarrow p_2 : \ell_2.p \rightarrow p_1 : \ell_1
\]

and then \( C_Q[\cdot] = [\cdot] | p_1 \triangleright G|_{p_1} | p_2 \triangleright G|_{p_2} = [\cdot] | p_1 \triangleright p?\ell_1 | p_2 \triangleright p?\ell_2 \). Being \( P = p_1!\ell_1.p_2!\ell_2 \), the session \( C_Q[P] \) reduces to \( p \triangleright 0 \). Instead

\[
G(Q,p) = p \rightarrow p_2 : \ell_2.p_2 \rightarrow p_1 : \ell_2.p \rightarrow p_1 : \ell_1.p_1 \rightarrow p_2 : \ell_1
\]

which implies \( G(Q,p)|_{p_1} = p_2?\ell_2 \ldots \) and \( G(Q,p)|_{p_2} = p?\ell_2 \ldots \). It is then easy to verify that

\[
p \triangleright P | p_1 \triangleright G(Q,p)|_{p_1} | p_2 \triangleright G(Q,p)|_{p_2}
\]

is stuck.

Theorems 5.3 and 5.8 imply the desired characterisation of observational preorder.

\( \textbf{Corollary 5.10. (Equivalence between Observational and Structural Preorders)} \)

\( P \leq Q \text{ iff } P \sqsubseteq Q \).

As a consequence of the equivalence between the structural preorder and the observational preorder we have that two processes are observationally equivalent if and only if they have the same regular tree.
6. Related Work

Multiparty sessions as extensions of binary sessions [15, 16] have been first proposed in [17, 1]. The fundamental notion of global type to govern communication protocols has been devised in the keystone paper [1]. Since then many papers have enhanced session calculus with various features and type disciplines, a recent survey is [18]. The majority of session calculi are asynchronous and channel based, and this strongly enlarges expressivity, but the drawback is that deadlock-freedom requires sophisticated type systems [6]. Observational equivalence of processes is naturally defined inside only one synchronous session, and therefore in the present paper we consider a single session with synchronous communications and without channels as in [19, 8].

Observational and structural preorders for session types are considered in [20, 21, 22]. For synchronous binary sessions types seen as contracts between clients and servers the compliance assures that all requests of the client are satisfied by the server. Compliance naturally induces an observational preorder on clients and servers, which is proved to coincide with a structural preorder [21, 22]. Multiparty session types in [20] are equipped with an operational semantics and the subtyping relation is shown to be exactly safe substitutability. A main difference between the above papers and the present one is that they consider session types, while we deal with processes and global types. Moreover in the case of multiparty session we are more demanding, since we require strong lock-freedom, while safety in [20] is deadlock-freedom.

In the present paper processes are defined coinductively as in [23], and this allows us to avoid an explicit recursion operator. We also define coinductively global types. Hence, the equalities between processes and global types are actually bisimulations and we have used that fact in the proof of unicity of the projection. The structural preorder is also defined by coinduction and its symmetric closure is a bisimulation, which we have proved to coincide with the observational equivalence on strong lock-freedom.

Bisimulations for well-typed binary sessions have been studied in [24], where asynchronous processes are augmented by event primitives, and in [25], where values exchanged can be λ-abstractions. For multiparty sessions an equivalence that takes into account global types as specifications is developed in [26]. The global types we use to show that the observational preorder implies the structural preorder play a role similar to that of characteristic processes in [25]. Moreover in the globally governed session bisimilarity of [26] the interaction patterns of the observers are built out of global types. The characteristic processes in [25] are inhabitants of session types and the contexts we build for showing that observational preorder implies structural preorder inhabit characteristic global types. In both cases the constructions are linear in the dimension of the types. This is in contrast to other type system where the inhabitation problem has very high complexity or sometimes it is even undecidable, as proved by Pawel Urzyczyn in a series of papers [27, 28, 29, 30, 31].

