# Typing Scalas-Yoshida Benchmarks

Franco Barbanera
University of Catania (Italy)
franco.barbanera@unict.it

Mariangiola Dezani-Ciancaglini
University of Torino (Italy)
dezani@di.unito.it

Ugo de'Liguoro
University of Torino (Italy)
ugo.deliguoro@unito.it

**Abstract**

We show how the Scalas–Yoshida "Less Is More" benchmark examples for multiparty protocols can be represented and typed within the Simple Multiparty Sessions (SMPS) framework. We reconstruct suitable global types for the relevant protocols and provide typing derivations for them. The examples include service–client authentication, the recursive two-buyer protocol, recursive Map/Reduce, and independent multiparty workers. These derivations demonstrate that SMPS can type benchmark sessions that are problematic or unprojectable in standard MPST settings, while retaining the behavioural guarantees of the system, such as session fidelity and lock freedom.

## 1 Introduction

In these notes, we show how the relevant examples from [10], also known as the "Less Is More" benchmark [3], can be rephrased within the Simple MultiParty Sessions (SMPS) framework of [2] as typable sessions. There is a basic mismatch between the approaches in [10] and [2], since the former uses local-type contexts and does away with global types. In contrast, sessions are typed directly by global types in [2]. Nonetheless, it is possible to express the examples from [10] in the type system of [2] as follows. First, we observe that when there is a unique session, as in all the examples in Figure 4 of [10], local types in [10] are akin to processes

in [2]. In the setting of [2], the typing of messages in [10] is irrelevant, since only values of ground types can be exchanged in a unique session. Then, we reconstruct the global type of the resulting session in the terms of [2].

The typability of the "Less is More" benchmark examples has recently been shown in [3] within a framework that approaches projectability from a typing perspective. In that approach, global types are assigned to individual participants (which, in the process calculus of [3], can carry values in messages), rather than to entire multiparty sessions, as in [2].

The results of [10] have recently been rephrased in a setting with global types and projections [8], where the projection operator is defined inductively to enable feasible implementation. A coinductively defined projection would make the global types of the benchmark examples projectable.

The benchmark examples, along with their representation and typing in the SMPS formalism of [2], are presented in Section 3 after a brief review of the formalism in Section 2.

# 2 SMPS calculus and type system

We briefly recall the main definitions of SMPS from [1] and [2]. For the calculus, global types, and type system, the following base sets and notation are used: *labels*, ranged over by $\lambda, \lambda', \ldots$; *participant names*, ranged over by $\mathsf{p}, \mathsf{q}, \mathsf{r}, \mathsf{s}, \ldots$; *processes*, ranged over by $P, Q, R, S, \ldots$; *sessions*, ranged over by $\mathbb{M}, \mathbb{M}', \ldots$; *global types*, ranged over by $\mathsf{G}, \mathsf{G}', \ldots$; *integers*, ranged over by $i, j, l, h, k, \ldots$; *(finite) integer sets*, ranged over by $I, J, \ldots$

**Definition 2.1** (Processes)**.** Processes *are defined by:*

$$P \quad ::=_{\rho} \quad \mathbf{0} \mid \mathsf{p}!\{\lambda_i.P_i\}_{i\in I} \mid \mathsf{p}?\{\lambda_i.P_i\}_{i\in I}$$

*where $I \neq \emptyset$ is finite and $\lambda_h \neq \lambda_k$ for $h, k \in I$ and $h \neq k$. We restrict the set of processes to the regular ones, i.e., terms having finitely many distinct subterms.*

The productions in the above definition are interpreted *coinductively*, as indicated by the symbol $::=_{\rho}$. This means that the set of processes is the greatest fixed point of the (monotonic) functor over sets defined by the grammar, restricted, however, to regular processes. Thus, processes may be infinite.

**Definition 2.2** (Multiparty Sessions)**.** Multiparty sessions *(*sessions *for short) are defined by:*

$$\mathbb{M} = \mathsf{p}_1[\, P_1 \,] \parallel \cdots \parallel \mathsf{p}_n[\, P_n \,]$$

*with $n \geq 1$ and $\mathsf{p}_h \neq \mathsf{p}_k$ for any $h \neq k$. The set of* participants *of a session $\mathbb{M}$,* $\mathsf{prt}(\mathbb{M})$*, is defined as* $\mathsf{prt}(\mathsf{p}_1[\, P_1 \,] \parallel \cdots \parallel \mathsf{p}_n[\, P_n \,]) = \{\mathsf{p}_i \mid P_i \neq \mathbf{0} \ \ \&\ \ 1 \leq i \leq n\}$

Because of the condition $\mathsf{p}_h \neq \mathsf{p}_k$ for any $h \neq k$, a session is essentially a finite set of (not necessarily distinct) processes $P_i$ located at distinct participants $\mathsf{p}_i$. To make this formal, over sessions we define the *structural congruence relation* $\mathbb{M} \equiv \mathbb{M}'$, which is the least congruence under which the parallel composition is commutative and associative, and such that, for all $\mathbb{M}$ and fresh $\mathsf{p}$, we have $\mathsf{p}[\,\mathbf{0}\,] \parallel \mathbb{M} \equiv \mathbb{M}$. This implies that $\mathsf{p}[\,\mathbf{0}\,] \equiv \mathsf{p}[\,\mathbf{0}\,] \parallel \mathsf{q}[\,\mathbf{0}\,] \equiv \mathsf{q}[\,\mathbf{0}\,]$ for all distinct $\mathsf{p}, \mathsf{q}$. In a sense, any session of the shape $\mathsf{p}[\,\mathbf{0}\,]$ represents the empty session. We omit writing trailing $\mathbf{0}$'s in processes and denote $\mathsf{p}!\{\lambda.P\}$ and $\mathsf{p}?\{\lambda.P\}$ by $\mathsf{p}!\lambda.P$ and $\mathsf{p}?\lambda.P$, respectively.

The (synchronous) operational semantics of sessions is formally defined by the following labelled transition system.

**Definition 2.3** (Session LTS)**.** *A* communication action *is a triple* $\mathsf{p}\lambda\mathsf{q}$*, where* $\mathsf{p} \neq \mathsf{q}$*. The* labelled transition system (LTS) for multiparty sessions*, with communication actions as labels, is the closure under structural congruence of the reduction specified by the following axiom:*

$$\mathsf{p}[\,P\,] \parallel \mathsf{q}[\,Q\,] \parallel \mathbb{M} \xrightarrow{\mathsf{p}\lambda_k\mathsf{q}} \mathsf{p}[\,P_k\,] \parallel \mathsf{q}[\,Q_k\,] \parallel \mathbb{M} \quad (k \in I \subseteq J) \quad \text{[Comm]}$$

*where* $P = \mathsf{q}!\{\lambda_i.P_i\}_{i\in I}$ *and* $Q = \mathsf{p}?\{\lambda_j.Q_j\}_{j\in J}$*.*

Axiom [Comm] above is non-deterministic in the choice of messages and makes the communication possible: participant $\mathsf{p}$ sends message $\lambda_k$ to participant $\mathsf{q}$. The sender can freely choose the message because the condition $I \subseteq J$ ensures that the receiver must offer all sender messages and possibly more. This allows distinguishing between internal and external choices in the operational semantics. This condition is always satisfied in well-typed sessions.

