Characterizing Implementability of Global Protocols with Infinite States and Data

ELAINE LI*, New York University, USA
FELIX STUTZ, University of Luxembourg, Luxembourg
THOMAS WIES, New York University, USA
DAMIEN ZUFFEREY, SonarSource, Switzerland

We study the implementability problem for an expressive class of symbolic communication protocols involving multiple participants. Our symbolic protocols describe infinite states and data values using dependent refinement predicates. Implementability asks whether a global protocol specification admits a distributed, asynchronous implementation, namely one for each participant, that is deadlock-free and exhibits the same behavior as the specification. We provide a unified explanation of seemingly disparate sources of non-implementability through a precise semantic characterization of implementability for infinite protocols. Our characterization reduces the problem of implementability to (co)reachability in the global protocol restricted to each participant. This compositional reduction yields the first sound and relatively complete algorithm for checking implementability of symbolic protocols. We use our characterization to show that for finite protocols, implementability is co-NP-complete for explicit representations and PSPACE-complete for symbolic representations. The finite, explicit fragment subsumes a previously studied fragment of multiparty session types for which our characterization yields a PTIME decision procedure, improving upon a prior PSPACE upper bound.

Additional Key Words and Phrases: Protocol verification, Multiparty session types, Refinement

1 INTRODUCTION

Concurrency is ubiquitous in modern computing, message-passing is a major concurrency paradigm, and communication protocols are therefore a key target for formal verification. Communication protocols specify distributed, message-passing behaviors from a global point of view, altogether describing the interactions between all participants in the protocol. Implementability and synthesis are two central questions to the verification of communication protocols. Implementability asks whether a protocol admits a distributed implementation, and synthesis in turn computes an admissible one. A distributed implementation is considered admissible if it is deadlock-free and exhibits exactly the same communication behaviors described by the specification. We refer to the latter property as *protocol fidelity*. The implementability question precedes the synthesis question in importance: synthesizing implementations for unrealizable protocols is a fruitless endeavor.

Global protocol specifications find industry applications in the form of UML's high-level message sequence charts and the Web Service Choreography Description Language, and are widely studied in academia in the form of multiparty session types and choreographic programming. Multiparty session types have been implemented in at least 16 programming languages [12, 43, 47, 49, 57, 60], and applied to operating systems [24], high performance computing [40], cyber-physical systems [53], web services [70], and smart contracts [20]. We refer the reader to [4] for a comprehensive survey of MST and choreography applicability respectively.

To model real-world verification targets, we desire for our protocol specifications to be as expressive as possible. Various dimensions of expressivity have been explored in the literature, such as arbitrary message payloads, non-deterministic choice, unrestricted recursion and parametricity.

*corresponding author

Authors' Contact Information: Elaine Li, New York University, USA; Felix Stutz, University of Luxembourg, Luxembourg; Thomas Wies, New York University, USA; Damien Zufferey, SonarSource, Switzerland.

Formalisms such as choreography automata [30], high-level messages sequence charts [1, 3, 26–29, 50, 54–56, 58] and session types [7, 8, 41, 48, 65, 72] correspond to syntactically-defined fragments that incorporate a selection of these features.

In this paper, we study the implementability problem for a semantically-defined class of communication protocols, which we call *global communicating labeled transition systems (GCLTS)*. GCLTS impose only modest syntactic restrictions and subsume many existing fragments of asynchronous multiparty session types and choreography automata. GCLTS captures the following important features:

- Asynchrony: the semantics are interpreted over a peer-to-peer, asynchronous network, with FIFO channels connecting each pair of protocol participants.
- Generalized sender-driven choice: the only notable syntactic restriction imposed by our formalism is that at each branching point of the protocol's control flow, a single participant chooses a branch. In other words, the first message that is sent in each branch of a choice must come from the same sender. However, we impose no restrictions on the recipient or the message payload other than that no two branches share the same recipient and message.
- Infinite protocol state: protocol states contain registers that take values from an infinite domain. This allows loops to carry memory across iterations, and allows the protocol to be specified in terms of dependent refinement predicates.
- Infinite message payloads: messages can carry values drawn from an infinite data domain.

Implementability is undecidable for this general class of protocols. The presence of and interaction between the aforementioned features means that even soundly approximating implementability is challenging. Existing work is either comparable in expressivity but does not solve the implementability problem, or solves the implementability problem but is incomparably restricted in expressivity. Zhou et al. [72] present a framework for synchronous, refined multiparty session types that soundly approximates implementability through its endpoint projection, but the latter may yield local specifications that are not implementable. [2, 48, 50, 63] precisely characterize implementability for finite protocol specifications. Furthermore, the implementability check in [2, 48] relies on synthesizing an implementation upfront, which is not possible for infinite state protocols. Das and Pfenning [21] study local session types with arithmetic refinements in a binary setting.

We address these challenges by decomposing the implementability problem into two steps. First, we give a precise, semantic characterization of implementability for GCLTS that we prove sound and complete once and for all. Our characterization is defined directly on the global specification, and thus forgoes the need to first synthesize a candidate implementation. Moreover, our characterization gives a unified semantic explanation to disparate causes of non-implementability that arise from the expressivity of our protocol fragment. We encapsulate the complexities introduced by communication-specific features such as asynchrony and partial information in the first step. Our semantic characterization reduces implementability to (co)reachability in the GCLTS. Second, we use this reduction to obtain a blueprint for solving implementability algorithmically. Because the reduction abstracts away from communication-specific features, we can use existing verification technology for checking (co)reachability in non-distributed transition systems. Our approach yields new algorithms that are sound and relatively complete for checking implementability of symbolic protocols, in addition to decision procedures with optimal complexity for various decidable classes. In particular, we show that implementability for the class of communication protocols considered in [48] can be decided in PTIME, improving upon the PSPACE bound obtained by explicit synthesis.

Contributions. In summary, our contributions are:

- Global communicating labeled transition systems (GCLTS): a semantically-defined class of asynchronous communication protocols that subsumes most formalisms in the literature.
- A precise characterization of implementability for GCLTS.
- The first symbolic algorithm for checking implementability of infinite, symbolic protocols that is sound and relatively complete.
- Optimal decision procedures for checking implementability of finite protocols. In particular, we show that the problem is co-NP-complete for finite protocols with explicit states and transitions and PSPACE-complete for symbolic finite protocols.
- As a corollary of the previous result, we obtain a PTIME decision procedure for implementability of global types [48], improving upon a prior PSPACE upper bound.

2 OVERVIEW

We motivate our work using an infinite state version of a two-bidder protocol, depicted as a high-level message sequence chart (HMSC) in Fig. 1. The protocol specifies the behavior of two bidders, B_1 and B_2, who negotiate to split the cost of purchasing a book from seller S.

The protocol begins with B_1 announcing to S and B_2 the book y it proposes to buy. The protocol requires that y signifies a valid ISBN number, which we abstract with the predicate $\mathsf{ISBN}(y)$. The seller S then informs B_1 the requested book's price z. After this, B_1 and B_2 enter a bidding phase in which they negotiate the split of their respective contributions b_1 and b_2 towards the purchase. In each round of the bidding phase, B_1 proposes its contribution b_1 to B_2. Bidder B_2 then decides to either abort the protocol by sending a quit message to S, or respond to B_1 with its own bid b_2. In case B_2 aborts, S echoes the abort message to B_1 and the protocol terminates. In the other case, if the sum of the proposed bids exceeds the book's price, B_1 informs S of the successful negotiation. Seller S in turn relays the message to B_2. Otherwise, B_1 sends a cont message to B_2, informing them that they need to enter another bidding round. Throughout the bidding phase, B_1 and B_2 track the values of their latest bids in the registers z_1 and z_2. The protocol ensures that the proposed bids are strictly increasing from one round to the next, thus enforcing termination.

Figs. 2 to 4 show an admissible implementation for the two-bidder protocol in Fig. 1, corresponding to a local implementation for each participant: S, B_1 and B_2. The transition labels specify their local behaviors: $\mathsf{B}_1 \triangleright \mathsf{S}!y\{\mathsf{ISBN}(y)\}$ specifies that B_1 sends a message y to S such that the condition $\mathsf{ISBN}(y)$ holds, i.e. y is a valid ISBN number; $\mathsf{S} \triangleleft \mathsf{B}_1?y\{\mathsf{ISBN}(y)\}$ specifies that S receives y from B_1,

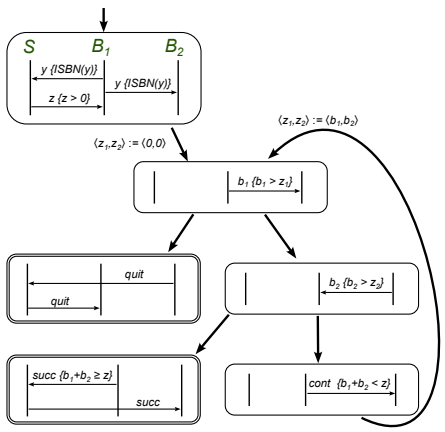

Fig. 1. Two-bidder protocol.

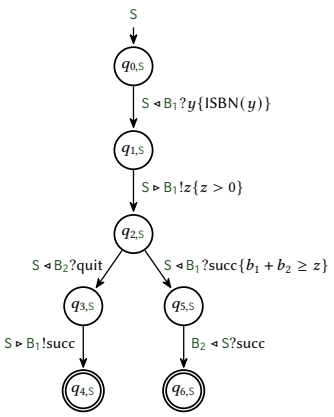

Fig. 2. State machine for seller S for Fig. 1.

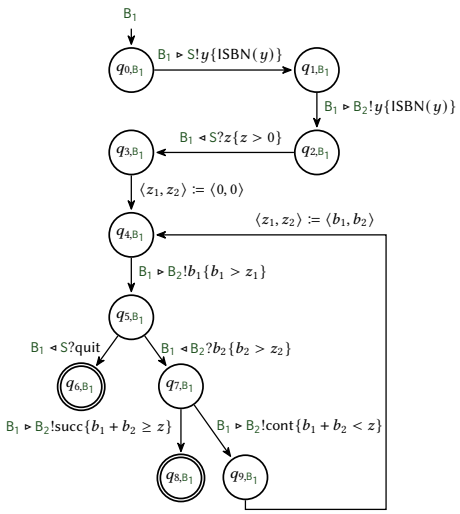

Fig. 3. State machine for bidder B_1 for Fig. 1.

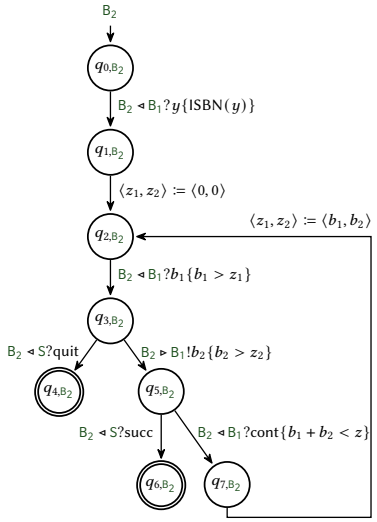

Fig. 4. State machine for bidder B_2 for Fig. 1.

and can assume that $\mathsf{ISBN}(y)$ about y. We assume an asynchronous setting in which every pair of participants is connected by a FIFO channel. In this case, Fig. 1 is implementable and Figs. 2 to 4 together yield an implementation, i.e. exhibits the same behaviours as the global protocol when executed together, and never gets stuck.

To see that the implementability problem is non-trivial, consider a variant of the protocol in Fig. 1 where the succ message to S is sent by B_2 instead of B_1. The resulting protocol is no longer implementable because B_2 never learns about the price z of the book y and is therefore unable to determine when the negotiation with B_1 has succeeded.

Our example features several important expressive features of GCLTS:

- Generalized sender-driven choice: after B_2 receives a bid from B_1, it has the option to either send a bid back to B_1 and continue the bidding process, or terminate the protocol by sending a quit message to the bookseller, who then relays the termination message to the first bidder. Due to this choice interaction alone, the protocol is not expressible in [72].
- Infinite state: the protocol state contains an assignment of registers to values from an infinite domain. Registers are updated to store the last bid from each round z_1 and z_2, to enforce that bidders make strictly increasing bids per round.
- Infinite message data: messages carry payload values drawn from an infinite data domain, such as the book price z and bids b_1 and b_2.
- Dependent refinement predicates: message payloads are constrained by data refinements such as $z_1 < b_1$ and $z < b_1 + b_2$. The refinement predicates can refer to current register values in addition to data values sent in prior messages.
- Partial information: each protocol participant only has a partial view of the global protocol state. For example, even though S participates in the bidding phase of the protocol, it never learns about the bids b_1 and b_2 in each bidding round. In fact, the registers z_1 and z_2 that store the last bid are known only to the bidders.

The presence of these features makes checking implementability for the class of communication protocols we consider uniquely challenging. For protocols with finite GCLTS specifications, deciding implementability in the presence of asynchrony and non-deterministic choice already presents

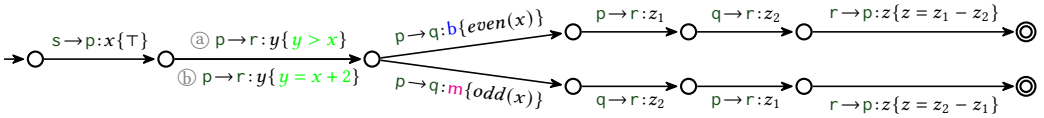

Fig. 5. Two protocols: $\mathbb{S}_1$ using ⓐ with receive order violation $\mathbb{S}_1'$ using ⓑ without receive order violation.

a challenge. Note that finiteness here refers only to the specification, and does not mean that the underlying protocol is finite-state, nor that it contains only finite traces. Most existing work has therefore focused on developing projection operators that are sound but incomplete [14, 41, 59, 65]. These projection operators solve implementability and synthesis simultaneously by computing a candidate implementation, but often fail eagerly for protocols for which an implementation exists. Li et al. [48] proposed the first sound and complete projection operator for finite, asynchronous, multiparty session types. The projection operator critically relies on the observation that if a global type is implementable, then a canonical implementation implements it. Thus, the implementability problem reduces to checking whether this canonical implementation indeed implements the global type, i.e. it recognizes the same set of behaviors and is deadlock-free. Towards these ends, [48] identifies sound and complete conditions, referred to as *Send Validity* and *Receive Validity*, that are checked on the states of the canonical implementation.

In the presence of dependent refinement predicates, checking these conditions is not straightforward. Consider the examples $\mathbb{S}_1$ (using ⓐ) and $\mathbb{S}_1'$ (using ⓑ) in Fig. 5, which are variations of the examples for Receive Validity from [48] featuring dependent predicates. Here, $\mathtt{p} \to \mathtt{r} : y\{y > x\}$, which is ⓐ for $\mathbb{S}_1$, atomically specifies the send event by p and the corresponding receive event by r, along with the constraint that y satisfies $y > x$. We first consider $\mathbb{S}_1$. participant p chooses a branch without explicitly informing r of their choice. In both branches, r is required to subtract the second value that is sent from the first value that is sent, and send the result back to p. However, due to asynchrony, both messages can arrive in r's message channels simultaneously, and r cannot tell which value was sent first. Therefore, r may subtract the values in the wrong order, rendering the protocol unimplementable.

In [48], this situation is resolved by introducing a dependency between the messages from p and q on each branch, so that they cannot appear in r's channels at the same time. The symbolic protocol gives way to a different method of resolution. This is applied for $\mathbb{S}_1'$: the predicate on the second transition is changed from $y > x$ to $y = x + 2$. Despite the fact that r is still not informed of p's choice, r can *infer* p's choice through the parity of the first value it received from p. Here, instead of introducing a dependency between p and q, we introduced a dependency between p's choice of payload value and choice of branch. The knowledge of this dependency enables r to use the parity of y to correctly follow the protocol.

We now turn our attention to send violations. In the protocol shown in Fig. 6, s chooses a branch and communicates its choice to q. Participant p is again not explicitly informed of the choice: in fact, p can receive 4 from q on both branches. At first glance, it appears as though it is safe for p

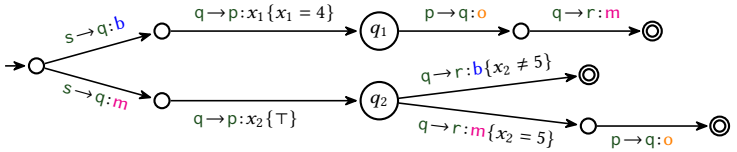

Fig. 6. $\mathbb{S}_2$: An protocol with a send violation.

to send o to q upon receiving 4 from q, because whilst p cannot distinguish the two branches, both branches contain the transition $\mathsf{p} \to \mathsf{q} : o$. Upon closer inspection, the predicate guarding the transition immediately preceding $\mathsf{p} \to \mathsf{q} : o$ on the lower branch, $x_2 = 5$, is only satisfied when p receives 5 from q. When p receives 4, the lower branch from q_2 is disabled, and since the upper branch from q_2 does not contain the transition $\mathsf{p} \to \mathsf{q} : o$, the protocol is not implementable.

The examples above exemplify the ways in which refinement predicates complicate implementability checking for symbolic protocols. We return to these examples, in addition to some others, in greater detail in §4 when we present our precise characterization of implementability. We structure the rest of the paper as follows. §3 presents relevant preliminary definitions and defines the class of communication protocols we consider. §4 presents our semantic characterization of implementability for GCLTS in terms of (co)reachability, and proves that it is precise. §5 describes our sound and complete reduction from the characterization in §4 to logical formulas in μCPL [67], and additionally presents improved complexity results under certain finiteness assumptions on the GCLTS. §6 discusses related work and concludes.

3 PRELIMINARIES

We introduce some basic concepts and notation before defining our class of protocols.

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 $\mathrm{pref}(u)$. 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 finite set of participants and $\mathcal{V}$ be a (possibly infinite) data domain. We define the set of *synchronous events* $\Gamma := \{\mathsf{p} \to \mathsf{q} : m \mid \mathsf{p}, \mathsf{q} \in \mathcal{P} \text{ and } m \in \mathcal{V}\}$ where $\mathsf{p} \to \mathsf{q} : m$ denotes a message exchange of m from sender p to receiver q. For each participant $\mathsf{p} \in \mathcal{P}$, we define a homomorphism $\Downarrow_{\Gamma_\mathsf{p}}$, where $x\Downarrow_{\Gamma_\mathsf{p}} = x$ if p is either the sender or receiver in x, and ε otherwise. A synchronous event is split into a send and receive event for the respective participant, yielding *asynchronous events*. 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, $\Sigma = \Sigma_! \cup \Sigma_?$. We define a homomorphism to map the synchronous alphabet to its asynchronous counterpart, defined as $\mathtt{split}(\mathsf{p} \to \mathsf{q} : m) := \mathsf{p} \triangleright \mathsf{q}!m.\ \mathsf{q} \triangleleft \mathsf{p}?m$. Because $\mathtt{split}$ is injective, there exists a unique inverse, which we denote $\mathtt{split}^{-1}$. We say that p is *active* in $x \in \Sigma$ 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 write $\mathcal{V}(w)$ to project the send and receive events in w onto their messages. We fix $\mathcal{P}$ and $\mathcal{V}$ in the remainder of the paper.

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 $(p, \alpha, q) \in \Gamma$. 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. Given an LTS $\mathcal{S} = (S, \Gamma, T, s_0, F)$ and a state $s \in S$, we use $\mathcal{S}_s$ to denote the LTS obtained by replacing s_0 with s as the initial state: (S, Γ, T, s, F).

3.1 Global Communicating Labeled Transition Systems (GCLTS)

We use LTSs over the synchronous alphabet Γ to model communication protocols from a global perspective. We impose three more conditions on the class of LTSs we consider: that final states do not contain outgoing transitions, that multiple outgoing transitions from a state share a sender, and that the LTS is deadlock-free.

