Implementability of Global Distributed Protocols modulo Network Architectures

ELAINE LI, New York University, USA
THOMAS WIES, New York University, USA

Global protocols specify distributed, message-passing protocols from a birds-eye view, and are used as a specification for synthesizing local implementations. Implementability asks whether a given global protocol admits a distributed implementation. We present the first comprehensive investigation of global protocol implementability modulo network architectures. We propose a set of network-parametric Coherence Conditions, and exhibit sufficient assumptions under which it precisely characterizes implementability. We further reduce these assumptions to a minimal set of operational axioms describing insert and remove behavior of individual message buffers. Our reduction immediately establishes that five commonly studied asynchronous network architectures, namely peer-to-peer FIFO, mailbox, senderbox, monobox and bag, are instances of our network-parametric result. We use our characterization to derive optimal complexity results for implementability modulo networks, relationships between classes of implementable global protocols, and symbolic algorithms for deciding implementability modulo networks. We implement the latter in the first network-parametric tool Sprout (𝔸), and show that it achieves network generality without sacrificing performance and modularity.

Additional Key Words and Phrases: Asynchronous communication, network semantics, global protocol verification, communicating state machines, multiparty session types

1 Introduction

Distributed, message-passing protocols are notoriously difficult to implement correctly. Asynchronous network interleavings of independent events make the detection of communication errors such as orphan messages, unspecified receptions and deadlocks especially challenging. Global protocol specifications enjoy the illusion of synchrony, specifying send and receive events atomically from the perspective of an omniscient observer, thereby ruling out large classes of communication errors by construction. Global protocols are then used as a blueprint from which to synthesize correct-by-construction distributed implementations. This approach inspires a top-down verification methodology, whose central decision problem is realizability, also called implementability. Implementability asks whether a given global specification admits a distributed implementation, in other words, whether each participant's local view is sufficient for all participants to collectively follow the protocol designer's omniscient intent without relying on covert coordination.

Global protocols as a specification mechanism do not a priori commit to a particular network model. This choice is instead deferred to the assignment of semantics to global protocols, which in turn depends on the target distributed implementation model. By far the most common network model studied in the literature is peer-to-peer (p2p) FIFO communication, in which protocol participants are pairwise connected by ordered channels. Realizability of global protocols targeting p2p FIFO communcation has been thoroughly studied in the domain of high-level message sequence charts (HMSCs) [2, 3, 27, 29–31, 59, 62, 64, 65, 74], and p2p FIFO is the predominant network model for multiparty session types [4, 5, 37–39, 43, 48, 54, 54, 77, 84] and choreographic programming frameworks [17, 34, 40, 41, 75]. Implementations of such frameworks can be found in more than a dozen programming languages. While p2p FIFO communication is ubiquitous in practice, other network models also hold practical and theoretical interest. For example, Erlang and Go implement mailbox communication, and many correctness proofs of classic distributed algorithms such as leader election [25] and clock synchronization [51] rely on bag semantics.

Authors' Contact Information: Elaine Li, New York University, New York, USA, efl9013@nyu.edu; Thomas Wies, New York University, New York, USA, wies@cs.nyu.edu.

Naturally, the question arises whether the top-down methodology of global protocols can be extended to target different network models. The diversity of communication models and ways to define them semantically has been studied as a topic of independent interest [12, 13, 35], but the more salient research question is whether existing results targeting a particular network model can be generalized to the others. Decision problems that have been studied in a multi-network setting include synchronizability and reachability [6, 20, 24, 59] of communicating finite-state machines. Communication errors in one network model do not necessarily arise in another, making the design of truly network-parametric algorithms very challenging. Consequently, the vast majority of existing results target a fixed network model, and the question of if and how they generalize is left unanswered.

We provide a positive answer to this question through the first comprehensive investigation of the implementability problem for global specifications modulo network architectures. We represent network architectures as formal languages, and propose a set of semantic conditions, called Generalized Coherence Conditions, parametric in a choice of network architecture. We exhibit sufficient conditions under which our network-parametric conditions are sound and complete with respect to implementability. These sufficient conditions take the shape of a set of abstract, algebraic properties over formal languages defining the behavior of network architectures, and are the key to enabling a unified, network-parametric proof of soundness and completeness. This first set of contributions constitutes a network-parametric, precise solution to global protocol implementability.

We take network parametricity one step further by reducing our formal language-theoretic sufficiency conditions to a set of simple, operational axioms defined over buffer data structures. These buffer axioms greatly simplify the task of determining whether a network architecture is an instance of our result, by boiling it down to proving a few specifications about insert and remove operations. As an immediate consequence of this reduction, we conclude that five commonly considered network models, namely peer-to-peer FIFO, mailbox (n-1), senderbox (1-n), monobox (1-1) and bag, are all instances of our parametric result. A perhaps more surprising result is that our axioms leave the communication topology completely unconstrained and accommodate even heterogeneous network architectures featuring multiple buffer data structures.

As a final set of contributions, we derive concrete algorithms for checking implementability modulo network architectures and obtain decidability and complexity results for various protocol fragments. For finite global protocols, we show that implementability is co-NP-complete for all aforementioned network architectures, even for the special case when global protocols belong to the fragment of directed choice multiparty session types [43]. For symbolic global protocols with first-order logic transition constraints, we show that our network-parametric implementability characterization admits an encoding in the first-order fixpoint logic μCLP [79].

Thus, our characterization yields a symbolic algorithm for checking implementability of protocols with infinitely many states and data. We implement our derived symbolic algorithm in a tool Sprout ($\mathbb{A}$), extending an existing tool that decides implementability for global protocols assuming p2p FIFO networks [57]. Our evaluation shows that Sprout ($\mathbb{A}$) extends modularly to different architectures without sacrificing performance.

Contributions. In summary, our contributions are:

- We introduce the implementability problem for global protocol specifications modulo network architectures.
- We give conditions that characterize implementability in a network-parametric fashion and provide sufficient conditions under which this characterization is sound and complete.
- We reduce these sufficient conditions to a simple axiomatic model of buffer data structures that can be easily checked for any given network architecture.

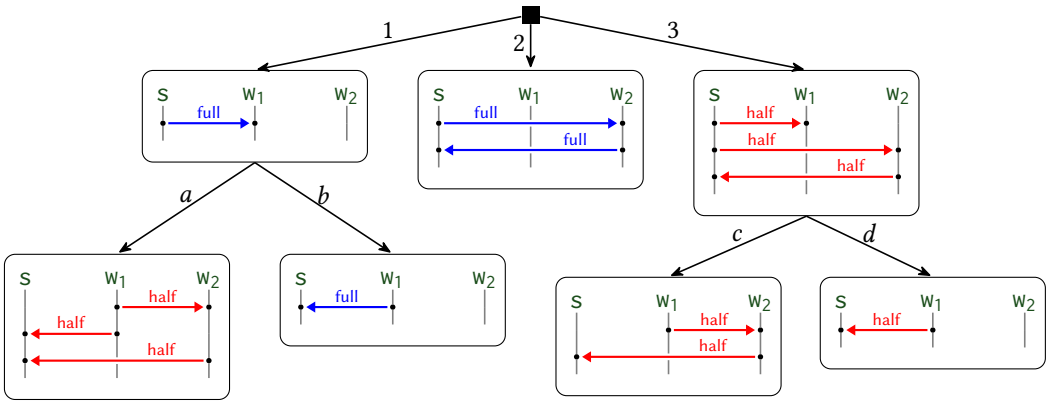

Fig. 1. Task scheduler with task delegation.

- We derive decidability and complexity results for finite-state protocol implementability modulo five commonly considered network architectures as well as algorithms for checking implementability of symbolic global protocols.
- We present Sprout($\mathbb{A}$), the first network-parametric implementability checker for symbolic global protocols.

2 Motivating Example

We use a task scheduling protocol to motivate the subtle differences across network architectures. The global specification of the protocol is shown in Fig. 1, using high-level message sequence chart (HMSC) notation. The protocol involves three participants: a scheduler s and two workers w_1 and w_2. Initially, s chooses to schedule the entire task to either only w_1 or only w_2, or it decides to split the workload between the two workers. The first two cases are depicted in branches 1 and 2, where s sends a full message to the respective worker. The third case is branch 3 where s sends a half message to both workers. Worker w_2 always completes its assigned task immediately and sends the result back to s by echoing the message it has received from s. However, whenever w_1 is assigned a task, it has the option to behave like w_2 (branches *b* and *d*) or delegate some or all of its work to w_2 (branches *a* and *c*). The protocol operates in a loop, but we omit the back-edges to the initial state in Fig. 1 for readability.

Implementability modulo network architectures. Implementability asks whether there exist local implementations for the three participants that behave according to the global protocol specification when executed concurrently on an asynchronous network architecture. In particular, implementations should never deadlock and all participants should behave consistently according to each locally chosen branch, executing send and receive actions exactly in the prescribed order. The latter property is known as *protocol fidelity*. The network architecture is left a parameter of the problem statement.

Determining implementability. A local implementation can only gain information about the global protocol state by making branching decisions and by receiving messages. Non-implementability arises when a participant's local information, comprising the decisions and observations it has made so far, is insufficient for determining its next action.

Let us start by analyzing the implementability question for the task scheduling protocol, assuming a standard peer-to-peer FIFO network architecture (also referred to as *peer-to-peer box* semantics

in this paper). Throughout the paper, we use $\mathtt{p} \triangleright \mathtt{q}!m$ to denote a send event where participant $\mathtt{p}$ sends m to $\mathtt{q}$. Likewise, we use $\mathtt{q} \triangleleft \mathtt{p}?m$ to denote an event where $\mathtt{q}$ receives message m that was previously sent from $\mathtt{p}$.

Under a peer-to-peer box network, the protocol is not implementable: $\mathtt{w}_2$ cannot distinguish between a run that follows branches 1 and a and a run that follows branches 3 and c. In both cases, $\mathtt{w}_1$ may find itself in the same local state q where only the half message from $\mathtt{w}_1$ is available in its associated channel buffer (i.e., in the $3c$ run, the half message from $\mathtt{s}$ to $\mathtt{w}_1$ may be delayed). If the protocol is following the $1a$ run, $\mathtt{w}_2$'s next action should be to send a reply to $\mathtt{s}$. However, in the $3c$ run it should first wait for the arrival of half from $\mathtt{s}$. If $\mathtt{w}_2$ were to wait in state q, this would lead to a deadlock in the $1a$ run and if it were to send to $\mathtt{s}$, it would violate protocol fidelity in the $3c$ run.

Perhaps surprisingly, replacing the peer-to-peer box network by another asynchronous network architecture does not resolve this problem. The reason for non-implementability solely depends on the asynchronous nature of communication and the fact that the two send events $\mathtt{s} \triangleright \mathtt{w}_2!\mathsf{half}$ and $\mathtt{w}_1 \triangleright \mathtt{w}_2!\mathsf{half}$ in the $3a$ run do not causally depend on each other. They can therefore happen concurrently, causing the two messages to arrive at $\mathtt{w}_2$ in any order. Thus, the protocol is non-implementable for any asynchronous network architecture.

However, in general, implementability depends on the specific network architecture. For example, consider a possible repair of the global specification that replaces the message value half of the send from $\mathtt{w}_1$ to $\mathtt{w}_2$ on branch $1a$ with delegate. Now, $\mathtt{w}_2$ can tell the two branches apart: it can wait until either half is available in its buffer from $\mathtt{s}$, indicating that the protocol follows branch 3, or delegate is available in the buffer from $\mathtt{w}_2$, indicating that the other participants chose to follow branch $1a$. Since the two cases are exclusive, $\mathtt{w}_2$ can make its decision as soon as it observes one of the two. This change renders the protocol implementable under peer-to-peer box semantics.

On the other hand, this repair does not help for the mailbox network architecture, in which all messages sent to the same recipient are collected in one FIFO buffer. The issue with mailbox is that in the $3c$ branch, the network may still asynchronously reorder the two messages sent to $\mathtt{w}_2$ by delaying the message from $\mathtt{s}$. Since messages are buffered in FIFO order of arrival, this would force $\mathtt{w}_2$ to first receive the message from $\mathtt{w}_1$ before being able to retrieve the one from $\mathtt{s}$ in the buffer. The resulting sequence of events would violate protocol fidelity. Note that this time, the ensuing violation of protocol fidelity has nothing to do with incomplete information by any of the protocol participants about what branch the protocol is following. Instead, it is solely due to the ability of the network architecture to reorder independent events in executions of individual protocol runs. A possible repair of the protocol for this architecture is to introduce a causal dependency between $\mathtt{s} \triangleright \mathtt{w}_2!\mathsf{half}$ and $\mathtt{w}_1 \triangleright \mathtt{w}_2!\mathsf{half}$, e.g., by inserting an additional message exchange $\mathtt{s} \triangleright \mathtt{w}_1!\mathsf{done}$ between the two exchanges, forcing $\mathtt{w}_1$ to wait until $\mathtt{s}$ has sent its message to $\mathtt{w}_2$.

Finally, if we swap the network architecture from mailbox to *mailbag*, where the single FIFO queue per recipient is replaced by an unordered multiset, the protocol becomes implementable again even without the proposed second repair for mailbox.

3 Preliminaries

Words. Let Σ be an alphabet. Σ^* denotes the set of finite words over Σ, Σ^ω the set of infinite words, and Σ^∞ their union $\Sigma^* \cup \Sigma^\omega$. A word $u \in \Sigma^*$ is a *prefix* of word $v \in \Sigma^\infty$, denoted $u \preceq v$, if there exists $w \in \Sigma^\infty$ with $u \cdot w = v$; we denote all prefixes of u with $\mathrm{pref}(u)$. We sometimes omit the concatenation symbol $\cdot$, instead writing $uw = v$. Given a word $w = w_0 \dots w_n$, we use $w[i]$ to denote the i-th symbol $w_i \in \Sigma$, and $w[0..i]$ to denote the subword between and including w_0 and w_i, i.e. $w_0 \dots w_i$.

Message Alphabets. Let $\mathcal{P}$ be a (possibly infinite) set of participants and $\mathcal{V}$ be a (possibly infinite) data domain. We define the set of *synchronous events* $\Gamma_{sync} \coloneqq \{\mathsf{p} \to \mathsf{q} : m \mid \mathsf{p}, \mathsf{q} \in \mathcal{P} \text{ and } m \in \mathcal{V}\}$ where $\mathsf{p} \to \mathsf{q} : m$ denotes a message exchange of m from sender p to receiver q. For a participant $\mathsf{p} \in \mathcal{P}$, we define the alphabet $\Gamma_{\mathsf{p}} = \{\mathsf{p} \to \mathsf{q} : m \mid \mathsf{q} \in \mathcal{P},\ m \in \mathcal{V}\} \cup \{\mathsf{q} \to \mathsf{p} : m \mid \mathsf{q} \in \mathcal{P},\ m \in \mathcal{V}\}$, and a homomorphism $\Downarrow_{\Gamma_{\mathsf{p}}}$, where $x\Downarrow_{\Gamma_{\mathsf{p}}} = x$ if $x \in \Gamma_{\mathsf{p}}$ and ε otherwise. For a participant $\mathsf{p} \in \mathcal{P}$, we define the alphabet $\Sigma_{\mathsf{p},!} = \{\mathsf{p} \triangleright \mathsf{q}!m \mid \mathsf{q} \in \mathcal{P},\ m \in \mathcal{V}\}$ of *send* events and the alphabet $\Sigma_{\mathsf{p},?} = \{\mathsf{p} \triangleleft \mathsf{q}?m \mid \mathsf{q} \in \mathcal{P},\ m \in \mathcal{V}\}$ of *receive* events. The event $\mathsf{p} \triangleright \mathsf{q}!m$ denotes participant p sending a message m to q, and $\mathsf{p} \triangleleft \mathsf{q}?m$ denotes participant p receiving a message m from q. We write $\Sigma_{\mathsf{p}} = \Sigma_{\mathsf{p},!} \cup \Sigma_{\mathsf{p},?}$, $\Sigma_{!} = \bigcup_{\mathsf{p} \in \mathcal{P}} \Sigma_{\mathsf{p},!}$, and $\Sigma_{?} = \bigcup_{\mathsf{p} \in \mathcal{P}} \Sigma_{\mathsf{p},?}$. Finally, the set of *asynchronous* events is $\Sigma_{async} = \Sigma_{!} \cup \Sigma_{?}$.

Projections. We say that p is *active* in $x \in \Sigma_{async}$ if $x \in \Sigma_{\mathsf{p}}$. For each participant $\mathsf{p} \in \mathcal{P}$, we define a homomorphism $\Downarrow_{\Sigma_{\mathsf{p}}}$, where $x\Downarrow_{\Sigma_{\mathsf{p}}} = x$ if $x \in \Sigma_{\mathsf{p}}$ and ε otherwise. We define a class of projections based on pattern-matching of alphabet symbols, denoted $\Downarrow_{\_}$. The result of the projection is determined by the unspecified parts of the pattern. For example, $\Downarrow_{\mathsf{p}\triangleright\_!\_}$ projects the symbol $\mathsf{p} \triangleright \mathsf{q}!m$ onto (q, m), and non-send symbols and send-symbols that do not have p as the sender onto ε. The function $\Downarrow_{\mathsf{q}\triangleleft\mathsf{p}?\_}$ projects receive events of p from q of any message value onto the message value, and all other events to ε.

We adopt labeled transition systems over the synchronous alphabet Γ_{sync} as our starting point for specifying global protocols.

Labeled Transition Systems. A *labeled transition system* (LTS) is a tuple $\mathcal{S} = (S, \Gamma, T, s_0, F)$ where S is a set of states, Γ is a set of labels, T is a set of transitions from $S \times \Gamma \times S$, $F \subseteq S$ is a set of final states, and $s_0 \in S$ is the initial state. We use $p \xrightarrow{\alpha} q$ to denote the transition $(p, \alpha, q) \in T$. Runs and traces of an LTS are defined in the expected way. A run is *maximal* if it is either finite and ends in a final state, or is infinite. The language of an LTS $\mathcal{S}$, denoted $\mathcal{L}(\mathcal{S})$, is defined as the set of maximal traces. A state $s \in S$ is a *deadlock* if it is not final and has no outgoing transitions. An LTS is *deadlock-free* if no reachable state is a deadlock.

