Synthetic Behavioural Typing: Sound, Regular Multiparty Sessions via Implicit Local Types

Sung-Shik Jongmans
Department of Computer Science, Open University, the Netherlands
Centrum Wiskunde & Informatica (CWI), NWO-I, the Netherlands

Francisco Ferreira
Department of Computer Science, Royal Holloway, University of London, UK

Abstract

Programming distributed systems is difficult. Multiparty session typing (MPST) is a method to automatically prove safety and liveness of protocol implementations relative to protocol specifications.

In this paper, we introduce two new techniques to significantly improve the expressiveness of the MPST method: projection is based on *implicit* local types instead of explicit; type checking is based on the *operational semantics* of implicit local types instead of on the syntax. That is, the reduction relation on implicit local types is used not only "a posteriori" to prove type soundness (as usual), but also "a priori" to define the typing rules—*synthetically*.

Classes of protocols that can now be specified/implemented/verified for the first time using the MPST method include: recursive protocols in which different roles participate in different branches; protocols in which a receiver chooses the sender of the first communication; protocols in which multiple roles synchronously choose both the sender and the receiver of a next communication, implemented as mixed input/output processes. We present the theory of the new techniques, as well as their future potential, and we demonstrate their present capabilities to effectively support regular expressions as global types (not possible before).

2012 ACM Subject Classification Theory of computation → Concurrency

Keywords and phrases behavioural types, multiparty session types, choreographies

Digital Object Identifier 10.4230/LIPIcs.ECOOP.2023.9

Funding *Sung-Shik Jongmans*: Netherlands Organisation of Scientific Research: 016.Veni.192.103

1 Introduction

Programming distributed systems is difficult. One of the challenges is to prove that the implementation of *protocols* (message passing) is safe and live relative to the specification. Safety means that "bad" communications never happen: if a communication happens in the implementation, then it is allowed to happen by the specification. Liveness means that "good" communications eventually happen. *Multiparty session typing* (MPST), proposed by Honda et al. [39, 40], is a method to automatically prove safety and liveness of protocol implementations relative to protocol specifications. Figure 1 visualises the idea:

1. First, a *global type* G specifies a protocol among roles/participants $r_1, \ldots, r_n$, while *processes* $P_1, \ldots, P_n$ implement it. A global type models the behaviour of all processes together (e.g., "first, a number from Alice to Bob; next, a boolean from Bob to Carol").
2. Next, *local types* $L_1, \ldots, L_n$ are extracted from global type G by *projecting* G onto every role r_i. Each local type models the behaviour of one process alone (e.g., for Bob, "first, he receives a number from Alice; next, he sends a boolean to Carol").
3. Last, the processes are verified by *type checking* every process P_i against its local type L_i. Well-typedness at compile-time implies safety and liveness at run-time.

The following simple example further demonstrates the MPST method.

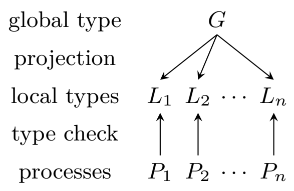
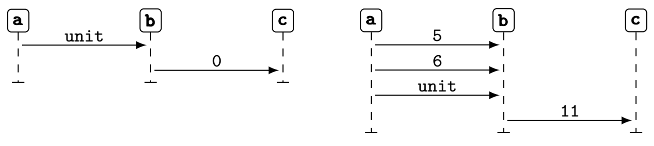


Figure 1 MPST method



(a) Sum of zero numbers **(b)** Sum of two numbers

Figure 2 Two executions of the protocol in Example 1

▶ **Example 1.** The *Summation* protocol consists of roles *Alice* ($\mathtt{a}$), *Bob* ($\mathtt{b}$), and *Carol* ($\mathtt{c}$). First, zero or more numbers are communicated from Alice to Bob. Next, a token ($\mathtt{unit}$) is communicated from Alice to Bob. Last, the sum of the numbers is communicated from Bob to Carol. Figure 2 visualises two executions of this protocol.

The following recursive global type specifies the protocol:

$$G = \mu X.\mathtt{a}{\rightarrowtriangle}\mathtt{b}{:}\begin{cases}\mathtt{Nat}.X\\ \mathtt{Unit}.\mathtt{b}{\rightarrowtriangle}\mathtt{c}{:}\mathtt{Nat}.\checkmark\end{cases}$$

Informally, global type $p{\rightarrowtriangle}q{:}\{t_i.G_i\}_{1\leq i\leq n}$ specifies the communication of a value of data type t_i from role p to role q, for some $1\leq i\leq n$; we omit braces when $n=1$.

The following recursive local types, projected from the global type, specify Alice and Bob:

$$L_{\mathtt{a}} = \mu X.\mathtt{b}{\oplus}\begin{cases}\mathtt{Nat}.X\\ \mathtt{Unit}.\checkmark\end{cases} \qquad L_{\mathtt{b}} = \mu X.\mathtt{a}\&\begin{cases}\mathtt{Nat}.X\\ \mathtt{Unit}.\mathtt{c}{\oplus}\mathtt{Nat}.\checkmark\end{cases}$$

Informally, local types $q{\oplus}\{t_i.L_i\}_{1\leq i\leq n}$ and $p\&\{t_i.L_i\}_{1\leq i\leq n}$ specify the send and the receive of a value of data type t_i from role p to role q, for some $1\leq i\leq n$; we omit braces when $n=1$.

The following processes, well-typed against the local types, implement Alice and Bob:

$$P_{\mathtt{a}} = \underbrace{\overline{\mathtt{b}}\langle 5\rangle.\overline{\mathtt{b}}\langle 6\rangle.\overline{\mathtt{b}}\langle\mathtt{unit}\rangle.\mathbf{0}}_{\text{specifically, Figure 2b}} \qquad P_{\mathtt{b}} = \mathbf{loop}(\mathtt{sum}{:}\mathtt{Nat}=0)\sum\begin{cases}\mathtt{a}(\mathtt{x}{:}\mathtt{Nat}).\mathbf{recur}(\mathtt{sum+x})\\ \mathtt{a}(\_{:}\mathtt{Unit}).\overline{\mathtt{c}}\langle\mathtt{sum}\rangle.\mathbf{0}\end{cases}$$

Informally, process $\overline{q}\langle e\rangle.P$ implements the send of the value of expression e to role q, while process $\sum\{p(x_i{:}t_i).P_i\}_{1\leq i\leq n}$ implements the receive of a value of data type t_i from role p into variable x_i, for some $1\leq i\leq n$; we omit $\sum$ and braces when $n=1$. Well-typedness means that every action implemented in $P_{\mathtt{a}}$ (resp. $P_{\mathtt{b}}$) is also specified in $L_{\mathtt{a}}$ (resp. $L_{\mathtt{b}}$). ◀

Over the past 10–15 years, substantial progress has been made both in MPST theory (e.g., extensions with advanced features, including time [10, 57], security [15–17, 24], and parametrisation [25, 33, 59]) and in MPST practice (e.g., tools for F# [58], F⋆ [71], Go [25], Java [41, 42], OCaml [70], PureScript [46], Rust [48, 49], Scala [26, 61], and TypeScript [56]).

1.1 Open Question: Regular Expressions as Global/Local Types

The expressiveness of the grammar of global/local types determines which protocols can be specified. In turn, this determines which protocols can be implemented in a provably safe and live fashion: the higher the expressiveness, the higher the applicability of the MPST method to program real(istic) distributed systems. For this reason, substantial research in the community has aimed to increase expressiveness. Doing so is not as simple as just adding new operators to the grammars; to be *effective*, these operators need to be supported by projection and type checking as well, which is actually complicated. As a result, regarding basic features, grammars of global types have effectively evolved as follows:

- In the original paper [39]:

$$G ::= p \twoheadrightarrow q{:}\{t_i . G_i\}_{1 \leq i \leq n} \mid \mu X . G \mid X \mid \checkmark$$

Thus, global types can specify that a sender chooses the data type *but not the receiver.*

- In recent papers [21–23, 54, 65]:

$$G ::= p \twoheadrightarrow \{q_i{:}t_i . G_i\}_{1 \leq i \leq n} \mid \mu X . G \mid X \mid \checkmark$$

Thus, global types can specify that a sender chooses the data type *and also the receiver.*

However, it remains an open question how to effectively generalise these *sub-regular* grammars to *regular* ones (e.g., global types that can specify that a receiver initially chooses the sender). This generalisation would enable the MPST-based verification of significantly more processes.

The notion of using regular expressions as global/local types, or *choreographies*, to specify protocols is intuitive, well-known, and actively studied. Early papers include those by Busi et al. [14], Bravetti–Zavattaro [12, 13], Lanese et al. [50], and Castagna et al. [20]; later papers include those by Guanciale–Tuosto et al. [27, 35, 64], Jongmans et al. [36, 37, 44], and De'Liguoro et al. [29]. Most of these many papers focus on projection, though, while *none* of them focus on type checking: typing rules to verify processes using regular expressions *do not yet exist* in the MPST literature. However, type checking is just as vital as projection in the MPST method (Figure 1). Thus, beyond the non-trivial achievements to *only project* regular expressions, the next elusive milestone is to *also type-check* processes against them.

In summary, the evolution of sub-regular grammars of global/local types has been hard and relatively slow; it also seems to remain relatively far from reaching an effective generalisation to regularity, despite considerable interest in the community. In contrast, for *binary* session typing, the state-of-the-art went beyond regularity already (including *mixed choice* [19]) and has started to explore *context-freeness* [2, 3, 45, 60, 63]. These observations suggest that the open question for multiparty must be significant, too, but apparently very hard to answer. In this paper, we rebuild the foundations of the MPST method using new techniques and answer the open question in the affirmative. For the first time, we effectively generalise the sub-regular grammar of global types to the following "open-ended" regular grammar:

$$G ::= p \twoheadrightarrow q{:}t \mid G_1 + G_2 \mid G_1 \cdot G_2 \mid G^* \mid \checkmark \mid \cdots$$

1.2 Contributions of This Paper

In existing papers in the MPST literature, there is a tight correspondence between the structure of global/local types and the structure of processes, instrumental to define projection and type checking. For instance, the global/local types and the processes in Example 1 have essentially the same structure: cosmetics aside, the processes are just syntactic refinements of the global/local types (choices resolved; loops unrolled; values instead of data types).

However, the usage of regular expressions as global/local types breaks the tight correspondence. Generally speaking—deliberately unspecific to regular expressions—the foundational challenge is to define projection and type checking when the grammars are *so* far apart that structurally matching processes to global/local types is prohibitively complicated. *The idea of this paper is to abandon such structural matching and use two new techniques instead:*

- **Local types and projection:** Projection is based on *implicit* local types instead of explicit. To clarify the difference, consider the following projections of global type G in Example 1:

$$L_{\text{old}} = \mu X . \mathtt{a} \& \{\mathtt{Nat} . X, \mathtt{Unit} . \mathtt{c} \oplus \mathtt{Nat} . \checkmark\} \qquad L_{\text{new}} = G \restriction \mathtt{b}$$

Explicit local type L_{old} is representative of existing techniques (same as L_{b} in Example 1): it has the same structure as G. In contrast, implicit local type L_{new} is representative of this paper's new technique; essentially, it is just a role-indexed global type.
Notably, the concept of *merging*, shown to be problematic for session types [62] (i.e., published results based on merging turned out to be defective), is no longer needed.

- **Type checking:** Type checking is based on the *operational semantics* of implicit local types instead of on the syntax. That is, the reduction relation on implicit local types is used not only "a posteriori" to prove type soundness (as usual), but also "a priori" to define the typing rules. To clarify the difference, consider the following typing rules:

$$\frac{\mathit{\Xi} \vdash e : t_k \qquad \mathit{\Xi} \vdash P : L_k}{\mathit{\Xi} \vdash \overline{q_k}\langle e\rangle.P : \sum\{q_i!t_i.L_i\}_{1\leq i\leq n}}\ [\textsc{Old}] \qquad \frac{\mathit{\Xi} \vdash e : t \qquad \mathit{\Xi} \vdash P : L' \qquad L \xrightarrow{q!t} L'}{\mathit{\Xi} \vdash \overline{q}\langle e\rangle.P : L}\ [\textsc{New}]$$

  Rule [Old] is representative of existing techniques: it states that an output process is well-typed by an explicit local type if it matches the structure. In contrast, rule [New] is representative of this paper's new technique: it states that an output process is well-typed by an implicit local type if it matches the behaviour.[1] As every local type is of the form $G\lceil r$, its reduction relation is derivable from the reduction relation of G. The applicability of rule [New] is decidable as the reduction relations constitute finite state machines.
  The programmer does not write implicit local types directly, but only global types; implicit local types are automatically extracted as role-indexed global types.

Our aim is to present the theory of the new techniques, as well as their future potential, and to demonstrate their present capabilities:

X. Protocols that could already be specified/implemented using sub-regular grammars, but not yet verified (i.e., the MPST method is sound but incomplete), *can now be verified.* This includes recursive protocols in which *different roles participate in different branches.*
Y. Protocols that could already be specified using sub-regular grammars, can now be specified *exponentially more succinct* using regular grammars.
Z. Protocols that could not yet be specified/implemented/verified using sub-regular grammars, *can now be specified/implemented/verified* using regular grammars. This includes protocols in which *a receiver chooses the sender of the first communication*, and also protocols in which *multiple roles synchronously choose both the sender and the receiver of a next communication* (implemented as mixed input/output processes, similar to `select` for Go channels and POSIX sockets).

We note that the idea of this paper *also* improves the effectiveness of sub-regular grammars (item **X**). This is because the new techniques are deliberately unspecific to regular expressions, but general: the theory readily supports *any* model of behaviour directly as a global type—be it state-based (e.g., finite automata or labelled transition systems), or event-based (e.g., pomsets or event structures), or logic-based (e.g., CTL or Hennessy–Milner logic)—so long as that model can be interpreted in our general format of operational semantics. Whether or not the usage of such models directly as global types is useful, or preferable over existing algebraic notation, is another research question. But, the future potential seems valuable.

[1] Rule [Old] is "analytic": every process/type term that occurs in the premise of a rule *must* also occur as a subterm in the conclusion. In contrast, rule [New] is "synthetic" (the dual of "analytic"; e.g., [7, 38]): every process/type term that occurs in the premise of a rule *may*—but does not have to—occur as a subterm in the conclusion. That is, meta-variable L' occurs only in the premise, but not in the conclusion, so it needs to be synthesised to prove well-typedness (by computing the reduction relation).

In §2, we further detail the contributions of this paper. In §3, we apply the new techniques to sub-regular grammars. Thus, we introduce the main concepts and complications in a familiar setting. In §4, we apply the new techniques to regular grammars. This section is surprisingly short, which is evidence of the generality of the idea: all complications are addressed in the familiar setting of sub-regular grammars in §3, and those results are almost directly applicable to regular grammars in §4. A separate technical report contains proofs [43].

2 Overview of the Techniques

In this section, using several examples, we further detail the contributions of this paper. The examples follow the three steps of the MPST method (§1), adapted to the new techniques:

1a. The programmer writes a global type G and processes $P_1, \dots, P_n$ for roles $r_1, \dots, r_n$.

1b. A tool computes the operational semantics of G and of the implicit local types $G \upharpoonright r_1, \dots, G \upharpoonright r_n$ in the form of a *termination predicate* and a *reduction relation* for every role. Every $G \upharpoonright r_i$ is an implicit local type; it does not compute an explicit one. That is, in this paper, projection is an operator for implicit local types instead of a function on global types.

2. A tool checks if every $G \upharpoonright r_i$ is *well-behaved.* If so, then G is *operationally equivalent* to $G \upharpoonright r_1, \dots, G \upharpoonright r_n$. That is, G mimics $G \upharpoonright r_1, \dots, G \upharpoonright r_n$, and vice versa. Well-behavedness of implicit local types is a new alternative to well-formedness of global types. Importantly, well-behavedness is fully *compositional*: it can be checked separately for every role.

3. A tool checks if every P_i is *well-typed* by $G \upharpoonright r_i$. If so, then $G \upharpoonright r_1, \dots, G \upharpoonright r_n$ is *operationally refined* by $P_1, \dots, P_n$. That is, $G \upharpoonright r_1, \dots, G \upharpoonright r_n$ mimics $P_1, \dots, P_n$, but not necessarily vice versa: $G \upharpoonright r_1, \dots, G \upharpoonright r_n$ may specify more behaviour than $P_1, \dots, P_n$ must implement.

2.1 Sub-Regular Grammars

In §3, we apply the new techniques of this paper to the following sub-regular grammars of global types and processes; they are representative of existing ones in the MPST literature:

$$G ::= p \twoheadrightarrow q{:}\{t_i . G_i\}_{1 \leq i \leq n} \mid \mu X . G \mid X \mid \checkmark \qquad O ::= \overbrace{\overline{q}\langle e \rangle . P}^{\text{output process}}$$
$$P ::= \sum\{O_1, \dots, O_n\} \mid \sum\{I_1, \dots, I_m\} \mid \cdots \qquad I ::= \overbrace{p(x{:}t) . P}^{\text{input process}}$$

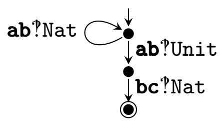
The informal meanings and notational conventions are the same as in Example 1 and further clarified in the examples in this subsection. The examples serve two purposes: to introduce the main concepts, and to demonstrate that the idea of this paper offers distinct expressive power, even in the familiar setting of sub-regular grammars (item **X** in §1.2).

