Multiparty Session Types, Beyond Duality

Alceste Scalas  
Imperial College London  
alceste.scalas@imperial.ac.uk

Nobuko Yoshida  
Imperial College London  
n.yoshida@imperial.ac.uk

Multiparty Session Types (MPST) are a well-established typing discipline for message-passing processes interacting on sessions involving two or more participants. Session typing can ensure desirable properties: absence of communication errors and deadlocks, and protocol conformance. However, existing MPST works provide a subject reduction result that is arguably (and sometimes, surprisingly) restrictive: it only holds for typing contexts with strong duality constraints on the interactions between pairs of participants. Consequently, many “intuitively correct” examples cannot be typed and/or cannot be proved type-safe. We illustrate some of these examples, and discuss the reason for these limitations. Then, we present a novel MPST typing system that removes these restrictions.

1 Introduction

The Multiparty Session Types (MPST) framework [7] provides a well-established typing discipline for processes interacting on multiparty sessions. In a nutshell, it fosters a top-down approach where:

1. A global type $G$ formalises a choreography involving $n$ interacting roles;
2. $G$ is projected onto a set of (local) session types $S_1, \ldots, S_n$ (one per role of $G$);
3. $S_1, \ldots, S_n$ are assigned to channels, used by MPST $\pi$-calculus processes that are type-checked. This ensures properties such as type safety and protocol conformance (i.e., all sent/received messages comply with the session types, and ultimately with their originating global type $G$). E.g., consider:

$$G = p \to q. \begin{cases} m1(\text{Int}).q \to r.m2(\text{Str}).r \to p.m3(\text{Bool}).\text{end} \\ \text{stop}.q \to r.\text{quit}.\text{end} \end{cases} \quad (1)$$

The global type $G$ involves three roles: $p$, $q$, $r$. It says that $p$ sends to $q$ either a message $m1$ (carrying an Integer) or $\text{stop}$; in the first case, $q$ sends $m2$ to $r$ (carrying a String), then $r$ sends $m3$ to $p$ (carrying a Boolean), and the session ends; otherwise, in the second case, $q$ sends $\text{quit}$ to $r$, and the session ends.

The projections of $G$ describe the I/O actions expected from processes that play the roles in $G$:

$$S_p = q \bigoplus \begin{cases} m1(\text{Int}).r & \text{m3(Bool)} \\ \text{stop} \end{cases}, S_q = p \& \begin{cases} m1(\text{Int}).r & \text{m2(Str)} \\ \text{stop} \end{cases} r & \text{quit}, S_r = q \& \begin{cases} m2(\text{Str}).p & \text{m3(Bool)} \\ \text{quit} \end{cases} \quad (2)$$

Here, $S_p$, $S_q$, $S_r$ are the projections of $G$ resp. onto $p$, $q$, $r$. E.g., $S_p$ is a session type that represents the behaviour of $p$ in $G$: it must send ($\oplus$) to $q$ either $m1(\text{Int})$ or $\text{stop}$; in the first case, the channel is then used to receive ($\&$) message $m3(\text{Bool})$ from $r$, and the session ends; otherwise, in the second case, the session ends. Below is an example of MPST process matching such types: $s$ is a multiparty session, and $s[p]$ (resp. $s[q]$, $s[r]$) is a channel with role used to interact on $s$ while playing the role of $p$ (resp. $q$, $r$).

$$\begin{cases} P_p = s[p][q] \oplus \text{stop} \\ P_q = s[q][p] \& \{m1(x).s[q][r] \oplus m2(\text{"Hi"})\}, \text{stop}, s[q][r] \oplus \text{quit} \\ P_r = s[r][q] \& \{m2(y).s[r][p] \oplus m3(\text{true})\}, \text{quit} \end{cases} \quad (3)$$

Here, $(vs) (P_p \mid P_q \mid P_r)$ is the parallel composition of processes $P_p$, $P_q$, $P_r$ in the scope of $s$. In $P_p$, “$s[p][q] \oplus \text{stop}$” means: $s[p]$ is used to send $\text{stop}$ to $q$. $P_q$ uses $s[q]$ to receive either $m1$ or $\text{stop}$ from $p$; then, in the first case it uses $s[q]$ to send $m2$ to $r$; in the second case, it uses $s[q]$ to send $\text{quit}$ to $r$. © Alceste Scalas and Nobuko Yoshida  
This work is licensed under the Creative Commons Attribution-Share Alike License.
Using the standard MPST typing rules, we have the following typing derivation:

\[
\begin{align*}
\forall \rho \in \text{roles}(G) = \{p, q, r\} : S_{\rho'} &= G \vdash \rho' \\
\vdash P_p \triangleright \Gamma & \quad \vdash P_q \triangleright \Gamma
\end{align*}
\]

(4)

Here, \( s \) has type \( G \), because in the premise of [STV-v], \( \triangleright \) (resp. \( \triangleright \)) has type \( G \vdash p \) (resp. \( G \vdash q \)), i.e., the projection of \( G \) onto \( p \) (resp. \( q \)). The parallel process \( P_p \parallel P_q \parallel P_r \) is typed by rule \([\text{STD}-\triangleright]\), that splits the typing context so that a channel endpoint is not used by two parallel sub-processes. In the omitted part of the derivation, \( P_p, P_q, P_r \) are typed: they use resp. \( s[p], s[q], s[r] \) abiding by \( S_p, S_q, S_r \).

**Inter-role dependencies** Note that after sending \( \text{stop} \) to \( q \), \( P_p \) above terminates without receiving messages from \( r \) via \( s[p] \). Indeed, according to \( S_p, P_p \) must input a message from \( r \) only after sending \( m1 \) (instead of \( \text{stop} \)) to \( p \) — and this would be an error. However, such an error does not occur: by observing the whole ensemble \( P_p \parallel P_q \parallel P_r \), we see that when \( P_q \) receives \( \text{stop} \) from \( P_p \), it “forwards” \( r \) to \( (as \ per \ S_q) \); and when \( P_r \) receives \( r \) from \( q \), it does not try to communicate with \( p \) (as per \( S_r \)). Hence, \( S_p \) makes \( P_p \) “assume” that by sending a certain message to the process playing role \( q \), it will also influence the process playing \( r \): this is an implicit inter-role communication dependency, that can be more explicitly observed in \( G \).

**Subject reduction** Type safety is based on the subject reduction property: by showing that typed processes can only reduce to typed processes, we obtain that no (untypeable) error configurations can be reached. Hence, we would expect that subject reduction holds for our typed example above — roughly:

\[
\vdash P \triangleright \Gamma \quad \text{and} \quad P \rightarrow^* P' \implies \exists \Gamma' : \vdash P' \triangleright \Gamma'
\]

(5)

But surprisingly, this is not the case! In MPST works (e.g., [3]), the subject reduction statement reads:

\[
\vdash P \triangleright \Gamma \quad \text{consistent} \quad \text{and} \quad P \rightarrow^* P' \implies \exists \Gamma' \quad \text{consistent} \quad \text{such that} \quad \vdash P' \triangleright \Gamma'
\]

(6)

i.e., subject reduction is only proved for consistent (also called coherent) typing contexts. The reason is technical: the proof of MPST subject reduction needs an invariant that (\( R1 \)) makes the typing context “safe” (e.g., if the type of \( s'[p'] \) sends a message to \( q' \), then the type of \( s'[q'] \) can input such a message from \( p' \)), (\( R2 \)) is preserved when processes communicate and reduce, and (\( R3 \)) is preserved when typing contexts are (de)composed by parallel composition (cf. [\text{STD}-\triangleright] in [3]). Consistency satisfies (\( R1 \)) and (\( R3 \)) by imposing the duality of the partial projections \( S_{p'} \vdash q' \) and \( S_{q'} \vdash p' \), for each pair \( \{s[p'] : S_{p'}, s[q'] : S_{q'}\} \in \Gamma \) (cf. [7]). In our example, the pair \( s[p] : S_p, s[q] : S_q \) is consistent: the outputs of \( S_p \) to \( q \) are dual w.r.t. the inputs of \( S_q \) from \( p \). However, the pair \( s[p] : S_p, s[r] : S_r \) is not consistent: \( S_p \vdash r \) and \( S_r \vdash p \) are undefined—precisely because the inputs/outputs of \( S_p/S_q \) from/to \( r/p \) depend on previous I/O with \( q' \) (See Ex.A.6). Spotting “inconsistencies” is tricky: it requires examining all pairs of projections of a choreography; further, choreographies with inter-role dependencies are often “inconsistent”—even simple ones, like our \( G \) in [1]; and without consistency, subject reduction (and type safety) is not guaranteed.

**Contributions** We present a novel MPST typing system (Def.3.3) whose subject reduction property (Theorem 3.6) holds for a larger set of processes w.r.t. existing MPST literature. Our typing rules (\( P \)) record and exploit “global” typing information through an additional “typing rely context”, (2) require “live” (but not necessarily consistent) typing contexts, (3) do not require the existence of a global type (unlike [STV-v] in [4]), and (4) subsume the “standard” MPST typing rules as special cases. We claim that our “liveness” condition is a reasonable, semantically-grounded characterisation of “correct” multiparty interactions, amenable for “bottom-up” choreography synthesis. Proofs are available in §C.
2 Multiparty Sessions: Standard Calculus, Types and Typing Contexts

We first provide a series of standard MPST definitions, adopting a notation roughly based on [3]. For simplicity, we omit basic values (e.g., strings, integers, booleans...). Our new theory is presented in §3. We first provide a series of standard MPST definitions, adopting a notation roughly based on [3].

Definition 2.1 (MPST π-Calculus). MPST processes have the syntax and semantics shown in Fig.1.

A channel c or d can be a variable x, or a channel with role s[p], used to play the role p in session s, and send/receive messages to/from other roles in s. A process P or Q can be: a selection that uses c to send a message m with payload d to q, and continues as P; a branching that uses c to receive from q a message m, for any i ∈ I, binds x_i with its payload, and continues as P_i; a parallel composition where P, Q run concurrently (and possibly communicate); a delimitation of the scope of s to P; a definition of X (with parameters (x)) as P, with scope Q; a call of X with actual parameters c; a terminated process 0.

As expected, our MPST typing system (§3) will assign session types (Def 2.2) to channels (Def 2.1).

Definition 2.2 (Multiparty Session Types). Session types (ranked over by S, T) have the syntax:

$$ S, T ::= p \otimes_{i \in I} m_i(S_i) . S'_i | p \oplus_{i \in I} m_i(S_i) . S'_i | \text{end} | \mu t. S | t $$

where \( m_i \) range over pairwise distinct message labels. We define \( \equiv \) as the coinductive equivalence between type trees, such that: \( \mu t. S \equiv S'_i[\mu t/s_i] \) (i.e., a recursive type is equivalent to its unfolding).

The branching type (or external choice) \( p \otimes_{i \in I} m_i(S_i) . S'_i \) says that a channel must be used to receive from p one input of the form \( m_i(S_i) \), for any \( i \in I \) chosen by p; then, the channel must be used following the continuation type \( S'_i \). The selection type (or internal choice) \( p \oplus_{i \in I} m_i(S_i) . S'_i \), instead, requires to use a channel to perform one output \( m_i(S_i) \) towards p, for some \( i \in I \), and continue using the channel according to \( S'_i \). end is a terminated channel allowing no further inputs/outputs. \( \mu t. S \) is a recursive session type: \( \mu \) binds the recursion variable \( t \) in \( S \). Recursive types are contractive: i.e., in \( \mu t. S \) we have \( S \equiv t' \ (\forall t') \).