In [55], LTS over Γ_{sync} are constrained with three additional conditions, to yield a fragment called *global communicating labeled transition systems*, hereafter GCLTS. The three GCLTS assumptions are sink finality, sender-driven choice, and deadlock freedom. Sink finality is a purely syntactic condition enforcing that final states have no outgoing transitions. Sender-driven choice states that from any state, all outgoing transitions share a unique sender, and moreover are deterministic. That is, at any branching point in the protocol, there is a unique sender that decides which branch is taken. It was shown in [59, 76] that forgoing sender-driven choice leads to undecidability of implementability, though there has been recent interest in mixed choice for synchronous protocol fragments [73]. Deadlock freedom requires that all reachable states are either final, or have outgoing transitions.

Our task scheduling example (Fig. 1) satisfies all GCLTS assumptions. In particular, to see that it satisfies sender-driven choice, observe that s chooses among the branches $\{1, 2, 3\}$ and w_1 chooses among $\{a, b\}$ and $\{c, d\}$, respectively.

4 Implementability modulo network architectures

In this section, we present network-parametric definitions of distributed implementations, global protocol semantics, and finally the implementability problem.

Our implementation model is based on communicating state machines (CSMs) [8]. CSMs consist of a collection of finite state machines, one for each participant, that communicate via pairwise FIFO channels. We generalize CSMs along two key dimensions: the communication topology, and

the data structure for message buffers. We also lift the restriction imposed by CSMs that the number of participants and the state spaces of the local state machines must be finite, following [55, 58].

Definition 4.1 (Network architecture). A *network architecture* over a set of participants $\mathcal{P}$ and message values $\mathcal{V}$ is a tuple $\mathbb{A} = (\mathsf{C}, \mathsf{B}, \mathsf{ch}, \mathit{ins}, \mathit{rem}, b_0)$ where C is a set of channels, B a set of *channel contents* and $\mathsf{ch} : \mathcal{P} \times \mathcal{P} \to \mathsf{C}$ a map that associates each sender and receiver with a channel. Intuitively, $\mathsf{ch}(\mathsf{p}, \mathsf{q})$ denotes the message buffer to which messages sent from p to q are deposited. We refer to ch as the *communication topology* of $\mathbb{A}$. The set of *channel states* is $\mathbb{X} = \mathsf{C} \to \mathsf{B}$.

Messages $x \in \mathcal{M}$ in channel contents are tagged with their sender and receiver, i.e., $\mathcal{M} = \mathcal{P} \times \mathcal{P} \times \mathcal{V}$. Channel contents are equipped with partial insert and remove operations, $\mathit{ins}, \mathit{rem} : \mathcal{M} \to \mathsf{B} \to \mathsf{B}$, where $\mathit{ins}(x)(b)$ being undefined indicates that b blocks on inserting x and $\mathit{rem}(x)(b)$ is only defined when x is available for removal in b. Finally, $b_0 \in \mathsf{B}$ is the empty channel contents.

Example 4.2. We define eight concrete network architectures that we will revisit later. We consider four communication topologies: n-to-n, in which all senders and receivers share the same channel, one-to-n, in which receivers share the same channel to receive from a single sender, n-to-one, in which senders share the same channel to send to a single receiver, and one-to-one, in which each sender and receiver pair have a unique channel. We consider two message buffer data structures: ordered FIFO queues, and unordered multisets. The aforementioned network architectures often appear under the names of global bus (n-to-n FIFO), message soup (one-to-one multiset), and mailbox (n-to-one FIFO). Message soups are commonly found in leader election protocols such as Paxos and Raft, and one-to-n FIFO is found in work stealing patterns in parallel programming. Finally, the one-to-one or peer-to-peer FIFO is the standard network architecture for CSMs, and widely assumed in the theory and practice of message-passing concurrency.

The four communication topologies are defined as follows (with our naming conventions given in parenthesis, where "B" refers to the name of one of the buffer types below):

- n-to-n (peer-to-peer B): $\mathsf{C} = \mathcal{P} \times \mathcal{P}$, $\mathsf{ch}(\mathsf{p}, \mathsf{q}) = (\mathsf{p}, \mathsf{q})$,
- one-to-n (mailB): $\mathsf{C} = \mathcal{P}$, $\mathsf{ch}(\mathsf{p}, \mathsf{q}) = \mathsf{p}$,
- n-to-one (senderB): $\mathsf{C} = \mathcal{P}$, $\mathsf{ch}(\mathsf{p}, \mathsf{q}) = \mathsf{q}$,
- one-to-one (monoB): $\mathsf{C} = \{0\}$, $\mathsf{ch}(\mathsf{p}, \mathsf{q}) = 0$,

and the two buffer types are FIFO queues (B=box), $\mathsf{B} = \mathcal{M}^*$, and multisets (B=bag), $\mathsf{B} = \mathcal{M} \to \mathbb{N}$. In the case of FIFO queues, *insert* corresponds to appending at the end of the queue, *remove* corresponds to removing from the head; in the case of multisets, *insert* is multiset addition, and *remove* multiset deletion. The empty buffer contents b_0 is ε in the case of FIFO queues and $\emptyset$ in the case of multiset buffers.

We note that the four network architectures with homogeneous bag channels are operationally equivalent and collapse to the monobag case: since messages are all labeled with their sender and receiver, one can "on demand" separate the message soup into $\mathcal{P}^2$ multisets, or $\mathcal{P}$ multisets by sender or receiver whenever messages are sent or received, and thus simulate the other network architectures. This leaves us with the four FIFO network architectures, which we refer to as `p2p`, `mb`, `sb`, and `monob` in the rest of the paper, and the collapsed case for bag channels (`bag`).

We next present communicating labeled transition systems parametric in a choice of network architecture $\mathbb{A}$. For the following definitions, we fix a network architecture $\mathbb{A} = (\mathsf{C}, \mathsf{B}, \mathsf{ch}, \mathit{ins}, \mathit{rem}, b_0)$. We lift the *insert* and *remove* operations to channel states $\xi \in \mathbb{X}$ of $\mathbb{A}$ by defining $\mathit{insert}(\xi, \mathsf{p}, \mathsf{q}, m) = \xi'$ where $\xi'(\mathsf{ch}(\mathsf{p}, \mathsf{q})) = \mathit{ins}(\mathsf{p}, \mathsf{q}, m)(\xi(\mathsf{ch}(\mathsf{p}, \mathsf{q})))$ and all other channel contents remain unchanged; $\mathit{remove}(\xi, \mathsf{p}, \mathsf{q}, m)$ is defined analogously.

Definition 4.3 (Network-parametric CLTS). $\mathcal{T}_{\mathbb{A}} = \{\!\{T_{\mathsf{p}}\}\!\}_{\mathsf{p} \in \mathcal{P}}$ is a *communicating labeled transition system* (CLTS) over $\mathcal{P}$, $\mathcal{V}$ and $\mathbb{A}$ if T_{p} is a deterministic LTS over Σ_{p} for every $\mathsf{p} \in \mathcal{P}$, denoted by

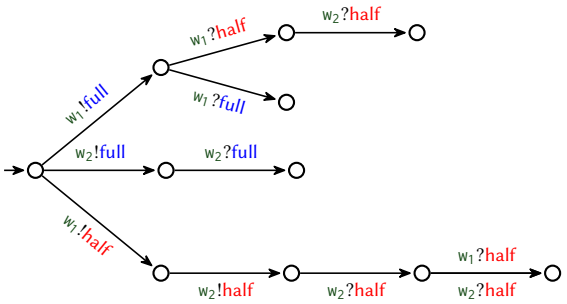

Fig. 2. Local implementation for s

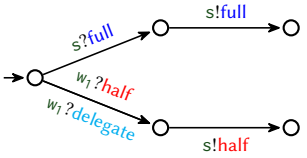

Fig. 3. Local implementation for w2

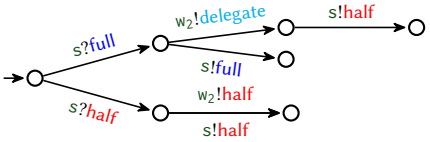

Fig. 4. Local implementation for w1

$(Q_\mathsf{p}, \Sigma_\mathsf{p}, \delta_\mathsf{p}, q_{0,\mathsf{p}}, F_\mathsf{p})$. Let $\prod_{\mathsf{p}\in\mathcal{P}} Q_\mathsf{p}$ denote the set of global states. A *configuration* of $\mathcal{T}_\mathbb{A}$ is a pair $(\vec{s}, \xi)$, where $\vec{s}$ is a global state and $\xi \in \mathbb{X}$ is a channel state. We use $\vec{s}_\mathsf{p}$ to denote the state of p in $\vec{s}$. The CLTS transition relation, denoted $\longrightarrow$, is defined as follows.

- $(\vec{s}, \xi) \xrightarrow{\mathsf{p}\triangleright\mathsf{q}!m} (\vec{s}', \xi')$ if $(\vec{s}_\mathsf{p}, \mathsf{p} \triangleright \mathsf{q}!m, \vec{s}'_\mathsf{p}) \in \delta_\mathsf{p}$, $\vec{s}_\mathsf{r} = \vec{s}'_\mathsf{r}$ for every participant $\mathsf{r} \neq \mathsf{p}$, $\xi' = \mathit{insert}(\xi, \mathsf{p}, \mathsf{q}, m)$.
- $(\vec{s}, \xi) \xrightarrow{\mathsf{q}\triangleleft\mathsf{p}?m} (\vec{s}', \xi')$ if $(\vec{s}_\mathsf{q}, \mathsf{q} \triangleleft \mathsf{p}?m, \vec{s}'_\mathsf{q}) \in \delta_\mathsf{q}$, $\vec{s}_\mathsf{r} = \vec{s}'_\mathsf{r}$ for every participant $\mathsf{r} \neq \mathsf{q}$, $\xi' = \mathit{remove}(\xi, \mathsf{p}, \mathsf{q}, m)$.

In the initial configuration $(\vec{s}_0, \xi_0)$, each participant's state in $\vec{s}_0$ is the initial state $q_{0,\mathsf{p}}$ of A_p, and ξ_0 maps each channel to the empty buffer b_0. A configuration $(\vec{s}, \xi)$ is *final* iff $\vec{s}_\mathsf{p}$ is final for every p and $\xi = \xi_0$. Runs and traces are defined in the expected way. A run is *maximal* if either it is finite and ends in a final configuration, or it is infinite. The language $\mathcal{L}(\mathcal{T}_\mathbb{A})$ of the CLTS $\mathcal{T}_\mathbb{A}$ is defined as the set of maximal traces, and $\mathrm{pref}(\mathcal{L}(\mathcal{T}_\mathbb{A}))$ is defined as the set of prefixes of maximal traces. A configuration $(\vec{s}, \xi)$ is a *deadlock* if it is not final and has no outgoing transitions. A CLTS is *deadlock-free* if no reachable configuration is a deadlock.

The local implementations for participants s, w_1 and w_2 of the repaired task scheduling protocol from §2 are depicted in Fig. 2, Fig. 4 and Fig. 3 respectively. Note that the active participants are omitted from transition labels for clarity.

Channel compliance. Before we define the semantics of global protocols, it is useful to disentangle the behavior of a given network architecture $\mathbb{A}$ from that of any particular CLTS $\mathcal{T}_\mathbb{A}$. To this end, we introduce the semantic notion of channel compliance that describes all words w that are consistent with $\mathbb{A}$. The channel-compliant words are those words that are traces of the universal CLTS, which lifts all constraints on traces imposed by participants' local states. Formally, let $\mathcal{U}_\mathbb{A}$ be the *universal CLTS* for $\mathbb{A}$, with $\mathcal{U}_\mathbb{A} = \{\!\{U_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ such that $\mathcal{L}(U_\mathsf{p}) = \Sigma_\mathsf{p}^*$ for every $\mathsf{p} \in \mathcal{P}$. We say that $w \in \Sigma_{\mathit{async}}^*$ is $\mathbb{A}$-channel-compliant if w is a trace of $\mathcal{U}_\mathbb{A}$. If $\mathbb{A}$ is understood, we just say w is channel-compliant. Moreover, if in fact $w \in \mathcal{L}(\mathcal{U}_\mathbb{A})$ holds, then we call w *channel-matched*. Intuitively, if one can execute a channel-matched word in any given CLTS for $\mathbb{A}$, then in the reached configuration all buffers will be empty. We use $\mathcal{L}(\mathbb{A}) \subseteq \Sigma_{\mathit{async}}^*$ to denote all channel-compliant words.

Equality under local projection. We say that w_1 and w_2 are equal under local projection, denoted $w_1 \equiv_\mathcal{P} w_2$, if for all p, $w_1{\Downarrow_{\Sigma_\mathsf{p}}} = w_2{\Downarrow_{\Sigma_\mathsf{p}}}$. We use $[w]_{\equiv_\mathcal{P}}$ to denote the equivalence class under local projection with representative w. We lift this to sets $W \subseteq \Sigma_{\mathit{async}}^\infty$, by defining $[W]_{\equiv_\mathcal{P}} = \bigcup_{w\in W} [w]_{\equiv_\mathcal{P}}$.

Global protocol semantics. We define the asynchronous semantics of a global protocol $\mathcal{S}$, denoted $\mathcal{L}_{\mathbb{A}}(\mathcal{S}) \subseteq \Sigma^{\infty}_{async}$, modulo a choice of network architecture $\mathbb{A}$. Recall that $\mathcal{S}$ is an LTS over the alphabet Γ_{sync}. The starting point for the semantics $\mathcal{L}_{\mathbb{A}}(\mathcal{S})$ is the synchronous language $\mathcal{L}(\mathcal{S})$. From $\mathcal{L}(\mathcal{S})$ we can obtain a set of 1-synchronous asynchronous words through `split`, which simply splits each atomic send and receive event into its two counterparts, denoted $\texttt{split}(\mathcal{L}(\mathcal{S}))$. We want to include all asynchronous words that are equal to these 1-synchronous words under local projection and the given network architecture $\mathbb{A}$.

We handle the finite and infinite words separately to define the global protocol semantics as the union of its finite and infinite semantics:

$$\mathcal{L}_{\mathbb{A}}(\mathcal{S}) = \mathcal{L}^{\text{fin}}_{\mathbb{A}}(\mathcal{S}) \cup \mathcal{L}^{\text{inf}}_{\mathbb{A}}(\mathcal{S})$$

Following the above recipe and restricting $\texttt{split}(\mathcal{L}(\mathcal{S}))$ to finite words yields the finite semantics:

$$\mathcal{L}^{\text{fin}}_{\mathbb{A}}(\mathcal{S}) = [\Sigma^{*}_{async} \cap \texttt{split}(\mathcal{L}(\mathcal{S}))]_{\equiv_{\mathcal{P}}} \cap \mathcal{L}(\mathbb{A}) \ .$$

The infinite semantics are those words whose prefixes are extensible to some word in $\mathcal{L}(\mathcal{S})$ modulo equality under local projection and the network semantics:

$$\mathcal{L}^{\text{inf}}_{\mathbb{A}}(\mathcal{S}) = \{w \in \Sigma^{\infty}_{async} \mid \forall u \leq w.\ u \in \text{pref}([\texttt{split}(\mathcal{L}(\mathcal{S}))]_{\equiv_{\mathcal{P}}} \cap \mathcal{L}(\mathbb{A}))\} \ .$$

For disambiguation, we refer to $\mathcal{L}(\mathcal{S}) \subseteq \Gamma^{\omega}_{sync}$ as the *LTS semantics* of $\mathcal{S}$, and refer to $\mathcal{L}_{\mathbb{A}}(\mathcal{S}) \subseteq \Sigma^{\omega}_{async}$ as the *protocol semantics* of $\mathcal{S}$.

Example 4.4. To illustrate our protocol semantics, consider the following protocol, which contains both finite and infinite words in its semantics: $\mathtt{p} \to \mathtt{q} : m^{\infty} + (\mathtt{p} \to \mathtt{q} : m)^{*} \cdot \mathtt{r} \to \mathtt{q} : m$. The synchronous runs of the protocol are either of the form $\mathtt{p} \to \mathtt{q} : m^{\infty}$, or of the form $(\mathtt{p} \to \mathtt{q} : m)^{n} \cdot \mathtt{r} \to \mathtt{q} : m$. The `split` runs are subsequently of the form $(\mathtt{p} \triangleright \mathtt{q}!m \cdot \mathtt{q} \triangleleft \mathtt{p}?m)^{\infty}$ or $(\mathtt{p} \triangleright \mathtt{q}!m \cdot \mathtt{q} \triangleleft \mathtt{p}?m)^{n} \cdot \mathtt{r} \triangleright \mathtt{q}!m \cdot \mathtt{q} \triangleleft \mathtt{r}?m$. Because our infinite word semantics do not impose any fairness assumptions, the unfairly scheduled word $\mathtt{p} \triangleright \mathtt{q}!m^{\infty}$ is part of the protocol's infinite semantics. The word $\mathtt{r} \triangleright \mathtt{q}!m \cdot \mathtt{p} \triangleright \mathtt{q}!m \cdot \mathtt{q} \triangleleft \mathtt{p}?m \cdot \mathtt{q} \triangleleft \mathtt{r}?m$ is part of the protocol's finite semantics under p2p, where the network reorders the send events from p and r, but q receives in the specified protocol order, first from p and then from r.

We are now ready to define the network-parametric implementability problem:

Definition 4.5 (Network-parametric Protocol Implementability). A protocol $\mathcal{S}$ is *implementable* under network architecture $\mathbb{A}$ if there exists a CLTS $\mathcal{T}_{\mathbb{A}} = \{\!|T_{\mathtt{p}}|\!\}_{\mathtt{p} \in \mathcal{P}}$ such that the following two properties hold: (i) *protocol fidelity*: $\mathcal{L}(\{\!|T_{\mathtt{p}}|\!\}_{\mathtt{p} \in \mathcal{P}}) = \mathcal{L}_{\mathbb{A}}(\mathcal{S})$, and (ii) *deadlock freedom*: $\{\!|T_{\mathtt{p}}|\!\}_{\mathtt{p} \in \mathcal{P}}$ is deadlock-free. We say that $\{\!|T_{\mathtt{p}}|\!\}_{\mathtt{p} \in \mathcal{P}}$ implements $\mathcal{S}$ under $\mathbb{A}$.

5 Characterization of generalized implementability

In this section, we present our sound and complete network-parametric implementability characterization. We take as a starting point a recently proposed precise characterization for p2p implementability [55]. We illuminate key abstract assumptions about the protocol semantics and implementation model made by the characterization in [55] that enable its soundness and completeness proofs. In a process analogous to computing weakest pre-conditions, we distill, and in some cases weaken these abstract assumptions, in tandem with developing our network-parametric characterization. Ultimately, we obtain a network-parametric set of conditions, that we call Generalized Coherence Conditions, along with a set of abstract assumptions that, when satisfied, render our characterization sound and complete with respect to implementability.