When $\sigma = \Lambda_1 \cdot \ldots \cdot \Lambda_n$ $(n \geq 0)$, we write $\mathbb{M} \xrightarrow{\sigma} \mathbb{M}'$ as shorthand for

$$\mathbb{M} \xrightarrow{\Lambda_1} \mathbb{M}_1 \cdots \xrightarrow{\Lambda_n} \mathbb{M}_n = \mathbb{M}'$$

where the $\Lambda_i$s range over communication actions.

We write $\mathbb{M} \xrightarrow{\sigma}$ to mean that $\mathbb{M} \xrightarrow{\sigma} \mathbb{M}'$ for some $\mathbb{M}'$. Moreover, we define $\mathsf{prt}(\mathsf{p}\lambda\mathsf{q}) = \{\mathsf{p}, \mathsf{q}\}$ and $\mathsf{prt}(\sigma)$ as its obvious extension to sequences of communication actions (traces).

Global types are usually represented by $\mu$-expressions, as in [6, 7] and in almost all papers on "classical" MultiParty Session Types (MPST). Instead, we define global types coinductively as potentially infinite regular terms.

**Definition 2.4** (Global types)**.** Global types *are defined by:*

$$\mathsf{G} ::=_\rho \mathsf{End} \mid \mathsf{p} \to \mathsf{q} : \{\lambda_i.\mathsf{G}_i\}_{i\in I}$$

*where* $I \neq \emptyset$ *and* $\lambda_i \neq \lambda_j$ *for* $i, j \in I$ *and* $i \neq j$*. We restrict the set of global types to the regular ones. Given a global type* $\mathsf{G}$*, the set of its* participants*,* $\mathsf{prt}(\mathsf{G})$*, is defined as the smallest set satisfying the following equations:*

$$\mathsf{prt}(\mathsf{End}) = \emptyset \qquad \mathsf{prt}(\mathsf{p} \to \mathsf{q} : \{\lambda_i.\mathsf{G}_i\}_{i\in I}) = \{\mathsf{p},\mathsf{q}\} \cup \bigcup_{i\in I} \mathsf{prt}(\mathsf{G}_i)$$

Since the definition of $\mathsf{G}$ is coinductive, so is the definition of $\mathsf{prt}(\mathsf{G})$. Because the syntactic tree of $\mathsf{G}$ is regular, $\mathsf{prt}(\mathsf{G})$ is well-defined and finite. In the following, trailing $\mathsf{End}$'s will be omitted, and $\mathsf{p} \to \mathsf{q} : \{\lambda.\mathsf{G}\}$ will be written as $\mathsf{p} \to \mathsf{q} : \lambda.\mathsf{G}$.

Now we recall the SMPS type assignment system, as defined in [1]. The (synchronous) calculus of SMPS, because of its simplicity, enables us to devise a type system in which global types are directly inferred for sessions. Double lines in the rules indicate that they are interpreted coinductively, following [9].

**Definition 2.5** (Type System)**.** *Judgements of the form* $\mathsf{G} \vdash \mathbb{M}$ *are coinductively derived by the type system below, by considering sessions up to structural congruence:*

$$\mathsf{End} \vdash \mathsf{p}[\,\mathbf{0}\,] \quad \text{[T-End]}$$

$$\frac{\begin{array}{c}\mathsf{G}_i \vdash \mathsf{p}[\,P_i\,] \parallel \mathsf{q}[\,Q_i\,] \parallel \mathbb{M} \\ \mathsf{prt}(\mathsf{G}_i) \setminus \{\mathsf{p},\mathsf{q}\} = \mathsf{prt}(\mathbb{M}) \quad \forall i \in I \subseteq J\end{array}}{\mathsf{p} \to \mathsf{q} : \{\lambda_i.\mathsf{G}_i\}_{i\in I} \vdash \mathsf{p}[\,\mathsf{q}!\{\lambda_i.P_i\}_{i\in I}\,] \parallel \mathsf{q}[\,\mathsf{p}?\{\lambda_j.Q_j\}_{j\in J}\,] \parallel \mathbb{M}} \text{ [T-Comm]}$$

Rule [T-Comm] adds simultaneous communications to global types and to the corresponding processes within sessions. This rule allows more inputs than corresponding outputs, in agreement with the condition in Rule [Comm] (Definition 2.3). It also allows more branches in the input process than in the global type, partially mimicking local type subtyping [4].

It is convenient to associate with a global type the set of communication actions that could label its transitions. We call these the *capabilities* of the global type.

**Definition 2.6** (Capabilities)**.** *Let* $\mathsf{G}$ *be a type. The set* $\mathsf{cap}(\mathsf{G})$ *of the* capabilities *of* $\mathsf{G}$ *is the smallest set satisfying the following equations:*

$$\mathsf{cap}(\mathsf{End}) = \emptyset$$
$$\mathsf{cap}(\mathsf{p} \to \mathsf{q} : \{\lambda_i.\mathsf{G}_i\}_{i\in I}) = \{\mathsf{p}\lambda_i\mathsf{q} \mid i \in I\} \cup \bigcup_{i\in I} \mathsf{cap}(\mathsf{G}_i)$$

By regularity, the set $\mathsf{cap}(\mathsf{G})$ is finite for all $\mathsf{G}$.

The properties of the type system use the following LTS for global types.

**Definition 2.7** (Coinductive LTS for global types)**.**

$$\mathsf{p} \to \mathsf{q} : \{\lambda_i.\mathsf{G}_i\}_{i\in I} \xrightarrow{\mathsf{p}\lambda_j\mathsf{q}} \mathsf{G}_j \;\; (j \in I) \;\; \text{[E-Comm]}$$

$$\frac{\begin{array}{c}\mathsf{G}_i \xrightarrow{\mathsf{p}\lambda\mathsf{q}} \mathsf{G}'_i \qquad \forall i \in I \\ \{\mathsf{p},\mathsf{q}\} \cap \{\mathsf{r},\mathsf{s}\} = \emptyset \qquad \mathsf{p}\lambda\mathsf{q} \in \bigcap_{i\in I} \mathsf{cap}(\mathsf{G}_i)\end{array}}{\mathsf{r} \to \mathsf{s} : \{\lambda_i.\mathsf{G}_i\}_{i\in I} \xrightarrow{\mathsf{p}\lambda\mathsf{q}} \mathsf{r} \to \mathsf{s} : \{\lambda_i.\mathsf{G}'_i\}_{i\in I}} \text{ [I-Comm]}$$

**Theorem 2.8** (Properties of typable sessions [2])**.** *Let* $\mathsf{G}$, $\mathbb{M}$ *be such that* $\mathsf{G} \vdash \mathbb{M}$.

1. **Subject Reduction:** *If* $\mathbb{M} \xrightarrow{\mathsf{p}\lambda\mathsf{q}} \mathbb{M}'$, *then* $\mathsf{G} \xrightarrow{\mathsf{p}\lambda\mathsf{q}} \mathsf{G}'$ *and* $\mathsf{G}' \vdash \mathbb{M}'$ *for some* $\mathsf{G}'$.
2. **Session Fidelity:** *If* $\mathsf{G} \xrightarrow{\mathsf{p}\lambda\mathsf{q}} \mathsf{G}'$, *then* $\mathbb{M} \xrightarrow{\mathsf{p}\lambda\mathsf{q}} \mathbb{M}'$ *and* $\mathsf{G}' \vdash \mathbb{M}'$ *for some* $\mathbb{M}'$.
3. **Lock Freedom:** $\mathbb{M} \xrightarrow{\sigma} \mathbb{M}'$ *and* $\mathsf{p} \in \mathsf{prt}(\mathbb{M}')$ *imply* $\mathbb{M}' \xrightarrow{\sigma' \cdot \Lambda}$ *for some* $\sigma'$ *and* $\Lambda$ *such that* $\mathsf{p} \in \mathsf{prt}(\Lambda)$.