Definition 3.1 (Global communicating LTS). An LTS $\mathcal{S} = (S, \Gamma, T, s_0, F)$ is a *global communicating labeled transition system* (GCLTS) if the following conditions hold:

(1) *sink-finality*: for every final state $s \in F$, there does not exist $l \in \Gamma$ and $s' \in S$ with $s \xrightarrow{l} s' \in T$;
(2) *sender-driven choice*: for all states $s, s_1, s_2 \in S$ and $l_1, l_2 \in \Gamma$ such that $s \xrightarrow{l_i} s_i \in T$ for $i \in \{1, 2\}$, there is a participant $\mathtt{p} \in \mathcal{P}$ who is the sender for both, i.e. $\mathtt{split}(l_i) \in \Sigma_{\mathtt{p},!}$ for $i \in \{1, 2\}$, and furthermore $l_1 = l_2 \implies s_1 = s_2$;
(3) *deadlock freedom*: $\mathcal{S}$ is deadlock-free.

Condition (1) is ubiquitous in the domain of multiparty session types and was also shown to require special treatment in the literature on high-level message sequence charts [17].

Condition (2) is a generalisation of most multiparty session types fragments, which require not only a dedicated sender but also a dedicated receiver. This more restrictive condition is called *directed choice*. *Mixed choice* lifts all restrictions on choice, and amounts to only requiring determinism. [50] showed that realizability is undecidable even for HMSC satisfying the other two conditions, but with determinism instead of Condition (2). Sender-driven choice thus represents a good middle ground, allowing to express interesting communication patterns while retaining decidability of implementability.

Condition (3) simply requires that protocols do not specify deadlocking behaviors.

For the remainder of the paper we refer to a GCLTS simply as a *protocol*.

Restricting Protocols to Participants. From a protocol $\mathcal{S}$, we can define a local protocol for each participant $\mathtt{p}$ via domain restriction to $\Sigma_{\mathtt{p}}$. Formally, given a protocol $\mathcal{S} = (S, \Gamma, T, s_0, F)$, we define $\mathcal{S}_{\mathtt{p}} \coloneqq (S, \Gamma_{\mathtt{p}} \uplus \{\varepsilon\}, T_{\mathtt{p}}, s_0, F)$ where $T_{\mathtt{p}} \coloneqq \{s \xrightarrow{l\Downarrow_{\Gamma_{\mathtt{p}}}} s' \mid s \xrightarrow{l} s' \in T\}$ for a participant $\mathtt{p} \in \mathcal{P}$.

Asynchronous Protocol Semantics. Note that a protocol is specified using the synchronous alphabet Γ. To define the *asynchronous* semantics of a protocol $\mathcal{S}$ we first map finite and infinite words of $\mathcal{S}$ onto their asynchronous counterpart using $\mathtt{split}$, thus obtaining a set of asynchronous words in which matching send and receive events are adjacent to each other. In an asynchronous, FIFO network, two events are independent if they are not related by the *happened-before* relation [46]. For example, any two send events from distinct senders are independent. Consequently, two words are *indistinguishable* if any CSM that recognizes one word must recognize the other, e.g. $w.\mathtt{p}{\triangleright}\mathtt{q}!m.\mathtt{r}{\triangleright}\mathtt{s}!m'.u$ and $w.\mathtt{r}{\triangleright}\mathtt{s}!m'.\mathtt{p}{\triangleright}\mathtt{q}!m.u$, where $\mathtt{p} \neq \mathtt{r}$. We define our protocol semantics as the set of *channel-compliant* [51] words that are closed under this notion of indistinguishability. Channel compliance characterizes words that respect FIFO order, i.e. receive events appear after their matching send event, and the order of receive events follows that of send events in each channel.

Definition 3.2 (Channel compliance). Let $w \in \Sigma^\infty$. We say that w is *channel-compliant* if for all prefixes $w' < w$, for all $\mathtt{p} \neq \mathtt{q} \in \mathcal{P}$, $\mathcal{V}(w'\Downarrow_{\mathtt{q}\triangleleft\mathtt{p}?\_}) < \mathcal{V}(w'\Downarrow_{\mathtt{p}\triangleright\mathtt{q}!\_})$.

The asynchronous semantics of a protocol is defined as follows:

$$
\begin{aligned}
C^{\frown}(\mathcal{S}) = \{w' \in \Sigma^* \mid \exists w \in \Sigma^*.\, w \in \mathtt{split}(\mathcal{L}(\mathcal{S})) \wedge w' \text{ is channel-compliant } \wedge \forall \mathtt{p} \in \mathcal{P}.\, w'\Downarrow_{\Sigma_{\mathtt{p}}} = w\Downarrow_{\Sigma_{\mathtt{p}}}\} \\
\cup \{w' \in \Sigma^\omega \mid \exists w \in \Sigma^\omega.\, w \in \mathtt{split}(\mathcal{L}(\mathcal{S})) \wedge \forall v' \leq w'.\, \exists u, u' \in \Sigma^*. \\
v'u' \text{ is channel-compliant } \wedge\ u \leq w\ \wedge \forall \mathtt{p} \in \mathcal{P}.\, v'u'\Downarrow_{\Sigma_{\mathtt{p}}} = u\Downarrow_{\Sigma_{\mathtt{p}}}\}\ .
\end{aligned}
$$

Membership of infinite words is defined in terms of their prefixes: every prefix v' must be channel-compliant, and moreover extensible to a word $v'u'$ that is indistinguishable from another prefix already in the language. Since we do not make any fairness assumptions on scheduling, the semantics of infinite words includes traces such as $(\mathsf{p} \triangleright \mathsf{q}!m)^\omega$ for the protocol $(\mathsf{p} \triangleright \mathsf{q}!m.\mathsf{q} \triangleleft \mathsf{p}?m)^\omega$, where only the sender is scheduled. Membership of finite words follows standard MSC semantics.

In the remainder of the paper, we overload the notation and use $\mathcal{L}(\mathcal{S})$ to denote $\mathcal{C}^\frown(\mathcal{S})$.

Communicating LTSs. We use communicating LTSs to model the local behaviors of participants: $\mathcal{T} = \{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ is a *communicating labeled transition system* (CLTS) over $\mathcal{P}$ and $\mathcal{V}$ if T_p is a deterministic LTS over Σ_p for every $\mathsf{p} \in \mathcal{P}$, denoted by $(Q_\mathsf{p}, \Sigma_\mathsf{p}, \delta_\mathsf{p}, q_{0,\mathsf{p}}, F_\mathsf{p})$. Let $\prod_{\mathsf{p}\in\mathcal{P}} Q_\mathsf{p}$ denote the set of global states and $\mathsf{Chan} = \{(\mathsf{p},\mathsf{q}) \mid \mathsf{p},\mathsf{q} \in \mathcal{P}, \mathsf{p} \neq \mathsf{q}\}$ denote the set of channels. A *configuration* of $\mathcal{A}$ is a pair $(\vec{s}, \xi)$, where $\vec{s}$ is a global state and $\xi : \mathsf{Chan} \to \mathcal{V}^*$ is a mapping from each channel to a sequence of messages. We use $\vec{s}_\mathsf{p}$ to denote the state of p in $\vec{s}$. The communicating LTS transition relation, denoted $\rightarrow$, 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 $\mathsf{r} \neq \mathsf{p}$, $\xi'(\mathsf{p},\mathsf{q}) = \xi(\mathsf{p},\mathsf{q}) \cdot m$ and $\xi'(c) = \xi(c)$ for every other channel $c \in \mathsf{Chan}$.
- $(\vec{s}, \xi) \xrightarrow{\mathsf{q}\triangleleft\mathsf{p}?m} (\vec{s}', \xi')$ 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 $\mathsf{r} \neq \mathsf{q}$, $\xi(\mathsf{p},\mathsf{q}) = m \cdot \xi'(\mathsf{p},\mathsf{q})$ and $\xi'(c) = \xi(c)$ for every other channel $c \in \mathsf{Chan}$.

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 ε. A configuration $(\vec{s}, \xi)$ is *final* iff $\vec{s}_\mathsf{p}$ is final for every p and ξ maps each channel to ε. 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 $\mathcal{L}(\mathcal{T})$ of the communicating LTS $\mathcal{T}$ is defined as the set 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.

Observe that in a CLTS, send transitions are always enabled, whereas receive transitions are only enabled if the message exists at the head of its corresponding channel. Communicating state machines [9] are a special case of communicating LTS where the LTS for each participant $\mathsf{p} \in \mathcal{P}$ is a deterministic finite state machine. Finally, we define protocol implementability.

Definition 3.3 (Protocol Implementability). A protocol $\mathcal{S}$ is *implementable* if there exists a CLTS $\{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ such that the following two properties hold: (i) *protocol fidelity*: $\mathcal{L}(\{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}}) = \mathcal{L}(\mathcal{S})$, and (ii) *deadlock freedom*: $\{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ is deadlock-free. We say that $\{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ implements $\mathcal{S}$.

3.2 Symbolic protocols with dependent refinements

We now introduce our model for finitely representing infinite state protocols. We refer to these representations simply as *symbolic protocols*. Figure 7 shows the two-bidder protocol from Fig. 1 expressed as a symbolic protocol.

The formal definition of this class of symbolic protocols is given below. In this definition, we assume 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 some set X. We assume that these variables are interpreted over the set of message values $\mathcal{V}$. For a valuation $\rho \in X \to \mathcal{V}$ and $\varphi \in \mathcal{F}(X)$, we write $\rho \models \phi$ to indicate that φ evaluates to true under ρ in the underlying theory.

Definition 3.4 (Symbolic protocol). 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,

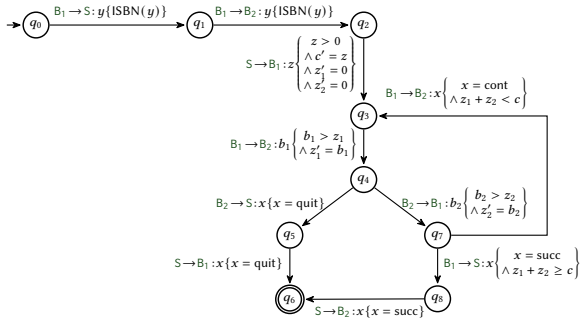

Fig. 7. The two bidder protocol from Fig. 1 as symbolic protocol with three registers c, z_1, and z_2.

- $\Delta \subseteq S \times \mathcal{P} \times X \times \mathcal{P} \times \mathcal{F} \times S$ 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.

To streamline our definition, we choose not to separate register update expressions from predicates describing the communication. Rather, we specify everything together in a single formula φ, that can only talk about the current value being communicated, and the register values in the pre- and post-state. Thus, φ can describe values that are communicated between participants, in addition to register assignments and updates. For example, $\mathsf{p} \to \mathsf{q} : x\{even(x) \wedge r_1' = r_1 + 2 \wedge r_2' = x\}$ describes p sending q an even number x, storing the value of x in register r_2, and incrementing the value of register r_1 by 2. We formally specify the two-bidder protocol from Fig. 1 as a symbolic protocol in Fig. 7 for demonstration purposes. For readability and conciseness, we employ the following conventions from now on. We treat communication variables as registers that are automatically assigned the communicated value, e.g. $\mathsf{S} \to \mathsf{B}_1 : z\{z > 0\}$ should be understood as $\mathsf{S} \to \mathsf{B}_1 : x\{x > 0 \wedge z' = x\}$ for some fresh x. Furthermore, if the communicated value is a constant c and there is no need to store this value, we inline it and write $\mathsf{S} \to \mathsf{B}_2 : \mathrm{succ}\{\top\}$ instead of $\mathsf{S} \to \mathsf{B}_2 : x\{x = \mathrm{succ}\}$. We may omit the condition $\top$, turning $\mathsf{S} \to \mathsf{B}_2 : \mathrm{succ}\{\top\}$ into $\mathsf{S} \to \mathsf{B}_2 : \mathrm{succ}$.

Symbolic protocols are specification-wise similar to symbolic register automata [18], but allow more general patterns of register manipulation and do not a priori require formulas to come from an effective Boolean algebra. Symbolic protocols can be seen as a finite description of an infinite-state LTS, whose concrete states consist of a control state along with an assignment for the register variables R. Transitions are concrete communication events that optionally modify register values. We formally define the concrete LTS of a symbolic protocol as follows.

Definition 3.5 (Concretization of symbolic protocols). For a symbolic protocol $\mathbb{S} = (S, R, \Delta, s_0, \rho_0, F)$, let $\mathcal{S}_\mathbb{S}$ denote its concrete protocol. The set of states of $\mathcal{S}_\mathbb{S}$ is $S \times (R \to \mathcal{V})$.

Transitions in $\mathcal{S}_\mathbb{S}$ are defined as follows:

$$\frac{s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:x\{\varphi\}} s_2 \in \Delta \qquad \rho_1\rho_2'[x \mapsto v] \models \varphi}{(s_1, \rho_1) \xrightarrow{\mathsf{p}\to\mathsf{q}:v} (s_2, \rho_2)}$$

Intuitively, the rule says that a symbolic transition from s_1 to s_2 can be instantiated to one from (s_1, ρ_1) to (s_2, ρ_2) on value v when v together with the register assignments in the pre- and post-state satisfy the transition constraint φ. Here, we use juxtaposition $\rho_1\rho_2'$ of register assignments to express their disjoint union. The assignment ρ_2' is obtained from ρ_2 by replacing registers r in the domain with their primed version in R'. The initial state is defined as (s_0, ρ_0). A state (s, ρ) in $\mathcal{S}_{\mathbb{S}}$ is final when $s \in F$. Thus, $\mathcal{S}_{\mathbb{S}}$ is a protocol over the alphabet Γ, and language and deadlock freedom etc. are defined in the standard way as above.

The language of a symbolic protocol $\mathbb{S}$ is the language of its concretization $\mathcal{S}_{\mathbb{S}}$. Consequently, we say a symbolic protocol is implementable if its concretization is implementable.

4 CHARACTERIZING PROTOCOL IMPLEMENTABILITY

We motivate our precise characterization of protocol implementability through examples of non-implementable protocols, and show that seemingly disparate sources of non-implementability share a unified semantic explanation. Recall the protocol $\mathbb{S}_1$ from §2 with a receiver violation, depicted in Fig. 5. The infinite-state LTS $\mathbb{S}_1$ contains the two concrete run prefixes depicted in Fig. 8, where the values of x, y are 2, 3 and 1, 3 respectively.

Inspecting $\mathbb{S}_1$'s specification reveals that the protocol expects r to receive messages from p and q in a different order depending on the branch that q chooses to follow. However, this expectation is unreasonable in a distributed setting. Between the two concrete runs, r's partial view of the protocol's behavior is the same: r receives a value 3 from p. Yet r is expected to receive in p, q order in one run, and receive in q, p order in the other.

Recall again the protocol $\mathbb{S}_2$ from §2 with a sender violation, depicted in Fig. 6. Again inspecting $\mathbb{S}_2$'s specification, the branching structure imposes the expectation that on the top branch, p should send q an o message, whereas on the bottom branch, p should immediately terminate. The two concrete runs in Fig. 9 and Fig. 10 again demonstrate that this expectation is unreasonable: p receives the value 3 from q in both runs, but in one run is expected to send a message, whereas in the other is expected to terminate.

The non-implementability in the examples above can be attributed to insufficient local information about protocol control flow. This source of non-implementability is inherent to the expressive power of branching choice in protocol specifications, and is present even in finite protocols with more restricted choice constructs. While most existing works soundly detect insufficient local

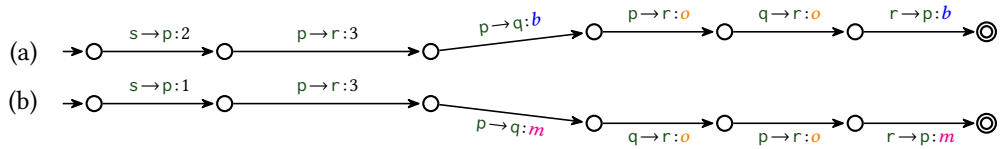

Fig. 8. Two concrete runs of $\mathbb{S}_1$ (Fig. 5): (a) with $x = 2$ and $y = 3$ and (b) with $x = 1$ and $y = 3$.

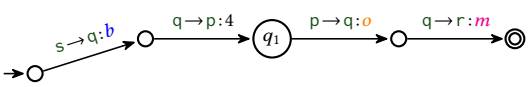

Fig. 9. A concrete run of $\mathbb{S}_2$ (Fig. 6) with s choosing the top branch.

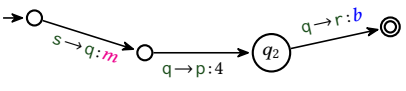

Fig. 10. A concrete run of $\mathbb{S}_2$ (Fig. 6) with s choosing the bottom branch and $x_2 = 4$.

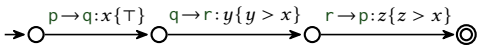

Fig. 11. $\mathbb{S}_3$: A non-implementable protocol with dependent refinements.

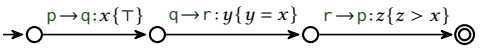

Fig. 12. $\mathbb{S}'_3$: An implementable protocol with dependent refinements.

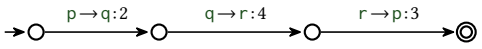

Fig. 13. A run of $\mathbb{S}_3$ with $x = 2, y = 4, z = 3$.

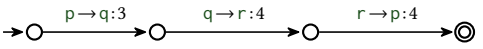

Fig. 14. Another run of $\mathbb{S}_3$ with $x = 3, y = 4, z = 4$.

information through conservative projection algorithms [14, 41, 59, 65], Li et al. [48] give a complete characterization. To check implementability, they first obtain a candidate implementation by restricting the global protocol onto each participant's alphabet, and then determinize the resulting finite state automaton. Then, they check sound and complete conditions directly on the states of the candidate implementation.

Our first observation towards a precise characterization is that implementability can be checked directly on the global protocol specification, without relying on first synthesizing a candidate implementation. This is especially important in the general case, when synthesizing a candidate implementation is itself challenging and not always possible. Our analysis of the protocols above shows that non-implementability can be blamed solely on the existence of certain states in the concrete LTS represented by the global protocol. In fact, we show in §5 that the implementability check for global types in [48] can be made more efficient by forgoing the synthesis step.

Let us now turn our attention to a different source of non-implementability that is unique to the setting of dependent data refinements. Consider the following pair of symbolic protocols $\mathbb{S}_3$ and $\mathbb{S}'_3$, depicted in Fig. 11 and Fig. 12.

Non-implementability is again caused by insufficient local information, but this time with respect to message data rather than control flow: in fact, no branching choice appears in this pair of simple protocols. The problem instead arises in the fact that in both $\mathbb{S}_3$ and $\mathbb{S}'_3$, r does not know the value of x, yet must send z values that satisfy a constraint on x. Partial information of protocol variables is addressed in [72], where the authors syntactically classify whether a variable is known or unknown to a participant, and annotate the variables accordingly in the typing context: a variable is known to its sender and receiver, and unknown to all other participants. However, this syntactic analysis is itself insufficient, as demonstrated by these examples: both protocols yield the same classification of variables per participant, yet one is implementable and the other is not.

We instead turn to concrete runs of $\mathbb{S}_3$ to find the source of non-implementability. Let us consider the concrete runs of $\mathbb{S}_3$ depicted in Figs. 13 and 14, where the values of x, y are 2, 4 and 3, 4 respectively.

In this pair of runs, r observes the same behaviors, namely receiving the value 4 from q. While $\mathbb{S}_3$ also permits r to send 4 to p in the first run, sending 3 to p in the second run constitutes a violation to the refinement predicate $z > x$, i.e. $3 > 3$ is false. Again, this presents a problem because the two run prefixes are indistinguishable to r. However, observe that in this example, non-implementability can again be blamed solely on the existence of states in the global protocol.

We formalize a participant's local information about the protocol using two variations on the standard notion of reachability. Let $\mathcal{S} = (S, \Gamma, T, s_0, F)$ be a protocol and let $\mathsf{p} \in \mathcal{P}$ be a participant. The standard notion defines s' as reachable from s in $\mathcal{S}$ on $w \in \Gamma^*$, denoted $s \xrightarrow{w}{}^* s'$, when there

exists a sequence of transitions $s_1 \xrightarrow{l_1} s_2 \ldots s_{n-1} \xrightarrow{l_{n-1}} s_n$, such that $s_1 = s$, $s_n = s'$, $l_1 \ldots l_{n-1} = w$ and for each $1 \leq i < n$, it holds that $s_i \xrightarrow{l_i} s_{i+1} \in T$. We first define a notion of reachability that restricts the transitions to only the actions observable by a single participant.