Remark 2.3. For brevity, we often omit 0/end in processes/types, and end-typed message payloads.

Remark 2.4. Unlike most MPST works, our processes and types coalesce branching/input and selection/output. This reduces the amount of notation, without harming expressiveness: a "pure" input/output can be encoded as a singleton branch/select prefix, while a "pure" branching/session can be represented by letting each branch carry an end-typed channel.

We adopt a standard definition of MPST typing context (Def 2.5), slightly adapting the reduction \( \rightarrow \).

Definition 2.5 (Typing Context). Let \( \Gamma \) be a partial mapping defined as: \( \Gamma ::= \emptyset | \Gamma, x : S | \Gamma, s[p] : S \), with the composition \( \Gamma_1, \Gamma_2 \) defined iff \( \text{dom}(\Gamma_1) \cap \text{dom}(\Gamma_2) = \emptyset \). We write \( \Gamma \equiv \Gamma' \) iff \( \text{dom}(\Gamma) = \text{dom}(\Gamma') \) and \( \forall c \in \text{dom}(\Gamma) : (\Gamma(c)) \equiv (\Gamma')(c) \). We define \( \Gamma \setminus c \) so that \( (\Gamma \setminus c)(d) = (\Gamma(d)) \text{ iff } d \neq c \) (and is undefined otherwise).

The typing contexts reduction relation \( \rightarrow \) is inductively defined as shown on the right (where \( * \) stands for either \& or \or).
An MPST typing context $\Gamma$ maps variables and channels with roles to session types. The reduction relation $\rightarrow$ allows $s[p]:S_p,s[q]:S_q$ to “interact”, provided that (1) $S_p$ is a selection towards $q$, (2) $S_q$ is a branching from $p$, and (3) the message labels of $S_q$ are a superset of those in $S_p$, with equivalent carried types. Clause [3] implies that if $S_p$ has an “unreceivable” output message, then $s[p]:S_p,s[q]:S_q$ is stuck. Note that we allow $x:S$ to reduce autonomously: we add this minor extension to simplify our work in §3.

3  Multiparty Session Typing, Beyond Duality and Consistency

We now introduce our new MPST typing system. We start with a coinductive notion of typing context liveness (Def. 3.1), based on the reductions of MPST typing contexts (Def. 2.5): intuitively, it ensures that each selection is matched by a compatible branching, and each branching can receive a value from a compatible selection.

Definition 3.1 (Liveness). The predicate live($\Gamma$) (read “$\Gamma$ is live”) is the largest predicate such that:

- [L-&] live($\Gamma, s[p]:S$) with $S = q \otimes_{i \in I} m_i(S_i).S_i'$ implies $\exists i \in I : \exists \Gamma' : \Gamma, s[p]:S \rightarrow^{*} \Gamma', s[p]:S_i';$
- [L-@] live($\Gamma, s[p]:S$) with $S = q \otimes_{i \in I} m_i(S_i).S_i'$ implies $\forall i \in I : \exists \Gamma' : \Gamma, s[p]:S \rightarrow^{*} \Gamma', s[p]:S_i';$
- [L-µ] live($\Gamma, s[p]:\mu t.S$) implies live($\Gamma, s[p]:S(\mu t,s_i)$);
- [L-→] live($\Gamma$) and $\Gamma \rightarrow \Gamma'$ implies live($\Gamma'$).

Assume some live typing context $\Gamma,s[p]:S$. By clause [L-@] of Def. 3.1, if $S$ is an external choice, then $\Gamma$ must be able to reduce until some branch of $S$ is triggered. By clause [L-@], if $S$ is an internal choice, then $\Gamma$ must be able to reduce allowing to send each message of $S$. Clause [L-µ] says that if $S$ is recursive, its unfolding must be live, too. Clause [L-→] ensures that if a live context reduces, the reduc is live, too.

Notably, a live typing context never deadlocks: i.e., if it cannot further reduce, then all its entries are end-typed. The vice versa is not true: a deadlock-free typing context might not be live. Moreover, liveness and consistency are incomparable, i.e., one does not imply the other, and vice versa.

Example 3.2. The typing context $s[p]:S_p,s[q]:S_q,s[r]:S_r$ in (4) is live, but not consistent. The context $s[p]:q \& m_1.r \oplus m_3,s[q]:r \& m_2.p \oplus m_1,s[r]:p \& m_3,q \oplus m_2$ is consistent but not live (nor deadlock-free): its inputs/outputs occur in the wrong order, thus preventing reductions. Finally, the typing context $s[p]:\mu t.q \oplus m_1.t,s[q]:\mu t.p \& m_2.t,s[r]:p \& m_2$ is deadlock-free but not live: $s[p]$ and $s[q]$ generate infinite reductions, but $s[r]$ cannot possibly perform its input (thus violating clause [L-@] of Def. 3.1).

Definition 3.3 (MPST Typing Judgement). Let $\Theta$ be a partial mapping defined as: $\Theta ::= \emptyset \mid \Theta,X:\tilde{S}$. An MPST typing judgement has the form: $\Theta \vdash P \triangleright \Gamma_g < \Gamma_r$ with live($\Gamma_g,\Gamma_r$), and is inductively defined by the rules in Fig. 2. Above, $\Gamma_g$ and $\Gamma_r$ are called respectively guarantee and rely contexts.

Above, $\Theta$ assigns an n-uple of types to each process variable $X$ (one type per argument). Intuitively, the judgement $\Theta \vdash P \triangleright \Gamma_g < \Gamma_r$ says: given the definitions in $\Theta$, $P$ “guarantees” to use its channels linearly according to $\Gamma_g$, by “relying” on the assumption that other processes use their channels according to $\Gamma_r$. The additional $\Gamma_r$ is the most visible departure from standard MPST typing rules (albeit we subsume them: see p. 4). Note: live($\Gamma_g,\Gamma_r$) implies that $\Gamma_g,\Gamma_r$ must be defined, i.e., dom($\Gamma_g$) \cap dom($\Gamma_r$) = \emptyset (Def. 2.5).

In Fig. 2, the first three rules define auxiliary judgements: end($\Gamma$) holds if all entries of $\Gamma$ are end-typed; $\Gamma \vdash c:S$ holds if $\Gamma$ only contains one entry $c:S'$, with $S' \equiv S$; $\Theta \vdash X:S_1,\ldots,S_n$ holds if $\Theta$ maps $X$ to an n-uple of types equivalent to $S_1,\ldots,S_n$. The typing rule $[\triangleright]$ says that $\Theta$ is typed if all channels in $\Gamma_g$ are end-typed. By $[\triangleright-\Theta]$ def $X(\vec{x}) = P$ in $Q$ is typed if $P$ uses the arguments $x_1,\ldots,x_n$ according to $S_1,\ldots,S_n$ (with empty rely context), and the latter is the type of $X$ when typing $Q$. By $[\triangleright-X]$, $X(\vec{x})$ is typed if the types of $\vec{x}$ match those of the formal parameters of $X$, and any unused channel (in $\Gamma_g$) is
end-typed. By \([\text{T-v}]\), \((vs)P\) is typed if \(P\) can be typed by adding \(\Gamma\) (that annotates \(s\), and only has \(s\)-based entries) to \(\Gamma_s\); note that since live(\(\Gamma_s, \Gamma_r\)), the premise \(\Theta \vdash P \triangleright \Gamma_r, \Gamma_s < \Gamma_r\) implies live(\(\Gamma\)). By \([\text{T-j}]\), \(P_1 \triangleright P_2\) is typed by splitting the guarantee context in the premises (similarly to \([\text{T-\&}]\) in \([\text{T-\&}]\)—but crucially, we record the “lost” information in the rely context, i.e., \(P_1\) is typed by relying on the guarantees of \(P_2\), and vice versa: this allows the typing rules to preserve liveness across a typing derivation. By \([\text{T-\&}]\), \(c[q] \{\&_{c_{d,l,j}} \{m(y_i), P\}\} \triangleright c[q] \{\&_{c_{d,l,j}} \{m(y_i), d\}\} \triangleright \Gamma_s, c:S < \Gamma_r\) is typed if \(c\) has type \(S\), where \(S\) is (a possibly recursive) external choice from \(q\), with message labels \(m_i\) (where \(i \in I\); as usual, the process can have more labels \(m_j\) with \(j \in J\), which are immaterial); further, if, \(\Gamma_r\) reduces to \(\Gamma_s\)' while moving \(c:S\) to \(c:S_s\) (for any \(i \in I\)), then \(\Gamma_s\) and \(\Gamma_s\) must type the continuation \(P_i\), with the received channel \(y_i\) added to the guarantee context — i.e., \(P_i\) must use it correctly. By \([\text{T-\&}]\), \(c[q] \{\&_{c_{d,l,j}} \{m(y_i), d\}\} \triangleright \Gamma_s, c:S < \Gamma_r\) is typed if \(c\) has type \(S\), where \(S\) is (a possibly recursive) internal choice towards \(q\), whose message labels include \(m_k\); further, if \(\Gamma_r\) reduces to \(\Gamma_s\) while moving \(c:S\) to \(c:S_s\), then \(\Gamma_s\) and \(\Gamma_s\) must type the continuation \(P\) — but with the sent channel \(d:S_s\) now transferred into the rely context: i.e., after sending \(d\), \(P\) cannot use it, but relies on its correct usage by the recipient.

Example 3.4. We now revise the typing derivation \([\text{4}]\) in \(\Gamma\) using our rules from Def. 3.3.

We have live(\(\Gamma\)) (see Ex. 3.2), and each application of \([\text{T-v}]\) propagates the entries of \(\Gamma\) in its premises’ guarantee/rely contexts, throughout the derivation. Now, consider the sub-derivation \(\mathcal{D}\) (top left):

Here, the rely context \(s[p]:S_p, s[r]:S_r\) cannot reduce (i.e., in \([\text{T-\&}]\) in Fig. 2, we have \(\Gamma' = \Gamma_s\)). By composing it with \(s[p]:S_p\) (that outputs \(\text{stop}\) like the process) we get the new rely context \(s[p]:S_p, s[q]:s[r]:S_r\); by \([\text{T-\&}]\), it must type the continuation \(0\). Note: in the instance of \([\text{T-\&}]\), the rely context never interacts with \(s[p]\), albeit \(p\) occurs in \(S_p\): this captures the inter-role dependency discussed in §7.

Subject reduction (without surprises) Our typing rules enjoy the Substitution Lemma 3.5 below: if \(P\) is typed with an \(S\)-typed variable \(x\), and a \(\Gamma_r\) containing an \(S\)-typed channel with role \(s[p]\), then \(P[s[p]/x]\) can be typed by (1) removing \(s[p]\) from \(\Gamma_r\), and (2) using it in place of \(x\) in the guarantee context.
Lemma 3.5. Let $\Theta \vdash P \triangleright_{\Gamma_r} x:S \triangleright_{\Gamma_r}$ such that $\Gamma_r \vdash S[P]:S$. Then, $\Theta \vdash P[s[P]/x] \triangleright_{\Gamma_r} x:S \triangleright_{\Gamma_r} \backslash s[P]$.

