An Automata-theoretic Basis for Specification and Type Checking of Multiparty Protocols

Felix Stutz[1,2](✉) and
Emanuele D'Osualdo[1,3]

[1] MPI-SWS, Saarbrücken, Germany
felix.stutz@uni.lu
[2] University of Luxembourg, Esch-sur-Alzette, Luxembourg
emanuele.dosualdo@uni-konstanz.de
[3] University of Konstanz, Konstanz, Germany

Abstract. We propose the Automata-based Multiparty Protocols framework (AMP) for top-down protocol development. The framework features a new very general formalism for global protocol specifications called Protocol State Machines (PSMs), Communicating State Machines (CSMs) as specifications for local participants, and a type system to check a π-calculus with session interleaving and delegation against the CSM specification. Moreover, we define a large class of PSMs, called "tame", for which we provide a sound and complete PSPACE projection operation that computes a CSM describing the same protocol as a given PSM if one exists. We propose these components as a backwards-compatible new backend for frameworks in the style of Multiparty Session Types. In comparison to the latter, AMP offers a considerable improvement in expressivity, decoupling of the various components (e.g. projection and typing), and robustness (thanks to the complete projection).

Keywords: Communication protocols · Verification · Multiparty session types · Communicating state machines · Type checking

1 Introduction

Designing correct distributed communication protocols is an important and hard problem. Consider a finite set of protocol *participants* (i.e. independent processes) whose only means of interaction between each other is asynchronous message passing through reliable FIFO channels. The goal is to program each participant so that some global emergent behaviour is achieved, e.g. a leader is elected. Unfortunately, even when each participant is finite-state, the presence of unbounded delays (i.e. unbounded communication channels) makes any non-trivial property of the emergent global behaviour undecidable [10].

The top-down protocol design approach proposes to work around this issue by a reversal in the methodology: instead of first programming the participants and then checking that their global behaviour is what we desired it to be, we first specify the desired global behaviour and then synthesize each participant's local

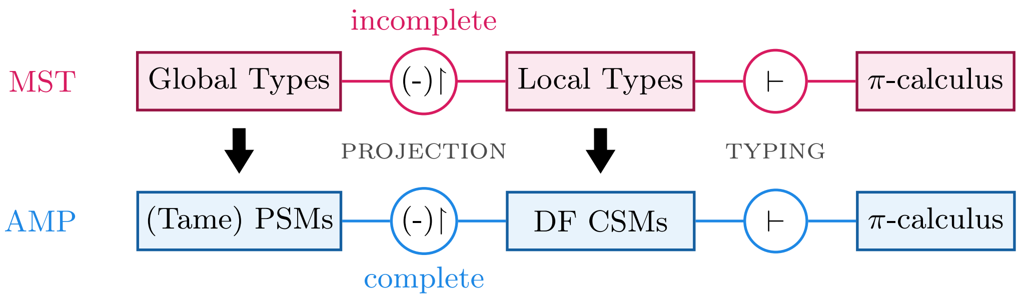


Fig. 1: The components of top-down frameworks.

specification so that local behaviour gives rise to the correct global behaviour by construction. Each participant's concrete implementation is then checked against its local specification, which (a) can be achieved by static means like type systems, and (b) makes the verification of the implementation local and modular.

Multiparty Session Types (MSTs) [37] is one of the most prominent and extensively studied formalisms supporting this top-down design methodology. The key components of the framework, depicted in Fig. 1, are: (1) **Global Types:** a dedicated language to specify correct global behaviour; (2) **Local Types:** a dedicated language to specify each participant's actions in the protocol; (3) **Programs:** a programming language (typically a π-calculus) to express concrete implementations of each participant of the protocol.

Imagine, as a simple example, we want to specify a centralised leader election protocol, where an arbiter $\mathtt{a}$ selects a leader among $\mathtt{p}$ and $\mathtt{q}$ and the selected participant communicates the win to the other. A possible global type representing the protocol is $G = (\mathtt{a}{\to}\mathtt{p}{:}\mathsf{sel}\,.\,\mathtt{p}{\to}\mathtt{q}{:}\mathsf{win}) + (\mathtt{a}{\to}\mathtt{q}{:}\mathsf{sel}\,.\,\mathtt{q}{\to}\mathtt{p}{:}\mathsf{win})$ where $\mathtt{a}{\to}\mathtt{q}{:}\mathsf{sel}$ says that $\mathtt{a}$ sends sel and $\mathtt{q}$ receives it and $+$ denotes branching. The local type of $\mathtt{p}$ would then be $(\mathtt{p}{\lhd}\mathtt{a}?\mathsf{sel}\,.\,\mathtt{p}{\rhd}\mathtt{q}!\mathsf{win}) + \mathtt{p}{\lhd}\mathtt{q}?\mathsf{win}$ where $\mathtt{p}{\lhd}\mathtt{a}?\mathsf{sel}$ means "$\mathtt{p}$ receives message sel from $\mathtt{a}$" and $\mathtt{q}\lhd\mathtt{p}?\mathsf{win}$ means "$\mathtt{p}$ sends message win to $\mathtt{q}$". Thus, $\mathtt{p}$ is supposed to listen for a message from $\mathtt{a}$ or from $\mathtt{q}$; in the former case it would then communicate the win to $\mathtt{q}$, in the latter, just concede. A program implementation may consist of a process for each participant; the process for the arbiter $\mathtt{a}$ may implement any specific policy for selecting the leader (e.g. always choose $\mathtt{p}$), as long as the communications follow the protocol.

The relationship between the three representations of the protocol, i.e. global types, local types, and programs, is delicate. First, the global type and the local types should give rise to the same behaviour; however it is not always possible to capture the behaviour of a global type with local types. Suppose, for instance, that we modified the leader election protocol G to $G' = (\mathtt{a}{\to}\mathtt{p}{:}\mathsf{sel}\,.\,\mathtt{q}{\to}\mathtt{p}{:}\mathsf{lose}) + (\mathtt{a}{\to}\mathtt{q}{:}\mathsf{sel}\,.\,\mathtt{p}{\to}\mathtt{q}{:}\mathsf{lose})$. While, from a global perspective, it is possible to insist on the losing participant informing the winner that they lost, locally, the losing participant has no way to determine whether they won or not. Therefore G' is not realisable by local processes: we say it is not projectable. Second, local types are a more abstract representation of the system than programs, but we still want to show that, when implementation details are omitted, a program only performs communications that adhere to the local specification.

In MSTs, the relationship between the three layers of the framework are enforced through two procedures: (i) **Projection**, which (when possible) extracts, from a global type G, some local types that are guaranteed to behave as described by G; and (ii) **Type Checking**, which checks that the program implementation of each participant adheres to the behaviour specified in its local type.
In a perfect world, a framework for top-down protocol design should be:

1. **Expressive:** It should support as many protocols as possible.
2. **Decoupled:** Its components (global/local specifications, programs, projection, type checking) should depend on each other as little as possible, and be specified independently of their algorithmic implementation, to allow for reuse and modularity.
3. **Robust:** It should only reject a global specification if there is a genuine issue with it (i.e. no false positives).

Unfortunately, the MST frameworks in the literature leave something to be desired against this ideal picture. They all suffer from:

– **Expressivity Limitations:** Many legitimate protocols are rejected either because the global specification syntax is too restricted, or because the projection algorithm cannot handle them. For example, every MST framework we are aware of can only handle global types with *directed choice*, i.e. where every branching point involves exactly one sender and one receiver. This immediately rules out our example leader election protocol G because the branches involve different receivers.
– **Decoupling Limitations:** In MSTs, the syntax of global types directly influences the definition of the projection algorithm and the syntax of the local types, which in turn influence the type system design. Typically, changing one of the framework's components requires adapting (and reproving correctness of) all the others. Furthermore, many MST frameworks solely give the intended relation between global and local types through the projection algorithm and do not give a declarative definition.
– **Robustness Limitations:** The heuristic nature of the projection algorithms makes it very hard to predict if a global type will be handled or not by an MST framework, even in the case where the behaviour specified by the global type is unproblematic.

In this paper, we propose a new foundation for top-down protocol design machinery, dubbed AMP (Automata-based Multiparty Protocols), that achieves the expressivity, decoupling and robustness goals.

Expressivity of AMP. Figure 1 shows the components of AMP. To achieve expressivity, we propose a new general formalism for (finite-control) global protocol specifications, which we call *Protocol State Machines* (PSMs). The formalism is based on automata which are given semantics in terms of (finite and infinite) sets of words, over an alphabet of send ($\mathtt{p}\triangleright\mathtt{q}!m$) and receive ($\mathtt{q}\triangleleft\mathtt{p}?m$) actions. PSMs remove many of the restrictions of global types, while retaining their character: they specify the expected behaviour from a global perspective, and satisfy some basic correctness properties by construction (e.g. every send is eventually received, no type mismatches, etc). Owing to their generality, PSMs can represent any global type, but can go well beyond them: they also strictly subsume High-Level Message Sequence Charts (HMSCs). For maximizing expressivity at the local level, we adopt Communicating State Machines (CSMs) as the formalism for local protocol specifications. They are a canonical representation for decentralised asynchronous communication and as such do not impose constraints over what can be represented. Finally, to maximise expressivity at the program level, we consider a π-calculus with session interleaving and delegation.

Decoupling of AMP. In AMP, decoupling is achieved through its handling of projection and type checking. For projection, the framework merely specifies the semantic requirements that a correct projection algorithm needs to satisfy: essentially that it produces a deadlock-free CSM which represents the same language as the input PSM. This limits the impact of projection on global and local specifications, and leaves open any algorithmic/manual strategy to achieve the projection goal. (We discuss how AMP proposes to actually implement projection in the discussion of robustness). For example, scenarios in which the user provides both the PSM and the CSM and a proof that they represent the same language and the CSM satisfies desirable properties (like deadlock freedom) or where an algorithm infers a PSM from a CSM, are both compatible with our framework thanks to this declarative approach to projection. This treatment is in line with some MST works where the fundamental property of projection is expressed in terms of some behavioural equivalence between local and global types. For type checking, decoupling is achieved by defining the type system by depending exclusively on programs and CSMs. The standard guarantees of subject reduction, communication safety and session fidelity are proven by appealing to properties of CSMs. This demonstrates how effective CSMs are in providing a clean decoupled interface between projection and type checking.

Robustness of AMP. Finally, we demonstrate how robustness can be achieved in AMP, by identifying a large class of PSMs, called *Tame PSMs*, for which we provide a decidable, sound and complete projection operation. Tame PSMs extend the reach of sound and complete projection beyond global types and can handle a large class of HMSCs as well as protocols that cannot be expressed as either global types nor HMSCs. The main constraint that makes a PSM tame is what we call *sender-driven choice*: that at any branching point, the sender in all the branches is the same participant and takes distinct actions in the branches. Our projection algorithm builds on a recently proposed complete projection for sender-driven global types [48]. Thanks to a surprising reduction, we manage to extend the algorithm to tame PSMs while keeping the complexity in PSPACE. Due to the fact that our projection operation is complete, only protocols that do not admit any valid projection will be rejected: those are protocols which simply cannot be implemented by local processes. We also show that our class is in a sense "maximally robust": lifting the sender-driven restriction makes projection undecidable, even for global types. AMP is also robust in the sense that one can select the desired guarantees of the type system and check whether they can be enforced by checking (syntactic) properties of the global protocol, pinpointing exactly which guarantee is provided by a PSM. Finally, we show that the framework is backwards-compatible with MSTs: not only can we encode global types into PSMs and project them, we also pinpoint the (simple) conditions under which our projection yields CSMs which are equivalent to local types.

Contributions. In summary:

- We propose PSMs as an expressive general formalism for (finite-control) global protocol specifications.
- We propose CSMs as a canonical model for local protocol specifications and specify their desired relationship with PSMs declaratively.
- We define the first session type system based on CSMs, pinpointing exactly the properties of the CSM that are needed to provide each of the desired guarantees; these properties can be enforced by construction by ensuring the PSMs conform to some simple checks.
- We define Tame PSMs (encompassing all directed and sender-driven global types) and give a sound and complete projection algorithm for them.
- We show that sender-driven choice is a necessary restriction even for global types: projection is undecidable otherwise.
- We characterise which class of PSMs corresponds to global types, and which CSMs correspond to local types, giving us full backward-compatibility with standard MST theory.

We think of AMP as a backend for top-down protocol design tools with the following workflow. Any specific tool, e.g. Scribble [71], provides a dedicated syntax for types and processes. Then, a global specification is compiled to a PSM (where the compiler guarantees its tameness, which would be trivial for global types) and invokes the projection of AMP, producing a CSM. This could be re-translated for user consumption, but also be used to drive typing using AMP. Failure of projection can be directly translated by the frontend to an explanation of why the protocol is flawed and must be repaired. Given the generality of PSMs, it should also be easier to experiment with extensions of the frontend language.

All proofs, omitted details, and additional examples can be found in [62].

2 Motivation and Key Ideas

In this section, we give an informal overview of the key ideas behind AMP before proceeding with the formal development from Section 3.

2.1 Global Specifications via Protocol State Machines

Our first goal is to define an expressive formalism for specifying global protocols, that is also constrained enough to make it tractable for top-down protocol development. One of the most accomplished such formalisms, used in MSTs, is *global types*. Figure 2 shows an example of a global type, represented in Figure 3 as an HMSC.

The term $\mathtt{p}{\to}\mathtt{q}{:}m_1$ indicates the transmission of message m_1 from $\mathtt{p}$ to $\mathtt{q}$. The symbol $\mathbf{0}$ denotes termination of the protocol. Recursion can be specified by binding a recursion variable X with μX and using X subsequently to jump back to where X was bound. Branching is denoted by $+$. In the example, $\mathtt{p}$ can pick between three branches by sending different messages to $\mathtt{q}$. Subsequently, $\mathtt{q}$ sends messages to $\mathtt{r}$ in all branches: 1 in the top and middle branch and 3 in the bottom branch. Participant $\mathtt{r}$ is supposed to send messages v_1 or v_2 in the top and middle branch while it receives from $\mathtt{p}$ in the bottom branch, which also recurses using X.