Participant-based Reachability. We say that $s \in S$ is *reachable for* p *on* $u \in \Gamma_{\mathsf{p}}^*$ when there exists $w \in \Gamma^*$ such that $s_0 \xrightarrow{w}{}^* s \in T$ and $w{\Downarrow_{\Gamma_{\mathsf{p}}}} = u$, which we denote $s_0 \overset{u}{\underset{\mathsf{p}}{\Longrightarrow}}{}^* s$. We characterize *simultaneously reachable* pairs of states for each participant using the notion of participant-based reachability.

Simultaneous Reachability. We say that $s_1, s_2 \in S$ are simultaneously reachable for participant p on $u \in \Gamma_{\mathsf{p}}^*$, denoted $s_0 \overset{u}{\underset{\mathsf{p}}{\Longrightarrow}}{}^* s_1, s_2$, if there exist $w_1, w_2 \in \Gamma^*$ such that $s_0 \xrightarrow{w_1}{}^* s_1 \in T$, $s_0 \xrightarrow{w_2}{}^* s_2 \in T$ and $w_1{\Downarrow_{\Gamma_{\mathsf{p}}}} = w_2{\Downarrow_{\Gamma_{\mathsf{p}}}} = u$. Simultaneous reachability captures the notion of *locally indistinguishable* states: to a participant, two states are locally distinguishable if they are simultaneously reachable.

Send Coherence requires that any message m that can be sent from a state s_1 can also be sent from all other states s_1' that are locally indistinguishable from s_1 to the sender of m.

Definition 4.1 (Send Coherence). A protocol $\mathcal{S} = (S, \Gamma, T, s_0, F)$ satisfies Send Coherence (SC) when for every $s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:m} s_2 \in T$:

$$\forall s_1' \in S.\ (\exists u \in \Gamma_{\mathsf{p}}^*.\ s_0 \overset{u}{\underset{\mathsf{p}}{\Longrightarrow}}{}^* s_1, s_1') \implies (\exists s_2' \in S.\ s_1' \overset{\mathsf{p}\to\mathsf{q}:m}{\underset{\mathsf{p}}{\Longrightarrow}}{}^* s_2')\ .$$

Receive Coherence, on the other hand, requires that no message which can be received from a state can be received from any other state that is locally indistinguishable to the receiver.

Definition 4.2 (Receive Coherence). A protocol $\mathcal{S} = (S, \Gamma, T, s_0, F)$ satisfies Receive Coherence (RC) when for every $s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:m} s_2, s_1' \xrightarrow{\mathsf{r}\to\mathsf{q}:m} s_2' \in T$:

$$(\mathsf{r} \neq \mathsf{p} \wedge \exists u \in \Gamma_{\mathsf{q}}^*.\ s_0 \overset{u}{\underset{\mathsf{q}}{\Longrightarrow}}{}^* s_1, s_1') \implies \forall w \in \mathrm{pref}(\mathcal{L}(\mathcal{S}_{s_2'})).\ w{\Downarrow_{\Sigma_{\mathsf{q}}}} \neq \varepsilon \vee \mathcal{V}(w{\Downarrow_{\mathsf{p}\triangleright\mathsf{q}!\_}}) \neq \mathcal{V}(w{\Downarrow_{\mathsf{q}\triangleleft\mathsf{p}?\_}}) \cdot m)\ .$$

No Mixed Choice requires that roles cannot equivocate between sending and receiving in two locally indistinguishable states.

Definition 4.3 (No Mixed Choice). A protocol $\mathcal{S} = (S, \Gamma, T, s_0, F)$ satisfies No Mixed Choice (NMC) when for every $s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:m} s_2, s_1' \xrightarrow{\mathsf{r}\to\mathsf{p}:m} s_2' \in T$: $(\exists u \in \Gamma_{\mathsf{q}}^*.\ s_0 \overset{u}{\underset{\mathsf{q}}{\Longrightarrow}}{}^* s_1, s_1') \implies \bot$.

Our semantic characterization of protocol implementability is the conjunction of the above three conditions, and is defined below. In contrast to the syntactic analysis in [72], our semantic approach is sound and complete. In contrast to the sound and complete approach in [48], our implementability conditions do not rely on synthesizing an implementation upfront.

Definition 4.4 (Coherence Conditions). A protocol satisfies Coherence Conditions (CC) when it satisfies Send Coherence, Receive Coherence and No Mixed Choice.

The preciseness of *CC* is stated as follows.

Theorem 4.5. *Let $\mathcal{S}$ be a protocol. Then, $\mathcal{S}$ is implementable if and only if it satisfies CC.*

In the next two sections, we illustrate the key steps for proving Theorem 4.5. We refer the reader to Appendix A for the complete proofs.

4.1 Soundness

Soundness requires us to show that if a protocol satisfies *CC*, then it is implementable. We begin by mirroring the observation made in [1, 48, 63], that is for any global protocol there exists a *canonical* implementation, consisting of one local implementation per participant. We formally define what it means for an implementation to be canonical in our setting below.

Definition 4.6 (Canonical implementations). We say a CLTS $\{\!|T_{\mathsf{p}}|\!\}_{\mathsf{p}\in\mathcal{P}}$ is a *canonical implementation* for a protocol $\mathcal{S} = (S, \Gamma, T, s_0, F)$ 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) $\mathrm{pref}(\mathcal{L}(T_{\mathsf{p}})) = \mathrm{pref}(\mathcal{L}(\mathcal{S})\Downarrow_{\Sigma_{\mathsf{p}}})$.

We first prove that following fact about canonical implementations of protocols satisfying *CC*, which states that the canonical implementations for a protocol satisfying NMC themselves do not exhibit mixed choice.

Lemma 4.7 (No Mixed Choice). *Let $\mathcal{S}$ be a protocol satisfying NMC (Definition 4.3) and let $\{\!|T_{\mathsf{p}}|\!\}_{\mathsf{p}\in\mathcal{P}}$ be a canonical implementation for $\mathcal{S}$. Let $wx_1, wx_2 \in \mathrm{pref}(\mathcal{L}(T_{\mathsf{p}}))$ with $x_1 \neq x_2$ for some $\mathsf{p} \in \mathcal{P}$. Then, $x_1 \in \Sigma_!$ iff $x_2 \in \Sigma_!$.*

We choose the canonical implementation as our existential witness to show that any protocol satisfying *CC* is implementable. By the definition of implementability (Definition 3.3), soundness therefore amounts to showing the following three conditions:
(a) $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\{\!|T_{\mathsf{p}}|\!\}_{\mathsf{p}\in\mathcal{P}})$, (b) $\mathcal{L}(\{\!|T_{\mathsf{p}}|\!\}_{\mathsf{p}\in\mathcal{P}}) \subseteq \mathcal{L}(\mathcal{S})$, and (c) $\{\!|T_{\mathsf{p}}|\!\}_{\mathsf{p}\in\mathcal{P}}$ is deadlock-free.
Condition (a) states that any canonical implementation recognizes at least the global protocol behaviors. This fact can be shown for any LTS and canonical CLTS, and does not rely on assumptions about determinism or sender-drivenness, nor assumptions about the LTS satisfying *CC*.

Lemma 4.8 (Canonical implementation language contains protocol language). *Let $\mathcal{S}$ be an LTS and let $\{\!|T_{\mathsf{p}}|\!\}_{\mathsf{p}\in\mathcal{P}}$ be a canonical implementation for $\mathcal{S}$. Then, $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\{\!|T_{\mathsf{p}}|\!\}_{\mathsf{p}\in\mathcal{P}})$.*

Condition (b) states that any behavior recognized by the canonical implementation is a global protocol behavior, in other words, that the canonical CLTS does not add behaviors. This is only true for protocols that satisfy *CC*.
Furthermore, the acceptance condition for infinite words in $\mathcal{L}(\mathcal{S})$ differs from that in $\{\!|T_{\mathsf{p}}|\!\}_{\mathsf{p}\in\mathcal{P}}$: the latter accepts all infinite traces, whereas the former requires to show that an infinite word w satisfies $w \preceq_{\sim}^{\omega} w'$ for some other infinite word $w' \in \mathcal{L}(\mathcal{S})$. Therefore, showing prefix set inclusion is not sufficient, and we must reason about the finite and infinite case separately.

Lemma 4.9 (Global protocol language contains canonical implementation language). *Let $\mathcal{S}$ be a protocol satisfying* CC *and let $\{\!|T_{\mathsf{p}}|\!\}_{\mathsf{p}\in\mathcal{P}}$ be a canonical implementation for $\mathcal{S}$ such that for all $w \in \Sigma^*$, if w is a trace of $\{\!|T_{\mathsf{p}}|\!\}_{\mathsf{p}\in\mathcal{P}}$, then $I(w) \neq \emptyset$. Then, $\mathcal{L}(\{\!|T_{\mathsf{p}}|\!\}_{\mathsf{p}\in\mathcal{P}}) \subseteq \mathcal{L}(\mathcal{S})$.*

Towards these ends, we adapt the key intermediate lemma from [48] to our setting, and show the inductive invariant that every trace in the canonical implementation of a protocol satisfying *CC* satisfies *intersection set non-emptiness*. Note that although our intermediate lemma statements are similar to those in [48] in structure, [48] reasons about a particular implementation that is assumed to satisfy Send Validity and Receive Validity, namely the subset construction obtained from the global type, whereas our proofs reason about any canonical implementation of a global protocol that satisfies *CC*. As a result, the proof arguments differ significantly.
We adapt the relevant definitions to our setting below.

Definition 4.10 (LTS intersection sets). Let $\mathcal{S}$ be an LTS. Let p be a participant and $w \in \Sigma^*$ be a word. We define the set of possible runs $\mathrm{R}_{\mathsf{p}}^{\mathcal{S}}(w)$ as all maximal runs of $\mathcal{S}$ that are consistent with

p's local view of w:

$$\mathrm{R}_{\mathsf{p}}^{\mathcal{S}}(w) \coloneqq \{\rho \text{ is a maximal run of } \mathcal{S} \mid w{\Downarrow_{\Sigma_{\mathsf{p}}}} \leq \mathtt{split}(\mathtt{trace}(\rho)){\Downarrow_{\Sigma_{\mathsf{p}}}}\} \ .$$

We denote the intersection of the possible run sets for all participants as $I^{\mathcal{S}}(w) \coloneqq \bigcap_{\mathsf{p}\in\mathcal{P}} \mathrm{R}_{\mathsf{p}}^{\mathcal{S}}(w)$.

Definition 4.11 (Unique splitting of a possible run). Let $\mathcal{S}$ be an LTS, p a participant, and $w \in \Sigma^*$ a word. Let ρ be a run in $\mathrm{R}_{\mathsf{p}}^{\mathcal{S}}(w)$. We define the longest prefix of ρ matching w:

$$\alpha' \coloneqq \max\{\rho' \mid \rho' \leq \rho \ \wedge \ \mathtt{split}(\mathtt{trace}(\rho')){\Downarrow_{\Sigma_{\mathsf{p}}}} \leq w{\Downarrow_{\Sigma_{\mathsf{p}}}}\} \ .$$

If $\alpha' \neq \rho$, we can split ρ into $\rho = \alpha \cdot s \xrightarrow{l} s' \cdot \beta$ where $\alpha' = \alpha \cdot s.$, which we call the unique splitting of ρ for p matching w. Uniqueness follows from the maximality of α'.

For example, the unique splitting of $\rho = s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:m} s_2 \xrightarrow{\mathsf{r}\to\mathsf{q}:b_1} s_3 \xrightarrow{\mathsf{r}\to\mathsf{q}:b_2} s_4 \xrightarrow{\mathsf{q}\to\mathsf{p}:o} s_5$ for p matching $w = \mathsf{r} \triangleright \mathsf{q}!b_1.\, \mathsf{p} \triangleright \mathsf{q}!m$ is $\alpha \cdot s_3 \xrightarrow{\mathsf{r}\to\mathsf{q}:b_2} s_4 \cdot \beta$, where $\alpha = s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:m} s_2 \xrightarrow{\mathsf{r}\to\mathsf{q}:b_1} s_3$ and $\beta = s_4 \xrightarrow{\mathsf{q}\to\mathsf{p}:o} s_5$.

Our intersection non-emptiness inductive invariant is stated below. The proof proceeds by induction on the length of a prefix w of the canonical implementation, and case splits based on whether w is extended by a send or receive action. Lemma 4.14 and Lemma 4.13 provide a characterization for each of these cases respectively.

Lemma 4.12 (Intersection set non-emptiness). *Let $\mathcal{S}$ be a protocol satisfying* CC*, and let $\{\!\{T_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ be a canonical CLTS for $\mathcal{S}$. Then, for every trace $w \in \Sigma^*$ of $\{\!\{T_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}}$, it holds that $I(w) \neq \emptyset$.*

Lemma 4.13 (Receive events do not shrink intersection sets). *Let $\mathcal{S}$ be a protocol satisfying* CC*, and let $\{\!\{T_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ be a canonical communicating LTS for $\mathcal{S}$. Let wx be a trace of $\{\!\{T_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ such that $x \in \Sigma_?$. Then, $I(w) = I(wx)$.*

Lemma 4.14 (Send events preserve run prefixes). *Let $\mathcal{S}$ be a protocol satisfying* CC *and $\{\!\{T_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ be a canonical communicating LTS for $\mathcal{S}$. Let wx be a trace of $\{\!\{T_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ such that $x \in \Sigma_{\mathsf{p},!}$ for some $\mathsf{p} \in \mathcal{P}$. Let ρ be a run in $I(w)$, and $\alpha \cdot s_{pre} \xrightarrow{l} s_{post} \cdot \beta$ be the unique splitting of ρ for p with respect to w. Then, there exists a run ρ' in $I(wx)$ such that $\alpha \cdot s_{pre} \leq \rho'$.*

Finally, we show that for protocols that satisfy *CC* and intersection set non-emptiness, the canonical implementation is deadlock-free. The proof follows immediately from the following lemma and the fact that CLTS are deterministic, and is thus omitted.

Lemma 4.15 (Channel compliance and intersection set non-emptiness implies prefix). *Let $\mathcal{S} = (S, \Gamma, T, s_0, F)$ be a protocol and let $w \in \Sigma^*$ be a word such that (i) w is channel-compliant, and (ii) $I(w) \neq \emptyset$. Then, $w \in \mathrm{pref}(\mathcal{L}(\mathcal{S}))$.*

Lemma 4.16 (Canonical implementation deadlock freedom). *Let $\mathcal{S} = (S, \Gamma, T, s_0, F)$ be a protocol satisfying* CC *and let $\{\!\{T_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ be a canonical implementation for $\mathcal{S}$ such that for all $w \in \Sigma^*$, if w is a trace of $\{\!\{T_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}}$, then $I(w) \neq \emptyset$. Then, $\{\!\{T_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ is deadlock-free.*

Soundness thus follows from the three conditions above.

Lemma 4.17 (Soundness of *CC*). *Let $\mathcal{S}$ be a protocol. If $\mathcal{S}$ satisfies* CC*, then $\mathcal{S}$ is implementable.*

4.2 Completeness

Completeness requires us to show that if a protocol is implementable, then it satisfies *CC*. We prove completeness by modus tollens, and assume that a protocol $\mathcal{S}$ does not satisfy *CC*. We negate SC, RC and NMC in turn: from the negation of SC we obtain a simultaneously reachable pair of states in $\mathcal{S}$ such that a send event that is enabled in one is never enabled from the other. From the negation of RC there exists a simultaneously reachable pair of states in $\mathcal{S}$ such that a receive event that is enabled in one is also enabled in the other. From the negation of NMC we obtain a simultaneously reachable pair of transitions where a participant is the sender in one and the receiver in the other. We assume an arbitrary CLTS that implements $\mathcal{S}$, and use these witnesses to show that this CLTS must recognize a trace that is not a prefix in $\mathcal{L}(\mathcal{S})$, thereby either violating protocol fidelity or deadlock freedom.

Lemma 4.18 (Completeness). *Let $\mathcal{S}$ be a protocol. If $\mathcal{S}$ is implementable, then $\mathcal{S}$ satisfies* CC.

4.3 Synthesis

When proving soundness, we chose the canonical implementation as our witness to implementability. In other words, if a protocol satisfies *CC*, then the canonical implementation implements it. When proving completeness, we showed that *any* implementation would cause a violation to protocol fidelity or deadlock-freedom. In other words, if a protocol violates *CC*, then no implementation exists. Having established that *CC* precisely characterizes implementable protocols, we combine these observations to yield the following corollary:

Corollary 4.19 (Canonical implementation is all you need). *A protocol is implementable if and only if the canonical implementation implements it.*

For an implementable protocol, this fact serves as a criterion for synthesizing implementations: any implementation that is canonical will suffice. For the general class of protocols, synthesis is undecidable. However, for many expressive fragments of protocols that still feature infinite data, e.g. corresponding to symbolic finite automata [19, 62] and certain classes of timed and register automata [6, 13], one can simply use off-the-shelf determinization algorithms to compute canonical implementations [5, 68, 69].

5 CHECKING IMPLEMENTABILITY

Having established that *CC* is precise for protocol implementability, we next present sound and relatively complete algorithms to check *CC* for several protocol classes. We start with the most general case of symbolic protocols before considering decidable classes of finite-state protocols.

5.1 Symbolic Protocols

In the remainder of the section, we fix a symbolic protocol $\mathbb{S} = (S, R, \Delta, s_0, \rho_0, F)$. We assume that the concretization of $\mathbb{S}$ is a GCLTS. Additionally, we define two copies of the symbolic protocol, denoted $\mathbb{S}_1$ and $\mathbb{S}_2$ that we will use in describing our symbolic implementability check. Each copy $\mathbb{S}_i = (R_i, S, \Delta_i, \rho_i, s_0, F)$ with $i \in \{1, 2\}$ is obtained from $\mathbb{S}$ by renaming each register r to a fresh register r_i, each unique communication variable x to x_i, and obtaining new transition constraints φ_i from φ by substituting the register and communication variables accordingly. The domain of the initial register assignment is also modified according to the new register names; the control states remain the same.

Because symbolic protocols describe concrete protocols with infinitely many states and transitions, implementability cannot be checked explicitly using our *CC* characterization for protocols,

i.e. by iterating over all states and transitions. Instead, we present symbolic conditions that are valid on the symbolic protocol if and only if its concrete protocol is implementable.

Specifically, we provide a sound and complete reduction to the first-order fixpoint logic μCLP [67]. The μCLP calculus can express recursive predicates with least and greatest fixpoint semantics where the predicate body is constrained by a first-order logic formula over a background theory. The expressive power of μCLP is needed because Send Coherence reasons about both reachability and coreachability, which naturally translate to least and greatest fixpoints over the protocol's transition relation.

Theorem 5.1 (Symbolic Implementability). *$\mathbb{S}$ is implementable if and only if it satisfies Symbolic Send Coherence, Symbolic Receive Coherence, and Symbolic No Mixed Choice.*

We now present these symbolic conditions, starting with Symbolic Send Coherence.

Send Coherence first requires us to characterize pairs of states in a protocol that are simultaneously reachable for each participant on some prefix in its local language. In the symbolic setting, this amounts to the following: given a participant and a pair of control states (s_1, s_2) in the symbolic protocol, characterize the register assignments for pairs of concrete states (s_1, ρ_1), (s_2, ρ_2) that are in the respective control states. The predicate $\mathsf{prodreach}_{\mathsf{p}}(s_1, \boldsymbol{r_1}, s_2, \boldsymbol{r_2})$ describes this for each $\mathsf{p} \in \mathcal{P}$ where $\boldsymbol{r_i}$ are vectors of the registers in R_i obtained by ordering them according to some fixed total order. We define this predicate as a least fixpoint as follows.