# 3 Typing the Scalas-Yoshida benchmark examples

The typings of the first two examples were already presented in [2] and are reported here for completeness. To improve readability, the conditions on participants will be written only in the derivation for the first example.

**Service, client and authentication protocol.** This is the example in the Introduction of [10], further detailed in Figure 4(1) of the same paper.

> The service s sends to the client c either a request to login or cancel. If login is issued, then c sends a password pwd to the authorisation server a and then s receives from a the authorisation auth. In case c receives cancel from s the interaction with a is aborted by c sending quit to a.

This session is encoded by

$$\mathbb{M}_{\mathsf{sca}} = \mathsf{s}[\, S \,] \parallel \mathsf{c}[\, C \,] \parallel \mathsf{a}[\, A \,]$$

where the processes are

$S = \mathsf{c}!\{\mathtt{login}.\mathsf{a}?\mathtt{auth}, \mathtt{cancel}\}$
$C = \mathsf{s}?\{\mathtt{login}.\mathsf{a}!\mathtt{pwd},\ \mathtt{cancel}.\mathsf{a}!\mathtt{quit}\}$
$A = \mathsf{c}?\{\mathtt{pwd}.\mathsf{s}!\mathtt{auth},\ \mathtt{quit}\}$

Then we derive $\mathsf{G}_{\mathsf{sca}} \vdash \mathbb{M}_{\mathsf{sca}}$ where

$$\begin{array}{rcl} \mathsf{G}_{\mathsf{sca}} & = & \mathsf{s} \to \mathsf{c} : \{\mathtt{login}.\mathsf{G}'_{\mathsf{sca}},\ \mathtt{cancel}.\mathsf{c} \to \mathsf{a} : \mathtt{quit}\} \\ \mathsf{G}'_{\mathsf{sca}} & = & \mathsf{c} \to \mathsf{a} : \mathtt{pwd}.\ \mathsf{a} \to \mathsf{s} : \mathtt{auth} \end{array}$$

as shown on the right of Figure 1, where $\mathtt{q} = \mathtt{quit}$, $\mathtt{p} = \mathtt{pwd}$, and $\mathtt{a} = \mathtt{auth}$.

We observe that $\mathsf{G}_{\mathsf{sca}}$ is exactly the global type derived for this session in [10]. In that paper, it is shown that $\mathsf{G}_{\mathsf{sca}}$ is projectable, but $S$ and $A$ do not have dual input/output communications. The problem is that the input/output behaviours between s and a depend on their inputs/outputs with c. More precisely, s and a exchange the message auth only when c sends login to s. Hence, this example is ruled out by the classical MPST system in [6, 7, 5]. In this case, it seems that our system addresses this inconsistency by directly embodying in Rule [T-Comm] the

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{\mathsf{End} \vdash \mathsf{s}[\,\mathbf{0}\,] \parallel \mathsf{c}[\,\mathbf{0}\,] \parallel \mathsf{a}[\,\mathbf{0}\,]}
\quad \mathsf{prt}(\mathsf{End}) \setminus \{\mathsf{a},\mathsf{s}\} = \emptyset = \mathsf{prt}(\mathsf{c}[\,\mathbf{0}\,])
}{\mathsf{a} \to \mathsf{s} : \mathrm{A} \vdash \mathsf{s}[\,\mathsf{a}?_{\mathrm{A}}\,] \parallel \mathsf{c}[\,\mathbf{0}\,] \parallel \mathsf{a}[\,\mathsf{s}!_{\mathrm{A}}\,]}
\quad \mathsf{prt}(\mathsf{a} \to \mathsf{s} : \mathrm{A}) \setminus \{\mathsf{c},\mathsf{a}\} = \{\mathsf{s}\} = \mathsf{prt}(\mathsf{s}[\,\mathsf{a}?_{\mathrm{A}}\,])
}{\mathsf{G}'_{\mathsf{sca}} \vdash \mathsf{s}[\,\mathsf{a}?_{\mathrm{A}}\,] \parallel \mathsf{c}[\,\mathsf{a}!_{\mathrm{P}}\,] \parallel \mathsf{a}[\,A\,]}
\quad \mathsf{prt}(\mathsf{G}'_{\mathsf{sca}}) \setminus \{\mathsf{s},\mathsf{c}\} = \mathsf{prt}(\mathsf{c} \to \mathsf{a} : \mathrm{Q}) \setminus \{\mathsf{s},\mathsf{c}\} = \{\mathsf{a}\} = \mathsf{prt}(\mathsf{a}[\,A\,])
}{\mathsf{G}_{\mathsf{sca}} \vdash \mathbb{M}_{\mathsf{sca}}}
$$

$$
\dfrac{
\dfrac{
\dfrac{}{\mathsf{End} \vdash \mathsf{s}[\,\mathbf{0}\,] \parallel \mathsf{c}[\,\mathbf{0}\,] \parallel \mathsf{a}[\,\mathbf{0}\,]}
\quad \mathsf{prt}(\mathsf{End}) \setminus \{\mathsf{a},\mathsf{c}\} = \emptyset = \mathsf{prt}(\mathsf{s}[\,\mathbf{0}\,])
}{\mathsf{c} \to \mathsf{a} : \mathrm{Q} \vdash \mathsf{s}[\,\mathbf{0}\,] \parallel \mathsf{c}[\,\mathsf{a}!_{\mathrm{Q}}\,] \parallel \mathsf{a}[\,A\,]}
}{\mathsf{G}_{\mathsf{sca}} \vdash \mathbb{M}_{\mathsf{sca}}}
$$

$$
\dfrac{
\dfrac{
\dfrac{}{\mathsf{End} \vdash \mathsf{a}[\,\mathbf{0}\,] \parallel \mathsf{s}[\,\mathbf{0}\,] \parallel \mathsf{b}[\,\mathbf{0}\,]}
}{\mathsf{G}_{\mathrm{Y}} \vdash \mathsf{a}[\,\mathsf{s}!_{\mathrm{B}}\,] \parallel \mathsf{s}[\,\mathsf{a}?_{\{\mathrm{B},\,\mathrm{N}\}}\,] \parallel \mathsf{b}[\,\mathbf{0}\,]}
\quad \mathcal{D}
}{\mathsf{b} \to \mathsf{a} : \{\mathrm{Y}.\mathsf{G}_{\mathrm{Y}},\ \mathrm{N}.\mathsf{G}'_{\mathsf{asb}}\} \vdash \mathbb{M}'_{\mathsf{asb}}}
\quad \mathsf{a} \to \mathsf{s} : \mathrm{N} \vdash \mathsf{a}[\,\mathsf{s}!_{\mathrm{N}}\,] \parallel \mathsf{s}[\,\mathsf{a}?_{\{\mathrm{B},\,\mathrm{N}\}}\,] \parallel \mathsf{b}[\,\mathbf{0}\,]
}{\mathsf{G}'_{\mathsf{asb}} \vdash \mathsf{a}[\,A'\,] \parallel \mathsf{s}[\,\mathsf{a}?_{\{\mathrm{B},\,\mathrm{N}\}}\,] \parallel \mathsf{b}[\,B\,]}
$$