What makes these formalisms tractable? Their first key characteristic is that, as a specification tool, they allow the user to (a) adopt a global point of view, describing what coordination between all the participants is induced by the protocol; (b) express this coordination without enumerating all possible interleavings of the send and receive events that can happen due to the asynchronous nature of communication, e.g. $\mathtt{p}{\to}\mathtt{q}{:}m_1$ indicates the send of the message immediately followed by its receipt, although in any asynchronous implementation, the receipt might happen at a much later point, after other independent events took place. In Fig. 2, $\mathtt{r}$ may lag behind arbitrarily while $\mathtt{p}$ and $\mathtt{q}$ keep sending messages. The second key characteristic of global types and HMSCs is that they are *finite-control*: their control structure can be described using a finite graph. This makes it possible to algorithmically manipulate them, e.g. for verifying they satisfy some desirable properties, or for extracting local protocol specifications.

Our aim is to distil these two characterising features and remove any other restriction that is not necessary, to obtain a more expressive global specification formalism. To do this, we take a language-theoretic view of protocols, where a protocol is seen as the set of sequences of send and receive events that are considered compliant with it. More precisely, a send event $\mathtt{p}\triangleright\mathtt{q}!m$ records that $\mathtt{p}$ sent the message m to $\mathtt{q}$; a receive event $\mathtt{q}\triangleleft\mathtt{p}?m$ records that $\mathtt{q}$ received message m from $\mathtt{p}$. A protocol specification is the language of desired finite or infinite words of events. For the purpose of this section, we will focus on finite words, but the technical development considers both finite and infinite words.

Not all languages over these events are meaningful in the context of protocols. First, the sequences of events might not be feasible when using FIFO channels (e.g. $\mathtt{p}\triangleright\mathtt{q}!1\cdot\mathtt{q}\triangleleft\mathtt{p}?2$ is not FIFO); we write FIFO for the language of all words that satisfy FIFO order. Second, if $\mathtt{p}\triangleright\mathtt{q}!m_1\cdot\mathtt{r}\triangleright\mathtt{q}!m_2\cdot\mathtt{q}\triangleleft\mathtt{p}?m_1\cdot\mathtt{q}\triangleleft\mathtt{r}?m_2$ is accepted by a procotol, it ought to also accept $\mathtt{r}\triangleright\mathtt{q}!m_2\cdot\mathtt{p}\triangleright\mathtt{q}!m_1\cdot\mathtt{q}\triangleleft\mathtt{p}?m_1\cdot\mathtt{q}\triangleleft\mathtt{r}?m_2$ as this kind of reorderings are induced by the scheduling of participants and network delays which are out of the control of participants. We write $\mathcal{C}(L)$ for the closure of the language L under such reorderings. Thus a language $L\subseteq\mathsf{FIFO}$ represents the global interaction patterns of the protocols; moreover L can specify only some

$$\mu X.+\begin{cases}\mathtt{p}{\to}\mathtt{q}{:}m_1\,.\,\mathtt{q}{\to}\mathtt{r}{:}1\,.\,\mathtt{r}{\to}\mathtt{p}{:}v_1\,.\,\mathbf{0}\\ \mathtt{p}{\to}\mathtt{q}{:}m_2\,.\,\mathtt{q}{\to}\mathtt{r}{:}1\,.\,\mathtt{r}{\to}\mathtt{p}{:}v_2\,.\,\mathbf{0}\\ \mathtt{p}{\to}\mathtt{q}{:}m_3\,.\,\mathtt{q}{\to}\mathtt{r}{:}3\,.\,\mathtt{p}{\to}\mathtt{r}{:}v_3\,.\,X\end{cases}$$

Fig. 2: Example global type.

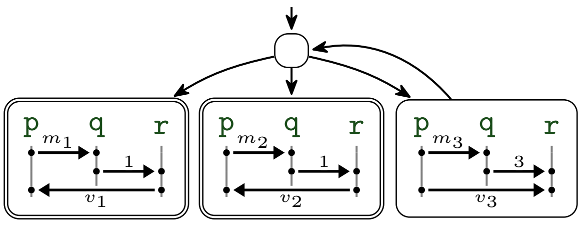
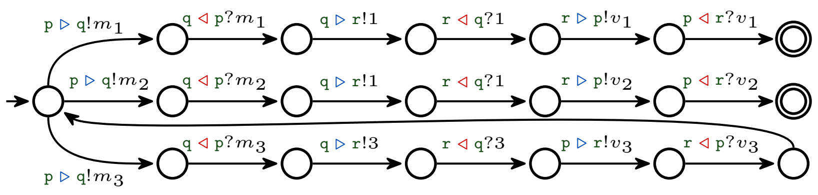


Fig. 3: A protocol as an HMSC.



$\mathtt{p} \triangleright \mathtt{q}!m_1$
$\mathtt{q} \triangleleft \mathtt{p}?m_1$
$\mathtt{q} \triangleright \mathtt{r}!1$
$\mathtt{r} \triangleleft \mathtt{q}?1$
$\mathtt{r} \triangleright \mathtt{p}!v_1$
$\mathtt{p} \triangleleft \mathtt{r}?v_1$
$\mathtt{p} \triangleright \mathtt{q}!m_2$
$\mathtt{q} \triangleleft \mathtt{p}?m_2$
$\mathtt{q} \triangleright \mathtt{r}!1$
$\mathtt{r} \triangleleft \mathtt{q}?1$
$\mathtt{r} \triangleright \mathtt{p}!v_2$
$\mathtt{p} \triangleleft \mathtt{r}?v_2$
$\mathtt{q} \triangleleft \mathtt{p}?m_3$
$\mathtt{q} \triangleright \mathtt{r}!3$
$\mathtt{r} \triangleleft \mathtt{q}?3$
$\mathtt{p} \triangleright \mathtt{r}!v_3$
$\mathtt{r} \triangleleft \mathtt{p}?v_3$
$\mathtt{p} \triangleright \mathtt{q}!m_3$

Fig. 4: A PSM encoding for the protocol of Fig. 2.

of these interactions and get all the ones that should also be possible under the asynchronous semantics by declaring the full set of acceptable words to be $\mathcal{C}(L)$.

Now, to obtain a finite-control formalism, we propose to express such a "core" language L for the protocol $\mathcal{C}(L)$ using a finite state machine M with $\mathcal{L}(M) = L$. Since FIFO is not regular, the only feasible way of ensuring $\mathcal{L}(M) \subseteq \mathsf{FIFO}$ is by requiring M to keep track of which sent messages are still pending, which a finite-state machine can only do up to some maximum capacity for the send buffers. We thus arrive at the requirement that $\mathcal{L}(M) \subseteq \mathsf{FIFO}_B$, where FIFO_B is the set of words respecting FIFO but where the number of pending sends never exceed $B \in \mathbb{N}$ at any point in time. Note that FIFO_B is regular.

Building on these observations, we define a *Protocol State Machine* (PSM) to be a finite state machine M recognising words of send and receive events, with $\mathcal{L}(M) \subseteq \mathsf{FIFO}_B$ for some $B \in \mathbb{N}$. Fig. 4 shows the protocol of Fig. 2 as a PSM. Interpreted as a mere automaton M, it recognises a language $\mathcal{L}(M)$ of words with at most one pending send at all time. (We call M Σ1-PSM because the total number of messages in flight is at most 1; if we allowed 1 message *per channel*, it would be called a 1-PSM.) As a PSM, however, M denotes the language $\mathcal{C}(\mathcal{L}(M))$, which admits words with unbounded channel behaviours and is not even regular in general. For instance, $\mathcal{C}(\mathcal{L}(M))$ includes words starting with $(\mathtt{p} \triangleright \mathtt{q}!m_3 \cdot \mathtt{q} \triangleleft \mathtt{p}?m_3 \cdot \mathtt{q} \triangleright \mathtt{r}!3 \cdot \mathtt{p} \triangleright \mathtt{r}!v_3)^n \cdot (\mathtt{r} \triangleleft \mathtt{q}?3 \cdot \mathtt{r} \triangleleft \mathtt{p}?v_3)^n \dots$ where $\mathtt{r}$ is running at a lower rate than the other participants, and leaves n pending sends from $\mathtt{p}$ and from $\mathtt{q}$ before it consumes them.

PSMs achieve a substantial gain in expressivity while retaining the key characteristics of global types. In terms of expressivity, every global type can be encoded as a Σ1-PSM; furthermore PSMs can be used to encode HMSCs, which strictly subsume global types because the latter cannot specify simultaneous message exchanges between a pair of participants [63]. PSMs can even represent protocols that are outside the reach of HMSCs. Consider, for example, the PSM in Fig. 5. In that protocol, $\mathtt{p}$ commits to some integer (abstracted as the label int) at the beginning by sending it to $\mathtt{r}$ and sends a go signal to $\mathtt{q}$. Note that here we use the paired send and receive notation $\mathtt{p} \!\to\! \mathtt{q}{:}\mathsf{ok}$ to emit the two events in sequence. Then $\mathtt{q}$ and $\mathtt{r}$ engage in some negotiation of arbitrary length until $\mathtt{q}$ decides to exit the loop, at which point $\mathtt{r}$ is finally allowed to receive the message sent by $\mathtt{p}$. No HMSC can represent such protocol: the matching events $\mathtt{p} \triangleright \mathtt{r}!\mathsf{int}$ and $\mathtt{r} \triangleleft \mathtt{p}?\mathsf{int}$ are separated by an arbitrary number of events (with no opportunity for reordering up to $\mathcal{C}$(-)); since matching events in HMSCs need to belong to the same basic block, such block would also need to contain the arbitrarily many events in between, which is impossible.

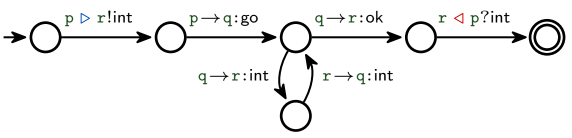


Fig. 5: A protocol not expressible as an HMSC. Transitions labelled with $\mathtt{p}{\to}\mathtt{q}{:}m$ should be interpreted as emitting the sequence $\mathtt{p} \triangleright \mathtt{q}!m \cdot \mathtt{q} \triangleleft \mathtt{p}?m$.

Of course, this level of generality would be pointless if we were not able to provide for it in the other components of top-down protocol design. We start by studying the first crucial component: projection.

2.2 From Global to Local Specifications: Projection

When considering projection, our first concern is the goal of decoupling: we want to define a general interface for projection, such that both different algorithmic implementations of projection can be used without altering the design of the rest of the framework; and such that typing is not dependent on global specifications (nor projection details).

In AMP, the key to decoupling is in choosing *Communicating State Machines* (CSMs) as the formalism for local specifications. A CSM $\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ associates a finite state automaton $A_\mathtt{p}$ to each participant $\mathtt{p} \in \mathcal{P}$, where transitions can either send or receive on the channels of $\mathtt{p}$; the semantics of $\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ is defined on configurations that include the local states for each participant and an (unbounded) FIFO buffer for each channel. They induce a FIFO language $\mathcal{L}(\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}})$ over send/receive events, by considering as final the configurations where all the participants are in final local states and all the buffers are empty. CSMs thus represent a canonical general model of finite-control asynchronous protocol implementations.

Per se, this is not a particularly original choice: MST's local types have been linked to CSMs of a certain shape before [23, 60], and HMSC-based work used them as local specifications. What AMP demonstrates is that it is possible to build the entire top-down methodology around CSMs (with fewer restrictions), including a session type system, gaining both in expressivity and in decoupling.

Having fixed our model for local behaviour, we can ask when it defines behaviour consistent with a global specification. We say a CSM $\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ *is a projection of* a PSM M if $\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ is deadlock-free and $\mathcal{L}(\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}) = \mathcal{C}(\mathcal{L}(M))$.

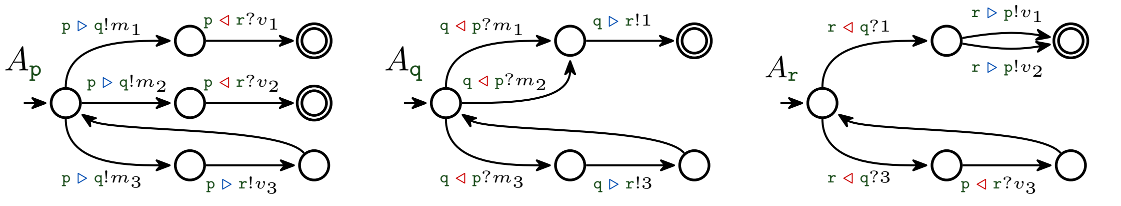


Fig. 6: Example CSM.

We focus on the *(projection) synthesis problem*, producing a CSM as result. The corresponding decision problem is the *projectability problem*, which simply asks if there exists such a CSM. Notably, projectability can have lower complexity.

Even for simple protocols, projection can be tricky. Take the example of Fig. 2: $\mathtt{r}$ can never distinguish between the top two branches, as its only observations would be to have received 1 from $\mathtt{q}$. The instance of the protocol with $m_1 \neq m_2$ and $v_1 \neq v_2$ would thus not be projectable. If $m_2 = m_3$ then $\mathtt{q}$ would not be able to send the appropriate message to $\mathtt{r}$. Therefore, the only projectable instances with no redundant branches are the ones where m_1, m_2, and m_3 are pair-wise distinct and $v_1 = v_2$. Figure 6 shows a candidate projection of the PSM in Fig. 2. If $m_2 = m_3$ or $v_1 \neq v_2$, the PSM is not projectable, and in fact the CSM can reach a deadlock.

Given CSMs are Turing-complete models, it is unsurprising that checking if a given CSM is a projection of a given PSM is undecidable. The key advantage of the top-down approach boils down to the fact that it is nevertheless often possible to efficiently *compute* a valid projection from a global specification. This is precisely the goal of the *projection* operation. A projection operation $(\text{-})\!\restriction$ is a function taking a PSMs as input and returning either $\bot$ or a CSM; it is a *sound projection* if for all PSM M, if $M\!\restriction = \{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ then $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ is a projection of M; it is a *complete projection* if for all projectable M, $M\!\restriction \neq \bot$.