Lemma 3.5 allows to prove the Subject Reduction Theorem 3.6 below. "As expected" (cf. [5] in §1), it holds for all typed processes. The only condition, by Def. 3.3, is that rely/guarantee contexts must be live (Def. 3.1): this invariant (that replaces consistency) satisfies the requirements [R1] and [R3] seen in §1.

Theorem 3.6. If $\Theta \vdash P \triangleright_{\Gamma_r} \Gamma_r$ and $P \rightarrow^{\ast} P'$, then $\exists \Gamma_r'$ such that $\Gamma_r \rightarrow^{\ast} \Gamma_r'$ and $\Theta \vdash P' \triangleright_{\Gamma_r} \Gamma_r'$.

From “new” to (simpler) “old” rules Rules [T-\&] and [T-\oplus] (Fig. 2) can be demanding: the quantification "\forall \Gamma_r' : \Gamma_r \rightarrow^{\ast} \Gamma_r'" can yield many premises (albeit they are finite: by Def. 2.5, the relation $\rightarrow$ reduces finite-state systems). Lemma 3.7 allows to reduce the number of premises that must be checked explicitly.

Lemma 3.7 (Rely context strengthening). If $\Theta \vdash P \triangleright_{\Gamma_r} \Gamma_r$ and $\Gamma_r \rightarrow^{\ast} \Gamma_r'$, then $\Theta \vdash P \triangleright_{\Gamma_r} \Gamma_r'$. (Note: by Def. 2.1, $x$ cannot be a channel with role)

By Prop. 3.8 variables in $\Gamma_r$ are immaterial—intuitively, because they do not influence liveness (Def. 3.1). Lemma 3.7 and Prop. 3.8 become particularly useful when typing a process that branches/selects from/to a variable—and (in the latter case) sends a variable. In this case, rules [T-\&] and [T-\oplus] become the standard MPST typing rules for branch/input and select/output (see [4]), plus an unused rely context:

$$S \equiv q \{ T_{\text{def}} m_i(S_i) S_i' \} \quad \forall i \in I \quad \Theta \vdash P ; I \triangleright_{\Gamma_r} \Gamma_r, y_i:S_i \triangleright_{\Gamma_r} \Gamma_r \quad S \equiv q \{ T_{\text{def}} m_i(S_i) S_i' \} \quad k \in I \quad \Gamma_{y_k} \triangleright y_k S_k \quad \Theta \vdash P \triangleright_{\Gamma_r} x:S \triangleright_{\Gamma_r} \Gamma_r$$

This is because in [T-\oplus], Prop. 3.8 allows to omit $\Gamma_{y_k}$ from the rely context of the rightmost premise. Further, since $x:S$ reduces autonomously (cf. Def. 2.5), in both [T-\&] and [T-\oplus] the quantification "\forall $\Gamma_r', \Gamma_r''$ : $\Gamma_r \rightarrow^{\ast} \Gamma_r'$ and $\Gamma_r'' \triangleright_{\Gamma_r} \Gamma_r' \rightarrow^{\ast} \Gamma_r''$" amounts to "\forall $\Gamma_r'$ : $\Gamma_r \rightarrow^{\ast} \Gamma_r'$ and $\Gamma_r'' \triangleright_{\Gamma_r} \Gamma_r' \rightarrow^{\ast} \Gamma_r''"; hence, by Lemma 3.7 checking the rightmost premise once with $\Gamma_r$ is sufficient to know that it holds \forall $\Gamma_r'$ such that $\Gamma_r \rightarrow^{\ast} \Gamma_r'$.

4 Conclusions and Discussion

We presented a new MPST typing system (Def. 3.3) providing subject reduction (Theorem 3.6) for a larger set of processes w.r.t. existing works. We record “global” typing information in a “rely context”, thus removing the consistency/duality restrictions of previous MPST works.

On inter-role dependencies For the first time, we prove type safety of processes implementing choreographies with delegation and complex inter-role dependencies. (For more examples and discussion, in addition to §1 and Ex. 3.4, see Ex. A.6 and §B). In this respect, to the best of our knowledge, the only comparable work is [6]; however, its process calculus only supports one session, and the paper crucially exploits this restriction to type parallel compositions without “splitting” them (cf. Table 8, rule [T-SESS]) and track a global type along reductions. Hence, unlike the present work, [6] does not support multiple sessions and delegation — and extending it to support them appears challenging. Moreover, we argue that our approach is simpler, since it does not depend on global types and makes them orthogonal w.r.t. typing rules.

On global types and liveness Our type system does not assume a global type for each session: it only (implicitly) requires that its local types are live (Def. 3.1). By rule [T-V] (Fig. 2). Def. 3.1 is inspired by the notion of “safety” for Communicating Finite-State Machines (CFSMs) [2] (no deadlocks, orphan messages, unspecified receptions) and the idea of “MPSTs as CFSMs” [5]—that is not captured by previous MPST works, due to their consistency clause. The synchronous typing context reduction (Def. 2.5) implies decidable liveness; we plan to study the asynchronous setting. When needed, a set of local types
can be related to a choreography either via “top-down” projection (§1), or “bottom-up” synthesis [8]. Similarly to most previous MPST works, our approach ensures that a typed process \((\nu s)(\mid p \in I p_p)\), with each \(P_p\) only interacting on \(s[p]\), is deadlock-free—but does not guarantee deadlock freedom in presence of multiple interleaved sessions [1][4]: we leave this topic as future work.

**Relation with “standard” MPST typing**  As shown above, our typing rules become the “standard” ones if processes only use variables. We think that, by distinguishing a “static” process syntax (with variables only) from a “runtime” syntax (with variables and channels with roles), we could combine the “standard” typing rules for the former, with our typing rules for the latter (as required by Theorem 3.6).

**Encodability** [9] proves that consistency [7] creates a strong link between MPST and linear \(\pi\)-calculus (with binary channels), allowing to encode the former into the latter, and naturally guiding MPST implementations. The encodability of our “consistency-free” MPST processes (and their implementation) is now an open problem. Also, typed behavioural congruences and bisimulations need to be investigated.

**Acknowledgements**  We thank the anonymous reviewers for their valuable comments and suggestions.

**References**


**A Auxiliary Definitions and Details**

This appendix contains several definitions taken from MPST literature, and further explanations on the example discussed in §1 (see Ex. A.6).

### A.1 Global, Local and Partial Session Types

**Definition A.1 (Global Types).** The syntax of a global type \(G\) is:

\[
G ::= p\rightarrow q; \{m_i(T_i), G_i\}_{i \in I} \mid \mu t.G \mid t \end
\]

where \(p \neq q\), \(I \neq \emptyset\), and \(\forall i \in I : fv(T_i) = \emptyset\)
The global and local type projections in Def. A.2 and Def. A.5 below, with their respective merging operators, are based on [11, 10]. Compared with simpler projection/merging definitions (e.g., those in [3]), they allow for more global types to be projectable, and more typing contexts to be deemed consistent. Intuitively, the projection and merging in Def. A.2 below allow to “reconstruct” the expected behaviour of a role in a choreography, by collecting and combining its internal/external choices along different execution paths.

**Definition A.2 (Projection of a Global Type).** The projection of \( G \) onto role \( p \), written \( G \mid p \), is:

\[
(q \mapsto r \cdot \{m_i(T_i \cdot G_i)\}) \mid p = \begin{cases} 
\tau \oplus \cup_i m_i(T_i) \cdot (G_i \mid p) & \text{if } p = q \\
\sigma \cup_i m_i(T_i) \cdot (G_i \mid p) & \text{if } p = r \\
\cap_i G_i \mid p & \text{if } q \neq p \neq r
\end{cases}
\]

\[
(\mu t. G) \mid p = \begin{cases} 
\mu t. (G \mid p) & \text{if } G \mid p \neq t' \ (\forall t') \\
\text{end} & \text{otherwise}
\end{cases}
\]

where \( \cap \) is the merge operator for (local) session types, defined as:

\[
p \cup \cup_i m_i(S_i) \cdot S_i' \cap p \cup \cup_i m_i(S_i) \cdot T_i' = p \cup \cup_i m_i(S_i) \cdot (S_i' \cap T_i') & \cap p \cup \cup_i m_i(S_i) \cdot S_i' & \cup p \cup \cup_i m_i(S_i) \cdot S_i'
\]

\[
(\mu t. S \cap \mu t. T) \cap t = t & \text{end} \cap \text{end} = \text{end}
\]

**A.2 Partial Session Types and Duality**

A session type \( S \) can be further projected onto a role \( p \), to “isolate” the interactions of \( S \) involving \( p \) from those involving other roles (Def. A.3 below). The result is a partial session type (Def. A.5 below). Partial session types feature branches, selections and recursion — but no role annotations: therefore, they are similar to binary session types (except that their payload types are multiparty), and endow a notion of duality (Def. A.4 below). As shown in §1, these technicalities are necessary to define consistency (cf. (7)).

**Definition A.3 (Partial Session Types).** Partial session types, ranged over by \( H \), have the syntax:

\[
H ::= \cup_i m_i(S_i) \cdot H_i | \cup \cup_i m_i(S_i) \cdot H_i | (\mu t. H) | t \quad \text{with } 1 \neq \emptyset \text{ and } \forall i \in 1 : \text{fv}(S_i) = \emptyset
\]

where \( m_i \) range over pairwise distinct message labels.

**Definition A.4 (Duality of Partial Session Types).** The dual of a partial type \( H \), written \( \bar{H} \), is:

\[
\bar{\cup_i m_i(S_i) \cdot H_i} = \cup_i m_i(S_i) \cdot H_i & \quad \bar{\cup \cup_i m_i(S_i) \cdot H_i} = \cup \cup_i m_i(S_i) \cdot H_i & \quad \bar{\mu t. H} = \mu t. H & \quad \bar{t} = t & \bar{\text{end}} = \text{end}
\]

We say that \( H \) and \( H' \) are dual iff \( \bar{H} = H' \).

**Definition A.5 (Partial Projection).** The projection of \( S \) onto \( p \), written \( S \mid p \), is a partial type defined as:

\[
(q \cup \cup_i m_i(S_i) \cdot S_i') \mid p = \begin{cases} 
\cup_i m_i(S_i) \cdot (S_i' \mid p) & \text{if } p = q \\
\cup \cup_i S_i' \mid p & \text{if } p \neq q
\end{cases}
\]

\[
(\mu t. S) \mid p = \begin{cases} 
\mu t. (S \mid p) & \text{if } S \mid p \neq t' \ (\forall t') \\
\text{end} \mid p = \text{end}& \text{otherwise}
\end{cases}
\]

where \( \cap \) is the merge operator for partial session types, defined as:

\[
\cup_i m_i(S_i) \cdot H_i \cap \cup_i m_i(S_i) \cdot H_i' = \cup_i m_i(S_i) \cdot (H_i \cap H_i') & \cup \cup_i m_i(S_i) \cdot H_i \cap \cup \cup_i m_i(S_i) \cdot H_i' = \cup \cup_i m_i(S_i) \cdot (H_i \cap H_i') & \mu t. H \cap \mu t. H' = \mu t. (H \cap H')
\]

\[
\text{end} \cap t = t & \text{end} \cap \text{end} = \text{end}
\]

---

1These merging operators introduce a subtle discrepancy between consistency and subject reduction, as stated in (6). This does not impact our treatment, since we lift the consistency requirement. For more details, see [9] §2.1, “On Consistency”).
Note that Def. A.5 allows to merge different internal choices, but not different external choices. In the mechanics of “standard” MPST typing systems, this guarantees that a consistent typing context never reduces to configurations where an output from $p$ to $q$ is not matched by a corresponding input.