The paper more related to the present one is [8], which shows the preciseness of subtyping for synchronous multiparty sessions. A subtyping relation is precise if it is both sound and complete. Soundness is the usual property allowing to replace terms of smaller type to terms of bigger type in arbitrary well-typed contexts preserving the properties guaranteed by types. Completeness was first defined in [32]: it requires that if a type \( T \) is not smaller than a type \( S \), then we can find a term \( t \) and a context such that the context filled with all terms of type \( S \) is well typed, while the context filled
with $t$ looses the properties assured by typing. The present calculus is essentially that of [8], but for the use of coinduction, which better fits the characterisation of observational equivalence, instead of explicit recursion. The main simplifications (allowed by our different aim) is the dropping of session types, the definition of structural preorder on processes and the direct projection of global types into processes. Our structural preorder on processes mimics the subtyping relation between session types of [5]. This choice is justified by the fact that the subtyping of [5] allows process substitution, while the subtyping of [4] allows channel substitution, as observed in [33]. The subtyping of [5] is used in [8]. As a consequence our proof of Theorem 5.8 is similar to the proof of Theorem 4.4 in [8], also if the new definition of characteristic global type is a simplification of that one in [8]. Finally we recall that the paper [34] characterises by means of evaluation trees the observational equivalence of $\lambda$-terms in a concurrent $\lambda$-calculus. That paper can then be considered an ancestor of the present one.

7. Conclusion

In this paper we strengthened the notion of lock-freedom and proved its decidability. We defined a type system for sessions and proved that well-typed sessions are strong lock-free. We considered a structural preorder between processes, which allows smaller processes to select less outputs and offer more inputs, and proved its decidability. We have proved that (1) if the process $P$ is smaller than the process $Q$, then $P$ can be plugged into any session instead of $Q$ without losing the property of strong lock-freedom and (2) if the process $P$ is not smaller than the process $Q$, then there is a session which is strongly lock-free by plugging $Q$, but it is not strongly lock-free when plugging $P$.

The interest of our result on the equivalence between the preorders lies on the fact that the programmer can know which syntactic changes on local processes would preserve strong lock-freedom of the whole session and which ones not. Strong lock-freedom for this particular calculus could be guaranteed either by typing the session or by running the strong lock-freedom algorithm described.

We plan to investigate observational equivalences for more expressive calculi such as the asynchronous multiparty session calculus [2] and the lazy lambda calculus with futures [35]. It would also be interesting to find structural preorders for removing or adding communications in a global type (or session) safely or in the presence of participants that can be connected and disconnected [36].

Acknowledgements The friendship and collaboration between Paweł and Mariangiola started in the 80s, when they were both working on $\lambda$-calculus with types. The occasions were conferences, meetings of European projects and more importantly exchanged long visits. During these visits the pleasant relationship extended to families. Mariangiola feel really grateful to Paweł for what she understood and learned in discussing with him various subjects.

We would like to thank the anonymous referees for their helpful comments.

References


A. Finiteness of Weight and Unicity of the Projection

We start by re-defining the projection as the largest relation between global types and processes such that \((G, P) \in \downarrow_p\) implies that

- If \(p \notin \text{ppts}(G)\), then \(P = 0\)
- If \(G = p \to q : \{\ell_i, G_i \mid 1 \leq i \leq n\}\), then \(P = q!\{\ell_i, P_i \mid 1 \leq i \leq n\}\) and \((G_i, P_i) \in \downarrow_p\)
- If \(G = q \to p : \{\ell_i, G_i \mid 1 \leq i \leq n\}\), then \(P = q?\{\ell_i, P_i \mid 1 \leq i \leq n\}\) and \((G_i, P_i) \in \downarrow_p\)
- If \(G = q \to r : \ell.G'\) and \(p \notin \{q, r\}\) and \(p \in \text{ppts}(G)\), then \((G', P) \in \downarrow_p\) and \(P \neq 0\)
- If \(G = q \to r : \{\ell_i, G_i \mid 1 \leq i \leq n\}\) and \(p \notin \{q, r\}\) and \(n > 1\) and \(p \in \text{ppts}(G)\), then \(P = s?\langle \Lambda_1 \cup \ldots \cup \Lambda_n \rangle\) and \((G_i, s?\Lambda_i) \in \downarrow_p\) for \(1 \leq i \leq n\) and all messages in \(\Lambda_1 \cup \ldots \cup \Lambda_n\) are different.