▶ **Example 2.** We apply steps **1a**, **1b**, **2**, and **3** to the Summation protocol in Example 1:

1a. The following global type and processes specifies and implement the protocol (same as in Example 1, except the process for Carol, which is new here):

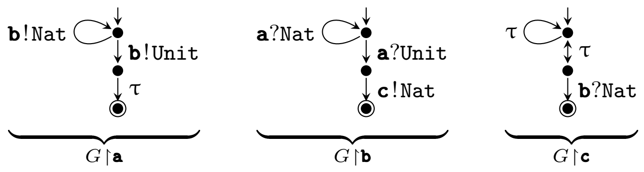
$$G = \mu X . \mathbf{a} \twoheadrightarrow \mathbf{b}{:} \begin{cases} \texttt{Nat}. X \\ \texttt{Unit}. \mathbf{b} \twoheadrightarrow \mathbf{c}{:}\texttt{Nat}. \checkmark \end{cases} \qquad \begin{aligned} P_{\mathbf{a}} &= \overline{\mathbf{b}}\langle 5 \rangle . \overline{\mathbf{b}}\langle 6 \rangle . \overline{\mathbf{b}}\langle \texttt{unit} \rangle . \mathbf{0} \\ P_{\mathbf{b}} &= \mathbf{loop}(\texttt{sum:Nat=0}) \sum \begin{cases} \mathbf{a}(\texttt{x:Nat}) . \mathbf{recur}(\texttt{sum+x}) \\ \mathbf{a}(\texttt{\_:Unit}) . \overline{\mathbf{c}}\langle \texttt{x} \rangle . \mathbf{0} \end{cases} \\ P_{\mathbf{c}} &= \mathbf{b}(\texttt{\_:Nat}) . \mathbf{0} \end{aligned}$$

1b. In the style of process algebra, we define a termination predicate and a reduction relation on global/local types to formalise their operational semantics. The following graph visualises the operational semantics of G:



Legend:
– Nodes (●) represent global/local types.
– Circled nodes (◉), if any, represent termination.
– Edges represent reduction.

Every reduction is labelled with a *global action* of the form $pq‽t$; it models a synchronous communication of a value of data type t from role p to role q. The following graphs visualise the operational semantics of $G\restriction\texttt{a}$, $G\restriction\texttt{b}$, and $G\restriction\texttt{c}$ (the *projections* of G):



Every reduction is labelled with a *local action* of the form $q!t$, $p?t$, or τ; they model a send, a receive, or "idling" (passage of time in which a role internally waits). The operational semantics of $G\restriction r$ is straightforwardly derived from the operational semantics of G, by replacing every global action $pq‽t$ with the corresponding local action: $q!t$ when $p = r \neq q$ (i.e., r is the sender), or $p?t$ when $p \neq r \neq p$ (i.e., r is the receiver), or τ when $p \neq r \neq q$ (i.e., r does not participate in the communication). We note that two τ-reductions are superimposed in the visualisation for Carol; the details do not matter yet (see §3.2).

2. To assure that a global type is operationally equivalent to the family of projections, we define a predicate that analyses the operational semantics of implicit local types, called *well-behavedness*. An implicit local type is well-behaved when:

   - **idling is neutral:** the same reductions are possible before and after a τ-reduction;
   - **sending is causal:** a !-reduction is possible initially, or after a !-reduction, or after a ?-reduction, or after a τ-reduction when it was possible already before that τ-reduction;
   - **receiving is deterministic:** multiple ?-reductions from the same source to different destinations must have different labels.

   Thus: *every send must have at least one cause; every receive must have at most one effect.* Our first main result is that if every projection is well-behaved, then the global type is operationally equivalent to the family of projections (Theorem 23). It can be checked that $G\restriction\texttt{a}$, $G\restriction\texttt{b}$, and $G\restriction\texttt{c}$ are well-behaved, so G is operationally equivalent to $\{G\restriction\texttt{a}, G\restriction\texttt{b}, G\restriction\texttt{c}\}$.

3. To assure that a family of projections is operationally refined by a family of processes, we define a typing relation that compares the syntax of processes with the operational semantics of implicit local types. Roughly, $P = \sum\{O_1, \ldots, O_n\}$ is well-typed by L when:

   - for every O_i, if $O_i = \overline{q}\langle e\rangle$, then L has a $q!t$-reduction to L' (modulo τ-reductions);
   - for every $q!t$-reduction of L (modulo τ-reductions), P has a subprocess O_i.

   Furthermore, roughly, $P = \sum\{I_1, \ldots, I_m\}$ is well-typed by L when:

   - for every I_j, if $I_j = p(x{:}t)$, then L has a $p?t$-reduction to L' (modulo τ-reductions);
   - for every $p?t$-reduction of L (modulo τ-reductions), P has a subprocess $I_j = p(x{:}t)$.

   We note that, as usual in the MPST literature, there is asymmetry between well-typedness of selections of output processes and selections of input processes (see §3.6).
   Our second main result is that if every process is well-typed by its projection, then the family of projections is operationally refined by the family of processes (Theorem 39). It can be checked that $P_{\texttt{a}}$ is well-typed by $G\restriction\texttt{a}$ (traverse the cycle in $G\restriction\texttt{a}$ twice), $P_{\texttt{b}}$ is well-typed by $G\restriction\texttt{b}$, and $P_{\texttt{c}}$ is well-typed by $G\restriction\texttt{c}$ (traverse the downwards τ-reduction in $G\restriction\texttt{c}$; this is sound), so $\{G\restriction\texttt{a}, G\restriction\texttt{b}, G\restriction\texttt{c}\}$ is operationally refined by $\{P_{\texttt{a}}, P_{\texttt{b}}, P_{\texttt{c}}\}$.

Together, operational equivalence (step **2**) and operational refinement (step **3**) imply that $\{P_{\mathtt{a}}, P_{\mathtt{b}}, P_{\mathtt{c}}\}$ is safe and live relative to G (Corollary 41). ◀

The Summation protocol is simple. However, it is not yet supported by existing techniques based on global types and projection in the MPST literature. For instance, G in Example 2 is grammatical in the state-of-the-art papers of Majumdar et al. [54] and Van Glabbeek et al. [65], but the projection onto Carol is undefined by Majumdar et al. and ill-formed due to unguarded recursion by Van Glabbeek et al.; as a result, in those papers, G cannot be used to verify processes. The following example further demonstrates the expressive power of our development.

▶ **Example 3.** The *Recursive Two Buyer* protocol [62] (extension of the *Two Buyer* protocol [39]) consists of roles *Alice* ($\mathtt{a}$), *Bob* ($\mathtt{b}$), and *Seller* ($\mathtt{s}$). Sequentially, it has three subprotocols:

- **Alice and Seller (part 1):** First, the name of an item (`String`) is communicated from Alice to Seller. Next, the price (`Nat`) is communicated from Seller to Alice.
- **Alice and Bob:** When Alice wants to negotiate with Bob to split the price, an offer (`Nat`) is communicated from her to him. Next, an acceptance (`Acc`) is communicated from Bob to Alice and the subprotocol ends, or a rejection (`Rej`) and the subprotocol loops. When Alice wants not to negotiate, a rejection of the sale is communicated from her to him.
- **Alice and Seller (part 2):** When the negotiation between Alice and Bob succeeded (resp. failed), an acceptance (resp. rejection) of the sale is communicated from Alice to Seller.

We apply steps **1a**, **1b**, **2**, and **3** to the Recursive Two Buyer protocol:

1a. The following global type specifies the protocol:

$$G = \mathbf{a}\twoheadrightarrow\mathbf{s}{:}\mathtt{String}.\mathbf{s}\twoheadrightarrow\mathbf{a}{:}\mathtt{Nat}.\mu X.\mathbf{a}\twoheadrightarrow\mathbf{b}{:}\begin{cases}\mathtt{Nat}.\mathbf{b}\twoheadrightarrow\mathbf{a}{:}\{\mathtt{Acc}.\mathbf{a}\twoheadrightarrow\mathbf{s}{:}\mathtt{Acc}.\checkmark, \mathtt{Rej}.X\}\\ \mathtt{Rej}.\mathbf{a}\twoheadrightarrow\mathbf{s}{:}\mathtt{Rej}.\checkmark\end{cases}$$

The following processes implement Alice, Bob, and Seller:

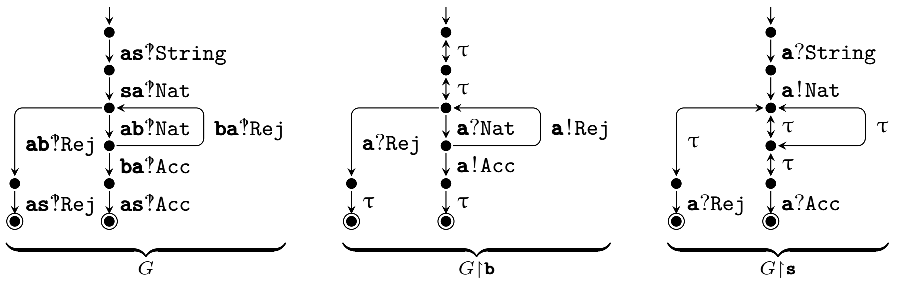
$$P_{\mathtt{a}} = \bar{\mathbf{s}}\langle\mathtt{"foo"}\rangle.\mathbf{s}(\mathtt{x}{:}\mathtt{Nat}).\bar{\mathbf{b}}\langle\mathtt{x/2}\rangle.\sum\begin{cases}\mathbf{b}(\_{:}\mathtt{Acc}).\bar{\mathbf{s}}\langle\mathtt{acc}\rangle.\mathbf{0}\\ \mathbf{b}(\_{:}\mathtt{Rej}).\bar{\mathbf{b}}\langle\mathtt{x/3}\rangle.\sum\begin{cases}\mathbf{b}(\_{:}\mathtt{Acc}).\bar{\mathbf{s}}\langle\mathtt{acc}\rangle.\mathbf{0}\\ \mathbf{b}(\_{:}\mathtt{Rej}).\bar{\mathbf{b}}\langle\mathtt{rej}\rangle.\bar{\mathbf{s}}\langle\mathtt{rej}\rangle.\mathbf{0}\end{cases}\end{cases}$$

$$P_{\mathtt{b}} = \mathbf{loop}\sum\{\mathbf{a}(\mathtt{y}{:}\mathtt{Nat}).\mathbf{if}\ \mathtt{y<=10}\ (\bar{\mathbf{a}}\langle\mathtt{acc}\rangle.\mathbf{0})\ (\bar{\mathbf{a}}\langle\mathtt{rej}\rangle.\mathbf{recur}), \mathbf{a}(\_{:}\mathtt{Rej}).\mathbf{0}\}$$

$$P_{\mathtt{s}} = \mathbf{a}(\mathtt{z}{:}\mathtt{String}).\bar{\mathbf{a}}\langle\mathtt{price(z)}\rangle.\sum\{\mathbf{a}(\_{:}\mathtt{Acc}).\mathbf{0}, \mathbf{a}(\_{:}\mathtt{Rej}).\mathbf{0}\}$$

$P_{\mathtt{a}}$ implements that Alice offers Bob to contribute half the price; when Bob rejects, Alice offers Bob to contribute a third of the price; when Bob rejects again, Alice rejects the sale. $P_{\mathtt{b}}$ implements that Bob is wiling to contribute at most ten units of currency.

1b. The following graphs visualise the operational semantics of G, $G{\upharpoonright}\mathbf{b}$, and $G{\upharpoonright}\mathbf{s}$:



2. It can be checked that $G \upharpoonright \mathbf{b}$ and $G \upharpoonright \mathbf{s}$ are well-behaved, in the same way as in Example 2. Furthermore, $G \upharpoonright \mathbf{a}$ is trivially well-behaved, as Alice participates in every communication, so the operational semantics of $G \upharpoonright \mathbf{a}$ has no τ-reductions. Thus, G is operationally equivalent to $\{G{\upharpoonright}\mathbf{a}, G{\upharpoonright}\mathbf{b}, G{\upharpoonright}\mathbf{s}\}$ (Theorem 23).
3. It can be checked that $P_{\mathbf{a}}$ is well-typed by $G{\upharpoonright}\mathbf{a}$ (by twice traversing the cycle in $G{\upharpoonright}\mathbf{a}$), $P_{\mathbf{b}}$ is well-typed by $G{\upharpoonright}\mathbf{b}$, and $P_{\mathbf{s}}$ is well-typed by $G{\upharpoonright}\mathbf{s}$. Thus, $\{G{\upharpoonright}\mathbf{a}, G{\upharpoonright}\mathbf{b}, G{\upharpoonright}\mathbf{s}\}$ is operationally refined by $\{P_{\mathbf{a}}, P_{\mathbf{b}}, P_{\mathbf{s}}\}$ (Theorem 39).

Together, operational equivalence (step **2**) and operational refinement (step **3**) imply that $\{P_{\mathbf{a}}, P_{\mathbf{b}}, P_{\mathbf{s}}\}$ is safe and live relative to G (Corollary 41). ◀

The Recursive Two Buyer protocol was introduced by Scalas–Yoshida to demonstrate the limitations of previous papers based on global types and projection [62]: existing techniques do not support recursive protocols in which *different roles participate in different branches*. The solution proposed by Scalas–Yoshida is to remove global types and projection from the MPST method altogether and, instead, manually write explicit local types for Alice, Bob, and Seller (i.e., they effectively avoid the problem instead of solving it). In contrast, using the new techniques of this paper, we can specify such recursive protocols as global types, *and* automatically extract implicit local types from them, *and* automatically verify processes.

2.2 Regular Grammars

In §4, we apply the new techniques of this paper to the following regular grammars:

$$G ::= p \rightarrowtriangle q{:}t \mid G_1 + G_2 \mid G_1 \cdot G_2 \mid G^* \mid \checkmark \qquad P ::= \textstyle\sum\{O_1, \ldots, O_n, I_1, \ldots, I_m\} \mid \cdots$$

The informal meanings are further clarified in the examples in this subsection. The examples serve two purposes: to evidence generality (i.e., no extra main concepts need to be introduced), and to demonstrate that the idea of this paper offers distinct expressive power. This power arises both in the "soft" sense (i.e., protocols that could already be specified, can now be specified exponentially more succinct; item **Y** in §1.2) and in the "hard" sense (i.e., protocols that could not yet be specified/implemented/verified, can now be; item **Z** in §1.2).

▶ **Example 4.** The *Binomial$_k$* protocol consists of roles *Alice* ($\mathbf{a}$) and *Bob* ($\mathbf{b}$). A choice between red (`Red`) and blue (`Blu`) is communicated from Alice to Bob, k times, independently. The following global types, which are equivalent, specify the protocol for $k = 3$:

$$G = \mathbf{a}\rightarrowtriangle\mathbf{b}{:}\begin{cases} \texttt{Red}.\mathbf{a}\rightarrowtriangle\mathbf{b}{:}\begin{cases} \texttt{Red}.\mathbf{a}\rightarrowtriangle\mathbf{b}{:}\begin{cases}\texttt{Red}.\checkmark\\ \texttt{Blu}.\checkmark\end{cases}\\ \texttt{Blu}.\mathbf{a}\rightarrowtriangle\mathbf{b}{:}\begin{cases}\texttt{Red}.\checkmark\\ \texttt{Blu}.\checkmark\end{cases}\end{cases}\\ \ldots\text{ (similar to the subtree above)}\end{cases} \qquad \begin{aligned} G = {} & (\mathbf{a}\rightarrowtriangle\mathbf{b}{:}\texttt{Red} + \mathbf{a}\rightarrowtriangle\mathbf{b}{:}\texttt{Blu}) \cdot \\ & (\mathbf{a}\rightarrowtriangle\mathbf{b}{:}\texttt{Red} + \mathbf{a}\rightarrowtriangle\mathbf{b}{:}\texttt{Blu}) \cdot \\ & (\mathbf{a}\rightarrowtriangle\mathbf{b}{:}\texttt{Red} + \mathbf{a}\rightarrowtriangle\mathbf{b}{:}\texttt{Blu}) \end{aligned}$$

Informally, global types $G_1 + G_2$ and $G_1 \cdot G_2$ specify choice and sequencing. ◀

The Binomial$_k$ protocol could already be specified using existing sub-regular grammars of global types in the MPST literature. However, due to the usage of a prefixing operator, the size of G_1 in Example 4 is exponential in k. In contrast, due to the usage of a sequencing operator, the size of G_2 in Example 4 is linear in k. Thus, the Binomial$_k$ protocol can now be specified exponentially more succinct. The following example demonstrates that another version of Binomial$_k$, which could not yet be specified/implemented/verified, can now be.