**Example A.6.** Take $S_p$, $S_q$ and $S_r$ from (7) in §7. By Def. A.5 these partial projections are defined:

$$S_p | q = \{ m1(\text{Int}), \text{stop} \} \quad S_q | p = \{ q \{ m1(\text{Int}), \text{stop} \} \}
$$

Moreover, by Def. A.4 we have $S_p | q = S_q | p$ — i.e., the projections are dual. Therefore, by (7), the pair $s[p] : S_p, s[q] : S_q$ is consistent.

Now, consider the following projections, that are not defined:

$$S_p | r = r \& m3(\text{Bool}) \notin r \quad \text{end} | r = \& m3(\text{Bool}) \notin \text{end} \quad \text{(undefined)}
$$

and therefore, by (7), the pair $s[p] : S_p, s[r] : S_r$ is not consistent.

The reason for $S_p | r$ (resp. $S_q | p$) being undefined is that the session type $S_p$ (resp. $S_q$) starts with an interaction with $q$ — and thus, when projecting the type onto $r$ (resp. $p$), Def. A.5 requires to “skip” the interaction with $q$, project the continuations onto $r$ (resp. $p$), and merge them. The merge operation between partial types, however, is quite restrictive: it does not allow to combine an branching (resp. selection) with end. These restrictions are in place to rule out erroneous communications (e.g., outputs from $p$ to $r$ that are not matched by corresponding input capabilities) that could be otherwise accepted by the “standard” MPST typing rules. The drawback is that the interaction between $p$ (resp. $r$) and $q$ can only have a limited influence on the interactions between $p$ (resp. $r$) and other roles. This curtails the variety of inter-role dependencies supported by the “standard” MPST theory: see §7 and §B.

### A.3 MPST Process Calculus

**Definition A.7 (MPST $\pi$-Calculus Congruence).** Let us define the process declaration $D$ as:

$$D ::= \ X(\overline{x}) = P$$

so that process definitions in Fig. 7 can be written as $\text{def } D \text{ in } Q$. We write $\text{fc}(P)$ for the free channels with roles in $P$, $\text{dpv}(D)$ for the process variables declared in $D$, $\text{fpv}(D)$ for the free process variables in $D$, and $\text{fpv}(P)$ for the free process variables in $P$. We write $s \notin \text{fc}(P)$ to mean that there does not exist a $p$ such that $s[p] \in \text{fc}(P)$. We define the structural congruence $\equiv$ as the smallest relation such that:

$$\begin{align*}
P | Q & \equiv Q | P \quad (P | Q) | R \equiv P | (Q | R) \quad P | 0 \equiv P \quad (v s) | 0 \equiv (v s) | 0 \quad (v s') | P \equiv (v s') | (v s) | P
\end{align*}$$

$$\begin{align*}
(P | Q) & \equiv P | (v s) Q \quad \text{if } s \notin \text{fc}(P) \quad \text{def } D \text{ in } \theta \equiv \theta \quad \text{def } D \text{ in } (v s) P \equiv (v s) | \text{def } D \text{ in } P \quad \text{if } s \notin \text{fc}(P)
\end{align*}$$

$$\begin{align*}
\text{def } D \text{ in } (P | Q) & \equiv \text{def } D \text{ in } P | Q \quad \text{if } \text{dpv}(D) \cap \text{fpv}(Q) = \emptyset \quad \text{def } D \text{ in } (\text{def } D' \text{ in } P) & \equiv \text{def } D' \text{ in } (\text{def } D \text{ in } P) \quad \text{if } (\text{dpv}(D) \cup \text{fpv}(D')) \cap \text{dpv}(D) = \emptyset
\end{align*}$$

### B More “Correct” but Non-Consistent Examples

We now reprise the projection/merging/consistency restrictions discussed in §1 and Ex. A.6. We present more examples of “correct” choreographies that, when projected, yield typing contexts that are not consistent according to (7) — and therefore, do not provide subject reduction guarantees under the “standard” MPST theory. However, all these examples are accepted by our theory (i.e., can provide subject reduction guarantees, by Theorem 3.6) because they all yield live typing contexts (by Def. 3.1).
B.1 Consistency vs. History-Dependent Branching

In Ex.A.6, consistency does not hold because the partial projections \( S_p \mid r \) and \( S_r \mid p \) are undefined, due to the non-mergeability of internal/external choice and \texttt{end} (as per Def.A.5). We now try to “adjust” \( G \) in \([\text{I}]\) and make it consistent, by adding another communication between \( r \) and \( p \):

\[
G' = p \rightarrow q \{ \begin{align*}
&m1(\text{Int}). q \rightarrow r \cdot m2(\text{Str}). r \rightarrow p \cdot m3(\text{Bool}) \cdot \texttt{end}, \\
& \text{stop} \cdot q \rightarrow r \cdot \text{quit} \cdot r \rightarrow p \cdot m4(\text{Int}) \cdot \texttt{end}
\end{align*} \}
\]

The projections of \( G' \) onto \( p, q \) and \( r \) are respectively:

\[
\begin{align*}
S'_p &= q \oplus \{ m1(\text{Int}). r \& m3(\text{Bool}) \cdot \text{stop} \cdot r \& m4(\text{Int}) \} \\
S'_q &= p \& \{ m1(\text{Int}). r \& m2(\text{Str}) \cdot \text{stop} \cdot r \& \texttt{quit} \} \\
S'_r &= q \& \{ m2(\text{Str}) \cdot p \& m3(\text{Bool}) \cdot \text{stop} \cdot p \& m4(\text{Int}) \}
\end{align*}
\]

Let us now examine the partial projections \( S'_p \mid r \) and \( S'_r \mid p \):

\[
\begin{align*}
S'_p \mid r &= r \& m3(\text{Bool}) \mid r \cap r \& m4(\text{Int}) \mid r = \& m3(\text{Bool}) \cap \& m4(\text{Int}) \quad \text{(undefined)} \\
S'_r \mid p &= p \& m3(\text{Bool}) \mid p \cap p \& m4(\text{Int}) \mid p = \& m3(\text{Bool}) \cap \& m4(\text{Int}) = \& \{ m3(\text{Bool}), m4(\text{Int}) \}
\end{align*}
\]

As noted in the comment of Def.A.5, partial type merging does not allow to combine different external choices, thus making \( S'_p \mid r \) undefined. Hence, external choices cannot depend on the outcome of previous interactions with other roles: i.e., \( p \) cannot wait for different messages from \( r \), depending on the interaction between \( p \) and \( q \). However, different \textit{internal} choices can be merged: e.g., \( r \) can send a different message to \( p \), depending on the previous interaction between \( r \) and \( q \). Hence, \( S'_r \mid p \) is defined.

B.2 Consistency vs. Recursion

Consider the global type: \( G'' = \mu t. p \rightarrow q \{ \begin{align*}
&m1(\text{Int}). q \rightarrow r \cdot m2(\text{Str}) \cdot r \rightarrow p \cdot m3(\text{Bool}) \cdot \texttt{t}, \\
& \text{stop} \cdot q \rightarrow r \cdot \texttt{quit} \cdot r \rightarrow p \cdot m3(\text{Bool}) \cdot \texttt{end}
\end{align*} \}
\]

The global type \( G'' \) above is again similar to \( G \) in \([\text{I}]\) — but now, if \( p \) sends \( m1(\text{Int}) \) to \( q \), the choreography loops: in fact, \( q \) then sends \( m2(\text{Str}) \) to \( r \), which sends \( m3(\text{Bool}) \) to \( p \), and the choreography \texttt{ends}. The projections of \( G'' \) onto \( p, q \) and \( r \) are respectively:

\[
\begin{align*}
S''_p &= \mu t. q \oplus \{ m1(\text{Int}). r \& m3(\text{Bool}) \cdot \texttt{t} \cdot \text{stop} \cdot r \& m3(\text{Bool}) \cdot \texttt{end} \} \\
S''_q &= \mu t. p \& \{ m1(\text{Int}). r \& m2(\text{Str}) \cdot \texttt{t} \cdot \text{stop} \cdot r \& \texttt{quit} \cdot \texttt{end} \} \\
S''_r &= \mu t. q \& \{ m2(\text{Str}) \cdot p \& m3(\text{Bool}) \cdot \texttt{t} \cdot \texttt{quit} \cdot p \& m3(\text{Bool}) \cdot \texttt{end} \}
\end{align*}
\]

Note that in all cases, \( r \) sends to \( p \) the message \( m3(\text{Bool}) \): this circumvents the merging problems described in Ex.A.6 and [B.1]. However, we stumble into another roadblock:

\[
\begin{align*}
S''_p \mid r &= \mu t. ((r \& m3(\text{Bool}) \cdot \texttt{t}) \mid r \cap (r \& m3(\text{Bool}) \cdot \texttt{end}) \mid r) = \mu t. (\& m3(\text{Bool}) \cdot (\texttt{t} \cap \texttt{end})) \quad \text{(undefined)} \\
S''_r \mid p &= \mu t. ((p \& m3(\text{Bool}) \cdot \texttt{t}) \mid p \cap (p \& m3(\text{Bool}) \cdot \texttt{end}) \mid p) = \mu t. (\& m3(\text{Bool}) \cdot (\texttt{t} \cap \texttt{end})) \quad \text{(undefined)}
\end{align*}
\]

This is another case of (unsupported) inter-role dependency: in \( S''_p \) (resp. \( S''_r \)), the interaction with \( q \) determines whether, after receiving (resp. sending) \( m3(\text{Bool}) \) from \( r \) (resp. to \( p \)), the type will \( (a) \) loop on \( t \) and, thus, keep interacting with \( r \) (resp. \( p \)), or \( (b) \) \texttt{end} the session. Hence, the partial projection \( S''_p \mid r \) (resp. \( S''_r \mid p \)) tries to merge \( t \) (from case \( (a) \)) with \texttt{end} (from case \( (b) \)) — but the operation is undefined.

Additional references for the Appendix


C Proofs

Proposition C.1. If \( \text{live}(\Gamma) \) and \( \Gamma \rightarrow^* \Gamma' \), then \( \text{live}(\Gamma') \).

Proof. By induction on the number of reductions in \( \Gamma \rightarrow^* \Gamma' \). The base case (0 reductions) is trivial, as it implies \( \Gamma' = \Gamma \). The inductive case \((n+1)\) reductions) is proved by the induction hypothesis, and clause \([L\rightarrow]\) of Def.3.1 (liveness).