$$
\dfrac{
\dfrac{
\dfrac{}{\mathsf{End} \vdash \mathsf{a}[\,\mathbf{0}\,] \parallel \mathsf{s}[\,\mathbf{0}\,] \parallel \mathsf{b}[\,\mathbf{0}\,]}
}{\mathsf{G}_{\mathrm{Y}} \vdash \mathsf{a}[\,\mathsf{s}!_{\mathrm{B}}\,] \parallel \mathsf{s}[\,\mathsf{a}?_{\{\mathrm{B},\,\mathrm{N}\}}\,] \parallel \mathsf{b}[\,\mathbf{0}\,]}
\quad \mathcal{D}
}{\mathsf{b} \to \mathsf{a} : \{\mathrm{Y}.\mathsf{G}_{\mathrm{Y}},\ \mathrm{N}.\mathsf{G}'_{\mathsf{asb}}\} \vdash \mathbb{M}'_{\mathsf{asb}}}
}{\mathsf{G}'_{\mathsf{asb}} \vdash \mathsf{a}[\,A'\,] \parallel \mathsf{s}[\,\mathsf{a}?_{\{\mathrm{B},\,\mathrm{N}\}}\,] \parallel \mathsf{b}[\,B\,]}
$$

$$
\mathsf{G}_{\mathsf{asb}} \vdash \mathbb{M}_{\mathsf{asb}}
$$

$$
\mathcal{D} = \dfrac{\mathsf{s} \to \mathsf{a} : \mathrm{P}.\mathsf{G}'_{\mathsf{asb}} \vdash \mathsf{a}[\,\mathsf{s}?_{\mathrm{P}}.\,A'\,] \parallel \mathsf{s}[\,\mathsf{a}!_{\mathrm{P}}.\,\mathsf{a}?_{\{\mathrm{B},\,\mathrm{N}\}}\,] \parallel \mathsf{b}[\,B\,]}{\mathsf{G}_{\mathsf{asb}} \vdash \mathbb{M}_{\mathsf{asb}}}
$$

where

$$
\mathsf{End} \vdash \mathsf{a}[\,\mathbf{0}\,] \parallel \mathsf{s}[\,\mathbf{0}\,] \parallel \mathsf{b}[\,\mathbf{0}\,]
$$

and $\mathbb{M}'_{\mathsf{asb}} = \mathsf{a}[\,\mathsf{b}?\{\mathrm{Y}.\,\mathsf{s}!_{\mathrm{B}},\ \mathrm{N}.\,A'\}\,] \parallel \mathsf{s}[\,\mathsf{a}?_{\{\mathrm{B},\,\mathrm{N}\}}\,] \parallel \mathsf{b}[\,\mathsf{a}!\{\mathrm{Y},\ \mathrm{N}.\,B\}\,]$

Figure 1: Derivations for $\mathsf{G}_{\mathsf{sca}} \vdash \mathbb{M}_{\mathsf{sca}}$ (right) and $\mathsf{G}_{\mathsf{asb}} \vdash \mathbb{M}_{\mathsf{asb}}$ (left).

guarantee that any implementation of the protocol $\mathbb{M}_{\mathsf{sca}}$ will be well behaved and lock-free.

**Recursive two-buyer protocol.** This is the example in Figure 4(2) of [10].

> After querying the store s for the price of some good, Alice, represented by role a, asks Bob, represented by b, to split the price. If Bob replies by issuing yes, then Alice sends buy to s; otherwise, she insists by asking Bob for a different splitting (the fact that the subsequent

proposals by Alice are actually different is not explicitly represented in the protocol below nor in the type of a in [10]). At any time, Alice might quit the protocol, sending esc to Bob and no to the store.

The protocol is encoded by the session

$$\mathbb{M}_{\mathsf{asb}} = \mathsf{a}[\,A\,] \parallel \mathsf{s}[\,S\,] \parallel \mathsf{b}[\,B\,]$$

where the processes are

$$A = \mathsf{s}!\mathtt{que}.\ \mathsf{s}?\mathtt{pri}.\ A'$$

$$A' = \mathsf{b}!\begin{cases}\mathtt{spl}.\ \mathsf{b}?\{\mathtt{yes}.\ \mathsf{s}!\mathtt{buy},\ \mathtt{no}.\ A'\}\\ \mathtt{esc}.\ \mathsf{s}!\mathtt{no}\end{cases}$$

$$S = \mathsf{a}?\mathtt{que}.\ \mathsf{a}!\mathtt{pri}.\ \mathsf{a}?\{\mathtt{buy},\ \mathtt{no}\}$$

$$B = \mathsf{a}?\begin{cases}\mathtt{spl}.\ \mathsf{a}!\{\mathtt{yes},\ \mathtt{no}.\ B\}\\ \mathtt{esc}\end{cases}$$

The session $\mathbb{M}_{\mathsf{asb}}$ can be typed by

$$\mathsf{G}_{\mathsf{asb}} = \mathsf{a} \to \mathsf{s} : \mathtt{que}.\mathsf{s} \to \mathsf{a} : \mathtt{pri}.\mathsf{G}'_{\mathsf{asb}}$$

where

$$\mathsf{G}'_{\mathsf{asb}} = \mathsf{a} \to \mathsf{b} : \begin{cases}\mathtt{split}.\mathsf{b} \to \mathsf{a} : \{\mathtt{yes}.\mathsf{G}_{\mathtt{yes}}, \mathtt{no}.\mathsf{G}'_{\mathsf{asb}}\},\\ \mathtt{esc}.\mathsf{a} \to \mathsf{s} : \mathtt{no}\end{cases}$$

$$\mathsf{G}_{\mathtt{yes}} = \mathsf{a} \to \mathsf{s} : \mathtt{buy}$$

as shown on the left of Figure 1, where p = pri, b = buy, y = yes, and n = no.

This example is interesting for two reasons. In [10] it is argued that the typing context corresponding to $\mathbb{M}_{\mathsf{asb}}$ cannot be obtained by projecting any global type, no matter whether one uses plain or full merging (see Definition 3.3 in Figure 3 of [10]). Instead in our type system we can type $\mathbb{M}_{\mathsf{asb}}$ with the un-projectable global type $\mathsf{G}_{\mathsf{asb}}$. The other reason is that $\mathsf{G}_{\mathsf{asb}}$ is a good example of an unbounded global type [2] that is inhabited.

**Recursive Map/Reduce (n = 2).** This is the example in Figure 4(3) of [10]. We consider only the case of two workers for the sake of readability.

> The mapper (m) assigns one datum to both Worker1 ($\mathsf{w}_1$) and Worker2 ($\mathsf{w}_2$) for processing. Once completed, both $\mathsf{w}_1$ and $\mathsf{w}_2$ send their results to the reducer (r). Then r informs m whether to continue or to stop. If termination is chosen, the m also instructs $\mathsf{w}_1$ and $\mathsf{w}_2$ to stop.