The MST literature proposed a number of sound but incomplete projection algorithms for global types. Incompleteness makes MST frameworks lack robustness: a projectable global type might still be rejected by the framework because the projection is unable to handle it; this leaves the user in the awkward position of having to build a mental model for the projection algorithm to be able to design viable global types. Li et al. [48] proposed the first sound and complete projection algorithm for sender-driven global types. Its PSPACE complexity stems from the need for determinisation. Their evaluation, though, showed that these corner cases will likely not occur in reality. This provides initial evidence that robustness is achievable without compromising efficiency.

As is to be expected, the jump in expressivity by adopting PSMs cannot come for free: the problem of computing a sound and complete projection for PSMs is in general undecidable, a fact inherited from being able to encode HMSCs. This does not defeat us, however: one of our main positive results is the definition of a very large class of PSMs, called *Tame PSMs*, that enjoys sound and complete PSPACE projection. A PSM is *tame* if it satisfies three constraints: (a) a technical refinement of the notion of the bound B for buffers, (b) that final states have no outgoing transitions, and (c) *sender-driven choice*: at each branching point, there is a single sender taking distinct actions.

Our proof works by reducing the problem to an instance of projectability of MSTs with sender-driven choice, which was proven to be decidable in PSPACE [48]. Our reduction is surprising because it produces a transformed protocol which is different from the original one: the encoded protocol language is different and involves additional participants and additional message exchanges; and yet its synthesized local specifications can be transformed back to local specifications for the original protocol. Due to the mismatch in expressivity between PSMs and global types, it is necessary that the reduction modifies the protocol semantics. Furthermore, we show the reduction preserves the complexity class, giving us a PSPACE algorithm for projectability of sender-driven PSMs.

Despite being a restriction, Tame PSMs are still much more general than global types: every sender-driven global type gives rise to a Tame PSM; moreover, every example given so far is tame. While the first two constraints (a) and (b) are not severe, the third condition (c) imposes a genuine restriction on expressivity.

In fact our main negative result is that sender-driven choice is in a sense “minimal”: we prove that projectability is undecidable for global types (the most primitive kind of PSMs) with general choice (aka “mixed choice”).

2.3 Processes and Typing

To complete the top-down toolkit, we provide a mean to check that a program correctly implements a protocol specified as a CSM. We achieve this by defining a CSM-based session type system for an expressive variant of π-calculus with session interleaving and delegation. The process calculus is adapted from [59] which represents a feature-rich modern presentation of multiparty session typing.

The type system’s main soundness argument hinges, as is standard, on a subject reduction result: if a typable program can take a step, it remains typable. From this, we derive two main safety correctness guarantees: typable programs cannot produce type mismatches (i.e. receiving a message that the process is not expecting) and terminated sessions do not leave orphan messages behind. We further prove a progress property under standard restrictions: roughly speaking, if the process contains only one session, then, if the type of the session is not final, the process can take a step (among the ones allowed by the type). Global progress in the presence of session interleaving is out of scope of this paper, but it may be attainable by adapting the (orthogonal) analysis employed in [18, 44].

In line with our decoupling goal, the guarantees of the type system are derived from the key properties of CSMs produced by projection (e.g. deadlock freedom). This makes it even compatible with the bottom-up methodology of [59] which forgoes global types and proposes to check key properties on local types directly. If a CSM satisfying the desired properties is provided to our type system, the corresponding guarantees apply to typable processes regardless of the existence of a PSM representing the protocol. This also liberates the type system completely from the choice of representation for global protocols.

Overall we obtain an expressive, decoupled and robust backend for top-down protocol development. Finally, we also show that this backend is backwards-compatible with MSTs: not only every sender-driven global type can be encoded as Tame PSM, but we also prove that, when there exists a local type that is a projection of a global type, our projection produces a CSM that can be translated back to a local type. This shows under which conditions PSMs and global types as well as CSMs and local types are equivalent, despite their structural differences.

3 Automata-based Protocol Specifications

We start our technical development by introducing a language-theoretic view of protocol specifications. We define protocols as special languages of words, and use CSMs as our local specifications of such languages. Finally, we introduce PSMs as global protocol specifications.

3.1 State Machines and Protocol Languages

Let Δ be a finite alphabet. The set of finite words over Δ is denoted by Δ^*, the set of infinite words by Δ^ω, and their union by Δ^∞. We write ε for the empty word. For two strings $u \in \Delta^*$ and $v \in \Delta^\infty$, their concatenation is $u \cdot w$, and we say that u is a *prefix* of v, written $u \leq v$, if there is some $w \in \Delta^\infty$ such that $u \cdot w = v$; $\mathrm{pref}(v)$ denotes all prefixes of v and is lifted to languages as expected. For a language $L \subseteq \Delta^\infty$, we distinguish between the language of finite words $L_{\text{fin}} := L \cap \Delta^*$ and the language of infinite words $L_{\text{inf}} := L \cap \Delta^\omega$.

Definition 3.1 (State machines). *A* state machine $A = (Q, \Delta, \delta, q_0, F)$ *is a* 5*-tuple with a finite set of states* Q*, an alphabet* Δ*, a transition relation* $\delta \subseteq Q \times (\Delta \cup \{\varepsilon\}) \times Q$*, an initial state* $q_0 \in Q$ *from the set of states, and a set of final states* F *with* $F \subseteq Q$*. If* $(q, a, q') \in \delta$*, we also write* $q \xrightarrow{a} q'$*. A* run *is a finite or infinite sequence* $q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \ldots$*, with* $q_i \in Q$ *and* $a_i \in \Delta \cup \{\varepsilon\}$ *for* $i \geq 0$*, such that* q_0 *is the initial state, and for each* $i \geq 0$*, it holds that* $(q_i, a_i, q_{i+1}) \in \delta$*. The* trace *of such run is the word* $a_0 \cdot a_1 \cdot \ldots \in \Delta^\infty$*. A run is* maximal *if it ends in a final state or is infinite. The* (core) language $\mathcal{L}(A)$ *of* A *is the set of traces of all maximal runs. If* Q *is finite, we say* A *is a* finite state machine *(FSM). A state machine is* dense *if for every* $q \xrightarrow{x} q' \in \delta$*, the transition label* x *is* ε *implies that* q *has only one outgoing transition. A state machine is* deterministic *if* $\forall (q, a, q') \in \delta.\, a \neq \varepsilon$ *and* $\forall (q, a, q'), (q, a, q'') \in \delta.\, q' = q''$*. We call a dense state machine* deterministic *if* $\forall (q, a, q'), (q, a, q'') \in \delta.\, q' = q''$*. A state* $q \in Q$ *is called a* sink state *if it has no outgoing transitions, i.e.* $\forall a \in \Delta \cup \{\varepsilon\}, q' \in Q.\, (q, a, q') \notin \delta$*. We say a state machine is* sink-final *if, for every state, it is final iff it is a sink.*

A language-theoretic view of protocols. Let $m \in \mathcal{V}$ be a finite set of messages and $\mathtt{p}, \mathtt{q}, \ldots \in \mathcal{P}$ be a finite set of participants. The alphabet of $\mathtt{p}$'s send and receive events is the set $\Gamma_{\mathtt{p}} := \bigcup_{\mathtt{q} \in \mathcal{P}, m \in \mathcal{V}} \{\mathtt{p} \triangleright \mathtt{q}!m,\ \mathtt{p} \triangleleft \mathtt{q}?m\}$. A send event $\mathtt{p} \triangleright \mathtt{q}!m$ records that $\mathtt{p}$ sent the message m to $\mathtt{q}$; a receive event $\mathtt{p} \triangleleft \mathtt{q}?m$ records that $\mathtt{p}$ received message m from $\mathtt{q}$. The alphabet of all events is the set $\Gamma_{\mathcal{P}} := \bigcup_{\mathtt{p} \in \mathcal{P}} \Gamma_{\mathtt{p}}$. A paired event is a send event and its corresponding receive event: $\mathtt{p} \to \mathtt{q} : m := \mathtt{p} \triangleright \mathtt{q}!m \cdot \mathtt{q} \triangleleft \mathtt{p}?m$. We define the alphabet of paired events as $\Sigma_{\mathcal{P}} := \{\mathtt{p} \to \mathtt{q} : m \mid \mathtt{p}, \mathtt{q} \in \mathcal{P} \text{ and } m \in \mathcal{V}\}$. For the remainder of the paper, we fix an arbitrary set of participants $\mathcal{P}$ and messages $\mathcal{V}$, and often write Γ for $\Gamma_{\mathcal{P}}$ and Σ for $\Sigma_{\mathcal{P}}$. Given a word, we can project it to all letters of a certain shape: for instance, $w \Downarrow_{\mathtt{p} \triangleright \mathtt{q}!\_}$ is the subword of w with all of its send events where $\mathtt{p}$ sends any message to $\mathtt{q}$. We write $\mathcal{V}(w)$ for the sequence of values in w (in the same order). In $w = w_1 \ldots \in \Gamma^\infty$, a send event $w_i = \mathtt{p} \triangleright \mathtt{q}!m$ is *matched* by a receive event $w_j = \mathtt{q} \triangleleft \mathtt{p}?m$ if $i < j$

and $\mathcal{V}((w_1 \dots w_i){\Downarrow}_{\mathtt{p} \triangleright \mathtt{q}!}) = \mathcal{V}((w_1 \dots w_j){\Downarrow}_{\mathtt{q} \triangleleft \mathtt{p}?})$. A send event w_i is *unmatched* if there is no such receive event w_j. A language $L \subseteq \Gamma^\infty$ satisfies *feasible eventual reception* if for every finite word $w := w_1 \dots w_n \in \text{pref}(L)$ with an unmatched send event w_i, there is an extension $w \leq w' \in L$ such that w_i is matched in w'.

A sequence of send and receive events shall describe the execution of a protocol. We define when such a sequence uses channels in FIFO manner.

Definition 3.2 (FIFO Language). *A word $w \in \Gamma^\infty$ is FIFO-compliant if for each prefix w' of w, it holds that $\mathcal{V}(w'{\Downarrow}_{\mathtt{q} \triangleleft \mathtt{p}?\_})$ is a prefix of $\mathcal{V}(w'{\Downarrow}_{\mathtt{p} \triangleright \mathtt{q}!\_})$, for every $\mathtt{p}, \mathtt{q} \in \mathcal{P}$. We denote the set of all infinite FIFO-compliant words by $\mathsf{FIFO}_{\text{inf}}$. For finite words, we require that all send events are matched. Thus, $\mathsf{FIFO}_{\text{fin}} := \{w \mid w \text{ is FIFO-compliant and } \mathcal{V}(w{\Downarrow}_{\mathtt{p} \triangleright \mathtt{q}!\_}) = \mathcal{V}(w{\Downarrow}_{\mathtt{q} \triangleleft \mathtt{p}?\_})\ \forall \mathtt{p}, \mathtt{q} \in \mathcal{P}\}$. We denote the (non-regular) set of all FIFO words by $\mathsf{FIFO} = \mathsf{FIFO}_{\text{inf}} \uplus \mathsf{FIFO}_{\text{fin}}$. A language $L \subseteq \mathsf{FIFO}$ is a called a* FIFO language.

As the model of distributed implementation of a protocol, we adopt communicating state machines: parallel compositions of finite-control processes communicating asynchronously via point-to-point FIFO channels.

Definition 3.3 (Communicating state machines). *We call $\mathcal{A} = \{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p} \in \mathcal{P}}$ a* communicating state machine *(CSM) over $\mathcal{P}$ and $\mathcal{V}$ if $A_\mathtt{p}$ is a finite state machine with alphabet $\Gamma_\mathtt{p}$ for every $\mathtt{p} \in \mathcal{P}$. The semantics of a CSM $\mathcal{A}$ is the language $\mathcal{L}(\mathcal{A}) \subseteq \mathsf{FIFO}$ whose definition is standard (see [62]). Roughly, for each pair of distinct participants $\mathtt{p}, \mathtt{q} \in \mathcal{P}$ there are two FIFO channels $\langle \mathtt{p}, \mathtt{q} \rangle, \langle \mathtt{q}, \mathtt{p} \rangle \in$ Chan allowing communication between the participants in the two directions. The FSM $A_\mathtt{p} = (Q_\mathtt{p}, \Gamma_\mathtt{p}, \delta_\mathtt{p}, q_{0,\mathtt{p}}, F_\mathtt{p})$ describes the possible actions of participant $\mathtt{p}$. A transition $(q_\mathtt{p}, \mathtt{p} \triangleright \mathtt{q}!m, q'_\mathtt{p}) \in \delta_\mathtt{p}$ indicates that when $\mathtt{p}$ takes a step from $q_\mathtt{p}$ to $q'_\mathtt{p}$, it will send a message m to $\mathtt{q}$ by enqueuing it in channel $\langle \mathtt{p}, \mathtt{q} \rangle$. Similarly, $(q_\mathtt{p}, \mathtt{p} \triangleleft \mathtt{q}?m, q'_\mathtt{p}) \in \delta_\mathtt{p}$ prescribes the reception by $\mathtt{p}$ of message m from the channel $\langle \mathtt{q}, \mathtt{p} \rangle$. A CSM's run always starts with empty channels and each participant running its respective initial state. We denote the set of all reachable configurations (from the initial configuration) by* reach$(\mathcal{A})$. *A* deadlock *of $\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p} \in \mathcal{P}}$ is a reachable configuration with no outgoing transition that has at least one non-empty channel or at least one participant not in a (local) final state.*

The formal definition is given in [62]. As an example, Fig. 6 shows the three state machines constituting a CSM.

The goal of a protocol designer is to define a protocol that can be realised as a CSM. The *projectable* languages are exactly those protocols which can.

