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

Sung-Shik Jongmans
Department of Computer Science, Open University, Heerlen, The Netherlands
Centrum Wiskunde & Informatica (CWI), NWO-I, Amsterdam, 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.2023.42

Category Pearl/Brave New Idea

Supplementary Material Software (ECOOP 2023 Artifact Evaluation approved artifact):
https://doi.org/10.4230/DARTS.9.2.18


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”).
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.

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.

**Example 1.** The *Summation* protocol consists of roles *Alice* ($a$), *Bob* ($b$), and *Carol* ($c$). First, zero or more numbers are communicated from Alice to Bob. Next, a token (*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:

$$G = \mu X. \text{a} \to \text{b} : \left\{ \begin{array}{l} \text{Nat} \cdot X \\ \text{Unit} \cdot \text{b} \to \text{c} : \text{Nat} \cdot \check \end{array} \right. $$

Informally, global type $p \to q : \{ t_i, G_i \}_{1 \leq i \leq n}$ 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:

$$L_a = \mu X. \text{b} \bowtie \left\{ \begin{array}{l} \text{Nat} \cdot X \\ \text{Unit} \cdot \check \end{array} \right. $$

$$L_b = \mu X. \text{a} \bowtie \left\{ \begin{array}{l} \text{Nat} \cdot X \\ \text{Unit} \cdot \text{c} : \text{Nat} \cdot \check \end{array} \right. $$

Informally, local types $q \bowtie \{ t_i, L_i \}_{1 \leq i \leq n}$ and $p \bowtie \{ t_i, L_i \}_{1 \leq i \leq n}$ 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:

$$P_a = \text{sum} : \text{Nat} \bowtie 0 $$

$$P_b = \text{loop} (\text{sum} : \text{Nat} \bowtie 0) \left\{ \begin{array}{l} \text{a} (\text{x} : \text{Nat} \cdot \text{rec} (\text{sum} + \text{x})) \\ \text{a} (\_ : \text{Unit} \cdot \text{c} (\text{sum}) \bowtie 0) \end{array} \right. $$

Informally, process $\pi(e).P$ implements the send of the value of expression $e$ to role $q$, while process $\sum [p(x : t_i) . P_i]_{1 \leq i \leq n}$ 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 $P_a$ (resp. $P_b$) is also specified in $L_a$ (resp. $L_b$).}

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.
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]:
  \[ G ::= p \rightarrow q; \{ t_i . G_i \}_{1 \leq i \leq n} \mid \mu X . G \mid X \mid \checkmark \]
  Thus, global types can specify that a sender chooses the data type but not the receiver.

- In recent papers [21–23, 54, 65]:
  \[ G ::= p \rightarrow \{ q ; t_i . G_i \}_{1 \leq i \leq n} \mid \mu X . G \mid X \mid \checkmark \]
  Thus, global types can specify that a sender chooses the data type 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:

\[ G ::= p \rightarrow q : t \mid G_1 + G_2 \mid G_1 \cdot G_2 \mid G^* \mid \checkmark \mid \cdots \]

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:
42:4  Sound, Regular Multiparty Sessions via Implicit Local Types

- **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:

$$L_{old} = \mu X. a \& \{\text{Nat}.X, \text{Unit}.c \oplus \text{Nat}.✓\} \quad L_{new} = G \upharpoonright b$$

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:

$$\Xi \vdash e : t_k \quad \Xi \vdash P : L_k \quad L \xrightarrow{q_{TI}} L' \quad \Xi \vdash \{q\}.P : L$$

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 $G \upharpoonright r$, 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.,

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).
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.

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 $P_1, \ldots, P_n$ for roles $r_1, \ldots, r_n$.

1b. A tool computes the operational semantics of $G$ and of the implicit local types $G|_{r_1}, \ldots, G|_{r_n}$ in the form of a termination predicate and a reduction relation for every role. Every $G|_{r_i}$ 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 $G|_{r_i}$ is well-behaved. If so, then $G$ is operationally equivalent to $G|_{r_1}, \ldots, G|_{r_n}$. That is, $G$ mimics $G|_{r_1}, \ldots, G|_{r_n}$, 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 $G|_{r_i}$. If so, then $G|_{r_1}, \ldots, G|_{r_n}$ is operationally refined by $P_1, \ldots, P_n$. That is, $G|_{r_1}, \ldots, G|_{r_n}$ mimics $P_1, \ldots, P_n$, but not necessarily vice versa: $G|_{r_1}, \ldots, G|_{r_n}$ may specify more behaviour than $P_1, \ldots, P_n$ 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:

$$G ::= p \mapsto q; \{t_i, G_i\}_{1 \leq i \leq n} \mid \mu X.G \mid X \checkmark \quad \text{output process} \quad \text{input process}$$

$$P ::= \sum\{O_1, \ldots, O_n\} \mid \sum\{I_1, \ldots, I_n\} \mid \cdots \quad O ::= \overline{c}.P \quad I ::= p(x:t).P$$

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):

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

$$P_a = 5(5).5(6).5(\text{unit}).0$$

$$P_b = \text{loop}(\text{sum}:\text{Nat}=0) \sum \begin{cases} a(x:\text{Nat}).\text{recur}(\text{sum}+x) \\ a(x:Unit).\overline{c}.(x).0 \end{cases}$$

$$P_c = b(c:\text{Nat}).0$$
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 $G$:

```
ab?Nat ↦ ab?Unit
bc?Nat
```

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 $G|a$, $G|b$, and $G|c$ (the projections of $G$):

```
b!Nat ↦ b!Unit
T
```

```
a!Nat ↦ a!Unit
c!Nat
```

```
τ ↦ τ
```

Thus:

- 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 $G|a$, $G|b$, and $G|c$ are well-behaved, so $G$ is operationally equivalent to $\{G|a, G|b, G|c\}$.

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.

