

# <sup>1</sup> Formally Verified Liveness with Synchronous <sup>2</sup> Multiparty Session Types in Rocq

<sup>3</sup> **Anonymous author**

<sup>4</sup> **Anonymous affiliation**

<sup>5</sup> **Anonymous author**

<sup>6</sup> **Anonymous affiliation**

---

## <sup>7</sup> — Abstract —

<sup>8</sup> Multiparty session types (MPST) offer a framework for the description of communication-based  
<sup>9</sup> protocols involving multiple participants. In the *top-down* approach to MPST, the communication  
<sup>10</sup> pattern of the session is described using a *global type*. Then the global type is *projected* on to a *local*  
<sup>11</sup> *type* for each participant, and the individual processes making up the session are type-checked against  
<sup>12</sup> these projections. Typed sessions possess certain desirable properties such as *safety*, *deadlock-freedom*  
<sup>13</sup> and *liveness* (also called *lock-freedom*).

<sup>14</sup> In this work, we present the first mechanised proof of liveness for synchronous multiparty session  
<sup>15</sup> types in the Rocq Proof Assistant. Building on recent work, we represent global and local types as  
<sup>16</sup> coinductive trees using the paco library. We use a coinductively defined *subtyping* relation on local  
<sup>17</sup> types together with another coinductively defined *plain-merge* projection relation relating local and  
<sup>18</sup> global types . We then *associate* collections of local types, or *local type contexts*, with global types  
<sup>19</sup> using this projection and subtyping relations, and prove an *operational correspondence* between a  
<sup>20</sup> local type context and its associated global type. We then utilize this association relation to prove  
<sup>21</sup> the safety and liveness of associated local type contexts and, consequently, the multiparty sessions  
<sup>22</sup> typed by these contexts.

<sup>23</sup> Besides clarifying the often informal proofs of liveness found in the MPST literature, our Rocq  
<sup>24</sup> mechanisation also enables the certification of lock-freedom properties of communication protocols.  
<sup>25</sup> Our contribution amounts to around 12K lines of Rocq code.

<sup>26</sup> **2012 ACM Subject Classification** Replace ccsdesc macro with valid one

<sup>27</sup> **Keywords and phrases** Dummy keyword

<sup>28</sup> **Digital Object Identifier** 10.4230/LIPIcs.CVIT.2016.23

<sup>29</sup> **Acknowledgements** Anonymous acknowledgements

## <sup>30</sup> 1 Introduction

<sup>31</sup> Multiparty session types [19] provide a type discipline for the correct-by-construction spe-  
<sup>32</sup> cification of message-passing protocols. Desirable protocol properties guaranteed by session  
<sup>33</sup> types include *safety* (the labels and types of senders' payloads cohere with the capabilities of  
<sup>34</sup> the receivers), *deadlock-freedom* (also called *progress* or *non-stuck property* [14]) (it is possible  
<sup>35</sup> for the session to progress so long as it has at least one active participant), and *liveness* (also  
<sup>36</sup> called *lock-freedom* [41] or *starvation-freedom* [8]) (if a process is waiting to send and receive  
<sup>37</sup> then a communication involving it eventually happens).

<sup>38</sup> There exists two common methodologies for multiparty session types. In the *bottom-up*  
<sup>39</sup> approach, the individual processes making up the session are typed using a collection of  
<sup>40</sup> *participants* and *local types*, that is, a *local type context*, and the properties of the session is  
<sup>41</sup> examined by model-checking this local type context. Contrastingly, in the *top-down* approach  
<sup>42</sup> sessions are typed by a *global type* that is related to the processes using endpoint *projections*  
<sup>43</sup> and *subtyping*. The structure of the global type ensures that the desired properties are  
<sup>44</sup> satisfied by the session. These two approaches have their advantages and disadvantages:



© **Anonymous author(s)**;

licensed under Creative Commons License CC-BY 4.0

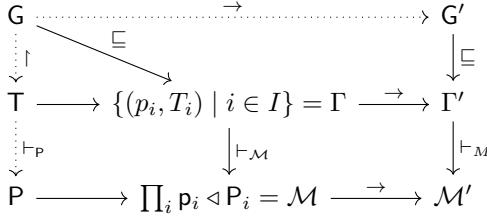
42nd Conference on Very Important Topics (CVIT 2016).

Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:21



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



**Figure 1** Design overview. The dotted lines correspond to relations inherited from [14] while the solid lines denote relations that are new, or substantially rewritten, in this paper.

45 the bottom-up approach is generally able to type more sessions, while type-checking and  
 46 type-inferring in the top-down approach tend to be more efficient than model-checking the  
 47 bottom-up system [40].

48 In this work, we present the Rocq [4] formalisation of a synchronous MPST that that  
 49 ensures the aforementioned properties for typed sessions. Our type system uses an *association*  
 50 relation ( $\sqsubseteq$ ) [44, 32] defined using (coinductive plain) projection [38] and subtyping, in order  
 51 to relate local type contexts and global types. This association relation ensures *operational*  
 52 *correspondence* between the labelled transition system (LTS) semantics we define for local  
 53 type contexts and global types. We then type ( $\vdash_M$ ) sessions using local type contexts that  
 54 are associated with global types, which ensure that the local type context, and hence the  
 55 session, is well-behaved in some sense. Whenever an associated local type context  $\Gamma$  types a  
 56 session  $M$ , our type system guarantees the following properties:

- 57 1. **Subject Reduction** (Theorem 6.2): If  $M$  can progress into  $M'$ , then  $\Gamma$  can progress  
 58 into  $\Gamma'$  such that  $\Gamma'$  types  $M'$ .
- 59 2. **Session Fidelity** (Theorem 6.5): If  $\Gamma$  can progress into  $\Gamma'$ , then  $M$  can progress into  
 60  $M'$  such that  $M'$  is typable by  $\Gamma'$ .
- 61 3. **Safety** (Theorem 6.7): If  $M$  can progress into  $M'$  by one or more communications,  
 62 participant  $p$  in  $M'$  sends to participant  $q$  and  $q$  receives from  $p$ , then the labels of  $p$  and  
 63  $q$  cohere.
- 64 4. **Deadlock-Freedom** (Theorem 6.4): Either every participant in  $M$  has terminated, or  
 65  $M$  can progress.
- 66 5. **Liveness** (Theorem 6.11): If participant  $p$  attempts to communicate with participant  $q$   
 67 in  $M$ , then  $M$  can progress (in possibly multiple steps) into a session  $M'$  where that  
 68 communication has occurred.

69 To our knowledge, this work presents the first mechanisation of liveness for multiparty session  
 70 types in a proof assistant.

71 Our Rocq implementation builds upon the recent formalisation of subject reduction for  
 72 MPST by Ekici et. al. [14], which itself is based on [17]. The methodology in [14] takes an  
 73 equirecursive approach where an inductive syntactic global or local type is identified with  
 74 the coinductive tree obtained by fully unfolding the recursion. It then defines a coinductive  
 75 projection relation between global and local type trees, the LTS semantics for global type  
 76 trees, and typing rules for the session calculus outlined in [17]. We extensively use these  
 77 definitions and the lemmas concerning them, but we still depart from and extend [14] in  
 78 numerous ways by introducing local typing contexts, their correspondence with global types  
 79 and a new typing relation. Our addition to the code amounts to around 12K lines of Rocq  
 80 code.

81 As with [14], our implementation heavily uses the parameterized coinduction technique  
 82 of the paco [20] library. Namely, our liveness property is defined using possibly infinite

83 *execution traces* which we represent as coinductive streams. The relevant predicates on these  
 84 traces, such as fairness, are then defined using linear temporal logic (LTL)[33]. The LTL  
 85 modalities eventually ( $\diamond$ ) and always ( $\square$ ) can be expressed as least and greatest fixpoints  
 86 respectively using expansion laws. This allows us to represent the properties that use these  
 87 modalities as inductive and coinductive predicates in Rocq. This approach, together with  
 88 the proof techniques provided by paco, results in compositional and clear proofs.

89 **Outline.** In Section 2 we define our session calculus and its LTS semantics. In Section 3  
 90 we introduce local and global type trees. In Section 4 we give LTS semantics to local type  
 91 contexts and global types, and detail the association relation between them. In Section 5  
 92 we define safety and liveness for local type contexts, and prove that they hold for contexts  
 93 associated with a global type tree. In Section 6 we give the typing rules for our session  
 94 calculus, and prove the desired properties of these typable sessions.

## 95 2 The Session Calculus

96 We introduce the simple synchronous session calculus that our type system will be used  
 97 on.

### 98 2.1 Processes and Sessions

99 ► **Definition 2.1** (Expressions and Processes). *We define processes as follows:*

$$100 \quad P ::= p!\ell(e).P \mid \sum_{i \in I} p?\ell_i(x_i).P_i \mid \text{if } e \text{ then } P \text{ else } P \mid \mu X.P \mid X \mid 0$$

101 where  $e$  is an expression that can be a variable, a value such as `true`, `0` or `-3`, or a term  
 102 built from expressions by applying the operators `succ`, `neg`, `¬`, non-deterministic choice  $\oplus$   
 103 and  $>$ .

104  $p!\ell(e).P$  is a process that sends the value of expression  $e$  with label  $\ell$  to participant  $p$ , and  
 105 continues with process  $P$ .  $\sum_{i \in I} p?\ell_i(x_i).P_i$  is a process that may receive a value from  $p$  with  
 106 any label  $\ell_i$  where  $i \in I$ , binding the result to  $x_i$  and continuing with  $P_i$ , depending on  
 107 which  $\ell_i$  the value was received from.  $X$  is a recursion variable,  $\mu X.P$  is a recursive process,  
 108 if  $e$  then  $P$  else  $P$  is a conditional and  $0$  is a terminated process.

109 Processes can be composed in parallel into sessions.

110 ► **Definition 2.2** (Multiparty Sessions). *Multiparty sessions are defined as follows.*

$$111 \quad M ::= p \triangleleft P \mid (M \mid M) \mid \mathcal{O}$$

112  $p \triangleleft P$  denotes that participant  $p$  is running the process  $P$ ,  $|$  indicates parallel composition.

113 We write  $\prod_{i \in I} p_i \triangleleft P_i$  to denote the session formed by  $p_i$  running  $P_i$  in parallel for all  $i \in I$ .