Proposition C.2. For all \( \Gamma, \Gamma', \Gamma_0 \) such that \( \text{dom}(\Gamma) \cap \text{dom}(\Gamma_0) = \emptyset \): \( \Gamma \rightarrow \Gamma' \) if and only if \( \Gamma, \Gamma_0 \rightarrow \Gamma', \Gamma_0 \), with \( \Gamma_0 \) not reducing.

Proof. (\( \implies \)) By induction on the size of \( \Gamma_0 \), and by Def.2.5.

(\( \impliedby \)) By induction on the size of \( \Gamma_0 \), and by Def.2.5.

Proposition C.3. For all \( \Gamma, \Gamma', \Gamma_0 \) such that \( \text{dom}(\Gamma) \cap \text{dom}(\Gamma_0) = \emptyset \): \( \Gamma \rightarrow \cdots \rightarrow \Gamma' \) if and only if \( \Gamma, \Gamma_0 \rightarrow \cdots \rightarrow \Gamma', \Gamma_0 \), with \( \Gamma_0 \) not reducing.

Proof. By induction on the number of reductions, using (C.2) in the inductive case.

Proposition C.4. Assume \( \text{live}(\Gamma_0) \). Then, \( \Gamma, \Gamma_0 \rightarrow \Gamma' \) implies \( \Gamma' = \Gamma'', \Gamma_0'' \) such that either:

(a) \( \Gamma'' = \Gamma \) and \( \Gamma_0 \rightarrow \Gamma_0'' \), or

(b) \( \Gamma' \rightarrow \Gamma'' \) and \( \Gamma_0'' = \Gamma_0 \).

Proof. By Def.3.1, noticing that we must have \( \text{dom}(\Gamma) \cap \text{dom}(\Gamma_0) = \emptyset \) (otherwise, \( \Gamma, \Gamma_0 \) would be undefined by Def.2.5); and since \( \Gamma_0 \) is live, it cannot interact with \( \Gamma \).

Proposition C.5. Assume \( \text{live}(\Gamma_0) \). Then, \( \Gamma, \Gamma_0 \rightarrow \cdots \rightarrow \Gamma' \) implies \( \Gamma' = \Gamma'', \Gamma_0'' \) such that, for some \( m \leq n \), \( \Gamma \rightarrow \cdots \rightarrow \Gamma'' \) and \( \Gamma_0 \rightarrow \cdots \rightarrow \Gamma_0'' \); moreover, \( \Gamma, \Gamma_0 \rightarrow \cdots \rightarrow \Gamma'', \Gamma_0 \).

Proof. By induction on \( n \), using Prop. C.4. The “moreover…” part of the statement is an immediate consequence of the first part.

Proposition C.6. Assume \( \text{live}(\Gamma_0) \) and \( \text{dom}(\Gamma) \cap \text{dom}(\Gamma_0) = \emptyset \). Then, \( \text{live}(\Gamma, \Gamma_0) \) if and only if \( \text{live}(\Gamma) \).

Proof. (\( \implies \)) We show that the following predicate:

\[ P = \{ \Gamma \mid \text{live}(\Gamma, \Gamma_0) \} \]

satisfies the clauses of Def.3.1 (liveness). We proceed by examining each \( \Gamma' \in P \), reminding that:

\[ \text{dom}(\Gamma) \cap \text{dom}(\Gamma_0) = \emptyset \quad \text{(by hypothesis)} \]

We have the following (non-mutually exclusive) cases:
\[ \Gamma = \Gamma', s[p]; S \text{ with } S = q \cup \bigcup_{i \in I} m_i(S_i), S' \]. We have:

\[
\begin{align*}
\text{live}(\Gamma, \Gamma_0) \quad & \quad \text{and thus \ live}(\Gamma', s[p]; S, \Gamma_0) \quad \text{(by (8))} \quad (10) \\
\forall i \in I : \exists \Gamma'' : \Gamma', s[p]; S, \Gamma_0 \rightarrow^* \Gamma''', s[p]; S_i', \Gamma_0 \quad \text{(by (10), [L-\&])} \quad (11) \\
\forall i \in I : \exists \Gamma'' : \Gamma', s[p]; S \rightarrow^* \Gamma''', s[p]; S'_i \quad \text{(by (11) and Prop. C.5)} \quad (12)
\end{align*}
\]

and by the shape of \( \Gamma \) and (12), we conclude that \( \Gamma \) satisfies clause [L-\&];

\[ \Gamma = \Gamma', s[p]; S \text{ with } S = q \cup \bigcup_{i \in I} m_i(S_i), S'_i. \] We have:

\[
\begin{align*}
\text{live}(\Gamma, \Gamma_0) \quad & \quad \text{and thus \ live}(\Gamma', s[p]; S, \Gamma_0) \quad \text{(by (8))} \quad (13) \\
\exists i \in I : \exists \Gamma'' : \Gamma', s[p]; S, \Gamma_0 \rightarrow^* \Gamma''', s[p]; S'_i, \Gamma_0 \quad \text{(by (13), [L-\&] and Prop. C.5)} \quad (14) \\
\exists i \in I : \exists \Gamma'' : \Gamma', s[p]; S \rightarrow^* \Gamma''', s[p]; S'_i \quad \text{(by (14) and Prop. C.3)} \quad (15)
\end{align*}
\]

and by the shape of \( \Gamma \) and (15), we conclude that \( \Gamma \) satisfies clause [L-\&];

\[ \Gamma = \Gamma', s[p]; \mu S. \] We have:

\[
\begin{align*}
\text{live}(\Gamma, \Gamma_0) \quad & \quad \text{and thus \ live}(\Gamma', s[p]; \mu S, \Gamma_0) \quad \text{(by (8))} \quad (16) \\
\text{live}(\Gamma', s[p]; S[\mu S/\{i\}], \Gamma_0) \quad \text{(by (16) and [L-\mu])} \quad (17) \\
\Gamma', s[p]; S[\mu S/\{i\}] \in \mathbb{P} \quad \text{(by (17) and (8))} \quad (18)
\end{align*}
\]

and by the shape of \( \Gamma \) and (18), we conclude that \( \Gamma \) satisfies clause [L-\mu];

\[ \Gamma \rightarrow \Gamma'. \] We have:

\[
\begin{align*}
\Gamma, \Gamma_0 \rightarrow \Gamma', \Gamma_0 \quad & \quad \text{(by (9) and Prop. C.2)} \quad (19) \\
\text{live}(\Gamma', \Gamma_0) \quad & \quad \text{(by (8))} \quad (20) \\
\text{live}(\Gamma', \Gamma_0) \quad & \quad \text{(by (20), (19) and [L-\rightarrow])} \quad (21) \\
\Gamma' \in \mathbb{P} \quad & \quad \text{(by (21) and (8))} \quad (22)
\end{align*}
\]

and by the hypothesis \( \Gamma \rightarrow \Gamma' \) and (22), we conclude that \( \Gamma \) satisfies clause [L-\rightarrow].

We have proved that \( \mathbb{P} \) satisfies all clauses of Def. 3.1 (liveness). Since \( \text{live}(\cdot) \) is the largest predicate satisfying such clauses, we have that \( \forall \Gamma : \Gamma \in \mathbb{P} \implies \text{live}(\Gamma) \); and since \( \forall \Gamma : \text{live}(\Gamma, \Gamma_0) \implies \Gamma \in \mathbb{P} \), we conclude that \( \forall \Gamma : \text{live}(\Gamma, \Gamma_0) \implies \text{live}(\Gamma) \).

( \leftrightarrow \quad ) We show that the following predicate:

\[ \mathbb{P} = \{ \Gamma, \Gamma_0 \mid \text{live}(\Gamma) \} \quad (23) \]

satisfies the clauses of Def. 3.1 (liveness). We proceed similarly to the case above ( \( \quad \implies \quad \)) by examining each \( \Gamma, \Gamma_0 \in \mathbb{P} \). Then, we know that \( \text{live}(\cdot) \) is the largest predicate satisfying all clauses of Def. 3.1 (liveness); therefore, we have that \( \forall \Gamma : \Gamma \in \mathbb{P} \implies \text{live}(\Gamma) \); and since \( \forall \Gamma : \text{live}(\Gamma) \implies \Gamma \in \mathbb{P} \), we conclude that \( \forall \Gamma : \text{live}(\Gamma) \implies \text{live}(\Gamma, \Gamma_0) \).

\[ \square \]

**Proposition C.7.** If \( \text{end}(\Gamma) \), then \( \text{live}(\Gamma) \).

**Proof.** By Def. 3.3 we know that \( \forall c \in \text{dom}(\Gamma) : \Gamma(c) = \text{end} \); therefore, \( \Gamma \) (vacuously) satisfies clauses [L-\&], [L-\&] and [L-\mu] of Def. 3.1 (liveness). Furthermore, by Def. 2.5 (reductions of \( \Gamma \)), we have \( \Gamma \rightarrow^* \); hence, \( \Gamma \) also (vacuously) satisfies clause [L-\rightarrow] of Def. 3.1 (liveness). We conclude \( \text{live}(\Gamma) \). \( \square \)
**Proposition C.8.** \( \text{live}(\Gamma, x; S) \) if and only if \( \text{live}(\Gamma) \).

*Proof.* Straightforward by Def. 3.1 (liveness), noticing that \( x; S \) is not relevant for the definition (which only considers channels with roles). \( \square \)

**Proposition 3.8.** \( \Theta \vdash P \triangleright \Gamma_g \triangleleft \Gamma, x; S \) iff \( \Theta \vdash P \triangleright \Gamma_g \triangleleft \Gamma_r \). (*Note: by Def. 2.1, \( x \) cannot be a channel with role*)

*Proof.* By induction on the derivation of \( \Theta \vdash P \triangleright \Gamma_g \triangleleft \Gamma, x; S \), using Prop. C.8 in the base cases. \( \square \)

**Corollary C.9.** \( \Theta \vdash P \triangleright \Gamma_g \triangleleft \Gamma_r \) iff \( \Theta \vdash P \triangleright \Gamma_g \triangleleft \Gamma_r \setminus x \)

*Proof.* If \( x \notin \text{dom}(\Gamma_r) \), then the statement holds vacuously. Otherwise, if \( x \in \text{dom}(\Gamma_r) \), it is a direct consequence of Prop. 3.8. \( \square \)

**Lemma 3.5.** Let \( \Theta \vdash P \triangleright \Gamma_g \triangleleft \Gamma_r \), such that \( \Gamma_r \vdash s[p]; S \). Then, \( \Theta \vdash P \{ s[p]/x \} \triangleright \Gamma_g, s[p]; S \triangleleft \Gamma_r \setminus s[p] \).