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 $O_i = \vec{t}(e)$, 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 $I_j = p(x:t)$.
The following global type specifies the protocol:

\[ G = \sigma \text{String} \cdot \sigma \text{Nat} \cdot \mu X. a \to b \cdot \begin{cases} \text{Nat} \cdot b \to a : \{ \text{Acc} \cdot a \to s : \text{Acc} \cdot \checkmark, \text{Rej} \cdot X \} \\ \text{Rej} \cdot a \to s : \text{Rej} \cdot \checkmark \end{cases} \]

The following processes implement Alice, Bob, and Seller:

\[
P_a = \tilde{\alpha}("\text{foo}" : \text{String}) \cdot s(x : \text{Nat}) \cdot \tilde{\delta}(x/2) \cdot \sum \left\{ \begin{array}{l} b(- : \text{Acc}) \cdot \tilde{\eta}(\text{acc}) \cdot 0 \\ b(- : \text{Rej}) \cdot \tilde{\delta}(x/3) \cdot \sum \left\{ \begin{array}{l} b(- : \text{Acc}) \cdot \tilde{\eta}(\text{acc}) \cdot 0 \\ b(- : \text{Rej}) \cdot \tilde{\delta}(\text{rej}) \cdot \tilde{\eta}(\text{rej}) \cdot 0 \end{array} \right. \end{array} \right. \\
P_b = \text{loop} \sum \{ a(y : \text{Nat}) : \text{if } y < 10 \ (\tilde{\delta}(\text{acc}) \cdot 0) \ (\tilde{\delta}(\text{rej}) \cdot \text{recur}) \cdot a(- : \text{Rej}) \cdot 0 \} \\
P_c = a(z : \text{String}) \cdot \tilde{\alpha}((\text{price}(z) \cdot \sum \{ a(- : \text{Acc}) \cdot 0, a(- : \text{Rej}) \cdot 0 \})
\]

\(P_a\) 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. \(P_b\) implements that Bob is willing to contribute at most ten units of currency.
1b. The following graphs visualise the operational semantics of \( G \), \( G \mid \text{b} \), and \( G \mid \text{s} \):

![Graphs visualising operational semantics](image)

2. It can be checked that \( G \mid \text{b} \) and \( G \mid \text{s} \) are well-behaved, in the same way as in Example 2. Furthermore, \( G \mid \text{a} \) is trivially well-behaved, as Alice participates in every communication, so the operational semantics of \( G \mid \text{a} \) has no \( \tau \)-reductions. Thus, \( G \) is operationally equivalent to \( \{G \mid \text{a}, G \mid \text{b}, G \mid \text{s}\} \) (Theorem 23).

3. It can be checked that \( P \mid \text{a} \) is well-typed by \( G \mid \text{a} \) (by twice traversing the cycle in \( G \mid \text{a} \)), \( P \mid \text{b} \) is well-typed by \( G \mid \text{b} \), and \( P \mid \text{s} \) is well-typed by \( G \mid \text{s} \). Thus, \( \{G \mid \text{a}, G \mid \text{b}, G \mid \text{s}\} \) is operationally refined by \( \{P \mid \text{a}, P \mid \text{b}, P \mid \text{s}\} \) (Theorem 39).

Together, operational equivalence (step 2) and operational refinement (step 3) imply that \( \{P \mid \text{a}, P \mid \text{b}, P \mid \text{s}\} \) 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:

\[
G ::= p \rightarrow q : t \mid G_1 + G_2 \mid G_1 \cdot G_2 \mid G^* \mid \checkmark \quad P ::= \sum \{O_1, \ldots, O_n, I_1, \ldots, I_m\} \mid \cdots
\]

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

\[
G = (a \rightarrow b:\text{Unit} + b \rightarrow a:\text{Unit}) \cdot (a \rightarrow b:\text{Unit} + b \rightarrow a:\text{Unit}) \cdot (a \rightarrow b:\text{Unit} + b \rightarrow a:\text{Unit})
\]

The following processes implement Alice and Bob:

\[
P_a = \sum \left\{ \begin{array}{l}
\sum \{ \text{\#unit}.0 \\ b(\_:\text{Unit}).0 \\
\} \\
\sum \{ \text{\#unit}.0 \\ b(\_:\text{Unit}).0 \\
\}
\end{array} \right.
\]

\(P_b = \ldots \) (similar to \(P_a\))

1b. The following graphs visualise the operational semantics of \(G\) and \(G|a\):

2. It can be checked that \(G|a\) is well-behaved, in the same way as in Example 2. Similarly, \(G|b\) is well-behaved. Thus, \(G\) is operationally equivalent to \(\{G|a, G|b\\}\) (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 $P_\mathbf{a}$ is well-typed by $G\mid \mathbf{a}$. In particular, as $P_\mathbf{a}$ 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 $G\mid \mathbf{a}$. Similarly, $P_\mathbf{b}$ is well-typed by $G\mid \mathbf{b}$. Thus, $\{G\mid \mathbf{a}, G\mid \mathbf{b}\}$ is operationally refined by $\{P_\mathbf{a}, P_\mathbf{b}\}$ (Theorem 39).

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

The Role-based Binomial 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., $P_\mathbf{a}$ and $P_\mathbf{b}$ 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 ($\mathbf{a}$), Bob ($\mathbf{b}$), and Server ($\mathbf{s}$). Concurrently, it has three subprotocols:

- **Alice and Server (AS):** First, an “acquire” message ($\mathbf{acq}$) is communicated from Alice to Server. Next, a “permission” message ($\mathbf{perm}$) is communicated from Server to Alice. Next, zero or more “usage” messages ($\mathbf{use}$) are communicated from Alice to Server. Last, a “release” message ($\mathbf{rel}$) 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 = \{a \rightarrow s: \mathbf{Acq} + \{a \rightarrow a: \mathbf{Perm} \cdot a \rightarrow s: \mathbf{Use}^* \cdot a \rightarrow a: \mathbf{Perm} \cdot a \rightarrow s: \mathbf{Use}^* \cdot a \rightarrow s: \mathbf{Rel} \cdot G'_2\} \begin{cases} \mathbf{G}_1' \begin{cases} a \rightarrow s: \mathbf{Acq} \cdot + \{a \rightarrow a: \mathbf{Perm} \cdot a \rightarrow s: \mathbf{Use}^* \cdot b \rightarrow s: \mathbf{Acq} \cdot + \{b \rightarrow s: \mathbf{Perm} \cdot b \rightarrow s: \mathbf{Use}^* \cdot b \rightarrow s: \mathbf{Rel} \cdot G'_2\} \end{cases} \end{cases} \}