The implementability characterization of [55] takes the form of three Coherence Conditions (*CC*) that a global protocol must satisfy. The Coherence Conditions are 2-hyperproperties that scrutinize pairs of global protocol states from which a participant can perform different actions,

but whose distinction may not be locally observable to the participant. *CC* describes the kinds of local actions that are safe to perform in this state of unawareness. *Send Coherence* says that if a participant has the option to perform a send action from one state, it must have the option to perform the same send action from any indistinguishable state. *Receive Coherence* says that if a participant has the option to perform a receive action from one state, then this same receive action could not possibly be performed from any other indistinguishable state. *No Mixed Choice* says that a participant cannot equivocate between performing a send and receive action.

Formally, a global protocol state s is reachable for p on $u \in \Gamma_{\mathsf{p}}^*$ when there exists $w \in \Gamma_{sync}^*$ such that s is reachable with trace w and $w{\Downarrow_{\Gamma_{\mathsf{p}}}} = u$. If two global protocol states s_1 and s_2 are both reachable for p on the same $u \in \Gamma_{\mathsf{p}}^*$, we call them *simultaneously reachable*. Simultaneous reachability captures pairs of states that are locally indistinguishable to a participant.

Li et al. [55] show that *CC* is sound by invoking a *canonical implementation*, which serves as the witness to implementability.

Definition 5.1 (Canonical implementations [55]). A CLTS $\{\!|T_{\mathsf{p}}|\!\}_{\mathsf{p}\in\mathcal{P}}$ is a *canonical implementation* for a protocol $\mathcal{S} = (S, \Gamma_{sync}, T, s_0, F)$ if for every $\mathsf{p} \in \mathcal{P}$, T_{p} satisfies:
(i) $\forall w \in \Sigma_{\mathsf{p}}^*.\ w \in \mathcal{L}(T_{\mathsf{p}}) \Leftrightarrow w \in \mathcal{L}(\mathcal{S}){\Downarrow_{\Sigma_{\mathsf{p}}}}$, and (ii) $\mathrm{pref}(\mathcal{L}(T_{\mathsf{p}})) = \mathrm{pref}(\mathcal{L}(\mathcal{S}){\Downarrow_{\Sigma_{\mathsf{p}}}})$.

As observed in [58], canonicity can be defined directly in terms of a global protocol's LTS semantics, since local projections are oblivious to asynchronous reorderings. As discussed in [55] and formalized in [58], canonical implementations can always be constructed using a generalized subset construction, and synthesis is separate from considerations of network architecture for implementability. We discuss synthesis further in §10.

The key technical argument for soundness lies in showing that the canonical implementation's language is a subset of global protocol semantics, and is deadlock-free. This requires showing that every canonical implementation trace can be associated with a run in the global protocol that each participant has partially completed the prescribed actions of. This in turn is shown by induction on canonical implementation traces, appealing to *CC* to argue that the extension by either a send or receive event retains the existence of a global run that can be associated with the resulting trace.

Completeness of *CC* is established in [55] via modus tollens: from the negation of each Coherence Condition a trace is constructed that is compliant with no protocol run, yet must be admitted by any candidate implementation of the protocol. This suffices for an implementability violation, because either the trace leads to a deadlock, or to a maximal word not in the global protocol semantics.

Revisiting Send Coherence and No Mixed Choice. These two Coherence Conditions remain sound and complete for network-parametric implementability in their original form. However, their soundness and completeness proofs still need to be generalized to the network-parametric case.

Towards generalizing the completeness proofs of Send Coherence and No Mixed Choice, we observe that the witness constructed by the completeness proof in [55] can be adapted to one that is the prefix of a 1-synchronous trace. Concretely, the witnesses for both conditions assume the form of $\mathtt{split}(\alpha) \cdot x$, where α is a synchronous word and x is a send event. The proof then shows that $\mathtt{split}(\alpha) \cdot x$ is channel-compliant, and together with the fact that it must be executable by any candidate implementation, thus consistutes a counterexample to implementability. The generalization thus first requires the assumption that send transitions are always enabled in the network architecture under consideration. We formalize this assumption as follows:

$$\forall w \in \Sigma_{async}^*, x \in \Sigma_!.\ w \text{ is channel-compliant} \implies wx \text{ is channel-compliant.} \tag{F1}$$

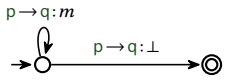

Fig. 5. A protocol $\mathcal{S}_a$ that is not implementable on a bag network but implementable on a peer-to-peer box network.

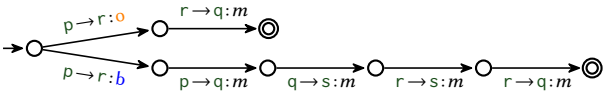

Fig. 6. A protocol $\mathcal{S}_b$ that is not implementable on a peer-to-peer box network but implementable on a senderbox network.

To complete the network-parametric proof, we show that $\texttt{split}(\alpha) \cdot x$ is channel-compliant. This fact follows from F1, provided $\texttt{split}(\alpha)$ is channel-compliant. This yields our second assumption:

$$\forall \alpha \in \Gamma^*_{sync}.\ \texttt{split}(\alpha) \text{ is channel-compliant.} \tag{F3}$$

Intuitively, a reasonable network should allow sends and their receives to be executed consecutively.

The soundness of Send Coherence seeks to establish that any send extension by the canonical implementation remains a protocol prefix. It turns out that this fact depends on both No Mixed Choice and Receive Coherence. Next, we discuss how to generalize Receive Coherence.

Revisiting Receive Coherence. In contrast to send events, receive events are conditioned on transitions as well as the availability of the message in question to be received. Message availability in turn highly depends on the network architecture, and requires considering possible asynchronous reorderings, as evidenced by participant $\texttt{w}_2$'s plight in the task delegation protocol from §2.

Consider the protocols $\mathcal{S}_a$ and $\mathcal{S}_b$, depicted in Fig. 5 and Fig. 6, both from the perspective of the receiver q. In $\mathcal{S}_a$ under a monobag network architecture, q's bag can contain any number of m messages from p, in addition to a $\bot$ message. Participant q can never know when it is safe to receive the $\bot$ message, and thus the protocol is non-implementable. If we replace the monobag network with a peer-to-peer box network, the final $\bot$ message is ordered after all m messages have been sent, and thus participant q can always receive the message at the head of its FIFO queue from p. In $\mathcal{S}_b$ under a peer-to-peer box network, when message m from q is available to receive, the same message m could also be available from p simultaneously, leading q to a protocol violation. If we replace the peer-to-peer box network with a senderbox network, this ambiguity again goes away: r's message to s effectively blocks the message to q from being available to receive, thus a unique message is available for q to receive regardless of the branch selected by p.

To generalize Receive Coherence, we first make the following key observation about all implementations of global protocols. If at any point during the implementation's execution, a participant has the choice between receiving two different messages, then one of these choices must constitute a protocol violation. Because protocol runs totally order the events of all participants, there exists no protocol run that is compliant with both choices of receptions. Assume that for each choice of reception, there exists a protocol run that is compliant with the resulting execution trace. Due to the sender-driven nature of global protocols, for any two runs of a global protocol, the first event along which they differ must share a common sender. The fact that the receiver can choose between two receptions means it is oblivious to the difference between the corresponding compliant runs, and their choice thus forces the sender to commit to one of two branches, a coordination between participants that is impossible since either the send event causally precedes the receive event, or the send event is independent of the receive event. Indeed, the soundness proof of Receive Coherence in [55] argues that all receive extensions to the canonical implementation's traces are unique, and the completeness proof of Receive Coherence constructs a counterexample to implementability from the existence of an implementation prefix that admits two different receive extensions.

It follows from this key observation that irrespective of network architecture, it is sufficient and necessary to guarantee that no two distinct receptions are enabled for a given participant from any

configuration. We then capture this simpler property in a network-parametric way. The notion of m being receivable by $\mathtt{q}$ from $\mathtt{p}$ in some configuration after executing a word w is captured simply as the channel compliance of $w \cdot \mathtt{q} \triangleleft \mathtt{p}?m$. We desire two more constraints on w: first, the sending of m by $\mathtt{p}$ should not depend on any events by $\mathtt{q}$, and thus we require that $w{\Downarrow_{\Sigma_\mathtt{q}}} = \varepsilon$, and second, a different message must also have been sent to $\mathtt{q}$ in w: either the sender or the message value differ.

This leads to our Generalized Receive Coherence condition, defined as follows:

Definition 5.2 (Generalized Receive Coherence). A protocol $\mathcal{S}$ satisfies Generalized Receive Coherence when for every two simultaneously reachable by $\mathtt{q}$ states s_1, s_1' with transitions $s_1 \xrightarrow{\mathtt{p}\to\mathtt{q}:m} s_2$ and $s_1' \xrightarrow{\mathtt{r}\to\mathtt{q}:m'} s_2'$, if $\mathtt{r} \neq \mathtt{p}$ or $m' \neq m$, then for any $w \in \mathrm{pref}(\mathcal{L}(\mathcal{S}_{s_1'}))$ such that $w{\Downarrow_{\Sigma_\mathtt{q}}} = \varepsilon$ and $\mathtt{r} \triangleright \mathtt{q}!m' \leq w{\Downarrow_{\Sigma_\mathtt{r}}}$, the extension $w \cdot \mathtt{q} \triangleleft \mathtt{p}?m$ is not channel-compliant.

We highlight two differences from Receive Coherence for p2p from [55]: w is required to be a protocol prefix for the protocol reinitialized at s_1' instead of s_2', and we require an additional conjunct that $\mathtt{r} \triangleright \mathtt{q}!m'$ is the first event for participant $\mathtt{r}$ in w. It is easy to see that for p2p channel compliance, the two RHSs are equivalent, because messages from $\mathtt{r}$ to $\mathtt{q}$ do not share a channel with messages from a different sender, and thus unmatched send events can always be prepended to traces without affecting channel compliance. However, imposing this fact as an assumption would a priori rule out mb and monob network architectures. Thus, the RHS of Generalized Receive Coherence captures that a send from $\mathtt{r}$ to $\mathtt{q}$ exists in w, without committing to where in w it resides, and we formulate the following assumption on channel compliance instead, which we call *history insensitivity*: for all $w \in \Sigma^*_{async}$, $\alpha, \beta \in \Gamma^*_{sync}$, $\mathtt{p} \neq \mathtt{q} \in \mathcal{P}$, $m \in \mathcal{V}$, then

$$\begin{aligned}&(\forall \mathtt{p} \in \mathcal{P}.\ w{\Downarrow_{\Sigma_\mathtt{p}}} \leq \mathtt{split}(\alpha\beta){\Downarrow_{\Sigma_\mathtt{p}}}) \wedge w{\Downarrow_{\Sigma_\mathtt{q}}} = \mathtt{split}(\alpha){\Downarrow_{\Sigma_\mathtt{q}}} \wedge w \cdot \mathtt{q} \triangleleft \mathtt{p}?m \text{ is channel-compliant}\\ &\implies \exists w'.(\forall \mathtt{p} \in \mathcal{P}.\ w'{\Downarrow_{\Sigma_\mathtt{p}}} \leq \mathtt{split}(\beta){\Downarrow_{\Sigma_\mathtt{q}}}) \wedge w'{\Downarrow_{\Sigma_\mathtt{q}}} = \varepsilon \wedge w' \cdot \mathtt{q} \triangleleft \mathtt{p}?m \text{ is channel-compliant} \quad \text{(F6)}\end{aligned}$$

Intuitively, F6 allows an asynchronous trace that is compliant with a a synchronous trace $\alpha\beta$ to selectively forget events from α, resulting in a trace w' that is compliant with β, such that if $\mathtt{q}$ completed all events in α in w, then all messages that were receivable in w remain receivable in w'. F6 is thus sufficient to establish the soundness of Generalized Receive Coherence: from an execution trace extensible with a receive event, we argue that this receive event must be uniquely determined by all compliant protocol runs of the trace thus far, otherwise we use F6 to construct a new trace that contradicts Generalized Receive Coherence.

Prefix Extensibility. The three Coherence Conditions for p2p alone are no longer sufficient to guarantee implementability, even with Generalized Receive Coherence. It turns out that there is a fourth source of non-implementability that never occurs for p2p networks but does occur for other architectures. We capture this source of non-implementability in a new fourth Coherence Condition, coined Prefix Extensibility.

To motivate the new condition, consider the simple straight-line global specification consisting only of the synchronous word $\mathtt{p}_1 \to \mathtt{q}:m \cdot \mathtt{p}_2 \to \mathtt{q}:m$. Despite its simplicity, there does not exist a deadlock-free mb (or monob) CLTS that implements this specification. Any candidate CLTS must exhibit the following deadlocking trace: $\mathtt{p}_2 \triangleright \mathtt{q}!m \cdot \mathtt{p}_1 \triangleright \mathtt{q}!m$. Because the network can reorder $\mathtt{p}_2$'s send event before $\mathtt{p}_1$'s send event, yet the local actions of $\mathtt{q}$ tell it to receive in the opposite order, the only way for the CLTS to not deadlock is for $T_\mathtt{q}$ to admit the local trace $\mathtt{q} \triangleleft \mathtt{p}_2?m \cdot \mathtt{q} \triangleleft \mathtt{p}_1?m$.

To rule out such cases of non-implementability, Prefix Extensibility requires that protocol prefixes are closed under per-participant equality.

Definition 5.3 (Prefix Extensibility). A protocol $\mathcal{S}$ satisfies Prefix extensibility when for every protocol trace $\rho \in \mathrm{pref}(\mathcal{L}(\mathcal{S}))$ and $w \in \Sigma^*_{async}$ such that w is $\mathbb{A}$-channel-compliant and agrees with ρ, there exists $u \in \Sigma^*_{async}$ such that wu is channel compliant, and $wu \equiv_{\mathcal{P}} \mathtt{split}(\rho)$.

Note that the protocol above violates Prefix Extensibility: let $\rho = \mathsf{p}_1 \to \mathsf{q} : m \cdot \mathsf{p}_2 \to \mathsf{q} : m$ and $w = \mathsf{p}_2 \triangleright \mathsf{q}!m$, then w agrees with ρ and is mb channel-compliant. However, all extensions of w that are per-participant equal to $\mathtt{split}(\rho)$ are not mb channel-compliant because they deadlock.

Generalized Coherence Conditions. Our generalized characterization of implementability is thus defined as the conjunction of four conditions.

Definition 5.4 (Generalized Coherence Conditions). A protocol $\mathcal{S}$ satisfies Generalized Coherence Conditions under network architecture $\mathbb{A}$ when it satisfies Send Coherence, Generalized Receive Coherence, No Mixed Choice, and Prefix Extensibility.

We present the three remaining assumptions we impose on the network architecture that are required for our preciseness proof. We refer to the collection of six assumptions F1 through F6 altogether as *channel compliance facts*, and these together constitute the sufficient conditions under which our network-parametric characterization is sound and complete. Let $w \in \Sigma^*_{async}$ be channel-compliant.

- F2 For all $\mathsf{p} \neq \mathsf{q} \in \mathcal{P}$ and $m \in \mathcal{V}$, $|w{\Downarrow_{\mathsf{p}\triangleright\mathsf{q}!m}}| \geq |w{\Downarrow_{\mathsf{q}\triangleleft\mathsf{p}?m}}|$.
- F4 For all $\rho \in \Gamma^*_{sync}$, if $w \equiv_{\mathcal{P}} \mathtt{split}(\rho)$, then w is channel-matched.
- F5 For all $x \in \Sigma_!$, $y \in \Sigma_?$, if wy is channel-compliant then wxy is channel-compliant.

These basic assumptions are used throughout the proofs. For example, F2 states that messages cannot be received before they were sent.

The following theorem states that our Generalized Coherence Conditions are sound and complete with respect to implementability, assuming a network architecture that satisfies our channel compliance facts. The proof of Theorem 5.5 is fully mechanized in Rocq, and builds on the Rocq mechanization for p2p networks in [58].

Theorem 5.5 (Preciseness of Generalized Coherence Conditions). *Let $\mathbb{A}$ be a network architecture that satisfies F1 through F6 and let $\mathcal{S}$ be a global protocol. Then, $\mathcal{S}$ is implementable under $\mathbb{A}$ if and only if it satisfies Generalized Coherence Conditions under $\mathbb{A}$.*

6 Reducing channel compliance facts to buffer axioms

With a network-parametric implementability characterization and preciseness proof in place, what remains is determining whether a given network architecture $\mathbb{A}$ is a suitable instance of the framework. Based on our technical development thus far, this amounts to showing that $\mathbb{A}$-channel compliance satisfies the six channel compliance facts assumed in the theorem. However, proving these facts for every network architecture individually is tedious and repetitive at best, and non-trivial at worst: in particular, history insensitivity (F6) requires reasoning globally across events in a word that are not causally ordered either by participants or by channels. As empirical evidence of the proof effort required to prove the channel compliant facts, the Rocq mechanization of F1 through F6 for sb comprises some 5000 lines of code!

We thus take a further step in reducing channel compliance facts to a set of simple, operational buffer data structure specifications that only describe the behavior of operations per individual channel in $\mathbb{A}$. We call these *buffer axioms*, and the set of buffer axioms constitute our axiomatic model of network architectures to which our implementability characterization applies.

In the following, for a network architecture $\mathbb{A}$, we denote by O the set of all operations $ins(x), rem(x) : \mathsf{B} \rightharpoonup \mathsf{B}$ for $x \in \mathcal{M}$. By slight abuse of notation, we identify finite sequences

$\delta = o_1 \dots o_n \in \mathrm{O}^*$ with the partial function $o_n \circ \cdots \circ o_1$ obtained by composing $o_1, \dots, o_n$ in reverse order, and the empty sequence ε with the identity function on B. In other words, δ is identified with the composite operation that captures the cumulative effect of δ's constituent operations. We further write $f \bullet x$ for reverse function application, i.e., $x \bullet f = f(x)$.