To show that \textit{weight} is finite it is handy to use the following measure for the induction:

\[
\gamma_p(G, P) = \begin{cases} 
(0, 0) & \text{if } P = 0 \\
(#(\text{msg}(\Lambda)), \delta_p(G)) & \text{if } P = q?\Lambda \\
(#(\text{msg}(\Lambda)), \delta_p(G)) & \text{if } P = q!\Lambda
\end{cases}
\]

with the lexicographic order, where \(#(X)\) denotes the cardinality of the set \(X\) and \(\delta_p(G)\) is defined by:

\[
\delta_p(G) = \begin{cases} 
\min\{\text{weight}(\rho, p) \mid \rho \in \text{paths}(G)\} & \text{if } p \in \text{ppts}(G) \\
0 & \text{otherwise}
\end{cases}
\]

using \textit{weight} as given in Definition 3.10. This measure is illustrated by the example in Figure 1.

\textbf{Proof of Lemma 3.11.}

We show that if \((G, P) \in \downarrow_p\) then either \(p \notin \text{ppts}(G)\) and \(P = 0\) or

1. \(p \in \text{ppts}(G)\) and \(P \neq 0\) and
2. for all \(\rho \in \text{paths}(G)\), \(\rho\) is a path of the form \(\rho_1 (r \to s) \ell \rho_2\), where \(p \in \{r, s\}\) and \(p\) does not occur in the prefix \(\rho_1\)

by induction on \(\gamma_p(G, P)\).

The basic case \(\gamma_p(G, P) = (0, 0)\) gives \(P = 0\) by definition of \(\gamma_p\) and \(p \notin \text{ppts}(G)\) by definition of \(\downarrow_p\).

For the induction step suppose \((G, P) \in \downarrow_p\). From \(\gamma_p(G, P) \neq (0, 0)\) we get \(P \neq 0\), which implies \(p \in \text{ppts}(G)\) by definition of \(\downarrow_p\).

Case \(G = q \to p : \{\ell_i, G_i \mid 1 \leq i \leq n\}\). Then \(P = q?\Lambda\) and \(\Lambda = \{\ell_i, P_i \mid 1 \leq i \leq n\}\) by definition of projection. Clearly each path of \(G\) is of the form \((q \to p) \ell_i \rho\).
Definition A.1. A bisimulation is a symmetric binary relation $R$ on processes that satisfies the following clauses:

1. if $P = 0$ and $(P, Q) \in R$, then $Q = 0$;
2. if $P = q!\{\ell_i, P_i \mid 1 \leq i \leq n\}$ and $(P, Q) \in R$, then $Q = q!\{\ell_i, Q_i \mid 1 \leq i \leq n\}$ and $(P_i, Q_i) \in R$ for all $1 \leq i \leq n$;
3. if $P = q?\{\ell_i, P_i \mid 1 \leq i \leq n\}$ and $(P, Q) \in R$, then $Q = q?\{\ell_i, Q_i \mid 1 \leq i \leq n\}$ and $(P_i, Q_i) \in R$ for all $1 \leq i \leq n$. 

Figure 1. Measure $\gamma_p(\cdot, \cdot)$ on a global type and its subtrees.

Case $G = p \rightarrow q : \{\ell, G_i \mid 1 \leq i \leq n\}$. Similar to the previous case.

Case $G = r \rightarrow s : \ell, G_i$ and $p \notin \{r, s\}$ and $p \in \text{plets}(G)$. By definition of projection, $(G', P) \in \downarrow P$ and $P \neq 0$. Then either $P = q?\Lambda$ or $P = q!\Lambda$ and in both cases $\gamma_p(G, P) = (\#(\Lambda), \delta_p(G)) > (\#(\Lambda), \delta_p(G'))$. By induction hypothesis all $p' \in \text{paths}(G')$ have the form $p_1 (r' \rightarrow s') \ell' \rho_2$, where $p \in \{r', s'\}$ and $p$ does not occur in $p_1$.