G'_1 = s \rightarrow a: \mathbf{Perm} \cdot a \rightarrow s: \mathbf{Use}^* \cdot a \rightarrow s: \mathbf{Rel} \quad G'_2 = s \rightarrow b: \mathbf{Perm} \cdot b \rightarrow s: \mathbf{Use}^* \cdot b \rightarrow s: \mathbf{Rel}

The following processes implement Alice, Bob, and Server:

$$
\begin{align*}
P_\mathbf{a} &= \mathbb{H}(\mathbf{acq}) \cdot \mathbb{a}(\_ : \mathbf{Perm}) \cdot \mathbb{a}(\_ : \mathbf{use}) \cdot \mathbb{a}(\_ : \mathbf{rel}) \cdot 0 \\
P_\mathbf{b} &= \mathbb{H}(\mathbf{acq}) \cdot \mathbb{a}(\_ : \mathbf{Perm}) \cdot \mathbb{b}(\_ : \mathbf{use}) \cdot \mathbb{b}(\_ : \mathbf{rel}) \cdot 0 \\
P_\mathbf{s} &= \sum \{a(\mathbf{acq1}: \mathbf{Acq}) \cdot \mathbb{H}(\_ : \mathbf{perm}) \cdot \text{loop} \sum \{b(\_ : \mathbf{acq2}: \mathbf{Acq}) \cdot \mathbb{H}(\_ : \mathbf{rel}) \cdot b(\_ : \mathbf{acq1}: \mathbf{Acq}) \cdot (\cdots) \} \}
\end{align*}

(version 1) \quad P''_\mathbf{s} = \sum \{a(\mathbf{acq1}: \mathbf{Acq}) \cdot \mathbb{H}(\_ : \mathbf{perm}) \cdot (\cdots) \}

(version 2) \quad P''_\mathbf{s} = \text{if } \text{alice \_ goes \_ first}(\mathbf{acq1}, \mathbf{acq2}) \cdot (\mathbb{H}(\_ : \mathbf{perm}) \cdot (\cdots) \}) \cdot (\mathbb{H}(\_ : \mathbf{perm}) \cdot (\cdots) \}

(version 3) \quad P''_\mathbf{s} = \mathbb{H}(\_ : \mathbf{perm}) \cdot (\cdots) \}

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 $P_\mathbf{a}$ is between a send and a receive (mixed input/output).
1b. The following graphs visualise the operational semantics of $G$ and $G\mid a$:

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

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

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

Together, operational equivalence (step 2) and operational refinement (step 3) imply that $\{P\mid a, P\mid b, P\mid s\}$ 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., $P\mid a$, $P\mid b$, and $P\mid s$ 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:

$$G ::= \cdots \mid G_1 ; G_2 \mid G_1 \| G_2 \mid G_1 \times G_2 \mid [G]_{\gamma_1} \mid \cdots$$

**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.

$$G = (a\to b_1: Unit \cdot b_1\to c_1: Unit) ;$$

$$(a\to b_2: Unit \cdot b_2\to c_2: Unit)$$
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. 

\[\textbf{Example 8.} \text{ We re-apply steps 1a, 1b, 2, and 3 to the Acquire–Use–Release protocol:} \]

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

\[G = (G_{\text{AS}} \parallel G_{\text{BS}}) \times G_{\text{ME}}\]

\[G_{\text{AS}} = a \rightarrow s: \text{Acq} \cdot a \rightarrow s: \text{Perm} \cdot a \rightarrow s: \text{Use} \cdot a \rightarrow s: \text{Rel} \]

\[G_{\text{BS}} = b \rightarrow s: \text{Acq} \cdot b \rightarrow s: \text{Perm} \cdot b \rightarrow s: \text{Use} \cdot b \rightarrow s: \text{Rel} \]

\[G_{\text{ME}} = + \{s \rightarrow a: \text{Perm} \cdot a \rightarrow s: \text{Rel} \cdot s \rightarrow b: \text{Perm}
\{s \rightarrow b: \text{Perm} \cdot b \rightarrow s: \text{Rel} \cdot s \rightarrow a: \text{Perm}\}\]

Informally, global types \(G_1 \parallel G_2\) and \(G_1 \times G_2\) specify interleaving and join. In general, join demands that every role complies with both of its operands. In this example, join specifies that subprotocol \(\text{ME}\) in Example 6 constrains the interleaving of subprotocols \(\text{AS}\) and \(\text{BS}\). That is, the three subprotocols are modularly specified as global types \(G_{\text{AS}}, G_{\text{BS}},\) and \(G_{\text{ME}},\) and composed as intended using \(\parallel\) and \(\times\); the result is an exponentially more succinct – and arguably easier to write – specification than in Example 6.

The same processes \(P_a, P_b,\) and \(P_s\), with any version of \(P_s''\), as in Example 6 are used.

1b. The same graphs as in Example 6 visualise the operational semantics of \(G\) and \(G|a\).

2. As in Example 6, \(G\) is operationally equivalent to \(\{G|a, G|b, G|s\}\).

3. As in Example 6, \(\{G|a, G|b, G|s\}\) is operationally refined by \(\{P_a, P_b, P_s\}\).

\[\textbf{Example 9.} \text{ We apply steps 1a, 1b, 2, and 3 to a restricted version of the Acquire–Use–Release protocol in which, after receiving an “acquire” message from both Alice and Bob, Server must send a “permission” message first to Alice and second to Bob (static order):} \]

1a. The following global type specifies the protocol:

\[G = ([G_{\text{AS}} \parallel G_{\text{BS}}] \times G_{\text{ME}})_{\text{Perm}} \times G_{\text{AS}}, G_{\text{BS}}, G_{\text{ME}} = \ldots \text{ (same as in Example 8)}\]

Informally, global type \([G|a]_{\gamma_1}\) specifies the prioritisation of global action \(\gamma_1\) (superscript indicates “high” priority) over global action \(\gamma_2\) (subscript indicates “low” priority) in \(G\).