Definition 6.1 (Axiomatic network model). We denote by $\mathfrak{A}$ the set of all network architectures $\mathbb{A}$ that satisfy the following properties, for all $b \in \mathrm{B}$, $x, y \in \mathcal{M}$, and $\delta \in \mathrm{O}^*$:

- B1 $b \bullet \mathit{ins}(x)$ is defined.
- B2 $b_0 \bullet \mathit{ins}(x) \bullet \mathit{rem}(x)$ is defined.
- B3 $b_0 \bullet \mathit{rem}(x)$ is undefined.
- B4 If $b \bullet \mathit{rem}(x)$ is defined, then $b \bullet \mathit{ins}(y) \bullet \mathit{rem}(x)$ is defined.
- B5 If $b \bullet \mathit{ins}(x) \bullet \delta$ is defined and $\mathit{rem}(x)$ does not occur in δ, then $b \bullet \delta$ is defined.
- B6 If $b \bullet \mathit{rem}(x)$ is defined, then $b \bullet \mathit{ins}(x) \bullet \mathit{rem}(x) = b \bullet \mathit{rem}(x) \bullet \mathit{ins}(x)$.
- B7 If $x \neq y$, then $b \bullet \mathit{ins}(x) \bullet \mathit{rem}(y) = b \bullet \mathit{rem}(y) \bullet \mathit{ins}(x)$.
- B8 If $b \bullet \mathit{rem}(x)$ is undefined, $b \bullet \mathit{ins}(x) \bullet \delta \bullet \mathit{rem}(x)$ is defined, and $\mathit{rem}(x)$ does not occur in δ, then $b \bullet \mathit{ins}(x) \bullet \delta \bullet \mathit{rem}(x) = b \bullet \delta$.

Intuitively, B1 says that *ins* is total, B2 says that one can always insert and immediately remove the same message from an empty channel, B3 says that one cannot remove from the empty channel, B4 says that adding inserts does not disable subsequent removes, B5 says that unmatched inserts can be omitted, B6 says that removes left-commute with inserts of the same message, provided the message has already been inserted earlier, B7 says that removes left-commute with inserts of different messages, and B8 says that matching inserts and removes cancel each other out.

It is now trivial to observe that FIFO queues and multisets satisfy these axioms. Thus all network architectures of Example 4.2 are in $\mathfrak{A}$. However, some common channel data structures are ruled out. In particular, B1 rules out bounded channels and B8 rules out channels that allow message duplication.

The following lemma states that our network model implies the precondition of Theorem 5.5.

Lemma 6.2. *All $\mathbb{A} \in \mathfrak{A}$ satisfy F1 through F6.*

Corollary 6.3. *Let $\mathbb{A} \in \mathfrak{A}$. Then for all protocols $\mathcal{S}$, $\mathcal{S}$ is implementable under $\mathbb{A}$ if and only if it satisfies Generalized Coherence Conditions.*

The proof of Lemma 6.2 is in Appendix A.1. For example, to show that F3 holds one proves the stronger property that $\mathtt{split}(\alpha)$ is channel-matched by induction on the length of α. In the induction step, one uses B2 to show that the extended word remains channel-compliant, and then B8 to show that it leaves all channels empty, i.e., is channel-matched.

7 Derivatives of Generalized *CC*

In this section, we derive a series of results from our sound and complete characterization of network-agnostic implementability. First, we introduce symbolic protocols featuring dependent refinements. Following the blueprint in [55], we show that Generalized *CC* similarly admits an encoding as least and greatest fixpoints over the symbolic protocol's transition relation. This allows us to define a sound and relatively complete algorithm for deciding implementability of symbolic protocols for the five network architectures considered in Example 4.2. As an interlude, we use our encoding to establish the relationship between classes of implementable protocols for each network architecture. Finally, we present complexity results for finite fragments under the considered architectures.

7.1 Checking Implementability of Symbolic Protocols

Symbolic protocols [55] are defined over a fixed but unspecified first-order background theory of message values (e.g. linear integer arithmetic). We assume standard syntax and semantics of first-order formulas and denote by $\mathcal{F}$ the set of first-order formulas with free variables drawn from an infinite set X. We assume that these variables are interpreted over the set of message values $\mathcal{V}$.

Definition 7.1 (Symbolic protocol [55]). A symbolic protocol is a tuple $\mathbb{S} = (S, R, \Delta, s_0, \rho_0, F)$ where

- S is a finite set of control states,
- R is a finite set of register variables,
- $\Delta \subseteq S \times \mathcal{P} \times X \times \mathcal{P} \times \mathcal{F} \times S$ is a finite set that consists of symbolic transitions of the form $s \xrightarrow{\mathsf{p}\to\mathsf{q}:x\{\varphi\}} s'$ where the formula φ with free variables $R \uplus R' \uplus \{x\}$ expresses a transition constraint that relates the old and new register values (R and R'), and the sent value x,
- $s_0 \in S$ is the initial control state,
- $\rho_0 : R \to \mathcal{V}$ is the initial register assignment, and
- $F \subseteq S$ is a set of final states.

In the remainder of the section, we fix a symbolic protocol $\mathbb{S} = (S, R, \Delta, s_0, \rho_0, F)$ whose concretization satisfies the GCLTS assumptions.

First, we show how to encode Generalized Receive Coherence (GRC) into μCLP. We denote the four conjuncts on the right-hand side of the implication of GRC by $\mathrm{RHS} \coloneqq w \in \mathrm{pref}(\mathcal{L}(\mathcal{S}_{s_1'})) \wedge w{\Downarrow_{\Sigma_\mathsf{q}}} = \varepsilon \wedge \mathsf{r} \triangleright \mathsf{q}!m' \leq w{\Downarrow_{\Sigma_\mathsf{r}}} \wedge w \cdot \mathsf{q} \triangleleft \mathsf{p}?m$ is channel-compliant. Let $s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:m} s_2, s_1' \xrightarrow{\mathsf{r}\to\mathsf{q}:m'} s_2' \in T$ such that s_1, s_2 are simultaneously reachable by q. We first factor GRC into two conditions by doing case analysis on its precondition. Let GRC(A) denote $(\mathsf{r} \neq \mathsf{p} \implies \neg\, \exists\, w.\ \mathrm{RHS}(w))$, and let GRC(B) denote $(\mathsf{r} = \mathsf{p} \wedge m' \neq m \implies \neg\, \exists\, w.\ \mathrm{RHS}(w))$. It is clear that GRC(B) is unsatisfiable for network models with FIFO channels. In a FIFO channel, reception order follows send order. By the semantics of global protocols, per-participant events are totally ordered, and thus $\mathsf{p} \triangleright \mathsf{q}!m'$ by p is ordered before $\mathsf{p} \triangleright \mathsf{q}!m$ by p, which must exist in w because channel compliant words satisfy send-before-receive order. The only way in which m is receivable by q in w is if q has already received m', but the conjunct $w{\Downarrow_{\Sigma_\mathsf{q}}} = \varepsilon$ says that q has no events in w. In conclusion, GRC(B) need only be checked only for $\mathbb{A} = \mathsf{bag}$.

Next, we show how to encode $\mathrm{RHS}(w)$ as a least fixpoint. Li et al. [55] define a family of predicates $\mathrm{avail}_{\mathsf{p},\mathsf{q},\mathcal{B}}(x_1, s_2, \boldsymbol{r_2})$ that captures whether x_1 may be available as the first message from q to p in a $\mathsf{p2p}$ network, while tracking causal dependencies using $\mathcal{B}$. $\mathcal{B}$ tracks the set of participants that are blocked from sending a message because their send action causally depends on q first receiving from r. However, their avail predicate is specific to the $\mathsf{p2p}$ network architecture.

We present a network-parametric version of avail below, parametric in a choice of sender p and receiver q whose message exchange we are searching for, a blocked set $\mathcal{B}$ that tracks causal dependencies between participants, and finally a network architecture $\mathbb{A} \in \{\mathsf{p2p}, \mathsf{sb}, \mathsf{mb}, \mathsf{monob}, \mathsf{bag}\}$.

Definition 7.2 (Symbolic Availability).

$$\begin{aligned}\mathsf{avail}_{\mathbb{A},\mathsf{p},\mathsf{q},\mathcal{B}}(x_1,s,\boldsymbol{r}) :=_{\mu}\quad & \Big(\bigvee_{\substack{(s,\mathsf{r}\to\mathsf{t}:x\{\varphi\},s')\in\Delta\\ \mathsf{r}\in\mathcal{B}}} \exists x\,\boldsymbol{r'}.\,\mathsf{avail}_{\mathbb{A},\mathsf{p},\mathsf{q},\mathcal{B}\cup\{\mathsf{t}\}}(x_1,s',\boldsymbol{r'})\wedge\varphi\Big)\\ \vee\ & \Big(\bigvee_{\substack{(s,\mathsf{r}\to\mathsf{t}:x\{\varphi\},s')\in\Delta\\ \mathsf{r}\notin\mathcal{B}\\ \mathsf{ch}(\mathsf{r},\mathsf{t})\neq\mathsf{ch}(\mathsf{p},\mathsf{q})\vee\mathsf{t}\notin\mathcal{B}\vee\mathbb{A}=\mathsf{bag}}} \exists x\,\boldsymbol{r'}.\,\mathsf{avail}_{\mathbb{A},\mathsf{p},\mathsf{q},\mathcal{B}}(x_1,s',\boldsymbol{r'})\wedge\varphi\Big)\\ \vee\ & \Big(\bigvee_{\substack{(s,\mathsf{p}\to\mathsf{q}:x\{\varphi\},s')\in\Delta\\ \mathsf{p}\notin\mathcal{B}}} \varphi[x_1/x]\Big)\ .\end{aligned}$$

The last disjunct in the definition handles the cases where the message x_1 from p is immediately available to be received by q in symbolic state $(s,\boldsymbol{r})$ and p has not been blocked from sending. The first disjunct skips all transitions where the sender is blocked, and adds the receiver to $\mathcal{B}$ in the recursive call to avail. The second disjunct skips transitions where sender is not blocked, but the message exchange does not interfere with the message x_1 from p to q. That is, one of three scenarios is true: the message exchange occurs at a different channel, or the message exchange occurs at the same channel but the receiver is unblocked, or $\mathbb{A}$ is bag (i.e., x and x_1 can be reordered).

The key observation underpinning our generalized avail predicate is that the conditions for each disjunct determining how each event encountered along a protocol run ought to be handled can be made fully parametric in the communication topology. To demonstrate, observe that when we instantiate $\mathsf{ch}(\mathsf{r},\mathsf{t}) \neq \mathsf{ch}(\mathsf{p},\mathsf{q})$ with $\mathsf{r}\neq\mathsf{p}\vee\mathsf{t}\neq\mathsf{q}$ and $\mathbb{A}=\mathsf{bag}$ with $\bot$ for a p2p box network, and remove the assumption that q is always an element of $\mathcal{B}$, we obtain Definition 5.5 from [55].

The following equivalences show how to instantiate avail to decide Generalized Receive Coherence for each homogeneous network architecture. Note that RHS takes as implicit arguments $\mathsf{p},\mathsf{q},\mathsf{r},m,m'$ and s_1'.

Proposition 7.3. *For* $\mathbb{A}\in\{\mathsf{p2p},\mathsf{bag}\}$, $\exists\, w.\ \mathit{RHS}_{\mathbb{A}}(w) \iff \exists\, x_1,s_2',\boldsymbol{r}.\ \mathsf{avail}_{\mathbb{A},\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(x_1,s_2',\boldsymbol{r})$. *For* $\mathbb{A}\in\{\mathsf{sb},\mathsf{monob},\mathsf{mb}\}$, $\exists\, w.\ \mathit{RHS}_{\mathbb{A}}(w) \iff \exists\, x_1,s_2',\boldsymbol{r}.\ \mathsf{avail}_{\mathbb{A},\mathsf{p},\mathsf{q},\{\mathsf{q},\mathsf{r}\}}(x_1,s_2',\boldsymbol{r})$.

Using the equivalence above and our definitions of avail, we can then define Generalized Receive Coherence as a predicate over symbolic protocols. We give the predicate for $\mathbb{A}=\mathsf{sb}$ as an example.

Definition 7.4 (Symbolic Generalized Receive Coherence for sb*).* A symbolic protocol $\mathbb{S}$ satisfies Symbolic Generalized Receive Coherence for sb when for every pair of transitions $s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:x_1\{\varphi_1\}} s_1'\in\Delta$ and $s_2 \xrightarrow{\mathsf{r}\to\mathsf{q}:x_2\{\varphi_2\}} s_2'\in\Delta$ with $\mathsf{p}\neq\mathsf{r}$:

$$\mathsf{prodreach}_{\mathsf{q}}(s_1,\boldsymbol{r_1},s_2,\boldsymbol{r_2})\ \wedge\ \varphi_1[\boldsymbol{r_1}\boldsymbol{r_1}'/\boldsymbol{r}\boldsymbol{r}']\ \wedge\ \varphi_2[\boldsymbol{r_2}\boldsymbol{r_2}'/\boldsymbol{r}\boldsymbol{r}']\ \wedge\ \mathsf{avail}_{\mathsf{sb},\mathsf{p},\mathsf{q},\{\mathsf{q},\mathsf{r}\}}(x_1,s_2',\boldsymbol{r_2'}) \implies \bot,$$

Finally, we describe how to check Symbolic Prefix Extensibility. First, we establish that a stronger form of prefix extensibility holds for p2p, bag and sb networks.

Lemma 7.5. *For* $\mathbb{A}\in\{\mathsf{p2p},\mathsf{sb},\mathsf{bag}\}$, *for every* $\rho\in\Gamma^*_{sync}$, $w\in\Sigma^*_{async}$ *such that* w *is* $\mathbb{A}$*-channel compliant and agrees with* ρ, *there exists* $u\in\Sigma^*_{async}$ *such that* wu *is channel compliant, and* $wu\equiv_{\mathcal{P}}\mathtt{split}(\rho)$.

The proof is in Appendix A.2. Prefix Extensibility constrains the runs of a global protocol, and requires that every channel-compliant word that partially completes a run can be extended to complete some run. Lemma 7.7 states that for the aforementioned network architectures, every channel-compliant word that partially completes any synchronous word can be extended to complete the same synchronous word. Clearly, Lemma 7.7 implies Prefix Extensibility. The key fact

enabling Lemma 7.7 lies in the ability to fastforward the first event z with active participant p ahead of any word u_1 while maintaining channel compliance, under the condition that u_1 contains no events with p as active participant. The examples given in §5 serve as easy counterexamples to Lemma 7.7 for mb and monob channel compliance. Thus, for these two networks, we must check prefix extensibility on runs of the protocol under consideration explicitly.

To motivate our encoding of prefix extensibility, we first observe that any monob or mb channel compliant word is also p2p channel-compliant. Thus, given a word monob or mb channel compliant word w, let u be its completion such that wu is p2p channel-compliant. On the contrary, not every p2p channel-compliant word is monob or mb channel-compliant. We focus our attention on events that monob or mb channel compliance can order, but that p2p cannot. Both monob and mb networks can order independent sends to a *single* receiver, whose total order is fixed by ρ. Thus, the construction of u fails when it encounters such independent sends. Monobox networks can additionally order independent sends to different receivers. In a p2p network, only independent sends to different receivers that are not causally dependent can be reordered. A key observation here is that introducing a causal dependency between independent sends to different receivers necessarily involves ordering independent sends to a single receiver, and from this we conclude the monob and mb prefix extensibility are equivalent.

Thus, to check prefix extensibility for both mb and monob, one must check that when a message is sent, there cannot exist a different message sent to the same *receiver* that is available. The recursive predicate required is similar in spirit to the avail predicate for the various network models. Formally, it is sufficient to ensure that from any transition $s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:x\{\varphi\}} s_2 \in \Delta$, there does not exist a run passing through s_2 that subsequently passes through a transition $s_3 \xrightarrow{\mathsf{r}\to\mathsf{q}:x\{\varphi\}} s_4$ such that $\mathsf{r} \neq \mathsf{p}$'s send does not depend on p or q. We capture this in the predicate avail′.

Definition 7.6 (Symbolic Availability).

$$\begin{aligned}\mathsf{avail}'_{\mathsf{q},\mathcal{B}}(s,\boldsymbol{r}) :=_\mu \quad & \Big(\bigvee_{\substack{(s,\mathsf{r}\to\mathsf{t}:x\{\varphi\},s')\in\Delta\\ \mathsf{r}\in\mathcal{B}}} \exists x\,\boldsymbol{r'}.\,\mathsf{avail}'_{\mathsf{q},\mathcal{B}\cup\{\mathsf{t}\}}(s',\boldsymbol{r'})\wedge\varphi\Big)\\ \vee \Big(\bigvee_{\substack{(s,\mathsf{r}\to\mathsf{t}:x\{\varphi\},s')\in\Delta\\ \mathsf{r}\notin\mathcal{B}\\ \mathsf{t}\neq\mathsf{q}\vee\mathsf{t}\notin\mathcal{B}}} \exists x\,\boldsymbol{r'}.\,\mathsf{avail}'_{\mathsf{q},\mathcal{B}}(s',\boldsymbol{r'})\wedge\varphi\Big) \;\vee\; \Big(\bigvee_{\substack{(s,\mathsf{r}\to\mathsf{q}:x\{\varphi\},s')\in\Delta\\ \mathsf{r}\notin\mathcal{B}}} \exists x\,\boldsymbol{r'}.\,\varphi\Big)\;.\end{aligned}$$

We additionally require a predicate capturing all reachable register and variable assignments to a given control state, from [57].

Definition 7.7 (Reachability in symbolic protocol). Let $s \in S$. Then,

$$\mathsf{reach}(s',\boldsymbol{r'}) :=_\mu \quad (\,s' = s_0 \wedge \boldsymbol{r'} = \rho_0\,) \;\vee\; \Big(\bigvee_{(s,\mathsf{p}\to\mathsf{q}:x\{\varphi\},s')\in\Delta} \exists x\,\boldsymbol{r}.\,\mathsf{reach}(s,r)\wedge\varphi\Big)\;.$$

With this at hand, we present Symbolic Prefix Extensibility for $\mathbb{A} \in \{\mathsf{mb},\mathsf{monob}\}$.

Definition 7.8 (Prefix extensibility for mailbox, monobox). A symbolic protocol $\mathbb{S}$ satisfies $\mathbb{A}$-prefix extensibility for $\mathbb{A} \in \{\mathsf{mb},\mathsf{monob}\}$ when for every transition $s \xrightarrow{\mathsf{p}\to\mathsf{q}:x\{\varphi\}} s' \in \Delta$:

$$\mathsf{reach}(s',\boldsymbol{r'}) \;\wedge\; \mathsf{avail}'_{\mathsf{q},\{\mathsf{p},\mathsf{q}\}}(s',\boldsymbol{r'}) \implies \bot\;.$$

7.2 Implementability relationships

From the network-parametric definition of avail, it is clear that sb-avail implies p2p-avail implies bag-avail. Because avail appears in a negative position in Generalized Receive Coherence, and the other two coherence conditions are network-agnostic, we obtain the following.