▶ **Example 5.** The *Role-based Binomial$_k$* protocol consists of roles *Alice* ($\mathtt{a}$) and *Bob* ($\mathtt{b}$). A unit is communicated from Alice to Bob, or from Bob to Alice, k times, independently. We apply steps **1a**, **1b**, **2**, and **3** to the Role-based Binomial$_k$ protocol:

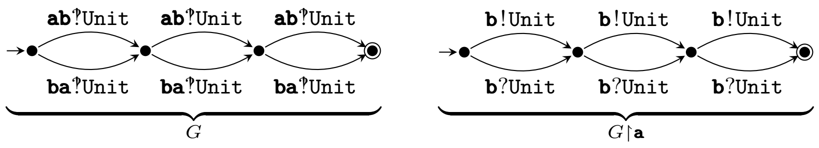
1a. The following global type specifies the protocol for $k = 3$:

$$G = (\mathtt{a}{\rightarrowtriangle}\mathtt{b}{:}\mathtt{Unit} + \mathtt{b}{\rightarrowtriangle}\mathtt{a}{:}\mathtt{Unit}) \cdot (\mathtt{a}{\rightarrowtriangle}\mathtt{b}{:}\mathtt{Unit} + \mathtt{b}{\rightarrowtriangle}\mathtt{a}{:}\mathtt{Unit}) \cdot (\mathtt{a}{\rightarrowtriangle}\mathtt{b}{:}\mathtt{Unit} + \mathtt{b}{\rightarrowtriangle}\mathtt{a}{:}\mathtt{Unit})$$

The following processes implement Alice and Bob:

$$P_{\mathtt{a}} = \sum\begin{cases} \bar{\mathtt{b}}\langle\mathtt{unit}\rangle.\sum\begin{cases} \bar{\mathtt{b}}\langle\mathtt{unit}\rangle.\sum\begin{cases}\bar{\mathtt{b}}\langle\mathtt{unit}\rangle.\mathbf{0}\\ \mathtt{b}(\_{:}\mathtt{Unit}).\mathbf{0}\end{cases}\\ \mathtt{b}(\_{:}\mathtt{Unit}).\sum\begin{cases}\bar{\mathtt{b}}\langle\mathtt{unit}\rangle.\mathbf{0}\\ \mathtt{b}(\_{:}\mathtt{Unit}).\mathbf{0}\end{cases}\end{cases}\\ \ldots \text{ (similar to the subtree above)}\end{cases} \qquad P_{\mathtt{b}} = \ldots \text{ (similar to } P_{\mathtt{a}}\text{)}$$

1b. The following graphs visualise the operational semantics of G and $G{\restriction}\mathtt{a}$:



2. It can be checked that $G \restriction \mathtt{a}$ is well-behaved, in the same way as in Example 2. Similarly, $G \restriction \mathtt{b}$ is well-behaved. Thus, G is operationally equivalent to $\{G{\restriction}\mathtt{a}, G{\restriction}\mathtt{b}\}$ (Theorem 23). We note that we can use the same definition of well-behavedness as in §2.1, whereas the grammar differs: well-behavedness is independent of structure, so directly re-applicable.

3. Process $P = \sum\{O_1, \ldots, O_n, I_1, \ldots, I_m\}$ is well-typed by L when:

- $\sum\{O_1, \ldots, O_n\}$ is well-typed by L, in the same way as in Example 2;
- $\sum\{I_1, \ldots, I_m\}$ is well-typed by L, in the same way as in Example 2.

It can be checked that $P_{\mathtt{a}}$ is well-typed by $G{\restriction}\mathtt{a}$. In particular, as $P_{\mathtt{a}}$ consists of only three unique subprocesses, no other subprocesses (duplicates) need to be type-checked when memoization is used. The three unique subprocesses are well-typed by the three non-final nodes in the visualisation of $G{\restriction}\mathtt{a}$. Similarly, $P_{\mathtt{b}}$ is well-typed by $G{\restriction}\mathtt{b}$. Thus, $\{G{\restriction}\mathtt{a}, G{\restriction}\mathtt{b}\}$ is operationally refined by $\{P_{\mathtt{a}}, P_{\mathtt{b}}\}$ (Theorem 39).

Together, operational equivalence (step **2**) and operational refinement (step **3**) imply that $\{P_{\mathtt{a}}, P_{\mathtt{b}}\}$ is safe and live relative to G (Corollary 41). ◀

The Role-based Binomial$_k$ protocol could not yet be specified/implemented/verified in previous papers in the MPST literature: existing techniques do not support protocols in which *multiple roles synchronously choose both the sender and the receiver of a next communication*. In contrast, using the new techniques of this paper, we can specify such protocols (e.g., G in Example 5), implement them as *mixed input/output processes* (e.g., $P_{\mathtt{a}}$ and $P_{\mathtt{b}}$ in Example 5), and verify. The following example further demonstrates mixed input/output, and more.

▶ **Example 6.** The *Acquire–Use–Release* protocol consists of roles *Alice* ($\mathtt{a}$), *Bob* ($\mathtt{b}$), and *Server* ($\mathtt{s}$). Concurrently, it has three subprotocols:

- **Alice and Server (AS):** First, an "acquire" message (`Acq`) is communicated from Alice to Server. Next, a "permission" message (`Perm`) is communicated from Server to Alice. Next, zero or more "usage" messages (`Use`) are communicated from Alice to Server. Last, a "release" message (`Rel`) is communicated from Alice to Server.

- **Bob and Server (BS):** Similar to **AS**.
- **Mutual Exclusion (ME):** Between sending "permission" and receiving "release", Server cannot send another "permission", thereby constraining the interleaving of **AS** and **BS**.

We apply steps **1a**, **1b**, **2**, and **3** to the Acquire–Use–Release protocol:

1a. The following global type specifies the protocol:

$$G = +\begin{cases} \mathbf{a}\rightarrowtriangle\mathbf{s}{:}\texttt{Acq}\cdot + \begin{cases} \mathbf{s}\rightarrowtriangle\mathbf{a}{:}\texttt{Perm}\cdot\mathbf{a}\rightarrowtriangle\mathbf{s}{:}\texttt{Use}^{*}\cdot + \begin{cases} \mathbf{a}\rightarrowtriangle\mathbf{s}{:}\texttt{Rel}\cdot\mathbf{b}\rightarrowtriangle\mathbf{s}{:}\texttt{Acq}\cdot G_2' \\ \mathbf{b}\rightarrowtriangle\mathbf{s}{:}\texttt{Acq}\cdot\mathbf{a}\rightarrowtriangle\mathbf{s}{:}\texttt{Use}^{*}\cdot\mathbf{a}\rightarrowtriangle\mathbf{s}{:}\texttt{Rel}\cdot G_2' \end{cases} \\ \mathbf{b}\rightarrowtriangle\mathbf{s}{:}\texttt{Acq}\cdot + \begin{cases} G_1'\cdot G_2' \\ G_2'\cdot G_1' \end{cases} \end{cases} \\ \mathbf{b}\rightarrowtriangle\mathbf{s}{:}\texttt{Acq}\cdot\ldots \text{ (similar to the subtree above)} \end{cases}$$

$$G_1' = \mathbf{s}\rightarrowtriangle\mathbf{a}{:}\texttt{Perm}\cdot\mathbf{a}\rightarrowtriangle\mathbf{s}{:}\texttt{Use}^{*}\cdot\mathbf{a}\rightarrowtriangle\mathbf{s}{:}\texttt{Rel} \qquad G_2' = \mathbf{s}\rightarrowtriangle\mathbf{b}{:}\texttt{Perm}\cdot\mathbf{b}\rightarrowtriangle\mathbf{s}{:}\texttt{Use}^{*}\cdot\mathbf{b}\rightarrowtriangle\mathbf{s}{:}\texttt{Rel}$$

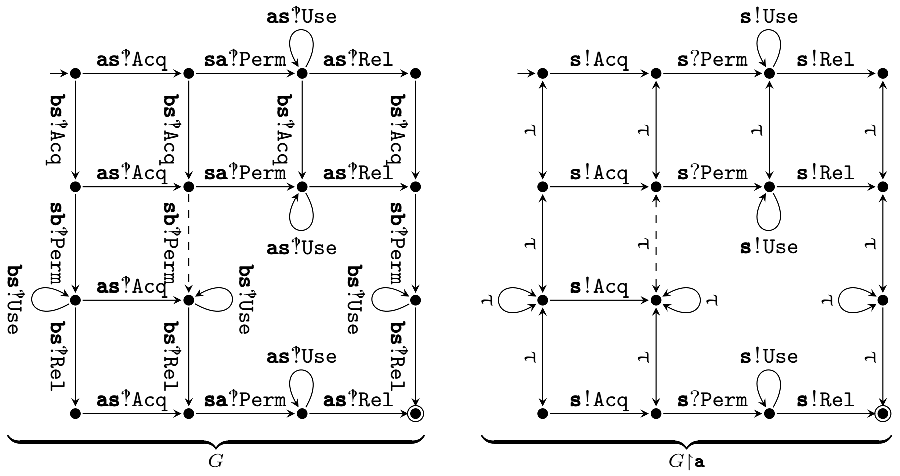
The following processes implement Alice, Bob, and Server:

$$\begin{aligned} P_{\mathbf{a}} &= \bar{\mathbf{s}}\langle\texttt{acq}\rangle.\mathbf{s}(\_{:}\texttt{Perm}).\bar{\mathbf{s}}\langle\texttt{use}\rangle.\bar{\mathbf{s}}\langle\texttt{use}\rangle.\bar{\mathbf{s}}\langle\texttt{use}\rangle.\bar{\mathbf{s}}\langle\texttt{rel}\rangle.\mathbf{0} \\ P_{\mathbf{b}} &= \bar{\mathbf{s}}\langle\texttt{acq}\rangle.\mathbf{s}(\_{:}\texttt{Perm}).\bar{\mathbf{s}}\langle\texttt{use}\rangle.\bar{\mathbf{s}}\langle\texttt{rel}\rangle.\mathbf{0} \\ P_{\mathbf{s}} &= \sum\begin{cases} \mathbf{a}(\texttt{acq1:Acq}).\sum\begin{cases} \bar{\mathbf{a}}\langle\texttt{perm}\rangle.\mathbf{loop}\sum\begin{cases} \mathbf{a}(\_{:}\texttt{Use}).\mathbf{recur} \\ \mathbf{a}(\_{:}\texttt{Rel}).\mathbf{b}(\_{:}\texttt{Acq}).(\cdots) \\ \mathbf{b}(\_{:}\texttt{Acq}).(\cdots) \end{cases} \\ \mathbf{b}(\texttt{acq2:Acq}).P_{\mathbf{s}}'' \end{cases} \\ \mathbf{b}(\_{:}\texttt{Acq}).(\cdots) \end{cases} \end{aligned}$$

$$\begin{aligned} &\text{(version 1)} \quad P_{\mathbf{s}}'' = \textstyle\sum\{\bar{\mathbf{a}}\langle\texttt{perm}\rangle.(\cdots),\bar{\mathbf{b}}\langle\texttt{perm}\rangle.(\cdots)\} \\ &\text{(version 2)} \quad P_{\mathbf{s}}'' = \textbf{if } \texttt{alice\_goes\_first(acq1,acq2)}\ (\bar{\mathbf{a}}\langle\texttt{perm}\rangle.(\cdots))\ (\bar{\mathbf{b}}\langle\texttt{perm}\rangle.(\cdots)) \\ &\text{(version 3)} \quad P_{\mathbf{s}}'' = \bar{\mathbf{a}}\langle\texttt{perm}\rangle.(\cdots) \end{aligned}$$

Version 1 of $P_{\mathbf{s}}''$ implements that, after receiving an "acquire" message from both Alice and Bob, Server chooses non-deterministically between sending a "permission" message to Alice or Bob. Versions 2 and 3 of $P_{\mathbf{s}}''$ implement that Server chooses deterministically. We note that the second choice in $P_{\mathbf{s}}$ is between a send and a receive (mixed input/output).

1b. The following graphs visualise the operational semantics of G and $G\restriction\mathbf{a}$:



The dash pattern on the vertical edges is unimportant at this point (see Example 9).

2. It can be checked that $G\restriction\mathbf{a}$ is well-behaved, in the same way as in Example 2. Similarly, $G\restriction\mathbf{b}$ is well-behaved. Furthermore, $G\restriction\mathbf{s}$ is trivially well-behaved, as Server participates in every communication, so the operational semantics of $G\restriction\mathbf{s}$ has no τ-reductions. Thus, G is operationally equivalent to $\{G\restriction\mathbf{a}, G\restriction\mathbf{b}, G\restriction\mathbf{s}\}$ (Theorem 23).

3. It can be checked that $P_{\mathbf{a}}$ is well-typed by $G\upharpoonright\mathbf{a}$, $P_{\mathbf{b}}$ is well-typed by $G\upharpoonright\mathbf{b}$, and $P_{\mathbf{s}}$ is well-typed by $G\upharpoonright\mathbf{s}$. Thus, $\{G\upharpoonright\mathbf{a}, G\upharpoonright\mathbf{b}, G\upharpoonright\mathbf{s}\}$ is operationally refined by $\{P_{\mathbf{a}}, P_{\mathbf{b}}, P_{\mathbf{s}}\}$ (Theorem 39).

Together, operational equivalence (step **2**) and operational refinement (step **3**) imply that $\{P_{\mathbf{a}}, P_{\mathbf{b}}, P_{\mathbf{s}}\}$ is safe and live relative to G (Corollary 41). ◀

The Role-based Acquire–Use–Release protocol could not yet be specified/implemented/verified in previous papers in the MPST literature: existing techniques do not support protocols in which *a receiver chooses the sender of the first communication.* In contrast, using the new techniques of this paper, we can specify such protocols (e.g., G in Example 6), implement them as processes (e.g., $P_{\mathbf{a}}$, $P_{\mathbf{b}}$, and $P_{\mathbf{s}}$ in Example 5), and verify.

In this paper, projection (including well-behavedness) and type checking are independent of the syntax of global types; they are dependent only on the operational semantics. The formulations and proofs of our main results are similarly independent. As a result of this independence, our regular grammar of global types is actually "open ended": it can be readily extended with additional global type operators (closed under regularity), intended to serve as higher-level abstractions to make the specification of protocols easier. As a first demonstration of this extensibility, we freely add the following operators:

$$G ::= \cdots \mid G_1 \,;\, G_2 \mid G_1 \parallel G_2 \mid G_1 \bowtie G_2 \mid [G]^{\gamma_1}_{\gamma_2} \mid \cdots$$

▶ **Example 7.** The following global type specifies that units are communicated first between Alice, Bob1, and Carol1, and second between Alice, Bob2, and Carol2, in that order; the communication from Bob1 to Carol1, and the communication from Bob2 to Carol2, may happen out-of-order, though.

$$\begin{aligned} G = {} & (\mathbf{a}\rightarrow\mathbf{b1}{:}\mathtt{Unit} \cdot \mathbf{b1}\rightarrow\mathbf{c1}{:}\mathtt{Unit})\,; \\ & (\mathbf{a}\rightarrow\mathbf{b2}{:}\mathtt{Unit} \cdot \mathbf{b2}\rightarrow\mathbf{c2}{:}\mathtt{Unit}) \end{aligned}$$

Informally $G_1\,;G_2$ specifies a "weak" sequence: it is similar to $G_1 \cdot G_2$, except that *independent* communications in G_1 and G_2 (when disjoint roles participate) can happen out-of-order. ◀

▶ **Example 8.** We re-apply steps **1a**, **1b**, **2**, and **3** to the Acquire–Use–Release protocol:

1a. The following global type, which is equivalent to G in Example 6, specifies the protocol:

$$G = (G_{\mathsf{AS}} \parallel G_{\mathsf{BS}}) \bowtie G_{\mathsf{ME}} \qquad \begin{aligned} G_{\mathsf{AS}} &= \mathbf{a}\rightarrow\mathbf{s}{:}\mathtt{Acq} \cdot \mathbf{s}\rightarrow\mathbf{a}{:}\mathtt{Perm} \cdot \mathbf{a}\rightarrow\mathbf{s}{:}\mathtt{Use}^* \cdot \mathbf{a}\rightarrow\mathbf{s}{:}\mathtt{Rel} \\ G_{\mathsf{BS}} &= \mathbf{b}\rightarrow\mathbf{s}{:}\mathtt{Acq} \cdot \mathbf{s}\rightarrow\mathbf{b}{:}\mathtt{Perm} \cdot \mathbf{b}\rightarrow\mathbf{s}{:}\mathtt{Use}^* \cdot \mathbf{b}\rightarrow\mathbf{s}{:}\mathtt{Rel} \\ G_{\mathsf{ME}} &= +\begin{cases} \mathbf{s}\rightarrow\mathbf{a}{:}\mathtt{Perm} \cdot \mathbf{a}\rightarrow\mathbf{s}{:}\mathtt{Rel} \cdot \mathbf{s}\rightarrow\mathbf{b}{:}\mathtt{Perm} \\ \mathbf{s}\rightarrow\mathbf{b}{:}\mathtt{Perm} \cdot \mathbf{b}\rightarrow\mathbf{s}{:}\mathtt{Rel} \cdot \mathbf{s}\rightarrow\mathbf{a}{:}\mathtt{Perm} \end{cases} \end{aligned}$$

Informally, global types $G_1 \parallel G_2$ and $G_1 \bowtie G_2$ specify interleaving and join. In general, join demands that every role complies with both of its operands. In this example, join specifies that subprotocol **ME** in Example 6 constrains the interleaving of subprotocols **AS** and **BS**. That is, the three subprotocols are modularly specified as global types G_{AS}, G_{BS}, and G_{ME}, and composed as intended using $\parallel$ and $\bowtie$; the result is an exponentially more succinct—and arguably easier to write—specification than in Example 6.
The same processes $P_{\mathbf{a}}$, $P_{\mathbf{b}}$, and $P_{\mathbf{s}}$, with any version of $P''_{\mathbf{s}}$, as in Example 6 are used.

1b. The same graphs as in Example 6 visualise the operational semantics of G and $G\upharpoonright\mathbf{a}$.

2. As in Example 6, G is operationally equivalent to $\{G\upharpoonright\mathbf{a}, G\upharpoonright\mathbf{b}, G\upharpoonright\mathbf{s}\}$.