The same processes \(P_a, P_b,\) and \(P_s\), with version 3 of \(P_s''\), as in Example 6 are used.

1b. The same graphs as in Example 6 visualise the operational semantics of \(G\) and \(G|a\), but without the dashed edges.

2. As in Example 6, \(G\) is operationally equivalent to \(\{G|a, G|b, G|s\}\).

3. As in Example 6, \(\{G|a, G|b, G|s\}\) is operationally refined by \(\{P_a, P_b, P_s\}\).

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

<table>
<thead>
<tr>
<th>name</th>
<th>global type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example 9 [vG21]</td>
<td>$G = (b \rightarrow se:Talk) \cdot b \rightarrow se:Buy \cdot se \rightarrow ab:Order$</td>
</tr>
<tr>
<td>Example 13 [vG21]</td>
<td>$G = ((b1 \rightarrow s1:Wait) \cdot b1 \rightarrow s1:Order) \parallel ((b2 \rightarrow s2:Wait) \cdot b2 \rightarrow s2:Order)$</td>
</tr>
<tr>
<td>Example 15 [vG21]</td>
<td>$G = (b \rightarrow s1:Order_1 \cdot b \rightarrow s2:Wait) \cdot b \rightarrow s2:Order_2 \cdot b \rightarrow s1:Done$</td>
</tr>
<tr>
<td>OAuth2 [SY19]</td>
<td>$G = (s \rightarrow c:Login \cdot c \rightarrow a:Password \cdot a \rightarrow s:Auth) + (s \rightarrow c:Cancel \cdot c \rightarrow a:Quit)$</td>
</tr>
<tr>
<td>Rec. map/reduce [SY19]</td>
<td>$G = G_1 \cdot (r \rightarrow m:Continue \cdot G_1)^* \cdot r \rightarrow m:Stop \cdot m \rightarrow w1:Result$</td>
</tr>
<tr>
<td>MP workers [SY19]</td>
<td>$G_1 = m \rightarrow w1:Data\cdot w1 \rightarrow r:\text{Result}$; $G_2 = (w1 \rightarrow w1 \rightarrow w1:Data \cdot w1 \rightarrow w1:Result)^* \cdot w1 \rightarrow w1 \rightarrow w1:Result$</td>
</tr>
</tbody>
</table>

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 $X$ denote a set of variables, ranged over by $x$. Let $\mathbb{V} = \{\text{true, false}, 0, 1, 2, \ldots\}$ denote a set of values, ranged over by $v$. Let $\mathbb{E} = X \cup \mathbb{V} \cup \{\text{false}, 2+3, \ldots\}$ denote a set of expressions, ranged over by $e$. Let $e[v/x]$ denote substitution of $v$ for $x$ in $e$.

- **Static semantics**: Let $\mathbb{T} = \{\text{Bool, Nat, \ldots}\}$ denote a set of data types, ranged over by $t$. Let $(X \times T)^*$ denote the set of data typing contexts (i.e., lists of variable–type pairs), ranged over by $\Xi$. Let $\Xi \vdash e : t$ denote well-typedness of $e$ by $t$ in $\Xi$.

- **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. Undefinability of eval($e$) is a form of “going wrong” [55]; it can give rise to deadlock (Remark 33), prevented by well-typedness ($\S$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:

  $$G ::= p \rightarrow q; \{t_i, G_i\}_{1 \leq i \leq n} \mid \mu X.G \mid X \mid \top$$

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

  Global type $p \rightarrow q; \{t_i, G_i\}_{1 \leq i \leq n}$ 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 $\top$ specifies the empty protocol. Local type $G \mid r$ specifies the projection of $G$ onto $r$. Thus, projection is a local type operator instead of a function on global types: $G \mid r$ 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 $\mathbb{I} = \{pq!t.pq?t!t\mid p \neq q\}$ and $\mathbb{A} = \bigcup\{pq!t.pq?t!t\mid p \neq q\} \cup \{t\}$ denote the sets of (abstract) global actions and (abstract) local actions, ranged over by $\gamma$ and $\lambda$. Let $\mathbb{A} = \mathbb{I} \cup \mathbb{A}$ denote the set of (abstract) actions, ranged over by $\alpha$. 

ECOOP 2023
Sound, Regular Multiparty Sessions via Implicit Local Types

The predicate induced by the rules in Figure 3a, while the relation induced by the rules in Figure 3b, while Termination

Definition 13. Below, we define the termination predicate and reduction relation on global/local types.

3.2 Global Types – Operational Semantics

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

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

- Definition 14. Let $G \xrightarrow{r} G'$, $L \xrightarrow{\gamma} L'$, and $L \xrightarrow{\lambda \gamma \lambda \gamma} L'$ denote reduction from $G$ to $G'$ with $\gamma$, from $L$ to $L'$ with $\lambda$, and from $L$ to $L'$ with $\lambda$, and $\lambda$, together (synchronously); we omit the label and/or the destination of a reduction if it does not matter. Formally, $\xrightarrow{r}$ is the relation induced by the rules in Figure 3b, while $\xrightarrow{\gamma}$ is its complement (not derivable).

- Rule $[\rightarrow \text{G-Com}]$ 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 $[\rightarrow \text{G-Rec}]$ states that a recursive protocol can reduce when its body can.

- Rule $[\rightarrow \text{L-At}]$ states that a projection can reduce when the global type can.

- Rule $[\rightarrow \text{L-Rev}]$ states that a $\tau$-reduction into a non-terminated branch can be reversed: after “doing nothing” (the $\tau$-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:

$$G = a \rightarrow b : \{\text{Nat} \rightarrow c : \text{Nat}, \vee, \text{Bool} \rightarrow c : \text{Bool}\}$$

The following graph visualises the operational semantics of $G$ and $G|c$.
Dashed edges represent reductions induced by rule $\rightarrow L$-Rev.