Definition 5.2 (Simultaneous reachability in product symbolic protocol). Let $\mathsf{p} \in \mathcal{P}$ be a participant and let $s_1, s_1', s_2, s_2' \in S$. Then,

$$\begin{aligned}
\mathsf{prodreach}_{\mathsf{p}}(s_1', \boldsymbol{r_1'}, s_2', \boldsymbol{r_2'}) \coloneqq_{\mu} \; & (\, s_1' = s_0 \wedge s_2' = s_0 \wedge \boldsymbol{r_1'} = \rho_0 \wedge \boldsymbol{r_2'} = \rho_0 \,) \\
\vee \; & \Big(\bigvee_{\substack{(s_1,\, \mathsf{r}\to\mathsf{s}:x_1\{\varphi_1\},\, s_1') \in \Delta_1 \\ (s_2,\, \mathsf{r}\to\mathsf{s}:x_2\{\varphi_2\},\, s_2') \in \Delta_2 \\ \mathsf{p}=\mathsf{r} \vee \mathsf{p}=\mathsf{s}}} \exists x_1 x_2.\, \mathsf{prodreach}_{\mathsf{p}}(s_1, \boldsymbol{r_1}, s_2, \boldsymbol{r_2}) \wedge \varphi_1 \wedge \varphi_2 \wedge x_1 = x_2 \,\Big) \\
\vee \; & \Big(\bigvee_{(s_1,\, \mathsf{r}\to\mathsf{s}:x_1\{\varphi_1\},\, s_1') \in \Delta_1 \,\wedge\, \mathsf{p}\neq\mathsf{r}\wedge\mathsf{p}\neq\mathsf{s}} \exists x_1.\, \mathsf{prodreach}_{\mathsf{p}}(s_1, \boldsymbol{r_1}, s_2', \boldsymbol{r_2'}) \wedge \varphi_1 \,\Big) \\
\vee \; & \Big(\bigvee_{(s_2,\, \mathsf{r}\to\mathsf{s}:x_2\{\varphi_2\},\, s_2') \in \Delta_2 \,\wedge\, \mathsf{p}\neq\mathsf{r}\wedge\mathsf{p}\neq\mathsf{s}} \exists x_2.\, \mathsf{prodreach}_{\mathsf{p}}(s_1', \boldsymbol{r_1'}, s_2, \boldsymbol{r_2}) \wedge \varphi_2 \,\Big) \;.
\end{aligned}$$

The second top-level disjunct in the definition (after the base case) handles the cases where $\mathbb{S}_1$ and $\mathbb{S}_2$ synchronize on a common action involving p. The remaining two disjuncts correspond to the cases where either $\mathbb{S}_1$ or $\mathbb{S}_2$ follows an ε transition.

Given a pair of simultaneously reachable states (s_1, ρ_1), (s_2, ρ_2) in p, Send Coherence now checks whether all values x_1 that can be sent to some q in (s_1, ρ_1) can also be sent from (s_2, ρ_2), modulo following ε transitions to reach the actual state where p can send to q. We thus need to express ε-reachability. We formalize the dual: the predicate $\mathsf{unreach}^{\varepsilon}_{\mathsf{p},\mathsf{q}}(s_2, \boldsymbol{r_2}, x_1)$ expresses that p *cannot* reach any state where it may send x_1 to q, by following ε transitions from symbolic state $(s_2, \boldsymbol{r_2})$. This is formulated as a greatest fixpoint as follows:

Definition 5.3 (ε-unreachability of p sending x to q). For $\mathsf{p}, \mathsf{q} \in \mathcal{P}$ and $s \in S$, let

$$\mathsf{unreach}^{\varepsilon}_{\mathsf{p},\mathsf{q}}(s, \boldsymbol{r}, x) \coloneqq_{\nu} \quad \Big(\bigwedge_{(s,\, \mathsf{p}\to\mathsf{q}:y\{\varphi\},\, s') \in \Delta} \neg\varphi[x/y] \,\Big) \wedge \Big(\bigwedge_{\substack{(s,\, \mathsf{q}\to\mathsf{t}:y\{\varphi\},\, s') \in \Delta \\ \mathsf{p}\neq\mathsf{q}\wedge\mathsf{p}\neq\mathsf{t}}} \forall y.\, \varphi \Rightarrow \mathsf{unreach}^{\varepsilon}_{\mathsf{p},\mathsf{q}}(s', \boldsymbol{r'}, x) \,\Big) \;.$$

The first conjunct checks that whenever p reaches a state with an outgoing send transition to q, it cannot send the value x because the transition constraint φ is not satisfied. The second conjunct

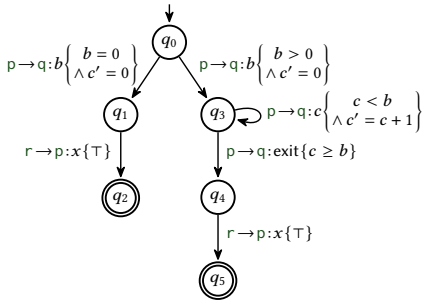

Fig. 15. Example where states q_1 and q_3 satisfy Send Coherence for r.

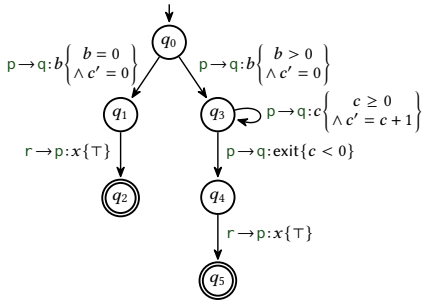

Fig. 16. Example where states q_1 and q_3 violate Send Coherence for r.

checks that every outgoing ε transition is either disabled ($\neg\varphi$ holds) or following the transition does not reach an appropriate send state.

We combine the auxiliary predicates into our Symbolic Send Coherence condition.

Definition 5.4 (Symbolic Send Coherence). A symbolic protocol $\mathbb{S}$ satisfies Symbolic Send Coherence when for each transition $s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:x_1\{\varphi_1\}} s_1' \in \Delta_1$ and state $s_2 \in S$, the following is valid:

$$\mathsf{prodreach}_{\mathsf{p}}(s_1, \boldsymbol{r_1}, s_2, \boldsymbol{r_2}) \ \wedge\ \varphi_1 \wedge \mathsf{unreach}^{\varepsilon}_{\mathsf{p},\mathsf{q}}(s_2, \boldsymbol{r_2}, x_1) \implies \bot \ .$$

A keen reader may have noticed that because the symbolic characterization of Send Coherence involves a greatest fixpoint, it is a liveness property. Thus, proving Send Coherence, in general, involves a termination argument. To see this, consider the two protocols shown in Figs. 15 and 16. Consider the pair of states $(q_1, [c \mapsto 0])$ and $(q_3, [c \mapsto 0])$ which are simultaneously reachable for r in both protocols. The send transition for r enabled in q_1 needs to be matched with a corresponding send transition in an ε-reachable state from q_3. The only candidate states for this match in both protocols are those at control state q_4. However, these states are reachable from q_3 iff the loop in q_3 terminates, which it does for the protocol in Fig. 15 but not for the one in Fig. 16.

Receive Coherence is conditioned on two simultaneously reachable states $(s_1, \boldsymbol{r_1})$ and $(s_2, \boldsymbol{r_2})$ for a participant q. It checks that if q can receive x from p in the first state, q should not also be able to receive x as the first message from p after first receiving from a different participant r in the second state. That is, unless p sending x causally depends on q first receiving from r. We thus need to define a predicate that captures whether x_1 may be available as the first message from q to p, while tracking causal dependencies. We introduce a family of predicates $\mathsf{avail}_{\mathsf{p},\mathsf{q},\mathcal{B}}(x_1, s_2, \boldsymbol{r_2})$ for this purpose. Here, $\mathcal{B}$ is used to track the causal dependencies. $\mathcal{B}$ tracks the set of participants that are blocked from sending a message (because the send action causally depends on q first receiving from r). The predicates are defined as the least fixpoint of the following mutually recursive definition.

Definition 5.5 (Symbolic Availability).

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

Algorithm 1 Check *CC* for finite protocols

▷ *Let LTS* $\mathcal{S} = (S, \Gamma, T, s_0, F)$
▷ *Checking Send Coherence*
for $s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:m} s_2 \in T$ **do**
  for $s \neq s_1 \in S$ **do**
    if $\mathcal{L}(S, \Gamma_\mathsf{p} \uplus \{\varepsilon\}, T_\mathsf{p}, s_0, \{s\}) \cap \mathcal{L}(S, \Gamma_\mathsf{p} \uplus \{\varepsilon\}, T_\mathsf{p}, s_0, \{s_1\}) \neq \emptyset$ **then**
      $b \leftarrow \bot$
      for $s_3 \xrightarrow{\mathsf{p}\to\mathsf{q}:m} s_4 \in T$ **do** $b \leftarrow b \vee \left(s \underset{\mathsf{p}}{\overset{\varepsilon}{\Rightarrow}}{}^{*}\, s_3\right)$
      if $\neg b$ **then return** $\bot$
▷ *Checking Receive Coherence*
for $s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:m} s_2, s_3 \xrightarrow{\mathsf{r}\to\mathsf{q}:m} s_4 \in T, s_1 \neq s_2, \mathsf{p} \neq \mathsf{r}$ **do**
  if $\mathcal{L}(S, \Gamma_\mathsf{q} \uplus \{\varepsilon\}, T_\mathsf{q}, s_0, \{s_1\}) \cap \mathcal{L}(S, \Gamma_\mathsf{q} \uplus \{\varepsilon\}, T_\mathsf{q}, s_0, \{s_3\}) \neq \emptyset$ **then**
    if $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, s_4)$ **then return** $\bot$
▷ *Checking No Mixed Choice*
for $s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:m} s_2, s_3 \xrightarrow{\mathsf{r}\to\mathsf{p}:m} s_4 \in T, s_1 \neq s_2$ **do**
  if $\mathcal{L}(S, \Gamma_\mathsf{q} \uplus \{\varepsilon\}, T_\mathsf{q}, s_0, \{s_1\}) \cap \mathcal{L}(S, \Gamma_\mathsf{q} \uplus \{\varepsilon\}, T_\mathsf{q}, s_0, \{s_3\}) \neq \emptyset$ **then return** $\bot$
return $\top$

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 other two disjuncts handle the cases when x_1 becomes available after some other message exchange between r and t. Here, if r is blocked, then t also becomes blocked since it depends on r sending before it can receive (the first disjunct). Otherwise, no participant is added to the blocked set (the second disjunct).

With the available message predicate in place, we can now define Symbolic Receive Coherence.

Definition 5.6 (Symbolic Receive Coherence). A symbolic protocol $\mathbb{S}$ satisfies Symbolic Receive Coherence when for every pair of transitions $s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:x_1\{\varphi_1\}} s_1' \in \Delta_1$ and $s_2 \xrightarrow{\mathsf{r}\to\mathsf{q}:x_2\{\varphi_2\}} s_2' \in \Delta_2$ with $\mathsf{p} \neq \mathsf{r}$:

$$\mathsf{prodreach}_\mathsf{q}(s_1, \boldsymbol{r_1}, s_2, \boldsymbol{r_2}) \;\wedge\; \varphi_1 \;\wedge\; \varphi_2 \;\wedge\; \mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(x_1, s_2', \boldsymbol{r_2'}) \implies \bot \;.$$

Finally, No Mixed Choice is conditioned on two simultaneously reachable states $(s_1, \boldsymbol{r_1})$ and $(s_2, \boldsymbol{r_2})$ with outgoing send and receive transitions for a participant p.

Definition 5.7 (Symbolic No Mixed Choice). A symbolic protocol $\mathbb{S}$ satisfies Symbolic No Mixed Choice when for every pair of transitions $s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:x_1\{\varphi_1\}} s_1' \in \Delta_1$ and $s_2 \xrightarrow{\mathsf{r}\to\mathsf{p}:x_2\{\varphi_2\}} s_2' \in \Delta_2$:

$$\mathsf{prodreach}_\mathsf{q}(s_1, \boldsymbol{r_1}, s_2, \boldsymbol{r_2}) \;\wedge\; \varphi_1 \;\wedge\; \varphi_2 \implies \bot \;.$$

We next apply our framework to decidable fragments of symbolic protocols, some of which have previously been studied in the literature.

5.2 Finite Protocols

We start with finite protocols. Let $\mathcal{S} = (S, \Gamma, T, s_0, F)$ be a protocol with finite S and T. Because $\mathcal{S}$ is finite, we can transform *CC* into an imperative algorithm (see Algorithm 1) and use it to check implementability directly. For checking Receive Coherence, we need to decide the predicate $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, s)$ which is defined like the symbolic availability predicate $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(x, s, \boldsymbol{r})$, except on protocols instead of symbolic protocols.

It is easy to see that the algorithm runs in time polynomial in the size of $\mathcal{S}$, provided $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, s)$ can be decided in polynomial time. However, we have the following.

Lemma 5.8. *Given* $\mathsf{p}, \mathsf{q} \in \mathcal{P}$, $m \in \mathcal{V}$, *and* $s \in S$, $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, s)$ *is NP-complete.*

Proof. NP membership is immediate: we guess a simple path in $\mathcal{S}$ from s to some state s' with an outgoing transition labeled with $\mathsf{p} \to \mathsf{q} : m$. We then evaluate $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, s)$ along that path, which can be done in polynomial time. We can restrict ourselves to simple paths because the blocked set $\mathcal{B}$ monotonically increases when traversing a path in $\mathcal{S}$. Moreover, $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, s)$ is antitone in the blocked set.

For proving NP-hardness we do a reduction from the 3-SAT problem. Assume a 3-SAT instance $\varphi = C_1 \wedge \ldots \wedge C_k$. Let $x_1, \ldots, x_n$ be the variables occurring in φ and let L_{ij} be the jth literal of clause C_i. We construct a protocol $\mathcal{S}_\varphi$ over $\mathcal{V} = \{m, m_1, m_2, m_3\}$ such that φ is satisfiable iff $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, s_1)$ holds for a state s_1 in $\mathcal{S}_\varphi$:

(1) Define the set of participants as $\mathcal{P} = \{\mathsf{p}, \mathsf{q}, \mathsf{x}_1, \overline{\mathsf{x}}_1, \ldots, \mathsf{x}_\mathsf{n}, \overline{\mathsf{x}}_\mathsf{n}\}$.
(2) Define a protocol $\mathcal{S}_R$ with states $s_1, \ldots, s_{n+1}$ as follows: for every $1 \leq i \leq n$ there are two transitions between s_i and s_{i+1} labeled with $\mathsf{q} \to \mathsf{x}_\mathsf{i} : m$ and $\mathsf{q} \to \overline{\mathsf{x}_\mathsf{i}} : m$, respectively.
(3) Define a protocol $\mathcal{S}_C$ representing the clauses C_i with states $t_1, t_{1,1}, t_{1,2}, t_{1,3}, \ldots, t_{k+1}$ as follows. For each $1 \leq i \leq k$ and $1 \leq j \leq 3$: we have (i) a transition between state t_i and $t_{i,j}$ labeled $\mathsf{p} \to r : m_j$ where $r = \mathsf{x}$ if $L_{ij} = x$ and $r = \overline{\mathsf{x}}$ if $L_{ij} = \neg x$ for $x \in \{x_1, \ldots, x_n\}$, and (2) a transition between state $t_{i,j}$ and t_{i+1} labeled $r \to \mathsf{p} : m$.
(4) Define a protocol $\mathcal{S}_F$ with two states q_1 and q_2 and a single transition from q_1 to q_2 labeled with $\mathsf{p} \to \mathsf{q} : m$.
(5) Obtain $\mathcal{S}_\varphi$ by merging all the protocols $\mathcal{S}_R$, $\mathcal{S}_C$, and $\mathcal{S}_F$ into a single protocol by identifying the state s_{n+1} with t_1 and t_{k+1} with q_1. The initial state is s_1 and q_2 is the only final state.

Observe that the size of $\mathcal{S}$ is linear in the size of φ. Moreover, it is easy to check that $\mathcal{S}_\varphi$ is indeed a GCLTS: all choices are sender-driven and deterministic. Moreover, q_2 is sink-final and the only state with no outgoing transitions. Thus, $\mathcal{S}$ is deadlock-free.

To see that $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, s_1)$ holds in $\mathcal{S}_\varphi$ iff φ is satisfiable, observe that the blocked set $\mathcal{B}$ that $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, s_1)$ computes along a path between s_1 and s_{n+1} contains for each variable x_i either x_i or $\overline{\mathsf{x}_\mathsf{i}}$. The blocked set represents the complement of a truth assignment $\rho_\mathcal{B}$ for the x_i where $\rho_\mathcal{B}(x_i) = \top$ iff $\mathsf{x}_\mathsf{i} \notin \mathcal{B}$. By construction of $\mathcal{S}_R$, for every truth assignment ρ, there exists a path between s_1 and s_{n+1} such that $\rho = \rho_\mathcal{B}$ for the blocked set $\mathcal{B}$ computed along that path.

The paths between states t_i and t_{i+1} in subprotocol $\mathcal{S}_C$ allow p to proceed and not be blocked if one of $t_{i,1}$, $t_{i,2}$, and $t_{i,3}$ has a participant not in $\mathcal{B}$, i.e., C_i is satisfied by $\rho_\mathcal{B}$. Thus, a path from $s_{n+1} = t_1$ to $t_{k+1} = q_1$ adds p to $\mathcal{B}$ at t_1 iff $\rho_\mathcal{B}$ does not satisfy at least one of the clauses C_i. Therefore, m is available in q_2 for that $\mathcal{B}$ iff $\rho_\mathcal{B}$ satisfies φ. □

It follows that checking RC is co-NP-complete for finite protocols and, thus, so is checking implementability.

Theorem 5.9. *Implementability of finite protocols is co-NP-complete.*

Inspection of the implementability algorithm also yields the following fact about the special case of binary protocols, when $|\mathcal{P}| = 2$:

Lemma 5.10. *Every binary protocol is implementable.*

This follows immediately from the observation that RC holds vacuously: with two participants there exists no different receiver that can satisfy the condition. Furthermore, in the binary case, participant-based reachability is equivalent to standard reachability, because each participant is involved in every synchronous action. Because global types are deterministic, there exist no two distinct states in a binary global type that are simultaneously reachable for either participant.

Global multiparty session types. As a corollary of the above discussion, we improve upon on the PSPACE decision procedure given in [48] for checking implementability of global multiparty session types. Global types for MSTs [48] are defined by the grammar:

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

where p, q_i range over $\mathcal{P}$, m_i over a finite set $\mathcal{V}$, and t over a set of recursion variables. The semantics of a global type $\mathbf{G}$ are defined using a finite state machine $\mathrm{GAut}(\mathbf{G}) = (Q_{\mathrm{G}}, \Gamma \uplus \{\varepsilon\}, \delta_{\mathrm{G}}, q_{0,\mathrm{G}}, F_{\mathrm{G}})$ where Q_{G} is the set of all syntactic subterms in $\mathbf{G}$ together with the term 0, δ_{G} is the smallest set containing $(\sum_{i \in I} \mathsf{p} \to \mathsf{q}_i : m_i.G_i, \mathsf{p} \to \mathsf{q}_i : m_i, G_i)$ for each $i \in I$, as well as $(\mu t.G', \varepsilon, G')$ and $(t, \varepsilon, \mu t.G')$ for each subterm $\mu t.G'$, $q_{0,\mathrm{G}} = \mathbf{G}$ and $F_{\mathrm{G}} = \{0\}$.

Each branch of a choice is assumed to be distinct: $\forall i, j \in I.\, i \neq j \Rightarrow (\mathsf{q}_i, m_i) \neq (\mathsf{q}_j, m_j)$, and the sender and receiver of an atomic action is assumed to be distinct: $\forall i \in I.\, \mathsf{p} \neq \mathsf{q}_i$. Recursion is guarded: in $\mu t.\, G$, there is at least one message between μt and each t in G.

Each ε-transition in $\mathrm{GAut}(\mathbf{G})$ is the only transition from the state it originates from. This makes removing them easy, yielding an protocol $\mathcal{S}_{\mathrm{G}} = (Q_{\mathrm{G}}, \Gamma, \delta'_{\mathrm{G}}, q_{0,\mathrm{G}}, F_{\mathrm{G}})$, where δ'_{G} contains only transitions labeled with $l \in \Gamma$. It is easy to verify that $\mathcal{S}_{\mathrm{G}}$ is indeed a GCLTS.

In fact, global multiparty session types yield a special class of tree-shaped GCLTSs where recursion only occurs from leaves to ancestors [63]. Because of this restricted tree-like structure, $\mathrm{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, s)$ can be decided in polynomial time for $\mathcal{S}_{\mathrm{G}}$ using the algorithm given in [48]. Intuitively, the only state in $\mathcal{S}_{\mathrm{G}}$ that can have more than one simple path from any other state s is the final state 0. Hence, the number of simple paths in $\mathcal{S}_{\mathrm{G}}$ is linear in the size of $\mathbf{G}$.