3. As in Example 6, $\{G\upharpoonright\mathbf{a}, G\upharpoonright\mathbf{b}, G\upharpoonright\mathbf{s}\}$ is operationally refined by $\{P_{\mathbf{a}}, P_{\mathbf{b}}, P_{\mathbf{s}}\}$. ◀

Table 1 Example protocols of Van Glabbeek et al. [65] and Scalas–Yoshida [62]

name	global type
Example 9 [vG21]	$G = (\mathbf{b} \rightarrowtriangle \mathbf{se}{:}\mathtt{Talk})^* \cdot \mathbf{b} \rightarrowtriangle \mathbf{se}{:}\mathtt{Buy} \cdot \mathbf{se} \rightarrowtriangle \mathbf{sh}{:}\mathtt{Order}$
Example 13 [vG21]	$G = ((\mathbf{b1} \rightarrowtriangle \mathbf{s1}{:}\mathtt{Wait})^* \cdot \mathbf{b1} \rightarrowtriangle \mathbf{s1}{:}\mathtt{Order}) \parallel ((\mathbf{b2} \rightarrowtriangle \mathbf{s2}{:}\mathtt{Wait})^* \cdot \mathbf{b2} \rightarrowtriangle \mathbf{s2}{:}\mathtt{Order})$
Example 15 [vG21]	$G = (\mathbf{b} \rightarrowtriangle \mathbf{s1}{:}\mathtt{Order1} \cdot \mathbf{b} \rightarrowtriangle \mathbf{s2}{:}\mathtt{Wait})^* \cdot \mathbf{b} \rightarrowtriangle \mathbf{s2}{:}\mathtt{Order2} \cdot \mathbf{b} \rightarrowtriangle \mathbf{s1}{:}\mathtt{Done}$
OAuth2 [SY19]	$G = (\mathbf{s} \rightarrowtriangle \mathbf{c}{:}\mathtt{Login} \cdot \mathbf{c} \rightarrowtriangle \mathbf{a}{:}\mathtt{Passwd} \cdot \mathbf{a} \rightarrowtriangle \mathbf{s}{:}\mathtt{Auth}) + (\mathbf{s} \rightarrowtriangle \mathbf{c}{:}\mathtt{Cancel} \cdot \mathbf{c} \rightarrowtriangle \mathbf{a}{:}\mathtt{Quit})$
Rec. map/reduce [SY19]	$G = G_1 \cdot (\mathbf{r} \rightarrowtriangle \mathbf{m}{:}\mathtt{Continue} \cdot G_1)^* \cdot \mathbf{r} \rightarrowtriangle \mathbf{m}{:}\mathtt{Stop} \cdot \mathbf{m} \rightarrowtriangle \mathbf{w1}{:}\mathtt{Stop}$
	$G_1 = \mathbf{m} \rightarrowtriangle \mathbf{w1}{:}\mathtt{Datum} \cdot \mathbf{w1} \rightarrowtriangle \mathbf{r}{:}\mathtt{Result}$
MP workers [SY19]	$G = \mathbf{s} \rightarrowtriangle \mathbf{wa1}{:}\mathtt{Datum} \cdot (G_1 \parallel (\mathbf{s} \rightarrowtriangle \mathbf{wa2}{:}\mathtt{Datum} \cdot G_2))$
	$G_1 = (\mathbf{wa1} \rightarrowtriangle \mathbf{wb1} \rightarrowtriangle \mathbf{wc1}{:}\mathtt{Datum} \cdot \mathbf{wc1} \rightarrowtriangle \mathbf{wa1}{:}\mathtt{Result})^* \cdot \mathbf{wa1} \rightarrowtriangle \mathbf{wb1} \rightarrowtriangle \mathbf{wc1}{:}\mathtt{Stop}$
	$G_2 = (\mathbf{wa2} \rightarrowtriangle \mathbf{wb2} \rightarrowtriangle \mathbf{wc2}{:}\mathtt{Datum} \cdot \mathbf{wc2} \rightarrowtriangle \mathbf{wa2}{:}\mathtt{Result})^* \cdot \mathbf{wa2} \rightarrowtriangle \mathbf{wb2} \rightarrowtriangle \mathbf{wc2}{:}\mathtt{Stop}$

▶ **Example 9.** We apply steps **1a**, **1b**, **2**, and **3** to a restricted version of the Acquire–Use–Release protocol in which, after receiving an "acquire" message from both Alice and Bob, Server must send a "permission" message *first* to Alice and *second* to Bob (static order):

1a. The following global type specifies the protocol:

$$G = [(G_{\mathsf{AS}} \parallel G_{\mathsf{BS}}) \bowtie G_{\mathsf{ME}}]^{\mathbf{as}?\mathtt{Perm}}_{\mathbf{bs}?\mathtt{Perm}} \qquad G_{\mathsf{AS}}, G_{\mathsf{BS}}, G_{\mathsf{ME}} = \ldots \text{ (same as in Example 8)}$$

Informally, global type $[G]^{\gamma_1}_{\gamma_2}$ specifies the prioritisation of global action γ_1 (superscript indicates "high" priority) over global action γ_2 (subscript indicates "low" priority) in G. The same processes $P_{\mathbf{a}}$, $P_{\mathbf{b}}$, and $P_{\mathbf{s}}$, with version 3 of $P''_{\mathbf{s}}$, as in Example 6 are used.

1b. The same graphs as in Example 6 visualise the operational semantics of G and $G \upharpoonright \mathbf{a}$, but without the dashed edges.

2. As in Example 6, G is operationally equivalent to $\{G \upharpoonright \mathbf{a}, G \upharpoonright \mathbf{b}, G \upharpoonright \mathbf{s}\}$.

3. As in Example 6, $\{G \upharpoonright \mathbf{a}, G \upharpoonright \mathbf{b}, G \upharpoonright \mathbf{s}\}$ is operationally refined by $\{P_{\mathbf{a}}, P_{\mathbf{b}}, P_{\mathbf{s}}\}$. ◀

▶ Remark 10. *Merging* has historically been crucial to support sufficiently expressive kinds of choice in the MPST literature, but it is not needed in this paper. Instead, the issues that merging-based well-formedness of global types address, are covered by well-behavedness of implicit local types. Example 2 and Example 3 already demonstrated this point. To further illustrate it, Table 1 lists global types for examples of Van Glabbeek et al. [65] and Scalas–Yoshida [62]: the examples of van Glabbeek et al. require merging; the examples of Scalas–Yoshida require a more advanced concept (i.e., they use these examples to demonstrate limitations of merging). Every projection of every global type in Table 1 is well-behaved. ◀

3 Sub-Regular Grammars

In this section, we apply the new techniques for projection and type checking to sub-regular grammars of global types and processes; they are representative of existing ones in the MPST literature. Thus, we introduce the main concepts and complications in a familiar setting.

As this paper is about "processes that communicate" instead of "data that are communicated", we leave the data language largely unspecified, except for some notation:

- **Syntax:** Let $\mathbb{X}$ denote a set of *variables*, ranged over by x. Let $\mathbb{V} = \{\mathtt{true}, \mathtt{false}, \mathtt{0}, \mathtt{1}, \mathtt{2}, \ldots\}$ denote a set of *values*, ranged over by v. Let $\mathbb{E} = \mathbb{X} \cup \mathbb{V} \cup \{\mathtt{!false}, \mathtt{2+3}, \ldots\}$ denote a set of *expressions*, ranged over by e. Let $e[v/x]$ denote *substitution* of v for x in e.
- **Static semantics:** Let $\mathbb{T} = \{\mathtt{Bool}, \mathtt{Nat}, \ldots\}$ denote a set of *data types*, ranged over by t. Let $(\mathbb{X} \times \mathbb{T})^*$ denote the set of *data typing contexts* (i.e., lists of variable–type pairs), ranged over by Ξ. Let $\Xi \vdash e : t$ denote *well-typedness* of e by t in Ξ.

$$\frac{}{\checkmark\downarrow}\;[\downarrow\text{G-END}] \qquad \frac{1 \leq i \leq n}{p \rightarrowtriangle q{:}\{t_i.G_i\}_{1\leq i\leq n} \xrightarrow{pq\text{‽}t_i} G_i}\;[\rightarrow\text{G-COM}] \qquad \frac{G[\mu X.G/X] \xrightarrow{\gamma} G'}{\mu X.G \xrightarrow{\gamma} G'}\;[\rightarrow\text{G-REC}]$$

$$\frac{G\downarrow}{G\restriction r\downarrow}\;[\downarrow\text{L-AT}] \qquad \frac{G \xrightarrow{\gamma} G'}{G\restriction r \xrightarrow{\gamma\restriction r} G'\restriction r}\;[\rightarrow\text{L-AT}] \qquad pq\text{‽}t\restriction r = \begin{cases} pq!t & \text{if: } p = r \neq q \\ pq?t & \text{if: } q \neq r = q \\ \tau & \text{if: } p \neq r \neq q \end{cases} \qquad \frac{L \xrightarrow{\tau} L' \not\downarrow}{L' \xrightarrow{\tau} L}\;[\rightarrow\text{L-REV}]$$

$$\frac{L_r\downarrow \text{ for every } r}{\{L_r\}_{r\in R}\downarrow}\;[\downarrow\text{L}] \qquad \frac{L_p \xrightarrow{pq!t} L'_p \qquad L_q \xrightarrow{pq?t} L'_q \qquad L_r = L'_r \text{ for every } r \notin \{p,q\}}{\{L_r\}_{r\in R} \xrightarrow{pq\text{‽}t} \{L'_r\}_{r\in R}}\;[\rightarrow\text{L1}] \qquad \frac{L_{\bar{r}} \xrightarrow{\tau} L'_{\bar{r}} \qquad L_r = L'_r \text{ for every } r \notin \{\bar{r}\}}{\{L_r\}_{r\in R} \xrightarrow{\tau} \{L'_r\}_{r\in R}}\;[\rightarrow\text{L2}]$$

(a) Termination **(b)** Reduction. Let $G[\mu X.G/X]$ denote unfolding of X into $\mu X.G$ in G.

Figure 3 Operational semantics of sub-regular global/local types

- **Dynamic semantics:** Let eval(e) denote *evaluation* of e; it can be undefined. For instance, eval(`2+3`) = `5`, but eval(`2+true`) is undefined. Undefinedness of eval(e) is a form of "going wrong" [55]; it can give rise to deadlock (Remark 33), prevented by well-typedness (§3.7).

3.1 Global Types – Syntax

Below, we define the grammar of global/local types and abstract global/local actions.

▶ **Definition 11.** *Let $\mathbb{R}$ denote a set of roles, ranged over by p, q, r. Let $\mathbb{G}$ and $\mathbb{L}$ denote the sets of global types and (implicit) local types, ranged over by G and L; they are induced by the following grammar:*

$$G ::= p \rightarrowtriangle q{:}\{t_i.G_i\}_{1\leq i\leq n} \;\big|\; \mu X.G \;\big|\; X \;\big|\; \checkmark \qquad L ::= G\restriction r$$

Let $\mathbb{R} \rightharpoonup \mathbb{L}$ denote the set of role-indexed families of local types (partial functions), ranged over by $\mathcal{L}$. Let $\mathbb{S} = \mathbb{G} \cup \mathbb{L} \cup (\mathbb{R} \rightharpoonup \mathbb{L})$ denote the set of specifications, ranged over by S. ◀

Global type $p \rightarrowtriangle q{:}\{t_i.G_i\}_{1\leq i\leq n}$ specifies the synchronous *communication* of a value of data type t_i from role p to role q, for some $1 \leq i \leq n$. Global types $\mu X.G$ and X specify a *recursive protocol.* Global type ✓ specifies the *empty protocol.* Local type $G\restriction r$ specifies the *projection* of G onto r. Thus, projection is a local type operator instead of a function on global types: $G\restriction r$ does not compute an explicit local type; it is an implicit one. The programmer does not write implicit local types directly, but only global types.

▶ **Definition 12.** *Let $\mathbf{\Gamma} = \{(pq!t, pq?t) \mid p \neq q\}$ and $\mathbf{\Lambda} = \bigcup\{\{pq!t, pq?t\} \mid p \neq q\} \cup \{\tau\}$ denote the sets of (abstract) global actions and (abstract) local actions, ranged over by γ and λ. Let $\mathbf{A} = \mathbf{\Gamma} \cup \mathbf{\Lambda}$ denote the set of (abstract) actions, ranged over by α.* ◀

Local actions $pq!t$ and $pq?t$ model the *send* and the *receive* of a value of data type t from role p to role q; we omit p or q when it is clear from the context. Local action τ models internal *idling.* Global action $(pq!t, pq?t)$ models a communication; we often write $pq\text{‽}t$.

3.2 Global Types – Operational Semantics

Below, we define the termination predicate and reduction relation on global/local types.

▶ **Definition 13.** *Let $G\downarrow$, $L\downarrow$, and $\mathcal{L}\downarrow$ denote termination of G, L, and $\mathcal{L}$. Formally, $\downarrow$ is the predicate induced by the rules in Figure 3a, while $\not\downarrow$ is its complement (not derivable).* ◀

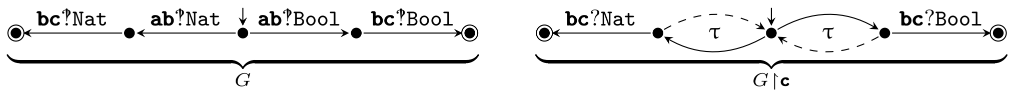
▶ **Definition 14.** *Let $G \xrightarrow{\gamma} G'$, $L \xrightarrow{\lambda} L'$, and $\mathcal{L} \xrightarrow{\lambda_p,\lambda_q} \mathcal{L}'$ denote reduction from G to G' with γ, from L to L' with λ alone, and from $\mathcal{L}$ to $\mathcal{L}'$ with λ_p and λ_q together (synchronously); we* *omit the label and/or the destination of a reduction if it does not matter. Formally, $\rightarrow$ is the relation induced by the rules in Figure 3b, while $\not\rightarrow$ is its complement (not derivable).* ◀

- Rule [$\rightarrow$G-Com] states that a communication can reduce with a global action chosen from the alternatives. Following the recent paper of Gheri et al. [34], and for the same reason as them, we omit a reduction rule for out-of-order execution of independent global actions; its interplay with recursion may give rise to infinite reduction relations (e.g., [34, Exmp. 5.1]). We recover out-of-order execution in §4, as already demonstrated in Example 7.
- Rule [$\rightarrow$G-Rec] states that a recursive protocol can reduce when its body can.
- Rule [$\rightarrow$L-At] states that a projection can reduce when the global type can.
- Rule [$\rightarrow$L-Rev] states that a τ-reduction into a non-terminated branch can be *reversed*: after "doing nothing" (the τ-reduction from L to L'), a role is always permitted to backtrack by "doing more nothing" (the reverse). This rule ensures that a role r *cannot* commit—unilaterally and irrevocably—to a future communication with another role r' by internally "doing nothing" (i.e., morally, the decision to communicate cannot be made by r alone, but only together with r', so r should not be able to make a premature commitment and get stuck). Conversely, it *can* commit to local termination by internally "doing nothing" (i.e., morally, the decision to locally terminate can be made by r alone).

  ▶ **Example 15.** The following global type specifies that either a number is communicated from Alice to Bob, and from Bob to Carol, or a boolean:

  $$G = \mathbf{a}{\rightarrow\!\triangleright}\mathbf{b}\mathtt{:}\{\mathtt{Nat}.\mathbf{b}{\rightarrow\!\triangleright}\mathbf{c}\mathtt{:Nat}.\checkmark, \mathtt{Bool}.\mathbf{b}{\rightarrow\!\triangleright}\mathbf{c}\mathtt{:Bool}.\checkmark\}$$

  The following graph visualises the operational semantics of G and $G{\upharpoonright}\mathbf{c}$:

  Dashed edges represent reductions induced by rule [$\rightarrow$L-Rev].
  Without the τ-reductions of rule [$\rightarrow$L-Rev], for instance, Carol can commit to the receive of a number by internally "doing nothing" (τ-reduction leftwards). Morally, however, this decision cannot be made by Carol alone, but only together with Bob (depending, in turn, on his previous communication with Alice). *With* the τ-reductions of rule [$\rightarrow$L-Rev], in contrast, Carol cannot commit: after the τ-reduction leftwards, there is still a sequence of τ-reductions rightwards (which Carol can freely make, because they are internal to her, unobservable to Alice and Bob) to receive a boolean. ◀

- Rules [$\rightarrow$L1] and [$\rightarrow$L2] state that a family can reduce, when two local types can reduce with a matching send/receive pair (synchronously), or when one can reduce by idling.

The following propositions state basic properties of the operational semantics.

▶ **Proposition 16** (type-level progress). *$G\downarrow$, or $G \rightarrow$ (for every G).* ◀

▶ **Proposition 17** (type-level finiteness). *$|\{G^\dagger \mid G \rightarrow \cdots \rightarrow G^\dagger\}| \in \mathbb{N}$ (for every G).* ◀

Type-level progress and finiteness, which follow straightforwardly from Figure 3, will be used to assure liveness of families of well-typed processes and decidability of type checking.