Lemma 7.9 (Implementability relationships). *Any* bag*-implementable global protocol is* p2p*-implementable, and any* p2p*-implementable global protocol is* sb*-implementable.*

The strictness of these inclusions is witnessed by examples G_a and G_b in Fig. 5 and Fig. 6.

As a consequence of the equivalence of mb and monob prefix extensibility, we obtain the following:

Corollary 7.10. *A global protocol is* mb*-implementable if and only if it is* monob*-implementable.*

We further conjecture that the mb/monob class is contained within the p2p class.

We note that the relationship between the sb, p2p and bag network architectures induced by Lemma 7.9 coincide with the relationship between their semantics, defined as sets of MSCs or sets of executable traces [13, 35]. We revisit this point in further detail in §9.

7.3 Complexity of finite fragments

We generalize the co-NP-completeness complexity result from [55] to the other network architectures, and include a detailed discussion in the appendix Appendix A.3. We additionally show that the full generality of GCLTS is not required for the co-NP-hardness lower bound; in particular, the reduction holds even for global types with directed choice [43].

Theorem 7.11. *For* $\mathbb{A} \in \{\mathsf{sb}, \mathsf{mb}, \mathsf{monob}, \mathsf{bag}\}$*, implementability of finite protocols is co-NP complete.*

Corollary 7.12. *For* $\mathbb{A} \in \{\mathsf{p2p}, \mathsf{sb}, \mathsf{mb}, \mathsf{monob}, \mathsf{bag}\}$*, implementability of global multiparty session types with directed choice is co-NP complete.*

8 Implementation and evaluation

We extend the Sprout [57] protocol implementability checker and verification tool so that it is parametric in the network architecture. The original Sprout takes a symbolic global protocol $\mathbb{S}$ as introduced in §7.1 as input. It then checks whether $\mathbb{S}$ represents a GCLTS and if so, whether it is implementable. The implementability check uses an encoding of the Coherence Conditions of [55] to μCLP instances. MuVal is used as the backend solver for the generated muCLP instances. Sprout also provides rudimentary support for verifying protocol-specific properties.

Our extension of Sprout replaces the encoding of *CC* from [55] with an encoding of our new network-parametric Generalized *CC*, following the discussion in §7.1. We instantiate the encoding for the five concrete network architectures $\mathbb{A} \in \{\mathsf{p2p}, \mathsf{sb}, \mathsf{mb}, \mathsf{monob}, \mathsf{bag}\}$. The new tool, Sprout($\mathbb{A}$), allows users to select from one of these architectures when invoking the tool.

Absent any implementation bugs unbeknownst to us, Sprout($\mathbb{A}$) is sound and complete relative to the soundness and completeness of MuVal. Since implementability of symbolic protocols is undecidable, MuVal may diverge on some of the generated μCLP instances.

Our evaluation of Sprout($\mathbb{A}$) aims to answer the following questions:

- Q1 Does Sprout($\mathbb{A}$) correctly decide implementability?
- Q2 How does Sprout($\mathbb{A}$)'s performance (i.e., verification times / termination behavior) change across different network architectures?

Benchmarks. To help answer both questions, we start from the benchmark suite used in the evaluation of Sprout [57]. These benchmarks subsume various benchmark suites in the multiparty session type literature but also includes new ones for further diversification, in particular, to increase the number of interesting non-implementable benchmarks. Examples include idealized specifications of various web services and distributed protocols. To further help us answer Q1 we added select mini benchmarks that separate each pair of considered network architectures with respect to implementability. These benchmarks are discussed in §7.2.

Source	Example	$\lvert\mathcal{P}\rvert$	p2p	Time	sb	Time	mb	Time	monob	Time	bag	Time
[84]	Calculator	2	✓	0.6s	✓	0.7s	✓	0.7s	✓	0.8s	×	16.5s
	Fibonacci	2	✓	0.5s	✓	0.5s	✓	0.5s	✓	0.5s	×	3.3s
	HigherLower	3	✓	15.8s	✓	13.6s	×	17.0s	×	18.5s	×	45.6s
	HTTP	2	✓	0.5s	✓	0.4s	✓	0.4s	✓	0.5s	×	24.4s
	Negotiation	2	✓	1.0s	✓	1.0s	✓	1.0s	✓	1.0s	×	26.1s
	OnlineWallet	3	✓	17.5s	✓	16.6s	✓	16.5s	✓	20.6s	×	100.0s
	SH	3	✓	237.1s	✓	244.0s	✓	256.1s	✓	257.2s	?	T/O
	Ticket	2	✓	0.6s	✓	0.6s	✓	0.6s	✓	0.6s	×	3.8s
	TravelAgency	2	✓	9.6s	✓	9.7s	×	12.2s	×	11.6s	×	18.3s
	TwoBuyer	3	✓	4.2s	✓	4.0s	×	6.3s	×	6.3s	✓	9.3s
[80]	DoubleBuffering	3	✓	1.5s	✓	1.5s	×	2.4s	×	2.3s	✓	2.5s
	OAuth	3	✓	6.5s	✓	6.6s	✓	10.5s	✓	10.3s	×	18.1s
	PlusMinus	3	✓	5.5s	✓	5.3s	×	7.7s	×	8.6s	✓	13.2s
	RingMax	7	✓	3.7s	✓	3.6s	✓	5.3s	✓	6.8s	✓	16.5s
	SimpleAuth	2	✓	0.5s	✓	0.5s	✓	0.5s	✓	0.5s	×	4.7s
	TravelAgency2	2	✓	0.5s	✓	0.5s	✓	0.5s	✓	0.5s	✓	3.7s
[54]	send-validity-yes	4	✓	1.9s	✓	2.7s	✓	2.7s	✓	2.7s	✓	3.6s
	send-validity-no	4	×	1.9s	×	1.9s	×	2.7s	×	2.7s	×	3.6s
	receive-validity-yes	3	✓	5.3s	✓	5.3s	✓	6.5s	✓	5.9s	✓	7.8s
	receive-validity-no	3	×	3.7s	×	3.8s	×	4.6s	×	4.0s	×	5.4s
[55]	symbolic-two-bidder-yes	3	✓	24.0s	✓	21.6s	✓	25.2s	✓	22.0s	✓	31.7s
	symbolic-two-bidder-no1	3	×	23.9s	×	20.3s	×	22.9s	×	18.3s	×	29.5s
	figure12-yes	3	✓	2.0s	✓	2.0s	✓	8.7s	✓	8.5s	✓	12.8s
	figure12-no	3	×	3.0s	×	3.0s	×	9.7s	×	10.7s	×	13.8s
	symbolic-send-validity-yes	4	✓	6.6s	✓	6.5s	✓	10.0s	✓	10.5s	✓	16.0s
	symbolic-send-validity-no	4	×	5.6s	×	5.4s	×	8.3s	×	8.8s	×	13.2s
	symbolic-receive-validity-yes	3	✓	6.4s	✓	6.3s	×	8.7s	×	8.4s	✓	11.8s
	symbolic-receive-validity-no	3	×	8.0s	×	7.8s	×	12.3s	×	12.3s	×	14.5s
[15]	fwd-auth-yes	3	✓	10.5s	✓	10.5s	✓	15.6s	✓	16.1s	×	27.3s
	fwd-auth-no	3	?	T/O	?	T/O	×	157.4s	×	163.0s	×	176.1s
[57]	symbolic-two-bidder-no2	3	×	23.8s	×	29.5s	×	31.1s	×	28.5s	×	40.1s
	higher-lower-ultimate	3	✓	13.3s	✓	13.1s	×	20.5s	×	25.3s	×	30.8s
	higher-lower-winning	3	?	T/O	✓	36.3s	×	37.5s	×	45.1s	✓	62.2s
	higher-lower-no	3	×	13.4s	×	12.9s	×	20.9s	×	29.9s	×	45.3s
	higher-lower-encrypt-yes	4	✓	10.0s	✓	9.8s	×	12.9s	×	14.3s	✓	25.0s
	higher-lower-encrypt-no	4	×	77.3s	×	88.8s	×	100.0s	×	109.2s	×	126.1s
	higher-lower-mixed	3	×	33.6s	×	33.8s	×	36.4s	×	36.2s	×	41.3s
new	bag-no-p2p-yes	2	✓	0.4s	✓	0.4s	✓	3.4s	✓	0.6s	×	1.8s
	mb-no-p2p-yes	3	✓	0.9s	✓	0.9s	×	1.3s	×	1.3s	✓	1.3s
	p2p-no-sb-yes	4	×	4.0s	✓	4.1s	×	5.3s	×	5.3s	×	6.7s
	mb-no-monob-no	3	×	3.2s	✓	3.2s	×	4.1s	×	4.1s	×	5.4s
	monob-no-mb-no	4	✓	1.3s	✓	1.3s	×	1.9s	×	1.9s	✓	1.9s
	Average			6.7s		5.8s		7.6s		7.1s		12.6s

Table 1. Run times and verification results for all five network architectures. For each example, we report the number of participants ($\lvert\mathcal{P}\rvert$), and for each architecture, the result: ✓ for implementable, × for non-implementable, and ? for inconclusive due to timeout, and run time, with a 15s timeout per μCLP instance (T/O). The average run times across all benchmark listed in the last row are computed only over those benchmarks for which none of the configurations timed out.

Experiment and results. To answer our two questions, we run Sprout($\mathbb{A}$) on the full benchmark suite and all considered network architectures. The experiment is run on a 2024 MacBook Air with an Apple M3 chip and 16GB of RAM, with run times averaged over 10 runs. The results are summarized in Table 1. Run times reported are the sum of GCLTS checking time and implementability checking time, with timeouts for individual μCLP instances specified separately.

Correctness (Q1). We first observe that the implementability results for peer-to-peer box are consistent with those reported in [57] for the common benchmarks. Next, note that the results are also consistent with the implementability relationships between network architectures identified in Lemma 7.9. For example, the subset of benchmarks identified as implementable under sb is a strict superset of those for p2p. Finally, we did a manual inspection of individual results for our separation mini benchmarks and a random selection of other results. In all cases, the produced results were determined to be correct.

Performance (Q2). Sprout (𝔸) shows similar running times and termination behavior for the new network architectures compared to the previously supported p2p, with bag behaving the slowest. The slower behavior of bag is to be expected since the relaxed ordering on message buffers increases the state space to be explored by the message availability check used to implement the GRC condition. In general, the relationships of the average run times appear to track the theoretical relationships between network architectures identified in §7.2.

9 Related Work

Global protocol specifications. Global protocol specifications in the form of message sequence charts found early industry adoption by the ITU standard [47] in 1993, was subsequently incorporated into UML [**?**] in 2005, and is part of the Web Service Choreography Description Language [81]. Global specifications are widely studied in academia in the form of multiparty session types and choreographic programming. Multiparty session types (MSTs) have enjoyed widespread implementation in a variety of programming languages including Python [21, 66, 68], Java [44, 45], C [69], Go [9, 52], Scala [10], Rust [18, 50], OCaml [46] and F# [67]. Application domains for MSTs include operating systems [23], high performance computing [19, 42, 70], cyber-physical systems [60, 61], and web services [83]. Choreographic programming frameworks have been implemented in Java [33], Haskell [75], Rust [49**?**] and applied to distributed architecture [72], cryptographic security protocols [26], and cyber-physical systems [16]. We refer the reader to [82] and [63] for a comprehensive survey of MST and choreography applicability respectively.

Network-parametric results from concurrency theory. Bollig et al. [6] and Giusto et al. [35] study the synchronizability problem, which asks whether a communicating finite state machine can be precisely approximated in terms of its "synchronous" executions. Bollig et al. [6] presents a framework based on MSO logic and special treewidth, and uses it to establish decidability results for existential and universal k-boundedness [28], weak synchrony and weak k-synchrony [7] of p2p and mb communication.

Giusto et al. [35] generalize Bollig et al. [6] to five additional communication models, defined as sets of message sequence charts (MSCs): sb, monob, bag, causally ordered and realizable with synchronous communication. MSCs are partial order graphs on asynchronous events that specify a total order per participant. MSCs can thus specify unmatched sends and out-of-order receptions, but not branching behavior or recursion. Giusto et al. [35] further generalize synchronizability to any notion expressible in terms of a class of MSCs with bounded special treewidth. The generalization thus rests on an encoding of communication models and synchronizability properties as MSO formulae, and Courcelle's Theorem, which applies to any graph with bounded special treewidth.

Non-FIFO (equivalent to bag) communication is considered alongside FIFO communication in [59], which studies the safe realizability problem of HMSCs with respect to communicating finite state machines. HMSCs generalizee GCLTS along two dimensions: firstly, while specifications are required to satisfy FIFO restrictions, they are not required to be 1-synchronous; secondly, branching choice is unrestricted, relaxing the sender-driven choice assumption of GCLTS. Unlike our setting, HMSCs are not given infinite word semantics. Lohrey [59] shows that all (un)decidability results generalize to bag communication, by exploiting a reduction between FIFO and non-FIFO communication to transfer upper bounds of safe realizability.

Recently, the realizability of global types for both p2p and rendezvous synchronous communication was studied in [36]. Similar to our notion of channel compliance, the work defines communication models as sets of linearizations. Rendezvous synchrony is essentially asynchronous communication with a global channel bound of size 1, *à la* rsc from [13] and [35], and corresponds to $\mathtt{split}(\rho)$ asynchronous words in our setting. [36] lifts all structural restrictions on the graph

structure, notably including sender-driven choice. No decision procedure for realizability of finite global types is given, either in a network-specific or network-agnostic manner. The relationship between p2p FIFO-realizable and synchronously realizable global types is instead established through the definition of realizability directly.

Chevrou et al. [13] study the same seven communication models considered by [35], and establish a hierarchy that differs from [35] in the placement of sb and mb communication: [13] judges the two incomparable, whereas [35] judges sb to subsume mb. The discrepancy is due to the difference between a universal and existential quantifier: Chevrou et al. define communication models as sets of linearizations, all of which respect the communication model, whereas Giusto et al. define communication models as sets of MSCs, of which one exhibits a linearization that respects the communication model. The focus is on establishing a taxonomy, and not on specific decision problems concerning one or more communication models. Their results are mechanized in TLA+.

Our notion of network architecture identifies each channel with a single message buffer, as is common in the context of global protocol specification frameworks. Engels et al. [22] consider a more fine-grained model where each channel is associated with an input and and output buffer. This results in a third asynchronous event per message exchange that corresponds to the transmission of a message from the output to the input buffer. They then study, among others, the implementability problem for MSCs under different communication topologies for FIFO channels. However, their notion of implementability differs from ours in that they only check whether all linearizations of the MSC are consistent with the network semantics, but not whether this set can be exactly realized by a CLTS where participants only make local observations. Their results also do not easily generalize to HMCSs and, thus, rule out features like recursion and loops in specifications.

Our axiomatic network model allows for network architectures that exhibit channels with different semantics such as FIFO and bag within a single network. Clemente et al. [14] study the decidability of the reachability problem for finite-state CLTS with such hybrid networks. Specifically, they provide a complete characterization of the communication topologies for which decidability can be retained.

Tools. To our knowledge Sprout($\mathbb{A}$) is the first tool that can check implementability of global protocols for a range of different network architectures. In this work, we build on Sprout [57], which implements the symbolic algorithm for checking implementability developed in [55] for the p2p case. Sprout's closest competitors are Session* [84] and Rumpsteak with refinements [80]. We refer to [57] for a detailed comparison.

Synchrony. We have excluded synchronous communication models from our investigation in this paper. Top-down verification for synchronous communication models have been widely studied in their own right, prominently in the form of synchronous multiparty session types [11, 32, 73, 78]. Zielonka automata [85] and their "realistic" variant [1] are a natural starting point for developing a unifying theory of protocols with synchronous or asynchronous semantics. Notably, the Send Coherence condition employed in [55] and this work can be viewed as a special instance of the third semantical condition, *causally closed* [1, Definition 9, (LC3)], characterizing synchronous realizability of global specifications. This connection points towards the possibility of an implementability characterization that simultaneously handles synchronous and asynchronous communication.

10 Conclusion

We presented a network-parametric, precise solution to global protocol implementability and a simple axiomatic network model that characterizes the network architectures to which this solution applies. Equipped with this solution, we derived decidability and complexity results for finite state fragments of global protocols and commonly considered network architectures as well

symbolic algorithms for global protocols with potentially infinite state spaces and data domains. We implemented the latter in an existing tool and showed that the resulting new tool, Sprout ($\mathbb{A}$), provides increased applicability across architectures without sacrificing performance.

One implication of our work is that the chosen network architecture for the implementability problem does not affect synthesis: for all considered network architectures, the global protocol is implementable if and only if the canonical implementation obtainable by local projection implements it. This modularity makes us hopeful that our results can be directly applied to broaden the scope of top-down methodologies.

Acknowledgements. This work is supported by the National Science Foundation under the grant agreement 2304758.