114  $\mathcal{O}$  is an empty session with no participants, that is, the unit of parallel composition. In  
 115 Rocq processes and sessions are defined with the inductive types `process`  and `session` .

```
Inductive process : Type ≡
| p_send : part → label → expr → process →
  process
| p_recv : part → list(option process) → process
| p_ite : expr → process → process → process
| p_rec : process → process
| p_var : nat → process
| p_inact : process.
```

```
Inductive session : Type ≡
| s_ind : part → process → session
| s_par : session → session → session
| s_zero : session.
Notation "p '←→' P" ≡ (s_ind p P) (at level 50, no
associativity).
Notation "s1 '|||' s2" ≡ (s_par s1 s2) (at level 50, no
associativity).
```

## 117 2.2 Structural Congruence and Operational Semantics

118 We define a structural congruence relation  $\equiv$  on sessions which expresses the commutativity,  
 119 associativity and unit of the parallel composition operator.

$$\begin{array}{ll} [\text{SC-SYM}] & [\text{SC-ASSOC}] \\ p \triangleleft P \mid q \triangleleft Q \equiv q \triangleleft Q \mid p \triangleleft P & (p \triangleleft P \mid q \triangleleft Q) \mid r \triangleleft R \equiv p \triangleleft P \mid (q \triangleleft Q \mid r \triangleleft R) \\ [\text{SC-O}] \\ p \triangleleft P \mid \mathcal{O} \equiv p \triangleleft P \end{array}$$

■ Table 1 Structural Congruence over Sessions

120 We omit the semantics for expressions, they are standard and can be found in e.g. [17].  
 121 We now give the operational semantics for sessions by the means of a labelled transition  
 122 system. We use labelled *reactive* semantics [41, 6] which doesn't contain explicit silent  $\tau$   
 123 actions for internal reductions (that is, evaluation of if expressions and unfolding of recursion)  
 124 while still considering  $\beta$  reductions up to those internal reductions by using an unfolding  
 125 relation. This stands in contrast to the more standard semantics used in [14, 17, 41]. For  
 126 the advantages of our approach see Remark 6.3.

127 In reactive semantics silent transitions are captured by an *unfolding* relation ( $\Rightarrow$ ), and  $\beta$   
 reductions are defined up to this unfolding (Table 2).

$$\begin{array}{c} [\text{UNF-STRUCT}] \quad [\text{UNF-REC}] \quad [\text{UNF-COND}] \\ \mathcal{M} \equiv \mathcal{N} \quad p \triangleleft \mu X.P \mid \mathcal{N} \Rightarrow p \triangleleft P[\mu X.P/X] \mid \mathcal{N} \quad p \triangleleft \text{if } e \text{ then } P \text{ else } Q \mid \mathcal{N} \xrightarrow{e \downarrow \text{true}} p \triangleleft P \mid \mathcal{N} \\ \hline [\text{UNF-CONDF}] \quad [\text{UNF-TRANS}] \\ p \triangleleft \text{if } e \text{ then } P \text{ else } Q \mid \mathcal{N} \xrightarrow{e \downarrow \text{false}} p \triangleleft Q \mid \mathcal{N} \quad \frac{\mathcal{M} \Rightarrow \mathcal{M}' \quad \mathcal{M}' \Rightarrow \mathcal{N}}{\mathcal{M} \Rightarrow \mathcal{N}} \end{array}$$

■ Table 2 Unfolding of Sessions

128  
 129  $\mathcal{M} \Rightarrow \mathcal{N}$  means that  $\mathcal{M}$  can transition to  $\mathcal{N}$  through some internal actions, that is, a  
 130 reduction that doesn't involve a communication. We say that  $\mathcal{M}$  *unfolds* to  $\mathcal{N}$ . In Rocq it's  
 captured by the predicate `unfoldP : session → session → Prop` 

$$\begin{array}{c} [\text{R-COMM}] \\ j \in I \quad e \downarrow v \\ \hline p \triangleleft \sum_{i \in I} q? \ell_i(x_i).P_i \mid q \triangleleft p! \ell_j(e).Q \mid \mathcal{N} \xrightarrow{(p,q)\ell_j} p \triangleleft P_j[v/x_j] \mid q \triangleleft Q \mid \mathcal{N} \\ [\text{R-UNFOLD}] \\ \mathcal{M} \Rightarrow \mathcal{M}' \quad \mathcal{M}' \xrightarrow{\lambda} \mathcal{N}' \quad \mathcal{N}' \Rightarrow \mathcal{N} \\ \hline \mathcal{M} \xrightarrow{\lambda} \mathcal{N} \end{array}$$

■ Table 3 Reactive Semantics of Sessions

131  
 132 Table 3 illustrates the rules for communicating transitons. [R-COMM] captures communica-  
 133 tions between processes, and [R-UNFOLD] lets us consider reductions up to unfoldings.

134 In Rocq, `betaP_1bl M lambda M'` denotes  $M \xrightarrow{\lambda} M'$ . We write  $M \rightarrow M'$  if  $M \xrightarrow{\lambda} M'$  for  
135 some  $\lambda$ , which is written `betaP M M'` in Rocq. We write  $\rightarrow^*$  to denote the reflexive transitive  
136 closure of  $\rightarrow$ , which is called `betaRtc` in Rocq.

### 137 3 The Type System

138 We briefly recap the core definitions of local and global type trees, subtyping and projection  
139 from [17]. We take an equirecursive approach and work directly on the possibly infinite local  
140 and global type trees obtained by unfolding the recursion in guarded syntactic types, details  
141 of this approach can be found in [14] and hence are omitted here.

#### 142 3.1 Local Type Trees

143 We start by defining the sorts that will be used to type expressions, and local types that will  
144 be used to type single processes.

145 ▶ **Definition 3.1** (Sorts and Local Type Trees). *We define three atomic sorts: `int`,*  
146 *`bool` and `nat`. Local type trees are then defined coinductively with the following syntax:*

```
147 T ::= end
      | p&{ $\ell_i(S_i).$ Ti}i ∈ I
      | p⊕{ $\ell_i(S_i).$ Ti}i ∈ I
```

```
Inductive sort : Type ≡
| sbool : sort | sint : sort | snat : sort.

CoInductive ltt : Type ≡
| ltt_end : ltt
| ltt_recv : part → list (option(sort*ltt)) → ltt
| ltt_send : part → list (option(sort*ltt)) → ltt.
```

148 In the above definition, `end` represents a role that has finished communicating.  
149  $p\&\{\ell_i(S_i).\mathbb{T}_i\}_{i \in I}$  denotes a role that may, from any  $i \in I$ , receive a value of sort  $S_i$  with  
150 message label  $\ell_i$  and continue with  $\mathbb{T}_i$ . Similarly,  $p\&\{\ell_i(S_i).\mathbb{T}_i\}_{i \in I}$  represents a role that may  
151 choose to send a value of sort  $S_i$  with message label  $\ell_i$  and continue with  $\mathbb{T}_i$  for any  $i \in I$ .

152 In Rocq we represent the continuations using a `list` of `option` types. In a continuation  
153 `gcs : list (option(sort*ltt))`, index  $k$  (using zero-indexing) being equal to `Some (s_k,`  
154  $T_k)$  means that  $\ell_k(S_k).$ T<sub>k</sub> is available in the continuation. Similarly index  $k$  being equal to  
155 `None` or being out of bounds of the list means that the message label  $\ell_k$  is not present in the  
156 continuation.

157 ▶ **Remark 3.2.** Note that Rocq allows us to create types such as `ltt_send q []` which don't  
158 correspond to well-formed local types as the continuation is empty. In our implementation  
159 we define a predicate `wfLtt : ltt → Prop` capturing that all the continuations in the local  
160 type tree are non-empty. Henceforth we assume that all local types we mention satisfy this  
161 property.

#### 162 3.2 Subtyping

163 We define the subsorting relation on sorts and the subtyping relation on local type trees.

164 ▶ **Definition 3.3** (Subsorting and Subtyping). *Subsorting  $\leq$  is the least reflexive binary  
165 relation that satisfies `nat ≤ int`. Subtyping  $\leqslant$  is the largest relation between local type trees*

## 23:6 Dummy short title

166 coinductively defined by the following rules:

$$\frac{\text{===== [SUB-END]} \quad \frac{\forall i \in I : S'_i \leq S_i \quad T_i \leqslant T'_i}{\text{end} \leqslant \text{end}}}{\text{p} \& \{\ell_i(S_i).T_i\}_{i \in I \cup J} \leqslant \text{p} \& \{\ell_i(S'_i).T'_i\}_{i \in I}}$$

$$\frac{\forall i \in I : S_i \leq S'_i \quad T_i \leqslant T'_i}{\text{p} \oplus \{\ell_i(S_i).T_i\}_{i \in I} \leqslant \text{p} \oplus \{\ell_i(S'_i).T'_i\}_{i \in I \cup J}} \quad \text{[SUB-OUT]}$$

168 Intuitively,  $T_1 \leqslant T_2$  means that a role of type  $T_1$  can be supplied anywhere a role of type  $T_2$  is needed. [SUB-IN] captures the fact that we can supply a role that is able to receive more labels than specified, and [SUB-OUT] captures that we can supply a role that has fewer labels available to send. Note the contravariance of the sorts in [SUB-IN], if the supertype demands the ability to receive an `nat` then the subtype can receive `nat` or `int`.

173 In Rocq, the subtyping relation `subtypeC : ltt → ltt → Prop` is expressed as a greatest fixpoint using the `Paco` library [20], for details of we refer to [17].

### 175 3.3 Global Types and Type Trees

176 We now define global types which give a bird's eye view of the whole protocol. As before, we 177 work directly on infinite trees and omit the details which can be found in [14]. `end` denotes 178 a protocol that has ended,  $p \rightarrow q : \{\ell_i(S_i).G_i\}_{i \in I}$  denotes a protocol where for any  $i \in I$ , 179 participant  $p$  may send a value of sort  $S_i$  to another participant  $q$  via message label  $\ell_i$ , after 180 which the protocol continues as  $G_i$ .

181 ▶ **Definition 3.4** (Global type trees). *We define global type trees coinductively as follows:*

$$182 \quad G ::= \text{end} \mid p \rightarrow q : \{\ell_i(S_i).G_i\}_{i \in I}$$

$$\text{CoInductive gtt: Type } \triangleq$$

$$\mid \text{gtt\_end} : \text{gtt}$$

$$\mid \text{gtt\_send} : \text{part} \rightarrow \text{part} \rightarrow \text{list}(\text{option}(\text{sort}^*\text{gtt})) \rightarrow \text{gtt}.$$

183 We further define the function  $\text{pt}(G)$  that denotes the participants of the global type  $G$  as 184 the least solution <sup>1</sup> to the following equations:

$$185 \quad \text{pt}(\text{end}) = \emptyset$$

$$186 \quad \text{pt}(p \rightarrow q : \{\ell_i(S_i).G_i\}_{i \in I}) = \{p, q\} \cup \bigcup_{i \in I} \text{pt}(G_i)$$

187 We extend the function  $\text{pt}$  onto trees by defining  $\text{pt}(G) = \text{pt}(G)$  where the global type 188  $G$  corresponds to the global type tree  $G$ . Technical details of this definition such as well-189 definedness can be found in [14, 17].

190 In Rocq  $\text{pt}$  is captured with the predicate `isgPartsC : part → gtt → Prop`, where 191 `isgPartsC p G` denotes  $p \in \text{pt}(G)$ .

### 192 3.4 Projection

193 We now define coinductive projections with plain merging (see [40] for a survey of other 194 notions of merge).

---

<sup>1</sup> Here we adopt a simplified presentation as  $\text{pt}(G)$  is actually defined by extending it from an inductively defined function on syntactic types, we refer to [14] for details.