Recall that S ranges over global types, local types, and families of local types (Definition 11), and α over global actions and local actions (Definition 12):

- Let $S \Rightarrow S^\dagger$ denote τ*-reachability* from S to $S^\dagger$: either $S = S^\dagger$, or $S \xrightarrow{\tau} \cdots \xrightarrow{\tau} S^\dagger$.
- Let $S \Downarrow$ denote *weak termination* of S: $S \Rightarrow S^\dagger \downarrow$, for some $S^\dagger$.
- Let $S \overset{\alpha}{\Rightarrow} S^\P$ denote *weak reduction* from S to $S^\P$ with α: either $\alpha = \tau$ and $S \Rightarrow S^\P$, or $S \Rightarrow S^\dagger \xrightarrow{\alpha} S^\ddagger \Rightarrow S^\P$, for some $S^\dagger, S^\ddagger$.

As a notational convention, we use “/” to indicate destinations after 1 reduction, while we use “†”, “‡”, “§”, and “¶” to indicate destinations after 0-or-more reductions.

3.3 Main Result 1: Well-Behavedness Implies Operational Equivalence

Intuition The following global type specifies that a unit is communicated *first* from Alice to Bob, and *second* from Carol to Dave, in-order: $\mathbf{a}\rightarrowtriangle\mathbf{b}{:}\mathtt{Unit}.\mathbf{c}\rightarrowtriangle\mathbf{d}{:}\mathtt{Unit}.\checkmark$ (i.e., the independent actions of Alice–Bob and Carol–Dave cannot be executed out-of-order according to the operational semantics in Figure 3; we recover out-of-order execution in §4). However, this protocol is *unrealisable*: fundamentally, it cannot be implemented as a family of processes without additional covert synchronisation between Bob–Carol. This makes the global type effectively useless. Thus, we need a decision procedure to distinguish “bad” global types from “good” global types, to be able to rule out the bad ones from usage. To achieve this, we define sufficient conditions to ensure that a global type is *operationally equivalent* to the family of projections. That is, operational equivalence formalises protocol realisability.

Instead of defining the conditions on the syntax of global types in terms of *well-formedness* (as usual), we define the conditions on the operational semantics of implicit local types in terms of *well-behavedness*. If the operational semantics of every projection of a global type satisfies every condition, then operational equivalence is guaranteed. Conversely, if the operational semantics of any projection violates any condition, then the global type is ruled out. Well-behavedness is fully *compositional*: it can be checked separately for every role.

▶ Remark 18. A key advantage of well-behavedness of implicit local types over well-formedness of global types is that it allows us to prove the main results *independently of the set of global type operators*. Thus, the grammar can be extended with new global type operators (such that Propositions 16–17 continue to be valid) without reproving the theorems (§4). ◀

Before defining them formally, we informally introduce the main well-behavedness conditions:

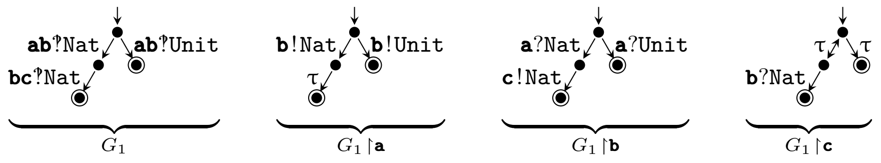
C1. Idling is neutral: A local type must always have the same weak termination/reductions before τ-reductions as after them. This means that a role can neither increase nor decrease its behavioural alternatives by idling.

C2. Sending is causal: A local type must always have the same strong !-reductions before τ-reductions as after them. This means that if a role can send after idling (later in the future), then it can also send immediately (already in the present). That is, the ability to send cannot arise out of “doing nothing”; there must be an observable cause.

C3. Receiving is deterministic: A local type must never have multiple weak ?-reductions with the same label but different destinations. This means that if a role receives, then its continuation is uniquely determined. Conditions **C2** and **C3** yield the following duality: *every send must have at least one cause; every receive must have at most one effect.*

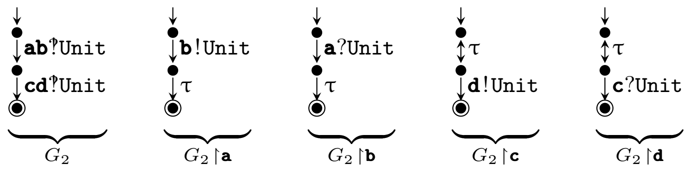
▶ **Example 19.** We illustrate the conditions with three problematic cases, each of which demonstrates a different reason for operational inequivalence. In each case: first, we define a bad global type that specifies an unrealisable protocol; next, we visualise the operational semantics of it and the projections; next, we argue that they are indeed inequivalent; last, we state the well-behavedness condition that is violated by at least one projection.

C1. If $G_1 = \mathbf{a}\rightarrowtriangle\mathbf{b}{:}\{\mathtt{Nat}.\mathbf{b}\rightarrowtriangle\mathbf{c}{:}\mathtt{Nat}.\checkmark, \mathtt{Unit}.\checkmark\}$, then:



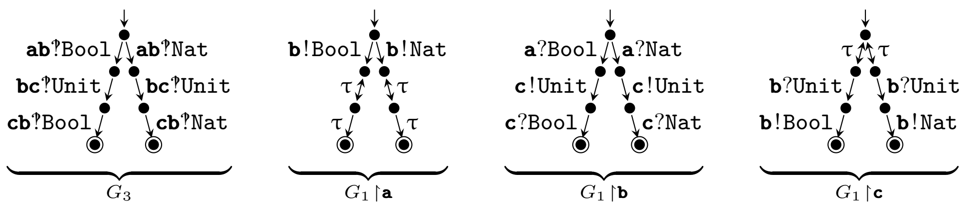
The global type cannot be stuck after weak reduction $\xRightarrow{\mathbf{ab}?\mathtt{Nat}}$: it can always reduce onwards. In contrast, the family of projections can be stuck after weak reduction $\xRightarrow{\mathbf{ab}?\mathtt{Nat}}$, namely when $G_1 \upharpoonright \mathbf{c}$ weakly reduced rightwards instead of leftwards. In that case, $G_1 \upharpoonright \mathbf{b}$ neither can terminate, nor can reduce onwards (i.e., it needs to synchronise its !-reduction with a ?-reduction of $G_1 \upharpoonright \mathbf{c}$, but $G_1 \upharpoonright \mathbf{c}$ has become unable to reciprocate). Thus, G_1 and $\{G_1 \upharpoonright r\}_{r \in \{\mathbf{a},\mathbf{b},\mathbf{c}\}}$ are inequivalent. This is caught by **C1**: $G_1 \upharpoonright \mathbf{c}$ has a weak ?-reduction before the rightwards τ-reduction, but not after it, which violates **C1** (i.e., idling is non-neutral), so G_1 is ruled out from usage.

C2. If $G_2 = \mathbf{a} \twoheadrightarrow \mathbf{b}{:}\mathtt{Unit}.\mathbf{c} \twoheadrightarrow \mathbf{d}{:}\mathtt{Unit}.\checkmark$, then:



The global type can terminate *only* after weak reductions $\xRightarrow{\mathbf{ab}?\mathtt{Unit}}\xRightarrow{\mathbf{cd}?\mathtt{Unit}}$. In contrast, the family of projections can terminate *also* after weak reductions $\xRightarrow{\mathbf{cd}?\mathtt{Unit}}\xRightarrow{\mathbf{ab}?\mathtt{Unit}}$, when $G_2 \upharpoonright \mathbf{c}$ and $G_2 \upharpoonright \mathbf{d}$ begin with $\xrightarrow{\tau}\xrightarrow{\mathbf{d}!\mathtt{Unit}}$ and $\xrightarrow{\tau}\xrightarrow{\mathbf{c}?\mathtt{Unit}}$, and when $G_2 \upharpoonright \mathbf{a}$ and $G_2 \upharpoonright \mathbf{b}$ end with $\xrightarrow{\mathbf{b}!\mathtt{Unit}}\xrightarrow{\tau}$ and $\xrightarrow{\mathbf{a}?\mathtt{Unit}}\xrightarrow{\tau}$. Thus, G_2 and $\{G_2 \upharpoonright r\}_{r \in \{\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d}\}}$ are inequivalent. This is caught by **C2**: $G_2 \upharpoonright \mathbf{c}$ does not have a !-reduction before its τ-reduction, but it does have one after it, which violates **C2** (i.e., sending is non-causal), so G_2 is ruled out from usage. We note that if we allowed out-of-order execution of independent global actions, then G_2 would satisfy **C2**. We recover out-of-order execution in §4. The corresponding global type will be $\mathbf{a} \twoheadrightarrow \mathbf{b}{:}\mathtt{Unit}\,;\,\mathbf{c} \twoheadrightarrow \mathbf{d}{:}\mathtt{Unit}$, each of whose projections will be well-behaved.

C3. If $G_3 = \mathbf{a} \twoheadrightarrow \mathbf{b}{:}\{\mathtt{Bool}.\mathbf{b} \twoheadrightarrow \mathbf{c}{:}\mathtt{Unit}.\mathbf{c} \twoheadrightarrow \mathbf{b}{:}\mathtt{Bool}.\checkmark, \mathtt{Nat}.\mathbf{b} \twoheadrightarrow \mathbf{c}{:}\mathtt{Unit}.\mathbf{c} \twoheadrightarrow \mathbf{b}{:}\mathtt{Nat}.\checkmark\}$, then:



The global type cannot be stuck after weak reductions $\xRightarrow{\mathbf{ab}?\mathtt{Bool}}\xRightarrow{\mathbf{bc}?\mathtt{Unit}}$: it can always reduce onwards. In contrast, the family of projections can be stuck after weak reductions $\xRightarrow{\mathbf{ab}?\mathtt{Bool}}\xRightarrow{\mathbf{bc}?\mathtt{Unit}}$, namely when $G_1 \upharpoonright \mathbf{c}$ weakly reduced rightwards instead of leftwards. In that case, $G_3 \upharpoonright \mathbf{b}$ neither can terminate, nor can reduce onwards (i.e., it needs to synchronise its $\mathbf{c}?\mathtt{Bool}$-reduction with a $\mathbf{b}!\mathtt{Bool}$-reduction of $G_3 \upharpoonright \mathbf{c}$, but $G_3 \upharpoonright \mathbf{c}$ has become unable to reciprocate). Thus, G_3 and $\{G_3 \upharpoonright r\}_{r \in \{\mathbf{a},\mathbf{b},\mathbf{c}\}}$ are inequivalent. This is caught by condition **C3**: $G_3 \upharpoonright \mathbf{c}$ has two weak ?-reductions with the same label, but to different destinations, which violates **C3** (i.e., receiving is non-deterministic), so G_3 is ruled out from usage. ◀

We relate the well-behavedness conditions on implicit local types in this paper to well-formedness conditions on global types in the MPST literature in the terminology of Castagna et al. [20]. Condition **C1** is usually enforced through projection (i.e., projection determinises explicit local types as they are computed, without using τ-based operators). Condition **C2** is the *sequentiality* principle of Castagna et al.; it is usually enforced by allowing out-of-order execution of independent global actions. Condition **C3** is the *knowledge for choice* principle of Castagna et al. (i.e., a receiver must always be able to uniquely determine which branch the sender was in); it is usually enforced through merging. We note that well-behavedness is relatively permissive regarding branching (some non-directed and non-located choice patterns are allowed), whereas well-formedness is relatively restrictive (all such patterns are forbidden).

Technicalities First, we define operational equivalence as a relation $\approx$ on specifications (global types, local types, and families of local types). We derive the following requirements from Example 19: **(a)** $\approx$ must be insensitive to idling (i.e., we argued in terms of weak reductions); **(b)** $\approx$ must be sensitive to deadlock (i.e., we distinguished between termination and "being stuck"). Out of many candidates [66, 67], we adopt *weak bisimilarity* (e.g., [68]): it meets both requirements **a** and **b**; additionally, it is sensitive to branching, which is not a requirement, but which makes our proofs easier. Intuitively, two specifications are weak bisimilar when they can mimick each other's termination/reductions modulo τ-reductions.

▶ **Definition 20.** *Recall that $\mathbb{S}$ denotes the set of all global types, local types, and families of local types, ranged over by S. A weak bisimulation $\heartsuit \subseteq \mathbb{S} \times \mathbb{S}$ is a relation that satisfies the following conditions, for every $(S_1, S_2) \in \heartsuit$, (and for every $S_1^¶, S_2^¶, \alpha$):*

- *If $S_1 \Downarrow$, then $S_2 \Downarrow$.*
- *If $S_2 \Downarrow$, then $S_1 \Downarrow$.*
- *If $S_1 \overset{\alpha}{\Rrightarrow} S_1^¶$, then $S_1^¶ \heartsuit S_2^¶$ and $S_2 \overset{\alpha}{\Rrightarrow} S_2^¶$, for some $S_2^¶$.*
- *If $S_2 \overset{\alpha}{\Rrightarrow} S_2^¶$, then $S_1^¶ \heartsuit S_2^¶$ and $S_1 \overset{\alpha}{\Rrightarrow} S_1^¶$, for some $S_1^¶$.*

Let $S_1 \approx S_2$ denote weak bisimilarity. Formally, $\approx$ is the largest weak bisimulation. ◀

Next, we define well-behavedness by formalising the main conditions (plus two more).

▶ **Definition 21.** *Let $\mathsf{wb}(L)$ denote well-behavedness of L. Formally, it is the largest predicate that satisfies the following conditions, for every $L \in \mathsf{wb}$ (and for every $L', L^\dagger, L_1^\ddagger, L_2^\ddagger, p, q, t$):*

C1. *If $L \Rightarrow L^\dagger$, then $L \approx L^\dagger$.* *[If $L^\dagger$ is τ-reachable, then L and $L^\dagger$ are weak bisimilar.]*
C2. *If $L \overset{pq!t}{\Longrightarrow} L^\dagger$, then $L \xrightarrow{pq!t}\approx L^\dagger$.* *[If L has a weak $!$-reduction to $L^\dagger$, then it has the same strong $!$-reduction to a weak bisimilar destination.]*
C3. *If $L \overset{pq?t}{\Longrightarrow} L_1^\ddagger$ and $L \overset{pq?t}{\Longrightarrow} L_2^\ddagger$, then $L_1^\ddagger \approx L_2^\ddagger$.* *[If L has the same weak $?$-reductions to $L_1^\ddagger$ and $L_2^\ddagger$, then $L_1^\ddagger$ and $L_2^\ddagger$ are weak bisimilar.]*
C4. *If $L \rightarrow$, then $L \not\Downarrow$.* *[If L can reduce, then it cannot terminate.]*
C5. *If $L \rightarrow L'$, then $\mathsf{wb}(L')$.* *[Reduction preserves well-behavedness.]* ◀

Last, we prove that the conditions of well-behavedness are sufficient to ensure operational equivalence. The idea is to define a *correspondence relation* between global types and families of well-behaved local types. We can then show that correspondence is a weak bisimulation.

▶ **Definition 22.** *Let $G \leftrightharpoons \{L_r\}_{r\in R}$ denote correspondence of G and $\{L_r\}_{r\in R}$. Formally:*

$$\frac{\left[\mathsf{wb}(G\restriction r) \text{ and } \mathsf{wb}(L_r) \text{ and } G\restriction r \approx L_r\right] \quad \text{for every } r \in R}{G \leftrightharpoons \{L_r\}_{r\in R}}$$ ◀

▶ **Theorem 23** (equivalence). *If $\mathsf{wb}(G\restriction r)$, for every $r \in R$, then $G \approx \{G\restriction r\}_{r\in R}$.* ◀

The proof of this main result is based on two auxiliary lemmas. They state that well-behavedness implies correspondence, and that correspondence implies weak bisimilarity.

▶ **Lemma 24.** *If $\mathsf{wb}(G\restriction r)$, for every $r \in R$, then $G \leftrightharpoons \{G\restriction r\}_{r\in R}$.* ◀

▶ **Lemma 25.** *If $G \leftrightharpoons \{L_r\}_{r\in R}$, then $G \approx \{L_r\}_{r\in R}$.* ◀

The first lemma follows directly from the definition of correspondence and the reflexivity of weak bisimilarity. The proof of the second lemma relies on the definition of well-behavedness.

▶ Remark 26. Theorem 23 depends on premise $\mathsf{wb}(G\restriction r)$. To see that checking this premise is decidable, observe that the reduction relation of G is finite by Proposition 17. As the reduction relation of $G\restriction r$ has exactly the same structure by rules [↓L-At] and [→L-At], and at most linearly many extra τ-transition by rule [→L-Rev], it is finite as well. Consequently, checking well-behavedness (including weak bisimilarity [1]) of $G\restriction r$ is trivially decidable. ◀

▶ Remark 27. As an alternative to Theorem 23, of course, it is also possible to check weak bisimilarity between G and $\{G\restriction r\}_{r\in R}$ directly. However, this would require one to compute the reduction relation of $\{G\restriction r\}_{r\in R}$, which is exponentially large in the worst case. In contrast, as well-behavedness is fully compositional, such a computation is avoided. Thus, direct weak bisimilarity is of exponential complexity (in the size of the reduction relations), whereas well-behavedness is of linear complexity and, as a result, better scalable to many roles. ◀

3.4 Processes – Syntax

Below, we define the grammar of processes and concrete local actions.