References

[1] S. Akshay, Ionut Dinca, Blaise Genest, and Alin Stefanescu. 2013. Implementing Realistic Asynchronous Automata. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, December 12-14, 2013, Guwahati, India (LIPIcs, Vol. 24)*, Anil Seth and Nisheeth K. Vishnoi (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 213–224. https://doi.org/10.4230/LIPICS.FSTTCS.2013.213

[2] Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. 2003. Inference of Message Sequence Charts. *IEEE Trans. Software Eng.* 29, 7 (2003), 623–633. https://doi.org/10.1109/TSE.2003.1214326

[3] Rajeev Alur and Mihalis Yannakakis. 1999. Model Checking of Message Sequence Charts. In *CONCUR '99: Concurrency Theory, 10th International Conference, Eindhoven, The Netherlands, August 24-27, 1999, Proceedings (Lecture Notes in Computer Science, Vol. 1664)*, Jos C. M. Baeten and Sjouke Mauw (Eds.). Springer, 114–129. https://doi.org/10.1007/3-540-48320-9_10

[4] Laura Bocchi, Romain Demangeon, and Nobuko Yoshida. 2012. A Multiparty Multi-session Logic. In *Trustworthy Global Computing - 7th International Symposium, TGC 2012, Newcastle upon Tyne, UK, September 7-8, 2012, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 8191)*, Catuscia Palamidessi and Mark Dermot Ryan (Eds.). Springer, 97–111. https://doi.org/10.1007/978-3-642-41157-1_7

[5] Laura Bocchi, Kohei Honda, Emilio Tuosto, and Nobuko Yoshida. 2010. A Theory of Design-by-Contract for Distributed Multiparty Interactions. In *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6269)*, Paul Gastin and François Laroussinie (Eds.). Springer, 162–176. https://doi.org/10.1007/978-3-642-15375-4_12

[6] Benedikt Bollig, Cinzia Di Giusto, Alain Finkel, Laetitia Laversa, Étienne Lozes, and Amrita Suresh. 2021. A Unifying Framework for Deciding Synchronizability. In *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference (LIPIcs, Vol. 203)*, Serge Haddad and Daniele Varacca (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 14:1–14:18. https://doi.org/10.4230/LIPICS.CONCUR.2021.14

[7] Ahmed Bouajjani, Constantin Enea, Kailiang Ji, and Shaz Qadeer. 2018. On the Completeness of Verifying Message Passing Programs Under Bounded Asynchrony. In *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)*, Hana Chockler and Georg Weissenbacher (Eds.). Springer, 372–391. https://doi.org/10.1007/978-3-319-96142-2_23

[8] Daniel Brand and Pitro Zafiropulo. 1983. On Communicating Finite-State Machines. *J. ACM* 30, 2 (1983), 323–342. https://doi.org/10.1145/322374.322380

[9] David Castro-Perez, Raymond Hu, Sung-Shik Jongmans, Nicholas Ng, and Nobuko Yoshida. 2019. Distributed programming using role-parametric session types in go: statically-typed endpoint APIs for dynamically-instantiated communication structures. *Proc. ACM Program. Lang.* 3, POPL (2019), 29:1–29:30. https://doi.org/10.1145/3290342

[10] David Castro-Perez and Nobuko Yoshida. 2023. Dynamically Updatable Multiparty Session Protocols: Generating Concurrent Go Code from Unbounded Protocols. In *37th European Conference on Object-Oriented Programming, ECOOP 2023, July 17-21, 2023, Seattle, Washington, United States (LIPIcs, Vol. 263)*, Karim Ali and Guido Salvaneschi (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 6:1–6:30. https://doi.org/10.4230/LIPICS.ECOOP.2023.6

[11] Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. 2024. On the Preciseness of Subtyping in Session Types: 10 Years Later. In *Proceedings of the 26th International Symposium on Principles and Practice of Declarative Programming, PPDP 2024, Milano, Italy, September 9-11, 2024*, Alessandro Bruni, Alberto Momigliano, Matteo Pradella, Matteo Rossi, and James Cheney (Eds.). ACM, 2:1–2:3. https://doi.org/10.1145/3678232.3678258

[12] Florent Chevrou, Aurélie Hurault, Shin Nakajima, and Philippe Quéinnec. 2019. A Map of Asynchronous Communication Models. In *Formal Methods. FM 2019 International Workshops - Porto, Portugal, October 7-11, 2019, Revised Selected Papers, Part II (Lecture Notes in Computer Science, Vol. 12233)*, Emil Sekerinski, Nelma Moreira, José N. Oliveira, Daniel Ratiu, Riccardo Guidotti, Marie Farrell, Matt Luckcuck, Diego Marmsoler, José Creissac Campos, Troy Astarte, Laure Gonnord, Antonio Cerone, Luis Couto, Brijesh Dongol, Martin Kutrib, Pedro Monteiro, and David Delmas (Eds.). Springer, 307–322. https://doi.org/10.1007/978-3-030-54997-8_20

[13] Florent Chevrou, Aurélie Hurault, and Philippe Quéinnec. 2016. On the diversity of asynchronous communication. *Formal Aspects Comput.* 28, 5 (2016), 847–879. https://doi.org/10.1007/S00165-016-0379-X

[14] Lorenzo Clemente, Frédéric Herbreteau, and Grégoire Sutre. 2014. Decidable Topologies for Communicating Automata with FIFO and Bag Channels. In *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*, Vol. 8704. Springer, 281–296. https://doi.org/10.1007/978-3-662-44584-6_20

[15] Luís Cruz-Filipe, Eva Graversen, Lovro Lugovic, Fabrizio Montesi, and Marco Peressotti. 2022. Functional Choreographic Programming. In *Theoretical Aspects of Computing - ICTAC 2022 - 19th International Colloquium, Tbilisi, Georgia, September 27-29, 2022, Proceedings (Lecture Notes in Computer Science, Vol. 13572)*, Helmut Seidl, Zhiming Liu, and Corina S. Pasareanu (Eds.). Springer, 212–237. https://doi.org/10.1007/978-3-031-17715-6_15

[16] Luís Cruz-Filipe and Fabrizio Montesi. 2016. Choreographies in Practice. In *Formal Techniques for Distributed Objects, Components, and Systems - 36th IFIP WG 6.1 International Conference, FORTE 2016, Held as Part of the 11th International Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete, Greece, June 6-9, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9688)*, Elvira Albert and Ivan Lanese (Eds.). Springer, 114–123. https://doi.org/10.1007/978-3-319-39570-8_8

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

[18] Zak Cutner, Nobuko Yoshida, and Martin Vassor. 2022. Deadlock-free asynchronous message reordering in rust with multiparty session types. In *PPoPP '22: 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Seoul, Republic of Korea, April 2 - 6, 2022*, Jaejin Lee, Kunal Agrawal, and Michael F. Spear (Eds.). ACM, 246–261. https://doi.org/10.1145/3503221.3508404

[19] Jan de Muijnck-Hughes and Wim Vanderbauwhede. 2019. A Typing Discipline for Hardware Interfaces. In *33rd European Conference on Object-Oriented Programming (ECOOP 2019) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 134)*, Alastair F. Donaldson (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 6:1–6:27. https://doi.org/10.4230/LIPIcs.ECOOP.2019.6

[20] Romain Delpy, Anca Muscholl, and Grégoire Sutre. 2024. An Automata-Based Approach for Synchronizable Mailbox Communication. In *35th International Conference on Concurrency Theory, CONCUR 2024, September 9-13, 2024, Calgary, Canada (LIPIcs, Vol. 311)*, Rupak Majumdar and Alexandra Silva (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 22:1–22:19. https://doi.org/10.4230/LIPICS.CONCUR.2024.22

[21] Romain Demangeon, Kohei Honda, Raymond Hu, Rumyana Neykova, and Nobuko Yoshida. 2015. Practical interruptible conversations: distributed dynamic verification with multiparty session types and Python. *Formal Methods Syst. Des.* 46, 3 (2015), 197–225. https://doi.org/10.1007/S10703-014-0218-8

[22] André Engels, Sjouke Mauw, and Michel A. Reniers. 2002. A hierarchy of communication models for Message Sequence Charts. *Sci. Comput. Program.* 44, 3 (2002), 253–292. https://doi.org/10.1016/S0167-6423(02)00022-9

[23] Manuel Fähndrich, Mark Aiken, Chris Hawblitzel, Orion Hodson, Galen C. Hunt, James R. Larus, and Steven Levi. 2006. Language support for fast and reliable message-based communication in singularity OS. In *Proceedings of the 2006 EuroSys Conference, Leuven, Belgium, April 18-21, 2006*, Yolande Berbers and Willy Zwaenepoel (Eds.). ACM, 177–190. https://doi.org/10.1145/1217935.1217953

[24] Alain Finkel and Étienne Lozes. 2023. Synchronizability of Communicating Finite State Machines is not Decidable. *Log. Methods Comput. Sci.* 19, 4 (2023). https://doi.org/10.46298/LMCS-19(4:33)2023

[25] Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. 1983. A Distributed Algorithm for Minimum-Weight Spanning Trees. *ACM Trans. Program. Lang. Syst.* 5, 1 (1983), 66–77. https://doi.org/10.1145/357195.357200

[26] Joshua Gancher, Sydney Gibson, Pratap Singh, Samvid Dharanikota, and Bryan Parno. 2023. Owl: Compositional Verification of Security Protocols via an Information-Flow Type System. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*. IEEE, 1130–1147. https://doi.org/10.1109/SP46215.2023.10179477

[27] Thomas Gazagnaire, Blaise Genest, Loïc Hélouët, P. S. Thiagarajan, and Shaofa Yang. 2007. Causal Message Sequence Charts. In *CONCUR 2007 - Concurrency Theory, 18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceedings (Lecture Notes in Computer Science, Vol. 4703)*, Luís Caires and Vasco Thudichum Vasconcelos (Eds.). Springer, 166–180. https://doi.org/10.1007/978-3-540-74407-8_12

[28] Blaise Genest, Dietrich Kuske, and Anca Muscholl. 2006. A Kleene theorem and model checking algorithms for existentially bounded communicating automata. *Inf. Comput.* 204, 6 (2006), 920–956. https://doi.org/10.1016/J.IC.2006.01.005

[29] Blaise Genest and Anca Muscholl. 2005. 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*. IEEE Computer Society, 2–4. https://doi.org/10.1109/ACSD.2005.25

[30] Blaise Genest, Anca Muscholl, and Doron A. Peled. 2003. Message Sequence Charts. In *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)*, Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg (Eds.). Springer, 537–558. https://doi.org/10.1007/978-3-540-27755-2_15

[31] Blaise Genest, Anca Muscholl, Helmut Seidl, and Marc Zeitoun. 2006. Infinite-state high-level MSCs: Model-checking and realizability. *J. Comput. Syst. Sci.* 72, 4 (2006), 617–647. https://doi.org/10.1016/j.jcss.2005.09.007

[32] Silvia Ghilezan, Svetlana Jaksic, Jovanka Pantovic, Alceste Scalas, and Nobuko Yoshida. 2019. Precise subtyping for synchronous multiparty sessions. *J. Log. Algebraic Methods Program.* 104 (2019), 127–173. https://doi.org/10.1016/J.JLAMP.2018.12.002

[33] Saverio Giallorenzo, Fabrizio Montesi, and Marco Peressotti. 2024. Choral: Object-oriented Choreographic Programming. *ACM Trans. Program. Lang. Syst.* 46, 1 (2024), 1:1–1:59. https://doi.org/10.1145/3632398

[34] Saverio Giallorenzo, Fabrizio Montesi, Marco Peressotti, David Richter, Guido Salvaneschi, and Pascal Weisenburger. 2021. Multiparty Languages: The Choreographic and Multitier Cases (Pearl). In *35th European Conference on Object-Oriented Programming, ECOOP 2021, July 11-17, 2021, Aarhus, Denmark (Virtual Conference) (LIPIcs, Vol. 194)*, Anders Møller and Manu Sridharan (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 22:1–22:27. https://doi.org/10.4230/LIPIcs.ECOOP.2021.22

[35] Cinzia Di Giusto, Davide Ferré, Laetitia Laversa, and Étienne Lozes. 2023. A Partial Order View of Message-Passing Communication Models. *Proc. ACM Program. Lang.* 7, POPL (2023), 1601–1627. https://doi.org/10.1145/3571248

[36] Cinzia Di Giusto, Étienne Lozes, and Pascal Urso. 2025. Realisability and Complementability of Multiparty Session Types. *CoRR* abs/2507.17354 (2025). https://doi.org/10.48550/ARXIV.2507.17354 arXiv:2507.17354

[37] Jonas Kastberg Hinrichsen, Jesper Bengtson, and Robbert Krebbers. 2020. Actris: session-type based reasoning in separation logic. *Proc. ACM Program. Lang.* 4, POPL (2020), 6:1–6:30. https://doi.org/10.1145/3371074

[38] Jonas Kastberg Hinrichsen, Jesper Bengtson, and Robbert Krebbers. 2022. Actris 2.0: Asynchronous Session-Type Based Reasoning in Separation Logic. *Log. Methods Comput. Sci.* 18, 2 (2022). https://doi.org/10.46298/LMCS-18(2:16)2022

[39] Jonas Kastberg Hinrichsen, Jules Jacobs, and Robbert Krebbers. 2024. Multris: Functional Verification of Multiparty Message Passing in Separation Logic. (2024). https://jihgfee.github.io/papers/multris_manuscript.pdf

[40] Andrew K. Hirsch and Deepak Garg. 2021. Pirouette: Higher-Order Typed Functional Choreographies. *CoRR* abs/2111.03484 (2021). arXiv:2111.03484 https://arxiv.org/abs/2111.03484

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

[42] Kohei Honda, Eduardo R. B. Marques, Francisco Martins, Nicholas Ng, Vasco Thudichum Vasconcelos, and Nobuko Yoshida. 2012. Verification of MPI Programs Using Session Types. In *Recent Advances in the Message Passing Interface - 19th European MPI Users' Group Meeting, EuroMPI 2012, Vienna, Austria, September 23-26, 2012. Proceedings (Lecture Notes in Computer Science, Vol. 7490)*, Jesper Larsson Träff, Siegfried Benkner, and Jack J. Dongarra (Eds.). Springer, 291–293. https://doi.org/10.1007/978-3-642-33518-1_37

[43] Kohei Honda, Nobuko Yoshida, and Marco Carbone. 2008. Multiparty asynchronous session types. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, George C. Necula and Philip Wadler (Eds.). ACM, 273–284. https://doi.org/10.1145/1328438.1328472

[44] Raymond Hu and Nobuko Yoshida. 2016. Hybrid Session Verification Through Endpoint API Generation. In *Fundamental Approaches to Software Engineering - 19th International Conference, FASE 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9633)*, Perdita Stevens and Andrzej Wasowski (Eds.). Springer, 401–418. https://doi.org/10.1007/978-3-662-49665-7_24

[45] Raymond Hu and Nobuko Yoshida. 2017. Explicit Connection Actions in Multiparty Session Types. In *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)*, Marieke Huisman and Julia Rubin (Eds.). Springer, 116–133. https://doi.org/10.1007/978-3-662-54494-5_7

[46] Keigo Imai, Rumyana Neykova, Nobuko Yoshida, and Shoji Yuen. 2020. Multiparty Session Programming With Global Protocol Combinators. In *34th European Conference on Object-Oriented Programming, ECOOP 2020, November 15-17, 2020, Berlin, Germany (Virtual Conference) (LIPIcs, Vol. 166)*, Robert Hirschfeld and Tobias Pape (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 9:1–9:30. https://doi.org/10.4230/LIPICS.ECOOP.2020.9

[47] International Telecommunication Union. 2011. *ITU-T Recommendation Z.120: Message Sequence Chart (MSC).* ITU-T Recommendation Z.120. International Telecommunication Union, Geneva. https://www.itu.int/rec/T-REC-Z.120-201102-I/en

[48] Jules Jacobs, Jonas Kastberg Hinrichsen, and Robbert Krebbers. 2023. Dependent Session Protocols in Separation Logic from First Principles (Functional Pearl). *Proc. ACM Program. Lang.* 7, ICFP (2023), 768–795. https://doi.org/10.1145/3607856

[49] Shun Kashiwa, Gan Shen, Soroush Zare, and Lindsey Kuper. 2023. Portable, Efficient, and Practical Library-Level Choreographic Programming. arXiv:2311.11472 [cs.PL] https://arxiv.org/abs/2311.11472

[50] Nicolas Lagaillardie, Rumyana Neykova, and Nobuko Yoshida. 2022. Stay Safe Under Panic: Affine Rust Programming with Multiparty Session Types (Artifact). *Dagstuhl Artifacts Ser.* 8, 2 (2022), 09:1–09:16. https://doi.org/10.4230/DARTS.8.2.9

[51] Leslie Lamport. 2019. Time, clocks, and the ordering of events in a distributed system. In *Concurrency: the Works of Leslie Lamport*, Dahlia Malkhi (Ed.). ACM, 179–196. https://doi.org/10.1145/3335772.3335934

[52] Julien Lange, Nicholas Ng, Bernardo Toninho, and Nobuko Yoshida. 2018. A static verification framework for message passing in Go using behavioural types. In *Proceedings of the 40th International Conference on Software Engineering,*

ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018, Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman (Eds.). ACM, 1137–1148. https://doi.org/10.1145/3180155.3180157

[71]]chorus Languages, Systems, and Data Lab, UC Santa Cruz. [n. d.]. ChoRus: Choreographic Programming in Rust. https://lsd-ucsc.github.io/ChoRus/introduction.html. Accessed: 2025-07-10.

[54] Elaine Li, Felix Stutz, Thomas Wies, and Damien Zufferey. 2023. Complete Multiparty Session Type Projection with Automata. In *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 13966)*, Constantin Enea and Akash Lal (Eds.). Springer, 350–373. https://doi.org/10.1007/978-3-031-37709-9_17

[55] Elaine Li, Felix Stutz, Thomas Wies, and Damien Zufferey. 2025. Characterizing Implementability of Global Protocols with Infinite States and Data. *PACMPL* 9, Object-oriented Programming, Systems, Languages, and Applications (OOPSLA) (2025).

[56] Elaine Li, Felix Stutz, Thomas Wies, and Damien Zufferey. 2025. Characterizing Implementability of Global Protocols with Infinite States and Data. arXiv:2411.05722 [cs.PL] https://arxiv.org/abs/2411.05722

[57] Elaine Li, Felix Stutz, Thomas Wies, and Damien Zufferey. 2025. Sprout: A Verifier for Symbolic Multiparty Protocols. In *Computer Aided Verification - 37th International Conference, CAV 2025, Zagreb, Croatia, July 23-25, 2025, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 15933)*, Ruzica Piskac and Zvonimir Rakamaric (Eds.). Springer, 304–317. https://doi.org/10.1007/978-3-031-98682-6_16

[58] Elaine Li and Thomas Wies. 2025. Certified Implementability of Global Multiparty Protocols. In *16th International Conference on Interactive Theorem Proving, ITP 2025, September 28 to October 1, 2025, Reykjavik, Iceland (LIPIcs, Vol. 352)*, Yannick Forster and Chantal Keller (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 15:1–15:20. https://doi.org/10.4230/LIPICS.ITP.2025.15

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

[60] Rupak Majumdar, Marcus Pirron, Nobuko Yoshida, and Damien Zufferey. 2019. Motion Session Types for Robotic Interactions (Brave New Idea Paper). In *33rd European Conference on Object-Oriented Programming, ECOOP 2019, July 15-19, 2019, London, United Kingdom (LIPIcs, Vol. 134)*, Alastair F. Donaldson (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 28:1–28:27. https://doi.org/10.4230/LIPIcs.ECOOP.2019.28

[61] Rupak Majumdar, Nobuko Yoshida, and Damien Zufferey. 2020. Multiparty motion coordination: from choreographies to robotics programs. *Proc. ACM Program. Lang.* 4, OOPSLA (2020), 134:1–134:30. https://doi.org/10.1145/3428202

[62] Sjouke Mauw and Michel A. Reniers. 1997. High-level message sequence charts. In *SDL '97 Time for Testing, SDL, MSC and Trends - 8th International SDL Forum, Evry, France, 23-29 September 1997, Proceedings*, Ana R. Cavalli and Amardeo Sarma (Eds.). Elsevier, 291–306.