195 ► **Definition 3.5** (Projection). *The projection of a global type tree onto a participant r is the  
196 largest relation  $\upharpoonright_r$  between global type trees and local type trees such that, whenever  $G \upharpoonright_r T$ :*

- 197 ■  $r \notin \text{pt}\{G\}$  implies  $T = \text{end}$ ; [PROJ-END]
- 198 ■  $G = p \rightarrow r : \{\ell_i(S_i).G_i\}_{i \in I}$  implies  $T = p \& \{\ell_i(S_i).T_i\}_{i \in I}$  and  $\forall i \in I, G \upharpoonright_r T_i$  [PROJ-IN]
- 199 ■  $G = r \rightarrow q : \{\ell_i(S_i).G_i\}_{i \in I}$  implies  $T = q \oplus \{\ell_i(S_i).T_i\}_{i \in I}$  and  $\forall i \in I, G \upharpoonright_r T_i$  [PROJ-OUT]
- 200 ■  $G = p \rightarrow q : \{\ell_i(S_i).G_i\}_{i \in I}$  and  $r \notin \{p, q\}$  implies that there are  $T_i, i \in I$  such that  
201  $T = \prod_{i \in I} T_i$  and  $\forall i \in I, G \upharpoonright_r T_i$  [PROJ-CONT]

202 where  $\prod$  is the plain merging operator, defined as

$$203 T_1 \prod T_2 = \begin{cases} T_1 & \text{if } T_1 = T_2 \\ \text{undefined} & \text{otherwise} \end{cases}$$

204 Informally, the projection of a global type tree  $G$  onto a participant  $r$  extracts a specification  
205 for participant  $r$  from the protocol whose bird's-eye view is given by  $G$ . [PROJ-END]  
206 expresses that if  $r$  is not a participant of  $G$  then  $r$  does nothing in the protocol. [PROJ-IN]  
207 and [PROJ-OUT] handle the cases where  $r$  is involved in a communication in the root of  $G$ .  
208 [PROJ-CONT] says that, if  $r$  is not involved in the root communication of  $G$ , then the only  
209 way it knows its role in the protocol is if there is a role for it that works no matter what  
210 choices  $p$  and  $q$  make in their communication. This "works no matter the choices of the other  
211 participants" property is captured by the merge operations.

212 In Rocq, projection is defined as a `Paco` greatest fixpoint as the relation `projectionC` :  
213 `gtt` → `part` → `ltt` → `Prop`.

214 We further have the following fact about projections that lets us regard it as a partial  
215 function:

216 ► **Lemma 3.6** ([14]). *If  $\text{projectionC } G \ p \ T$  and  $\text{projectionC } G \ p \ T'$  then  $T = T'$ .*

217 We write  $G \upharpoonright r = T$  when  $G \upharpoonright_r T$ . Furthermore we will be frequently be making assertions  
218 about subtypes of projections of a global type e.g.  $T \leqslant G \upharpoonright r$ . In our Rocq implementation  
219 we define the predicate `issubProj` : `ltt` → `gtt` → `part` → `Prop` as a shorthand for this.

## 220 3.5 Balancedness, Global Tree Contexts and Grafting

221 We introduce an important constraint on the types of global type trees we will consider,  
222 balancedness.

223 ► **Definition 3.7** (Balanced Global Type Trees). *A global tree  $G$  is balanced if for any subtree  
224  $G'$  of  $G$ , there exists  $k$  such that for all  $p \in \text{pt}(G')$ ,  $p$  occurs on every path from the root of  
225  $G'$  of length at least  $k$ .*

226 We omit the technical details of this definition and the Rocq implementation, they can be  
227 found in [17] and [14].

228 Intuitively, balancedness is a regularity condition that imposes a notion of *liveness* on the  
229 protocol described by the global type tree. Indeed, our liveness results in Section 6 hold only  
230 for balanced global types. Another reason for formulating balancedness is that it allows us  
231 to use the "grafting" technique, turning proofs by coinduction on infinite trees to proofs by  
232 induction on finite global type tree contexts.

233 ► **Definition 3.8** (Global Type Tree Context). *Global type tree contexts are defined inductively  
234 with the following syntax:*

235

$$\mathcal{G} ::= \quad p \rightarrow q : \{\ell_i(S_i).G_i\}_{i \in I} \quad | \quad [ ]_i$$

```
Inductive gtth: Type  $\triangleq$ 
| gtth_hol : fin  $\rightarrow$  gtth
```

```
| gtth_send : part  $\rightarrow$  part  $\rightarrow$  list (option (sort * gtth))  $\rightarrow$  gtth.
```

236

We additionally define `pt` and `ishParts` on contexts analogously to `pt` and `isgPartsC` on trees.

238 A global type tree context can be thought of as the finite prefix of a global type tree, where  
 239 holes  $[ ]_i$  indicate the cutoff points. Global type tree contexts are related to global type trees  
 240 with the grafting operation.

241 ► **Definition 3.9** (Grafting). *Given a global type tree context  $\mathcal{G}$  whose holes are in the  
 242 indexing set  $I$  and a set of global types  $\{G_i\}_{i \in I}$ , the grafting  $\mathcal{G}[G_i]_{i \in I}$  denotes the global type  
 243 tree obtained by substituting  $[ ]_i$  with  $G_i$  in  $\mathcal{G}$ .*

244 In Rocq the indexed set  $\{G_i\}_{i \in I}$  is represented using a list (option gtt). Grafting is  
 245 expressed with the inductive relation `typ_gtth` :  $\text{list } (\text{option gtt}) \rightarrow \text{gtth} \rightarrow \text{gtt} \rightarrow$   
 246 `Prop`. `typ_gtth gs gcx gt` means that the grafting of the set of global type trees `gs` onto the  
 247 context `gcx` results in the tree `gt`.

248 Furthermore, we have the following lemma that relates global type tree contexts to  
 249 balanced global type trees.

250 ► **Lemma 3.10** (Proper Grafting Lemma, [14]). *If  $G$  is a balanced global type tree and  
 251 `isgPartsC p G`, then there is a global type tree context  $G_{ctx}$  and an option list of global type  
 252 trees `gs` such that `typ_gtth gs G_{ctx} G`,  $\sim \text{ishParts p G}_{ctx}$  and every `Some` element of `gs` is of  
 253 shape `gtt_end`, `gtt_send p q` or `gtt_send q p`.*

254 3.10 enables us to represent a coinductive global type tree featuring participant `p` as the  
 255 grafting of a context that doesn't contain `p` with a list of trees that are all of a certain  
 256 structure. If `typ_gtth gs G_{ctx} G`,  $\sim \text{ishParts p G}_{ctx}$  and every `Some` element of `gs` is of shape  
 257 `gtt_end`, `gtt_send p q` or `gtt_send q p`, then we call the pair `gs` and `G_{ctx}` as the `p`-grafting  
 258 of `G`, expressed in Rocq as `typ_p_gtth gs G_{ctx} p G`. When we don't care about the contents  
 259 of `gs` we may just say that `G` is `p`-grafted by `G_{ctx}`.

260 ► **Remark 3.11.** From now on, all the global type trees we will be referring to are assumed  
 261 to be balanced. When talking about the Rocq implementation, any  $G : \text{gtt}$  we mention  
 262 is assumed to satisfy the predicate `wfgC G`, expressing that `G` corresponds to some global  
 263 type and that `G` is balanced. Furthermore, we will often require that a global type is  
 264 projectable onto all its participants. This is captured by the predicate `projectableA G =  $\forall$`   
 265 `p,  $\exists T$ , projectionC G p T`. As with `wfgC`, we will be assuming that all types we mention  
 266 are projectable.

267

## 4 Semantics of Types

268 In this section we introduce local type contexts, and define Labelled Transition System  
 269 semantics on these constructs.

270 **4.1 Typing Contexts**

271 We start by defining typing contexts as finite mappings of participants to local type trees.

► **Definition 4.1** (Typing Contexts).

272

$$\Gamma ::= \emptyset \mid \Gamma, p : T$$

```
Module M  $\triangleq$  MMaps.RET.Make(Nat).
Module MF  $\triangleq$  MMaps.Facts.Properties Nat M.
Definition tctx: Type  $\triangleq$  M.t ltt.
```

273 Intuitively,  $p : T$  means that participant  $p$  is associated with a process that has the type  
 274 tree  $T$ . We write  $\text{dom}(\Gamma)$  to denote the set of participants occurring in  $\Gamma$ . We write  $\Gamma(p)$  for  
 275 the type of  $p$  in  $\Gamma$ . We define the composition  $\Gamma_1, \Gamma_2$  iff  $\text{dom}(\Gamma_1) \cap \text{dom}(\Gamma_2) = \emptyset$ .

276 In the Rocq implementation we implement local typing contexts as finite maps of  
 277 participants, which are represented as natural numbers, and local type trees. We use  
 278 the red-black tree based finite map implementation of the MMMaps library [27].

279 ► **Remark 4.2.** From now on, we assume the all the types in the local type contexts always  
 280 have non-empty continuations. In Rocq terms, if  $T$  is in context `gamma` then `wfltt T` holds.  
 281 This is expressed by the predicate `wfltt: tctx  $\rightarrow$  Prop`.

282 **4.2 Local Type Context Reductions**

283 We now give LTS semantics to local typing contexts, for which we first define the transition  
 284 labels.

285 ► **Definition 4.3** (Transition labels). *A transition label  $\alpha$  has the following form:*

$$\begin{aligned} 286 \quad \alpha ::= & p : q \& \ell(S) \quad (p \text{ receives a value of sort } S \text{ from } q \text{ with message label } \ell) \\ 287 \quad | \quad & p : q \oplus \ell(S) \quad (p \text{ sends a value of sort } S \text{ to } q \text{ with message label } \ell) \\ 288 \quad | \quad & (p, q) \ell \quad (A \text{ synchronized communication from } p \text{ to } q \text{ occurs via message label } \ell) \end{aligned}$$

289

290 In Rocq they are defined as follows:

```
Notation opt_lbl  $\triangleq$  nat.
Inductive label: Type  $\triangleq$ 
| lrecv: part  $\rightarrow$  part  $\rightarrow$  option sort  $\rightarrow$  opt_lbl  $\rightarrow$  label
| lsend: part  $\rightarrow$  part  $\rightarrow$  option sort  $\rightarrow$  opt_lbl  $\rightarrow$  label
| lcomm: part  $\rightarrow$  part  $\rightarrow$  opt_lbl  $\rightarrow$  label.
```

291

292 Next we define labelled transitions for local type contexts.

293 ► **Definition 4.4** (Typing context reductions). *The typing context transition  $\xrightarrow{\alpha}$  is defined  
 294 inductively by the following rules:*

$$295 \frac{k \in I}{p : q \& \{ \ell_i(S_i).T_i \}_{i \in I} \xrightarrow{p : q \& \ell_k(S_k)} p : T_k} [\Gamma\text{-}\&]$$

$$\frac{k \in I}{p : q \oplus \{ \ell_i(S_i).T_i \}_{i \in I} \xrightarrow{p : q \oplus \ell_k(S_k)} p : T_k} [\Gamma\text{-}\oplus] \quad \frac{\Gamma \xrightarrow{\alpha} \Gamma'}{\Gamma, p : T \xrightarrow{\alpha} \Gamma', p : T} [\Gamma\text{-},]$$

$$\frac{\Gamma_1 \xrightarrow{p : q \oplus \ell(S)} \Gamma'_1 \quad \Gamma_2 \xrightarrow{q : p \& \ell(S')} \Gamma'_2 \quad S \leq S'}{\Gamma_1, \Gamma_2 \xrightarrow{(p, q) \ell} \Gamma'_1, \Gamma'_2} [\Gamma\text{-}\oplus\&]$$

## 23:10 Dummy short title

296 We write  $\Gamma \xrightarrow{\alpha} \Gamma'$  if there exists  $\Gamma'$  such that  $\Gamma \xrightarrow{a} \Gamma'$ . We define a reduction  $\Gamma \rightarrow \Gamma'$  that holds  
 297 iff  $\Gamma \xrightarrow{(p,q)\ell} \Gamma'$  for some  $p, q, \ell$ . We write  $\Gamma \rightarrow$  iff  $\Gamma \rightarrow \Gamma'$  for some  $\Gamma'$ . We write  $\rightarrow^*$  for  
 298 the reflexive transitive closure of  $\rightarrow$ .