▶ **Definition 28.** *Let $\mathbb{O}$, $\mathbb{I}$, and $\mathbb{P}$ denote the sets of output processes, input processes, and processes, ranged over by O, I, and P; they are induced by the following grammar:*

$$P ::= \sum\{O_1,\dots,O_n\} \mid \sum\{I_1,\dots,I_m\} \mid \quad O ::= \overline{q}\langle e\rangle.P \quad I ::= p(x{:}t).P$$
$$\textbf{if}\ e\ P_1\ P_2 \mid \textbf{loop}\ P \mid \textbf{recur} \mid \mathbf{0}$$

Let $\mathbb{R} \rightharpoonup \mathbb{P}$ denote the set of role-indexed families of processes, ranged over by $\mathcal{P}$. ◀

Output process $\overline{pq}\langle e\rangle.P$ implements the *send* of the value of expression e from role p to role q; we omit p when it is clear from the context. Input process $pq(x{:}t).P$ implements the *receive* of a value of data type t into variable x from role p to role q; we omit q when it is clear from the context; we omit "$:t$" when the data type does not matter. Processes $\sum\{O_1,\dots,O_n\}$ and $\sum\{I_1,\dots,I_m\}$ implement non-deterministic *selections* of n output processes (sends) and m input processes (receives); we omit "$\sum$" and braces when $n=m=1$. Process **if** $e\ P_1\ P_2$ implements a *conditional choice.* Processes **loop** P and **recur** implement a *loop.* Process $\mathbf{0}$ implements the *empty process.* We note that data parameters can be added to loops in the standard way (e.g., [62]). Process creation and session creation are orthogonal to the contributions of this paper and thus we omit them.

▶ Remark 29. We stipulate that every process is *guarded* (i.e., **recur** occurs only inside $\sum$-processes) and *closed* (i.e., **recur** occurs only inside **loop**-processes), while every family $\{P_r\}_{r\in R}$ is *well-formed* (i.e., for every $r \in R$, every output process that occurs in P_r is of the form $\overline{rq}\langle e\rangle.P'$, while every input process is of the form $pr(x{:}t).P'$). ◀

▶ **Definition 30.** *Let $\mathbf{\Pi} = \bigcup\{\{pq!v, pq?v\} \mid p \neq q\}$ denote the set of (concrete) local actions, ranged over by π.* ◀

3.5 Processes – Operational Semantics

Below, we define the termination predicate and reduction relation on processes.

▶ **Definition 31.** *Let $P\downarrow$ and $\mathcal{P}\downarrow$ denote termination of P and $\mathcal{P}$. Formally, $\downarrow$ is the predicate induced by the rules in Figure 4a.* ◀

▶ **Definition 32.** *Let $P \xrightarrow{\pi} P'$ and $\mathcal{P} \xrightarrow{\pi_p,\pi_q} \mathcal{P}'$ denote reduction from P to P' with π alone, and from $\mathcal{P}$ to $\mathcal{P}'$ with π_p and π_q together (synchronously). Formally, $\rightarrow$ is the relation induced by the rules in Figure 4b.* ◀

$$\frac{}{\mathbf{0}\downarrow}\,[\downarrow\text{P-E}\textsc{nd}]$$

$$\frac{P_{\mathsf{eval}(e)}\downarrow}{\mathbf{if}\ e\ P_{\mathtt{true}}\ P_{\mathtt{false}}\downarrow}\,[\downarrow\text{P-I}\textsc{f}]$$

$$\frac{P[\mathbf{loop}\ P/\mathbf{recur}]\downarrow}{\mathbf{loop}\ P\downarrow}\,[\downarrow\text{P-L}\textsc{oop}]$$

$$\frac{P_r\downarrow\ \text{for every}\ r}{\{P_r\}_{r\in R}\downarrow}\,[\downarrow\text{P}]$$

(a) Termination

$$\frac{\overline{pq}\langle e\rangle.P\in\{O_1,\ldots,O_n\}}{\sum\{O_1,\ldots,O_n\}\xrightarrow{pq\,!\,\mathsf{eval}(e)}P}\,[\rightarrow\text{P-S}\textsc{um}1]$$

$$\frac{pq(x{:}t).P\in\{I_1,\ldots,I_m\}\qquad\vdash v:t}{\sum\{I_1,\ldots,I_m\}\xrightarrow{pq?v}P[v/x]}\,[\rightarrow\text{P-S}\textsc{um}2]$$

$$\frac{P_{\mathsf{eval}(e)}\xrightarrow{\pi}P'}{\mathbf{if}\ e\ P_{\mathtt{true}}\ P_{\mathtt{false}}\xrightarrow{\pi}P'}\,[\rightarrow\text{P-I}\textsc{f}]\qquad\frac{P[\mathbf{loop}\ P/\mathbf{recur}]\xrightarrow{\pi}P'}{\mathbf{loop}\ P\xrightarrow{\pi}P'}\,[\rightarrow\text{P-L}\textsc{oop}]$$

$$\frac{P_p\xrightarrow{pq!v}P'_p\qquad P_q\xrightarrow{pq?v}P'_q\qquad P_r=P'_r\ \text{for every}\ r\notin\{p,q\}}{\{P_r\}_{r\in R}\xrightarrow{pq\overset{!}{?}v}\{P'_r\}_{r\in R}}\,[\rightarrow\text{P}]$$

(b) Reduction

Figure 4 Operational semantics of sub-regular processes. Let $P[v/x]$ denote capture-avoiding substitution of v for x in P. Let $P[\mathbf{loop}\ P/\mathbf{recur}]$ denote unfolding of **recur** into **loop** P in P.

Rule [$\rightarrow$P-Sum1] (resp. [$\rightarrow$P-Sum2]) states that a selection can reduce with a send (resp. receive) when there is a corresponding output process (resp. input process) among the alternatives and $\mathsf{eval}(e)$ is defined (resp. v is well-typed by t and bound to x). Rule [$\rightarrow$P-If] states that a conditional choice can reduce when $\mathsf{eval}(e)\in\{\mathtt{true},\mathtt{false}\}$ and the corresponding branch can reduce. Rule [$\rightarrow$P-Loop] states that a recursive loop can reduce when its body can. Rule [$\rightarrow$P] states that a family can reduce when two processes can reduce with a matching send/receive pair (synchronously).

▶ Remark 33. Figure 4 contains no rules for communication errors: "going wrong" manifests as *deadlock*. There are three situations in which this can happen for a process P or family $\mathcal{P}$:

- If $P=\mathbf{if}\,e\,P_1\,P_2$, but $\mathsf{eval}(e)\notin\{\mathtt{true},\mathtt{false}\}$, then rules [$\downarrow$P-If]/[$\rightarrow$P-If] are inapplicable.
- If $P=\sum\{\overline{p_1q_1}\langle e_1\rangle.P_1,\ldots,\overline{p_nq_n}\langle e_n\rangle.P_n\}$ and $n>1$, but $\mathsf{eval}(e_i)$ is undefined for every $1\le i\le n$, then rule [$\rightarrow$P-Sum1] is inapplicable.
- If not all processes in $\mathcal{P}$ can terminate, while no two processes in $\mathcal{P}$ can reduce with a matching send and receive, then rules [$\downarrow$P]/[$\rightarrow$P] are inapplicable.

In each situation, P or $\mathcal{P}$ cannot terminate/reduce. Well-typedness will prevent this. ◀

3.6 Main Result 2: Well-Typedness Implies Operational Refinement

Now comes the pivotal concept among our contributions: the typing rules are based on the operational semantics of implicit local types instead of on their syntax. That is, the termination predicate and the reduction relation on local types are used not only "a posteriori" to prove type soundness (as usual), but also "a priori" to define the typing rules. This allows us to break the historically tight correspondence between the structure of global/local types and the structure of processes (§1.2).

▶ **Definition 34.** *Recall that $(\mathbb{X}\times\mathbb{T})^*$ denotes the set of data typing contexts, ranged over by Ξ. Let $(\mathbf{recur}\times\mathbb{L})^*$ denote the set of process typing contexts, ranged over by Υ. Let $\Xi,\Upsilon\vdash O:L$, $\Xi,\Upsilon\vdash I:L$, $\Xi,\Upsilon\vdash P:L$ and $\vdash\mathcal{P}:\mathcal{L}$ denote well-typedness of O, I, P by L in Ξ,Υ, and of $\mathcal{P}$ by $\mathcal{L}$. Formally, $\vdash$ is the relation induced by the rules in Figure 5.* ◀

Rule [$\vdash$-End] states that the empty process is well-typed when the local type can weakly terminate. Rule [$\vdash$-If] states that a conditional choice is well-typed when the condition and the branches are well-typed. Rules [$\vdash$-Loop]/[$\vdash$-Recur] state that a loop is well-typed when

$$\frac{\mathit{\Xi} \vdash e : t \qquad \mathit{\Xi}, \mathit{\Upsilon} \vdash P : L^{¶} \qquad L \xRightarrow{pq!t} L^{¶}}{\mathit{\Xi}, \mathit{\Upsilon} \vdash \overline{pq}\langle e \rangle.P : L}\ [\vdash\text{-Out}] \qquad \frac{\mathit{\Xi}, x : t, \mathit{\Upsilon} \vdash P : L^{¶} \qquad L \xRightarrow{pq?t} L^{¶}}{\mathit{\Xi}, \mathit{\Upsilon} \vdash pq(x{:}t).P : L}\ [\vdash\text{-In}]$$

$$\frac{\mathit{\Xi}, \mathit{\Upsilon} \vdash O_i : L \text{ for every } 1 \le i \le n \qquad [O_i = \overline{p\_}\langle \_ \rangle.\_ \text{ for some } 1 \le i \le n] \text{ for every } L \xRightarrow{pq!t}}{\mathit{\Xi}, \mathit{\Upsilon} \vdash \sum\{O_1, \ldots, O_n\} : L}\ [\vdash\text{-Sum1}]$$

$$\frac{\mathit{\Xi}, \mathit{\Upsilon} \vdash I_j : L \text{ for every } 1 \le j \le m \qquad [I_j = pq(\_{:}t).\_ \text{ for some } 1 \le j \le m] \text{ for every } L \xRightarrow{pq?t}}{\mathit{\Xi}, \mathit{\Upsilon} \vdash \sum\{I_1, \ldots, I_m\} : L}\ [\vdash\text{-Sum2}]$$

$$\frac{L \Downarrow}{\mathit{\Xi}, \mathit{\Upsilon} \vdash \mathbf{0} : L}\ [\vdash\text{-End}] \qquad \frac{\mathit{\Xi}, \mathit{\Upsilon}, \mathbf{recur} : L \vdash P : L}{\mathit{\Xi}, \mathit{\Upsilon} \vdash \mathbf{loop}\ P : L}\ [\vdash\text{-Loop}] \qquad \frac{L_1 \approx L_2}{\mathit{\Xi}, \mathit{\Upsilon}, \mathbf{recur} : L_1 \vdash \mathbf{recur} : L_2}\ [\vdash\text{-Recur}]$$

$$\frac{\mathit{\Xi} \vdash e : \texttt{Bool} \qquad \mathit{\Xi}, \mathit{\Upsilon} \vdash P_1 : L \qquad \mathit{\Xi}, \mathit{\Upsilon} \vdash P_2 : L}{\mathit{\Xi}, \mathit{\Upsilon} \vdash \mathbf{if}\ e\ P_1\ P_2 : L}\ [\vdash\text{-If}] \qquad \frac{[\vdash P_r : L_r \text{ and } \mathsf{wb}(L_r)] \text{ for every } r}{\vdash \{P_r\}_{r \in R} : \{L_r\}_{r \in R}}\ [\vdash]$$

Figure 5 Well-typedness ("_" is a meta-variable to indicate that the object does not matter)

the body and the recursive calls are well-typed. Rule [⊢] states that a family of processes is well-typed when every process is well-typed by a well-behaved local type.

Rule [⊢-Out] states that an output process is well-typed when the local type has an *analogous* weak !-reduction such that the expression and the continuation are well-typed; "analogous" means "same sender, same receiver, same data type". Rule [⊢-In] states that an input process is well-typed when the local type has an analogous weak ?-transition, and the continuation is well-typed. Rule [⊢-Sum1] states that a selection of output processes is well-typed when every subprocess is well-typed, and there is a *possibly non-analogous* subprocess for every weak !-reduction of the local type. Rule [⊢-Sum2] states that a selection of input processes is well-typed when every subprocess is well-typed, and there is an *analogous* subprocess for every weak ?-reduction of the local type.

▶ Remark 35. As usual in the MPST literature, there is asymmetry between well-typedness of selections of output processes and selections of input processes: if the local type specifies $\geq$1 sends, then the process *may* implement *one* of them (i.e., the programmer statically chooses what/whereto the process sends); if it specifies $\geq$1 receives, then it *must* implement *all* of them (i.e., the environment dynamically chooses what/wherefrom the process receives). ◀

▶ **Example 36.** Let $G = \mathbf{a}{\twoheadrightarrow}\mathbf{b}{:}\{\texttt{Nat}.\mathbf{b}{\twoheadrightarrow}\mathbf{c}{:}\texttt{Nat}.\checkmark, \texttt{Bool}.\mathbf{b}{\twoheadrightarrow}\mathbf{c}{:}\texttt{Bool}.\checkmark\}$; the operational semantics of this global type was previously visualised in Example 15.

- **Alice:** Process $\overline{\mathbf{b}}\langle\texttt{5}\rangle.\mathbf{0}$, process $\overline{\mathbf{b}}\langle\texttt{true}\rangle.\mathbf{0}$, and process $\mathbf{if}\ \texttt{cond()}\ (\overline{\mathbf{b}}\langle\texttt{5}\rangle.\mathbf{0})\ (\overline{\mathbf{b}}\langle\texttt{true}\rangle.\mathbf{0})$ are all well-typed by $G\restriction\mathbf{a}$, because rule [⊢-Sum1] requires only one send specified to be implemented. Process $\overline{\mathbf{b}}\langle\texttt{"foo"}\rangle.\mathbf{0}$ is ill-typed, because rule [⊢-Sum1] requires every send implemented to be specified. Process $\mathbf{loop}\ (\overline{\mathbf{b}}\langle\texttt{5}\rangle.\mathbf{recur})$ is ill-typed by $G\restriction\mathbf{a}$ as well, because rule [⊢-Loop] adds $\mathbf{recur} : G\restriction\mathbf{a}$ to the process typing context at the root of the derivation tree, but rule [⊢-Recur] requires $\mathbf{recur} : \checkmark$ at the leaf, and $G\restriction\mathbf{a} \not\approx \checkmark$.
- **Carol:** Process $\mathbf{b}(\texttt{x:Nat}).\mathbf{0}$ and process $\mathbf{if}\ \texttt{cond()}\ (\mathbf{b}(\texttt{x:Nat}).\mathbf{0})\ (\mathbf{b}(\texttt{x:Bool}).\mathbf{0})$ are both ill-typed by $G\restriction\mathbf{c}$, because rule [⊢-Sum2] requires every receive specified to be implemented. Process $\sum\{\mathbf{b}(\texttt{x:Nat}).\mathbf{0}, \mathbf{b}(\texttt{x:Bool}).\mathbf{0}\}$ is well-typed by $G\restriction\mathbf{c}$. ◀

The asymmetry between the right-sided premises of rules [⊢-Sum1]/[⊢-Sum2] ensure that if a family of well-typed local types can reduce, then the family of well-typed processes can reduce, too, *but possibly with a non-analogous communication.* This is *progress*:

▶ **Lemma 37.** *If $G \leftrightarrows \mathcal{L}$ and $\vdash \mathcal{P} : \mathcal{L}$, then $\mathcal{P}\downarrow$ or $\mathcal{P} \rightarrow$.* ◀

Complementary, the symmetry between the left-sided premises of rules [$\vdash$-Sum1]/[$\vdash$-Sum2] (i.e., every subprocess of every selection needs to be well-typed) ensure that if a family of well-typed processes can reduce, then the family of well-behaved local types can reduce, too, *and necessarily with an analogous communication.* This is *preservation*:

▶ **Lemma 38.** *If* $G \Leftrightarrow \mathcal{L}$ *and* $\vdash \mathcal{P} : \mathcal{L}$, *then (for every* $\mathcal{P}', p, q, v$*):*

- *If* $\mathcal{P}\downarrow$, *then* $G\downarrow$ *and* $\mathcal{L}\Downarrow$.
- *If* $\mathcal{P} \xrightarrow{pq\text{‽}v} \mathcal{P}'$, *then* $G' \Leftrightarrow \mathcal{L}^{¶}$ *and* $\vdash \mathcal{P}' : \mathcal{L}^{¶}$ *and* $\vdash v : t$ *and* $G \xrightarrow{pq\text{‽}t} G'$ *and* $\mathcal{L} \xRightarrow{pq\text{‽}t} \mathcal{L}^{¶}$, *for some* $t, G', \mathcal{L}^{¶}$. ◀

Progress and preservation entail *operational refinement*: every trace of the family of processes (with concrete actions) is also a trace of the family of projections (with analogous abstract actions); moreover, if the family of processes can terminate or deadlock, then also the family of projections can. We formalise this concept directly in the following theorem.