[63] Fabrizio Montesi. 2023. *Introduction to Choreographies.* Cambridge University Press. https://doi.org/10.1017/9781108981491

[64] Rémi Morin. 2002. Recognizable Sets of Message Sequence Charts. In *STACS 2002, 19th Annual Symposium on Theoretical Aspects of Computer Science, Antibes - Juan les Pins, France, March 14-16, 2002, Proceedings (Lecture Notes in Computer Science, Vol. 2285)*, Helmut Alt and Afonso Ferreira (Eds.). Springer, 523–534. https://doi.org/10.1007/3-540-45841-7_43

[65] Anca Muscholl and Doron A. Peled. 1999. Message Sequence Graphs and Decision Problems on Mazurkiewicz Traces. In *Mathematical Foundations of Computer Science 1999, 24th International Symposium, MFCS'99, Szklarska Poreba, Poland, September 6-10, 1999, Proceedings (Lecture Notes in Computer Science, Vol. 1672)*, Miroslaw Kutylowski, Leszek Pacholski, and Tomasz Wierzbicki (Eds.). Springer, 81–91. https://doi.org/10.1007/3-540-48340-3_8

[66] Rumyana Neykova, Laura Bocchi, and Nobuko Yoshida. 2017. Timed runtime monitoring for multiparty conversations. *Formal Aspects Comput.* 29, 5 (2017), 877–910. https://doi.org/10.1007/S00165-017-0420-8

[67] Rumyana Neykova, Raymond Hu, Nobuko Yoshida, and Fahd Abdeljallal. 2018. A session type provider: compile-time API generation of distributed protocols with refinements in F#. In *Proceedings of the 27th International Conference on Compiler Construction, CC 2018, February 24-25, 2018, Vienna, Austria*, Christophe Dubach and Jingling Xue (Eds.). ACM, 128–138. https://doi.org/10.1145/3178372.3179495

[68] Rumyana Neykova and Nobuko Yoshida. 2017. Multiparty Session Actors. *Log. Methods Comput. Sci.* 13, 1 (2017). https://doi.org/10.23638/LMCS-13(1:17)2017

[69] Nicholas Ng, Nobuko Yoshida, and Kohei Honda. 2012. Multiparty Session C: Safe Parallel Programming with Message Optimisation. In *Objects, Models, Components, Patterns - 50th International Conference, TOOLS 2012, Prague, Czech Republic, May 29-31, 2012. Proceedings (Lecture Notes in Computer Science, Vol. 7304)*, Carlo A. Furia and Sebastian Nanz (Eds.). Springer, 202–218. https://doi.org/10.1007/978-3-642-30561-0_15

[70] Xinyu Niu, Nicholas Ng, Tomofumi Yuki, Shaojun Wang, Nobuko Yoshida, and Wayne Luk. 2016. EURECA compilation: Automatic optimisation of cycle-reconfigurable circuits. In *26th International Conference on Field Programmable Logic and Applications, FPL 2016, Lausanne, Switzerland, August 29 - September 2, 2016*, Paolo Ienne, Walid A. Najjar, Jason Helge Anderson, Philip Brisk, and Walter Stechele (Eds.). IEEE, 1–4. https://doi.org/10.1109/FPL.2016.7577359

[71]]umlwebsite Object Management Group. [n. d.]. Unified Modeling Language (UML) Website. https://www.uml.org/. Accessed: 2025-07-10.

[72] Giuseppe De Palma, Saverio Giallorenzo, Jacopo Mauro, Matteo Trentin, and Gianluigi Zavattaro. 2024. Towards a Function-as-a-Service Choreographic Programming Language: Examples and Applications. arXiv:2406.09099 [cs.PL] https://arxiv.org/abs/2406.09099

[73] Kirstin Peters and Nobuko Yoshida. 2024. Separation and Encodability in Mixed Choice Multiparty Sessions. In *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024, Tallinn, Estonia, July 8-11, 2024*, Pawel Sobocinski, Ugo Dal Lago, and Javier Esparza (Eds.). ACM, 62:1–62:15. https://doi.org/10.1145/3661814.3662085

[74] Abhik Roychoudhury, Ankit Goel, and Bikram Sengupta. 2012. Symbolic Message Sequence Charts. *ACM Trans. Softw. Eng. Methodol.* 21, 2 (2012), 12:1–12:44. https://doi.org/10.1145/2089116.2089122

[75] Gan Shen, Shun Kashiwa, and Lindsey Kuper. 2023. HasChor: Functional Choreographic Programming for All (Functional Pearl). *CoRR* abs/2303.00924 (2023). https://doi.org/10.48550/ARXIV.2303.00924 arXiv:2303.00924

[76] Felix Stutz. 2024. *Implementability of Asynchronous Communication Protocols - The Power of Choice.* Ph. D. Dissertation. Kaiserslautern University of Technology, Germany. https://kluedo.ub.rptu.de/frontdoor/index/index/docId/8077

[77] Bernardo Toninho and Nobuko Yoshida. 2017. Certifying data in multiparty session types. *J. Log. Algebraic Methods Program.* 90 (2017), 61–83. https://doi.org/10.1016/J.JLAMP.2016.11.005

[78] Thien Udomsrirungruang and Nobuko Yoshida. 2025. Top-Down or Bottom-Up? Complexity Analyses of Synchronous Multiparty Session Types. *Proc. ACM Program. Lang.* 9, POPL (2025), 1040–1071. https://doi.org/10.1145/3704872

[79] Hiroshi Unno, Tachio Terauchi, Yu Gu, and Eric Koskinen. 2023. Modular Primal-Dual Fixpoint Logic Solving for Temporal Verification. *Proc. ACM Program. Lang.* 7, POPL (2023), 2111–2140. https://doi.org/10.1145/3571265

[80] Martin Vassor and Nobuko Yoshida. 2024. Refinements for Multiparty Message-Passing Protocols: Specification-Agnostic Theory and Implementation. In *38th European Conference on Object-Oriented Programming, ECOOP 2024, September 16-20, 2024, Vienna, Austria (LIPIcs, Vol. 313)*, Jonathan Aldrich and Guido Salvaneschi (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 41:1–41:29. https://doi.org/10.4230/LIPICS.ECOOP.2024.41

[81] Web Services Choreography Working Group. 2005. *Web Services Choreography Description Language Version 1.0.* W3C Candidate Recommendation. World Wide Web Consortium (W3C). Available at http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/.

[82] Nobuko Yoshida. 2024. Programming Language Implementations with Multiparty Session Types. In *Active Object Languages: Current Research Trends*, Frank S. de Boer, Ferruccio Damiani, Reiner Hähnle, Einar Broch Johnsen, and Eduard Kamburjan (Eds.). Lecture Notes in Computer Science, Vol. 14360. Springer, 147–165. https://doi.org/10.1007/978-3-031-51060-1_6

[83] Nobuko Yoshida, Raymond Hu, Rumyana Neykova, and Nicholas Ng. 2013. The Scribble Protocol Language. In *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)*, Martín Abadi and Alberto Lluch-Lafuente (Eds.). Springer, 22–41. https://doi.org/10.1007/978-3-319-05119-2_3

[84] Fangyi Zhou, Francisco Ferreira, Raymond Hu, Rumyana Neykova, and Nobuko Yoshida. 2020. Statically verified refinements for multiparty protocols. *Proc. ACM Program. Lang.* 4, OOPSLA (2020), 148:1–148:30. https://doi.org/10.1145/3428216

[85] Wieslaw Zielonka. 1987. Notes on Finite Asynchronous Automata. *RAIRO Theor. Informatics Appl.* 21, 2 (1987), 99–135. https://doi.org/10.1051/ITA/1987210200991

A Appendix

A.1 Proofs for §6

In this section, we prove Lemma 6.2. Let $\mathbb{A} \in \mathfrak{A}$ be a network architecture that satisfies our axiomatic network model. We need to prove that $\mathbb{A}$ satisfies F1 through F6. For convenience, we state these facts again here in one place:

Definition A.1 (Channel compliance facts). Let $w \in \Sigma^*_{async}$ be channel-compliant.

- F1 For all $x \in \Sigma_!$, wx is channel-compliant.
- F2 For all $\mathtt{p} \neq \mathtt{q} \in \mathcal{P}$ and $m \in \mathcal{V}$, $|w{\Downarrow_{\mathtt{p}\triangleright\mathtt{q}!m}}| \geq |w{\Downarrow_{\mathtt{q}\triangleleft\mathtt{p}?m}}|$.
- F3 For all $\rho \in \Gamma^*_{sync}$, $\mathtt{split}(\rho)$ is channel-compliant.
- F4 For all $\rho \in \Gamma^*_{sync}$, if $w \equiv_{\mathcal{P}} \mathtt{split}(\rho)$, then w is channel-matched.
- F5 For all $x \in \Sigma_!$, $y \in \Sigma_?$, if wy is channel-compliant then wxy is channel-compliant.
- F6 Let $\alpha, \beta \in \Gamma^*_{sync}$, $\mathtt{p} \neq \mathtt{q} \in \mathcal{P}$ and $m \in \mathcal{V}$ such that for all $\mathtt{p} \in \mathcal{P}$, $w{\Downarrow_{\Sigma_\mathtt{p}}} \leq \mathtt{split}(\alpha\beta){\Downarrow_{\Sigma_\mathtt{p}}}$, and $w{\Downarrow_{\Sigma_\mathtt{q}}} = \mathtt{split}(\alpha){\Downarrow_{\Sigma_\mathtt{q}}}$, and $w \cdot \mathtt{q} \triangleleft \mathtt{p}?m$ is channel-compliant. Then, there exists w' such that w' is compliant with β, $w'{\Downarrow_{\Sigma_\mathtt{q}}} = \varepsilon$ and $w' \cdot \mathtt{q} \triangleleft \mathtt{p}?m$ is channel-compliant.

Channel states are maps from channels to buffer contents, and thus a word is channel-compliant if and only if its projection onto each channel is defined for the respective buffer. In the proofs below, we thus only need to reason about buffer insert and remove operations, and buffer states.

Lemma A.2 (F2). *For all $\mathtt{p} \neq \mathtt{q} \in \mathcal{P}$ and $m \in \mathcal{V}$, $|w{\Downarrow_{\mathtt{p}\triangleright\mathtt{q}!m}}| \geq |w{\Downarrow_{\mathtt{q}\triangleleft\mathtt{p}?m}}|$.*

Proof. By strong induction on $|w|$. Let $w = w'x$. We case split on whether x is a send or receive event. If $x \in \Sigma_!$, the claim immediately follows. If $x \in \Sigma_?$, let $x = \mathtt{s} \triangleleft \mathtt{r}?m$. We want to show that $|w'x{\Downarrow_{\mathtt{r}\triangleright\mathtt{s}!m}}| \geq |w'x{\Downarrow_{\mathtt{s}\triangleleft\mathtt{r}?m}}|$. From the induction hypothesis, $|w'{\Downarrow_{\mathtt{r}\triangleright\mathtt{s}!m}}| \geq |w'{\Downarrow_{\mathtt{s}\triangleleft\mathtt{r}?m}}|$. It suffices to analyze the case when $|w'{\Downarrow_{\mathtt{r}\triangleright\mathtt{s}!m}}| = |w'{\Downarrow_{\mathtt{s}\triangleleft\mathtt{r}?m}}|$, i.e. every message m sent in w' is received. Let $w' = w''y$. Next, we case split on whether y is a send or receive event. If $y \in \Sigma_!$, and moreover y's message is not equal to m, then it follows from (B7) that $w' \cdot \mathtt{s} \triangleleft \mathtt{r}?m \cdot y$ Let m_y be the message value of y. If $m_y \neq m$, then $w = w' \cdot \mathtt{p} \triangleright \mathtt{q}!m \cdot \mathtt{s} \triangleleft \mathtt{r}?m$, and by B7 lifted to words, we obtain that $w'.\mathtt{s} \triangleleft \mathtt{r}?m \cdot \mathtt{p} \triangleright \mathtt{q}!m$ reaches the same channel state as w, and we use the induction hypothesis on $w' \cdot \mathtt{s} \triangleleft \mathtt{r}?m$. If $m_y = m$, note that because messages are tagged with sender and receiver information, it follows that $y = \mathtt{r} \triangleright \mathtt{s}!m$. We use the induction hypothesis on w' directly. If $y \in \Sigma_?$, we first use the induction hypothesis on $w'y$ to exhibit at least one occurrence of $\mathtt{r} \triangleright \mathtt{s}!m$ in w'. We find the first occurrence of $\mathtt{r} \triangleright \mathtt{s}!m$ in w': let $w' = u_1 \cdot \mathtt{r} \triangleright \mathtt{s}!m \cdot u_2$ such that $u_1{\Downarrow_{\mathtt{r}\triangleright\mathtt{s}!m}} = \varepsilon$. If $u_1 \cdot \mathtt{s} \triangleleft \mathtt{r}?m$ is buffer compliant, we use the induction hypothesis on u_1. If not, we find the first occurrence of $\mathtt{s} \triangleleft \mathtt{r}?m$ in u_2. If $\mathtt{s} \triangleleft \mathtt{r}?m$ does not occur in u_2, we instantiate the induction hypothesis with w'. Otherwise, we further split up $u_2 = v_1 \cdot \mathtt{s} \triangleleft \mathtt{r}?m \cdot v_2$ such that $v_1{\Downarrow_{\mathtt{s}\triangleleft\mathtt{r}?m}} = \varepsilon$. We then use (B8) on $w = u_1 \cdot \mathtt{r} \triangleright \mathtt{s}!m \cdot v_1 \cdot \mathtt{s} \triangleleft \mathtt{r}?m \cdot v_2$ to conclude that $u_1v_1v_2$ reaches the same state as w. We use the induction hypothesis on $u_1v_1v_2 \cdot \mathtt{q} \triangleleft \mathtt{p}?m$ to prove the claim. □

Lemma A.3 (F1). *For all $x \in \Sigma_!$, wx is channel compliant.*

Proof. Follows immediately from (B1) stating the totality of insert. □

Lemma A.4 (F5). *For all $x \in \Sigma_!$, $y \in \Sigma_?$, if wy is channel compliant then wxy is channel compliant.*

Proof. Follows immediately from (B4). □

Lemma A.5 (F3). *For all $\rho \in \Gamma^*_{sync}$, $\mathtt{split}(\rho)$ is channel compliant.*

Proof. We prove a stronger property, which is that for all $w \in \Sigma^*_{async}$, $\mathtt{split}(w)$ reaches the empty channel state ξ_0. We prove this by induction on $|w|$. The base case is trivial. In the induction

step, let $w = w' \cdot \mathtt{p} \to \mathtt{q} : m$. From the induction hypothesis, $\mathtt{split}(w')$ reaches the empty channel state ξ_0. Let $c = \mathsf{ch}(\mathtt{p}, \mathtt{q})$. Then, $\xi_0(c) = b_0$. By (B2), $b_0 \bullet \mathit{ins}(x) \bullet \mathit{rem}(x)$ is defined. By (B8), $b_0 \bullet \mathit{ins}(x) \bullet \mathit{rem}(x) = b_0$. Thus, $\mathtt{split}(w' \cdot \mathtt{p} \to \mathtt{q} : m)$ reaches the empty channel state, and is thus channel-compliant. □

Lemma A.6 (F4). *Let $\mathcal{T}_{\mathbb{A}}$ be a CLTS, $w \in \Sigma^*_{\mathit{async}}$ and $\rho \in \Gamma^*_{\mathit{sync}}$ such that $w \equiv_{\mathcal{P}} \mathtt{split}(\rho)$, and w is a trace of $\mathcal{T}_{\mathbb{A}}$. Then, in the configuration reached on w, all channels are empty.*

Proof. We prove that every channel-compliant word $\bar{w}$ that is per-role equal to $\mathtt{split}(\rho)$ for some $\rho \in \Gamma^*_{\mathit{sync}}$ reaches the empty channel, by induction on $|\rho|$. In the base case, $\rho = \varepsilon$ and $w = \varepsilon$, and the claim holds trivially. In the induction step, let $w = xw'$. We case split on whether x is a send or receive event. If $x \in \Sigma_?$, by assumption xw' is channel-compliant, and by prefix closure of channel compliance, x is channel-compliant. By (B3), removing from an empty channel is undefined, and we reach a contradiction. Let $x \in \Sigma_!$ be $\mathtt{p} \triangleright \mathtt{q}!m$. Because xw' is per-role equal to $\mathtt{split}(\rho)$, there is at least one occurrence of $\mathtt{q} \triangleleft \mathtt{p}?m$ in w'. We find the first occurrence, i.e. let $w = \mathtt{p} \triangleright \mathtt{q}!m \cdot u_1 \cdot \mathtt{q} \triangleleft \mathtt{p}?m \cdot u_2$ such that $\mathtt{q} \triangleleft \mathtt{p}?m$ does not occur in u_1. Furthermore, let $\rho = \rho_1 \cdot \mathtt{p} \to \mathtt{q} : m \cdot \rho_2$ such that $\mathtt{p} \to \mathtt{q} : m$ does not occur in ρ_1. Thus, u_1u_2 is per-role identical to $\rho_1\rho_2$, and by the induction hypothesis, u_1u_2 reaches the empty channel. By (B8) lifted to channel states, the channel state reached on u_1u_2 is the same as that reached on xw'. We conclude that the channel state reached on $w = xw'$ is the empty channel. □

Lemma A.7 (F6). *Let $w \in \Sigma^*_{\mathit{async}}$ be channel-compliant. Let $\alpha, \beta \in \Gamma^*_{\mathit{sync}}$, $\mathtt{p} \neq \mathtt{q} \in \mathcal{P}$ and $m \in \mathcal{V}$ such that for all $\mathtt{p} \in \mathcal{P}$, $w{\Downarrow_{\Sigma_\mathtt{p}}} \leq \mathtt{split}(\alpha\beta){\Downarrow_{\Sigma_\mathtt{p}}}$, and $w{\Downarrow_{\Sigma_\mathtt{q}}} = \mathtt{split}(\alpha){\Downarrow_{\Sigma_\mathtt{q}}}$, and $w \cdot \mathtt{q} \triangleleft \mathtt{p}?m$ is channel-compliant. Then, there exists w' such that w' is compliant with β, $w{\Downarrow_{\Sigma_\mathtt{q}}} = \varepsilon$ and $w' \cdot \mathtt{q} \triangleleft \mathtt{p}?m$ is channel-compliant.*

Proof. We construct w' in two steps. In the first step, we remove matched pairs of events from w and α.