This protocol is implemented by the multiparty session

$$\mathbb{M}_{\mathrm{MR}} = \mathsf{m}[\,M\,] \parallel \mathsf{r}[\,R\,] \parallel \mathsf{w}_1[\,W_1\,] \parallel \mathsf{w}_2[\,W_2\,]$$

$$\mathcal{D} = \dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{End} \vdash \mathsf{m}[\,\mathbf{0}\,] \parallel \mathsf{w}_2[\,\mathbf{0}\,] \parallel \cdots}}{\mathsf{m} \to \mathsf{w}_2 : \mathrm{S} \vdash \mathsf{m}[\,\mathsf{w}_2!\mathrm{S}\,] \parallel \mathsf{w}_2[\,W_2\,] \parallel \mathsf{w}_1[\,\mathbf{0}\,] \parallel \cdots}}{\mathsf{m} \to \mathsf{w}_1 : \mathrm{S}.\mathsf{m} \to \mathsf{w}_2 : \mathrm{S} \vdash \mathsf{m}[\,\mathsf{w}_1!\mathrm{S}.\mathsf{w}_2!\mathrm{S}\,] \parallel \mathsf{w}_1[\,W_1\,] \parallel \mathsf{r}[\,\mathbf{0}\,] \parallel \cdots}}{G'_{\mathrm{MR}} \vdash \mathsf{r}[\,\mathsf{m}!\begin{cases} \mathrm{C}.R \\ \mathrm{S} \end{cases}] \parallel \mathsf{m}[\,\mathsf{r}?\begin{cases} \mathrm{C}.M \\ \mathrm{S}.\mathsf{w}_1!\mathrm{S}.\mathsf{w}_2!\mathrm{S} \end{cases}] \parallel \mathsf{w}_2[\,W_2\,] \parallel \cdots}}{\mathsf{w}_2 \to \mathsf{r} : \mathrm{R}.G'_{\mathrm{MR}} \vdash \mathsf{r}[\,\mathsf{w}_2?\mathrm{R}.\mathsf{m}!\begin{cases} \mathrm{C}.R \\ \mathrm{S} \end{cases}] \parallel \mathsf{w}_2[\,\mathsf{r}!\mathrm{R}.W_2\,] \parallel \mathsf{w}_1[\,W_1\,] \parallel \cdots}$$

$$= \dfrac{\mathsf{w}_1 \to \mathsf{r} : \mathrm{R}.\mathsf{w}_2 \to \mathsf{r} : \mathrm{R}.G'_{\mathrm{MR}} \vdash \mathsf{r}[\,\mathsf{w}_1?\mathrm{R}.\mathsf{w}_2?\mathrm{R}.\mathsf{m}!\begin{cases} \mathrm{C}.R \\ \mathrm{S} \end{cases}] \parallel \mathsf{m}[\,\mathsf{r}?\begin{cases} \mathrm{C}.M \\ \mathrm{S}.\mathsf{w}_1!\mathrm{S}.\mathsf{w}_2!\mathrm{S} \end{cases}] \parallel \mathsf{w}_1[\,\mathsf{r}!\mathrm{R}.W_1\,] \parallel \mathsf{w}_2[\,\mathsf{r}!\mathrm{R}.W_2\,] \parallel \cdots}{\mathsf{m} \to \mathsf{w}_2 : \mathrm{D}.\mathsf{w}_1 \to \mathsf{r} : \mathrm{R}.\mathsf{w}_2 \to \mathsf{r} : \mathrm{R}.G'_{\mathrm{MR}} \vdash \mathsf{m}[\,\mathsf{w}_2!\mathrm{D}.\mathsf{r}?\begin{cases} \mathrm{C}.M \\ \mathrm{S}.\mathsf{w}_1!\mathrm{S}.\mathsf{w}_2!\mathrm{S} \end{cases}] \parallel \mathsf{w}_2[\,\mathsf{m}?\begin{cases} \mathrm{D}.\mathsf{r}!\mathrm{R}.W_2 \\ \mathrm{S} \end{cases}] \parallel \mathsf{w}_1[\,\mathsf{r}!\mathrm{R}.W_1\,] \parallel \cdots}{G_{\mathrm{MR}} \vdash \mathsf{m}[\,M\,] \parallel \mathsf{r}[\,R\,] \parallel \mathsf{w}_1[\,W_1\,] \parallel \mathsf{w}_2[\,W_2\,]}$$

Figure 2: A derivation of $\mathsf{G}_{\mathrm{MR}} \vdash \mathbb{M}_{\mathrm{MR}}$.

$$
\begin{array}{rcl}
\mathsf{G}^1_{\mathrm{MW}} & = & \mathsf{a}_1 \to \mathsf{b}_1 : \left\{ \begin{array}{l} \texttt{Datum}.\mathsf{b}_1 \to \mathsf{c}_1 : \texttt{Datum}.\mathsf{c}_1 \to \mathsf{a}_1 : \texttt{Result}.\mathsf{G}^2_{\mathrm{MW}} \\ \texttt{stop}.\mathsf{b}_1 \to \mathsf{c}_1 : \texttt{stop}.\mathsf{G}^3_{\mathrm{MW}} \end{array} \right. \\
\mathsf{G}^2_{\mathrm{MW}} & = & \mathsf{a}_2 \to \mathsf{b}_2 : \left\{ \begin{array}{l} \texttt{Datum}.\mathsf{b}_2 \to \mathsf{c}_2 : \texttt{Datum}.\mathsf{c}_2 \to \mathsf{a}_2 : \texttt{Result}.\mathsf{G}^1_{\mathrm{MW}} \\ \texttt{stop}.\mathsf{b}_2 \to \mathsf{c}_2 : \texttt{stop}.\mathsf{G}^4_{\mathrm{MW}} \end{array} \right. \\
\mathsf{G}^3_{\mathrm{MW}} & = & \mathsf{a}_2 \to \mathsf{b}_2 : \left\{ \begin{array}{l} \texttt{Datum}.\mathsf{b}_2 \to \mathsf{c}_2 : \texttt{Datum}.\mathsf{c}_2 \to \mathsf{a}_2 : \texttt{Result}.\mathsf{G}^3_{\mathrm{MW}} \\ \texttt{stop}.\mathsf{b}_2 \to \mathsf{c}_2 : \texttt{stop} \end{array} \right. \\
\mathsf{G}^4_{\mathrm{MW}} & = & \mathsf{a}_1 \to \mathsf{b}_1 : \left\{ \begin{array}{l} \texttt{Datum}.\mathsf{b}_1 \to \mathsf{c}_1 : \texttt{Datum}.\mathsf{c}_1 \to \mathsf{a}_1 : \texttt{Result}.\mathsf{G}^4_{\mathrm{MW}} \\ \texttt{stop}.\mathsf{b}_1 \to \mathsf{c}_1 : \texttt{stop} \end{array} \right.
\end{array}
$$

Figure 3: Definition of $\mathsf{G}^1_{\mathrm{MW}}$.

where

$$
M = \mathsf{w}_1!\texttt{Datum}.\mathsf{w}_2!\texttt{Datum}.\mathsf{r}? \left\{ \begin{array}{l} \texttt{continue}.M \\ \texttt{stop}.\mathsf{w}_1!\texttt{stop}.\mathsf{w}_2!\texttt{stop} \end{array} \right.
$$

$$
R = \mathsf{w}_1?\texttt{Result}.\mathsf{w}_2?\texttt{Result}.\mathsf{m}! \left\{ \begin{array}{l} \texttt{continue}.R \\ \texttt{stop} \end{array} \right.
$$

$$
W_i = \mathsf{m}? \left\{ \begin{array}{l} \texttt{Datum}.\mathsf{r}!\texttt{Result}.W_i \\ \texttt{stop} \end{array} \right. \quad \text{for } i = 1, 2.
$$