Definition 3.4 (Projections and Projectability). *A language $L \subseteq \Gamma^\infty$ is said to be* projectable *if there exists a* deadlock-free *CSM $\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p} \in \mathcal{P}}$ such that it generates the same language* (protocol fidelity), *i.e., $L = \mathcal{L}(\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p} \in \mathcal{P}})$. We say that $\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p} \in \mathcal{P}}$ is a projection of L.*

The asynchronous nature of CSMs makes them unable to enforce the order between certain events without explicit synchronisation. For instance, any CSM producing a word $\mathtt{p}\triangleright\mathtt{q}!m\cdot\mathtt{r}\triangleright\mathtt{s}!m'\cdot w$ will necessarily produce also $\mathtt{r}\triangleright\mathtt{s}!m'\cdot\mathtt{p}\triangleright\mathtt{q}!m\cdot w$. Which events can be reordered is context-dependent: the events in the word $\mathtt{p}\triangleright\mathtt{q}!m\cdot\mathtt{q}\triangleleft\mathtt{p}?m$ cannot be swapped, as the receive is only possible after the send. But in $\mathtt{p}\triangleright\mathtt{q}!m\cdot\mathtt{p}\triangleright\mathtt{q}!m\cdot\mathtt{q}\triangleleft\mathtt{p}?m$ the last two events can be reordered. This has been formalised as equivalence relation by Majumdar et al. [51], which can be seen as an instance of Lamport's happens-before relation [46] for systems with point-to-point FIFO channels.

Definition 3.5. *The* indistinguishability relation $\sim\subseteq\Gamma^*\times\Gamma^*$ *is the smallest equivalence relation such that*

(1) *If* $\mathtt{p}\neq\mathtt{r}$, *then* $w\cdot\mathtt{p}\triangleright\mathtt{q}!m\cdot\mathtt{r}\triangleright\mathtt{s}!m'\cdot u\ \sim\ w\cdot\mathtt{r}\triangleright\mathtt{s}!m'\cdot\mathtt{p}\triangleright\mathtt{q}!m\cdot u$.
(2) *If* $\mathtt{q}\neq\mathtt{s}$, *then* $w\cdot\mathtt{q}\triangleleft\mathtt{p}?m\cdot\mathtt{s}\triangleleft\mathtt{r}?m'\cdot u\ \sim\ w\cdot\mathtt{s}\triangleleft\mathtt{r}?m'\cdot\mathtt{q}\triangleleft\mathtt{p}?m\cdot u$.
(3) *If* $\mathtt{p}\neq\mathtt{s}\wedge(\mathtt{p}\neq\mathtt{r}\vee\mathtt{q}\neq\mathtt{s})$, *then* $w\cdot\mathtt{p}\triangleright\mathtt{q}!m\cdot\mathtt{s}\triangleleft\mathtt{r}?m'\cdot u\ \sim\ w\cdot\mathtt{s}\triangleleft\mathtt{r}?m'\cdot\mathtt{p}\triangleright\mathtt{q}!m\cdot u$.
(4) *If* $|w{\Downarrow}_{\mathtt{p}\triangleright\mathtt{q}!}|>|w{\Downarrow}_{\mathtt{q}\triangleleft\mathtt{p}?}|$, *then* $w\cdot\mathtt{p}\triangleright\mathtt{q}!m\cdot\mathtt{q}\triangleleft\mathtt{p}?m'\cdot u\ \sim\ w\cdot\mathtt{q}\triangleleft\mathtt{p}?m'\cdot\mathtt{p}\triangleright\mathtt{q}!m\cdot u$.

We define $u\preceq_\sim v$ *if there is* $w\in\Gamma^*$ *such that* $u\cdot w\sim v$. *Observe that* $u\sim v$ *iff* $u\preceq_\sim v$ *and* $v\preceq_\sim u$. *For infinite words* $u,v\in\Gamma^\omega$, *we define* $u\preceq^\omega_\sim v$ *if for each finite prefix* u' *of* u, *there is a finite prefix* v' *of* v *such that* $u'\preceq_\sim v'$. *Define* $u\sim v$ *iff* $u\preceq^\omega_\sim v$ *and* $v\preceq^\omega_\sim u$. *We lift the equivalence relation* $\sim$ *on words to languages. For a language* L, *we define*

$$\mathcal{C}(L)=\left\{w'\;\middle|\;\bigvee\begin{array}{l}w'\in\Gamma^*\wedge\exists w\in\Gamma^*.\ w\in L\text{ and }w'\sim w\\ w'\in\Gamma^\omega\wedge\exists w\in\Gamma^\omega.\ w\in L\text{ and }w'\preceq^\omega_\sim w\end{array}\right\}.$$

Lemma 3.6 ([51]). *For any CSM* $\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}$, $\mathcal{L}(\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}})=\mathcal{C}(\mathcal{L}(\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}))$.

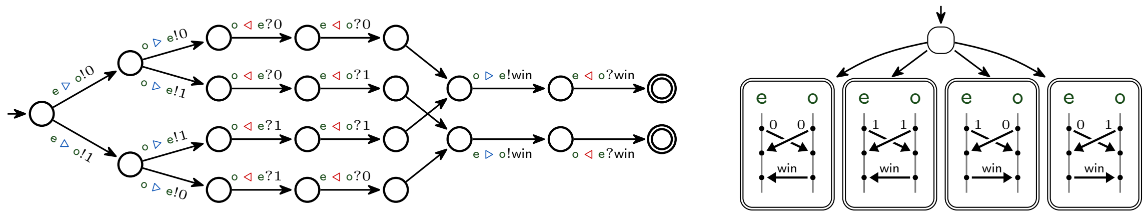
Example 3.7. For finite words $\mathcal{C}(\text{-})$ is standard. For infinite words, though, the situation is a bit counterintuitive. Let us consider $w:=(\mathtt{p}\triangleright\mathtt{q}!m\cdot\mathtt{q}\triangleleft\mathtt{p}?m)^\omega$. It is easy to construct a CSM $\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}$, with FSMs $A_\mathtt{p}$ and $A_\mathtt{q}$, that accepts w. CSMs do not promise any sort of fairness for infinite runs so there is an infinite run for $(\mathtt{p}\triangleright\mathtt{q}!m)^\omega$ where only $A_\mathtt{p}$'s transitions are scheduled. This is why $\mathcal{C}(\text{-})$ is defined using $\preceq^\omega_\sim$, giving $(\mathtt{p}\triangleright\mathtt{q}!m)^\omega\in\mathcal{C}((\mathtt{p}\triangleright\mathtt{q}!m\cdot\mathtt{q}\triangleleft\mathtt{p}?m)^\omega)$.

3.2 Protocol State Machines

We now introduce PSMs as a mean to specify protocol languages from a global, centralised perspective. The idea, shared with both global types and HMSCs, is to specify only a core subset of the admissible executions, e.g. the ones where there is a bounded delay between sends and matching receives, and obtain the full set of admissible executions by closing the core language using $\mathcal{C}(\text{-})$.

We adapt the notion of B-bounded from [27] to formalise the idea of "bounded delay" between matching events.

Definition 3.8 (B-bounded and ΣB-bounded). *Let* $B\in\mathbb{N}$ *be a natural number. A FIFO-compliant word* w *is* B-bounded, *resp.* ΣB-bounded, *if for every prefix* w' *of* w *and participants* $\mathtt{p},\mathtt{q}\in\mathcal{P}$, *it holds that* $|w'{\Downarrow}_{\mathtt{p}\triangleright\mathtt{q}!}|-|w'{\Downarrow}_{\mathtt{q}\triangleleft\mathtt{p}?}|\leq B$, *resp.* $\sum_{\mathtt{p}\neq\mathtt{q}\in\mathcal{P}}(|w'{\Downarrow}_{\mathtt{p}\triangleright\mathtt{q}!}|-|w'{\Downarrow}_{\mathtt{q}\triangleleft\mathtt{p}?}|)\leq B$. *We define the (regular) set of* B*-bounded FIFO words:* $\mathsf{FIFO}_B:=\{w\in\mathsf{FIFO}\mid w\text{ is }B\text{-bounded}\}$.



(a) KLE as a PSM. (b) KLE as an HMSC.

Fig. 7: Kindergarten Leader Election (KLE).

Definition 3.9 (Protocol State Machine). *A dense FSM $M = (Q, \Gamma, \delta, q_0, F)$ is a B-PSM if $\mathcal{L}(M) \subseteq \mathsf{FIFO}_B$ and $\mathcal{L}(M)$ satisfies feasible eventual reception. The semantics of M defined as $\mathcal{S}(M) := \mathcal{C}(\mathcal{L}(M))$. Moreover, M is a PSM if it is a B-PSM for some B.*

By definition, PSMs specify FIFO languages; importantly, although the core language $\mathcal{L}(M)$ is B-bounded, the semantics $\mathcal{C}(\mathcal{L}(M))$ includes non-B-bounded words and will not even be regular in general. Note that, it is decidable to check if an FSM is a B-PSM.

In [62], we show that $\mathcal{C}(\text{-})$ preserves and reflects feasible eventual reception: if $L \subseteq \Gamma^\infty$ satisfies feasible eventual reception, then $\mathcal{C}(L)$ does, and if $\mathcal{C}(L)$ satisfies feasible eventual reception, then L does. More generally, every property that is preserved by $\mathcal{C}(\text{-})$ can be soundly checked on the core language of a PSM. If the property is also reflected by $\mathcal{C}(\text{-})$, the property holds if and only if it holds for the core language.

Definition 3.10. *A PSM M is a $\Sigma 1$-PSM if its core language $\mathcal{L}(M)$ is $\Sigma 1$-bounded. We may abuse notation and use $\Sigma_\mathcal{P}$ as alphabet for $\Sigma 1$-PSMs.*

Example 3.11 (Kindergarten Leader Election). We consider a protocol between two participants e (evens) and o (odds). It can be used to quickly settle a dispute between children (hence the name). Both children pick 0 or 1 and tell each other their pick at the same time. Child e wins if the sum is even while o wins if the sum is odd. At the end, the loser concedes by sending the message win to the winner. The protocol is specified as a PSM in Fig. 7a (and as an HMSC in Fig. 7b). Note that specifying this protocol requires the ability of issuing send and receive events independently. If one insisted on issuing send and matching receives together, as in global types and Σ1-PSMs, one of the children would be forced to reveal their hand first, undermining the purpose of the protocol.

4 Projection: From PSMs to CSMs

A CSM $\mathcal{A}$ is a projection of a PSM M, if $\mathcal{A}$ is a projection of $\mathcal{S}(M)$. In this section, we explain two main results. The first is positive: we show that sound and complete projection is decidable for Tame PSMs. The second is negative: we show that the sender-driven restriction of Tame PSMs is necessary: if we drop the restriction, projectability becomes undecidable even for sink-final Σ1-PSMs. The full proofs can be found in [62].

4.1 Sound and Complete Projection for Tame PSMs

The idea of the decidability result is to reduce projectability of a Tame PSM to projectability of a (different) sender-driven global type, which can then be handled using the sound and complete algorithm of [48]. Furthermore, the reduction is such that a projection of the original PSM can be read off a projection of the global type. Before sketching the idea behind the reduction, we define Tame PSMs formally. Tame PSMs satisfy three conditions: they are sink-final, sender-driven, and satisfy some more fine-grained bounds on the message queues.

Definition 4.1 (Choice restrictions for PSMs). *Let $M = (Q, \Gamma, \delta, q_0, F)$ be a PSM. The PSM M satisfies* sender-driven choice *if there is a function $\lambda\colon Q \to \mathcal{P}$ such that for all states q, q' such that $q \xrightarrow{x} q'$ with $x \in \Gamma_!$, it holds that $\lambda(q)$ is the sender for x, i.e., $x = \lambda(q) \triangleright \_!\_$. In addition, we say M is* directed *if for every state q, there is also a dedicated receiver p, i.e., all transition labels from q are of the form $\lambda(q) \triangleright \mathsf{p}!\_$. Last, if there is no dedicated sender but all transitions are still distinct, i.e. M is deterministic, we say that it satisfies* mixed choice.

Definition 4.2 (Channel bounds for PSMs). *We define* channel bounds *as a partial function $\beta\colon \mathsf{Chan} \rightharpoonup \mathbb{N}$ from channels to natural numbers, where $\operatorname{dom}(\beta)$ denotes the domain of β. Given a PSM M, we say that M respects β if the following holds for every $\langle \mathsf{p}, \mathsf{q} \rangle \in \mathsf{Chan}$:*

- *If $\langle \mathsf{p}, \mathsf{q} \rangle \notin \operatorname{dom}(\beta)$, then every message from p to q is immediately followed by a receive: for every state q and transition from q to q' labelled with $\mathsf{p} \triangleright \mathsf{q}!m$, it holds that there is a single transition from q' and it is labelled with $\mathsf{q} \triangleleft \mathsf{p}?m$.*
- *If $\langle \mathsf{p}, \mathsf{q} \rangle \in \operatorname{dom}(\beta)$, then $w \Downarrow_{\mathsf{p} \triangleright \mathsf{q}!, \mathsf{q} \triangleleft \mathsf{p}?}$ is $\beta(\langle \mathsf{p}, \mathsf{q} \rangle)$-bounded for every $w \in \mathcal{S}(M)$.*

A PSM that respects β with $\beta = \emptyset$ is a PSM which only uses paired events, just like global types do. Thus checking the condition with $\beta = \emptyset$ is a trivial syntactic check. For general PSMs, it is possible to generate valid channel bounds with a sound algorithm we propose in [62]. We conjecture the algorithm to be also complete, i.e. to always output some bounds if they exist.

Definition 4.3 (Tame PSMs). *A* Tame *PSM is a pair (M, β) where the PSM M is sender-driven, sink-final, and respects the channel bounds β.*

We can now sketch the idea behind the reduction. Fundamentally, the gap in expressivity between Tame PSMs and sender-driven global types is that in PSMs sends and matching receives do not need to appear one right after the other. One can observe, however, that one could replicate the same asynchrony of some trace $\mathsf{p} \triangleright \mathsf{q}!m \cdots \mathsf{q} \triangleleft \mathsf{p}?m$ by introducing an intermediary participant (p, q) that is always ready to forward messages from p to q, leading to a trace $\mathsf{p} \to (\mathsf{p}, \mathsf{q}) : m \cdots (\mathsf{p}, \mathsf{q}) \to \mathsf{q} : m$ where the sends and matching receives between participants and the intermediaries are now immediately adjacent. The channel bounds β tell us exactly for which channels we need to introduce intermediaries; moreover the bound on the buffers induced by β makes sure that these intermediaries will not introduce any spurious dependency in the executions. To consolidate the idea, we show how it applies to our KLE example.