299  $[\Gamma \oplus]$  and  $[\Gamma \&]$ , express a single participant sending or receiving.  $[\Gamma \oplus]$  expresses a  
 300 synchronized communication where one participant sends while another receives, and they  
 301 both progress with their continuation.  $[\Gamma \neg]$  shows how to extend a context.

302 In Rocq typing context reductions are defined the following way:

```
Inductive tctxR: tctx → label → tctx → Prop ≡
| Rsend: ∀ p q xs n s T,
  p ≠ q →
  onth n xs = Some (s, T) →
  tctxR (M.add p (litt_send q xs) M.empty) (lsend p q (Some s) n) (M.add p T M.empty)
| Rrecv: ...
| Rcomm: ∀ p q g1' g2' g2'' s' n (H1: MF.Disjoint g1 g2') (H2: MF.Disjoint g1' g2''),
  p ≠ q →
  tctxR g1 (lsend p q (Some s) n) g1' →
  tctxR g2 (lrecv q p (Some s') n) g2' →
  subsort s s' →
  tctxR (disj_merge g1 g2 H1) (lcomm p q n) (disj_merge g1' g2' H2)
| RvarI: ∀ g l g' p T,
  tctxR g l g' →
  M.mem p g = false →
  tctxR (M.add p T g) l (M.add p T g')
| Rstruct: ∀ g1 g1' g2 g2' l, tctxR g1' l g2' →
  M.Equal g1 g1' →
  M.Equal g2 g2' →
  tctxR g1 l g2'.
```

303

304 **Rsend**, **Rrecv** and **RvarI** are straightforward translations of  $[\Gamma \neg \&]$ ,  $[\Gamma \neg \oplus]$  and  $[\Gamma \neg \neg]$ .  
 305 **Rcomm** captures  $[\Gamma \neg \oplus \&]$  using the `disj_merge` function we defined for the compositions, and  
 306 requires a proof that the contexts given are disjoint to be applied. **RStruct** captures the  
 307 indistinguishability of local contexts under the `M.Equal` predicate from the `MMaps` library.  
 308 We give an example to illustrate typing context reductions.

309 ▶ **Example 4.5.** Let

```
310   T_p = q ⊕ {ℓ_0(int).T_p , ℓ_1(int).end}
311   T_q = p & {ℓ_0(int).T_q , ℓ_1(int).r ⊕ {ℓ_2(int).end}}
312   T_r = q & {ℓ_2(int).end}
```

313 and  $\Gamma = \{p : T_p, q : T_q, r : T_r\}$ . We have the reductions  $\Gamma \xrightarrow{p:q \oplus \ell_0(int)} \Gamma$  and  $\Gamma \xrightarrow{q:p \& \ell_0(int)} \Gamma$ , which synchronise to give the reduction and  $\Gamma \xrightarrow{(p,q)\ell_0} \Gamma$ . Similarly via synchronised  
 314 communication of  $p$  and  $q$  via message label  $\ell_1$  we get  $\Gamma \xrightarrow{(p,q)\ell_1} \Gamma'$  where  $\Gamma'$  is defined as  
 315  $\{p : end, q : r \oplus \{\ell_2(int).end\}, r : T_r\}$ . We further have that  $\Gamma' \xrightarrow{(q,r)\ell_2} \Gamma_{end}$  where  $\Gamma_{end}$  is  
 316 defined as  $\{p : end, q : end, r : end\}$ .  
 317

318 In Rocq,  $\Gamma$  is defined the following way:

```
319 Definition prt_p ≡ 0.
Definition prt_q ≡ 1.
Definition prt_r ≡ 2.
CoFixpoint T_p ≡ litt_send prt_q [Some (sint,T_p); Some (sint,litt_end); None].
CoFixpoint T_q ≡ litt_recv prt_p [Some (sint,T_q); Some (sint,litt_send prt_r [None,None;Some (sint,litt_end)]); None].
Definition T_r ≡ litt_recv prt_q [None,None; Some (sint,litt_end)].
Definition gamma ≡ M.add prt_p T_p (M.add prt_q T_q (M.add prt_r T_r M.empty)).
```

319

320 Now  $\Gamma \xrightarrow{(p,q)\ell_0} \Gamma$  can be expressed as `tctxR gamma (lsend prt_p prt_q (Some sint) 0) gamma`.

### 4.3 Global Type Reductions

322 As with local typing contexts, we can also define reductions for global types.

323 ► **Definition 4.6** (Global type reductions). *The global type transition  $\xrightarrow{\alpha}$  is defined coinductively  
324 as follows.*

$$\frac{k \in I}{\boxed{p \rightarrow q : \{\ell_i(S_i).G_i\}_{i \in I} \xrightarrow{(p,q)\ell_k} G_k}} \text{ [GR-}\oplus\&\text{]}$$

$$\frac{\forall i \in I \ G_i \xrightarrow{\alpha} G'_i \quad \text{subject}(\alpha) \cap \{p, q\} = \emptyset \quad \forall i \in I \ \{p, q\} \subseteq \text{pt}\{G_i\}}{p \rightarrow q : \{\ell_i(S_i).G_i\}_{i \in I} \xrightarrow{\alpha} p \rightarrow q : \{\ell_i(S_i).G'_i\}_{i \in I}} \text{ [GR-CTX]}$$

326 [GR- $\oplus\&$ ] says that a global type tree with root  $p \rightarrow q$  can transition to any of its children  
327 corresponding to the message label chosen by  $p$ . [GR-CTX] says that if the subjects of  $\alpha$   
328 are disjoint from the root and all its children can transition via  $\alpha$ , then the whole tree can  
329 also transition via  $\alpha$ , with the root remaining the same and just the subtrees of its children  
330 transitioning.

331 In Rocq global type reductions are expressed using the coinductively defined predicate  
332 `gttstepC`. For example,  $G \xrightarrow{(p,q)\ell_k} G'$  translates to `gttstepC G G' p q k`. We refer to [14] for  
333 details.

#### 334 4.4 Association Between Local Type Contexts and Global Types

335 We have defined local type contexts which specifies protocols bottom-up by directly describing  
336 the roles of every participant, and global types, which give a top-down view of the whole  
337 protocol, and the transition relations on them. We now relate these local and global definitions  
338 by defining *association* between local type context and global types.

339 ► **Definition 4.7** (Association). *A local typing context  $\Gamma$  is associated with a global type tree  
340  $G$ , written  $\Gamma \sqsubseteq G$ , if the following hold:*

- 341 ■ *For all  $p \in \text{pt}(G)$ ,  $p \in \text{dom}(\Gamma)$  and  $\Gamma(p) \leqslant G \upharpoonright p$ .*
- 342 ■ *For all  $p \notin \text{pt}(G)$ , either  $p \notin \text{dom}(\Gamma)$  or  $\Gamma(p) = \text{end}$ .*

343 *In Rocq this is defined with the following:*

```
Definition assoc (g: tctx) (gt:gtt) ≡
  ∀ p, (isgPartsC p gt → ∃ Tp, M.find p g=Some Tp ∧
    issubProj Tp gt p) ∧
  (¬ isgPartsC p gt → ∀ Tpx, M.find p g = Some Tpx → Tpx=ltt_end).
```

344 Informally,  $\Gamma \sqsubseteq G$  says that the local type trees in  $\Gamma$  obey the specification described by the  
345 global type tree  $G$ .

346 ► **Example 4.8.** In Example 4.5, we have that  $\Gamma \sqsubseteq G$  where

348  $G := p \rightarrow q : \{\ell_0(\text{int}).G, \ell_1(\text{int}).q \rightarrow r : \{\ell_2(\text{int}).\text{end}\}\}$

349 In fact, we have  $\Gamma(s) = G \upharpoonright s$  for  $s \in \{p, q, r\}$ . Similarly, we have  $\Gamma' \sqsubseteq G'$  where

350  $G' := q \rightarrow r : \{\ell_2(\text{int}).\text{end}\}$

351 It is desirable to have the association be preserved under local type context and global  
352 type reductions, that is, when one of the associated constructs "takes a step" so should the  
353 other. We formalise this property with soundness and completeness theorems.

## 23:12 Dummy short title

354 ► **Theorem 4.9** (Soundness of Association). *If  $\text{assoc } \gamma$  and  $\text{gttstepC } G \rightarrow G' \ p \ q \ \ell$ ,  
 355 then there is a local type context  $\gamma'$ , a global type tree  $G''$  and a message label  $\ell'$  such  
 356 that  $\text{gttStepC } G \rightarrow G'' \ p \ q \ \ell'$ ,  $\text{assoc } \gamma' \rightarrow G''$  and  $\text{tctxR } \gamma \rightarrow (\text{lcomm } p \ q \ \ell') \ \gamma'$ .*

357 ► **Theorem 4.10** (Completeness of Association). *If  $\text{assoc } \gamma$  and  $\text{tctxR } \gamma \rightarrow (\text{lcomm } p \ q \ \ell) \ \gamma'$ ,  
 358 then there exists a global type tree  $G''$  such that  $\text{assoc } \gamma' \rightarrow G''$  and  $\text{gttstepC } G \rightarrow G'' \ p \ q \ \ell$ .*  
 359

360 ► **Remark 4.11.** Note that in the statement of soundness we allow the message label for the  
 361 local type context reduction to be different to the message label for the global type reduction.  
 362 This is because our use of subtyping in association causes the entries in the local type context  
 363 to be less expressive than the types obtained by projecting the global type. For example  
 364 consider

$$365 \quad \Gamma = p : q \oplus \{\ell_0(\text{int}).\text{end}\}, \ q : p \& \{\ell_0(\text{int}).\text{end}, \ell_1(\text{int}).\text{end}\}$$

366 and

$$367 \quad G = p \rightarrow q : \{\ell_0(\text{int}).\text{end}, \ell_1(\text{int}).\text{end}\}$$

368 We have  $\Gamma \sqsubseteq G$  and  $G \xrightarrow{(p,q)\ell_1}$ . However  $\Gamma \xrightarrow{(p,q)\ell_1}$  is not a valid transition. Note that  
 369 soundness still requires that  $\Gamma \xrightarrow{(p,q)\ell_x}$  for some  $x$ , which is satisfied in this case by the valid  
 370 transition  $\Gamma \xrightarrow{(p,q)\ell_0}$ .