Corollary 5.11. *Implementability of global types is in PTIME.*

5.3 Symbolic Finite Protocols

Finally, we study symbolic representations of finite protocols. More precisely, we consider the fragment of symbolic protocols where $\mathcal{V}$ is the set of Booleans and all transition constraints φ are given by propositional formulas. We show that for this class of symbolic protocols, the implementability problem is PSPACE-complete.

Theorem 5.12. *Implementability of symbolic finite protocols is PSPACE-complete.*

Proof sketch. To show that implementability is in PSPACE, we show that a witness to the negation of *CC* can be checked in nondeterministic polynomial space. This follows by a reduction to the reachability problem for extended finite state machines, which is in PSPACE [32]. By Savitch's Theorem, it follows that the negation of *CC* is in PSPACE. Because PSPACE is closed under complement and *CC* precisely characterizes implementability, it follows that implementability is in PSPACE.

We show PSPACE-hardness of the implementability problem by a reduction from the PSPACE-hard problem of deciding reachability for 1-safe Petri nets [23]. Let (N, M_0) be a 1-safe Petri net, with $N = (S, T, F)$. Let M be a marking of N.

We construct a symbolic protocol that is implementable iff N does not reach M. For ease of exposition, we present this symbolic protocol as a symbolic dependent global type $\mathbf{G}_N$ with the understanding that the encoding of $\mathbf{G}_N$ as a symbolic protocol is clear.

We first describe the construction of $\mathbf{G}_N$. The outermost structure of $\mathbf{G}_N$ consists of a participant r communicating a choice between two branches to s where the bottom branch solely consists of p sending l to q: $\mathbf{G}_N := (\mathsf{r} \to \mathsf{s} : m_1\{\top\}.\, G_t + \mathsf{r} \to \mathsf{s} : m_2\{\top\}.\, \mathsf{p} \to \mathsf{q} : l\{\top\}.\, 0)$. Since p is not informed about the choice of the branch taken by s, it will have to be able to match this send transition in every run that follows the continuation G_t of the top branch. We will construct G_t such that this match is possible iff M is reachable in N.

In G_t, participants r and s enter a loop that simulates N:

$$G_t \coloneqq \mu s[v \coloneqq M_0].\ + \begin{cases} \sum_{t \in T} \mathtt{r} \to \mathtt{s} : m_t\{v \Rightarrow t^-\}.\, s[v \coloneqq ((v \wedge \neg t^-) \vee t^+)] \\ \mathtt{r} \to \mathtt{s} : \mathit{restart}\{\top\}.\, s[v \coloneqq M_0] \\ \mathtt{r} \to \mathtt{s} : \mathit{reach}_M\{v = M\}.\, \mathtt{p} \to \mathtt{q} : l\{\top\}.\, 0 \end{cases}$$

The loop variable v is a $|S|$-length bitvector that tracks the current marking of the net. It is initialized to M_0. Inside the loop, r has the following choices. First, it may pick any transition $t \in T$ of the net and send an m_t message to s, provided the transition is enabled for firing (i.e., the input places of t all contain a token: $v \Rightarrow t^-$). After this communication, v is updated according to the fired transition t.

The last branch of the choice in the loop is enabled if v is equal to M. Here, r can send reach_M to s, which gives p the opportunity to send the l message to q, allowing it to match the send transition from the lower branch in the top level choice of G_N.

Finally, the middle branch allows r to abort the simulation at any point and start over. This ensures that if the simulation ever reaches a dead state due to firing a transition that would render M unreachable, it can recover by starting again from M_0. Thus, for all states of the simulator, p has an ε path from that state to a state where it can send l to q iff M is reachable from M_0 in N. The only other sender is r which makes all choices and, hence, never reaches two different states along the same prefix trace. It thus satisfies Send Coherence trivially. It follows that Send Coherence holds iff M is reachable from M_0 in N. To see that Receive Coherence holds, observe that s is always immediately informed about the choice taken by r. Moreover, the only other receiving participant s only ever receives one possible message value from p (if any).

G_N is deadlock-free because the branch in the loop of G_t where r sends the *restart* message is always enabled. Moreover, it is easy to see that G_N is deterministic because each branch of a choice sends a different message value.

In summary, G_N is a GCLTS that is implementable iff N reaches M. The size of $\mathbf{G}_N$ is linear in the size of N, so we obtain the desired reduction. □

6 RELATED WORK

Table 1 summarizes the most closely related works that address the implementability problem of communication protocols with data refinements. We discuss these works in terms of key expressive features and completeness of characterization.

Expressivity. All existing works in Table 1 effectively require *history-sensitivity*, which means that a "predicate guaranteed by a [participant p] can only contain those interaction variables that [p] knows" [8], see also [7, Def. 2]. As discussed in §4, syntactic approaches to analyzing variable knowledge is over-conservative, and as a result no prior work can handle protocols such as the example in Fig. 12. In a similar vein, [72] imposes the syntactic restriction that all participants in a

Paper	Communication paradigm	Branching restrictions	History sensitivity	Characterization
[8]	asynchronous	directed choice	required	incomplete
[7]	asynchronous	directed choice	required	incomplete
[65]	synchronous	directed choice	required	incomplete
[72]	synchronous	directed choice	required	incomplete
[30]	synchronous	well-sequencedness	required	unknown
this work	asynchronous	sender-driven choice	not required	relatively complete

Table 1. Comparison of related work (in chronological order)

loop must be able to update all loop registers, which rules out loops like the one in the two-bidder protocol (Fig. 1).

Furthermore, all prior works except for [30] employ the directed choice restriction, which is strictly less general than sender-driven choice. Many of these works also feature separate constructs for selecting branches and sending data. In our symbolic protocols, this is not necessary because selecting branches can be modeled with equality predicates, as demonstrated by Fig. 7. Gheri et al. [30] generalize choreography automata, which are finite-state LTSs with communciation events as transition labels but without final states. One major difference between our work and theirs lies in the treatment of interleavings. Unlike our protocol semantics, which are closed under the indistinguishability relation $\sim$, inspired by Lamport's happened-before relation, choreography automata languages do not include any interleavings not present in the language. Setting aside asynchronous traces, the protocol $\mathtt{p} \to \mathtt{q} : m.\, \mathtt{r} \to \mathtt{s} : m.\, 0$ in our setting would need to be represented as $\mathtt{p} \to \mathtt{q} : m.\, \mathtt{r} \to \mathtt{s} : m.\, 0 \;+\; \mathtt{r} \to \mathtt{s} : m.\, \mathtt{p} \to \mathtt{q} : m.\, 0$ in their setting, and the protocol $\mu t.\, \mathtt{p} \to \mathtt{q} : m.\, \mathtt{r} \to \mathtt{s} : m.\, t$ does not admit a representation as a choreography automaton. The branching behaviors are restricted with a well-sequencedness condition [30, Def. 3.2], a condition that has since been refined because it was shown to be flawed [25]. [52] showed that well-formedness conditions on synchronous choreography automata do not generalize soundly to the asynchronous setting.

Asynchronous communication is more challenging to analyze in general because it easily gives rise to infinite-state systems. Zhou [71] conjectures that the framework in [72] "can be extended to support asynchronous communication", but does not conjecture if and how the projection operator would change. Due to directed choice, the same projection operator may remain sound under asynchronous semantics, because it rules out protocols where participants have a choice to receive from different senders. However, it will also likely inherit the same sources of incompleteness present in the synchronous setting.

In contrast to all aforementioned works, [10, 11, 16] allow to specify send and receive events separately with "deconfined" global types. Deconfined global types are specified as a parallel composition of local processes, and then checked for desirable correctness properties, which were shown to be undecidable [16].

Completeness. Implementability is a thoroughly-studied problem in the high-level message sequence chart (HMSC) literature. HMSCs are a standardized formalism for describing communication protocols in industry [66] and are well-studied in academia [26–28, 54, 58]. In the HMSC setting, implementability is called safe realizability, and is defined with respect to the implementation model of communicating finite state machines [9]. Similar to our setting, a canonical implementation exists for any HMSC [1, Thm. 13]; unlike our setting, it is always computable. Therefore, existing work has focused less on synthesis and more on checking implementability. Despite having only finite states and data, HMSC implementability was shown to be undecidable in general [50]. Various fragments have since been identified in which the problem regains decidability. Lohrey [50] showed implementability to be EXPSPACE-complete for bounded HMSCs [3, 56] and globally-cooperative HMSCs [29, 55]. These fragments restrict the communication topology of loops to be strongly and weakly connected respectively. For HMSCs where every two consecutive communications share a participant, implementability was shown to be PSPACE-complete [50].

In contrast, works that study comparably expressive protocol fragments to ours often sidestep the implementability question. Instead, implementability is addressed in the form of syntactic well-formedness conditions, as mentioned above, or indirectly through synthesis. None of the prior works attempted to show completeness; [48, 63] later showed all but [30] to be incomplete. [7, 8, 65, 72] all synthesize local implementations using the "classical" projection from multiparty session types. One kind of merge operator, called the plain merge, allows only the two participants

in a choice to exhibit different behavior on each branch, a condition which is breached by our two-bidder protocol (Fig. 1). Zhou et al. [72] proves the soundness of projection with plain merge, but implements a more permissive variant called full merge in the toolchain. However, the projected local types are not guaranteed to be implementable: both Fig. 11 and Fig. 12 are projectable in [72]. Thus, the implementability problem is deferred to local types.

Our results show that synthesis is "as possible as" the determinization of the non-deterministic underlying automata fragment. This means that implementations can be synthesized even for expressive classes of protocols that correspond to e.g. symbolic finite automata [19, 62] and certain classes of timed and register automata [6, 13] due to the existence of off-the-shelf determinization algorithms for these classes [5, 68, 69].

Scalas and Yoshida [61] check safety properties of collections of local types by encoding the properties as μ-calculus formulas and then model checking the typing context against the specification. They focus primarily on finite-state typing contexts under synchronous semantics, and thus all properties in their setting are decidable. For the asynchronous setting, only three sound approximations of safety are proposed, one of which bounds channel sizes and thus falls back into the finite-state setting.

Next, we discuss further related works on choreographic programming and binary session types.

Choreographic programming. Choreographic programming [15, 31, 39] describes global message-passing behaviors as programs rather than protocols, and therefore incorporate many more programming language features that are abstracted away in our model, such as computation and mutable state, in addition to features that our model cannot express, such as higher-order computations and delegation. Endpoint projection for choreographic programs, which shares a theoretical basis with multiparty session type projection, then generates individual, executable programs for each participant. The question of implementability, though undecidable in the presence of such expressivity, remains relevant to the soundness of endpoint projections. We discuss three approaches to endpoint projection. Pirouette [38] requires the programmer to specify explicit synchronization messages to ensure that "different locations stay in lock-step with each other", and conservatively rejects programs that are underspecified in this regard. Pirouette provides a mechanized proof of deadlock freedom for endpoint projections in Coq. Note that the claims of soundness and completeness in [38] are not with respect to implementability, but with respect to the translation via endpoint projection. HasChor [62] rules out non-implementability by automatically incorporating location broadcasts when a choice is made. No formal correctness claims are made in [62]. Jongmans and van den Bos [44] allow if- and while- statements to be annotated with a conjunction of conditional choices for each participant, which expresses decentralized decision-making in protocols. Jongmans and van den Bos [44] show that their endpoint projection for well-formed choreographies guarantees deadlock freedom and functional correctness. All aforementioned choreographic programming works assume a synchronous network.

Binary session types with refinements. Finally, we briefly mention work on binary session types with refinements and data dependencies. In the binary setting, implementability is a less interesting problem due to the inherent duality between the two protocol participants; the distinction between global and local types is no longer meaningful. [34] refines binary sessions with basic data types, and shows decidability of the subtyping problem. [33] applies a similar type system for runtime monitoring of binary communication. Thiemann and Vasconcelos [64] propose a label-dependent binary session type framework which allows the subsequent behavior of the protocol to depend on previous labels, which are drawn from a finite set. Das and Pfenning [21] study the undecidable problem of local type equality, and provide a sound approximate algorithm. [22] and [20] further

apply binary session types with refinements to resource analysis of blockchain smart contracts and amortized cost analysis.

Actris [35] embeds binary session types into the Iris framework [45]. The framework assumes asynchronous communication with FIFO channels, and can verify programs that combine message-passing concurrency and shared-memory concurrency. Actris has been extended with session type subtyping (Actris 2.0 [36]) and with linearity to prove both preservation and progress (LinearActris [42]). Multris [37] is an extension of Actris in Iris to the multiparty setting. The message-passing layer of Multris is more restricted than Actris: Multris assumes synchronous communication and prohibits choice over channels: choices can only be made about message values between a given sender and receiver. Multris takes a bottom-up approach [61] to correctness: given a collection of local types, the type system checks that they can be safely combined. Currently, Multris guarantees protocol fidelity but not progress.

ACKNOWLEDGMENTS

This work is funded in parts by the National Science Foundation under grant 2304758.

REFERENCES

[1] 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

[2] Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. 2005. Realizability and verification of MSC graphs. *Theor. Comput. Sci.* 331, 1 (2005), 97–114. https://doi.org/10.1016/J.TCS.2004.09.034

[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] Davide Ancona, Viviana Bono, Mario Bravetti, Joana Campos, Giuseppe Castagna, Pierre-Malo Deniélou, Simon J. Gay, Nils Gesbert, Elena Giachino, Raymond Hu, Einar Broch Johnsen, Francisco Martins, Viviana Mascardi, Fabrizio Montesi, Rumyana Neykova, Nicholas Ng, Luca Padovani, Vasco T. Vasconcelos, and Nobuko Yoshida. 2016. Behavioral Types in Programming Languages. *Found. Trends Program. Lang.* 3, 2-3 (2016), 95–230. https://doi.org/10.1561/2500000031

[5] Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, and Pierre Carlier. 2018. When are stochastic transition systems tameable? *J. Log. Algebraic Methods Program.* 99 (2018), 41–96. https://doi.org/10.1016/J.JLAMP.2018.03.004

[6] Nathalie Bertrand, Amélie Stainer, Thierry Jéron, and Moez Krichen. 2015. A game approach to determinize timed automata. *Formal Methods Syst. Des.* 46, 1 (2015), 42–80. https://doi.org/10.1007/S10703-014-0220-1

[7] 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

[8] 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

[9] 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

[10] Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Paola Giannini. 2022. Asynchronous Sessions with Input Races. In *Proceedings of the 13th International Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software, PLACES@ETAPS 2022, Munich, Germany, 3rd April 2022 (EPTCS, Vol. 356)*, Marco Carbone and Rumyana Neykova (Eds.). 12–23. https://doi.org/10.4204/EPTCS.356.2

[11] Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Paola Giannini. 2024. Global Types and Event Structure Semantics for Asynchronous Multiparty Sessions. *Fundam. Informaticae* 192, 1 (2024), 1–75. https://doi.org/10.3233/FI-242188

[12] Ruofei Chen, Stephanie Balzer, and Bernardo Toninho. 2022. Ferrite: A Judgmental Embedding of Session Types in Rust. In *36th European Conference on Object-Oriented Programming, ECOOP 2022, June 6-10, 2022, Berlin, Germany (LIPIcs, Vol. 222)*, Karim Ali and Jan Vitek (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 22:1–22:28. https://doi.org/10.4230/LIPIcs.ECOOP.2022.22

[13] Lorenzo Clemente, Slawomir Lasota, and Radoslaw Piórkowski. 2022. Determinisability of register and timed automata. *Log. Methods Comput. Sci.* 18, 2 (2022). https://doi.org/10.46298/LMCS-18(2:9)2022

[14] Mario Coppo, Mariangiola Dezani-Ciancaglini, Luca Padovani, and Nobuko Yoshida. 2015. A Gentle Introduction to Multiparty Asynchronous Session Types. In *Formal Methods for Multicore Programming - 15th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2015, Bertinoro, Italy, June 15-19, 2015, Advanced Lectures (Lecture Notes in Computer Science, Vol. 9104)*, Marco Bernardo and Einar Broch Johnsen (Eds.). Springer, 146–178. https://doi.org/10.1007/978-3-319-18941-3_4

[15] 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

[16] Francesco Dagnino, Paola Giannini, and Mariangiola Dezani-Ciancaglini. 2021. Deconfined Global Types for Asynchronous Sessions. In *Coordination Models and Languages - 23rd IFIP WG 6.1 International Conference, COORDINATION 2021, Held as Part of the 16th International Federated Conference on Distributed Computing Techniques, DisCoTec 2021, Valletta, Malta, June 14-18, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12717)*, Ferruccio Damiani and Ornela Dardha (Eds.). Springer, 41–60. https://doi.org/10.1007/978-3-030-78142-2_3

[17] Haitao Dan, Robert M. Hierons, and Steve Counsell. 2010. Non-local Choice and Implied Scenarios. In *8th IEEE International Conference on Software Engineering and Formal Methods, SEFM 2010, Pisa, Italy, 13-18 September 2010*, José Luiz Fiadeiro, Stefania Gnesi, and Andrea Maggiolo-Schettini (Eds.). IEEE Computer Society, 53–62. https://doi.org/10.1109/SEFM.2010.14

[18] Loris D'Antoni, Tiago Ferreira, Matteo Sammartino, and Alexandra Silva. 2019. Symbolic Register Automata. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11561)*, Isil Dillig and Serdar Tasiran (Eds.). Springer, 3–21. https://doi.org/10.1007/978-3-030-25540-4_1

[19] Loris D'Antoni and Margus Veanes. 2017. The Power of Symbolic Automata and Transducers. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10426)*, Rupak Majumdar and Viktor Kuncak (Eds.). Springer, 47–67. https://doi.org/10.1007/978-3-319-63387-9_3

[20] Ankush Das, Stephanie Balzer, Jan Hoffmann, Frank Pfenning, and Ishani Santurkar. 2021. Resource-Aware Session Types for Digital Contracts. In *34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, June 21-25, 2021*. IEEE, 1–16. https://doi.org/10.1109/CSF51468.2021.00004

[21] Ankush Das and Frank Pfenning. 2020. Session Types with Arithmetic Refinements. In *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference) (LIPIcs, Vol. 171)*, Igor Konnov and Laura Kovács (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 13:1–13:18. https://doi.org/10.4230/LIPICS.CONCUR.2020.13

[22] Ankush Das and Frank Pfenning. 2022. Rast: A Language for Resource-Aware Session Types. *Log. Methods Comput. Sci.* 18, 1 (2022). https://doi.org/10.46298/LMCS-18(1:9)2022

[23] Javier Esparza and Mogens Nielsen. 1994. Decidability Issues for Petri Nets - a survey. *J. Inf. Process. Cybern.* 30, 3 (1994), 143–160.