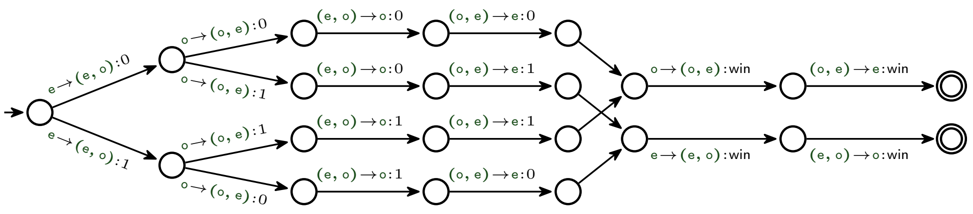


Fig. 8: Kindergarten Leader Election after the Channel-participant Encoding.

Example 4.4 (Revisiting the KLE protocol). In Example 3.11, we introduced the Kindergarten Leader Election protocol, whose communication pattern cannot be represented as a Σ1-PSM/global type: both children need to commit to the number they send before they receive the other's message. Its PSM (Fig. 7) is however tame: it is sink-final, sender-driven and respects $\beta(\texttt{e},\texttt{o}) = \beta(\texttt{o},\texttt{e}) = 1$. The "intermediary forwarders" idea applied to the protocol results in a protocol where some teachers (the intermediaries) will act as depositories for the initial choices of the two children. After committing their choice, each child is allowed to learn from the teacher the choice of the other child. The resulting PSM is given in Fig. 8. The names of the additional participants indicate the direction of communication: $(\texttt{e},\texttt{o})$ forwards messages from $\texttt{e}$ to $\texttt{o}$. Obviously, this encoding does not specify the same protocol. Still, our construction shows that one can obtain a projection of the original protocol from a projection of the modified one, by appropriately removing the forwarding actions of the teachers.

The example illustrates the simple case where $\beta(\text{-}) \leq 1$; in the more general case, the reduction is more involved and requires more intermediaries.

The workflow of our encoding is visualised in Fig. 9. Given a PSM M, one first computes its encoding $\texttt{enc}_{\mathrm{PSM}}(M)$. Second, one synthesizes a projection $\{\!\{A_{\texttt{p}}\}\!\}_{\texttt{p}\in\mathcal{P}}$ of the encoded protocol using results from [48]. Third, one decodes to obtain a projection $\{\!\{\texttt{dec}_{\mathrm{FSM}}(A_{\texttt{p}})\}\!\}_{\texttt{p}\in\mathcal{P}}$ of M.

$$\begin{tikzcd}
M \arrow[r, "① \ \texttt{enc}_{\mathrm{PSM}}(\text{-})"] & \texttt{enc}_{\mathrm{PSM}}(M) \arrow[d, bend left, "② \ \text{synthesis}"] \\
\{\!\{\texttt{dec}_{\mathrm{FSM}}(A_{\texttt{p}})\}\!\}_{\texttt{p}\in\mathcal{P}} \arrow[u, dashed, "\text{implements}"] & \{\!\{A_{\texttt{p}}\}\!\}_{\texttt{p}\in\mathcal{P}} \arrow[l, "③ \ \texttt{dec}_{\mathrm{FSM}}(\text{-})"'] \arrow[u, dashed, "\text{implements}"]
\end{tikzcd}$$

Fig. 9: Workflow of encoding.

Theorem 4.5. *Checking projectability of Tame PSMs is in PSPACE. One can also synthesize a projection in PSPACE.*

4.2 Mixed Choice yields Undecidable Projectability

Now, we show that the sender-driven choice restriction for Tame PSMs is necessary for projectability to be decidable. General PSMs inherit undecidability of projectability from HMSCs, which in turn was proven by Lohrey [50, Thm. 3.4]. Given our positive result for Tame PSMs, the proof for undecidability ought to break in the presence of sender-driven choice. The original proof goes through several (often implicitly given or omitted) automata-based transformations and does not give any insights about where and how the transformations break under the assumption of sender-driven choice.

Theorem 4.6. *The projectability problem for sink-final mixed-choice $\Sigma 1$-PSMs is undecidable.*

We reduce the membership problem for Turing Machines to checking projectability of a sink-final mixed-choice $\Sigma 1$-PSM with five participants. Initially, there is a branching which only two participants are involved in and learn about. Subsequently, all participants communicate Turing machine computations in the form of configurations in both branches. If the (projected) language of one of the other participants is not the same for both branches, the PSM cannot be implementable because they do not know which branch to comply with and easily deadlock. We also show that the reverse is the case. Hence, we specify a language for each branch and make both coincide if and only if the Turing Machine has no accepting computation, which is the case if and only if the PSM is projectable.

The full proof is in [62]. We adopt the proof strategy of Lohrey to PSMs and make every transformation explicit and carefully check which structural properties the transformations preserve, yielding a stronger undecidability result concerning the most rudimentary of PSMs: $\Sigma 1$-PSMs.

5 Typing Programs against CSMs

We now overview the key ideas behind AMP's type system. The formal details and full proofs can be found in [62]. To define programs, we take inspiration from the process calculus with session interleaving and delegation of [59]. The syntax of AMP's programs is reproduced in Fig. 10. The processes P represent the static program text. As is standard, $\mathbf{0}$ is the terminated process, $\parallel$ denotes parallel composition, $\mathsf{Q}[\vec{c}]$ denotes a sequential process running the code defined by a finite set of definitions $\mathcal{D}$. The prefixes $\oplus_{i\in I}\, c[\mathtt{q}_i]!l_i\langle c_i\rangle$ and $\&_{i\in I}\, c[\mathtt{q}_i]?l_i(y_i)$ denote internal and external choice respectively, with a non-empty finite set of indices I. The endpoint of participant $\mathtt{p}$ of a channel between $\mathtt{p}$ and $\mathtt{q}$ in a session s, is denoted by $s[\mathtt{p}][\mathtt{q}]$; $\mathtt{p}$ can send a label l and some payload p to $\mathtt{q}$ in session s by $s[\mathtt{p}][\mathtt{q}]!l\langle p\rangle$, the dual reception is denoted by $s[\mathtt{q}][\mathtt{p}]?l\langle x\rangle$ (which binds the payload to x). To model delegation, a process must be able to send to another the capability to act as participant $\mathtt{p}$ in session s, denoted $s[\mathtt{p}]$; the receiving process will bind such capability to a variable x and use it to form endpoints $x[\mathtt{q}]$; we thus have in general send/receive actions on $c[\mathtt{q}]$ where c can be a variable or some $s[\mathtt{p}]$.

The process $(\nu s : \mathcal{A})\, P$ denotes the creation of a new bound session s used in P. The session is annotated with a (computationally irrelevant) CSM $\mathcal{A}$, taking the place of what is often a global type. So far, we treated messages in CSMs very abstractly as elements of a finite alphabet. In processes, messages are more structured: they have a label (from a finite set) and a payload (of some type). The messages used by the CSM will thus be pairs $l(t)$ of a label l and a payload type t, with the convention that if, from a state q, there are two outgoing transitions with the same sender, receiver and label, they will agree on the type.

In applications, the payload can be of any base type (e.g. integers, strings), or be a channel capability $s[\mathtt{p}]$ (for delegation). Since supporting base types is a

$$
\begin{aligned}
c &::= x \mid s[\mathtt{p}] \\
P &::= \mathbf{0} \mid P_1 \parallel P_2 \mid (\nu s : \mathcal{A})\, P \mid \bigoplus_{i\in I} c[\mathtt{q}_i]!l_i\langle c_i\rangle \,.\, P_i \mid \mathop{\&}_{i\in I} c[\mathtt{q}_i]?l_i(y_i)\,.\,P_i \mid \mathsf{Q}[\vec{c}] \\
R &::= \mathbf{0} \mid R_1 \parallel R_2 \mid (\nu s : \mathcal{A})\, R \mid \bigoplus_{i\in I} c[\mathtt{q}_i]!l_i\langle c_i\rangle \,.\, P_i \mid \mathop{\&}_{i\in I} c[\mathtt{q}_i]?l_i(y_i)\,.\,P_i \mid \mathsf{Q}[\vec{c}] \\
&\quad\mid s \blacktriangleright \sigma \mid \mathbf{err} \\
\mathcal{D} &::= \big(\mathsf{Q}[\vec{x}] = \bigoplus_{i\in I} c[\mathtt{q}_i]!l_i\langle c_i\rangle\,.\,P_i\big);\ \mathcal{D} \mid \big(\mathsf{Q}[\vec{x}] = \mathop{\&}_{i\in I} c[\mathtt{q}_i]?l_i(y_i)\,.\,P_i\big);\ \mathcal{D} \mid \varepsilon
\end{aligned}
$$

Fig. 10: Syntax of AMP's π-calculus.

simple exercise, we follow [59] and focus on the harder case of channel capabilities as payloads. When using a CSM $\mathcal{A} = \{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ as a protocol specification for a session s, it is natural to consider the (control) states $Q_\mathtt{p}$ of $A_\mathtt{p}$ to be the local types that can be associated to $s[\mathtt{p}]$. Therefore, in our setting we will consider the set L of the states of any $\mathcal{A}$ annotating the process, as the possible payload types. For simplicity, we assume all CSMs use disjoint sets of states, so that we can unambiguously refer to the transitions from any state q by $\delta(q)$.

In particular, if the protocol specified by $\mathcal{A}$ can delegate channels of a session following some CSM $\mathcal{B}$, then the message alphabet of $\mathcal{A}$ will include states of $\mathcal{B}$. When the CSMs are obtained through projection, it is natural to first obtain $\mathcal{B}$ so we can refer to its states in writing the PSM that projects to $\mathcal{A}$. We thus assume there is an acyclic "delegation partial order" between the CSMs of a process: $\mathcal{B} < \mathcal{A}$ means that $\mathcal{A}$ can use the states of $\mathcal{B}$ in its messages.

The semantics of the calculus is defined on runtime configurations R (defined in Fig. 10), which are processes which additionally contain message queues $s \blacktriangleright \sigma$ for each active session s. Here σ is a map from pairs of participants to sequences of messages. The reduction semantics is standard (cf. [62]). The only reduction rules we highlight here are the ones leading to an error configuration:

$$
\frac{\forall i \in I.\, \sigma(\mathtt{q}_i, \mathtt{p}) = l\langle \_ \rangle \ldots \wedge l_i \neq l}{\mathop{\&}_{i\in I} s[\mathtt{p}][\mathtt{q}_i]?l_i(y_i)\,.\,P_i \parallel s \blacktriangleright \sigma \longrightarrow \mathbf{err}}
\qquad
\frac{\sigma(\mathtt{p},\mathtt{q}) \neq \varepsilon \text{ for some } \mathtt{p},\mathtt{q}}{(\nu s : \mathcal{A})\ s \blacktriangleright \sigma \longrightarrow \mathbf{err}}
$$

The first rule models *unsafe communication*: a process is stuck because all the queues it is waiting to receive from are not empty, but the labels of the first messages do not match any of the cases the process is expecting. The second rule models *orphan messages*: a session where all participants terminated but that has still non-empty message queues. The safety guarantees of our type system will rule out both cases. Note that [58, 59] focuses on communication safety. In addition, they consider S-deadlock freedom, which implies no orphan messages, but is an undecidable property that needs to be checked and is not necessarily transferred to processes by the type system: the property only holds if there is only one session, in which case much stronger conditions transfer. In our setting, deadlock freedom is transferred throughout by projection and the type system, yielding no orphan messages.

Figure 11 shows the crucial rules of AMP's type system. The typing judgement $\Theta \mid \Lambda \vdash P$ uses a process P, a typing context Θ for the types of the parameters $\vec{c}$ of sequential processes $\mathsf{Q}[\vec{c}]$ (the definitions of which are typed separately

$$\frac{}{\Theta \mid \emptyset \vdash \mathbf{0}}\ \text{PT-}\mathbf{0} \qquad \frac{\Theta \mid \Lambda \vdash P \qquad \mathrm{end}(q)}{\Theta \mid c : q, \Lambda \vdash P}\ \text{PT-END}$$

$$\frac{\delta(q) = \{(\mathsf{p} \lhd \mathsf{q}_i ? l_i(p_i), q_i) \mid i \in I\} \qquad \forall i \in I.\, \Theta \mid \Lambda, c : q_i, y_i : p_i \vdash P_i}{\Theta \mid \Lambda, c : q \vdash \mathop{\&}_{i \in I} c[\mathsf{q}_i]?l_i(y_i) \,.\, P_i}\ \text{PT-\&}$$

$$\frac{\delta(q) \supseteq \{(\mathsf{p} \rhd \mathsf{q}_i ! l_i(p_i), q_i) \mid i \in I\} \qquad \forall i \in I.\, \Theta \mid \Lambda, c : q_i, \{c_j : p_j\}_{j \in I \setminus \{i\}} \vdash P_i}{\Theta \mid \Lambda, c : q, \{c_i : p_i\}_{i \in I} \vdash \bigoplus_{i \in I} c[\mathsf{q}_i]!l_i\langle c_i \rangle \,.\, P_i}\ \text{PT-}\oplus$$

$$\frac{\Lambda_s = \{s[\mathsf{p}] : \mathrm{init}(\mathcal{A}_\mathsf{p})\}_{\mathsf{p} \in \mathcal{P}_\mathcal{A}} \qquad \Theta \mid \Lambda, \Lambda_s \vdash P}{\Theta \mid \Lambda \vdash (\nu s : \mathcal{A})\, P}\ \text{PT-}\nu$$

Fig. 11: Typing rules for processes; init(-) denotes a CSM's initial state.

against Θ); a typing context Λ associating the variables x and the channel capabilities $s[\mathsf{p}]$ occurring free in P, with some CSM state $q \in L$. Rule PT-**0** says that a terminated process is typable in the environment with no capabilities. Rule PT-END permits to discard the capabilities that have terminated: $\mathrm{end}(q)$ holds for final states with no outgoing receive transition. Rules PT-& and PT-$\oplus$ deal with communication. Assume $c = s[\mathsf{p}]$. According to PT-&, to receive a message as participant p in session c, we look for the type q of c in the typing context and check the CSM transitions $\delta(q)$ are all receives $\{(\mathsf{p} \lhd \mathsf{q}_i ? l_i(p_i), q_i) \mid i \in I\}$. Then the process needs to be able to receive any branch i, resulting in the continuation P_i which is typed in the context extended with the corresponding payload type $y_i : p_i$, and with the type of c changed to q_i. According to PT-$\oplus$, to send, as c with type q, a message non-deterministically picked from a number of branches $i \in I$, we have to make sure q allows each branch, including matching the types of the payloads. Then each branch i continues as P_i which is typed in a context where c has type q_i and we lost ownership of the payload c_i. Finally, PT-ν types a new session s used by P, by adding to the context a new binding $s[\mathsf{p}] : q_\mathsf{p}$ for each participant p of the CSM $\mathcal{A}$ annotating the session, with q_p being the initial state of p in $\mathcal{A}$.