Without the $\tau$-reductions of rule $\rightarrow L$-Rev, for instance, Carol can commit to the receive of a number by internally “doing nothing” ($\tau$-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 $\tau$-reductions of rule $\rightarrow L$-Rev, in contrast, Carol cannot commit: after the $\tau$-reduction leftwards, there is still a sequence of $\tau$-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). $||\{G^i | G \rightarrow \cdots \rightarrow G^i\}| \in \mathbb{N}$ (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 $\alpha$ over global actions and local actions (Definition 12):

- Let $S \Rightarrow S^i$ denote $\tau$-reachability from $S$ to $S^i$: either $S = S^i$, or $S \overset{\tau}{\rightarrow} \cdots \overset{\tau}{\rightarrow} S^i$.

- Let $S \Downarrow$ denote weak termination of $S$: $S \Rightarrow S^\downarrow$, for some $S^\downarrow$.

- Let $S \Rightarrow S^\epsilon$ denote weak reduction from $S$ to $S^\epsilon$ with $\alpha$: either $\alpha = \tau$ and $S \Rightarrow S^\epsilon$, or $S \Rightarrow S^\epsilon \overset{\alpha}{\rightarrow} S^{\epsilon'} \Rightarrow S^{\epsilon''}$, for some $S^{\epsilon'}$, $S^{\epsilon''}$.

As a notational convention, we use “$\downarrow$” to indicate destinations after 1 reduction, while we use “$\Rightarrow$”, “$\rightarrow\rightarrow$”, “$\Rightarrow\Rightarrow$”, and “$\Rightarrow\Rightarrow\Rightarrow$” 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: $a \rightarrow b: \text{Unit}, c \rightarrow d: \text{Unit} \checkmark$ (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 cannot 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 $G_1 = a \rightarrow b: \{\text{Nat, } b \rightarrow c: \text{Nat}, \text{Unit}\}$, then:

The global type cannot be stuck after weak reduction $\text{abNat} \rightarrow$: it can always reduce onwards. In contrast, the family of projections can be stuck after weak reduction $\text{abNat} \rightarrow$, namely when $G_1 \upharpoonright c$ weakly reduced rightwards instead of leftwards. In that case, $G_1 \upharpoonright b$ neither can terminate, nor can reduce onwards (i.e., it needs to synchronise its !-reduction with a ?-reduction of $G_1 \upharpoonright c$, but $G_1 \upharpoonright c$ has become unable to reciprocate). Thus, $G_1$ and $\{G_1 \upharpoonright r \mid r \in \{a, b, c\}\}$ are inequivalent. This is caught by C1: $G_1 \upharpoonright c$ 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 $G_2 = a \rightarrow b: \text{Unit, } c \rightarrow d: \text{Unit, √}$, then:

The global type can terminate only after weak reductions $\text{abUnit} \rightarrow \text{cdUnit}$. In contrast, the family of projections can terminate also after weak reductions $\text{cdUnit} \rightarrow \text{abUnit}$.
\[ G_2[\mathbf{c}] \text{ and } G_2[\mathbf{d}] \text{ begin with } \frac{\tau \rightarrow \mathbf{d}_{\text{Unit}} \rightarrow \mathbf{c}_{\text{Unit}}}{\tau \rightarrow \mathbf{c}_{\text{Unit}}}, \text{ and when } G_2[\mathbf{a}] \text{ and } G_2[\mathbf{b}] \text{ end with } \frac{\mathbf{b}_{\text{Unit}}}{\tau} \text{ and } \frac{\mathbf{a}_{\text{Unit}}}{\tau}. \text{ Thus, } G_2 \text{ and } \{G_2[r] \}_{r \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}} \text{ are inequivalent. This is caught by C2: } G_2[\mathbf{c}] \text{ does not have a } !\text{-reduction before its } \tau\text{-reduction, but it does have one after it, which violates C2 (i.e., sending is non-causal), so } G_2 \text{ is ruled out from usage.} \]

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 \( \mathbf{a} \rightarrow \mathbf{b}_{\text{Unit}}; \mathbf{c} \rightarrow \mathbf{d}_{\text{Unit}}, \) each of whose projections will be well-behaved.

**C3.** If \( G_3 = \mathbf{a} \rightarrow \mathbf{b}.\{\mathbf{Bool} \rightarrow \mathbf{c}_{\text{Unit}}. \mathbf{c} \rightarrow \mathbf{b}_{\text{Unit}}. \mathbf{Unit}\}, \) then:

\[
\begin{align*}
\mathbf{a} & \rightarrow \mathbf{b}, \mathbf{c} \rightarrow \mathbf{b}_{\text{Unit}}, \mathbf{c} \rightarrow \mathbf{b}_{\text{Unit}} \Rightarrow \\
\mathbf{b} & \rightarrow \mathbf{c}, \mathbf{c} \rightarrow \mathbf{b}_{\text{Unit}} \Rightarrow \\
\mathbf{a} & \rightarrow \mathbf{b}, \mathbf{c} \rightarrow \mathbf{b}_{\text{Unit}} \Rightarrow \\
\mathbf{b} & \rightarrow \mathbf{c}, \mathbf{c} \rightarrow \mathbf{b}_{\text{Unit}} \Rightarrow \\
\mathbf{a} & \rightarrow \mathbf{b}, \mathbf{c} \rightarrow \mathbf{b}_{\text{Unit}} \Rightarrow \\
\mathbf{b} & \rightarrow \mathbf{c}, \mathbf{c} \rightarrow \mathbf{b}_{\text{Unit}} \Rightarrow \\
\mathbf{a} & \rightarrow \mathbf{b}, \mathbf{c} \rightarrow \mathbf{b}_{\text{Unit}} \Rightarrow \\
\mathbf{b} & \rightarrow \mathbf{c}, \mathbf{c} \rightarrow \mathbf{b}_{\text{Unit}} \Rightarrow
\end{align*}
\]