Case $G = r \rightarrow s : \{\ell_j, G_j \mid 1 \leq j \leq m\}$ and $p \notin \{r, s\}$ and $m > 1$ and $p \notin \text{plets}(G)$. By definition of projection, $P = q?\Lambda$ and $\Lambda = \Lambda_1 \uplus \ldots \uplus \Lambda_m$ and $(G_j, q?\Lambda_j) \in \downarrow P$ for $1 \leq j \leq m$. Since all messages in $\Lambda_1 \uplus \ldots \uplus \Lambda_m$ are different, $\#(\Lambda) = \#(\Lambda_1) + \ldots + \#(\Lambda_m) > \#(\Lambda_j)$ for all $j$ ($1 \leq j \leq m$). Suppose $\rho \in \text{paths}(G)$. Then $\rho = (r \rightarrow s) \ell_j \rho'$ and $\rho' \in \text{paths}(G_j)$ for some $j$ ($1 \leq j \leq m$). By induction hypothesis, $\rho' = p_1 (r' \rightarrow s') \ell' \rho_2$, where $p$ does not occur in $p_1$ and $p \in \{r', s'\}$. The required path is $\rho = (r \rightarrow s) \ell_j p_1 (r' \rightarrow s') \ell' \rho_2$. 

To prove that the projection defined as a relation is a (partial) function we use bisimulation between processes defined as expected.
Lemma A.2. The equality between processes is the largest bisimulation.

Proof:
We prove that \( R = \{(P, Q) \mid (G, P) \in |_p \text{ and } (G, Q) \in |_p \text{ for some } G\} \) is a bisimulation in the sense of Definition A.1. The first clause is very easy to show.

For the second clause, we prove that if \( P = q!\Lambda \) and \( (P, Q) \in R \), then

a) \( Q = q!\Lambda' \);

b) \( msg(\Lambda) = msg(\Lambda') \);

c) \( (R, S) \in R \) for all \( \ell.R \in \Lambda \) and \( \ell.S \in \Lambda' \).

The proof is by induction on \( weight(G, p) \). By definition of projection, there are two cases.

Case \( G = p \rightarrow q : \{\ell_i.G_i \mid 1 \leq i \leq n\} \) and \( (G, P_i) \in |_p \). Then \( \Lambda = \{\ell_i.P_i \mid 1 \leq i \leq n\} \). It follows from \( (G, Q) \in |_p \) that \( Q = q!\{\ell_i.Q_i \mid 1 \leq i \leq n\} \) and \( (G_i, Q_i) \in |_p \). Hence by definition of bisimulation \( (P_i, Q_i) \in R \).

Case \( G = r \rightarrow s : \ell.G' \) and \( p \notin \{r, s\} \) and \( (G', P) \in |_p \). By definition of projection, \( (G', Q) \in |_p \).

Since \( weight(G, p) > weight(G', p) \), by induction \( a, b \) and \( d \) hold.

For the third clause, we prove that if \( P = q?\Lambda \) and \( (P, Q) \in R \) then

d) \( Q = q?\Lambda' \);

e) \( msg(\Lambda) = msg(\Lambda') \);

f) \( (R, S) \in R \) for all \( \ell.R \in \Lambda \) and \( \ell.S \in \Lambda' \).

The proof is by induction on \( weight(G, p) \). By definition of projection, there are three cases.

Case \( G = q \rightarrow p : \{\ell_i.G_i \mid 1 \leq i \leq n\} \) and \( (G_i, P_i) \in |_p \). Then \( \Lambda = \{\ell_i.P_i \mid 1 \leq i \leq n\} \). It follows from \( (G, Q) \in |_p \) that \( Q = q!\{\ell_i.Q_i \mid 1 \leq i \leq n\} \) and \( (G_i, Q_i) \in |_p \). Hence by definition of bisimulation \( (P_i, Q_i) \in R \).

Case \( G = r \rightarrow s : \ell.G' \) and \( p \notin \{r, s\} \) and \( (G', P) \in |_p \). By definition of projection, \( (G', Q) \in |_p \).