The above multiparty session can be typed by the following global type:

$$
\mathsf{G}_{\mathrm{MR}} = \mathsf{m} \to \mathsf{w}_1 : \texttt{Datum}.\mathsf{m} \to \mathsf{w}_2 : \texttt{Datum}.\mathsf{w}_1 \to \mathsf{r} : \texttt{Result}.\mathsf{w}_2 \to \mathsf{r} : \texttt{Result}.\mathsf{G}'_{\mathrm{MR}}
$$

where

$$
\mathsf{G}'_{\mathrm{MR}} = \mathsf{r} \to \mathsf{m} : \left\{ \begin{array}{l} \texttt{continue}.\mathsf{G}_{\mathrm{MR}} \\ \texttt{stop}.\mathsf{m} \to \mathsf{w}_1 : \texttt{stop}.\mathsf{m} \to \mathsf{w}_2 : \texttt{stop} \end{array} \right.
$$

as shown in Figure 2, where $\mathrm{D} = \texttt{Datum}$, $\mathrm{R} = \texttt{Result}$, $\mathrm{C} = \texttt{continue}$, and $\mathrm{S} = \texttt{stop}$. Notice that the implicit use of subtyping in the typing rule allows us to get simpler processes for the workers than the corresponding local types in [10] and processes in [3].

**Independent Multiparty Workers. (n = 2)**

This is the example in Figure 4(4) of [10]. For readability, we consider the case of two workers per class (A, B, C).

> A starter (s) participant tells to both Worker A1 ($\mathsf{a}_1$) and Worker A2 ($\mathsf{a}_2$) to independently process a Datum. The protocol then proceeds as follows:
>
> - $\mathsf{a}_1$ informs Worker B1 ($\mathsf{b}_1$) to either process the Datum or to stop. In the former case, $\mathsf{b}_1$ tells Worker C1 ($\mathsf{c}_1$) to process the Datum, after which $\mathsf{c}_1$ communicate the Result to $\mathsf{a}_1$. Upon receiving that, $\mathsf{a}_1$ can either repeat what was described above and again tell

$$
\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{D}_1}{\mathsf{c}_2 \to \mathsf{a}_2 : \mathtt{R}.\mathsf{G}^1_{\mathrm{MW}} \vdash \mathsf{a}_2[\,\mathsf{c}_2?\mathtt{R}.A_2\,] \parallel \mathsf{b}_2[\,B_2\,] \parallel \mathsf{c}_2[\,\mathsf{a}_2!\mathtt{R}.C_2\,] \parallel \cdots}}{\mathsf{b}_2 \to \mathsf{c}_2 : \mathtt{D}.\mathsf{c}_2 \to \mathsf{a}_2 : \mathtt{R}.\mathsf{G}^1_{\mathrm{MW}} \vdash \mathsf{a}_2[\,\mathsf{c}_2?\mathtt{R}.A_2\,] \parallel \mathsf{b}_2[\,\mathsf{c}_2!\mathtt{D}.B_2\,] \parallel \mathsf{c}_2[\,\mathsf{b}_2?\left\{\begin{array}{ll}\mathtt{D}.\mathsf{a}_2!\mathtt{R}.C_2 & \\ \mathtt{S} & \end{array}\right.\,] \parallel \cdots}\ \mathcal{D}_2}{\mathsf{G}^2_{\mathrm{MW}} \vdash \mathsf{a}_1[\,A_1\,] \parallel \mathsf{c}_1[\,C_1\,] \parallel \mathsf{a}_2[\,\mathsf{b}_2!\left\{\begin{array}{ll}\mathtt{D}.\mathsf{c}_2?\mathtt{R}.A_2 & \\ \mathtt{S} & \end{array}\right.\,] \parallel \mathsf{b}_2[\,\mathsf{a}_2?\left\{\begin{array}{ll}\mathtt{D}.\mathsf{c}_2!\mathtt{D}.B_2 & \\ \mathtt{S}.\mathsf{c}_2!\mathtt{S} & \end{array}\right.\,] \parallel \cdots}}{\mathsf{c}_1 \to \mathsf{a}_1 : \mathtt{R}.\mathsf{G}^2_{\mathrm{MW}} \vdash \mathsf{b}_1[\,B_1\,] \parallel \mathsf{c}_1[\,\mathsf{a}_1!\mathtt{R}.C_1\,] \parallel \mathsf{a}_1[\,\mathsf{c}_1?\mathtt{R}.C_1\,] \cdots}}{\mathsf{b}_1 \to \mathsf{c}_1 : \mathtt{D}.\mathsf{c}_1 \to \mathsf{a}_1 : \mathtt{R}.\mathsf{G}^2_{\mathrm{MW}} \vdash \mathsf{a}_1[\,\mathsf{c}_1?\mathtt{R}.A_1\,] \parallel \mathsf{b}_1[\,\mathsf{c}_1!\mathtt{D}.B_1\,] \parallel \mathsf{c}_1[\,\mathsf{b}_1?\left\{\begin{array}{ll}\mathtt{D}.\mathsf{a}_1!\mathtt{R}.C_1 & \\ \mathtt{S} & \end{array}\right.\,] \parallel \cdots}\ \mathcal{D}_3}{\mathsf{G}^1_{\mathrm{MW}} \vdash \mathsf{s}[\,\mathbf{0}\,] \parallel \mathsf{a}_1[\,\mathsf{b}_1!\left\{\begin{array}{ll}\mathtt{D}.\mathsf{c}_2?\mathtt{R}.A_1 & \\ \mathtt{S} & \end{array}\right.\,] \parallel \mathsf{b}_1[\,\mathsf{a}_1?\left\{\begin{array}{ll}\mathtt{D}.\mathsf{c}_1!\mathtt{D}.B_1 & \\ \mathtt{S}.\mathsf{c}_1!\mathtt{S} & \end{array}\right.\,] \parallel \mathsf{a}_2[\,\mathsf{b}_2!\left\{\begin{array}{ll}\mathtt{D}.\mathsf{c}_2?\mathtt{R}.A_2 & \\ \mathtt{S} & \end{array}\right.\,] \parallel \cdots
$$

Figure 4: Definition of $\mathcal{D}_1$ for deriving $\mathsf{G}_{\mathrm{MW}} \vdash \mathbb{M}_{\mathrm{MW}}$.

$\mathsf{b}_1$ to process the `Datum` or to `stop`. In the latter case, $\mathsf{b}_1$ instructs $\mathsf{c}_1$ to `stop`, too.

- Workers A2, B2, C2 ($\mathsf{a}_2$, $\mathsf{b}_2$, $\mathsf{c}_2$) follow the same sub-protocol as, respectively, $\mathsf{a}_1$, $\mathsf{b}_1$, $\mathsf{c}_1$, independently.