[24] 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

[25] 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

[26] 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

[27] 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

[28] 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

[29] 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

[30] Lorenzo Gheri, Ivan Lanese, Neil Sayers, Emilio Tuosto, and Nobuko Yoshida. 2022. Design-By-Contract for Flexible Multiparty Session Protocols. In *36th European Conference on Object-Oriented Programming, ECOOP 2022, June 6-10, 2022, Berlin, Germany (LIPIcs, Vol. 222)*, Karim Ali and Jan Vitek (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 8:1–8:28. https://doi.org/10.4230/LIPICS.ECOOP.2022.8

[31] 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

[32] Patrice Godefroid and Mihalis Yannakakis. 2013. Analysis of Boolean Programs. In *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7795)*, Nir Piterman and Scott A. Smolka (Eds.). Springer, 214–229. https://doi.org/10.1007/978-3-642-36742-7_16

[33] Hannah Gommerstadt, Limin Jia, and Frank Pfenning. 2018. Session-Typed Concurrent Contracts. In *Programming Languages and Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10801)*, Amal Ahmed (Ed.). Springer, 771–798. https://doi.org/10.1007/978-3-319-89884-1_27

[34] Dennis Griffith and Elsa L. Gunter. 2013. LiquidPi: Inferrable Dependent Session Types. In *NASA Formal Methods, 5th International Symposium, NFM 2013, Moffett Field, CA, USA, May 14-16, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7871)*, Guillaume Brat, Neha Rungta, and Arnaud Venet (Eds.). Springer, 185–197. https://doi.org/10.1007/978-3-642-38088-4_13

[35] 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

[36] 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

[37] 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

[38] 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

[39] 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

[40] 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

[41] 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

[42] Jules Jacobs, Jonas Kastberg Hinrichsen, and Robbert Krebbers. 2024. Deadlock-Free Separation Logic: Linearity Yields Progress for Dependent Higher-Order Message Passing. *Proc. ACM Program. Lang.* 8, POPL (2024), 1385–1417. https://doi.org/10.1145/3632889

[43] Thomas Bracht Laumann Jespersen, Philip Munksgaard, and Ken Friis Larsen. 2015. Session types for Rust. In *Proceedings of the 11th ACM SIGPLAN Workshop on Generic Programming, WGP@ICFP 2015, Vancouver, BC, Canada, August 30, 2015*, Patrick Bahr and Sebastian Erdweg (Eds.). ACM, 13–22. https://doi.org/10.1145/2808098.2808100

[44] Sung-Shik Jongmans and Petra van den Bos. 2022. A Predicate Transformer for Choreographies - Computing Preconditions in Choreographic Programming. In *Programming Languages and Systems - 31st European Symposium on Programming, ESOP 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings (Lecture Notes in Computer Science, Vol. 13240)*, Ilya Sergey (Ed.). Springer, 520–547. https://doi.org/10.1007/978-3-030-99336-8_19

[45] Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Ales Bizjak, Lars Birkedal, and Derek Dreyer. 2018. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *J. Funct. Program.* 28 (2018), e20. https://doi.org/10.1017/S0956796818000151

[46] Leslie Lamport. 1978. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM* 21, 7 (1978), 558–565. https://doi.org/10.1145/359545.359563

[47] 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

[48] 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

[49] Sam Lindley and J. Garrett Morris. 2016. Embedding session types in Haskell. In *Proceedings of the 9th International Symposium on Haskell, Haskell 2016, Nara, Japan, September 22-23, 2016*, Geoffrey Mainland (Ed.). ACM, 133–145. https://doi.org/10.1145/2976002.2976018

[50] 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

[51] Rupak Majumdar, Madhavan Mukund, Felix Stutz, and Damien Zufferey. 2021. Generalising Projection in Asynchronous Multiparty Session Types. 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, 35:1–35:24. https://doi.org/10.4230/LIPICS.CONCUR.2021.35

[52] Rupak Majumdar, Madhavan Mukund, Felix Stutz, and Damien Zufferey. 2021. Generalising Projection in Asynchronous Multiparty Session Types. *CoRR* abs/2107.03984 (2021). arXiv:2107.03984 https://arxiv.org/abs/2107.03984

[53] 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

[54] 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.

[55] 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

[56] 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

[57] 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

[58] 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

[59] Alceste Scalas, Ornela Dardha, Raymond Hu, and Nobuko Yoshida. 2017. A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming. In *31st European Conference on Object-Oriented Programming, ECOOP 2017, June 19-23, 2017, Barcelona, Spain (LIPIcs, Vol. 74)*, Peter Müller (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 24:1–24:31. https://doi.org/10.4230/LIPICS.ECOOP.2017.24

[60] Alceste Scalas and Nobuko Yoshida. 2016. Lightweight Session Programming in Scala. In *30th European Conference on Object-Oriented Programming, ECOOP 2016, July 18-22, 2016, Rome, Italy (LIPIcs, Vol. 56)*, Shriram Krishnamurthi and Benjamin S. Lerner (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 21:1–21:28. https://doi.org/10.4230/LIPIcs.ECOOP.2016.21

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

[62] 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

[63] Felix Stutz. 2023. Asynchronous Multiparty Session Type Implementability is Decidable - Lessons Learned from Message Sequence Charts. 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, 32:1–32:31. https://doi.org/10.4230/LIPICS.ECOOP.2023.32

[64] Peter Thiemann and Vasco T. Vasconcelos. 2020. Label-dependent session types. *Proc. ACM Program. Lang.* 4, POPL (2020), 67:1–67:29. https://doi.org/10.1145/3371135

[65] 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

[66] International Telecommunication Union. 1996. *Z.120: Message Sequence Chart.* Technical Report. International Telecommunication Union. https://www.itu.int/rec/T-REC-Z.120

[67] 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

[68] Margus Veanes and Nikolaj S. Bjørner. 2012. Symbolic Automata: The Toolkit. In *Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference, TACAS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings (Lecture Notes in Computer Science, Vol. 7214)*, Cormac Flanagan and Barbara König (Eds.). Springer, 472–477. https://doi.org/10.1007/978-3-642-28756-5_33

[69] Margus Veanes, Peli de Halleux, and Nikolai Tillmann. 2010. Rex: Symbolic Regular Expression Explorer. In *Third International Conference on Software Testing, Verification and Validation, ICST 2010, Paris, France, April 7-9, 2010.* IEEE Computer Society, 498–507. https://doi.org/10.1109/ICST.2010.15

[70] 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

[71] Fangyi Zhou. 2024. *Refining Multiparty Session Types.* Ph. D. Dissertation. Imperial College London.

[72] 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

A ADDITIONAL MATERIAL FOR SECTION 4

Lemma 4.7 (No Mixed Choice). *Let $\mathcal{S}$ be a protocol satisfying NMC (Definition 4.3) and let $\{\!\{T_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ be a canonical implementation for $\mathcal{S}$. Let $wx_1, wx_2 \in \mathrm{pref}(\mathcal{L}(T_{\mathsf{p}}))$ with $x_1 \neq x_2$ for some $\mathsf{p} \in \mathcal{P}$. Then, $x_1 \in \Sigma_!$ iff $x_2 \in \Sigma_!$.*

Proof. Suppose by contradiction that $x_1 \in \Sigma_?$ and $x_2 \in \Sigma_!$. Let ρ_1 be a run in $\mathcal{S}$ such that $wx_1 \leq \mathtt{split}(\mathtt{trace}(\rho_1))\Downarrow_{\Sigma_{\mathsf{p}}}$. Let $\alpha_1 \cdot s_1 \xrightarrow{l_1} s_1' \cdot \beta_1$ be the unique splitting of ρ for p with respect to w. Then, p is the receiver in l_1 and $\mathtt{split}(\mathtt{trace}(\alpha_1 \cdot s_1))\Downarrow_{\Sigma_{\mathsf{p}}} = w$. Let ρ_2 be a run in $\mathcal{S}$ such that $wx_2 \leq \mathtt{split}(\mathtt{trace}(\rho_2))\Downarrow_{\Sigma_{\mathsf{p}}}$. Let $\alpha_2 \cdot s_2 \xrightarrow{l_2} s_2' \cdot \beta_2$ be the unique splitting of ρ_2 for p with respect to w. Then, p is the sender in l_2 and $\mathtt{split}(\mathtt{trace}(\alpha_2 \cdot s_2))\Downarrow_{\Sigma_{\mathsf{p}}} = w$. If $s_1 = s_2$, then we find a violation to the assumption that $\mathcal{S}$ is sender-driven. Hence, $s_1 \neq s_2$ and we can instantiate NMC (Definition 4.3) with $s_2 \xrightarrow{l_2} s_2'$, s_1 and w to obtain a contradiction. □

Lemma 4.15 (Channel compliance and intersection set non-emptiness implies prefix). *Let $\mathcal{S} = (S, \Gamma, T, s_0, F)$ be a protocol and let $w \in \Sigma^*$ be a word such that (i) w is channel-compliant, and (ii) $I(w) \neq \emptyset$. Then, $w \in \mathrm{pref}(\mathcal{L}(\mathcal{S}))$.*

Proof. Let ρ be a run in $I(w)$, and let $w' = \mathtt{split}(\mathtt{trace}(\rho)) \in \mathcal{L}(\mathcal{S})$. In the case that $I(w)$ contains finite runs, we can pick a finite ρ. Otherwise, ρ is infinite. We reason about each case in turn.

Case: ρ is a finite run. In the case that ρ is a finite run, to show that $w \in \mathrm{pref}(\mathcal{L}(\mathcal{S}))$ we need to show the existence of a $w'' \in \mathcal{L}(\mathcal{S})$ such that $w \leq w''$. We construct such a w'' by constructing a u such that in wu, all participants have completed their actions in ρ, and furthermore wu is channel-compliant. Then, because w' is channel-compliant by construction, and for all participants $\mathsf{p} \in \mathcal{P}$, it holds that $wu\Downarrow_{\Sigma_{\mathsf{p}}} = w'\Downarrow_{\Sigma_{\mathsf{p}}}$, by [51, Lemma 23] it follows that $wu \sim w'$, and thus $wu \in \mathcal{L}(\mathcal{S})$.

For each participant $\mathsf{p} \in \mathcal{P}$, let y_{p} be defined such that $w\Downarrow_{\Sigma_{\mathsf{p}}} \cdot y_{\mathsf{p}} = w'\Downarrow_{\Sigma_{\mathsf{p}}}$. We construct u from the y_{p} for each participant, starting with $u = \varepsilon$. If there exists some participant in $\mathcal{P}$ such that $y_{\mathsf{p}}[0] \in \Sigma_{\mathsf{p},!}$, append y_{p} to u and update y_{p}. If not, for all participants $\mathsf{p} \in \mathcal{P}$, $y_{\mathsf{p}}[0] \in \Sigma_{\mathsf{p},?}$. Each symbol $y_{\mathsf{p}}[0]$ for all participants appears in v. Let i_{p} denote for each participant the index in w' such that $w'[i] = y_{\mathsf{p}}[0]$. Let r be the participant with the minimum index i_{r}: append y_{r} to u and update y_{r}. Termination is guaranteed by the strictly decreasing measure of $\sum_{\mathsf{p}\in\mathcal{P}} |y_{\mathsf{p}}|$. Furthermore, it is clear that upon termination, for all participants $\mathsf{p} \in \mathcal{P}$, $wu\Downarrow_{\Sigma_{\mathsf{p}}} = w'\Downarrow_{\Sigma_{\mathsf{p}}}$.

We argue that wu satisfies the inductive invariant of channel compliancy. In the case where u is extended with a send action, channel compliancy is trivially re-established. In the receive case, channel compliancy is re-established by the fact that the append order for receive actions follows that in v, which is channel-compliant by construction.

Case: ρ is an infinite run. In the case that ρ is a infinite run, to show that $w \in \mathrm{pref}(\mathcal{L}(\mathcal{S}))$ we likewise need to show the existence of a $w'' \in \mathcal{L}(\mathcal{S})$ such that $w \leq w''$. Like before, we construct a u and show that $wu \in \mathcal{L}(\mathcal{S})$. However, unlike before, we cannot rely on the fact that $wu \sim w'$ to show that $wu \in \mathcal{L}(\mathcal{S})$, because w' is an infinite word and [51, Lemma 23] applies only to finite words. Instead, we must prove that $wu \in \mathcal{L}(\mathcal{S})$ by the definition of infinite word membership in $\mathcal{L}(\mathcal{S})$, namely: $wu \preceq^{\omega}_{\sim} w'$. By the definition of $\preceq^{\omega}_{\sim}$, it further suffices to show that:

$$\forall v \leq wu,\ \exists v' \leq w', u' \in \Sigma^*.\ vu' \sim v' \quad .$$

For each participant $\mathsf{p} \in \mathcal{P}$, let ρ_{p} be defined as the largest prefix of ρ such that $\mathtt{split}(\mathtt{trace}(\rho_{\mathsf{p}}))\Downarrow_{\Sigma_{\mathsf{p}}} = w\Downarrow_{\Sigma_{\mathsf{p}}}$. Let s be the participant with the maximum $|\rho_{\mathsf{s}}|$ in $\mathcal{P}$. Clearly, $\rho_{\mathsf{s}} \leq \rho$. Let β be defined such that $\rho = \rho_{\mathsf{s}} \cdot \beta$. We split the construction of u into two parts: let $u = u_1u_2$. We construct u_1 as

above, by appending uncompleted actions in ρ_s, ordering send events before receive events, and further ordering receive events by the order in which they appear in ρ_s. Then, upon termination, wu_1 is channel-compliant and satisfies for all $\mathsf{p} \in \mathcal{P}$, $wu_1{\Downarrow_{\Sigma_\mathsf{p}}} = \mathtt{split}(\mathtt{trace}(\rho_s)){\Downarrow_{\Sigma_\mathsf{p}}}$. Let $u_2 = \mathtt{split}(\mathtt{trace}(\beta))$.

We now show that $wu_1u_2 \preceq^{\omega}_{\sim} w'$.

Let v be an arbitrary prefix of wu_1u_2. If $v \leq wu_1$, we pick $v' = \mathtt{split}(\mathtt{trace}(\rho_s)) \leq w'$ and $u' \in \Sigma^*$ to be such that $vu' = wu_1$. Otherwise, if $wu_1 < v$, let ρ' be defined as the smallest prefix of ρ such that for all participants $\mathsf{p} \in \mathcal{P}$, $v{\Downarrow_{\Sigma_\mathsf{p}}} = \mathtt{split}(\mathtt{trace}(\rho')){\Downarrow_{\Sigma_\mathsf{p}}}$. We pick $v' = \mathtt{split}(\mathtt{trace}(\rho'))$. Because v is channel-compliant, we can repeat the reasoning in the finite case to extend v with u' and apply [51, Lemma 23] to conclude that $vu' \sim v'$. □

Lemma 4.8 (Canonical implementation language contains protocol language). *Let $\mathcal{S}$ be an LTS and let $\{\!|T_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}}$ be a canonical implementation for $\mathcal{S}$. Then, $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\{\!|T_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}})$.*

Proof. Let w be a word in $\mathcal{L}(\mathcal{S})$. Prior to case splitting on whether w is a finite or infinite word, we establish a claim that is used in both cases.

Claim 1. $\mathrm{pref}(\mathcal{L}(\mathcal{S})) \subseteq \mathrm{pref}(\mathcal{L}(\{\!|T_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}}))$. Let $w \in \mathrm{pref}(\mathcal{L}(\mathcal{S}))$. We prove that $w \in \mathrm{pref}(\mathcal{L}(\{\!|T_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}}))$ via structural induction on w. The base case, $w = \varepsilon$, is trivial. For the inductive step, let $wx \in \mathrm{pref}(\mathcal{L}(\mathcal{S}))$. From the induction hypothesis, $w \in \mathrm{pref}(\mathcal{L}\{\!|T_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}})$. It suffices to show that the transition labeled with x is enabled for the active participant in x. Let $(\vec{s}, \xi)$ denote the $\{\!|T_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}}$ configuration reached on w. In the case that $x \in \Sigma_!$, let $x = \mathsf{p} \triangleright \mathsf{q}!m$. The existence of an outgoing transition $\xrightarrow{\mathsf{p}\triangleright\mathsf{q}!m}$ from $\vec{s}_\mathsf{p}$ follows from the fact that $\mathrm{pref}(\mathcal{L}(\mathcal{S})){\Downarrow_{\Sigma_\mathsf{p}}} \subseteq \mathrm{pref}(\mathcal{L}(T_\mathsf{p}))$ (Definition 4.6). The fact that $wx \in \mathrm{pref}(\mathcal{L}\{\!|T_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}})$ follows immediately from this and the fact that send transitions in a CLTS are always enabled. In the case that $x \in \Sigma_?$, let $x = \mathsf{p} \triangleleft \mathsf{q}?m$. We obtain an outgoing transition $\xrightarrow{\mathsf{p}\triangleleft\mathsf{q}?m}$ from $\vec{s}_\mathsf{p}$ analogously. We additionally need to show that $\xi(\mathsf{q}, \mathsf{p})$ contains m at the head. This follows from the fact that w is channel-compliant (**??**) and the induction hypothesis. This concludes our proof of prefix set inclusion. *End Proof of Claim 1.*

Case: $w \in \Sigma^*$. In the finite case, it remains to show that $\{\!|T_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}}$ reaches a final configuration on w. From the canonicity of $\{\!|T_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}}$, it holds that all states in $\vec{s}$ are final. From the fact that all finite words in $\mathcal{L}(\mathcal{S})$ contain matching receive events, all channels in ξ are empty.

Case: $w \in \Sigma^\omega$. The infinite case when $w \in \Sigma^\infty$ is immediate from Claim 1. □

Lemma 4.9 (Global protocol language contains canonical implementation language). *Let $\mathcal{S}$ be a protocol satisfying* CC *and let $\{\!|T_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}}$ be a canonical implementation for $\mathcal{S}$ such that for all $w \in \Sigma^*$, if w is a trace of $\{\!|T_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}}$, then $I(w) \neq \emptyset$. Then, $\mathcal{L}(\{\!|T_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}}) \subseteq \mathcal{L}(\mathcal{S})$.*

Proof. Let $w \in \mathcal{L}\{\!|T_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}}$. We again case split on whether w is a finite or infinite word.

Case: $w \in \Sigma^*$. First, we establish that w is terminated. Let $(\vec{s}, \xi)$ be the $\{\!|T_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}}$ configuration reached on w. Because w is a finite, maximal word in $\mathcal{L}(\{\!|T_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}})$, it holds that all states in $\vec{s}$ are final, and all channels in ξ are empty. Therefore, no receive transitions are enabled from $(\vec{s}, \xi)$. We argue that no send transitions are enabled from $(\vec{s}, \xi)$ either. Suppose by contradiction that there exists an outgoing transition $\vec{s}_\mathsf{p} \xrightarrow{\mathsf{p}\triangleright\mathsf{q}!m} s' \in T_\mathsf{p}$ for participant p. Then, $w{\Downarrow_{\Sigma_\mathsf{p}}} \cdot \mathsf{p} \triangleright \mathsf{q}!m \in \mathrm{pref}(\mathcal{L}(T_\mathsf{p}))$, and by the canonicity of T_p, $w{\Downarrow_{\Sigma_\mathsf{p}}} \cdot \mathsf{p} \triangleright \mathsf{q}!m \in \mathrm{pref}(\mathcal{L}(\mathcal{S})){\Downarrow_{\Sigma_\mathsf{p}}}$. Then, there exists a maximal run ρ' in $\mathcal{S}$ such that $w{\Downarrow_{\Sigma_\mathsf{p}}} \cdot \mathsf{p} \triangleright \mathsf{q}!m \leq \mathtt{split}(\mathtt{trace}(\rho')){\Downarrow_{\Sigma_\mathsf{p}}}$. Furthermore, there exists a finite, maximal run ρ_{fin} in $\mathcal{S}$ such that $w{\Downarrow_{\Sigma_\mathsf{p}}} = \mathtt{split}(\mathtt{trace}(\rho)){\Downarrow_{\Sigma_\mathsf{p}}}$. Let s_{fin} be the last state in ρ_{fin}. By assumption, $s_{fin} \in F$. Let $\alpha \cdot s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:m} s_2 \cdot \beta$ be the unique splitting of ρ' for p with respect to w. Then, s_1

and s_{fin} are simultaneously reachable for p on prefix $w{\Downarrow_{\Sigma_\mathsf{p}}}$. From SC, there exists a s_2' such that $s_{fin} \xRightarrow[\mathsf{p}]{\mathsf{p}\to\mathsf{q}:m}{}^{*} s_2'$. We find a contradiction to the assumption that final states in $\mathcal{S}$ do not have outgoing transitions.