*Proof.* Assume the hypotheses. We have:

1. \( \Theta \vdash P \triangleright \Gamma_g, x; S \triangleleft \Gamma_r \) (by hypothesis) \( \quad \) (24)
2. \( \Gamma_r \vdash s[p]; S \) (by hypothesis) \( \quad \) (25)
3. \( \text{live}(\Gamma_g, \Gamma_r) \) (by hypothesis (well-formedness) and Prop. C.8) \( \quad \) (26)

Note that, in the judgement in the conclusion, the liveness requirement of the guarantee/rely contexts (Def. 3.3) is trivially satisfied:

\[ \text{live}(\Gamma_g, s[p]; S, (\Gamma_r \setminus s[p])) \quad \text{(by } (25), \text{ Def. 3.3 and Def. 3.1) } \quad (27) \]

We proceed by induction on the derivation of \( \Theta \vdash P \triangleright \Gamma_g, x; S \triangleleft \Gamma_r \):

- **base case \([T-\emptyset]\).** We have:

  \[
  P = \emptyset \quad \text{and} \quad \text{end}(\Gamma_g, x; S) \quad \text{(from the rule conclusion and premises)} \quad (28)
  \]

  \[
  S = \text{end} \quad \text{(by } (28) \text{ and Def. 3.3) } \quad (29)
  \]

  \[
  \text{end}(\Gamma_g) \quad \text{(by } (28) \text{ and Def. 3.3) } \quad (30)
  \]

  \[
  \text{end}(\Gamma_g, s[p]; S) \quad \text{(by } (30), (29) \text{ and Def. 3.3) } \quad (31)
  \]

  \[
  P \{ s[p]/x \} = \emptyset \quad \text{and} \quad s[p]/x \quad = \emptyset \quad \text{(from } (28)\text{) } \quad (32)
  \]

  \[
  \Theta \vdash P \{ s[p]/x \} \triangleright \Gamma_g, s[p]; S \triangleleft \Gamma_r \setminus s[p] \quad \text{[T-\emptyset]} \quad \text{(by } (31), (32) \text{ and } (27)\text{) } \quad (33)
  \]

- **base case \([T-X]\).** We have:

  \[
  \Theta \vdash X:S_1, \ldots, S_n \quad \text{and} \quad \varnothing \vdash (\Gamma_0) \quad \forall i \in 1..n \quad \Gamma_i \vdash c_i; S_i \quad \text{[T-X]} \quad \text{(from the rule definition) } \quad (33)
  \]

  \[
  P = X(c_1, \ldots, c_n) \quad \text{(from the conclusion of } (33)\text{) } \quad (34)
  \]

  \[
  \Gamma_g, x; S = \Gamma_g_0, \Gamma_g_1, \ldots, \Gamma_g_n \quad \text{(by } (33)\text{) } \quad (35)
  \]

Now, from (35) and by Def. 3.3, we have two cases:
- $\Gamma_{g0} \vdash x.S$. Then:

\[
\begin{align*}
\Gamma_{g0} &= x.S \quad \text{and} \quad S \equiv \text{end} \quad \text{(by [33] and Def. [33])} \quad (36) \\
\text{end}(s[p];S) \quad \text{by [36] and Def. [33]} \quad (37) \\
\forall i \in 1..n: \quad c_i \neq x \quad \text{(by [36], [35] and Def. [2.5])} \quad (38) \\
P\{s[p]/x\} = P \quad \text{(by [34] and [38])} \quad (39)
\end{align*}
\]

\[\Theta \vdash X: S_1, \ldots, S_n \quad \text{end}(s[p];S) \quad \forall i \in 1..n \quad \Gamma_{gi} \vdash c_i;S_i \quad \text{[T-X]} \quad \text{(by [36], [37], [39])} \]

- $\Gamma_{gk} \vdash x.S$, for some $k \in 1..n$. Then:

\[
\begin{align*}
\Gamma_{gk} &\equiv x.S \quad \text{(by [33])} \quad (40) \\
c_k = x \quad \text{and} \quad \forall i \in 1..n: \quad i \neq k \text{ implies } c_i \neq x \quad \text{(by [35], [40] and Def. [2.5])} \quad (41) \\
P\{s[p]/x\} = X\{c_1, \ldots, c_{k-1}, s[p], c_{k+1}, \ldots, c_n\} \quad \text{(by [34] and [41])} \quad (42)
\end{align*}
\]

\[\Theta \vdash X: S_1, \ldots, S_n \quad \text{end}(\Gamma_{g0}) \quad s[p];S \vdash s[p];S \quad \forall i \in \{1..n\} \setminus \{k\} \quad \Gamma_{gi} \vdash c_i;S_i \quad \Theta \vdash P\{s[p]/x\} \triangleright \Gamma_{g};s[p];S \triangleleft \Gamma_{g}\backslash s[p] \quad \text{[T-X]} \quad \text{(by [40], [42])} \]

* inductive case [T-\&]. In this case, $P$ is a branching on some variable. We have two cases:

(a) $P$ is a branching on some $y \neq x$. Then, the statement holds by the induction hypothesis, and rule [T-\&]; or

(b) $P$ is a branching on $x$, i.e., $x$ is substituted by channel with role $s[p]$.

We detail the proof for the latter (and most interesting) case. We have:

\[
P = x[q] \&_{I \in I \cup J} \{m_i(y_i), P_i\} \quad \wedge \quad S \equiv q \&_{I \in I \cup J} m_i(S_i).S'_i \quad \text{(from rule conclusion/premises)} \quad (43)
\]

\[
\Gamma_r \vdash s[p];S = s[p];q \&_{I \in I \cup J} m_i(S_i).S'_i \quad \text{(by [25] and [43])} \quad (44)
\]

\[
\forall i \in I: \quad \forall \Gamma_r', \Gamma_r': \quad \Gamma_r \rightarrow \Gamma_r' \quad \text{and} \quad \Gamma_r', x:S \rightarrow \Gamma_r'', x:S_i' \quad \Gamma_r'' = \Gamma_r' \quad \text{(by Def. [2.5])} \quad (45)
\]

\[
\Theta \vdash P_i \triangleright \Gamma_{g_i};y_i;S_i;x:S_i' \triangleleft \Gamma_r' \quad \text{(from the rule premises and [45])} \quad (46)
\]

\[
\forall i \in I: \quad \forall \Gamma_r', \Gamma_r'' \quad \text{such that} \quad \left\{ \begin{array}{l}
\Gamma_r\backslash s[p] \rightarrow^* \Gamma_r' \\
\Gamma_r', s[p];S \rightarrow \Gamma_r'', s[p];S'_i \quad \text{and} \\
\Gamma_r'', s[p];S'_i;S \rightarrow \Gamma_r'', s[p];S'_i;S';S'_i \\
\end{array} \right. \quad \Theta \vdash P_i \triangleright \Gamma_{g_i};y_i;S_i;x:S_i' \triangleleft \Gamma_r'' \quad \text{(by [46])} \quad (47)
\]

\[
\Theta \vdash P_i \{s[p]/x\} \triangleright \Gamma_{g_i};y_i;S_i;S'_i \triangleleft \Gamma_r'' \quad \text{(by [47], [48] and i.h.)} \quad (49)
\]

\[
S \equiv q \&_{I \in I \cup J} m_i(S_i).S'_i \quad \forall i \in I \\
\forall \Gamma_r', \Gamma_r'': \quad \Gamma_r\backslash s[p] \rightarrow^* \Gamma_r' \quad \text{and} \quad \Gamma_r', s[p];S \rightarrow \Gamma_r'', s[p];S'_i \\
\Theta \vdash P_i \{s[p]/x\} \triangleright \Gamma_{g_i};y_i;S_i;S'_i \triangleleft \Gamma_r'' \quad \Theta \vdash s[p]\{q \&_{I \in I \cup J} m_i(y_i).P_i\{s[p]/x\}\} \triangleright \Gamma_{g_i};s[p];S \triangleleft \Gamma_{g_i}\backslash s[p] \quad \text{[T-\&]} \quad \text{(by [43], [49])}
\]

* inductive case [T-\&]. In this case, $P$ is a selection on some variable. We have two cases:
(a) \( P \) is a selection on some \( y \neq x \). Then, the statement holds by the induction hypothesis, and rule \( \text{[T-\theta]} \); or

(b) \( P \) is a selection on \( x \), i.e., \( x \) is substituted by channel with role \( s[p] \).

We detail the proof for the latter (and most interesting) case. We have:

\[
\exists k \in I: P = x[q] \oplus m_k(d) \cdot P \land S = q \oplus \bigoplus_{i \in I} c_i(S_i) \cdot S_i' \quad \text{(from rule conclusion/premises)} \tag{50}
\]

\[
\Gamma_g = \Gamma_g \land \Gamma_g \land d : S_k \quad \text{(from rule conclusion/premises)} \tag{51}
\]

\[
\Theta \vdash s[p] : S = s[p] \oplus q \oplus \bigoplus_{i \in I} c_i(S_i) \cdot S_i' \quad \text{(by (25) and (50))} \tag{52}
\]

\[
\forall \Gamma', \Gamma'' : \Gamma_r \rightarrow \Gamma_r' \land \Gamma_r' \cdot x : S \rightarrow \Gamma_r'' \cdot x : S_k' \quad \text{(by Def.2.5)} \tag{53}
\]

\[
\forall \Gamma', \Gamma'' \quad \text{such that} \quad \left\{ \begin{array}{l}
\Gamma' \vdash \Gamma_g \land \Gamma_g \land d : S_k \\
\forall \Gamma_g, \Gamma'_d \vdash s[p] : S_k' \quad \text{(by (54))} \tag{55}
\end{array} \right.
\]

\[
\Theta \vdash s[p] : S_k' \land \Gamma_g \land \Gamma_g \land d : S_k' \quad \text{(by Def.3.3)} \tag{56}
\]

\[
\Theta \vdash \Gamma_g \land s[p] : S_k' \land \Gamma_g \land d : S_k' \quad \text{(by (55), (56) and i.h.)} \tag{57}
\]

*** inductive case [T-I]. We have:

\[
\exists k \in I: \Gamma_g = \Gamma_g, \Gamma_g \quad \text{(from the rule conclusion)} \tag{58}
\]

\[
\Theta \vdash P_1 \land P_2 \vdash \Gamma_1 \land \Gamma_2 : s \land \Gamma_r \quad \text{(from the rule conclusion and (58))} \tag{59}
\]

Now, in the rule premises, \( x : S \) could be used to type either \( P_1 \) or \( P_2 \) — i.e., we have either:

\[
\Theta \vdash P_1 \land P_2 \vdash \Gamma_1 \land \Gamma_2 : s \land \Gamma_r, \Gamma_1 : x : S \land \text{x} \notin \text{fv}(P_2) \tag{60}
\]

or

\[
\Theta \vdash P_1 \land P_2 \vdash \Gamma_1 \land \Gamma_2 : s \land \Gamma_r, \Gamma_2 : x : S \land \text{x} \notin \text{fv}(P_1) \tag{61}
\]

We now proceed assuming (60) (the proof for (61) is symmetric). We have:

\[
\Theta \vdash P_1 \land P_2 \vdash \Gamma_1 \land \Gamma_2 : s \land \Gamma_r, \Gamma_1 : x : S \land \text{x} \notin \text{fv}(P_2) \quad \text{(by (60), (25) and i.h.)} \tag{62}
\]

\[
\Theta \vdash P_2 \land P_2 \vdash \Gamma_1 \land \Gamma_2 : s \land \Gamma_r, \Gamma_1 : x : S \land \text{x} \notin \text{fv}(P_2) \quad \text{(by (60), (25) and Prop.3.8)} \tag{63}
\]

\[
P_2 = P_2 \land P_2 \quad \text{(by (60), since \( x \notin \text{fv}(P_2) \))} \tag{64}
\]
Proof. Assuming \( \Theta \vdash P \overset{\{s[p]/x\}}{\rightarrow} \Gamma_g, s[p]:S \triangleleft \Gamma_r \), we proceed by induction on its derivation.

\[
\begin{align*}
\Theta &\vdash (P_1 | P_2) \overset{\{s[p]/x\}}{\rightarrow} \Gamma_g, s[p]:S \triangleleft \Gamma_r \quad \text{(by (62), (63), (64) and (58))} \\
\text{(by (62), (63), (64) and (58))} \quad \text{(by (62), (63), (64) and (58))} \\
\text{(by (62), (63), (64) and (58))} \\
\text{(by (62), (63), (64) and (58))} \\
\end{align*}
\]

**Proposition C.10.** If \( \Theta \vdash P \triangleright \Gamma_g \triangleleft \Gamma_r \) and \( \Gamma_g' \equiv \Gamma_g \) and \( \Gamma_r' \equiv \Gamma_r \), then \( \Theta \vdash P \triangleright \Gamma_g' \triangleleft \Gamma_r' \).