The above protocol is encoded by the following multiparty session:

$$
\mathbb{M}_{\mathrm{MW}} = \mathsf{s}[\,S\,] \parallel \mathsf{a}_1[\,A_1\,] \parallel \mathsf{b}_1[\,B_1\,] \parallel \mathsf{c}_1[\,C_1\,] \parallel \mathsf{a}_2[\,A_2\,] \parallel \mathsf{b}_2[\,B_2\,] \parallel \mathsf{c}_2[\,C_2\,]
$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{c}_2 \to \mathsf{a}_2 : \mathtt{R}.\mathsf{G}^3_{\mathrm{MW}} \vdash \mathsf{a}_2[\,\mathsf{c}_2?\mathtt{R}.A_2\,] \parallel \mathsf{b}_2[\,B_2\,] \parallel \mathsf{c}_2[\,\mathsf{a}_2!\mathtt{R}.C_2\,] \parallel \cdots}\,\mathcal{D}_{3,1}}{\mathsf{b}_2 \to \mathsf{c}_2 : \mathtt{D}.\mathsf{c}_2 \to \mathsf{a}_2 : \mathtt{R}.\mathsf{G}^3_{\mathrm{MW}} \vdash \mathsf{a}_2[\,\mathsf{c}_2?\mathtt{R}.A_2\,] \parallel \mathsf{b}_2[\,\mathsf{c}_2!\mathtt{D}.B_2\,] \parallel \mathsf{c}_2[\,\mathsf{b}_2?\left\{\begin{array}{l}\mathtt{D}.\mathsf{a}_2!\mathtt{R}.C_2\\ \mathtt{S}\end{array}\right.\,] \parallel \cdots}\,\mathcal{D}_{3,2}}{\mathsf{G}^3_{\mathrm{MW}} \vdash \mathsf{a}_2[\,\mathsf{b}_2!\left\{\begin{array}{l}\mathtt{D}.\mathsf{c}_2?\mathtt{R}.A_2\\ \mathtt{S}\end{array}\right.\,] \parallel \mathsf{b}_2[\,\mathsf{a}_2?\left\{\begin{array}{l}\mathtt{D}.\mathsf{c}_2!\mathtt{D}.B_2\\ \mathtt{S}.\mathsf{c}_2!\mathtt{S}\end{array}\right.\,] \parallel \cdots}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{c}_1 \to \mathsf{a}_1 : \mathtt{R}.\mathsf{G}^4_{\mathrm{MW}} \vdash \mathsf{a}_1[\,\mathsf{c}_1?\mathtt{R}.A_1\,] \parallel \mathsf{b}_1[\,B_1\,] \parallel \mathsf{c}_1[\,\mathsf{a}_1!\mathtt{R}.C_1\,] \parallel \cdots}\,\mathcal{D}_4}{\mathsf{b}_1 \to \mathsf{c}_1 : \mathtt{D}.\mathsf{c}_1 \to \mathsf{a}_1 : \mathtt{R}.\mathsf{G}^4_{\mathrm{MW}} \vdash \mathsf{a}_1[\,\mathsf{c}_1?\mathtt{R}.A_1\,] \parallel \mathsf{b}_1[\,\mathsf{c}_1!\mathtt{D}.B_1\,] \parallel \mathsf{c}_1[\,\mathsf{b}_1?\left\{\begin{array}{l}\mathtt{D}.\mathsf{a}_1!\mathtt{R}.C_1\\ \mathtt{S}\end{array}\right.\,] \parallel \cdots}\,\mathcal{D}_{4,1}}{\mathsf{G}^4_{\mathrm{MW}} \vdash \mathsf{a}_1[\,\mathsf{b}_1!\left\{\begin{array}{l}\mathtt{D}.\mathsf{c}_1?\mathtt{R}.A_1\\ \mathtt{S}\end{array}\right.\,] \parallel \mathsf{b}_1[\,\mathsf{a}_1?\left\{\begin{array}{l}\mathtt{D}.\mathsf{c}_1!\mathtt{D}.B_1\\ \mathtt{S}.\mathsf{c}_1!\mathtt{S}\end{array}\right.\,] \parallel \cdots}$$

Figure 5: Definitions of $\mathcal{D}_{3,1}$ (right) and $\mathcal{D}_4$ (left) for deriving $\mathsf{G}_{\mathrm{MW}} \vdash \mathbb{M}_{\mathrm{MW}}$.

where

$$S = \mathsf{a}_1!\mathtt{D}\mathtt{atum}.\mathsf{a}_2!\mathtt{D}\mathtt{atum}$$

$$A_i = \mathsf{s}?\mathtt{D}\mathtt{atum}.\mathsf{b}_i!\left\{\begin{array}{l}\mathtt{D}\mathtt{atum}.\mathsf{c}_i?\mathtt{R}\mathtt{esult}.A_i\\ \mathtt{S}\mathtt{top}\end{array}\right.$$

$$B_i = \mathsf{a}_i?\left\{\begin{array}{l}\mathtt{D}\mathtt{atum}.\mathsf{c}_i!\mathtt{D}\mathtt{atum}.B_i\\ \mathtt{S}\mathtt{top}.\mathsf{c}_i!\mathtt{S}\mathtt{top}\end{array}\right.$$

$$C_i = \mathsf{b}_i?\left\{\begin{array}{l}\mathtt{D}\mathtt{atum}.\mathsf{a}_i!\mathtt{R}\mathtt{esult}.C_i\\ \mathtt{S}\mathtt{top}\end{array}\right.$$

$$\mathcal{D}_2 = \dfrac{\mathcal{D}_4}{\mathsf{b}_2 \to \mathsf{c}_2 : \mathrm{s}.\mathsf{G}^4_{\mathrm{MW}} \vdash \mathsf{a}_2[\,\mathbf{0}\,] \parallel \mathsf{b}_2[\,\mathsf{c}_2!\mathrm{s}\,] \parallel \mathsf{c}_2[\,\mathsf{b}_2?\left\{\begin{array}{l}\mathrm{D}.\mathsf{a}_2!\mathrm{R}.C_2 \\ \mathrm{s}\end{array}\right.\,] \parallel \cdots}$$

$$\mathcal{D}_3 = \dfrac{\mathcal{D}_{3,1}}{\mathsf{b}_1 \to \mathsf{c}_1 : \mathrm{s}.\mathsf{G}^3_{\mathrm{MW}} \vdash \mathsf{a}_1[\,\mathbf{0}\,] \parallel \mathsf{b}_1[\,\mathsf{c}_1!\mathrm{s}\,] \parallel \mathsf{c}_1[\,\mathsf{b}_1?\left\{\begin{array}{l}\mathrm{D}.\mathsf{a}_1!\mathrm{R}.C_1 \\ \mathrm{s}\end{array}\right.\,] \parallel \cdots}$$

$$\mathcal{D}_{3,2} = \dfrac{\mathsf{End} \vdash \mathsf{s}[\,\mathbf{0}\,] \parallel \mathsf{a}_1[\,\mathbf{0}\,] \parallel \mathsf{b}_1[\,\mathbf{0}\,] \parallel \mathsf{c}_1[\,\mathbf{0}\,] \parallel \mathsf{a}_2[\,\mathbf{0}\,] \parallel \mathsf{b}_2[\,\mathbf{0}\,] \parallel \mathsf{c}_2[\,\mathbf{0}\,]}{\mathsf{b}_2 \to \mathsf{c}_2 : \mathrm{s} \vdash \mathsf{a}_2[\,\mathbf{0}\,] \parallel \mathsf{b}_2[\,\mathsf{c}_2!\mathrm{s}\,] \parallel \mathsf{c}_2[\,\mathsf{b}_2?\left\{\begin{array}{l}\mathrm{D}.\mathsf{a}_2!\mathrm{R}.C_2 \\ \mathrm{s}\end{array}\right.\,] \parallel \cdots}$$