Next, we show that for every $\rho \in I(w)$, it holds that for every $\mathsf{p} \in \mathcal{P}$, $w{\Downarrow_{\Sigma_\mathsf{p}}} = \mathtt{split}(\mathtt{trace}(\rho)){\Downarrow_{\Sigma_\mathsf{p}}}$. This implies that there exist no infinite runs in $I(w)$. Suppose by contradiction that there exists a run $\rho \in I(w)$ and a non-empty set of participants $\mathcal{Q}$ such that for every $\mathsf{r} \in \mathcal{Q}$, it holds that $w{\Downarrow_{\Sigma_\mathsf{r}}} < (\mathtt{split}(\mathtt{trace}(\rho))){\Downarrow_{\Sigma_\mathsf{r}}}$ (*). Given a participant p, let ρ_p denote the largest prefix of ρ that contains p's local view of w. Formally,

$$\rho_\mathsf{p} = max\{\rho' \mid \rho' \leq \rho \ \wedge \ \mathtt{split}(\mathtt{trace}(\rho')){\Downarrow_{\Sigma_\mathsf{p}}} = w{\Downarrow_{\Sigma_\mathsf{p}}}\} \ .$$

Note that due to maximality, the next transition in ρ after ρ_p must have p as its active participant. Let q be the participant in $\mathcal{S}$ for whom ρ_q is the smallest. From the canonicity of T_q and (*), it follows that $\vec{s}_\mathsf{q}$ has outgoing transitions. If $\vec{s}_\mathsf{q}$ has outgoing send transitions, then we reach a contradiction to the fact that w is terminated. If $\vec{s}_\mathsf{q}$ has outgoing receive transitions, it must be the case that the next transition in ρ after ρ_q is of the form $\mathsf{p} \to \mathsf{q} : m$ for some p and m. From the fact that q is the participant with the smallest ρ_q, we know that $\rho_\mathsf{q} < \rho_\mathsf{p}$, and from the FIFO property of CLTS channels it follows that m is in $\xi(\mathsf{p}, \mathsf{q})$. Then, the receive transition is enabled for q, and we again reach a contradiction to the fact that w is terminated.

Thus, we can pick any finite run $\rho \in I(w)$ which is maximal by definition, and invoke [51, Lemma 23] to conclude that $\mathtt{split}(\mathtt{trace}(\rho)) \sim w$, and thus $w \in \mathcal{L}(\mathcal{S})$.

Case: $w \in \Sigma^\infty$. By the semantics of $\mathcal{L}(\mathcal{S})$, to show $w \in \mathcal{L}(\mathcal{S})$ it suffices to show:

$$\exists w' \in \Sigma^\omega.\ w' \in \mathtt{split}(\mathcal{L}(\mathcal{S})) \wedge w \preceq_\sim^\omega w' \ .$$

Claim. $\bigcap_{u \leq w} I(u)$ contains an infinite run.

First, we show that there exists an infinite run in $\mathcal{S}$. We apply König's Lemma to an infinite tree where each vertex corresponds to a finite run. We obtain the vertex set from the intersection sets of w's prefixes; each prefix "contributes" a set of finite runs. Formally, for each prefix $u \leq w$, let V_u be defined as:

$$V_u \coloneqq \bigcup_{\rho_u \in I(u)} \min\{\rho' \mid \rho' \leq \rho_u \wedge \forall \mathsf{p} \in \mathcal{P}.\ u{\Downarrow_{\Sigma_\mathsf{p}}} \leq \mathtt{split}(\mathtt{trace}(\rho')){\Downarrow_{\Sigma_\mathsf{p}}}\} \ .$$

By the assumption that $I(u) \neq \emptyset$, V_u is guaranteed to be non-empty. We construct a tree $\mathcal{T}_w(V, E)$ with $V \coloneqq \bigcup_{u \leq w} V_u$ and $E \coloneqq \{(\rho_1, \rho_2) \mid \rho_1 \leq \rho_2\}$. The tree is rooted in the empty run, which is included in V by the prefix ε. V is infinite because there are infinitely many prefixes of w. $\mathcal{T}_w$ is finitely branching due to the fact that $\mathcal{S}$ is deterministic: while there can be infinitely many transitions from a given state in S, there are only finitely many transitions from a given state in S on a particular transition label. In fact, there is only a single transition. Therefore, we can apply König's Lemma to obtain a ray in $\mathcal{T}_w$ representing an infinite run in $\mathcal{S}$.

Let ρ' be such an infinite run. We now show that $\rho' \in \bigcap_{u \leq w} I(u)$. Let v be a prefix of w. To show that $\rho' \in I(v)$, it suffices to show that one of the vertices in V_v lies on ρ'. In other words,

$$V_v \cap \{v \mid v \in \rho'\} \neq \emptyset \ .$$

Assume by contradiction that ρ' passes through none of the vertices in V_v. Then, for any $u' \geq u$, because intersection sets are monotonically decreasing, it must be the case that ρ' passes through none of the vertices in V_u'. Therefore, ρ' can only pass through vertices in V_u'', where $u'' \leq u$. However, the set $\bigcup_{u'' \leq u} V_u''$ has finite cardinality. We reach a contradiction, concluding our proof of the above claim.

Let $\rho' \in \bigcap_{u \leq w} I(u)$, and let $w' = \mathtt{split}(\mathtt{trace}(\rho'))$. It is clear that $w' \in \Sigma^\omega$ and $w' \in \mathtt{split}(\mathcal{L}(\mathcal{S}))$. It remains to show that $w \preceq^\omega_\sim w'$. By the definition of $\preceq^\omega_\sim$, it further suffices to show that:

$$\forall u \leq w,\ \exists u' \leq w', v \in \Sigma^*.\ uv \sim u' \ .$$

Let u be an arbitrary prefix of w. Because by definition $\rho' \in I(u)$, it holds that $u{\Downarrow_{\Sigma_\mathsf{p}}} \leq \mathtt{split}(\mathtt{trace}(\rho')){\Downarrow_{\Sigma_\mathsf{p}}}$.

For each participant $\mathsf{p} \in \mathcal{P}$, let ρ'_p be defined as the largest prefix of ρ' such that $\mathtt{split}(\mathtt{trace}(\rho'_\mathsf{p})){\Downarrow_{\Sigma_\mathsf{p}}} = u{\Downarrow_{\Sigma_\mathsf{p}}}$. Such a run is well-defined by the fact that u is a prefix of an infinite word w, and there exists a longer prefix v such that $u \leq v$ and $v{\Downarrow_{\Sigma_\mathsf{p}}} \leq \mathtt{split}(\mathtt{trace}(\rho')){\Downarrow_{\Sigma_\mathsf{p}}}$.

Let s be the participant with the maximum $|\rho'_\mathsf{s}|$ in $\mathcal{P}$. Let $u' = \mathtt{split}(\mathtt{trace}(\rho'_\mathsf{s}))$. Clearly, $u' \leq w'$. Because u' is $\mathtt{split}(\mathtt{trace}(\rho'_\mathsf{s}))$ for the participant with the longest ρ'_s, it holds for all participants $\mathsf{p} \in \mathcal{P}$ that $u{\Downarrow_{\Sigma_\mathsf{p}}} \leq u'{\Downarrow_{\Sigma_\mathsf{p}}}$. Then, there must exist $y_\mathsf{p} \in \Sigma^*_\mathsf{p}$ such that

$$u{\Downarrow_{\Sigma_\mathsf{p}}} \cdot y_\mathsf{p} = u'{\Downarrow_{\Sigma_\mathsf{p}}} \ .$$

Let y_p be defined in this way for each participant. We construct $v \in \Sigma^*$ such that $uv \sim u'$. Let v be initialized with ε. If there exists some participant in $\mathcal{P}$ such that $y_\mathsf{p}[0] \in \Sigma_{\mathsf{p},!}$, append y_p to v and update y_p. If not, for all participants $\mathsf{p} \in \mathcal{P}$, $y_\mathsf{p}[0] \in \Sigma_{\mathsf{p},?}$. Each symbol $y_\mathsf{p}[0]$ for all participants appears in u'. Let i_p denote for each participant the index in u' such that $u'[i] = y_\mathsf{p}[0]$. Let r be the participant with the minimum index i_r. Append y_r to v and update y_r. Termination is guaranteed by the strictly decreasing measure of $\sum_{\mathsf{p} \in \mathcal{P}} |y_\mathsf{p}|$.

We argue that uv satisfies the inductive invariant of channel compliancy. In the case where v is extended with a send action, channel compliancy is trivially re-established. In the receive case, channel compliancy is re-established by the fact that the append order for receive actions follows that in u', which is channel-compliant by construction. We conclude that $uv \sim u'$ by applying [51, Lemma 22]. □

Lemma 4.12 (Intersection set non-emptiness). *Let $\mathcal{S}$ be a protocol satisfying* CC*, and let $\{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p} \in \mathcal{P}}$ be a canonical CLTS for $\mathcal{S}$. Then, for every trace $w \in \Sigma^*$ of $\{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p} \in \mathcal{P}}$, it holds that $I(w) \neq \emptyset$.*

Proof. We prove the claim by induction on the length of w.

Base Case. $w = \varepsilon$. The trace $w = \varepsilon$ is trivially consistent with all maximal runs, and $I(w)$ therefore contains all maximal runs. By assumption, $\mathcal{S}$ contains at least one maximal run. Thus, $I(w)$ is non-empty.

Induction Step. Let wx be an extension of w by $x \in \Sigma$.

The induction hypothesis states that $I(w) \neq \emptyset$. To re-establish the induction hypothesis, we need to show $I(wx) \neq \emptyset$. We proceed by case analysis on whether x is a receive or send event.

Send Case. Let $x = \mathsf{p} \triangleright \mathsf{q}!m$. By Lemma 4.14, there exists a run in $I(wx)$ that shares a prefix with a run in $I(w)$. $I(wx) \neq \emptyset$ again follows immediately.

Receive Case. Let $x = \mathsf{p} \triangleleft \mathsf{q}?m$. By Lemma 4.13, $I(wx) = I(w)$. $I(wx) \neq \emptyset$ follows trivially from the induction hypothesis and this equality. □

Lemma 4.13 (Receive events do not shrink intersection sets). *Let $\mathcal{S}$ be a protocol satisfying* CC*, and let $\{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p} \in \mathcal{P}}$ be a canonical communicating LTS for $\mathcal{S}$. Let wx be a trace of $\{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p} \in \mathcal{P}}$ such that $x \in \Sigma_?$. Then, $I(w) = I(wx)$.*

Proof. Let $x = \mathsf{p} \triangleleft \mathsf{q}?m$. Because wx is a trace of $\{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p} \in \mathcal{P}}$, there exists a run $(\vec{s}_0, \xi_0) \xrightarrow{w}{}^* (\vec{s}, \xi) \xrightarrow{x} (\vec{s}', \xi')$ such that m is at the head of $\xi(\mathsf{q}, \mathsf{p})$.

We assume that $I(w)$ is non-empty; if $I(w)$ is empty then $I(wx)$ is trivially empty. To show $I(w) = I(wx)$, let $\rho \in I(w)$ and we show that $\rho \in I(wx)$. Recall that $I(wx)$ is defined as $\bigcap_{\mathsf{r} \in \mathcal{P}} \mathrm{R}^{\mathcal{S}}_{\mathsf{r}}(wx)$. Because $\mathrm{R}^{\mathcal{S}}_{\mathsf{r}}(wx) = \mathrm{R}^{\mathcal{S}}_{\mathsf{r}}(w)$ for every $\mathsf{r} \in \mathcal{P}$ with $\mathsf{r} \neq \mathsf{p}$, it suffices to show that $\rho \in \mathrm{R}^{\mathcal{S}}_{\mathsf{p}}(wx)$ to show $\rho \in I(wx)$.

We proceed via proof by contradiction so let $\rho \notin \mathrm{R}^{\mathcal{S}}_{\mathsf{p}}(wx)$ for $\rho \in I(w)$.

Let $\alpha \cdot s_{pre} \xrightarrow{l} s_{post} \cdot \beta$ be the unique splitting of ρ for p matching w. By definition of unique splittings, p is the active participant in l. Because $\rho \notin \mathrm{R}^{\mathcal{S}}_{\mathsf{p}}(wx)$, it follows that $l \neq \mathsf{q} \to \mathsf{p} : m$. By Lemma 4.7, p is the receiver in l, and l is of the form $\mathsf{r} \to \mathsf{p} : m'$, where $\mathsf{r} \neq \mathsf{q}$ or $m' \neq m$.

Before performing case analysis, we first establish a claim that is used in both cases. Let ρ_{p} denote the largest prefix of ρ that is consistent with w for p. Formally, $\rho_{\mathsf{p}} = \max\{\rho \mid \rho \leq \rho \wedge (\mathtt{split}(\mathtt{trace}(\rho)))\Downarrow_{\Sigma_{\mathsf{p}}} \leq w\Downarrow_{\Sigma_{\mathsf{p}}}\}$. Let ρ_{q} be defined analogously. It is clear that $\rho_{\mathsf{p}} = \alpha \cdot s_{pre}$.

Claim I. $\rho_{\mathsf{q}} > \rho_{\mathsf{p}}$.

From **??**, $\mathcal{V}(w\Downarrow_{\mathsf{q}\triangleright\mathsf{p}!\_}) = \mathcal{V}(w\Downarrow_{\mathsf{p}\triangleleft\mathsf{q}?\_}) \cdot \xi(\mathsf{q},\mathsf{p})$. Because $\rho_{\mathsf{p}} = \alpha \cdot s_{pre}$, it follows that $\mathcal{V}(w\Downarrow_{\mathsf{p}\triangleleft\mathsf{q}?\_}) = \mathcal{V}(\mathtt{split}(\mathtt{trace}(\alpha \cdot s_{pre}))\Downarrow_{\mathsf{p}\triangleleft\mathsf{q}?\_})$. Because m is at the head of $\xi(\mathsf{q},\mathsf{p})$ by assumption, there exists $u \in \mathcal{V}^*$ such that $\mathcal{V}(w\Downarrow_{\mathsf{q}\triangleright\mathsf{p}!\_}) = \mathcal{V}(\mathtt{split}(\mathtt{trace}(\alpha \cdot s_{pre})))\Downarrow_{\mathsf{p}\triangleleft\mathsf{q}?\_} \cdot m \cdot u$. Thus, $\mathcal{V}(w\Downarrow_{\mathsf{q}\triangleright\mathsf{p}!\_}) > \mathcal{V}(\mathtt{split}(\mathtt{trace}(\alpha \cdot s_{pre})))\Downarrow_{\mathsf{p}\triangleleft\mathsf{q}?\_}$ and $\rho_{\mathsf{q}} > \rho_{\mathsf{p}}$ follows. *End Proof of Claim I.*

Case: $\mathsf{r} = \mathsf{q}$ and $m' \neq m$.

We discharge this case by showing a contradiction to the assumption that m is at the head of $\xi(\mathsf{q},\mathsf{p})$. Because $\alpha \cdot s_{pre} \leq \rho_{\mathsf{p}}$ and $\rho_{\mathsf{p}} < \rho_{\mathsf{q}}$ from Claim I, it must be the case that $\alpha \cdot s_{pre} \xrightarrow{l} s_{post} \leq \rho_{\mathsf{q}}$ and $\mathsf{q}\triangleright\mathsf{p}!m'$ is in $w\Downarrow_{\Sigma_{\mathsf{q}}}$. From **??**, it follows that $\mathcal{V}(w\Downarrow_{\mathsf{q}\triangleright\mathsf{p}!\_}) = \mathcal{V}(w\Downarrow_{\mathsf{p}\triangleleft\mathsf{q}?\_}) \cdot m' \cdot u'$ and $\xi(\mathsf{q},\mathsf{p}) = m' \cdot u'$, i.e. m' is at the head of $\xi(\mathsf{q},\mathsf{p})$. We find a contradiction to the assumption that $m' \neq m$.

Case: $\mathsf{r} \neq \mathsf{q}$.

We discharge this case by showing a contradiction to RC. First, we establish the existence of a transition $s_1 \xrightarrow{\mathsf{q}\to\mathsf{p}:m} s_2 \in T$ such that $s_1 \neq s_{pre}$ and s_1 is reachable by p on $\mathtt{split}^{-1}(w\Downarrow_{\Sigma_{\mathsf{p}}})$. By the assumption that wx is a trace of $\{\!|T_{\mathsf{p}}|\!\}_{\mathsf{p}\in\mathcal{P}}$, it follows that $wx\Downarrow_{\Sigma_{\mathsf{p}}}$ is a prefix of $\mathcal{L}(T_{\mathsf{p}})$. By the canonicity of $\{\!|T_{\mathsf{p}}|\!\}_{\mathsf{p}\in\mathcal{P}}$, it holds that $\mathrm{pref}(\mathcal{L}(T_{\mathsf{p}})) \subseteq \mathrm{pref}(\mathcal{L}(\mathcal{S})\Downarrow_{\Sigma_{\mathsf{p}}})$, and thus $wx\Downarrow_{\Sigma_{\mathsf{p}}} \in \mathrm{pref}(\mathcal{L}(\mathcal{S})\Downarrow_{\Sigma_{\mathsf{p}}})$. Thus, there exists a maximal run ρ' in $\mathcal{S}$ such that $wx\Downarrow_{\Sigma_{\mathsf{p}}} \leq \mathtt{split}(\mathtt{trace}(\rho'))\Downarrow_{\Sigma_{\mathsf{p}}}$ and $s_1 \xrightarrow{\mathsf{q}\to\mathsf{p}:m} s_2 \in \rho'$. Because $\mathcal{S}$ is sender-driven, there does not exist a state $s \in S$ with two outgoing transition labels with different senders. Therefore, $s_1 \neq s_{pre}$.

By the fact that $\alpha \cdot s_{pre} \xrightarrow{l} s_{post} \cdot \beta$ is the unique splitting of ρ for p matching w, it holds that s_{pre} is also reachable by p on $\mathtt{split}^{-1}(w\Downarrow_{\Sigma_{\mathsf{p}}})$.

We instantiate RC with $s_1 \xrightarrow{\mathsf{q}\to\mathsf{p}:m} s_2$, $s_{pre} \xrightarrow{\mathsf{r}\to\mathsf{p}:m} s_{post}$ and $\mathtt{split}^{-1}(w\Downarrow_{\Sigma_{\mathsf{p}}})$ to obtain:

$$\neg(\exists v \in \mathrm{pref}(\mathcal{L}_{s_{post}}).\ v\Downarrow_{\Sigma_{\mathsf{p}}} = \varepsilon \wedge \mathcal{V}(v\Downarrow_{\mathsf{q}\triangleright\mathsf{p}!\_}) = \mathcal{V}(v\Downarrow_{\mathsf{p}\triangleleft\mathsf{q}?\_}) \cdot m) \ .$$

We show, on the contrary, that

$$\exists v \in \mathrm{pref}(\mathcal{L}_{s_{post}}).\ v\Downarrow_{\Sigma_{\mathsf{p}}} = \varepsilon \wedge \mathcal{V}(v\Downarrow_{\mathsf{q}\triangleright\mathsf{p}!\_}) = \mathcal{V}(v\Downarrow_{\mathsf{p}\triangleleft\mathsf{q}?\_}) \cdot m \ .$$

It is clear that $s_{post} \cdot \beta$ is a maximal run in $\mathcal{S}_{s_{post}}$. By Lemma 4.15, to show that a witness $v \in \mathrm{pref}(\mathcal{L}_{s_{post}})$, it suffices to show that v is channel-compliant and furthermore, that for all participants $\mathsf{s} \in \mathcal{P}$, $v\Downarrow_{\Sigma_{\mathsf{s}}} \leq \mathtt{split}(\mathtt{trace}(s_{post} \cdot \beta))\Downarrow_{\Sigma_{\mathsf{s}}}$.

Recall that w is a trace of $\{\!|T_{\mathsf{p}}|\!\}_{\mathsf{p}\in\mathcal{P}}$ and is thus channel-compliant. Intuitively, we obtain a witness for v by deleting from w symbols that belong to $\mathtt{split}(\mathtt{trace}(\alpha \cdot s_{pre} \xrightarrow{l} s_{post}))$. Formally, let v be initialized to w and let $l_1 \ldots l_n = \mathtt{trace}(\alpha \cdot s_{pre} \xrightarrow{l} s_{post})$. For each $i \in \{1,\ldots,n\}$, let $l_i \coloneqq \mathsf{p}_i \to \mathsf{q}_i : m_i$. We check whether $\mathsf{p}_i \triangleright \mathsf{q}_i!m_i \leq w\Downarrow_{\Sigma_{\mathsf{p}_i}}$, and if so, we delete the symbol $\mathsf{p}_i \triangleright \mathsf{q}_i!m_i$ from w. We then check whether $\mathsf{q}_i \triangleleft \mathsf{p}_i?m_i \leq w\Downarrow_{\Sigma_{\mathsf{q}_i}}$, and again delete the symbol if so. Note that

due to the channel-compliancy of v, either both symbols are deleted, or only the send action is deleted. We argue that the inductive invariant of channel-compliancy is satisfied: if a matching pair of send and receive actions are found in v and deleted, each of $\mathcal{V}(v{\Downarrow_{\mathsf{q}_i \triangleleft \mathsf{p}_i ?\_}})$ and $\mathcal{V}(v{\Downarrow_{\mathsf{p}_i \triangleright \mathsf{q}_i !\_}})$ lose their head message, and $\mathcal{V}(v{\Downarrow_{\mathsf{q}_i \triangleleft \mathsf{p}_i ?\_}}) \leq \mathcal{V}(v{\Downarrow_{\mathsf{p}_i \triangleright \mathsf{q}_i !\_}})$ continues to hold; if only the send action is found and deleted, then it must be the case that $\mathcal{V}(v{\Downarrow_{\mathsf{q}_i \triangleleft \mathsf{p}_i ?\_}}) = \varepsilon$ and the invariant is trivially re-established. Thus, we establish that upon termination, v is channel-compliant. Furthermore, it holds that $s_{post} \cdot \beta \in I^{\mathcal{S}_{s_{post}}}(v)$.