The global type cannot be stuck after weak reductions \( \frac{\mathbf{a} \rightarrow \mathbf{b}}{\mathbf{a} \rightarrow \mathbf{b}, \mathbf{c} \rightarrow \mathbf{b}_{\text{Unit}}}, \) it can always reduce onwards. In contrast, the family of projections can be stuck after weak reductions \( \frac{\mathbf{b} \rightarrow \mathbf{c}}{\mathbf{b} \rightarrow \mathbf{c}, \mathbf{b} \rightarrow \mathbf{b}_{\text{Unit}}}, \) namely when \( G_1[\mathbf{c}] \) weakly reduced rightwards instead of leftwards. In that case, \( G_3[\mathbf{b}] \) neither can terminate, nor can reduce onwards (i.e., it needs to synchronise its \( \mathbf{c} ? \mathbf{Bool}\)-reduction with a \( \mathbf{b} ! \mathbf{Bool}\)-reduction of \( G_4[\mathbf{c}], \) but \( G_3[\mathbf{c}] \) has become unable to reciprocate). Thus, \( G_3 \) and \( \{G_3[r] \}_{r \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}} \) are inequivalent. This is caught by condition C3: \( G_3[\mathbf{c}] \) 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 \( \tau\)-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 mimic each other’s termination/reductions modulo \( \tau\)-reductions.

**Definition 20.** Recall that \( \mathcal{S} \) denotes the set of all global types, local types, and families of local types, ranged over by \( S \). A *weak bisimulation* \( \sqsubseteq \subseteq \mathcal{S} \times \mathcal{S} \) is a relation that satisfies the following conditions, for every \( (S_1, S_2) \in \sqsubseteq \), and for every \( S_1, S_2, S_3, \alpha \):

\[
\begin{align*}
\text{G2} & \text{ c and G2} \text{ d begin with } \frac{\tau \rightarrow \mathbf{d}_{\text{Unit}} \rightarrow \mathbf{c}_{\text{Unit}}}{\tau \rightarrow \mathbf{c}_{\text{Unit}}}, \text{ and when G2} \text{ a and G2} \text{ b end with } \frac{\mathbf{b}_{\text{Unit}}}{\tau} \text{ and } \frac{\mathbf{a}_{\text{Unit}}}{\tau}. \text{ Thus, G2 and } \{G2[r] \}_{r \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}} \text{ are inequivalent. This is caught by C2: G2} \text{ c does not have a } !\text{-reduction before its } \tau\text{-reduction, but it does have one after it, which violates C2 (i.e., sending is non-causal), so G2 is ruled out from usage.}

\[
\begin{align*}
\text{C3. If G3} & = \mathbf{a} \rightarrow \mathbf{b}.\{\mathbf{Bool} \rightarrow \mathbf{c}_{\text{Unit}}. \mathbf{c} \rightarrow \mathbf{b}_{\text{Unit}}. \mathbf{Unit}\}, \text{ then:}
\end{align*}
\]
Let $S_1 \approx S_2$ denote weak bisimilarity. Formally, $\approx$ is the largest weak bisimulation.

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 $\text{wb}(L)$ denote well-behavedness of $L$. Formally, it is the largest predicate that satisfies the following conditions, for every $L \in \text{wb}$ (and for every $L', L^1, L^2, L, p, q, t$):

- **C1.** If $L \rightarrow L^1$, then $L \approx L^1$.
  
  (If $L^1$ is $\tau$-reachable, then $L$ and $L^1$ are weak bisimilar.)

- **C2.** If $L \xrightarrow{pq^t} L^1$, then $L \xrightarrow{pq^t} \approx L^1$.
  
  (If $L$ has a weak $!$-reduction to $L^1$, then it has the same strong $!$-reduction to a weak bisimilar destination.)

- **C3.** If $L \xrightarrow{pq^t} L^1$ and $L \xrightarrow{pq^t} L^2$, then $L^1 \approx L^2$.
  
  (If $L$ has the same weak $?$-reductions to $L^1$ and $L^2$, then $L^1$ and $L^2$ are weak bisimilar.)

- **C4.** If $L \rightarrow$, then $L \not\approx$.
  
  (If $L$ can reduce, then it cannot terminate.)

- **C5.** If $L \rightarrow L'$, then $\text{wb}(L')$.
  
  (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 $G \models \{L_r\}_{r \in R}$ denote correspondence of $G$ and $\{L_r\}_{r \in R}$. Formally:

$$
G \models \{L_r\}_{r \in R} \quad \text{if} \quad \text{wb}(G|r) \text{ and } \text{wb}(L_r) \text{ and } G|r \approx L_r \text{ for every } r \in R
$$

**Theorem 23** (equivalence). If $\text{wb}(G|r)$, for every $r \in R$, then $G \approx \{G|r\}_{r \in R}$.

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 $\text{wb}(G|r)$, for every $r \in R$, then $G \models \{G|r\}_{r \in R}$.

**Lemma 25.** If $G \models \{L_r\}_{r \in R}$, 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 $\text{wb}(G|r)$. 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 $G|r$ has exactly the same structure by rules $[\downarrow \text{L-At}]$ and $[\rightarrow \text{L-At}]$, and at most linearly many extra $\tau$-transition by rule $[\rightarrow \text{L-Rev}]$, it is finite as well. Consequently, checking well-behavedness (including weak bisimilarity [1]) of $G|r$ is trivially decidable.

**Remark 27.** As an alternative to Theorem 23, of course, it is also possible to check weak bisimilarity between $G$ and $\{G|r\}_{r \in R}$ directly. However, this would require one to compute the reduction relation of $\{G|r\}_{r \in R}$, 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.
Let \( R \) denote the set of role-indexed families of processes, ranged over by \( P \).

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

\[ P := \sum\{O_1, \ldots, O_n\} \mid \sum\{I_1, \ldots, I_m\} \mid O := \overline{q}(e).P \mid I := p(x:t).P \]

if \( e \in P_1 P_2 \mid loop \ P \mid recur \mid 0 \)

Let \( \mathbb{R} \to \mathbb{P} \) denote the set of role-indexed families of processes, ranged over by \( \mathbb{P} \).