▶ **Theorem 39** (refinement). *If* $\vdash \mathcal{P} : \{G\lceil r\}_{r\in R}$, *and* $\mathcal{P} \xrightarrow{p_1q_1\text{‽}v_1} \cdots \xrightarrow{p_nq_n\text{‽}v_n} \mathcal{P}^{\dagger}$, *then:*

- $\{G\lceil r\}_{r\in R} \xRightarrow{p_1q_1\text{‽}t_1} \cdots \xRightarrow{p_nq_n\text{‽}t_n}$, *and* $\vdash v_1 : t_1$, *and* $\cdots$, *and* $\vdash v_n : t_n$, *for some* $t_1, \ldots, t_n$.
- *If* $\mathcal{P}^{\dagger}\downarrow$, *then* $\mathcal{L}^{\dagger}\downarrow$.
- *If* $\mathcal{P}^{\dagger} \not\downarrow$ *and* $\mathcal{P}^{\dagger} \not\rightarrow$, *then* $\mathcal{L}^{\dagger} \not\downarrow$ *and* $\mathcal{L}^{\dagger} \not\rightarrow$. ◀

▶ Remark 40. Theorem 39 depends on premise $\vdash \mathcal{P} : \{G\lceil r\}_{r\in R}$. To see that checking this premise is decidable, observe that we need to check two sets of properties by rule [$\vdash$]: (1) well-typedness of the processes in $\mathcal{P}$ by the local types in $\{G\lceil r\}_{r\in R}$; (2) well-behavedness of the local types in $\{G\lceil r\}_{r\in R}$. Regarding the first set, the typing rules for processes are defined inductively on the structure of processes. Consequently, the number of applications is finite. Furthermore, checking the premise of rule [$\vdash$-In]/[$\vdash$-Out]/[$\vdash$-Sum1]/[$\vdash$-Sum2] is decidable because the reduction relation of every local type in $\{G\lceil r\}_{r\in R}$ is finite (Remark 26). Thus, checking the first set of properties is decidable. Regarding the second set, see Remark 26. ◀

3.7 Safety and Liveness

We proved operational equivalence for projection (Theorem 23) and operational refinement for type checking (Theorem 39). Together, these main results entail type soundness.

▶ **Corollary 41** (type soundness). *If* $\vdash \mathcal{P} : \{G\lceil r\}_{r\in R}$ *and* $\mathcal{P} \xrightarrow{p_1q_1\text{‽}v_1} \cdots \xrightarrow{p_nq_n\text{‽}v_n} \mathcal{P}^{\dagger}$, *then:*

- Safety: $G \xrightarrow{p_1q_1\text{‽}t_1} \cdots \xrightarrow{p_nq_n\text{‽}t_n}$, *and* $\vdash v_1 : t_1$, *and* $\cdots$, *and* $\vdash v_n : t_n$, *for some* $t_1, \ldots, t_n$.
- Liveness: $\mathcal{P}^{\dagger}\downarrow$, *or* $\mathcal{P}^{\dagger} \rightarrow$. ◀

All communication errors that can give rise to deadlock (Remark 33) are ruled out when $\vdash \mathcal{P} : \{G\lceil r\}_{r\in R}$ holds. Moreover, checking if $\vdash \mathcal{P} : \{G\lceil r\}_{r\in R}$ holds, is decidable (Remark 40).

4 Regular Grammars

In this section, we apply the new techniques for projection and type checking to regular grammars of global types and processes for the first time. To achieve this, we need to revise and extend the definitions of the grammars in §3, termination/reduction rules, and typing rules. In contrast, the definitions of implicit local types, projection, and well-behavedness—as well as the main result of operational equivalence (Theorem 23)—can stay exactly the same as in §3: they were all formulated in general terms of termination and reduction, but not in specific terms of the rules that define them. Thus, only the following changes are needed:

$$\frac{}{\checkmark\downarrow} \qquad \frac{G_i\downarrow}{G_1+G_2\downarrow}$$

$$\frac{G_1\downarrow \quad G_2\downarrow}{G_1\cdot G_2\downarrow} \qquad \frac{}{G^*\downarrow}$$

(a) Termination

$$\frac{}{p \mathbin{\rightarrow\!\!\triangleright} q{:}t \xrightarrow{pq‽t} \checkmark} \qquad \frac{G_i \xrightarrow{\gamma} G'}{G_1+G_2 \xrightarrow{\gamma} G'}$$

$$\frac{G_1 \xrightarrow{\gamma} G_1'}{G_1\cdot G_2 \xrightarrow{\gamma} G_1'\cdot G_2} \qquad \frac{G_1\downarrow \quad G_2 \xrightarrow{\gamma} G'}{G_1\cdot G_2 \xrightarrow{\gamma} G'} \qquad \frac{G \xrightarrow{\gamma} G'}{G^* \xrightarrow{\gamma} G'\cdot G^*}$$

(b) Reduction

Figure 6 Operational semantics of regular global types (standard for regular expressions; e.g., [4])

$\cdots$ (same as in Figure 4b)

$$\frac{\sum\{O_1,\dots,O_n\} \xrightarrow{\pi} P'}{\sum\{O_1,\dots,O_n,I_1,\dots,I_m\} \xrightarrow{\pi} P'} \qquad \frac{\sum\{I_1,\dots,I_m\} \xrightarrow{\pi} P'}{\sum\{O_1,\dots,O_n,I_1,\dots,I_m\} \xrightarrow{\pi} P'}$$

Figure 7 Operational semantics of regular processes – Reduction rules

▶ **Definition 42** (revision of Definition 11). $G ::= p \mathbin{\rightarrow\!\!\triangleright} q{:}t \mid G_1+G_2 \mid G_1\cdot G_2 \mid G^* \mid \checkmark$ ◀

Global type $p \mathbin{\rightarrow\!\!\triangleright} q{:}t$ specifies a synchronous communication. Global types G_1+G_2 and $G_1\cdot G_2$ specify the *choice* between, and the *sequence* of, G_1 and G_2. Global type G^* specifies the *finite repetition* of G. Global type $\checkmark$ specifies the *empty protocol.*

▶ **Definition 43** (revision of Definition 13 and Definition 14). *See Figure 6.* ◀

▶ **Definition 44** (extension of Definition 28). $P ::= \cdots \mid \sum\{O_1,\dots,O_n,I_1,\dots,I_m\}$ ◀

Process $\sum\{O_1,\dots,O_n,I_1,\dots,I_m\}$ implements a non-deterministic *selection* of n output processes and m input processes, simultaneously (i.e., it is a *mixed input/output process*).

▶ **Definition 45** (extension of Definition 32). *See Figure 7.* ◀

▶ **Definition 46** (extension of Definition 34). *See Figure 8.* ◀

While Theorem 23 of operational equivalence is directly applicable to the regular grammar of global types in this section, Theorem 39 of operational refinement requires minor effort: we need to prove a new, yet simple, inductive case for the new typing rule in Figure 8. Then:

▶ **Corollary 47.** *Corollary 41 is applicable to the revisions and extensions in this section.* ◀

Extending the regular grammar of processes with new process operators (including typing rules) requires one to prove additional inductive cases. In contrast, extending the regular grammar of global types with new global type operators (such that Propositions 16–17 continue to be valid) is completely *free*. That is, in this paper, projection (including well-behavedness) and type checking are independent of the syntax of global types; they are dependent only on the operational semantics. The formulations and proofs of our main results are similarly independent. As a result of this independence, our regular grammar of global types is actually "open ended". As a first demonstration of this extensibility, we freely add a few global type operators; they are intended to serve as higher-level abstractions to make the specification of protocols easier. See also Example 8 and Example 9 in §2.2.

▶ **Definition 48** (extension of Definition 42). $G ::= \cdots \mid G_1{;}G_2 \mid G_1 \| G_2 \mid G_1 \bowtie G_2 \mid [G]^{\gamma_1}_{\gamma_2}$ ◀

- Global type $G_1{;}G_2$ specifies the *weak sequence* of G_1 and G_2. It is similar to $G_1\cdot G_2$, except that independent communications in G_1 and G_2 can happen out-of-order. Communications are independent when they have disjoint sets of participating roles. By using G^* instead of $\mu X.G$ for loops (wlog for regularity), weak sequencing yields finite reduction relations.

$\cdots$ (same as in Figure 5) $\qquad \dfrac{\Xi, \Upsilon \vdash \sum\{O_1, \ldots, O_n\} : L \qquad \Xi, \Upsilon \vdash \sum\{I_1, \ldots, I_m\} : L}{\Xi, \Upsilon \vdash \sum\{O_1, \ldots, O_n, I_1, \ldots, I_m\} : L}$

Figure 8 Well-typedness

$\cdots$ (same as in Figure 6a) $\qquad\qquad$ $\cdots$ (same as in Figure 6b)

$$\frac{G_1 \downarrow \qquad G_2 \downarrow}{G_1 \mathbin{;} G_2 \downarrow} \qquad \frac{G_1 \xrightarrow{\gamma} G_1'}{G_1 \mathbin{;} G_2 \xrightarrow{\gamma} G_1' \cdot G_2} \qquad \frac{\mathsf{r}(G_1) \cap \{p, q\} = \emptyset \qquad G_2 \xrightarrow{pq\text{‽}t} G_2'}{G_1 \mathbin{;} G_2 \xrightarrow{pq\text{‽}t} G_1 \mathbin{;} G_2'}$$

$$\frac{G_1 \downarrow \qquad G_2 \downarrow}{G_1 \parallel G_2 \downarrow} \qquad \frac{G_1 \xrightarrow{\gamma} G_1'}{G_1 \parallel G_2 \xrightarrow{\gamma} G_1' \parallel G_2} \qquad \frac{G_2 \xrightarrow{\gamma} G_2'}{G_1 \parallel G_2 \xrightarrow{\gamma} G_1 \parallel G_2'}$$

$$\frac{G_1 \bowtie G_2 \not\to}{G_1 \bowtie G_2 \downarrow} \qquad \frac{G_1 \xrightarrow{\gamma} G_1' \qquad \gamma \notin \mathsf{a}(G_2)}{G_1 \bowtie G_2 \xrightarrow{\gamma} G_1' \bowtie G_2} \qquad \frac{G_2 \xrightarrow{\gamma} G_2' \qquad \gamma \notin \mathsf{a}(G_1)}{G_1 \bowtie G_2 \xrightarrow{\gamma} G_1 \bowtie G_2'} \qquad \frac{G_1 \xrightarrow{\gamma} G_1' \qquad G_2 \xrightarrow{\gamma} G_2'}{G_1 \bowtie G_2 \xrightarrow{\gamma} G_1' \bowtie G_2'}$$

$$\frac{[G]^{\gamma_1}_{\gamma_2} \not\to}{[G]^{\gamma_1}_{\gamma_2} \downarrow} \qquad \frac{G \xrightarrow{\gamma_1} G'}{[G]^{\gamma_1}_{\gamma_2} \xrightarrow{\gamma} [G']^{\gamma_1}_{\gamma_2}} \qquad \frac{G \xrightarrow{\gamma_2} G' \qquad G \overset{\gamma_1}{\not\to}}{[G]^{\gamma_1}_{\gamma_2} \xrightarrow{\gamma} [G']^{\gamma_1}_{\gamma_2}} \qquad \frac{G \xrightarrow{\gamma} G' \qquad \gamma \notin \{\gamma_1, \gamma_2\}}{[G]^{\gamma_1}_{\gamma_2} \xrightarrow{\gamma} [G']^{\gamma_1}_{\gamma_2}}$$

(a) Termination $\qquad$ **(b)** Reduction. Let $\mathsf{a}(G) = \{\gamma \mid G \to \cdots \xrightarrow{\gamma}\}$ and $\mathsf{r}(G) = \{p, q \mid pq\text{‽}t \in \mathsf{a}(G)\}$.

Figure 9 Operational semantics of regular global types, extended

- Global type $G_1 \parallel G_2$ specifies the *interleaving* of G_1 and G_2. We note that interleaving was already present in the original paper on MPST [39], as well as in later papers (e.g., [20, 30, 31, 51]). However, in these papers, G_1 and G_2 need to have disjoint roles or disjoint channels, whereas in this paper, G_1 and G_2 need to have disjoint actions (as a result of well-behavedness); this is a weaker requirement. Interleaving allows us, for instance, to support the global types on rows "Example 13" and "MP workers" in Table 1.
- Global type $G_1 \bowtie G_2$ specifies the *join* of G_1 and G_2: every "unconstrained" communication that occurs only in G_1 or only in G_2 is enabled in $G_1 \bowtie G_2$ if, and only if, it is enabled in G_1 or G_2; every "constrained" communication that occurs both in G_1 and in G_2 is enabled if, and only if, it is enabled in G_1 and G_2. See also Example 49, below.
- Global type $[G]^{\gamma_1}_{\gamma_2}$ specifies the *prioritisation* of high-priority γ_1 over low-priority γ_2 in G.

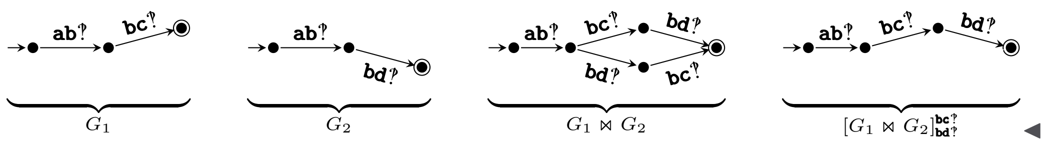
▶ **Example 49.** To exemplify join and prioritisation, let $G_1 = \mathbf{a}{\rightarrowtriangle}\mathbf{b}\cdot\mathbf{b}{\rightarrowtriangle}\mathbf{c}$ and $G_2 = \mathbf{a}{\rightarrowtriangle}\mathbf{b}\cdot\mathbf{b}{\rightarrowtriangle}\mathbf{d}$ (data types omitted). Unconstrained are $\mathbf{b}{\rightarrowtriangle}\mathbf{c}$ (only in G_1) and $\mathbf{b}{\rightarrowtriangle}\mathbf{d}$ (only in G_2); constrained is $\mathbf{a}{\rightarrowtriangle}\mathbf{b}$ (both in G_1 and in G_2). Thus, $G_1 \bowtie G_2$ is equivalent to $\mathbf{a}{\rightarrowtriangle}\mathbf{b} \cdot (\mathbf{b}{\rightarrowtriangle}\mathbf{c} \parallel \mathbf{b}{\rightarrowtriangle}\mathbf{d})$: after the constrained communication, the unconstrained communications are interleaved. Thus, $[G_1 \bowtie G_2]^{\mathbf{bc}\text{‽}}_{\mathbf{bd}\text{‽}}$ is equivalent to $[\mathbf{a}{\rightarrowtriangle}\mathbf{b} \cdot (\mathbf{b}{\rightarrowtriangle}\mathbf{c} \parallel \mathbf{b}{\rightarrowtriangle}\mathbf{d})]^{\mathbf{bc}\text{‽}}_{\mathbf{bd}\text{‽}}$, which is equivalent to $\mathbf{a}{\rightarrowtriangle}\mathbf{b} \cdot \mathbf{b}{\rightarrowtriangle}\mathbf{c} \cdot \mathbf{b}{\rightarrowtriangle}\mathbf{d}$: after $\mathbf{a}{\rightarrowtriangle}\mathbf{b}$ (no priority), $\mathbf{b}{\rightarrowtriangle}\mathbf{c}$ (high priority) must precede $\mathbf{b}{\rightarrowtriangle}\mathbf{d}$ (low priority). ◀

▶ **Definition 50** (extension of Definition 43)**.** *See Figure 9.* ◀

- The termination rules for join and prioritisation state that they can terminate when they cannot reduce. This formalises the design decision that operators that constrain the behaviour of operands should be liberal: they should permit as much behaviour as possible within the constraints they impose (avoid premature termination when reductions are still possible); if less behaviour is required, more constraints can always be imposed.
- The second reduction rule for weak sequencing states that G_2 can start reducing before G_1 has finished reducing, when the roles that participate in the reduction of G_2 are disjoint from those that participate in reductions of G_1 (i.e., these reductions are independent).

- The first reduction rule (resp. second) for join states that it can γ-reduce when G_1 (resp. G_2) can, now, but G_2 (resp. G_1) cannot, ever (i.e., γ is unconstrained). The third reduction rule states that it can γ-reduce when G_1 and G_2 can (i.e., γ is constrained).
- The first reduction rule for prioritisation states that it can γ_1-reduce (high priority) when G can. The second reduction rule states that it can γ_2-reduce (low priority) when it cannot γ_1-reduce. The third reduction rule states that it can γ-reduce when G can.

▶ **Example 51.** The following graphs visualise the operational semantics for Example 49:



◀

We note that an evaluation of the usefulness of the added operators, as practical language primitives, is not really part of the present scope; here, our only aim was to give an impression of the future potential of the new techniques. Other possible global type primitives that may deserve future consideration include delayed choice [5], roles-as-ports composition [47], stateful global types (cf. stateful choreographies [28]), operators for higher-order protocols, and syntax for general models of behaviour (as mentioned towards the end of §1.2).

5 Related Work

This paper contributes to a line of research to increase the expressiveness of MPST [39]. Regarding basic features, previous works have focussed on two limitations of *directed choice* of the form $\sum\{p \rightarrowtriangle q : t_i . G_i\}_{1 \le i \le n}$: **(1)** every branch must start with the same sender and the same receiver as every other branch; **(2)** every "third role" that does not participate in the first communication of every branch must have the same behaviour in every branch.