Since \( weight(G, p) > weight(G', p) \), by induction \( c \) and \( e \) hold.

Case \( G = r \rightarrow s : \{\ell_i.G_i \mid 1 \leq i \leq n\}, p \notin \{r, s\} \) and \( 1 > P = q?\Lambda_1 \uplus \ldots \uplus \Lambda_n \), \( (G_i, q?\Lambda_i) \in |_p \) for \( 1 \leq i \leq n \) and all messages in \( \Lambda_1 \uplus \ldots \uplus \Lambda_n \) are different. It follows from \( (G, Q) \in |_p \) that \( Q = q!(\Lambda_1 \uplus \ldots \uplus \Lambda_n) \), \( (G_i, q?\Lambda_i) \in |_p \) for \( 1 \leq i \leq n \) and all messages in \( \Lambda_1 \uplus \ldots \uplus \Lambda_n \) are different. Since \( weight(G, p) > weight(G_i, p) \) for \( 1 \leq i \leq n \), by induction \( f \) and \( g \) hold. This implies \( msg(\Lambda_i) = msg(\Lambda_i') \) and \( (R, S) \in R \) for all \( \ell.R \in \Lambda_i \) and \( \ell.S \in \Lambda_i' \) for \( 1 \leq i \leq n \). We conclude \( msg(\Lambda) = msg(\Lambda') \) and \( (R, S) \in R \) for all \( \ell.R \in \Lambda \) and \( \ell.S \in \Lambda' \). \( \Box \)
B. Algorithm for Checking the Structural Preorder

This section presents an algorithm for checking whether two processes are related by the structural preorder, and prove its correctness. For this, we define an algorithmic proof system similar in spirit to the one by Brandt and Henglein for checking subtyping [37]. The difference is that we consider regular processes instead of equi-recursive types using $\mu$-notation (see Figure 3 of [37] and also Figure 11.3 of [11]). The proof system is simple but the proofs of termination and correctness of the algorithm are delicate. Since we are not using $\mu$-notation, there is no obvious metric to do the induction.

**Definition B.1. (Algorithmic Proof System for $\preceq$)**

Let $\Theta$ be a set of pairs of processes. In writing $\Theta$, $(P,Q)$ we assume $(P,Q) \not\in \Theta$.

\begin{align*}
\text{[ASUB-0]} & \quad \Theta \vdash 0 \preceq 0 \\
\text{[ASUB-AXIOM]} & \quad \Theta, (P,Q) \vdash P \preceq Q \\
\text{[ASUB-IN]} & \quad P = p?\{(\ell_i.P_i \mid 1 \leq i \leq n) \cup \Lambda\} \\
& \quad \Theta, (P,Q) \vdash P_i \preceq Q_i \forall 1 \leq i \leq n \\
& \quad Q = p?\{(\ell_i.Q_i \mid 1 \leq i \leq n)\} \\
& \quad \Theta \vdash P \preceq Q \\
\text{[ASUB-OUT]} & \quad P = p!\{(\ell_i.P_i \mid 1 \leq i \leq n)\} \\
& \quad \Theta, (P,Q) \vdash P_i \preceq Q_i \forall 1 \leq i \leq n \\
& \quad Q = p!(\{(\ell_i.Q_i \mid 1 \leq i \leq n)\} \cup \Lambda) \\
& \quad \Theta \vdash P \preceq Q
\end{align*}

The regularity condition is crucial to guarantee the termination of the algorithm induced by the rules in Definition B.1. For example, if we try to find a derivation of $\Theta \vdash P \preceq P$ for the non-regular process $P = p!\ell_1.p!\ell_2.p!\ell_3\ldots$, then the last rule can be applied an infinite number of times.

Algorithm[1] gives a pseudo-code translating the proof system by reading the rules bottom-up. This pseudo-code should be compared with the pseudo-codes in Figure 5 of [37] and Figure 21-4 of [9].

We now prove that the derivations constructed from the algorithmic proof system of Definition B.1 are finite.