## 371 5 Properties of Local Type Contexts

372 We now use the LTS semantics to define some desirable properties on type contexts and their  
 373 reduction sequences. Namely, we formulate safety, liveness and fairness properties based on  
 374 the definitions in [44].

### 375 5.1 Safety

376 We start by defining safety:

377 ► **Definition 5.1** (Safe Type Contexts). *We define  $\text{safe}$  coinductively as the largest set of type  
 378 contexts such that whenever we have  $\Gamma \in \text{safe}$ :*

$$379 \quad \begin{array}{l} \Gamma \xrightarrow{p:q \oplus \ell(S)} \text{ and } \Gamma \xrightarrow{q:p \& \ell'(S')} \text{ implies } \Gamma \xrightarrow{(p,q)\ell} \\ \Gamma \rightarrow \Gamma' \text{ implies } \Gamma' \in \text{safe} \end{array} \quad \begin{array}{l} [\text{S-}\&\oplus] \\ [\text{S-}\rightarrow] \end{array}$$

381 We write  $\text{safe}(\Gamma)$  if  $\Gamma \in \text{safe}$ .

382 Informally, safety says that if  $p$  and  $q$  communicate with each other and  $p$  requests to send a  
 383 value using message label  $\ell$ , then  $q$  should be able to receive that message label. Furthermore,  
 384 this property should be preserved under any typing context reductions. Being a coinductive  
 385 property, to show that  $\text{safe}(\Gamma)$  it suffices to give a set  $\varphi$  such that  $\Gamma \in \varphi$  and  $\varphi$  satisfies  
 386  $[\text{S-}\&\oplus]$  and  $[\text{S-}\rightarrow]$ . This amounts to showing that every element of  $\Gamma'$  of the set of reducts  
 387 of  $\Gamma$ , defined  $\varphi := \{\Gamma' \mid \Gamma \rightarrow^* \Gamma'\}$ , satisfies  $[\text{S-}\&\oplus]$ . We illustrate this with some examples:

388 ► **Example 5.2.** Let  $\Gamma_A = p : \text{end}$ , then  $\Gamma_A$  is safe: the set of reducts is  $\{\Gamma_A\}$  and this set  
 389 respects  $[\text{S-}\&\oplus]$  as its elements can't reduce, and it respects  $[\text{S-}\rightarrow]$  as it's closed with  
 390 respect to  $\rightarrow$ .

391 Let  $\Gamma_B = p : q \oplus \{\ell_0(\text{int}).\text{end}\}, q : p \& \{\ell_0(\text{nat}).\text{end}\}$ .  $\Gamma_B$  is not safe as we have  
 392  $\Gamma_B \xrightarrow{p:q \oplus \ell_0} \Gamma_B \xrightarrow{q:p \& \ell_0} \Gamma_B \xrightarrow{(p,q)\ell_0}$  but we don't have  $\Gamma_B \xrightarrow{(p,q)\ell_0}$  as  $\text{int} \not\leq \text{nat}$ .

393 Let  $\Gamma_C = p : q \oplus \{\ell_1(\text{int}).q \oplus \{\ell_0(\text{int}).\text{end}\}\}, q : p \& \{\ell_1(\text{int}).p \& \{\ell_0(\text{nat}).\text{end}\}\}$ .  $\Gamma_C$  is not  
 394 safe as we have  $\Gamma_C \xrightarrow{(p,q)\ell_1} \Gamma_B$  and  $\Gamma_B$  is not safe.

395 Consider  $\Gamma$  from Example 4.5. All the reducts satisfy [S-& $\oplus$ ], hence  $\Gamma$  is safe.

396 Being a coinductive property, `safe` can be expressed in Rocq using Paco:

```
Definition weak_safety (c: tctx)  $\triangleq$ 
   $\forall p q s s' k k', \text{tctxRE} (\text{lsend } p q (\text{Some } s) k) c \rightarrow \text{tctxRE} (\text{lrecv } q p (\text{Some } s') k') c \rightarrow$ 
     $\text{tctxRE} (\text{lcomm } p q k) c$ 

Inductive safe (R: tctx  $\rightarrow$  Prop): tctx  $\rightarrow$  Prop  $\triangleq$ 
  | safety_red :  $\forall c, \text{weak\_safety } c \rightarrow (\forall p q c' k,$ 
     $\text{tctxR } c (\text{lcomm } p q k) c' \rightarrow R c')$ 
     $\rightarrow \text{safe } R c$ 

Definition safeC c  $\triangleq$  paco1 safe bot1 c.
```

397 398  $\text{weak\_safety}$  corresponds [S-& $\oplus$ ] where  $\text{tctxRE } l c$  is shorthand for  $\exists c', \text{tctxR } c l c'$ . In  
 399 the inductive `safe`, the constructor `safety_red` corresponds to [S $\rightarrow$ ]. Then `safeC` is defined  
 400 as the greatest fixed point of `safe`.

401 We have that local type contexts with associated global types are always safe.

402 ▶ **Theorem 5.3** (Safety by Association ). If `assoc gamma g` then `safeC gamma`.

## 403 5.2 Fairness and Liveness

404 We now focus our attention to fairness and liveness. We first restate the definition of fairness  
 405 and liveness for local type context paths from [44].

406 ▶ **Definition 5.4** (Fair, Live Paths). A local type context reduction path (also called executions  
 407 or runs) is a possibly infinite sequence of transitions  $\Gamma_0 \xrightarrow{\lambda_0} \Gamma_1 \xrightarrow{\lambda_1} \dots$  such that  $\lambda_i$  is a  
 408 synchronous transition label, that is, of the form  $(p, q)\ell$ , for all  $i$ .

409 We say that a local type context reduction path  $\Gamma_0 \xrightarrow{\lambda_0} \Gamma_1 \xrightarrow{\lambda_1} \dots$  is fair if, for all  
 410  $n \in N : \Gamma_n \xrightarrow{(p,q)\ell} \text{ implies } \exists k, \ell' \text{ such that } N \ni k \geq n \text{ and } \lambda_k = (p, q)\ell'$ , and therefore  
 411  $\Gamma_k \xrightarrow{(p,q)\ell'} \Gamma_{k+1}$ . We say that a path  $(\Gamma_n)_{n \in N}$  is live iff,  $\forall n \in N$ :

- 412 1.  $\forall n \in N : \Gamma_n \xrightarrow{p:q \oplus \ell(S)} \text{ implies } \exists k, \ell' \text{ such that } N \ni k \geq n \text{ and } \Gamma_k \xrightarrow{(p,q)\ell'} \Gamma_{k+1}$
- 413 2.  $\forall n \in N : \Gamma_n \xrightarrow{q:p \& \ell(S)} \text{ implies } \exists k, \ell' \text{ such that } N \ni k \geq n \text{ and } \Gamma_k \xrightarrow{(p,q)\ell'} \Gamma_{k+1}$

414 ▶ **Definition 5.5** (Live Local Type Context). A local type context  $\Gamma$  is live if whenever  $\Gamma \rightarrow^* \Gamma'$ ,  
 415 every fair path starting from  $\Gamma'$  is also live.

416 In general, fairness assumptions are used so that only the reduction sequences that are  
 417 "well-behaved" in some sense are considered when formulating other properties [42]. For our  
 418 purposes we define fairness such that, in a fair path, if at any point  $p$  attempts to send to  $q$   
 419 and  $q$  attempts to send to  $p$  then eventually a communication between  $p$  and  $q$  takes place.  
 420 Then live paths are defined to be paths such that whenever  $p$  attempts to send to  $q$  or  $q$   
 421 attempts to send to  $p$ , eventually a  $p$  to  $q$  communication takes place. Informally, this means  
 422 that every communication request is eventually answered. Then live typing contexts are  
 423 defined to be the  $\Gamma$  where all fair paths that start from  $\Gamma$  are also live.

424 ▶ **Example 5.6.** Consider the contexts  $\Gamma, \Gamma'$  and  $\Gamma_{\text{end}}$  from Example 4.5. One possible  
 425 reduction path is  $\Gamma \xrightarrow{(p,q)\ell_0} \Gamma \xrightarrow{(p,q)\ell_0} \dots$ . Denote this path as  $(\Gamma_n)_{n \in N}$ , where  $\Gamma_n = \Gamma$

## 23:14 Dummy short title

426 for all  $n \in \mathbb{N}$ . We have  $\forall n, \Gamma_n \xrightarrow{(p,q)\ell_0}$  and  $\Gamma_n \xrightarrow{(p,q)\ell_1}$  as the only possible synchronised  
 427 reductions from  $\Gamma_n$ . Accordingly, we also have  $\forall n, \Gamma_n \xrightarrow{(p,q)\ell_0} \Gamma_{n+1}$  in the path so this path  
 428 is fair. However, this path is not live as we have  $\Gamma_1 \xrightarrow{r:q\&\ell_2(\text{int})}$  but there is no  $n, \ell'$  with  
 429  $\Gamma_n \xrightarrow{(q,r)\ell'} \Gamma_{n+1}$  in the path. Consequently,  $\Gamma$  is not a live type context.

430 Now consider the reduction path  $\Gamma \xrightarrow{(p,q)\ell_0} \Gamma \xrightarrow{(p,q)\ell_0} \Gamma' \xrightarrow{(q,r)\ell_2} \Gamma_{\text{end}}$ . This path is fair and  
 431 live as it contains the  $(q, r)$  transition from the counterexample above.

432 Definition 5.4, while intuitive, is not really convenient for a Rocq formalisation due to  
 433 the existential statements contained in them. It would be ideal if these properties could  
 434 be expressed as a least or greatest fixed point, which could then be formalised via Rocq's  
 435 inductive or (via Paco) coinductive types. To achieve this, we recast fairness and liveness for  
 436 local type context paths in Linear Temporal Logic (LTL) [33].  $\diamond$  and  $\square$  can be characterised  
 437 as least and greatest fixed points using their expansion laws [2, Chapter 5.14]. Hence they  
 438 can be implemented in Rocq as the inductive type `eventually` and the coinductive type  
 439 `alwaysCG` . We can further represent reduction paths as *consequences*, or *streams*. Then the  
 440 Rocq definition of Definition 5.4 amounts to the following .

```
441 CoInductive coseq (A: Type): Type ≡
| conil : coseq A
| concons : A → coseq A → coseq A.
Notation local_path ≡ (coseq (tctx*option label)).
```

```
Definition fair_path_local_inner (pt: local_path): Prop ≡
  ∀ p q n, to_path_prop (tctxRE (lcomm p q n)) False pt →
  eventually (headComm p q) pt.
Definition fair_path ≡ alwaysCG fair_path_local_inner.
Definition live_path_inner (pt: local_path) : Prop ≡ ∀ p q s n,
  (to_path_prop (tctxRE (lsend p q (Some s) n)) False pt →
  eventually (headComm p q) pt) ∧
  (to_path_prop (tctxRE (lrecv p q (Some s) n)) False pt →
  eventually (headComm q p) pt).
Definition live_path ≡ alwaysCG live_path_inner.
```

442 With these definitions we can now prove that local type contexts associated with a global  
 443 type are live, which is the most involved of the results mechanised in this work. We now  
 444 detail the Rocq Proof that associated local type contexts are also live.