- **Merging:** Honda et al. address limitation **2** by allowing "third roles" to have different behaviour in different branches when they are timely informed of the chosen branch [18]. The approach relies on a function to syntactically *merge* local types; it is adopted by many (e.g., [25,33,57]), but shown to be brittle [62]. In contrast, using our new projection and type checking techniques, we address limitation **2** without merging (Remark 10). Another approach that addresses limitation **2** without merging was developed by Scalas–Yoshida [62]. It works in three steps: first, every local type is interpreted as an automaton that specifies one role alone (similar to our operational semantics of implicit local types); next, the automata are composed into a product automaton—exponentially sized in the worse case—that specifies all roles together; last, the product automaton is checked for satisfaction of a special temporal logic formula φ, which entails type soundness. However, this method is non-compositional: the premise of the typing rule for families of processes (which depends on satisfaction of φ) cannot be checked separately for every role. Such non-compositional approaches to MPST have already been shown to have scalability issues [44]. Conversely, our typing rule for families is fully compositional (Remark 27).
- **Located/mixed choice:** Several teams of authors address limitation **1** by allowing every branch to start with a different receiver than every other branch. In earlier works that support such *located choice* of the form $\sum\{p \rightarrowtriangle q_i : t_i . G_i\}_{1 \le i \le n}$, communication races in the continuations are forbidden [9,20,31,42,51]; in later works, they are allowed [21–23,54,65]. We support them, too. However, the authors of these papers prove theorems for a closed set of global type operators, including $\sum\{p \rightarrowtriangle q_i : t_i . G_i\}_{1 \le i \le n}$. Instead, we prove theorems for an open set of global type operators, as demonstrated in §2.2 and §4.

Verification of mixed input/output processes using session typing is a long-standing open problem. Progress was made by Casal et al. [19] (binary), Kouzapas–Yoshida (multiparty, but unpublished so far [69, ref. 24]), and Jongmans–Yoshida [44] (multiparty, but no type checking). We can verify multiparty non-deterministic mixed input/output processes for the first time (but not yet deterministic mixed choice), as demonstrated in §2.2.

The usage of the operational semantics of local types was first studied in the context of *multiparty compatibility* [32] and extensions [8, 51, 52]. The idea is to interpret local types as *communicating finite state machines* (CFSM) [11]. Multiparty compatibility, then, is a predicate on the joint state space of the CFSMs to ensure safety and liveness. As such, a key difference between multiparty compatibility (MC) and this paper's well-behavedness (WB) is that MC is non-compositional (i.e., the joint state space must be computed, so MC cannot be checked separately for every role), whereas WB is fully compositional (Remark 27). A rudimentary version of WB was studied by Jongmans–Yoshida [44], but it is less expressive (e.g., they do not support Example 15) and limited to projection (no type checking). A version of WB for global types was studied by Gheri et al. [34], in the context of choreography automata [6], but it is limit to projection (no type checking).

There are several non-traditional techniques for projection in the MPST literature. Lopez et al. [53] capture projection in a decidable type equivalence. Castellani et al. [22] and Hamers et al. [37] do not use projection at all, but type-check families of processes against global types (non-compositional). Last, the concept of implicit local types in this paper generalises an idea by Van Glabbeek et al. [65], who define merging as a local type operator.

6 Conclusion

We introduced two new techniques to significantly improve the expressiveness of the MPST method: projection is based on *implicit* local types instead of explicit; type checking is based on the *operational semantics* of implicit local types instead of on the syntax. Classes of protocols that can now be specified/implemented/verified for the first time using the MPST method include: recursive protocols in which different roles participate in different branches (Example 2, Example 3); protocols in which a receiver chooses the sender of the first communication (Example 6, Example 8, Example 9); protocols in which multiple roles synchronously choose both the sender and the receiver of a next communication (Example 5, Example 6), implemented as mixed input/output processes. We presented the theory of the new techniques, as well as their future potential, and we demonstrated their present capabilities to effectively support regular expressions as global types (not possible before).

As evidence that the new techniques are implementable, we implemented them; this implementation is available as a companion artefact (published in DARTS).

We aim to push the new techniques of this paper forward towards a new branch of research in MPST, centred around operational semantics of local types in typing rules; incidentally, it could be a natural path to explore for behavioural typing in general, too. In particular, we are keen to apply the new techniques for projection and type checking also to *asynchronous communication* and *parametrised protocols/indexed roles* [25, 33]. In both cases, the main challenge is how to ensure decidability (keep the reduction relations finite).

References

1 Luca Aceto, Anna Ingólfsdóttir, and Jirí Srba. The algorithmics of bisimilarity. In *Advanced Topics in Bisimulation and Coinduction*, volume 52 of *Cambridge tracts in theoretical computer science*, pages 100–172. Cambridge University Press, 2012.

2 Bernardo Almeida, Andreia Mordido, Peter Thiemann, and Vasco T. Vasconcelos. Polymorphic lambda calculus with context-free session types. *Inf. Comput.*, 289(Part):104948, 2022.

3 Bernardo Almeida, Andreia Mordido, and Vasco T. Vasconcelos. Deciding the bisimilarity of context-free session types. In *TACAS (2)*, volume 12079 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2020.

4 Jos C. M. Baeten, Flavio Corradini, and Clemens Grabmayer. A characterization of regular expressions under bisimulation. *J. ACM*, 54(2):6, 2007.

5 Jos C. M. Baeten and Sjouke Mauw. Delayed choice: an operator for joining message sequence charts. In *FORTE*, volume 6 of *IFIP Conference Proceedings*, pages 340–354. Chapman & Hall, 1994.

6 Franco Barbanera, Ivan Lanese, and Emilio Tuosto. Choreography automata. In *COORDINATION*, volume 12134 of *Lecture Notes in Computer Science*, pages 86–106. Springer, 2020.

7 J. F. A. K. Van Benthem. Hintikka on analyticity. *Journal of Philosophical Logic*, 3(4):419–431, 1974. doi:10.1007/bf00257484.

8 Laura Bocchi, Julien Lange, and Nobuko Yoshida. Meeting deadlines together. In *CONCUR*, volume 42 of *LIPIcs*, pages 283–296. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

9 Laura Bocchi, Hernán C. Melgratti, and Emilio Tuosto. Resolving non-determinism in choreographies. In *ESOP*, volume 8410 of *Lecture Notes in Computer Science*, pages 493–512. Springer, 2014.

10 Laura Bocchi, Weizhen Yang, and Nobuko Yoshida. Timed multiparty session types. In *CONCUR*, volume 8704 of *Lecture Notes in Computer Science*, pages 419–434. Springer, 2014.

11 Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.

12 Mario Bravetti and Gianluigi Zavattaro. Towards a unifying theory for choreography conformance and contract compliance. In *SC@ETAPS*, volume 4829 of *Lecture Notes in Computer Science*, pages 34–50. Springer, 2007.

13 Mario Bravetti and Gianluigi Zavattaro. Contract compliance and choreography conformance in the presence of message queues. In *WS-FM*, volume 5387 of *Lecture Notes in Computer Science*, pages 37–54. Springer, 2008.

14 Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Choreography and orchestration conformance for system design. In *COORDINATION*, volume 4038 of *Lecture Notes in Computer Science*, pages 63–81. Springer, 2006.

15 Sara Capecchi, Ilaria Castellani, and Mariangiola Dezani-Ciancaglini. Typing access control and secure information flow in sessions. *Inf. Comput.*, 238:68–105, 2014.

16 Sara Capecchi, Ilaria Castellani, and Mariangiola Dezani-Ciancaglini. Information flow safety in multiparty sessions. *Mathematical Structures in Computer Science*, 26(8):1352–1394, 2016.

17 Sara Capecchi, Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Tamara Rezk. Session types for access and information flow control. In *CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 237–252. Springer, 2010.

18 Marco Carbone, Nobuko Yoshida, and Kohei Honda. Asynchronous session types: Exceptions and multiparty interactions. In *SFM*, volume 5569 of *Lecture Notes in Computer Science*, pages 187–212. Springer, 2009.

19 Filipe Casal, Andreia Mordido, and Vasco T. Vasconcelos. Mixed sessions. *Theor. Comput. Sci.*, 897:23–48, 2022.

20 Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, and Luca Padovani. On global types and multi-party session. *Logical Methods in Computer Science*, 8(1), 2012.

21 Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Paola Giannini. Reversible sessions with flexible choices. *Acta Informatica*, 56(7-8):553–583, 2019.

22 Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Paola Giannini. Asynchronous sessions with input races. *CoRR*, abs/2203.12876, 2022.

23 Ilaria Castellani, Mariangiola Dezani-Ciancaglini, Paola Giannini, and Ross Horne. Global types with internal delegation. *Theor. Comput. Sci.*, 807:128–153, 2020.

24 Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Jorge A. Pérez. Self-adaptation and secure information flow in multiparty communications. *Formal Asp. Comput.*, 28(4):669–696, 2016.

25 David Castro, Raymond Hu, Sung-Shik Jongmans, Nicholas Ng, and Nobuko Yoshida. Distributed programming using role-parametric session types in go: statically-typed endpoint apis for dynamically-instantiated communication structures. *PACMPL*, 3(POPL):29:1–29:30, 2019.

26 Guillermina Cledou, Luc Edixhoven, Sung-Shik Jongmans, and José Proença. API generation for multiparty session types, revisited and revised using scala 3. In *ECOOP*, volume 222 of *LIPIcs*, pages 27:1–27:28. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

27 Alex Coto, Roberto Guanciale, and Emilio Tuosto. An abstract framework for choreographic testing. *J. Log. Algebraic Methods Program.*, 123:100712, 2021.

28 Luís Cruz-Filipe, Kim S. Larsen, and Fabrizio Montesi. The paths to choreography extraction. In *FoSSaCS*, volume 10203 of *Lecture Notes in Computer Science*, pages 424–440, 2017.

29 Ugo de'Liguoro, Hernán C. Melgratti, and Emilio Tuosto. Towards refinable choreographies. *J. Log. Algebraic Methods Program.*, 127:100776, 2022.

30 Pierre-Malo Deniélou and Nobuko Yoshida. Dynamic multirole session types. In *POPL*, pages 435–446. ACM, 2011.

31 Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty session types meet communicating automata. In *ESOP*, volume 7211 of *Lecture Notes in Computer Science*, pages 194–213. Springer, 2012.

32 Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *ICALP (2)*, volume 7966 of *Lecture Notes in Computer Science*, pages 174–186. Springer, 2013.

33 Pierre-Malo Deniélou, Nobuko Yoshida, Andi Bejleri, and Raymond Hu. Parameterised multiparty session types. *Logical Methods in Computer Science*, 8(4), 2012.

34 Lorenzo Gheri, Ivan Lanese, Neil Sayers, Emilio Tuosto, and Nobuko Yoshida. Design-by-contract for flexible multiparty session protocols. In *ECOOP*, volume 222 of *LIPIcs*, pages 8:1–8:28. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

35 Roberto Guanciale and Emilio Tuosto. Realisability of pomsets. *J. Log. Algebraic Methods Program.*, 108:69–89, 2019.

36 Ruben Hamers, Erik Horlings, and Sung-Shik Jongmans. The discourje project: run-time verification of communication protocols in clojure. *Int. J. Softw. Tools Technol. Transf.*, 24(5):757–782, 2022.

37 Ruben Hamers and Sung-Shik Jongmans. Discourje: Runtime verification of communication protocols in clojure. In *TACAS (1)*, volume 12078 of *Lecture Notes in Computer Science*, pages 266–284. Springer, 2020.

38 Jaakko Hintikka. *Logic, Language-Games and Information: Kantian Themes in the Philosophy of Logic.* Oxford, England: Oxford, Clarendon Press, 1973.

39 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *POPL*, pages 273–284. ACM, 2008.

40 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *J. ACM*, 63(1):9:1–9:67, 2016.

41 Raymond Hu and Nobuko Yoshida. Hybrid session verification through endpoint API generation. In *FASE*, volume 9633 of *Lecture Notes in Computer Science*, pages 401–418. Springer, 2016.

42 Raymond Hu and Nobuko Yoshida. Explicit connection actions in multiparty session types. In *FASE*, volume 10202 of *Lecture Notes in Computer Science*, pages 116–133. Springer, 2017.

43 Sung-Shik Jongmans and Francisco Ferreira. Synthetic Behavioural Typing: Sound, Regular Multiparty Sessions via Implicit Local Types (Technical Report). Technical Report OUNL-CS-2023-01, Open University of the Netherlands, 2023.

44 Sung-Shik Jongmans and Nobuko Yoshida. Exploring type-level bisimilarity towards more expressive multiparty session types. In *ESOP*, volume 12075 of *Lecture Notes in Computer Science*, pages 251–279. Springer, 2020.

45 Alex C. Keizer, Henning Basold, and Jorge A. Pérez. Session coalgebras: A coalgebraic view on regular and context-free session types. *ACM Trans. Program. Lang. Syst.*, 44(3):18:1–18:45, 2022.

46 Jonathan King, Nicholas Ng, and Nobuko Yoshida. Multiparty session type-safe web development with static linearity. In *PLACES@ETAPS*, volume 291 of *EPTCS*, pages 35–46, 2019.

47 Christian Koehler and Dave Clarke. Decomposing port automata. In *SAC*, pages 1369–1373. ACM, 2009.

48 Nicolas Lagaillardie, Rumyana Neykova, and Nobuko Yoshida. Implementing multiparty session types in rust. In *COORDINATION*, volume 12134 of *Lecture Notes in Computer Science*, pages 127–136. Springer, 2020.

49 Nicolas Lagaillardie, Rumyana Neykova, and Nobuko Yoshida. Stay safe under panic: Affine rust programming with multiparty session types. In *ECOOP*, volume 222 of *LIPIcs*, pages 4:1–4:29. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

50 Ivan Lanese, Claudio Guidi, Fabrizio Montesi, and Gianluigi Zavattaro. Bridging the gap between interaction- and process-oriented choreographies. In *SEFM*, pages 323–332. IEEE Computer Society, 2008.

51 Julien Lange, Emilio Tuosto, and Nobuko Yoshida. From communicating machines to graphical choreographies. In *POPL*, pages 221–232. ACM, 2015.

52 Julien Lange and Nobuko Yoshida. Verifying asynchronous interactions via communicating session automata. In *CAV (1)*, volume 11561 of *Lecture Notes in Computer Science*, pages 97–117. Springer, 2019.

53 Hugo A. López, Eduardo R. B. Marques, Francisco Martins, Nicholas Ng, César Santos, Vasco Thudichum Vasconcelos, and Nobuko Yoshida. Protocol-based verification of message-passing parallel programs. In *OOPSLA*, pages 280–298. ACM, 2015.

54 Rupak Majumdar, Madhavan Mukund, Felix Stutz, and Damien Zufferey. Generalising projection in asynchronous multiparty session types. In *CONCUR*, volume 203 of *LIPIcs*, pages 35:1–35:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

55 Robin Milner. A theory of type polymorphism in programming. *J. Comput. Syst. Sci.*, 17(3):348–375, 1978.

56 Anson Miu, Francisco Ferreira, Nobuko Yoshida, and Fangyi Zhou. Communication-safe web programming in typescript with routed multiparty session types. In *CC*, pages 94–106. ACM, 2021.

57 Rumyana Neykova, Laura Bocchi, and Nobuko Yoshida. Timed runtime monitoring for multiparty conversations. *Formal Asp. Comput.*, 29(5):877–910, 2017.

58 Rumyana Neykova, Raymond Hu, Nobuko Yoshida, and Fahd Abdeljallal. A session type provider: compile-time API generation of distributed protocols with refinements in f#. In *CC*, pages 128–138. ACM, 2018.

59 Nicholas Ng and Nobuko Yoshida. Pabble: parameterised scribble. *Service Oriented Computing and Applications*, 9(3-4):269–284, 2015.

60 Luca Padovani. Context-free session type inference. *ACM Trans. Program. Lang. Syst.*, 41(2):9:1–9:37, 2019.

61 Alceste Scalas, Ornela Dardha, Raymond Hu, and Nobuko Yoshida. A linear decomposition of multiparty sessions for safe distributed programming. In *ECOOP*, volume 74 of *LIPIcs*, pages 24:1–24:31. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

62 Alceste Scalas and Nobuko Yoshida. Less is more: multiparty session types revisited. *Proc. ACM Program. Lang.*, 3(POPL):30:1–30:29, 2019.

63 Peter Thiemann and Vasco T. Vasconcelos. Context-free session types. In *ICFP*, pages 462–475. ACM, 2016.

64 Emilio Tuosto and Roberto Guanciale. Semantics of global view of choreographies. *J. Log. Algebraic Methods Program.*, 95:17–40, 2018.

65 Rob van Glabbeek, Peter Höfner, and Ross Horne. Assuming just enough fairness to make session types complete for lock-freedom. In *LICS*, pages 1–13. IEEE, 2021.

66 Rob J. van Glabbeek. The linear time-branching time spectrum (extended abstract). In *CONCUR*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer, 1990.

67 Rob J. van Glabbeek. The linear time - branching time spectrum II. In *CONCUR*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 1993.

68 Rob J. van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43(3):555–600, 1996.

69 Vasco T. Vasconcelos, Filipe Casal, Bernardo Almeida, and Andreia Mordido. Mixed sessions. In *ESOP*, volume 12075 of *Lecture Notes in Computer Science*, pages 715–742. Springer, 2020.

70 Nobuko Yoshida, Fangyi Zhou, and Francisco Ferreira. Communicating finite state machines and an extensible toolchain for multiparty session types. In *FCT*, volume 12867 of *Lecture Notes in Computer Science*, pages 18–35. Springer, 2021.

71 Fangyi Zhou, Francisco Ferreira, Raymond Hu, Rumyana Neykova, and Nobuko Yoshida. Statically verified refinements for multiparty protocols. *Proc. ACM Program. Lang.*, 4(OOPSLA):148:1–148:30, 2020.