The first correctness criterion for the type system is to prove subject reduction: if a process is typable, then every configuration reachable from it will be typable. Thus, to state subject reduction, we need to define when a runtime configuration is typable. For this purpose, we define a second judgement $\Theta \mid \Lambda \mid \Omega \vdash R$ that includes a third typing context Ω used to type session message queues: associating to each channel $s[\mathsf{p}][\mathsf{q}]$ a sequence of message types $l(q)$ (label and payload type). The key to make typing of runtime configurations an inductive invariant, is the following rule:

$$\frac{\begin{array}{cc} (\vec{q}, \xi) \in \mathrm{reach}(\mathcal{A}) & \Lambda^s_{\vec{q}} = \{s[\mathsf{p}] : \vec{q}_\mathsf{p}\}_{\mathsf{p} \in \mathcal{P}_\mathcal{A}} \\ \Omega^s_\xi = \{s[\mathsf{p}][\mathsf{q}] :: \xi(\mathsf{p}, \mathsf{q})\}_{\mathsf{p}, \mathsf{q} \in \mathcal{P}_\mathcal{A}} & \Theta \mid \Lambda, \Lambda^s_{\vec{q}} \mid \Omega, \Omega^s_\xi \vdash R \end{array}}{\Theta \mid \Lambda \mid \Omega \vdash (\nu s : \mathcal{A})\, R}\ \text{RT-}\nu$$

The main difference between RT-ν and PT-ν is that the typing context is not populated with capabilities associated to initial states; instead the prover can pick any CSM configuration $(\vec{q}, \xi)$ —where $\vec{q}$ collects the local state of each participant, and ξ the contents of the message queues— that is reachable from the initial configuration of $\mathcal{A}$. The states and the queues are used to initialise the typing context to type the process R using the restricted session.

In what follows we assume the definitions $\mathcal{D}$ can be typed according to Θ. We say a process/runtime configuration is *well-annotated* if every CSM appearing in it is (1) deadlock-free, and (2) satisfies feasible eventual reception. Here, *annotated* indicates that the CSMs have no computational meaning but *well* shows the need for certain guarantees, which our type system can preserve. Note that a process is automatically well-annotated if the CSMs are obtained via projection.

Theorem 5.1 (Subject Reduction). *Given a well-annotated R, if $\Theta \mid \emptyset \mid \emptyset \vdash R$ and $R \longrightarrow R'$, then $\Theta \mid \emptyset \mid \emptyset \vdash R'$.*

Corollary 5.2 (Type Safety). *For a well-annotated R, if $\Theta \mid \emptyset \mid \emptyset \vdash R$ and $R \longrightarrow^* R'$, then R' does not contain* **err**.

For progress, the situation is more delicate: just like in [59] and most other MST systems, allowing session interleaving may introduce inter-session dependencies that are not modelled in the global protocol (which only pertains intra-session dependencies). We thus prove progress under these assumptions: (i) there is only one session running, and (ii) that each of its participants is implemented by a distinct process, and (iii) the CSM annotating it is sink-final. To encode these extra restrictions, we define a "Session Fidelity" variant of the typing judgement $\Theta \mid \Lambda \mid \Omega \vdash_{SF} R$ which uses a subset of the rules of $\vdash$ to enforce the restrictions above. Let $\Lambda^s_{\vec{q}}$ and Ω^s_{ξ} be defined as in the premises of RT-ν.

Theorem 5.3 (Progress). *Let $(\vec{q}, \xi)$ be a configuration of a sink-final, deadlock-free CSM $\mathcal{A}$ satisfying feasible eventual reception. If $\Theta \mid \Lambda^s_{\vec{q}} \mid \Omega^s_{\xi} \vdash_{SF} R$, and $(\vec{q}, \xi)$ can take a step, then there exist some R' and $(\vec{q}', \xi')$, such that $R \longrightarrow R'$, $(\vec{q}, \xi) \longrightarrow (\vec{q}', \xi')$, and $\Theta \mid \Lambda^s_{\vec{q}'} \mid \Omega^s_{\xi'} \vdash_{SF} R'$.*

Progress hinges on deadlock freedom of the CSM. In general, any (language) property of a PSM that is preserved and reflected by $\mathcal{C}(\text{-})$ holds for its projection. However, as for progress, it is not necessarily easy to make the type system enforce the preservation of these properties at the global process level and requires careful treatment. [18] demonstrated how Kobayashi-style techniques [44] that can be used to show progress in the presence of session interleaving. We conjecture a similar system can be added on top of AMP's type system.

6 Applications of AMP to MST Frameworks

Standard (expression-based) global types from MST frameworks can be seen as restricted special cases of PSMs. What is gained from using AMP for global types seen as PSMs? Is anything lost in doing so? In this section, we evaluate AMP as a backend for projection/typing of standard global types. The key consequences of our results are:

(a) Every sender-driven global type is a tame sink-final Σ1-PSM.
(b) Tame sink-final Σ1-PSMs can be represented as a sender-driven global type.
(c) Every collection of (expression-based) local types $\{\!\{L_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ can be expressed as a CSM $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ and vice versa.
(d) AMP's projection is deadlock-free by construction, but MSTs typically insist on freedom of a stricter notion of deadlock which we call *soft deadlock*. We show AMP's projection can also be set to ensure soft deadlock freedom, without losing completeness.

These results help us settle two open questions:

– Expression-based global/local types are equi-expressive with respect to state-machine-based global/local specifications.
– Allowing mixed-choice in global types makes projectability undecidable.

Here, we give an overview while [62] presents the results in detail.

Global and local types. In most MST frameworks, protocols are specified using expression-based global types (G), which get projected to expression-based local types (L). Their syntax is specified as follows:

$$G ::= \mathbf{0} \mid \sum_{i\in I} \mathtt{p}_i{\to}\mathtt{q}_i{:}m_i\,.\,G_i \mid \mu X\,.\,G \mid X \quad \text{(global types)}$$

$$L ::= \mathbf{0} \mid \bigoplus_{i\in I} \mathtt{q}_i!m_i\,.\,L_i \mid \mathop{\&}_{i\in I} \mathtt{q}_i?m_i\,.\,L_i \mid \mu X\,.\,L \mid X \quad \text{(local types)}$$

where $\mathbf{0}$ explicitly denotes termination and μX binds a recursion variable X. The remaining operators specify how messages are exchanged: for local types, sending and receiving are separate actions, while for global types they are specified in a single paired event. Typically only *deterministic* global types are considered, i.e. where every $\sum_{i\in I} \mathtt{p}_i{\to}\mathtt{q}_i{:}m_i\,.\,G_i$ has no $i \neq j$ with $\mathtt{p}_i{\to}\mathtt{q}_i{:}m_i = \mathtt{p}_j{\to}\mathtt{q}_j{:}m_j$. The choice restrictions we discussed, can be imposed on global types, e.g. sender-driven choice requires that, for $\sum_{i\in I} \mathtt{p}_i{\to}\mathtt{q}_i{:}m_i\,.\,G_i$, for all $i, j \in I$, $\mathtt{p}_i = \mathtt{p}_j$.

The standard semantics of global types has been given as a transition system, or as sets of traces. In both cases, the semantics allows reordering of events that are not causally related, e.g. $\mathtt{p}{\to}\mathtt{q}{:}1\,.\,\mathtt{r}{\to}\mathtt{s}{:}2\,.\,\mathbf{0}$ allows $\mathtt{r}$ and $\mathtt{s}$ to communicate before $\mathtt{p}$ and $\mathtt{q}$. This is formalised, in the presentation of [48, 51, 60] (which we adopt here) as defining the semantics of a global type to be a set of traces closed under the indistinguishability relation $\sim$. With this view, it is immediate to represent any global type as a PSM. Given the restricted format of global types, the PSMs corresponding to translations of global types (like the one in Fig. 4) are Σ1-PSMs with a specific shape: they are tree-like, sink-final and recursion only happens at leaves and to ancestors [60]. On the face of it, it is unclear whether every Σ1-PSM can be modelled as global type.

Theorem 6.1. *For every sink-final Σ1-PSM M, there is a global type* $\mathsf{GAut}(M)$ *with the same core language (and hence the same semantics). If M is non-deterministic (mixed-choice, sender-driven, or directed, resp.), then* $\mathsf{GAut}(M)$ *is non-deterministic (mixed-choice, sender-driven, or directed, resp.).*

The main idea of Theorem 6.1 is that one can see a global type as a special regular expression, and thus we can adapt techniques like Arden's lemma and Brzozowski derivatives to the case of PSMs. The key difficulty in the proof lays in showing the branching conditions are preserved: the standard automata transformations change the branching structure, and we need to produce new variants that do.

Similarly, local types can be directly read as the FSMs of a CSM. We can also provide an inverse transformation (preserving branching).

Theorem 6.2. *Let $A_{\mathtt{p}}$ be a sink-final FSM over $\Gamma_{\mathtt{p}}$ without mixed-choice states for a participant* $\mathtt{p}$*. One can construct a local type $L_{\mathtt{p}}$ for* $\mathtt{p}$ *with $\mathcal{L}(L_{\mathtt{p}}) = \mathcal{L}(A_{\mathtt{p}})$.*

Deadlocks and protocol termination. In MSTs, local types can only terminate with a $\mathbf{0}$, which signals at the same time that it is valid to stop the protocol, and that there is no further potential action. This implies that for a global type to be projectable into local types, all the participants need to know unambiguously when the protocol terminated globally. In contrast, using CSMs, it is possible to mark as final a state with outgoing transitions. Consider for instance the (directed) global type $G := (\mathtt{p}{\to}\mathtt{q}{:}m_1 \,.\, \mathtt{p}{\to}\mathtt{r}{:}m_1 \,.\, \mathbf{0}) + (\mathtt{p}{\to}\mathtt{q}{:}m_2 \,.\, \mathbf{0})$. AMP's projection would produce the FSM $\to\circledcirc \xrightarrow{\mathtt{r} \triangleleft \mathtt{p}?m_1} \circledcirc$ as the projection for $\mathtt{r}$. It contains a non-sink final state: $\mathtt{r}$ is not informed of which branch was taken and needs to be prepared to terminate, or receive one more message.

AMP's projection produces deadlock-free CSMs, where deadlocks are defined as configurations which cannot take a step, but their queues are not empty or some participant is in a non-final state. Projections to local types ensure the absence of another type of deadlock: a *soft deadlock*, i.e. a configuration that is a deadlock, or that cannot take a step but where some participant is in a *non-sink* state. Is the possibility of soft deadlocks desirable? We argue that this depends on the domain of application: in distributed computing, it would be fine if a server kept listening for incoming requests while, in embedded computing, it can be key that all participants eventually stop. We can show that it is possible to use AMP in both scenarios, without giving up on completeness.

Definition 6.3 (Strong Projectability). *A language $L \subseteq \Gamma^\omega$ is said to be* strongly projectable *if there exists a CSM $\{\!\{B_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ such that $\{\!\{B_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ is free from soft deadlocks* (soft deadlock freedom)*, and L is the language of $\{\!\{B_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$* (protocol fidelity)*. We say that $\{\!\{B_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ is a strong projection of L.*

Theorem 6.4. *Let G be a projectable global type. Then, the subset construction $\mathscr{C}(G, \mathtt{p})$* [48, Def. 5.4] *is sink-final for every participant* $\mathtt{p}$ *if and only if there is a CSM that is a strong projection of G and this CSM satisfies feasible eventual reception or every of its state machines is deterministic.*

If we aim for a strong projection of a projectable global type, we construct the global type's subset construction and check if it is sink-final. If it is not, there is no strong projection of it. If this is undesirable, the protocol needs redesigning. Theorem 6.2 can yield local types and $\mathsf{LAut}(L)$ is the FSM for a local type L.

Undecidability for mixed-choice. Finally, these results together with our results from Section 4.2, can settle the open question of whether we can project mixed-choice global types algorithmically.

Corollary 6.5. *Both the projectability problem and the strong projectability problem for mixed-choice global types are undecidable.*

7 Related Work

Multiparty session types. Inspired by linear logic [33], Honda [36] proposed binary sessions types for sessions with two participants. Multiparty session types [37] extended the idea to multiple participants. Deniélou and Yoshida [23] were the first to extensively explore the relation between CSMs and local types, but their projection is not complete and only supports directed choice; moreover the approach was found to be somewhat flawed [59]. MSTs have been incorporated into a number of programming languages [3, 17, 42, 47, 49, 54, 57]. They have also been applied to various other domains like operating systems [25], web services [71], distributed algorithms [45], timed systems [8], cyber-physical systems [52], and smart contracts [22]. A number of works are devoted to mechanising MST meta-theory [14, 40, 41, 64]. Our results could potentially extend the expressivity of the types involved in these applications.

MST projectability/projection. Via a reduction to globally-cooperative HMSCs, Stutz [60] proved MST projectability to be decidable for the class of global types that can —but do not have to— terminate (called **0**-reachable). Li et al. [48] provided a direct MST projection algorithm that is complete for sender-driven global types, providing a PSPACE upper bound. Our results use a reduction to these later developments. The global specifications in [48] can be shown to be special cases of Tame Σ1-PSMs so our results strictly expand the reach of their results. For example, the protocols of Figs. 5 and 7 are all tame, yet out of the reach of both works. We also clarify the discrepancy between the notion of deadlock in global types and in [48] (cf. Section 6). Finally, [48, 60] do not have a type system, providing no way to link properties of projections with the ones of processes. Preliminary versions of some of our results appeared in the PhD thesis of the first author [61].