Output process \( \overline{pq}(e).P \) implements the \textit{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 \textit{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 \( \sum\{O_1, \ldots, O_n\} \) and \( \sum\{I_1, \ldots, I_m\} \) implement non-deterministic \textit{selections} of \( n \) output processes (sends) and \( m \) input processes (receives); we omit “\( \sum \)” and braces when \( n = m = 1 \). Process \( if \in P_1 P_2 \) implements a \textit{conditional choice}. Processes \( loop \ P \) and \( recur \) implement a \textit{loop}. Process \( 0 \) implements the \textit{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.

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

\textbf{Definition 30.} Let \( \Pi = \bigcup\{pq!v, pq?v \mid p \neq q\} \) denote the set of (concrete) local actions, ranged over by \( \pi \).

\section{Processes – Operational Semantics}

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

\textbf{Definition 31.} Let \( \downarrow P \) and \( \downarrow \mathbb{P} \) denote termination of \( P \) and \( \mathbb{P} \). Formally, \( \downarrow \) is the predicate induced by the rules in Figure 4a.
**Definition 32.** Let $P \xrightarrow{\pi} P'$ and $\pi \xrightarrow{\pi} \pi'$ denote reduction from $P$ to $P'$ with $\pi$ alone, and from $\mathcal{P}$ to $\mathcal{P}'$ with $\pi_p$ and $\pi_q$ together (synchronously). Formally, $\Rightarrow$ is the relation induced by the rules in Figure 4b.

Rule $[\Rightarrow-P\text{-SUM1}]$ (resp. $[\Rightarrow-P\text{-SUM2}]$) 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 eval($e$) is defined (resp. $v$ is well-typed by $t$ and bound to $x$). Rule $[\Rightarrow-P\text{-IF}]$ states that a conditional choice can reduce when eval($e$) $\in \{true, false\}$ and the corresponding branch can reduce. Rule $[\Rightarrow-P\text{-LOOP}]$ 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 $P = if P_1 P_2$, but eval($e$) $\notin \{true, false\}$, then rules $[\Rightarrow-P\text{-IF}]$ are inapplicable.
- If $P = \sum_{1 \leq i \leq n} \{P_{p_i}(e_i) : P_1, \ldots, P_{p_n}(e_n) : P_n\}$ and $n > 1$, but eval($e_i$) is undefined for every $1 \leq i \leq n$, then rule $[\Rightarrow-P\text{-SUM1}]$ 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 $[\Rightarrow-P\text{-SUM}]$ 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 $(\Xi \times \mathbb{T})^*$ denotes the set of data typing contexts, ranged over by $\Xi$. Let (\text{rec} \times \mathbb{L})^* denote the set of process typing contexts, ranged over by $\mathcal{T}$. Let $\Xi, \mathcal{T} \vdash O : L$, $\Xi, \mathcal{T} \vdash I : L$, $\Xi, \mathcal{T} \vdash P : L$ and $\mathcal{P} : \mathcal{L}$ denote well-typedness of $O$, $I$, $P$ by $L$ in $\Xi, \mathcal{T}$, and of $\mathcal{P}$ by $\mathcal{L}$. Formally, $\Rightarrow$ is the relation induced by the rules in Figure 5.
Rule $\vdash \text{-End}$ states that the empty process is well-typed when the local type can weakly terminate. Rule $\vdash \text{-If}$ states that a conditional choice is well-typed when the condition and the branches are well-typed. Rules $\vdash \text{-Loop}$/$\vdash \text{-Recur}$ state that a loop is well-typed when the body and the recursive calls are well-typed. Rule $\vdash -$ states that a family of processes is well-typed when every process is well-typed by a well-behaved local type.

Rule $\vdash \text{-Out}$ 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 $\vdash \text{-In}$ states that an input process is well-typed when the local type has an analogous weak $?$-transition, and the continuation is well-typed. Rule $\vdash \text{-Sum1}$ 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 $\vdash \text{-Sum2}$ 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 $G = a \rightarrow b::\{\text{Nat.b} \rightarrow \text{c:Nat.✓}, \text{Bool.b} \rightarrow \text{c:Bool.✓}\}$; the operational semantics of this global type was previously visualised in Example 15.

- Alice: Process $\bar{5}(\bar{5}).0$, process $\bar{5}(\text{true}).0$, and process if $\text{cond()}\{\bar{5}(\bar{5}).0\} (\bar{5}(\text{true}).0)$ are all well-typed by $G[a]$, because rule $\vdash \text{-Sum1}$ requires only one send specified to be implemented. Process $\bar{5}(\text{"foo"}).0$ is ill-typed, because rule $\vdash \text{-Sum1}$ requires every send implemented to be specified. Process loop $\bar{5}($rec$)$ is ill-typed by $G[a]$ as well, because rule $\vdash \text{-Loop}$ adds recur : $G[a]$ to the process typing context at the root of the derivation tree, but rule $\vdash \text{-Recur}$ requires recur : $✓$ at the leaf, and $G[a] \not\approx ✓$.

- Carol: Process $b(x:\text{Nat}).0$ and process if $\text{cond()}\{b(x:\text{Nat}).0\} (b(x:\text{Bool}).0)$ are both ill-typed by $G[c]$, because rule $\vdash \text{-Sum2}$ requires every receive specified to be implemented. Process $\sum(b(x:\text{Nat}).0, b(x:\text{Bool}).0)$ is well-typed by $G[c]$.

The asymmetry between the right-sided premises of rules $\vdash \text{-Sum1}/\vdash \text{-Sum2}$ 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 $G \models L$ and $\vdash P : L$, then $P \Downarrow$ or $P \rightarrow$.