445 ▶ **Remark 5.7.** We once again emphasise that all global types mentioned are assumed to  
 446 be balanced (Definition 3.7). Indeed association with non-balanced global types doesn't  
 447 guarantee liveness. As an example, consider  $\Gamma$  from Example 4.5, which is associated with  $G$   
 448 from Example 4.8. Yet we have shown in Example 5.6 that  $\Gamma$  is not a live type context. This  
 449 is not surprising as  $G$  is not balanced.

450 ▶ **Theorem 5.8 (Liveness by Association .**)*If `assoc gamma g` then `gamma` is live.*

451 **Proof.** (Simplified, Outline) Our proof proceeds in two steps. First, we prove that the typing  
 452 context obtained by direct projections <sup>2</sup> of  $g$ , that is,  $\text{gamma\_proj} = \{p_i : G \mid p_i \in \text{pt}\{G\}\}$ ,  
 453 is live. We then leverage Theorem 4.10 to show that if  $\text{gamma\_proj}$  is live, so is  $\text{gamma}$ .

454 The proof that  $\text{gamma\_proj}$  is live proceeds by well-founded induction on the tree height  
 455 [12] of the grafting (Lemma 3.10) of the global type  $g$ . Suppose  $\text{gamma\_proj} \xrightarrow{p:q\oplus\ell(S)}$  (the  
 456 case for the receive is similar and omitted), and  $\text{xs}$  is a fair local type context reduction path  
 457 beginning with  $\text{gamma\_proj}$ . To show that  $\text{xs}$  is live we need to show the existence of a  $(p, q)\ell$   
 458 transition in  $\text{xs}$ . We prove the following helper lemmas:

459 ■ The height of the  $p$ -grafting of  $g$  is not smaller than the  $q$ -grafting .

---

<sup>2</sup> Note that the actual Rocq proof defines an equivalent "enabledness" predicate on global types instead of working with direct projections. The outline given here is a slightly simplified presentation.

460 ■ If the p-grafting and q-grafting of a global type  $g'$  have the same height, then any fair  
 461 path beginning with the direct projection context of  $g'$  eventually contains a  $(p, q)\ell$   
 462 transition  $\text{proj}$ .

463 ■ The height of the p-grafting of  $g$  strictly decreases with every transition involving  $q\text{proj}$ ,  
 464 and doesn't increase with the transitions not involving  $q\text{proj}$ .

465 These lemmas followed by well-founded induction on the height of the p-grafting of the global  
 466 type the head of  $\mathbf{xs}$  is projected from gives the desired transition.

467 In the second step of the proof we extend association on to paths to get, for each local  
 468 type context reduction path  $\mathbf{xs}$  that begins with  $\mathbf{gamma}$ , another local type context reduction  
 469 path  $\mathbf{ys}$  beginning with  $\mathbf{gamma\_proj}$  such that the elements of  $\mathbf{xs}$  are subtypes (subtyping  
 470 on contexts defined pointwise) of the corresponding elements of  $\mathbf{ys}$ . This is obtained from  
 471 Theorem 4.10, however the statement of Theorem 4.10 is implemented as an  $\exists$  statement  
 472 that lives in  $\mathbf{Prop}$ , hence we need to use the `constructive_indefinite_description` axiom to  
 473 construct a `CoFixpoint` returning the desired cosequence  $\mathbf{ys}$ . The proof then follows by the  
 474 definition of subtyping (Definition 3.3). ◀

## 475 6 Properties of Sessions

476 We give typing rules for the session calculus introduced in 2, and prove subject reduction and  
 477 progress for them. Then we define a liveness property for sessions, and show that processes  
 478 typable by a local type context that's associated with a global type tree are guaranteed to  
 479 satisfy this liveness property.

### 480 6.1 Typing rules

481 We give typing rules for our session calculus based on [17] and [14].

482 We distinguish between two kinds of typing judgements and type contexts.

- 483 1. A local type context  $\Gamma$  associates participants with local type trees, as defined in cdef-type-ctx.  
 484 Local type contexts are used to type sessions (Definition 2.2) i.e. a set of pairs  
 485 of participants and single processes composed in parallel. We express such judgements as  
 486  $\Gamma \vdash_M M$ , or as  $\mathbf{typ\_sess} M \mathbf{gamma}$  or  $\mathbf{gamma} \vdash M$  in Rocq.
- 487 2. A process variable context  $\Theta_T$  associates process variables with local type trees, and an  
 488 expression variable context  $\Theta_e$  assigns sorts to expresion variables. Variable contexts  
 489 are used to type single processes and expressions (Definition 2.1). Such judgements are  
 490 expressed as  $\Theta_T, \Theta_e \vdash_P P : T$ , or in Rocq as  $\mathbf{typ\_proc} \theta_T \theta_e P T$  or  $\mathbf{theta\_T},$   
 $\mathbf{theta\_e} \vdash P : T$ .

$$\begin{array}{c}
 \Theta \vdash_P n : \mathbf{nat} \quad \Theta \vdash_P i : \mathbf{int} \quad \Theta \vdash_P \mathbf{true} : \mathbf{bool} \quad \Theta \vdash_P \mathbf{false} : \mathbf{bool} \quad \Theta, x : S \vdash_P x : S \\
 \frac{\Theta \vdash_P e : \mathbf{nat}}{\Theta \vdash_P \mathbf{succ} e : \mathbf{nat}} \quad \frac{\Theta \vdash_P e : \mathbf{int}}{\Theta \vdash_P \mathbf{neg} e : \mathbf{int}} \quad \frac{\Theta \vdash_P e : \mathbf{bool}}{\Theta \vdash_P \neg e : \mathbf{bool}} \\
 \frac{\Theta \vdash_P e_1 : S \quad \Theta \vdash_P e_2 : S}{\Theta \vdash_P e_1 \oplus e_2 : S} \quad \frac{\Theta \vdash_P e_1 : \mathbf{int} \quad \Theta \vdash_P e_2 : \mathbf{int}}{\Theta \vdash_P e_1 > e_2 : \mathbf{bool}} \quad \frac{\Theta \vdash_P e : S \quad S \leq S'}{\Theta \vdash_P e : S'}
 \end{array}$$

■ Table 4 Typing expressions

$$\begin{array}{c}
 \frac{[\text{T-END}]}{\Theta \vdash_P \mathbf{0} : \text{end}} \quad \frac{[\text{T-VAR}]}{\Theta, X : T \vdash_P X : T} \quad \frac{[\text{T-REC}]}{\Theta, X : T \vdash_P P : T} \quad \frac{[\text{T-IF}]}{\Theta \vdash_P e : \text{bool} \quad \Theta \vdash_P P_1 : T \quad \Theta \vdash_P P_2 : T}{\Theta \vdash_P \mu X.P : T} \quad \frac{}{\Theta \vdash_P \text{if } e \text{ then } P_1 \text{ else } P_2 : T} \\
 \frac{[\text{T-SUB}]}{\Theta \vdash_P P : T \quad T \leqslant T'} \quad \frac{[\text{T-IN}]}{\forall i \in I, \quad \Theta, x_i : S_i \vdash_P P_i : T_i}{\Theta \vdash_P \sum_{i \in I} p? \ell_i(x_i).P_i : p \& \{\ell_i(S_i).T_i\}_{i \in I}} \quad \frac{[\text{T-OUT}]}{\Theta \vdash_P e : S \quad \Theta \vdash_P P : T}{\Theta \vdash_P p! \ell(e).P : p \oplus \{\ell(S).T\}}
 \end{array}$$

Table 5 Typing processes

492 Table 4 and Table 5 state the standard typing rules for expressions and processes which  
493 we don't elaborate on. We have a single rule for typing sessions:

$$\frac{[\text{T-SESS}]}{\forall i \in I : \quad \vdash_P P_i : \Gamma(p_i) \quad \Gamma \sqsubseteq G}{\Gamma \vdash_M \prod_i p_i \triangleleft P_i}$$

495 [T-SESS] says that a session made of the parallel composition of processes  $\prod_i p_i \triangleleft P_i$  can  
496 be typed by an associated local context  $\Gamma$  if the local type of participant  $p_i$  in  $\Gamma$  types the  
497 process

## 498 6.2 Properties of Typed Sessions

give theorem 499 no 500 The subject reduction, progress and non-stuck theorems from [14] also hold in this setting,  
with minor changes in their statements and proofs. We won't discuss these proofs in detail.

501 ▶ **Lemma 6.1.** If  $\gamma \vdash_M M$  and  $M \Rightarrow M'$ , then  $\text{typ\_sess } M' \text{ } \gamma$ .

502 ▶ **Theorem 6.2 (Subject Reduction)**. If  $\gamma \vdash_M M$  and  $M \xrightarrow{(p,q)\ell} M'$ , then there exists a  
503 typing context  $\gamma'$  such that  $\gamma \xrightarrow{(p,q)\ell} \gamma'$  and  $\gamma \vdash_M M$ .

504 ▶ **Remark 6.3.** Note that in Theorem 6.2 one transition between sessions corresponds to  
505 exactly one transition between local type contexts with the same label. That is, every session  
506 transition is observed by the corresponding type. This is the main reason for our choice of  
507 reactive semantics (Section 2.2) as  $\tau$  transitions are not observed by the type in ordinary  
508 semantics. In other words, with  $\tau$ -semantics the typing relation is a *weak simulation* [29],  
509 while it turns into a strong simulation with reactive semantics. For our Rocq implementation  
510 working with the strong simulation turns out to be more convenient.

511 ▶ **Theorem 6.4 (Deadlock Freedom)**. If  $\gamma \vdash_M M$ , one of the following hold :

- 512 1. Either  $M \Rightarrow M_{\text{inact}}$  where every process making up  $M_{\text{inact}}$  is inactive, i.e.  $M_{\text{inact}} \equiv \prod_{i=1}^n p_i \triangleleft \mathbf{0}$  for some  $n$ .
- 514 2. Or there is a  $M'$  such that  $M \rightarrow M'$ .

515 We can also prove the following correspondence result in the reverse direction to Theorem 6.2,  
516 analogous to Theorem 4.9.

517 ▶ **Theorem 6.5 (Session Fidelity)**. If  $\gamma \vdash_M M$  and  $\gamma \xrightarrow{(p,q)\ell} \gamma'$ , there exists  
518 a message label  $\ell'$ , a context  $\gamma''$  and a session  $M'$  such that  $M \xrightarrow{(p,q)\ell'} M'$ ,  $\gamma \xrightarrow{(p,q)\ell'} \gamma''$   
519 and  $\text{typ\_sess } M' \text{ } \gamma''$ .