$$\mathcal{D}_{4,1} = \dfrac{\mathsf{End} \vdash \mathsf{s}[\,\mathbf{0}\,] \parallel \mathsf{a}_1[\,\mathbf{0}\,] \parallel \mathsf{b}_1[\,\mathbf{0}\,] \parallel \mathsf{c}_1[\,\mathbf{0}\,] \parallel \mathsf{a}_1[\,\mathbf{0}\,] \parallel \mathsf{b}_1[\,\mathbf{0}\,] \parallel \mathsf{c}_1[\,\mathbf{0}\,]}{\mathsf{b}_1 \to \mathsf{c}_1 : \mathrm{s} \vdash \mathsf{a}_1[\,\mathbf{0}\,] \parallel \mathsf{b}_1[\,\mathsf{c}_1!\mathrm{s}\,] \parallel \mathsf{c}_1[\,\mathsf{b}_1?\left\{\begin{array}{l}\mathrm{D}.\mathsf{a}_1!\mathrm{R}.C_1 \\ \mathrm{s}\end{array}\right.\,] \parallel \cdots}$$

Figure 6: Definitions of $\mathcal{D}_2$, $\mathcal{D}_3$, $\mathcal{D}_{3,2}$, $\mathcal{D}_{4,1}$ for deriving $\mathsf{G}_{\mathrm{MW}} \vdash \mathbb{M}_{\mathrm{MW}}$.

for $i = 1, 2$.

The above multiparty session can be typed by the following global type:

$$\mathsf{G}_{\mathrm{MW}} = \mathsf{s} \to \mathsf{a}_1 : \mathrm{D}\mathtt{atum}.\mathsf{s} \to \mathsf{a}_2 : \mathrm{D}\mathtt{atum}.\mathsf{G}^1_{\mathrm{MW}}$$

where $\mathsf{G}^1_{\mathrm{MW}}$ is defined in Figure 3.
The typing derivation for this example is

$$\dfrac{\dfrac{\mathcal{D}_1}{\mathsf{s} \to \mathsf{a}_2 : \mathrm{D}.\mathsf{G}^1_{\mathrm{MW}} \vdash \mathsf{s}[\,\mathsf{a}_2!\mathrm{D}\,] \parallel \mathsf{a}_1[\,\mathsf{b}_1!\left\{\begin{array}{l}\mathrm{D}.\mathsf{c}_1?\mathrm{R}.A_1 \\ \mathrm{s}\end{array}\right.\,] \parallel \mathsf{a}_2[\,\mathrm{s}?\mathrm{D}.\mathsf{b}_2!\left\{\begin{array}{l}\mathrm{D}.\mathsf{c}_2?\mathrm{R}.A_2 \\ \mathrm{s}\end{array}\right.\,] \parallel \cdots}}{\mathsf{G}_{\mathrm{MW}} \vdash \mathbb{M}_{\mathrm{MW}}}$$

where $\mathcal{D}_1$ is defined in Figure 4, $\mathcal{D}_{3,1}$ and $\mathcal{D}_4$ are defined in Figure 5, the other subderivations are defined in Figure 6. In these figures we abbreviate $\mathrm{D} = \mathrm{D}\mathtt{atum}$, $\mathrm{R} = \mathrm{R}\mathtt{esult}$, and $\mathrm{s} = \mathrm{s}\mathtt{top}$.

We remark that by allowing a parallel composition operator between global types with different participants as in [6, 7] and the typing rule

$$\frac{\mathsf{G} \vdash \mathbb{M} \quad \mathsf{G}' \vdash \mathbb{M}'}{\mathsf{G} \parallel \mathsf{G}' \vdash \mathbb{M} \parallel \mathbb{M}'}$$

we could type this example with the simpler global type

$$\mathsf{s} \to \mathsf{a}_1 : \mathtt{Datum}.\mathsf{s} \to \mathsf{a}_2 : \mathtt{Datum}.(\mathsf{G}_1 \parallel \mathsf{G}_2)$$

where

$$\mathsf{G}_i = \mathsf{a}_i \to \mathsf{b}_i : \begin{cases} \mathtt{Datum}.\mathsf{b}_i \to \mathsf{c}_1 : \mathtt{Datum}.\mathsf{c}_i \to \mathsf{a}_i : \mathtt{Result}.\mathsf{G}_i \\ \mathtt{Stop}.\mathsf{b}_i \to \mathsf{c}_i : \mathtt{Stop}. \end{cases}$$

for $i = 1, 2$.

# References


[1] Franco Barbanera, Mariangiola Dezani-Ciancaglini & Ugo de'Liguoro (2022): *Open compliance in multiparty sessions*. In S. Lizeth Tapia Tarifa & José Proença, editors: *FACS*, *LNCS* 13712, Springer, pp. 222–243, doi:10.1007/978-3-031-20872-0_13.

[2] Franco Barbanera, Mariangiola Dezani-Ciancaglini & Ugo de'Liguoro (2024): *Unprojectable Global Types for Multiparty Sessions*. In Alessandro Bruni, Alberto Momigliano, Matteo Pradella, Matteo Rossi & James Cheney, editors: *PPDP*, ACM, pp. 15:1–15:13, doi:10.1145/3678232.3678245.

[3] David Castro-Perez, Francisco Ferreira & Sung-Shik Jongmans (2026): *A Synthetic Reconstruction of Multiparty Session Types*. *PACMPL* 10(POPL), pp. 1442–1470, doi:10.1145/3776692.

[4] Romain Demangeon & Kohei Honda (2011): *Full abstraction in a subtyped pi-calculus with linear types*. In J-P. Katoen & B. König, editors: *CONCUR*, *LNCS* 6901, Springer, Berlin, pp. 280–296, doi:10.1007/978-3-642-23217-6_19.

[5] Pierre-Malo Deniélou & Nobuko Yoshida (2011): *Dynamic multirole session types*. In Thomas Ball & Mooly Sagiv, editors: *POPL*, ACM Press, pp. 435–446, doi:10.1145/1926385.1926435.

[6] Kohei Honda, Nobuko Yoshida & Marco Carbone (2008): *Multiparty asynchronous session types*. In George C. Necula & Philip Wadler, editors: *POPL*, ACM Press, pp. 273–284, doi:10.1145/1328897.1328472.

[7] Kohei Honda, Nobuko Yoshida & Marco Carbone (2016): *Multiparty asynchronous session types*. *Journal of the ACM* 63(1), pp. 9:1–9:67, doi:10.1145/2827695.

[8] Ping Hou, Nobuko Yoshida & Iona Kuhn (2026): *Less is more revisited: Association with global protocols and multiparty sessions*. *Theoretical Computer Science* 1076, p. 115873, doi:10.1016/J.TCS.2026.115873.

[9] Xavier Leroy & Hervé Grall (2009): *Coinductive big-step operational semantics*. *Information and Computation* 207(2), pp. 284–304, doi:10.1016/J.IC.2007.12.004.

[10] Alceste Scalas & Nobuko Yoshida (2019): *Less is more: multiparty session types revisited*. *PACMPL* 3(POPL), pp. 30:1–30:29, doi:10.1145/3290343.