Complementary, the symmetry between the left-sided premises of rules $\vdash \text{-Sum1}/\vdash \text{-Sum2}$ (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 $G \models L$ and $\vdash P : L$, then (for every $P', p, q, v$):

- If $P \Downarrow$, then $G \Downarrow$ and $L \Downarrow$.
- If $P \overset{p,q,t}{\rightarrow} P'$, then $G' \equiv L'$ and $\vdash P' : L'$ and $\vdash v : t$ and $G \overset{p,q,t}{\rightarrow} G'$ and $L \overset{p,q,t}{\rightarrow} L'$, for some $t, G', L'$.

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 $\vdash \mathcal{P} : \{G \mid r \in R\}$ and $\mathcal{P} \xrightarrow{p \rightarrow q \cdot t} \cdots \xrightarrow{p_n \rightarrow q_n \cdot t_n} \mathcal{P}'$, then:

$$\{G \mid r \in R\} \xrightarrow{p \rightarrow q \cdot t} \cdots \xrightarrow{p_n \rightarrow q_n \cdot t_n}, \text{ and } \vdash v_1 : t_1, \text{ and } \cdots, \text{ and } \vdash v_n : t_n, \text{ for some } t_1, \ldots, t_n.$$

1. If $\mathcal{P}' \downarrow$, then $\mathcal{L}^\downarrow \downarrow$.
2. If $\mathcal{P}' \uparrow$ and $\mathcal{P}' \downarrow$, then $\mathcal{L}^\uparrow \downarrow$ and $\mathcal{L}^\downarrow \downarrow$.

Remark 40. Theorem 39 depends on premise $\vdash \mathcal{P} : \{G \mid r \in R\}$. To see that checking this premise is decidable, observe that we need to check two sets of properties by rule $\vdash$:\n
1. well-typedness of the processes in $\mathcal{P}$ by the local types in $\{G \mid r \in R\}$;
2. well-behavedness of the local types in $\{G \mid r \in R\}$. 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 $\vdash$ 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 $\vdash \mathcal{P} : \{G \mid r \in R\}$ and $\mathcal{P} \xrightarrow{p \rightarrow q \cdot t} \cdots \xrightarrow{p_n \rightarrow q_n \cdot t_n} \mathcal{P}'$, then:

- Safety: $\mathcal{P} \xrightarrow{p \rightarrow q \cdot t} \cdots \xrightarrow{p_n \rightarrow q_n \cdot t_n}$, and $\vdash v_1 : t_1$, and $\cdots$, and $\vdash v_n : t_n$, for some $t_1, \ldots, t_n$.
- Liveness: $\mathcal{P}' \downarrow$, or $\mathcal{P}' \rightarrow$.

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

Definition 42 (revision of Definition 11). $G ::= p \rightarrow q \cdot t \mid G_1 + G_2 \mid G_1 \cdot G_2 \mid G^* \mid \emptyset$.

Global type $p \rightarrow q \cdot t$ 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 $\emptyset$ specifies the empty protocol.

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

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

Process $\sum\{O_1, \ldots, O_n, I_1, \ldots, I_m\}$ 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 48. \( G ::= \cdots \mid G_1 ; G_2 \mid G_1 \parallel G_2 \mid G_1 \otimes G_2 \mid [G]_{\gamma_1} \gamma_2 \)  

- 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.
- 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 \( G_1 \otimes G_2 \) 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 \( G_1 \otimes G_2 \) 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 \( \gamma_1 \) over low-priority \( \gamma_2 \) in \( G \).
\[ \Xi, T \vdash \sum_{i=1}^{n} \{O_1, \ldots, O_n\} : L \quad \Xi, T \vdash \sum_{i=1}^{m} \{I_1, \ldots, I_m\} : L \]

**Figure 8** Well-typedness.

\[ (G_1 \downarrow, G_2 \downarrow) \quad (G_1, G_2 \downarrow) \quad G_1 \uparrow \quad G_2 \uparrow \]

**Example 49.** To exemplify join and prioritisation, let \( G_1 = a \rightarrow b \rightarrow c \) and \( G_2 = a \rightarrow b \rightarrow d \) (data types omitted). Unconstrained are \( b \rightarrow c \) (only in \( G_1 \)) and \( b \rightarrow d \) (only in \( G_2 \)); constrained is \( a \rightarrow b \) (both in \( G_1 \) and in \( G_2 \)). Thus, \( G_1 \parallel G_2 \) is equivalent to \( a \rightarrow b : (b \rightarrow c :: b \rightarrow d) \); after the constrained communication, the unconstrained communications are interleaved. Thus, \( [G_1 \parallel G_2]_{\parallel}^{\parallel} \) is equivalent to \( ]a \rightarrow b : (b \rightarrow c :: b \rightarrow d)\] which is equivalent to \( a \rightarrow b : b \rightarrow c : b \rightarrow d \); after \( a \rightarrow b \) (no priority), \( b \rightarrow c \) (high priority) must precede \( b \rightarrow d \) (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 \( \gamma \)-reduce when \( G_1 \) (resp. \( G_2 \)) can, now, but \( G_2 \) (resp. \( G_1 \)) cannot, ever (i.e., \( \gamma \) is unconstrained). The third reduction rule states that it can \( \gamma \)-reduce when \( G_1 \) and \( G_2 \) can (i.e., \( \gamma \) is constrained).
- The first reduction rule for prioritisation states that it can \( \gamma_{1} \)-reduce (high priority) when \( G \) can. The second reduction rule states that it can \( \gamma_{2} \)-reduce (low priority) when it cannot \( \gamma_{1} \)-reduce. The third reduction rule states that it can \( \gamma \)-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 \( \sum \{ p \rightarrow q : t_i \cdot G_i \}_{1 \leq i \leq n} \): (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 worst case – that specifies all roles together; last, the product automaton is checked for satisfaction of a special temporal logic formula \( \phi \), 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 \( \phi \)) 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 \( \sum \{ p \rightarrow q_i : t_i \cdot G_i \}_{1 \leq i \leq n} \), 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 \( \sum \{ p \rightarrow q_i : t_i \cdot G_i \}_{1 \leq i \leq n} \). 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 limited 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


42:28 Sound, Regular Multiparty Sessions via Implicit Local Types

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.


Sound, Regular Multiparty Sessions via Implicit Local Types