**Theorem B.2. (Finite Derivations)**

Any derivation of $\vdash P \preceq Q$ is finite.

**Proof:**

Suppose there is an infinite sequence

$$
\Theta_0 \vdash P_0 \preceq Q_0 \quad \Theta_1 \vdash P_1 \preceq Q_1 \quad \Theta_2 \vdash P_2 \preceq Q_2 \quad \ldots
$$

where $\Theta_0 = \Theta$, $P_0 = P$, $Q_0 = Q$ and $\Theta_i \vdash P_i \preceq Q_i$ is obtained from some $\Theta_j \vdash P_j \preceq Q_j$ with $j < i$ by applying some of the rules of Definition B.1. These rules have to be [ASUB-OUT] or [ASUB-IN] because they are the only ones with premises. An upper bound for $\Theta_i$ is the set

$$
\text{subtrees}(P,Q) = \{(P',Q') \mid P' \in S(P) \text{ and } Q' \in S(Q)\}
$$
Algorithm 1: Algorithm for checking $\leq$

**Function** subprocess?($\Theta, P, Q$)

**input**: $\Theta$ is a set of pair of processes, $P$ and $Q$ are processes

**output**: true iff $\Theta \vdash P \leq Q$

1. if $(P, Q) \in \Theta$ then return true;
2. else if $P = Q = 0$ then return true;
3. else
   4. case $P = p\{\ell_i.P_i \mid 1 \leq i \leq n\} \cup \Lambda\} $ and $Q = p\{\ell_i.Q_i \mid 1 \leq i \leq n\}$ /* Input */
   5. $b := true$;
   6. $\Theta' := \Theta \cup \{(P, Q)\}$;
   7. foreach $i$ such that $1 \leq i \leq n$ do
     8. $b := b$ and subprocess?($\Theta', P_i, Q_i$)
   9. return $b$
10. case $Q = p\!\{\ell_i.Q_i \mid 1 \leq i \leq n\} \cup \Lambda\} $ and $P = p\!\{\ell_i.P_i \mid 1 \leq i \leq n\}$ /* Output */
11. $b := true$;
12. $\Theta' := \Theta \cup \{(P, Q)\}$;
13. foreach $i$ such that $1 \leq i \leq n$ do
14. $b := b$ and subprocess?($\Theta', P_i, Q_i$)
15. return $b$
16. otherwise return false ;

where $S(P)$ is defined in the proof of Theorem 2.15 The set $\text{subtrees}(P, Q)$ is finite since $P$ and $Q$ are regular. This implies that there exists $k$ such that for all $i \geq k$, $\Theta_i = \Theta_k$. But this contradicts the fact that [ASUB-OUT] and [ASUB-IN] can be applied only if $(P_i, Q_i)$ does not belong to $\Theta_i$. $\square$

We now prove the equivalence of Definitions 3.12 and B.1 using preorder simulation.

**Definition B.3.** A preorder simulation is a binary relation $\mathcal{R}$ on processes that satisfies the following clauses:

1. if $P = 0$ and $(P, Q) \in \mathcal{R}$, then $Q = 0$;
2. if $P = p\!\{\ell_i.P_i \mid 1 \leq i \leq n\}$ and $(P, Q) \in \mathcal{R}$, then $Q = p\!\{\ell_i.Q_i \mid 1 \leq i \leq n\} \cup \Lambda\} $ and $(P_i, Q_i) \in \mathcal{R}$ for all $1 \leq i \leq n$;
3. if $P = p\!\{\ell_i.P_i \mid 1 \leq i \leq n\} \cup \Lambda\} $ and $(P, Q) \in \mathcal{R}$, then $Q = p\!\{\ell_i.Q_i \mid 1 \leq i \leq n\}$ and $(P_i, Q_i) \in \mathcal{R}$ for all $1 \leq i \leq n$.

The structural preorder is the largest preorder simulation.

**Theorem B.4.** (Soundness and Completeness of the Algorithmic Proof System for $\leq$)
$P \leq Q$ iff $\vdash P \leq Q$. 

**Lemma C.1.** **Negation of the Structural Preorder**

The correctness of the algorithm follows from Theorem B.4.