The almost totality of asynchronous MST works can only handle directed choice. An exception is [12], where unrestricted global types are considered (without a type system). They propose an incomplete projection algorithm that is correct with respect to a different notion of correct projection than the standard one we adopt and generalise. We refer to [51] for a survey on choice restrictions.

Hu and Yoshida [38] propose a scheme with global types and an incomplete projection, where the global types are not safe by construction and the restrictions on choice only appear at the local types. The safety of global types is ensured by a combination of model-checking with message buffers of size 1, and syntactic restrictions that ensure that any unsafety that might arise, will be visible in the 1-bounded executions. For PSMs satisfying the syntactic restrictions, the same approach could be applied. The types of [38] also include connect/disconnect actions, which can be emulated in AMP by excluding deadlocks (but not soft deadlocks) and using non-sink final states.

Choreography automata and languages. Choreography Automata [4] are syntactically similar to Σ1-PSMs, but do not employ any closure operation, requiring the user to specify all the allowed interleavings, and preventing finite state representations for many common communication patterns. In addition, Majumdar et al. [51] showed that their conditions for projectability are flawed for the asynchronous case (fixes for the synchronous case appeared in [31, p. 8]). Barbanera et al. [5] applies a language-theoretic approach to a limited class of *synchronous* choreographies (with no claim of completeness of projection).

Bottom-up MSTs. A number of MST-based works deviate from the top-down approach. For instance, [59] proposes a type system that only requires local types and not a global type. The typing ensures some operational correspondence between local types and processes, making it possible to model check local types to determine properties of the program. Their local types in the asynchronous setting are Turing-powerful, and therefore model checking is of limited use. By virtue of the decoupling achieved by our type system, AMP can be used in a bottom-up way too: safety of the CSM used to type a process, implies safety of the process, regardless of whether the CSM is obtained by projection or just given. Dagnino et al. [21] and Castellani et al. [13] also use a bottom-up strategy by reconstructing a so-called *deconfined* global type from the parallel composition of local programs of a single session. Deconfined global types are not automatically safe, and checking their safety is shown to be undecidable.

Extensions for MSTs. A number of extensions for MSTs have been considered (see [7] for a survey), including: parametrisation [16, 24], dependent types [24, 66, 67], graduality [39], fault-tolerance [6, 70]. Context-free session types [43, 65] specify binary sessions that are not representable with finite-control. It would be interesting to consider projection for PSMs generated by pushdown automata.

Distributed synthesis. In automata theory, distributed synthesis seeks a way to transform a sequential specification into an equivalent distributed implementation, which is close in spirit with the idea of extracting local types from global types. One of the few positive results in this area is Zielonka's theorem [72], which shows that every regular trace language can be recognised by a so-called "asynchronous automaton". Despite their name, asynchronous automata can be seen as a parallel composition of participants interacting through *synchronous* actions. In contrast, PSMs and CSMs represent non-regular FIFO languages, giving rise to a harder challenge.

High-level message sequence charts. HMSCs were defined in an industry standard [69], inspiring extensive academic research [26, 28, 29, 53, 56]. Projectability has been studied for HMSCs under the name "safe realisability" [2, 30, 50], and was shown to be undecidable in general [50]. Several restrictions of HMSCs have been proposed to make projectability decidable. For a detailed survey, we refer to [60]. Compared to PSMs, HMSCs only model finite runs; their PSM encoding equips them with an infinite run semantics. With our developments, it is fairly straightforward to obtain a projection operation for sender-driven, sink-final HMSCs that respect some channel bounds. This class is incomparable to any of the decidable HMSC classes proposed in the literature. Since our type system only depends on CSMs, regardless of how they are obtained, AMP can type check a program against a projectable HMSC.

Choreographic programming. Choreographic programming [19, 32, 35] adopts the top-down approach even more radically than MSTs. In choreographic programming, the endpoint projection (EPP) aims at synthesizing a fully-featured program implementation directly from the global specification. As a result, the global specification describes the local computation alongside the communication structure (requiring infinite-state formalisms). In choreographies, one typically works with non-finite-control-state global specifications, so the hopes for a complete and decidable EPP are slim, justifying giving up on completeness. By only considering local computation in processes, the MST/AMP approach avoids this issue. Nevertheless, our results could still be useful for EPP when applied to the pure communication structure of choreographies. Notably, our method can project examples that cannot be projected using EPPs from the literature. Consider the choreography **if** $\mathtt{p}.\star$ **then** $(\mathtt{p}\to\mathtt{q}\mathtt{:\_}\,.\,\mathtt{q}\to\mathtt{s}\mathtt{:\_})$ **else** $(\mathtt{p}\to\mathtt{s}\mathtt{:\_}\,.\,\mathtt{s}\to\mathtt{q}\mathtt{:\_})$, where message payloads are irrelevant and hence omitted and $\mathtt{p}.\star$ denotes non-deterministic choice by $\mathtt{p}$. The example is syntactically valid in [20] and can be easily encoded as a global type with sender-driven choice. However, their EPP would be undefined for $\mathtt{q}$ and $\mathtt{s}$: it uses the merge from [11], which can only merge same sender receives. Our results would instead produce the desired projection.

Communicating state machines. CSMs are the canonical automata model for distributed systems. They have been studied in the context of model checking projections and do not apply a top-down methodology. The verification problem is undecidable in general since CSMs are Turing-powerful [10]. Several strategies to yield decidable classes have been proposed: assuming channels are lossy [1], restricting the communication topology [55, 68], or only allowing half-duplex communication for two participants [15]. The concept of existential boundedness [27] was initially defined for CSMs and yields decidability of control state reachability. The same holds for synchronisability [9, 34], which, intuitively, requires that every execution can be re-ordered (up to $\sim$) into phases of sends and receives such that messages can only be received in the same phase. Global types can only express 1-synchronisable and half-duplex communication [63].

Acknowledgments. The authors thank Anca Muscholl and Jorge A. Pérez for their discussions. This work was partially supported by the Luxembourg National Research Fund (FNR) under the grant agreement C22/IS/17238244/AVVA and by the ERC Consolidator Grant for the project "PERSIST" under the EU's Horizon 2020 research and innovation programme (grant No. 101003349).

Bibliography

[1] Abdulla, P.A., Bouajjani, A., Jonsson, B.: On-the-fly analysis of systems with unbounded, lossy FIFO channels. In: Hu, A.J., Vardi, M.Y. (eds.) Computer Aided Verification, 10th International Conference, CAV'98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings, Lecture Notes in Computer Science, vol. 1427, pp. 305–318, Springer (1998), https://doi.org/10.1007/BFb0028754

[2] Alur, R., Etessami, K., Yannakakis, M.: Realizability and verification of MSC graphs. Theor. Comput. Sci. **331**(1), 97–114 (2005), https://doi.org/10.1016/j.tcs.2004.09.034

[3] Ancona, D., Bono, V., Bravetti, M., Campos, J., Castagna, G., Deniélou, P., Gay, S.J., Gesbert, N., Giachino, E., Hu, R., Johnsen, E.B., Martins, F., Mascardi, V., Montesi, F., Neykova, R., Ng, N., Padovani, L., Vasconcelos, V.T., Yoshida, N.: Behavioral types in programming languages. Found. Trends Program. Lang. **3**(2-3), 95–230 (2016), https://doi.org/10.1561/2500000031

[4] Barbanera, F., Lanese, I., Tuosto, E.: Choreography automata. In: Bliudze, S., Bocchi, L. (eds.) Coordination Models and Languages - 22nd IFIP WG 6.1 International Conference, COORDINATION 2020, Held as Part of the 15th International Federated Conference on Distributed Computing Techniques, DisCoTec 2020, Valletta, Malta, June 15-19, 2020, Proceedings, Lecture Notes in Computer Science, vol. 12134, pp. 86–106, Springer (2020), https://doi.org/10.1007/978-3-030-50029-0_6

[5] Barbanera, F., Lanese, I., Tuosto, E.: Formal choreographic languages. In: ter Beek, M.H., Sirjani, M. (eds.) Coordination Models and Languages - 24th IFIP WG 6.1 International Conference, COORDINATION 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, June 13-17, 2022, Proceedings, Lecture Notes in Computer Science, vol. 13271, pp. 121–139, Springer (2022), https://doi.org/10.1007/978-3-031-08143-9_8

[6] Barwell, A.D., Scalas, A., Yoshida, N., Zhou, F.: Generalised multiparty session types with crash-stop failures. In: Klin, B., Lasota, S., Muscholl, A. (eds.) 33rd International Conference on Concurrency Theory, CONCUR 2022, September 12-16, 2022, Warsaw, Poland, LIPIcs, vol. 243, pp. 35:1–35:25, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022), https://doi.org/10.4230/LIPIcs.CONCUR.2022.35

[7] Bejleri, A., Domnori, E., Viering, M., Eugster, P., Mezini, M.: Comprehensive multiparty session types. Art Sci. Eng. Program. **3**(3), 6 (2019), https://doi.org/10.22152/programming-journal.org/2019/3/6

[8] Bocchi, L., Murgia, M., Vasconcelos, V.T., Yoshida, N.: Asynchronous timed session types - from duality to time-sensitive processes. In: Caires, L. (ed.) Programming Languages and Systems - 28th European Symposium on

Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Lecture Notes in Computer Science, vol. 11423, pp. 583–610, Springer (2019), https://doi.org/10.1007/978-3-030-17184-1_21

[9] Bouajjani, A., Enea, C., Ji, K., Qadeer, S.: On the completeness of verifying message passing programs under bounded asynchrony. In: Chockler, H., Weissenbacher, G. (eds.) Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II, Lecture Notes in Computer Science, vol. 10982, pp. 372–391, Springer (2018), https://doi.org/10.1007/978-3-319-96142-2_23

[10] Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM **30**(2), 323–342 (1983), https://doi.org/10.1145/322374.322380

[11] Carbone, M., Honda, K., Yoshida, N.: Structured communication-centered programming for web services. ACM Trans. Program. Lang. Syst. **34**(2), 8:1–8:78 (2012), https://doi.org/10.1145/2220365.2220367

[12] Castagna, G., Dezani-Ciancaglini, M., Padovani, L.: On global types and multi-party session. Log. Methods Comput. Sci. **8**(1) (2012), https://doi.org/10.2168/LMCS-8(1:24)2012

[13] Castellani, I., Dezani-Ciancaglini, M., Giannini, P.: Asynchronous sessions with input races. In: Carbone, M., Neykova, R. (eds.) Proceedings of the 13th International Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software, PLACES@ETAPS 2022, Munich, Germany, 3rd April 2022, EPTCS, vol. 356, pp. 12–23 (2022), https://doi.org/10.4204/EPTCS.356.2

[14] Castro-Perez, D., Ferreira, F., Gheri, L., Yoshida, N.: Zooid: a DSL for certified multiparty computation: from mechanised metatheory to certified multiparty processes. In: Freund, S.N., Yahav, E. (eds.) PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021, pp. 237–251, ACM (2021), https://doi.org/10.1145/3453483.3454041

[15] Cécé, G., Finkel, A.: Verification of programs with half-duplex communication. Inf. Comput. **202**(2), 166–190 (2005), https://doi.org/10.1016/j.ic.2005.05.006

[16] Charalambides, M., Dinges, P., Agha, G.A.: Parameterized, concurrent session types for asynchronous multi-actor interactions. Sci. Comput. Program. **115-116**, 100–126 (2016), https://doi.org/10.1016/j.scico.2015.10.006

[17] Chen, R., Balzer, S., Toninho, B.: Ferrite: A judgmental embedding of session types in rust. In: Ali, K., Vitek, J. (eds.) 36th European Conference on Object-Oriented Programming, ECOOP 2022, June 6-10, 2022, Berlin, Germany, LIPIcs, vol. 222, pp. 22:1–22:28, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022), https://doi.org/10.4230/LIPIcs.ECOOP.2022.22

[18] Coppo, M., Dezani-Ciancaglini, M., Yoshida, N., Padovani, L.: Global progress for dynamically interleaved multiparty sessions. Math.