520 ► Remark 6.6. Again we note that by Theorem 6.5 a single-step context reduction induces a  
 521 single-step session reduction on the type. With the  $\tau$ -semantics the session reduction induced  
 522 by the context reduction would be multistep.

523 Now the following type safety property follows from the above theorems:

524 ► **Theorem 6.7** (Type Safety  $\models$ ). *If  $\gamma \vdash_M M$  and  $M \rightarrow^* M' \Rightarrow (p \leftarrow p\_send q \text{ ell } P$   
 525  $\parallel\parallel q \leftarrow p\_recv p \text{ xs } \parallel\parallel M')$ , then  $\text{onth ell xs} \neq \text{None}$ .*

526 The final, and the most intricate, session property we prove is liveness.

527 ► **Definition 6.8** (Session Liveness). *Session  $\mathcal{M}$  is live iff*

- 528 1.  $\mathcal{M} \rightarrow^* \mathcal{M}' \Rightarrow q \triangleleft p! \ell_i(x_i).Q \mid \mathcal{N}$  implies  $\mathcal{M}' \rightarrow^* \mathcal{M}'' \Rightarrow q \triangleleft Q \mid \mathcal{N}'$  for some  $\mathcal{M}'', \mathcal{N}'$
- 529 2.  $\mathcal{M} \rightarrow^* \mathcal{M}' \Rightarrow q \triangleleft \bigwedge_{i \in I} p? \ell_i(x_i).Q_i \mid \mathcal{N}$  implies  $\mathcal{M}' \rightarrow^* \mathcal{M}'' \Rightarrow q \triangleleft Q_i[v/x_i] \mid \mathcal{N}'$  for some  
 530  $\mathcal{M}'', \mathcal{N}', i, v$ .

531 In Rocq this is expressed with the predicate `live_sess`  $\models$ :

```
532 Definition live_sess Mp  $\triangleq$   $\forall M, \text{betaRtc Mp } M \rightarrow$ 
   $(\forall p q \text{ ell } e P' M', p \neq q \rightarrow \text{unfoldP } M ((p \leftarrow p\_send q \text{ ell } e P') \parallel\parallel M') \rightarrow \exists M'',$ 
   $\text{betaRtc } M ((p \leftarrow P') \setminus \setminus M'))$ 
   $\wedge$ 
   $(\forall p q \text{ llp } M', p \neq q \rightarrow \text{unfoldP } M ((p \leftarrow p\_recv q \text{ llp}) \parallel\parallel M') \rightarrow$ 
   $\exists M'', P' \in K, \text{onth } k \text{ llp} = \text{Some } P' \wedge \text{betaRtc } M ((p \leftarrow \text{subst\_expr\_proc } P', e \circ o) \parallel\parallel M'')).$ 
```

532

533 Session liveness, analogous to liveness for typing contexts (Definition 5.4), says that when  
 534  $\mathcal{M}$  is live, if  $\mathcal{M}$  reduces to a session  $\mathcal{M}'$  containing a participant that's attempting to send  
 535 or receive, then  $\mathcal{M}'$  reduces to a session where that communication has happened. It's also  
 536 called *lock-freedom* in related work ([41, 30]).

537 We can now prove that typed sessions are live. First we prove the following lemma:

538 ► **Lemma 6.9** (Fair Extension of Typed Sessions  $\models$ ). *If `typ_sess M gamma`, then there exists a  
 539 session reduction path `xs` starting from `M` such that the following fairness property holds:*

- 540 ■ *On `xs`, whenever a transition with label  $(p, q)\ell$  is enabled, a transition with label  $(p, q)\ell'$   
 541 eventually occurs for some  $\ell'$ .*

542 **Proof.** The desired path can be constructed by repeatedly cycling through all participants,  
 543 checking if there is a transition involving that participant, and executing that transition if  
 544 there is. Correctness follows from Theorem 6.2 and Theorem 6.5. ◀

545 Lemma 6.9 defines a "fairness" property for sessions analogous to Definition 5.4. It then  
 546 shows that there exists a fair path from any typable session. This resembles the *feasibility*  
 547 property expected from sensible notions of fairness [42], which states that any partial path  
 548 can be extended into a fair one <sup>3</sup>.

549 ► **Remark 6.10.** As in the proof of Theorem 5.8, the construction in Lemma 6.9 uses the  
 550 `constructive_indefinite_description` axiom to construct a **CoFixpoint**. Additionally, we  
 551 use the axiom `excluded_middle_informative` for the "check if there is a transition involving a  
 552 participant" part of the scheduling algorithm. The use of this axiom is probably not necessary  
 553 but it makes the proof easier.

---

<sup>3</sup> Note that this fairness property for sessions is not actually feasible as there are partial paths starting with an untypable session that can't be extended into a fair one. Nevertheless, Lemma 6.9 turns out to be enough to prove our liveness property.

554 ► **Theorem 6.11** (Liveness by Typing). *For a session  $M_p$ , if  $\exists \gamma \vdash_M M_p$  then  
 555  $\text{live\_sess } M_p$ .*

556 **Proof.** We detail the proof for the send case of Definition 6.8, the case for the receive is  
 557 similar. Suppose that  $M_p \rightarrow^* M$  and  $M \Rightarrow ((p \leftarrow p_{\text{send}} q \text{ ell } e P') ||| M')$ . Our goal is  
 558 to show that there exists a  $M''$  such that  $M \rightarrow^* ((p \leftarrow P') ||| M'')$ . First, observe that  
 559 by [R-UNFOLD] it suffices to show that  $((p \leftarrow p_{\text{send}} q \text{ ell } e P') ||| M') \rightarrow^* M''$  for  
 560 some  $M''$ . Also note that  $\gamma \vdash_M M$  for some  $\gamma$  by Theorem 6.2, therefore  $\gamma \vdash_M ((p \leftarrow p_{\text{send}} q \text{ ell } e P') ||| M')$  by Lemma 6.1.

561 Now let  $xs$  be a fair session reduction path starting from  $((p \leftarrow p_{\text{send}} q \text{ ell } e P') ||| M')$ , which exists by Lemma 6.9. By Theorem 6.2, let  $ys$  be a local type context  
 562 reduction path starting with  $\gamma$  such that every session in  $xs$  is typed by the context at  
 563 the corresponding index of  $ys$ , and the transitions of  $xs$  and  $ys$  at every step match. Now it  
 564 can be shown that  $ys$  is fair . Therefore by Theorem 5.8  $ys$  is live, so a  $\text{lcomm } p \ q \ \text{ell}$ ,  
 565 transition eventually occurs in  $ys$  for some  $\text{ell}'$ . Therefore  $ys = \gamma \xrightarrow{*} \gamma_0 \xrightarrow{(p,q)\ell'} \gamma_1 \rightarrow \dots$  for some  $\gamma_0, \gamma_1$ . Now consider the session  $M_0$  typed by  $\gamma_0$  in  
 566  $xs$ . We have  $((p \leftarrow p_{\text{send}} q \text{ ell } e P') ||| M') \rightarrow^* M_0$  by  $M_0$  being on  $xs$ . We also have  
 567 that  
 568  $M_0 \xrightarrow{(p,q)\ell''} M_1$  for some  $\ell''$ ,  $M_1$  by Theorem 6.5. Now observe that  $M_0 \equiv ((p \leftarrow p_{\text{send}} q \text{ ell } e P') ||| M'')$  for some  $M''$  as no transitions involving  $p$  have happened on the reduction  
 569 path to  $M_0$ . Therefore  $\ell = \ell''$ , so  $M_1 \equiv ((p \leftarrow P') ||| M'')$  for some  $M''$ , as needed. ◀

## 574 7 Conclusion and Related Work

575 **Liveness Properties.** Examinations of liveness, also called *lock-freedom*, guarantees of  
 576 multiparty session types abound in literature, e.g. [31, 23, 44, 35, 3]. Most of these papers use  
 577 the definition liveness proposed by Padovani [30], which doesn't make the fairness assumptions  
 578 that characterize the property [16] explicit. Contrastingly, van Glabbeek et. al. [41] examine  
 579 several notions of fairness and the liveness properties induced by them, and devise a type  
 580 system with flexible choices [6] that captures the strongest of these properties, the one  
 581 induced by the *justness* [42] assumption. In their terminology, Definition 6.8 corresponds  
 582 to liveness under strong fairness of transitions (ST), which is the weakest of the properties  
 583 considered in that paper. They also show that their type system is complete i.e. every live  
 584 process can be typed. We haven't presented any completeness results in this paper. Indeed,  
 585 our type system is not complete for Definition 6.8, even if we restrict our attention to safe  
 586 and race-free sessions. For example, the session described in [41, Example 9] is live but not  
 587 typable by a context associated with a balanced global type in our system.

588 Fairness assumptions are also made explicit in recent work by Ciccone et. al [10, 11]  
 589 which use generalized inference systems with coaxioms [1] to characterize *fair termination*,  
 590 which is stronger than Definition 6.8, but enjoys good composition properties.

591 **Mechanisation.** Mechanisation of session types in proof assistants is a relatively new  
 592 effort. Our formalisation is built on recent work by Ekici et. al. [14] which uses a coinductive  
 593 representation of global and local types to prove subject reduction and progress. Their work  
 594 uses a typing relation between global types and sessions while ours uses one between associated  
 595 local type contexts and sessions. This necessitates the rewriting of subject reduction and  
 596 progress proofs in addition to the operational correspondence, safety and liveness properties  
 597 we have proved. Other recent results mechanised in Rocq include Ekici and Yoshida's [15]  
 598 work on the completeness of asynchronous subtyping, and Tirore's work [37, 39, 38] on  
 599 projections and subject reduction for  $\pi$ -calculus.

600 Castro-Perez et. al. [8] devise a multiparty session type system that dispenses with  
601 projections and local types by defining the typing relation directly on the LTS specifying the  
602 global protocol, and formalise the results in Agda. Ciccone's PhD thesis [9] presents an Agda  
603 formalisation of fair termination for binary session types. Binary session types were also  
604 implemented in Agda by Thiemann [36] and in Idris by Brady[5]. Several implementations  
605 of binary session types are also present for Haskell [24, 28, 34].

606 Implementations of session types that are more geared towards practical verification  
607 include the Actris framework [18, 21] which enriches the separation logic of Iris [22] with  
608 binary session types to certify deadlock-freedom. In general, verification of liveness properties,  
609 with or without session types, in concurrent separation logic is an active research area that  
610 has produced tools such as TaDa [13], FOS [25] and LiLo [26] in the past few years. Further  
611 verification tools employing multiparty session types are Jacobs's Multiparty GV [21] based  
612 on the functional language of Wadler's GV [43], and Castro-Perez et. al's Zooid [7], which  
613 supports the extraction of certifiably safe and live protocols.