**Proof.** Assuming \( \Theta \vdash P \triangleright \Gamma_g \triangleleft \Gamma_r \), we proceed by induction on its derivation.

**Lemma 3.7 (Rely context strengthening).** If \( \Theta \vdash P \triangleright \Gamma_g \triangleleft \Gamma_r \) and \( \Gamma_r \rightarrow \Gamma_r' \), then \( \Theta \vdash P \triangleright \Gamma_g \triangleleft \Gamma_r' \).

**Proof.** By induction on the derivation of \( \Theta \vdash P \triangleright \Gamma_g \triangleleft \Gamma_r \), using Prop. C.1 in the base cases.

**Proposition C.11.** Assume \( \Theta \vdash P \triangleright \Gamma_g \triangleleft \Gamma_r \). Take some \( \Gamma_g' \) such that \( \text{live}(\Gamma_g') \) and \( (\text{dom}(\Gamma_g) \cup \text{dom}(\Gamma_g')) \cap \Gamma_g') = \emptyset \). Then, \( \Theta \vdash P \triangleright \Gamma_g \triangleleft \Gamma_r, \Gamma_g' \).

**Proof.** By induction on the derivation of \( \Theta \vdash P \triangleright \Gamma_g \triangleleft \Gamma_r \).

**Proposition C.12.** Assume \( \Theta \vdash P \triangleright \Gamma_g \triangleleft \Gamma_r \). Take some \( \Gamma_g' \) such that \( \text{end}(\Gamma_g') \) and \( (\text{dom}(\Gamma_g) \cup \text{dom}(\Gamma_g')) \cap \Gamma_g') = \emptyset \). Then, \( \Theta \vdash P \triangleright \Gamma_g, \Gamma_g' \triangleleft \Gamma_r \).

**Proof.** By induction on the derivation of \( \Theta \vdash P \triangleright \Gamma_g \triangleleft \Gamma_r \).

**Proposition C.13.** Assume \( \Theta \vdash P \triangleright \Gamma_g \triangleleft \Gamma_r \). If \( X \notin \text{dom}(\Theta) \), then \( \Theta, X: \exists s \vdash P \triangleright \Gamma_g \triangleleft \Gamma_r \).

**Proof.** By induction on the derivation of \( \Theta \vdash P \triangleright \Gamma_g \triangleleft \Gamma_r \).

**Theorem 3.6.** If \( \Theta \vdash P \triangleright \Gamma_g \triangleleft \Gamma_r \) and \( P \rightarrow P' \), then \( \exists \Gamma_g' \) such that \( \Gamma_g \rightarrow \Gamma_g' \) and \( \Theta \vdash P' \triangleright \Gamma_g' \triangleleft \Gamma_r \).
Proof. Assume:

\[ \Theta \vdash P \triangleright \Gamma_\chi \triangleleft \Gamma_r \]  
(by hypothesis)  \hfill (72)

We first prove the following statement, for one reduction step of \( P \):

\[ P \rightarrow P' \]  
implies  
\[ \exists \Gamma'_g \text{ such that } \Gamma_\chi \rightarrow^* \Gamma'_g \text{ and } \Theta \vdash P' \triangleright \Gamma'_g \triangleleft \Gamma_r \]  \hfill (73)

Assuming the transition \( P \rightarrow P' \), we proceed by induction on its derivation.

- base case \([R\text{-COMM}]\).  We have:

\[ P = s[p][q] \&_{i \in I} \{ m_i(y_i).P_i \} \mid s[q][p] \oplus m_k(s'[x]).Q \quad (k \in I) \]  
(from the rule definition)  \hfill (74)

\[ P' = P_k \{ s'[x]/\chi \} \mid Q \]  
(from the rule definition)  \hfill (75)

\[ \Theta \vdash s[p][q] \&_{i \in I} \{ m_i(y_i).P_i \} \triangleright \Gamma_1 \triangleleft \Gamma_r, \Gamma_2 \]  
(by \( \Theta \vdash P \triangleright \Gamma_\chi \triangleleft \Gamma_r \), \hfill (76)

\[ \exists \Gamma_1, \Gamma_2 : \Theta \vdash s[q][p] \oplus m_k(s'[x]).Q \triangleright \Gamma_2 \triangleleft \Gamma_1, \Gamma_r \]  
(by \( \Theta \vdash P \triangleright \Gamma_\chi \triangleleft \Gamma_r \), \hfill (77)

\[ \exists \Gamma_{r1} = \Gamma_r, \Gamma_2 \text{ and } \exists \Gamma_{r2} = \Gamma_1, \Gamma_r \]  
(by \( \exists \Gamma_1, \Gamma_2 \), \hfill (78)

\[ \exists J : \emptyset \neq J \subseteq I \exists \Gamma_{g1}, S : \Gamma_1 = \Gamma_{g1}, s[p]:S \text{ such that } \]  
\[ S \equiv s[q] \&_{i \in I} \{ m_i(S_i).S'_i \} \forall i \in J \forall \Gamma'_r, \Gamma''_r \]  
\[ \Gamma_{r1} \rightarrow^* \Gamma'_r \text{ and } \Gamma''_r, s[p]:S \rightarrow \Gamma''_r, s[p]:S' \]  
\[ \text{implies } \Theta \vdash P_1 \supset \Gamma_{g1}, y_i:S_i,s[p]:S' \triangleleft \Gamma''_r \]  
(by \( \Theta \vdash P \supset \Gamma_\chi \triangleleft \Gamma_r \), \hfill (79)

\[ \Theta \vdash s[p][q] \&_{i \in I} \{ m_i(y_i).P_i \} \triangleright \Gamma_1 \triangleleft \Gamma_{r1} \]  
(by \( \Theta \vdash P \supset \Gamma_\chi \triangleleft \Gamma_r \), \hfill (80)

\[ \exists \Gamma_{g2}, \Gamma_{g2'}, \Gamma_{g2''}, T : \Gamma_2 = \Gamma_{g2}, s[q]:T \text{ such that } \]  
\[ T \equiv p \oplus_{i \in I} \{ m_i(T_i), T'_i \} \quad \exists k \in I' \quad \Gamma_{g2}, s[p]:S \rightarrow \Gamma_{g2'}, s[p]:S' \]  
\[ \land \forall \Gamma'_r, \Gamma''_r \]  
\[ \Gamma_{g2} \rightarrow^* \Gamma'_r \text{ and } \Gamma''_r, s[q]:T \rightarrow \Gamma''_r, s[q]:T'_k \]  
\[ \text{implies } \Theta \vdash Q \triangleright \Gamma_{g2}, s[q]:T' \triangleleft \Gamma''_r, \Gamma_{g2''} \]  
(by \( \Theta \vdash Q \triangleright \Gamma_\chi \triangleleft \Gamma_r \), \hfill (81)

\[ \Theta \vdash s[q][p] \oplus m_k(s'[x]).Q \triangleright \Gamma_{g2}, \Gamma_{g2'} \]  
(by \( \Theta \vdash Q \triangleright \Gamma_\chi \triangleleft \Gamma_r \), \hfill (82)

\[ \forall i \in J : S_i \equiv T_i \]  
and therefore  \( k \in J \)  \hfill (83)

\[ \Gamma_{g2}, s[p]:S \rightarrow \Gamma_{g2'}, s[p]:S' \]  
(by \( \Theta \vdash P \supset \Gamma_\chi \triangleleft \Gamma_r \), \hfill (84)

\[ \forall i \in J : \Gamma_{r1} \rightarrow s'[x]:S_k \]  
(by \( \Theta \vdash P \supset \Gamma_\chi \triangleleft \Gamma_r \), \hfill (85)

\[ \forall i \in J : s[p]:S = \Gamma_{r1}, s[p]:S' \rightarrow \Gamma_{r1}, s[p]:S' \]  
(by \( \Theta \vdash P \supset \Gamma_\chi \triangleleft \Gamma_r \) and Def 3.1)  \hfill (86)

\[ \Theta \vdash P_k \triangleright \Gamma_{g1}, y_k:S_k, s[p]:S' \triangleleft \Gamma_{r1} \]  
(by \( \Theta \vdash P \triangleright \Gamma_\chi \triangleleft \Gamma_r \) and Def 2.5)  \hfill (87)

\[ \Gamma_{r1} \rightarrow s'[x]:S_k \]  
(by \( \Theta \vdash P \triangleright \Gamma_\chi \triangleleft \Gamma_r \), \hfill (88)
\[ \Theta \vdash P'_k\{s'[\{x\}_{y_3}]\} \triangleright \Gamma_{g1}, s'[\{x]\} : S_k, s[\{p]\} \triangleright \Gamma'_{r1} \setminus s'[\{x\}] \]
(by \[88, 89\] and Lemma \[3.5\] \[90\])

\[ \Gamma'_{r1} \setminus s'[\{x\}] \equiv \Gamma_r, \Gamma_{g2}, s[\{q]\] : T'_k \]
(by \[77, 79\] and \[87\] \[91\])

\[ \exists \Gamma_{r2}' : \Gamma_{r2}, s[\{q]\] : T \rightarrow \equiv \Gamma_{r2}', s[\{p]\] : T'_k \]
(by \[81, 86\] \[92\])

\[ \Theta \vdash Q \triangleright \Gamma_{g2}, s[\{q]\] : T'_k < \Gamma_{r2}, \Gamma_{g2}' \]
(by \[82\] and \[79\] \[93\])

\[ \exists \Gamma' = \Gamma_{1}', \Gamma_{2}' \text{ such that:} \]
\[ \Gamma_{1}' = \Gamma_{g1}, \Gamma_{g2}', s[\{p]\] : S_k \]
\[ \Gamma_{2}' = \Gamma_{g2}, s[\{q]\] : T'_k \]
\[ \Gamma \rightarrow \Gamma' \]
(by \[76, 78, 79, 96, 97\] \[98\])