Struct. Comput. Sci. **26**(2), 238–302 (2016), https://doi.org/10.1017/S0960129514000188

[19] Cruz-Filipe, L., Montesi, F.: A core model for choreographic programming. Theor. Comput. Sci. **802**, 38–66 (2020), https://doi.org/10.1016/j.tcs.2019.07.005

[20] Cruz-Filipe, L., Montesi, F., Peressotti, M.: Communications in choreographies, revisited. In: Haddad, H.M., Wainwright, R.L., Chbeir, R. (eds.) Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC 2018, Pau, France, April 09-13, 2018, pp. 1248–1255, ACM (2018), https://doi.org/10.1145/3167132.3167267

[21] Dagnino, F., Giannini, P., Dezani-Ciancaglini, M.: Deconfined global types for asynchronous sessions. In: Damiani, F., Dardha, O. (eds.) Coordination Models and Languages - 23rd IFIP WG 6.1 International Conference, COORDINATION 2021, Held as Part of the 16th International Federated Conference on Distributed Computing Techniques, DisCoTec 2021, Valletta, Malta, June 14-18, 2021, Proceedings, Lecture Notes in Computer Science, vol. 12717, pp. 41–60, Springer (2021), https://doi.org/10.1007/978-3-030-78142-2_3

[22] Das, A., Balzer, S., Hoffmann, J., Pfenning, F., Santurkar, I.: Resource-aware session types for digital contracts. In: 34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, June 21-25, 2021, pp. 1–16, IEEE (2021), https://doi.org/10.1109/CSF51468.2021.00004

[23] Deniélou, P., Yoshida, N.: Multiparty session types meet communicating automata. In: Seidl, H. (ed.) Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings, Lecture Notes in Computer Science, vol. 7211, pp. 194–213, Springer (2012), https://doi.org/10.1007/978-3-642-28869-2_10

[24] Deniélou, P., Yoshida, N., Bejleri, A., Hu, R.: Parameterised multiparty session types. Log. Methods Comput. Sci. **8**(4) (2012), https://doi.org/10.2168/LMCS-8(4:6)2012

[25] Fähndrich, M., Aiken, M., Hawblitzel, C., Hodson, O., Hunt, G.C., Larus, J.R., Levi, S.: Language support for fast and reliable message-based communication in singularity OS. In: Berbers, Y., Zwaenepoel, W. (eds.) Proceedings of the 2006 EuroSys Conference, Leuven, Belgium, April 18-21, 2006, pp. 177–190, ACM (2006), https://doi.org/10.1145/1217935.1217953

[26] Gazagnaire, T., Genest, B., Hélouët, L., Thiagarajan, P.S., Yang, S.: Causal message sequence charts. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007 - Concurrency Theory, 18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceedings, Lecture Notes in Computer Science, vol. 4703, pp. 166–180, Springer (2007), https://doi.org/10.1007/978-3-540-74407-8_12

[27] Genest, B., Kuske, D., Muscholl, A.: On communicating automata with bounded channels. Fundam. Inform. **80**(1-3), 147–167 (2007), URL http://content.iospress.com/articles/fundamenta-informaticae/fi80-1-3-09

[28] Genest, B., Muscholl, A.: Message sequence charts: A survey. In: Fifth International Conference on Application of Concurrency to System Design (ACSD 2005), 6-9 June 2005, St. Malo, France, pp. 2–4, IEEE Computer Society (2005), https://doi.org/10.1109/ACSD.2005.25

[29] Genest, B., Muscholl, A., Peled, D.A.: Message sequence charts. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets, Advances in Petri Nets [This tutorial volume originates from the 4th Advanced Course on Petri Nets, ACPN 2003, held in Eichstätt, Germany in September 2003. In addition to lectures given at ACPN 2003, additional chapters have been commissioned], Lecture Notes in Computer Science, vol. 3098, pp. 537–558, Springer (2003), https://doi.org/10.1007/978-3-540-27755-2_15

[30] Genest, B., Muscholl, A., Seidl, H., Zeitoun, M.: Infinite-state high-level mscs: Model-checking and realizability. J. Comput. Syst. Sci. **72**(4), 617–647 (2006), https://doi.org/10.1016/j.jcss.2005.09.007

[31] Gheri, L., Lanese, I., Sayers, N., Tuosto, E., Yoshida, N.: Design-by-contract for flexible multiparty session protocols. In: Ali, K., Vitek, J. (eds.) 36th European Conference on Object-Oriented Programming, ECOOP 2022, June 6-10, 2022, Berlin, Germany, LIPIcs, vol. 222, pp. 8:1–8:28, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022), https://doi.org/10.4230/LIPICS.ECOOP.2022.8

[32] Giallorenzo, S., Montesi, F., Peressotti, M., Richter, D., Salvaneschi, G., Weisenburger, P.: Multiparty languages: The choreographic and multitier cases (pearl). In: Møller, A., Sridharan, M. (eds.) 35th European Conference on Object-Oriented Programming, ECOOP 2021, July 11-17, 2021, Aarhus, Denmark (Virtual Conference), LIPIcs, vol. 194, pp. 22:1–22:27, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021), https://doi.org/10.4230/LIPIcs.ECOOP.2021.22

[33] Girard, J.: Linear logic. Theor. Comput. Sci. **50**, 1–102 (1987), https://doi.org/10.1016/0304-3975(87)90045-4

[34] Giusto, C.D., Laversa, L., Lozes, É.: On the k-synchronizability of systems. In: Goubault-Larrecq, J., König, B. (eds.) Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Lecture Notes in Computer Science, vol. 12077, pp. 157–176, Springer (2020), https://doi.org/10.1007/978-3-030-45231-5_9

[35] Hirsch, A.K., Garg, D.: Pirouette: higher-order typed functional choreographies. Proc. ACM Program. Lang. **6**(POPL), 1–27 (2022), https://doi.org/10.1145/3498684

[36] Honda, K.: Types for dyadic interaction. In: Best, E. (ed.) CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings, Lecture Notes in Computer Science, vol. 715, pp. 509–523, Springer (1993), https://doi.org/10.1007/3-540-57208-2_35

[37] Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: Necula, G.C., Wadler, P. (eds.) Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008, pp. 273–284, ACM (2008), https://doi.org/10.1145/1328438.1328472

[38] Hu, R., Yoshida, N.: Explicit connection actions in multiparty session types. In: Huisman, M., Rubin, J. (eds.) Fundamental Approaches to Software Engineering - 20th International Conference, FASE 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Lecture Notes in Computer Science, vol. 10202, pp. 116–133, Springer (2017), https://doi.org/10.1007/978-3-662-54494-5_7

[39] Igarashi, A., Thiemann, P., Tsuda, Y., Vasconcelos, V.T., Wadler, P.: Gradual session types. J. Funct. Program. **29**, e17 (2019), https://doi.org/10.1017/S0956796819000169

[40] Jacobs, J., Balzer, S., Krebbers, R.: Connectivity graphs: a method for proving deadlock freedom based on separation logic. Proc. ACM Program. Lang. **6**(POPL), 1–33 (2022), https://doi.org/10.1145/3498662

[41] Jacobs, J., Balzer, S., Krebbers, R.: Multiparty GV: functional multiparty session types with certified deadlock freedom. Proc. ACM Program. Lang. **6**(ICFP), 466–495 (2022), https://doi.org/10.1145/3547638

[42] Jespersen, T.B.L., Munksgaard, P., Larsen, K.F.: Session types for rust. In: Bahr, P., Erdweg, S. (eds.) Proceedings of the 11th ACM SIGPLAN Workshop on Generic Programming, WGP@ICFP 2015, Vancouver, BC, Canada, August 30, 2015, pp. 13–22, ACM (2015), https://doi.org/10.1145/2808098.2808100

[43] Keizer, A.C., Basold, H., Pérez, J.A.: Session coalgebras: A coalgebraic view on regular and context-free session types. ACM Trans. Program. Lang. Syst. **44**(3), 18:1–18:45 (2022), https://doi.org/10.1145/3527633

[44] Kobayashi, N.: A new type system for deadlock-free processes. In: CONCUR, Lecture Notes in Computer Science, vol. 4137, pp. 233–247, Springer (2006), https://doi.org/10.1007/11817949_16

[45] Kouzapas, D., Gutkovas, R., Voinea, A.L., Gay, S.J.: A session type system for asynchronous unreliable broadcast communication. CoRR **abs/1902.01353** (2019), URL http://arxiv.org/abs/1902.01353

[46] Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**(7), 558–565 (1978), https://doi.org/10.1145/359545.359563

[47] Lange, J., Ng, N., Toninho, B., Yoshida, N.: A static verification framework for message passing in go using behavioural types. In: Chaudron, M., Crnkovic, I., Chechik, M., Harman, M. (eds.) Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018, pp. 1137–1148, ACM (2018), https://doi.org/10.1145/3180155.3180157

[48] Li, E., Stutz, F., Wies, T., Zufferey, D.: Complete multiparty session type projection with automata. In: Enea, C., Lal, A. (eds.) Computer Aided

Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part III, Lecture Notes in Computer Science, vol. 13966, pp. 350–373, Springer (2023), https://doi.org/10.1007/978-3-031-37709-9_17

[49] Lindley, S., Morris, J.G.: Embedding session types in haskell. In: Mainland, G. (ed.) Proceedings of the 9th International Symposium on Haskell, Haskell 2016, Nara, Japan, September 22-23, 2016, pp. 133–145, ACM (2016), https://doi.org/10.1145/2976002.2976018

[50] Lohrey, M.: Realizability of high-level message sequence charts: closing the gaps. Theor. Comput. Sci. **309**(1-3), 529–554 (2003), https://doi.org/10.1016/j.tcs.2003.08.002

[51] Majumdar, R., Mukund, M., Stutz, F., Zufferey, D.: Generalising projection in asynchronous multiparty session types. In: Haddad, S., Varacca, D. (eds.) 32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference, LIPIcs, vol. 203, pp. 35:1–35:24, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021), https://doi.org/10.4230/LIPIcs.CONCUR.2021.35

[52] Majumdar, R., Pirron, M., Yoshida, N., Zufferey, D.: Motion session types for robotic interactions (brave new idea paper). In: Donaldson, A.F. (ed.) 33rd European Conference on Object-Oriented Programming, ECOOP 2019, July 15-19, 2019, London, United Kingdom, LIPIcs, vol. 134, pp. 28:1–28:27, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019), https://doi.org/10.4230/LIPIcs.ECOOP.2019.28

[53] Mauw, S., Reniers, M.A.: High-level message sequence charts. In: Cavalli, A.R., Sarma, A. (eds.) SDL '97 Time for Testing, SDL, MSC and Trends - 8th International SDL Forum, Evry, France, 23-29 September 1997, Proceedings, pp. 291–306, Elsevier (1997)

[54] Neykova, R., Hu, R., Yoshida, N., Abdeljallal, F.: A session type provider: compile-time API generation of distributed protocols with refinements in f#. In: Dubach, C., Xue, J. (eds.) Proceedings of the 27th International Conference on Compiler Construction, CC 2018, February 24-25, 2018, Vienna, Austria, pp. 128–138, ACM (2018), https://doi.org/10.1145/3178372.3179495

[55] Peng, W., Purushothaman, S.: Analysis of a class of communicating finite state machines. Acta Informatica **29**(6/7), 499–522 (1992), https://doi.org/10.1007/BF01185558

[56] Roychoudhury, A., Goel, A., Sengupta, B.: Symbolic message sequence charts. ACM Trans. Softw. Eng. Methodol. **21**(2), 12:1–12:44 (2012), https://doi.org/10.1145/2089116.2089122

[57] Scalas, A., Yoshida, N.: Lightweight session programming in scala. In: Krishnamurthi, S., Lerner, B.S. (eds.) 30th European Conference on Object-Oriented Programming, ECOOP 2016, July 18-22, 2016, Rome, Italy, LIPIcs, vol. 56, pp. 21:1–21:28, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016), https://doi.org/10.4230/LIPIcs.ECOOP.2016.21

[58] Scalas, A., Yoshida, N.: Less is more: multiparty session types revisited. Technical Report 6. Imperial College London (2018), URL https://www.doc.ic.ac.uk/research/technicalreports/2018/DTRS18-6.pdf

[59] Scalas, A., Yoshida, N.: Less is more: multiparty session types revisited. Proc. ACM Program. Lang. **3**(POPL), 30:1–30:29 (2019), https://doi.org/10.1145/3290343

[60] Stutz, F.: Asynchronous multiparty session type implementability is decidable - lessons learned from message sequence charts. In: Ali, K., Salvaneschi, G. (eds.) 37th European Conference on Object-Oriented Programming, ECOOP 2023, July 17-21, 2023, Seattle, Washington, United States, LIPIcs, vol. 263, pp. 32:1–32:31, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023), https://doi.org/10.4230/LIPIcs.ECOOP.2023.32

[61] Stutz, F.: Implementability of Asynchronous Communication Protocols - The Power of Choice. Ph.D. thesis, Kaiserslautern University of Technology, Germany (2024), URL https://kluedo.ub.rptu.de/frontdoor/index/index/docId/8077

[62] Stutz, F., D'Osualdo, E.: An automata-theoretic basis for specification and type checking of multiparty protocols. CoRR **abs/2501.16977** (2025), https://doi.org/10.48550/arXiv.2501.16977

[63] Stutz, F., Zufferey, D.: Comparing channel restrictions of communicating state machines, high-level message sequence charts, and multiparty session types. In: Ganty, P., Monica, D.D. (eds.) Proceedings of the 13th International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2022, Madrid, Spain, September 21-23, 2022, EPTCS, vol. 370, pp. 194–212 (2022), https://doi.org/10.4204/EPTCS.370.13

[64] Thiemann, P.: Intrinsically-typed mechanized semantics for session types. In: Komendantskaya, E. (ed.) Proceedings of the 21st International Symposium on Principles and Practice of Programming Languages, PPDP 2019, Porto, Portugal, October 7-9, 2019, pp. 19:1–19:15, ACM (2019), https://doi.org/10.1145/3354166.3354184

[65] Thiemann, P., Vasconcelos, V.T.: Context-free session types. In: Garrigue, J., Keller, G., Sumii, E. (eds.) Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016, pp. 462–475, ACM (2016), https://doi.org/10.1145/2951913.2951926

[66] Toninho, B., Caires, L., Pfenning, F.: Dependent session types via intuitionistic linear type theory. In: Schneider-Kamp, P., Hanus, M. (eds.) Proceedings of the 13th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 20-22, 2011, Odense, Denmark, pp. 161–172, ACM (2011), https://doi.org/10.1145/2003476.2003499

[67] Toninho, B., Yoshida, N.: Depending on session-typed processes. In: Baier, C., Lago, U.D. (eds.) Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Lecture

Notes in Computer Science, vol. 10803, pp. 128–145, Springer (2018), https://doi.org/10.1007/978-3-319-89366-2_7

[68] Torre, S.L., Madhusudan, P., Parlato, G.: Context-bounded analysis of concurrent queue systems. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings, Lecture Notes in Computer Science, vol. 4963, pp. 299–314, Springer (2008), https://doi.org/10.1007/978-3-540-78800-3_21

[69] Union, I.T.: Z.120: Message sequence chart. Tech. rep., International Telecommunication Union (October 1996), URL https://www.itu.int/rec/T-REC-Z.120

[70] Viering, M., Hu, R., Eugster, P., Ziarek, L.: A multiparty session typing discipline for fault-tolerant event-driven distributed programming. Proc. ACM Program. Lang. **5**(OOPSLA), 1–30 (2021), https://doi.org/10.1145/3485501

[71] Yoshida, N., Hu, R., Neykova, R., Ng, N.: The scribble protocol language. In: Abadi, M., Lluch-Lafuente, A. (eds.) Trustworthy Global Computing - 8th International Symposium, TGC 2013, Buenos Aires, Argentina, August 30-31, 2013, Revised Selected Papers, Lecture Notes in Computer Science, vol. 8358, pp. 22–41, Springer (2013), https://doi.org/10.1007/978-3-319-05119-2_3

[72] Zielonka, W.: Notes on finite asynchronous automata. RAIRO Theor. Informatics Appl. **21**(2), 99–135 (1987), https://doi.org/10.1051/ITA/1987210200991