---

614 ————— References —————

- 615 1 Davide Ancona, Francesco Dagnino, and Elena Zucca. Generalizing Inference Systems by  
616 Coaxioms. In Hongseok Yang, editor, *Programming Languages and Systems*, pages 29–55,  
617 Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.
- 618 2 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and  
619 Mind Series)*. The MIT Press, 2008.
- 620 3 Franco Barbanera and Mariangiola Dezani-Ciancaglini. Partially Typed Multiparty Ses-  
621 sions. *Electronic Proceedings in Theoretical Computer Science*, 383:15–34, August 2023.  
622 arXiv:2308.10653 [cs]. URL: <http://arxiv.org/abs/2308.10653>, doi:10.4204/EPTCS.383.2.
- 623 4 Yves Bertot and Pierre Castran. *Interactive Theorem Proving and Program Development:  
624 Coq'Art The Calculus of Inductive Constructions*. Springer Publishing Company, Incorporated,  
625 1st edition, 2010.
- 626 5 Edwin Charles Brady. Type-driven Development of Concurrent Communicating Systems.  
627 *Computer Science*, 18(3), July 2017. URL: [https://journals.agh.edu.pl/csci/article/  
628 view/1413](https://journals.agh.edu.pl/csci/article/view/1413), doi:10.7494/csci.2017.18.3.1413.
- 629 6 Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Paola Giannini. Reversible sessions  
630 with flexible choices. *Acta Informatica*, 56(7):553–583, November 2019. doi:10.1007/  
631 s00236-019-00332-y.
- 632 7 David Castro-Perez, Francisco Ferreira, Lorenzo Gheri, and Nobuko Yoshida. Zooid: a dsl for  
633 certified multiparty computation: from mechanised metatheory to certified multiparty processes.  
634 In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language  
635 Design and Implementation*, PLDI 2021, page 237–251, New York, NY, USA, 2021. Association  
636 for Computing Machinery. doi:10.1145/3453483.3454041.
- 637 8 David Castro-Perez, Francisco Ferreira, and Sung-Shik Jongmans. A synthetic reconstruction  
638 of multiparty session types. *Proc. ACM Program. Lang.*, 10(POPL), January 2026. doi:  
639 10.1145/3776692.
- 640 9 Luca Ciccone. Concerto grosso for sessions: Fair termination of sessions, 2023. URL: <https://arxiv.org/abs/2307.05539>, arXiv:2307.05539.
- 642 10 Luca Ciccone, Francesco Dagnino, and Luca Padovani. Fair termination of multi-  
643 party sessions. *Journal of Logical and Algebraic Methods in Programming*, 139:100964,  
644 2024. URL: <https://www.sciencedirect.com/science/article/pii/S2352220824000221>,  
645 doi:10.1016/j.jlamp.2024.100964.
- 646 11 Luca Ciccone and Luca Padovani. Fair termination of binary sessions. *Proc. ACM Program.  
647 Lang.*, 6(POPL), January 2022. doi:10.1145/3498666.

- 648    12 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction  
649    to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.
- 650    13 Emanuele D'Osualdo, Julian Sutherland, Azadeh Farzan, and Philippa Gardner. Tada live:  
651    Compositional reasoning for termination of fine-grained concurrent programs. *ACM Trans.  
652    Program. Lang. Syst.*, 43(4), November 2021. doi:10.1145/3477082.
- 653    14 Burak Ekici, Tadayoshi Kamegai, and Nobuko Yoshida. Formalising Subject Reduction and  
654    Progress for Multiparty Session Processes. In Yannick Forster and Chantal Keller, editors, *16th  
655    International Conference on Interactive Theorem Proving (ITP 2025)*, volume 352 of *Leibniz  
656    International Proceedings in Informatics (LIPICS)*, pages 19:1–19:23, Dagstuhl, Germany,  
657    2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: <https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITP.2025.19>. doi:10.4230/LIPIcs.ITP.2025.19.
- 659    15 Burak Ekici and Nobuko Yoshida. Completeness of Asynchronous Session Tree Subtyping  
660    in Coq. In Yves Bertot, Temur Kutsia, and Michael Norrish, editors, *15th International  
661    Conference on Interactive Theorem Proving (ITP 2024)*, volume 309 of *Leibniz International  
662    Proceedings in Informatics (LIPICS)*, pages 13:1–13:20, Dagstuhl, Germany, 2024. Schloss  
663    Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 1868-8969. URL: <https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITP.2024.13>. doi:10.4230/LIPIcs.ITP.2024.13.
- 665    16 Nissim Francez. *Fairness*. Springer US, New York, NY, 1986. URL: <http://link.springer.com/10.1007/978-1-4612-4886-6>. doi:10.1007/978-1-4612-4886-6.
- 667    17 Silvia Ghilezan, Svetlana Jakšić, Jovanka Pantović, Alceste Scalas, and Nobuko Yoshida.  
668    Precise subtyping for synchronous multiparty sessions. *Journal of Logical and Algebraic Meth-  
669    ods in Programming*, 104:127–173, 2019. URL: <https://www.sciencedirect.com/science/article/pii/S2352220817302237>. doi:10.1016/j.jlamp.2018.12.002.
- 671    18 Jonas Kastberg Hinrichsen, Jesper Bengtson, and Robbert Krebbers. Actris: Session-type  
672    based reasoning in separation logic. *Proceedings of the ACM on Programming Languages*,  
673    4(POPL):1–30, 2019.
- 674    19 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types.  
675    *SIGPLAN Not.*, 43(1):273–284, January 2008. doi:10.1145/1328897.1328472.
- 676    20 Chung-Kil Hur, Georg Neis, Derek Dreyer, and Viktor Vafeiadis. The power of parameterization  
677    in coinductive proof. *SIGPLAN Not.*, 48(1):193–206, January 2013. doi:10.1145/2480359.  
678    2429093.
- 679    21 Jules Jacobs, Jonas Kastberg Hinrichsen, and Robbert Krebbers. Deadlock-free separation  
680    logic: Linearity yields progress for dependent higher-order message passing. *Proceedings of the  
681    ACM on Programming Languages*, 8(POPL):1385–1417, 2024.
- 682    22 Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Aleš Bizjak, Lars Birkedal, and Derek  
683    Dreyer. Iris from the ground up: A modular foundation for higher-order concurrent separation  
684    logic. *Journal of Functional Programming*, 28:e20, 2018.
- 685    23 Naoki Kobayashi. A Type System for Lock-Free Processes. *Information and Computation*,  
686    177(2):122–159, September 2002. URL: <https://www.sciencedirect.com/science/article/pii/S0890540102931718>. doi:10.1006/inco.2002.3171.
- 688    24 Wen Kokke and Ornella Dardha. Deadlock-free session types in linear haskell. In *Proceedings of  
689    the 14th ACM SIGPLAN International Symposium on Haskell*, Haskell 2021, page 1–13, New  
690    York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3471874.3472979.
- 691    25 Dongjae Lee, Minki Cho, Jinwoo Kim, Soonwon Moon, Youngju Song, and Chung-Kil Hur.  
692    Fair operational semantics. *Proc. ACM Program. Lang.*, 7(PLDI), June 2023. doi:10.1145/  
693    3591253.
- 694    26 Dongjae Lee, Janggun Lee, Taeyoung Yoon, Minki Cho, Jeehoon Kang, and Chung-Kil Hur.  
695    Lilo: A higher-order, relational concurrent separation logic for liveness. *Proceedings of the  
696    ACM on Programming Languages*, 9(OOPSLA1):1267–1294, 2025.
- 697    27 Pierre Letouzey and Andrew W. Appel. Modular Finite Maps over Ordered Types. URL:  
698    <https://github.com/rocq-community/mmaps>.

- 699 28 Sam Lindley and J Garrett Morris. Embedding session types in haskell. *ACM SIGPLAN Notices*, 51(12):133–145, 2016.
- 700 29 Robin MILNER. Chapter 19 - operational and algebraic semantics of concurrent processes. In JAN VAN LEEUWEN, editor, *Formal Models and Semantics*, Handbook of Theoretical Computer Science, pages 1201–1242. Elsevier, Amsterdam, 1990. URL: <https://www.sciencedirect.com/science/article/pii/B978044488074150024X>, doi:10.1016/B978-0-444-88074-1.50024-X.
- 701 30 Luca Padovani. Deadlock and lock freedom in the linear pi-calculus. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, CSL-LICS ’14, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2603088.2603116.
- 702 31 Luca Padovani, Vasco Thudichum Vasconcelos, and Hugo Torres Vieira. Typing Liveness in Multiparty Communicating Systems. In Eva Kühn and Rosario Pugliese, editors, *Coordination Models and Languages*, pages 147–162, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
- 703 32 Kai Pischke and Nobuko Yoshida. *Asynchronous Global Protocols, Precisely*, pages 116–133. Springer Nature Switzerland, Cham, 2026. doi:10.1007/978-3-031-99717-4\_7.
- 704 33 Amir Pnueli. The temporal logic of programs. In *18th annual symposium on foundations of computer science (sfcs 1977)*, pages 46–57. ieee, 1977.
- 705 34 Riccardo Pucella and Jesse A Tov. Haskell session types with (almost) no class. In *Proceedings of the first ACM SIGPLAN symposium on Haskell*, pages 25–36, 2008.
- 706 35 Alceste Scalas and Nobuko Yoshida. Less is more: multiparty session types revisited. *Proc. ACM Program. Lang.*, 3(POPL), January 2019. doi:10.1145/3290343.
- 707 36 Peter Thiemann. Intrinsically-typed mechanized semantics for session types. In *Proceedings of the 21st International Symposium on Principles and Practice of Declarative Programming*, PPDP ’19, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3354166.3354184.
- 708 37 Dawit Tirole. A mechanisation of multiparty session types, 2024.
- 709 38 Dawit Tirole, Jesper Bengtson, and Marco Carbone. A sound and complete projection for global types. In *14th International Conference on Interactive Theorem Proving (ITP 2023)*, pages 28–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2023.
- 710 39 Dawit Tirole, Jesper Bengtson, and Marco Carbone. Multiparty asynchronous session types: A mechanised proof of subject reduction. In *39th European Conference on Object-Oriented Programming (ECOOP 2025)*, pages 31–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2025.
- 711 40 Thien Udomsrirungruang and Nobuko Yoshida. Top-down or bottom-up? complexity analyses of synchronous multiparty session types. *Proceedings of the ACM on Programming Languages*, 9(POPL):1040–1071, 2025.
- 712 41 Rob van Glabbeek, Peter Höfner, and Ross Horne. Assuming just enough fairness to make session types complete for lock-freedom. In *Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS ’21, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1109/LICS52264.2021.9470531.
- 713 42 Rob van Glabbeek and Peter Höfner. Progress, justness, and fairness. *ACM Computing Surveys*, 52(4):1–38, August 2019. URL: <http://dx.doi.org/10.1145/3329125>, doi:10.1145/3329125.
- 714 43 Philip Wadler. Propositions as sessions. *SIGPLAN Not.*, 47(9):273–286, September 2012. doi:10.1145/2398856.2364568.
- 715 44 Nobuko Yoshida and Ping Hou. Less is more revisited, 2024. URL: <https://arxiv.org/abs/2402.16741>, arXiv:2402.16741.