For the converse, we prove the following claim, where *subtrees* is the set defined in the proof of Theorem B.2.

**Claim.** If \( P \leq Q \), then \( \Theta \vdash P \leq Q \) for all \( \Theta \subseteq \text{subtrees}(P, Q) \).

The claim is proved by induction on the cardinality \( k \) of the set \( \text{subtrees}(P, Q) \setminus \Theta \).

If \( k = 0 \), then \( \Theta = \text{subtrees}(P, Q) \). Since \( (P, Q) \in \Theta \), we have that \( \Theta \vdash P \leq Q \).

Suppose \( k > 0 \). If \( (P, Q) \in \Theta \), then \( \Theta \vdash P \leq Q \). If \( (P, Q) \notin \Theta \), we have 3 cases:

- **Case 1**: \( P = 0 \). The definition of \( \leq \) implies \( Q = 0 \).

  By definition of \( \leq \), we get \( Q = p!\{\ell_i.P_i \mid 1 \leq i \leq n \} \) and \( P_i \leq Q_i \) for \( 1 \leq i \leq n \). The induction hypothesis implies \( \Theta, (P, Q) \vdash P_i \leq Q_i \) since \( \Theta \cup (P, Q) \supset \Theta \).

  Then, \( \Theta \vdash P \leq Q \).

- **Case 2**: \( P = p?\{\ell_i.P_i \mid 1 \leq i \leq n \} \cup L \). The proof is similar to that of previous case.

Termination of Algorithm 1 follows from Theorem B.2 because we add the pair \((P, Q)\) to \( \Theta \) on Line 6 and Line 12 only when we know that \((P, Q)\) is not in \( \Theta \). In fact Line 6 and Line 12 are in the else part of Line 1. The correctness of the algorithm follows from Theorem B.4.

**C. Negation of the Structural Preoder**

**Lemma C.1.** \( P \neq Q \) is the negation of \( P \leq Q \).

**Proof:**

If \( P \neq Q \), then we can show \( P \not\leq Q \) by induction on the derivation of \( P \neq Q \). We develop just two cases (the others are similar):

- **base case** [NSUB-DIFF-IN]. Then, \( P = p?\ell_1.P_1 \) and \( Q = q?\ell_2.P_2 \) with \( p \neq q \). It is easy to verify that \( P \) and \( Q \) do not match the conclusion of [SUB-END], nor [SUB-IN], nor [SUB-OUT], hence, we conclude \( P \not\leq Q \);

- **inductive case** [NSUB-CONT-IN]. Then, \( P = p?(\ell.P'(\cup L)) \) and \( Q = p?(\ell.Q'(\cup L')) \) and \( P' \neq Q' \). By the induction hypothesis, \( P' \leq Q' \). We now notice that \( P \leq Q \) can hold only by matching the rule [SUB-IN] with \( P' \leq Q' \), which is a contradiction. Hence, we conclude \( P \not\leq Q \).

Vice versa, assume \( P \not\leq Q \): if we try to apply the rules of Definition 8.12 to show \( P \leq Q \), we will “fail” after \( n \) derivation steps. We prove \( P \neq Q \) by induction on \( n \):

- **base case** \( n = 0 \). The derivation “fails” immediately. By cases on the possible shapes of \( P \) and \( Q \), we obtain \( P \neq Q \) by any of the rules except for [NSUB-CONT-IN] and [NSUB-CONT-OUT];
• inductive case $n = m + 1$. The shapes of $P, Q$ match the conclusion of either rule [SUB-IN] or [SUB-OUT]. Assume $P = p^?\{\ell_i.P_i \mid 1 \leq i \leq n\} \uplus \Lambda$ and $Q = p^?\{\ell_i.Q_i \mid 1 \leq i \leq n\}$. Then, there is some $i$ such that $P_i \not< Q_i$. The derivation of $P_i \leq Q_i$ fails after $m$ steps. By the induction hypothesis, we have $P_i \not< Q_i$; therefore, we can derive $P \not< Q$ by [NSUB-CONT-IN].

$\square$