**base case \( \{R, X\} \).** We have:

\[ P = \text{def} X(x_1, \ldots, x_n) = Q \text{ in \( X(s_1[p_1], \ldots, s_1[p_1]) \) \( R \)} \]
(from the rule definition) \[100\]

\[ P' = \text{def} X(x_1, \ldots, x_n) = Q \text{ in \( Q(s_1[p_1]/s_1) \cdots s_n[p_n]/s_n \) \( R \)} \]
(from the rule definition) \[101\]

\[ \Gamma_g = \Gamma_{g*}, \Gamma_{gR} \text{ and } \Gamma_{g*} = \Gamma_{g0} \Gamma_{g1}, \ldots, \Gamma_{gn} \text{ such that:} \]
\[ \Theta' = \Theta, X : S_1, \ldots, S_n \]
\[ \text{end}(\Gamma_{g0}) \forall i \in 1..n \quad \Gamma_{gi} \vdash s_i[p_i] : S_i \]
\[ \Theta' = X(s_1[p_1], \ldots, s_1[p_1]) \triangleright \Gamma_{g*} \triangleright \Gamma_r, \Gamma_{gR} \]
\[ \Theta' = R \triangleright \Gamma_{g*} \triangleright \Gamma_r, [\text{T-def}] \]
\[ \Theta = Q \triangleright x_1 : S_1, \ldots, x_n : S_n \triangleright \emptyset \quad \Theta' = X(s_1[p_1], \ldots, s_1[p_1]) \]
\[ \Theta' \vdash R \triangleright \Gamma_{g*} \triangleright \Gamma_r, [\text{T-def}] \]
\[ \Theta' \vdash P \triangleright \Gamma_g \triangleright \Gamma_r \]
(by \[100, 72\], inv. \[\text{T-def}, [\text{T-}\triangleright], [\text{T-X}]\]) \[103\]

\[ X \notin \text{dom}(\Theta) \]
(by \[102\] \[104\])

\[ \text{live}(\Gamma_{g0}) \]
(by \[103\] and Prop.C.7) \[105\]

\[ \text{live}(\Gamma_{g1}, \ldots, \Gamma_{gn}, \Gamma_r, \Gamma_{gR}) \]
(by \[103\], Def.3.3 \[105\] and Prop.C.6) \[106\]

\[ \text{live}(\Gamma_{g1}, \ldots, \Gamma_{gn}, (\Gamma_r, \Gamma_{gR}) \setminus x_1, \ldots, x_n) \]
(by \[106\] and Prop.C.8 \( \times n \)) \[107\]

\[ \text{live}(\Gamma_{g1}, \ldots, \Gamma_{gn}, (\Gamma_r, \Gamma_{gR}) \setminus x_1, \ldots, x_n) \]
(by \[107\] \[103\] and Def.3.3) \[108\]

\[ \Theta = Q \triangleright x_1 : S_1, \ldots, x_n : S_n < \Gamma_{g1}, \ldots, \Gamma_{gn} : ((\Gamma_r, \Gamma_{gR}) \setminus x_1, \ldots, x_n) \]
(by \[108\] \[103\] and Prop.C.11) \[109\]

\[ \Theta = Q(s_1[p_1]/s_1) \cdots s_n[p_n]/s_n) \triangleright \Gamma_{g1}, \ldots, \Gamma_{gn} < \Gamma_r, \Gamma_{gR} \]
(by \[109\] \[103\], Lemma 3.5 \( \times n \)) \[110\]

\[ \Theta' = Q(s_1[p_1]/s_1) \cdots s_n[p_n]/s_n) \triangleright \Gamma_{g*} < \Gamma_r, \Gamma_{gR} \]
(by \[111\] \[104\], \[102\] and Prop.C.13) \[112\]

Let \( \Gamma' = \Gamma_g = \Gamma_{g*}, \Gamma_{gR} = \Gamma_{g0}, \Gamma_{g1}, \ldots, \Gamma_{gn}, \Gamma_{gR} \)
(hence, \( \Gamma_g \rightarrow \Gamma' \)) \[102\]

(by \[102\] \[113\]
\[\Theta \vdash Q \triangleright x_1: S_1, \ldots, x_n: S_n \triangleleft \emptyset \quad \Theta' \vdash Q\{s_1[p_1]/x_1\} \cdots \{s_n[p_n]/x_n\} \triangleright \Gamma_g \triangleleft \Gamma_r, \Gamma_g \triangleright R \triangleright \Gamma_g \triangleleft \Gamma_r \quad [\text{T-def}]\]

\[\Theta' \vdash R \triangleright \Gamma_g \triangleleft \Gamma_r\]

\[\theta \vdash P' \triangleright \Gamma_g \triangleleft \Gamma_r \quad \text{(by (111), (113), (101))}\]

- **inductive case [R-].** We have:

\[P = P_1 | P_2\]  
(from the rule conclusion) \hspace{1cm} (114)

\[\exists \Gamma_1, \Gamma_2 : \quad \Gamma_g = \Gamma_1, \Gamma_2\]  
(by (114) only typable by [T-]) and (72) \hspace{1cm} (115)

\[\theta \vdash P_1 \triangleright \Gamma_1 \triangleleft \Gamma_r, \Gamma_2\]  
(by (114), (115) and inversion of [T-]) \hspace{1cm} (116)

\[\theta \vdash P_2 \triangleright \Gamma_2 \triangleleft \Gamma_r, \Gamma_1\]  
(by (114), (115) and inversion of [T-]) \hspace{1cm} (117)

Now, \(P \to P'\) implies either:

\[P_1 \to P'_1\]  
and \(P' = P'_1 | P_2\)  
(by inversion of [R-]) \hspace{1cm} (118)

or

\[P_2 \to P'_2\]  
and \(P' = P_1 | P'_2\)  
(by congruence \(\equiv\) and inversion of [R-]) \hspace{1cm} (119)

We proceed assuming (118) (the proof for (119) is symmetric).

\[\exists \Gamma'_1 : \quad \Gamma_1 \to \Gamma'_1\]  
and \(\theta \vdash P'_1 \triangleright \Gamma'_1 \triangleleft \Gamma_r, \Gamma_2\)  
(by (116), (118) and i.h.) \hspace{1cm} (120)

\[\Gamma_1 \to \Gamma'_1\]  
(by (120)) \hspace{1cm} (121)

\[\Gamma_r, \Gamma_1 \to \Gamma_r, \Gamma'_1\]  
(by (121) and Def 2.5) \hspace{1cm} (122)

\[\theta \vdash P_2 \triangleright \Gamma_2 \triangleleft \Gamma_r, \Gamma'_1\]  
(by (117), (122) and Lemma 3.7) \hspace{1cm} (123)

\[\exists \Gamma''_g = \Gamma'_1, \Gamma_2 \quad \text{such that} \quad \Gamma_g \to \Gamma''_g\]  
(by (115), (120) and Def 2.5) \hspace{1cm} (124)

\[\theta \vdash P'_1 \triangleright \Gamma''_g \triangleleft \Gamma_r, \Gamma_2\]
\[\theta \vdash P_2 \triangleright \Gamma_2 \triangleleft \Gamma_r, \Gamma'_1\]

\[\theta \vdash P' \triangleright \Gamma''_g \triangleleft \Gamma_r\]  
(by (118), (120), (123), (124)) \hspace{1cm} (125)

- **inductive case [R-v].** We have:

\[\exists Q : \quad P = (v s) Q\]  
(from the rule definition) \hspace{1cm} (125)

\[P \to P'\]  
implies \(\exists Q' : Q \to Q'\) and \(P' = (v s) Q'\)  
(by (125) and inversion of [R-v]) \hspace{1cm} (126)

\[\Gamma_x = \{c[p] : S_p\}_{p \in I}\]
\[\Theta \vdash Q \triangleright \Gamma_g, \Gamma_s \triangleleft \Gamma_r\]

\[\theta \vdash (v s : \Gamma_g) Q \triangleright \Gamma_g \triangleleft \Gamma_r\]  
(by (125), only typable by [T-v]) \hspace{1cm} (127)

\[\exists \Gamma''_g : \quad \Gamma_g, \Gamma_s \to \Gamma''_g\]  
and \(\theta \vdash Q' \triangleright \Gamma''_g \triangleleft \Gamma_r\)  
(by (127) (premise of [T-v]), (126) and i.h.) \hspace{1cm} (128)

\[\text{live}(\Gamma_g)\]  
(by (127) (premise and conclusion of [T-v]) and Prop C.6) \hspace{1cm} (129)

\[\exists \Gamma''_g, \Gamma'_s : \quad \Gamma_g, \Gamma_s \to \Gamma''_g, \Gamma'_s\]  
and \(\Gamma_g \to \Gamma''_g\) and \(\Gamma_s \to \Gamma'_s\)  
(by (129), (128) and Prop C.5) \hspace{1cm} (130)

\[\Gamma_g, \Gamma_s \to \Gamma''_g, \Gamma'_s\]  
(by (130) and Def 2.5) \hspace{1cm} (131)

\[\text{live}(\Gamma_g, \Gamma'_s)\]  
(by (127) (conclusion of [T-v]), (131) and Prop C.1) \hspace{1cm} (132)

\[\Gamma'_s = \{s[p] : S'_p\}_{p \in I}\]
\[\Theta \vdash Q' \triangleright \Gamma'_s \triangleleft \Gamma_r\]

\[\theta \vdash P' \triangleright \Gamma'_s \triangleleft \Gamma_r\]  
(by (126), (128), (130), (132)) \hspace{1cm} (132)
• inductive case [R-def]. We have:

\[ P = \text{def}\ X(\bar{x}) = Q \text{ in } R \quad \text{(from the rule definition)} \tag{133} \]
\[ P' = \text{def}\ X(\bar{x}) = Q \text{ in } R' \quad \text{with } R \rightarrow R' \quad \text{(from the rule premises)} \tag{134} \]

\[
\Theta \vdash Q \triangleright x_1:S_1, \ldots, x_n:S_n \triangleleft \emptyset \quad \Theta, X:S_1, \ldots, S_n \vdash \Gamma \triangleright \Gamma_g \triangleleft \Gamma_r \quad \Theta \vdash P \triangleright \Gamma_g \triangleleft \Gamma_r \quad \text{[\text{T-def}] (by (133), (72), inv. [T-def])} \tag{135} \]

\[
\exists \Gamma': \Gamma \rightarrow^* \Gamma' \text{ and } \Theta, X:S_1, \ldots, S_n \vdash R' \triangleright \Gamma'_g \triangleleft \Gamma_r \quad \text{[by (135), (134) and i.h).} \tag{136} \]

\[
\Theta \vdash Q \triangleright x_1:S_1, \ldots, x_n:S_n \triangleleft \emptyset \quad \Theta, X:S_1, \ldots, S_n \vdash R' \triangleright \Gamma'_g \triangleleft \Gamma_r \quad \Theta \vdash P' \triangleright \Gamma'_g \triangleleft \Gamma_r \quad \text{[\text{T-def}] (by (136) and (134))} \]

We have thus proved (73). Then, we prove the main statement by induction on the number of reductions in \( P \rightarrow^* P' \): the base case is trivial (0 reductions, and thus \( P = P' \)); in the inductive case (\( n + 1 \) reductions), we apply the induction hypothesis and (73). \( \square \)