Claim 1. There exists w_1 and α_1 such that w_1 is compliant with $\alpha_1\beta$, w_1 contains no matched pairs of events from α, and moreover reaches the same channel state as w. We prove the claim by induction on the number of matched pairs in w from α. In the base case, let $w_1 = w$ and $\alpha_1 = \alpha$. In the induction step, let $\alpha = \gamma_1 \cdot \mathtt{r} \to \mathtt{s} : m \cdot \gamma_2$ such that $\mathtt{split}(\gamma_1 \cdot \mathtt{r} \to \mathtt{s} : m){\Downarrow_{\Sigma_\mathtt{r}}} \leq w{\Downarrow_{\Sigma_\mathtt{r}}}$, and $\mathtt{split}(\gamma_1 \cdot \mathtt{r} \to \mathtt{s} : m){\Downarrow_{\Sigma_\mathtt{s}}} \leq w{\Downarrow_{\Sigma_\mathtt{s}}}$. Let $w_\mathtt{r}$ be the maximal prefix of w with respect to γ_1 for $\mathtt{r}$, such that $w = w_\mathtt{r} \cdot \mathtt{r} \triangleright \mathtt{s}!m \cdot u_\mathtt{r}$. Let $w_\mathtt{s}$ be the maximal prefix of w with respect to γ_1 for $\mathtt{s}$, such that $w = w_\mathtt{s} \cdot \mathtt{s} \triangleleft \mathtt{r}?m \cdot u_\mathtt{s}$. It follows from (F2) that $w_\mathtt{r} \cdot \mathtt{r} \triangleright \mathtt{s}!m \leq w_\mathtt{s}$, and we can write $w = w_\mathtt{r} \cdot \mathtt{r} \triangleright \mathtt{s}!m \cdot u \cdot \mathtt{s} \triangleleft \mathtt{r}?m \cdot v$ such that $\mathtt{s} \triangleleft \mathtt{r}?m$ does not occur in u. Thus, $w_\mathtt{s}uv$ is compliant with $\gamma_1\gamma_2$, contains one fewer matched event from α, and by (B8), reaches the same channel state as w. The claim thus follows from the induction hypothesis.

Because $w \cdot \mathtt{q} \triangleleft \mathtt{p}?m$ is channel-compliant, and w_1 reaches the same channel state as w, it follows that $w_1 \cdot \mathtt{q} \triangleleft \mathtt{p}?m$ is channel-compliant.

In the second step, we remove unmatched send events from w_1 and α_1.

Claim 2. There exists w_2 and α_2 such that w_2 is compliant with $\alpha_2\beta$, w_2 contains no matched pairs of events from α and no unmatched sends from α, and moreover $w_2 \cdot \mathtt{q} \triangleleft \mathtt{p}?m$ is channel-compliant. We prove the claim by induction on the number of unmatched sends in w_1 from α_1. In the base case, let $w_2 = w_1$ and $\alpha_2 = \alpha_1$. In the induction step, let $\alpha_1 = \gamma_3 \cdot \mathtt{r} \to \mathtt{s} : m \cdot \gamma_4$ such that $\mathtt{split}(\gamma_3 \cdot \mathtt{r} \to \mathtt{s} : m){\Downarrow_{\Sigma_\mathtt{r}}} \leq w{\Downarrow_{\Sigma_\mathtt{r}}}$, and the maximal prefix of w with respect to γ_3 for $\mathtt{r}$ is the longest relative to all over unmatched send events in α_1. In other words, we find the rightmost occurrence of an unmatched send event from α_1 in w_1. First, we establish that $\mathtt{r} \to \mathtt{s} : m \neq \mathtt{p} \to \mathtt{q} : m$. Because $w{\Downarrow_{\Sigma_\mathtt{q}}} = \mathtt{split}(\alpha){\Downarrow_{\Sigma_\mathtt{q}}}$, and w satisfies (F2), it follows that $\mathtt{split}(\alpha){\Downarrow_{\Sigma_\mathtt{p}}} \leq w{\Downarrow_{\Sigma_\mathtt{p}}}$. Thus, all

send events from p to q prescribed by α are not unmatched. We can split $w_1 = u_1 \cdot \mathsf{r} \triangleright \mathsf{s}!m \cdot u_2$ such that there are no unmatched sends from α_1 in u_2. It follows from (B5) that $u_1 \cdot u_2 \cdot \mathsf{q} \triangleleft \mathsf{p}?m$ is channel-compliant, contains one fewer unmatched send from α, and is compliant with $\gamma_3\gamma_4$. The claim thus follows from the induction hypothesis.

Finally, let α_3 be α_2 with all events that do not occur in w_2 removed. From the two steps above, it is clear that $\alpha_3 = \varepsilon$, and thus w_2 is compliant with β, $w_2 \cdot \mathsf{q} \triangleleft \mathsf{p}?m$ is channel-compliant, and contains no events of q. This concludes our proof that w_2 serves as a witness for w'. □

A.2 Proofs for §7.1

Lemma 7.5. *For $\mathbb{A} \in \{\mathsf{p2p}, \mathsf{sb}, \mathsf{bag}\}$, for every $\rho \in \Gamma^*_{sync}$, $w \in \Sigma^*_{async}$ such that w is $\mathbb{A}$-channel compliant and agrees with ρ, there exists $u \in \Sigma^*_{async}$ such that wu is channel compliant, and $wu \equiv_{\mathcal{P}} \mathtt{split}(\rho)$.*

Proof. We prove the claim by induction on the length of w. In the base case, let $u = \mathtt{split}(\rho)$. By (F3), u is channel-compliant, and by construction, $u \equiv_{\mathcal{P}} \mathtt{split}(\rho)$. In the induction step, let $w = w'x$. From the induction hypothesis, we are given u' such that $w'u'$ is channel-compliant and $w'u' \equiv_{\mathcal{P}} \mathtt{split}(\rho)$. Let p be the active participant in x, and let u_1 be the maximal prefix of u' with respect to ε for p. By definition, $u_1 \Downarrow_{\Sigma_\mathsf{p}} = \varepsilon$, and the next symbol in u' following u_1 has p as its active participant. Moreover, because both $w'x\Downarrow_{\Sigma_\mathsf{p}}$ and $w'u'\Downarrow_{\Sigma_\mathsf{p}}$ are prefixes of $\rho\Downarrow_{\Sigma_\mathsf{p}}$, the next symbol in u' following u_1 equals x. Thus, we can write $u' = u_1yu_2$. Let $u = u_1u_2$. Clearly, u is per-role equal to $\mathtt{split}(\rho)$. To show that $w'xu_1u_2$ is channel-compliant we require the following facts for the various considered network architectures. Let $v, u_1, u_2 \in \Sigma^*_{async}$ and $z \in \Sigma_{async}$.

Claim 1. If vu_1zu_2 is p2p, sb, or bag channel-compliant, $z = \mathsf{p} \triangleright \mathsf{q}!m$ and $u_1\Downarrow_{\Sigma_\mathsf{p}} = \varepsilon$, then vzu_1u_2 is respectively p2p, sb or bag channel-compliant.

Claim 2. If vu_1zu_2 is p2p or bag channel-compliant, $z = \mathsf{q} \triangleleft \mathsf{p}?m$ and $u_1\Downarrow_{\Sigma_\mathsf{q}} = \varepsilon$, then vzu_1u_2 is respectively p2p and bag channel-compliant.

For sb channel compliance when z is a receive event, we require slightly stronger assumptions on u_1.

Claim 3. If vu_1zu_2 is sb channel-compliant, $z = \mathsf{q} \triangleleft \mathsf{p}?m$, $u_1\Downarrow_{\Sigma_\mathsf{p}} = \varepsilon$, $u_1\Downarrow_{\mathsf{p}\triangleright\text{-}!\text{-}} = \varepsilon$, and $v\Downarrow_{\text{-}\triangleleft\mathsf{p}?\text{-}} \cdot z \leq v\Downarrow_{\mathsf{p}\triangleright\text{-}!\text{-}}$, then vzu_1u_2 is sb channel-compliant.

When $z = \mathsf{q}\triangleleft\mathsf{p}?m$, we reason from the fact that $w' \cdot \mathsf{q}\triangleleft\mathsf{p}?m$ is sb channel-compliant, $w'u_1 \cdot \mathsf{q}\triangleleft\mathsf{p}?m$ is sb channel-compliant, and moreover q has no events in u_1, that q does not receive the message from p in u_1, and thus no other participants receives messages from p's sb in u_1. Thus, the assumptions required to reorder z ahead of u_1 are satisfied. □

A.3 Complexity

Theorem 5.5 immediately yields a decision procedure for implementability of finite protocols for $\mathbb{A} \in \{\mathsf{p2p}, \mathsf{sb}, \mathsf{mb}, \mathsf{monob}, \mathsf{bag}\}$. Li et al. [55] showed that the implementability problem for finite global protocols on a p2p network is co-NP-complete. We first show that this result extends to all homogeneous network architectures under consideration, by examining the complexity of problem in light of relationships between the avail predicates for each network. Given our discussion of how to decide the generalized Coherence Conditions in §7, it suffices to argue that (a) avail for sb and bag are co-NP-complete, and (b) avail′ for mb and monob are co-NP-complete. It is easy to see that witnesses for the above are verifiable in polynomial time.

The proof of the co-NP lower bound by Li et al. [55] works by a reduction from 3-SAT to implementability. The proof assumes a 3-SAT instance $\varphi = C_1 \wedge \ldots \wedge C_k$ with variables $x_1, \ldots, x_n$ and literals L_{ij}, denoting the jth literal of clause C_i, with $1 \leq i \leq k$ and $1 \leq j \leq 3$. From this, it

constructs a global protocol $\mathcal{S}_\varphi$ such that ϕ is unsatisfiable iff $\mathcal{S}_\varphi$ is implementable. We summarize the construction pictorially in Fig. 7. We show that the same lower bound construction with a small modification works for the remaining network architectures.

The construction relies on two gadgets: $\mathcal{S}_X$, a gadget that encodes a variable assignment to variables $x_1, \ldots, x_n$ (Fig. 8), and $\mathcal{S}_C$, a gadget that encodes literal selection for clauses $C_1, \ldots, C_k$ (Fig. 9). The highlighted message in Fig. 7 is available for participant q in q_2 if and only if φ is satisfiable. Consequently, protocol $\mathcal{S}_\varphi$ is non-implementable if and only if the highlighted message is available for participant q in q_2, if and only if φ is satisfiable. In order to show that the construction carries over to bag and senderbox networks, one is only required to analyze the unique message receptions that appear in each gadget. Thus, Fig. 8 and Fig. 9 each depict the smallest portion of each gadget necessary to establish our new complexity results. We refer the reader to [55] for the full details of the construction.

First, we consider the bag network architecture. As established in §7, any message available in a p2p network is also available in a bag network. Thus, we only need to show that the rest of $\mathcal{S}_\varphi$ is implementable, which amounts to checking that no other bag Receive Coherence violations occur.

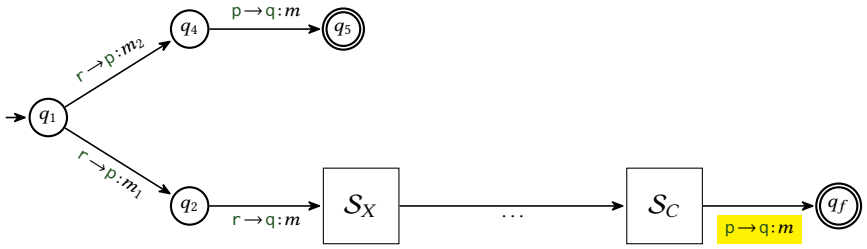

Fig. 7. Illustration of 3-SAT reduction for implementability from [55].

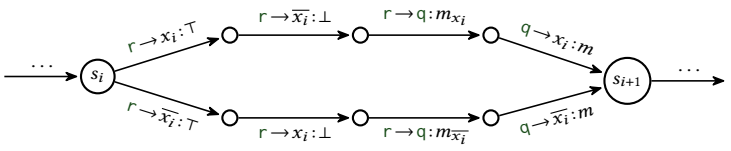

Fig. 8. Illustration of modified variable assignment gadget $\mathcal{S}_X$.

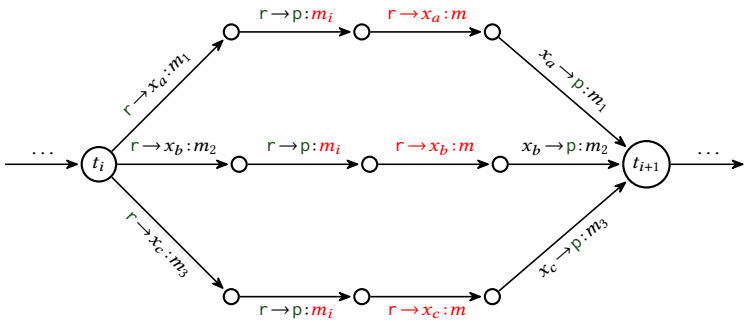

Fig. 9. Illustration of modified clause selection gadget $\mathcal{S}_C$. Modifications are highlighted in red.

Participant r does not receive messages, and can thus be ignored. Unlike p2p Receive Coherence, bag Receive Coherence additionally constrains pairs of receptions from the same sender. For the variable participants in $\mathcal{S}_X$, each participant receives either a $\bot$ from r, or a $\top$ followed by an m message from q. Thus, bag Receive Coherence is satisfied. Inspecting $\mathcal{S}_C$, each variable participant only receives one kind of message, which is m from r, and if so it sends an m message to p. Thus, bag Receive Coherence is satisfied as well. Participant p is uninvolved in $\mathcal{S}_X$, but in $\mathcal{S}_C$ receives m messages from r which tells it to anticipate a message from some variable participant. The original encoding uses the same message payload from r to tell p to anticipate a message, but we can modify the construction to let r send p a message encoding precisely which variable participant to anticipate a message from. This eliminates what would otherwise constitute a bag Receive Coherence violation for p, since r and the variable participants can overtake p's receptions. Finally, onto participant q, who is uninvolved in $\mathcal{S}_C$ and only involved in $\mathcal{S}_X$, q receives exactly n messages from r, that constitute n binary choices between receiving m_{x_i} and $m_{\bar{x_i}}$ interrupted by send events from q. Thus, we can conclude that the modified construction is non-implementable iff $\mathsf{bagavail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, q_3)$ holds in $\mathcal{S}_\varphi$ iff φ is satisfiable.

Next, we consider the sb network architecture. Because p2p implementability implies sb implementability, in this case we only need to independently establish that $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, q_3)$ holds for the sb avail setting. As illustrated above, senderbox receivability can be undermined by messages from the same sender to different receivers that are blocked, so we need to check whether any such messages from p to other receivers appear in the subprotocols $\mathcal{S}_X$ and $\mathcal{S}_C$. It is easy to see that no such messages appear, and thus senderbox implementability still holds.

For monob and mb, we establish that $\mathsf{avail}'_{\mathsf{p},\{\mathsf{r},\mathsf{p}\}}(q_2)$ holds if and only if φ is satisfiable. In gadget $\mathcal{S}_X$, no causally independent receptions to the same receiver occur along the same run. In gadget $\mathcal{S}_C$, the addition of the extra message exchange, $\mathsf{r} \to x_a : m$ enforces that the messages from r and x_a cannot be independently reordered in p's mailbox. Thus, the construction maintains that the only possible violation to Generalized Coherence Conditions lies in the availability of the highlighted message from state q_2.

Furthermore, we argue that the construction can be adapted to multiparty session types with directed choice, a syntactically defined fragment of finite global protocols, whose definition is given below:

Global Multiparty Session Types. Global types for MSTs [43] are defined by the grammar:

$$G ::= 0 \mid \sum_{i \in I} \mathsf{p} \to \mathsf{q} : m_i.G_i \mid \mu t.\ G \mid t$$

where p, q range over $\mathcal{P}$, m_i over a finite set $\mathcal{V}$, and t over a set of recursion variables.

Note that the top-level choice in Fig. 7 satisfies directed choice, and thus we only require to modify the assignment and clause selection gadgets. We depict the gadgets modified to satisfy directed choice in Fig. 10 and Fig. 11. The encoding from directed choice global protocols to directed choice multiparty session types follows [56], and we refer the reader to their appendix for details.

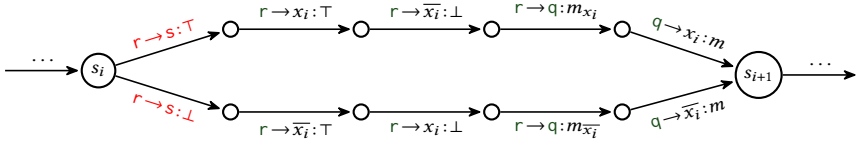

Fig. 10. Illustration of directed choice variable assignment gadget $\mathcal{S}_X$. Modifications are highlighted in red.

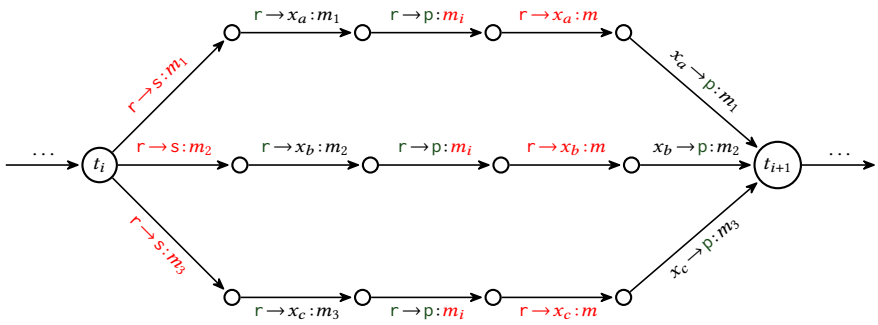

Fig. 11. Illustration of directed choice clause selection gadget $\mathcal{S}_C$. Modifications are highlighted in red.