Recall that $\mathcal{V}(w{\Downarrow_{\mathsf{q} \triangleright \mathsf{p} !\_}}) = \mathcal{V}(w{\Downarrow_{\mathsf{p} \triangleleft \mathsf{q} ?\_}}) \cdot m$. It remains to show that $\mathcal{V}(v{\Downarrow_{\mathsf{q} \triangleright \mathsf{p} !\_}}) = \mathcal{V}(v{\Downarrow_{\mathsf{p} \triangleleft \mathsf{q} ?\_}}) \cdot m$. This holds from the fact that $\alpha \cdot s_{pre} = \rho_{\mathsf{p}} < \rho_{\mathsf{q}}$, which means that any labels of the form $\mathsf{q} \to \mathsf{p} : \text{-}$ in $l_1 \dots l_n$ must find and delete a matching pair of send and receive actions in v, thus preserving the above equality.

□

Lemma 4.14 (Send events preserve run prefixes). *Let $\mathcal{S}$ be a protocol satisfying* CC *and $\{\!\{T_{\mathsf{p}}\}\!\}_{\mathsf{p} \in \mathcal{P}}$ be a canonical communicating LTS for $\mathcal{S}$. Let wx be a trace of $\{\!\{T_{\mathsf{p}}\}\!\}_{\mathsf{p} \in \mathcal{P}}$ such that $x \in \Sigma_{\mathsf{p},!}$ for some $\mathsf{p} \in \mathcal{P}$. Let ρ be a run in $I(w)$, and $\alpha \cdot s_{pre} \xrightarrow{l} s_{post} \cdot \beta$ be the unique splitting of ρ for p with respect to w. Then, there exists a run ρ' in $I(wx)$ such that $\alpha \cdot s_{pre} \leq \rho'$.*

Proof. Let $x = \mathsf{p} \triangleright \mathsf{q}!m$. We prove the claim by induction on the length of w.

Base Case. $w = \varepsilon$. By definition, $I(\varepsilon)$ contains all maximal runs in $\mathcal{S}$. Then, $\mathtt{split}(\mathtt{trace}(\alpha \cdot s_{pre})){\Downarrow_{\Sigma_{\mathsf{p}}}} = \varepsilon$ and it holds that $s_0 \overset{\varepsilon}{\underset{\mathsf{p}}{\Rightarrow}}{}^{*} s_{pre}$. We argue that there exists $s_1 \in S$ such that $s_0 \overset{\varepsilon}{\underset{\mathsf{p}}{\Rightarrow}}{}^{*} s_1$. From the canonicity of $\{\!\{T_{\mathsf{p}}\}\!\}_{\mathsf{p} \in \mathcal{P}}$ and the fact that $x \in \mathrm{pref}(\mathcal{L}(T_{\mathsf{p}}))$, it follows that $x \in \mathrm{pref}(\mathcal{L}(\mathcal{S}){\Downarrow_{\Sigma_{\mathsf{p}}}})$. Thus, there exists $w \in \mathcal{L}(\mathcal{S})$ such that $x \leq w{\Downarrow_{\Sigma_{\mathsf{p}}}}$, and consequently there exists a run ρ' such that $x \leq \mathtt{split}(\mathtt{trace}(\rho')){\Downarrow_{\Sigma_{\mathsf{p}}}}$. The unique splitting of ρ' for p with respect to ε gives us a candidate for s_1. By Definition 4.1, there exists a s_2 such that $s_1 \overset{l}{\underset{\mathsf{p}}{\Rightarrow}}{}^{*} s_2$. By the assumption that every run in $\mathcal{S}$ extends to a maximal run, there exists a maximal run in $I(x)$.

Induction Step. Let wx be an extension of w by $x \in \Sigma_{\mathsf{p},!}$. To re-establish the induction hypothesis, we need to show the existence of a run $\bar{\rho}$ in $I(wx)$ such that $\alpha \cdot s_{pre} \leq \bar{\rho}$. Since p is the active participant in x, it holds for any $\mathsf{r} \neq \mathsf{p}$ that $\mathrm{R}_{\mathsf{r}}^{\mathcal{S}}(w) = \mathrm{R}_{\mathsf{r}}^{\mathcal{S}}(wx)$. Therefore, to prove the existential claim, it suffices to construct a run $\bar{\rho}$ that satisfies:

(1) $\bar{\rho} \in \mathrm{R}_{\mathsf{p}}^{\mathcal{S}}(wx)$,
(2) $\bar{\rho} \in I(w)$, and
(3) $\alpha \cdot s_{pre} \leq \bar{\rho}$.

In the case that $l{\Downarrow_{\Sigma_{\mathsf{p}}}} = x$, we are done: Property 3 and 2 hold by construction, and Property 1 holds by the definition of possible run sets.

In the case that $l{\Downarrow_{\Sigma_{\mathsf{p}}}} \neq x$, we show the existence of a different continuation such that the resulting run satisfies all three conditions.

First, we establish that p is the sender in l. By definition of unique splitting, we know that p is active in l. Assume towards a contradiction that p is the receiver in l. Then, l is of the form $\mathsf{q} \to \mathsf{p} : m$. Because $\alpha \cdot s_{pre} \xrightarrow{\mathsf{q} \to \mathsf{p} : m} s_{post} \cdot \beta$ is a maximal run in $\mathcal{S}$, we have that $(w \cdot \mathsf{p} \triangleleft \mathsf{q}?m){\Downarrow_{\Sigma_{\mathsf{p}}}} \in \mathrm{pref}(\mathcal{L}(\mathcal{S}){\Downarrow_{\Sigma_{\mathsf{p}}}})$. By the canonicity of $\{\!\{T_{\mathsf{p}}\}\!\}_{\mathsf{p} \in \mathcal{P}}$, it holds that $\mathrm{pref}(\mathcal{L}(\mathcal{S}){\Downarrow_{\Sigma_{\mathsf{p}}}}) \subseteq \mathrm{pref}(\mathcal{L}(T_{\mathsf{p}}))$, and therefore $(w \cdot \mathsf{p} \triangleleft \mathsf{q}?m){\Downarrow_{\Sigma_{\mathsf{p}}}} \in \mathrm{pref}(\mathcal{L}(T_{\mathsf{p}}))$. By assumption that wx is a trace of $\{\!\{T_{\mathsf{p}}\}\!\}_{\mathsf{p} \in \mathcal{P}}$, it holds that $(wx){\Downarrow_{\Sigma_{\mathsf{p}}}} \in \mathrm{pref}(\mathcal{L}(T_{\mathsf{p}}))$. From the fact that $\mathsf{p} \triangleleft \mathsf{q}?m \in \Sigma_{\mathsf{p},?}$ and $x \in \Sigma_{\mathsf{p},!}$, we find a contradiction to Lemma 4.7. Therefore, l must be of the form $\mathsf{p} \to \mathsf{q}' : m'$, with $\mathsf{q}' \neq \mathsf{q}$ or $m' \neq m$.

By assumption that wx is a trace of $\{\!|T_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}}$, it holds that $wx{\Downarrow_{\Sigma_\mathsf{p}}} \in \mathrm{pref}(\mathcal{L}(T_\mathsf{p}))$. By the canonicity of $\{\!|T_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}}$ (Definition 4.6(ii)), we have $\mathrm{pref}(\mathcal{L}(T_\mathsf{p})) \subseteq \mathrm{pref}((\mathcal{L}(\mathcal{S}){\Downarrow_{\Sigma_\mathsf{p}}}))$ and hence, $wx{\Downarrow_{\Sigma_\mathsf{p}}} \in \mathrm{pref}(\mathcal{L}(\mathcal{S}){\Downarrow_{\Sigma_\mathsf{p}}})$. Thus, there exists $v \in \mathcal{L}(\mathcal{S})$ such that $wx{\Downarrow_{\Sigma_\mathsf{p}}} \leq v{\Downarrow_{\Sigma_\mathsf{p}}}$, and consequently there exists a run ρ' such that $wx{\Downarrow_{\Sigma_\mathsf{p}}} \leq \mathtt{split}(\mathtt{trace}(\rho')){\Downarrow_{\Sigma_\mathsf{p}}}$. The unique splitting of ρ' for p with respect to w gives us a transition $s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:m} s_2 \in T$.

If $s_1 = s_{pre}$, then $\alpha \cdot s_{pre} \xrightarrow{\mathsf{p}\to\mathsf{q}:m} s_2$ is a run in $\mathcal{S}$. Otherwise, we instantiate SC (Definition 4.1) with $s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:m} s_2$, s_{pre} and the witness $w{\Downarrow_{\Sigma_\mathsf{p}}}$. Then, there exists s' such that $s_{pre} \xRightarrow[\mathsf{p}]{\mathsf{p}\to\mathsf{q}:m}{}^{*} s'$. We argue that, in fact, $s_{pre} \xrightarrow{\mathsf{p}\to\mathsf{q}:m} s' \in T$. This follows from the fact established above that p is the sender in l, and that $s_{pre} \xrightarrow{l} s_{post} \in T$. By the assumption that $\mathcal{S}$ is sender driven, there does not exist a state with outgoing transitions that do not share a sender. Therefore, $\alpha \cdot s_{pre} \xrightarrow{\mathsf{p}\to\mathsf{q}:m} s'$ is a run in $\mathcal{S}$.

Either way, we have found a run that thus far satisfies Property 1 and 3 regardless of its choice of maximal suffix. Let $\alpha \cdot s_{pre} \xrightarrow{\mathsf{p}\to\mathsf{q}:m} \bar{s'}$ be a run in $\mathcal{S}$. Then, for all choices of $\bar{\beta}$ such that $\alpha \cdot s_{pre} \xrightarrow{\mathsf{p}\to\mathsf{q}:m} \bar{s'} \cdot \bar{\beta}$ is a maximal run, both $wx{\Downarrow_{\Sigma_\mathsf{p}}} \leq \mathtt{split}(\mathtt{trace}(\alpha \cdot s_{pre} \xrightarrow{\mathsf{p}\to\mathsf{q}:m} \bar{s'}))$ and $\alpha \cdot s_{pre} \leq \alpha \cdot s_{pre} \xrightarrow{\mathsf{p}\to\mathsf{q}:m} \bar{s'} \cdot \bar{\beta}$ hold.

Property 2, however, requires that the projection of w onto each participant is consistent with $\bar{\rho}$, and this cannot be ensured by the prefix alone.

We construct the remainder of $\bar{\rho}$ by picking an arbitrary maximal suffix to form a candidate run, and iteratively performing suffix replacements on the candidate run until it lands in $I(w)$. Let $\bar{\beta}$ be a run suffix such that $\alpha \cdot s_{pre} \xrightarrow{\mathsf{p}\to\mathsf{q}:m} \bar{s'} \cdot \bar{\beta}$ is a maximal run in $\mathcal{S}$. Let ρ_c denote this candidate run.

If $\rho_c \in I(w)$, we are done. Otherwise, $\rho_c \notin I(w)$ and there exists a non-empty set of processes $\mathcal{Q} \subseteq \mathcal{P}$ such that for each $\mathsf{r} \in \mathcal{Q}$,

$$w{\Downarrow_{\Sigma_\mathsf{r}}} \not\leq \mathtt{split}(\mathtt{trace}(\rho_c)){\Downarrow_{\Sigma_\mathsf{r}}} \ . \tag{1}$$

By the fact that $\rho \in I(w)$,

$$w{\Downarrow_{\Sigma_\mathsf{r}}} \leq \mathtt{split}(\mathtt{trace}(\rho)){\Downarrow_{\Sigma_\mathsf{r}}} \ . \tag{2}$$

We can rewrite (1) and (2) above to make explicit their shared prefix $\alpha \cdot s_{pre}$:

$$w{\Downarrow_{\Sigma_\mathsf{r}}} \not\leq \mathtt{split}(\mathtt{trace}(\alpha \cdot s_{pre} \xrightarrow{\mathsf{p}\to\mathsf{q}:m} \bar{s'} \cdot \bar{\beta})){\Downarrow_{\Sigma_\mathsf{r}}} \tag{3}$$

$$w{\Downarrow_{\Sigma_\mathsf{r}}} \leq \mathtt{split}(\mathtt{trace}(\alpha \cdot s_{pre} \xrightarrow{\mathsf{p}\to\mathsf{q}':m'} s_{post} \cdot \beta)){\Downarrow_{\Sigma_\mathsf{r}}} \ . \tag{4}$$

We can further rewrite (3) and (4) to make explicit their point of disagreement:

$$w{\Downarrow_{\Sigma_\mathsf{r}}} \not\leq (\mathtt{split}(\mathtt{trace}(\alpha \cdot s_{pre}))).\ \mathsf{p} \triangleright \mathsf{q}!m.\ \mathsf{q} \triangleleft \mathsf{p}?m.\ \mathtt{split}(\mathtt{trace}(\bar{\beta}))){\Downarrow_{\Sigma_\mathsf{r}}} \tag{5}$$

$$w{\Downarrow_{\Sigma_\mathsf{r}}} \leq (\mathtt{split}(\mathtt{trace}(\alpha \cdot s_{pre}))).\ \mathsf{p} \triangleright \mathsf{q}'!m'.\ \mathsf{q}' \triangleleft \mathsf{p}?m'.\ \mathtt{split}(\mathtt{trace}(\beta))){\Downarrow_{\Sigma_\mathsf{r}}} \tag{6}$$

It is clear that in order for both 5 and 6 to hold, it must be the case that $\mathtt{split}(\mathtt{trace}(\alpha{\cdot}s_{pre})){\Downarrow_{\Sigma_\mathsf{r}}} < w{\Downarrow_{\Sigma_\mathsf{r}}}$.

We formalize the point of disagreement between $w{\Downarrow_{\Sigma_\mathsf{r}}}$ and ρ_c using an index i_r representing the position of the first disagreeing symbol in $\mathtt{trace}(\rho_c)$:

$$i_\mathsf{r} \coloneqq \max\{i \mid \mathtt{split}(\mathtt{trace}(\rho_c)[0..i-1]){\Downarrow_{\Sigma_\mathsf{r}}} \leq w{\Downarrow_{\Sigma_\mathsf{r}}}\} \ .$$

By the maximality of i_r, it holds that r is the active participant in $\mathtt{trace}(\rho_c)[i_\mathsf{r}]$. By the fact that $\mathtt{split}(\mathtt{trace}(\alpha \cdot s_{pre})){\Downarrow_{\Sigma_\mathsf{r}}} < w{\Downarrow_{\Sigma_\mathsf{r}}}$ we know that

$$i_\mathsf{r} > |\mathtt{trace}(\alpha \cdot s_{pre})| \ .$$

We identify the participant in Q with the *earliest disagreement* in $\texttt{split}(\texttt{trace}(\rho_c))$: let $\bar{\mathsf{r}}$ be the participant in Q with the smallest $i_{\bar{\mathsf{r}}}$. If two participants that share the same smallest index, then by the fact that both participants are active in $\texttt{trace}(\rho_c)[i_{\bar{\mathsf{r}}}]$, it must be the case that one is the sender and one is the receiver: we pick the sender to be $\bar{\mathsf{r}}$. Let $y_{\bar{\mathsf{r}}}$ denote $\texttt{split}(\texttt{trace}(\rho_c[i_{\bar{\mathsf{r}}}]))\Downarrow_{\Sigma_{\bar{\mathsf{r}}}}$.

Claim I. $y_{\bar{\mathsf{r}}}$ is a send event.

Assume by contradiction that $y_{\bar{\mathsf{r}}}$ is a receive event. We identify the symbol in w that disagrees with $y_{\bar{\mathsf{r}}}$: let w' be the largest prefix of w such that $w'\Downarrow_{\Sigma_{\bar{\mathsf{r}}}} \leq \texttt{split}(\texttt{trace}(\rho_c))\Downarrow_{\Sigma_{\bar{\mathsf{r}}}}$. By definition, $w'\Downarrow_{\Sigma_{\bar{\mathsf{r}}}} = \texttt{split}(\texttt{trace}(\rho_c)[0..i_{\bar{\mathsf{r}}}-1])\Downarrow_{\Sigma_{\bar{\mathsf{r}}}}$. Let z be the next symbol following w' in w; then $w'z \leq w$ and $z \in \Sigma_{\bar{\mathsf{r}}}$ with $z \neq y_{\bar{\mathsf{r}}}$. Furthermore, by No Mixed Choice (4.7) we have that $z \in \Sigma_{\bar{\mathsf{r}},?}$.

By assumption, $w'z \not\leq \texttt{split}(\texttt{trace}(\rho_c)[0..i_{\bar{\mathsf{r}}}])$. Therefore, any run with a trace that begins with $\rho_c[0..i_{\bar{\mathsf{r}}}]$ cannot be contained in $\mathrm{R}^{\mathcal{S}}_{\bar{\mathsf{r}}}(w'z)$, or consequently in $I(w'z)$. We show however, that $I(w'z)$ must contain some runs that begin with $\rho_c[0..i_{\bar{\mathsf{r}}}]$. From Lemma 4.13 for traces w' and $w'z$, we obtain that $I(w') = I(w'z)$. Therefore, it suffices to show that $I(w')$ contains runs that begin with $\rho_c[0..i_{\bar{\mathsf{r}}}]$.

Claim II. $\forall w'' \leq w'.\, I(w'')$ contains runs that begin with $\rho_c[0..i_{\bar{\mathsf{r}}}]$.

We prove the claim via induction on w'.

The base case is trivial from the fact that $I(\varepsilon)$ contains all maximal runs.

For the inductive step, let $w''y \leq w'$.

In the case that $y \in \Sigma_?$, we know $I(w''y) = I(w'')$ from Lemma 4.13 and the witness from $I(w'')$ can be reused.

In the case that $y \in \Sigma_!$, let s be the active participant of y and let ρ' be a run in $I(w'')$ beginning with $\rho_c[0..i_{\bar{\mathsf{r}}}]$ given by the inner induction hypothesis. Let $\alpha' \cdot s_3 \xrightarrow{l'} s_4 \cdot \beta'$ be the unique splitting of ρ' for s with respect to w''. If $\texttt{split}(l')\Downarrow_{\Sigma_{\mathsf{s}}} = y$, then ρ' can be used as the witness. Otherwise, $\texttt{split}(l')\Downarrow_{\Sigma_{\mathsf{s}}} \neq y$, and $\rho' \notin \mathrm{R}^{\mathcal{S}}_{\mathsf{s}}(w''y)$.

The outer induction hypothesis holds for all prefixes of w: we instantiate it with w'' and y to obtain:

$$\exists\, \rho'' \in I(w''y).\, \alpha' \cdot s_3 \leq \rho'' \quad .$$

Let i_{s} be defined as before; it follows that $\rho'[i_{\mathsf{s}}] = s_3$. It must be the case that $i_{\mathsf{s}} > i_{\bar{\mathsf{r}}}$: if $i_{\mathsf{s}} \leq i_{\bar{\mathsf{r}}}$, because ρ_c and ρ' share a prefix $\rho_c[0..i_{\bar{\mathsf{r}}}]$ and $w''y \leq w$, s would be the earliest disagreeing participant instead of $\bar{\mathsf{r}}$.

Because $i_{\mathsf{s}} > i_{\bar{\mathsf{r}}}$, $\rho_c[0..i_{\bar{\mathsf{r}}}] = \rho'[0..i_{\bar{\mathsf{r}}}] \leq \rho'[0..i_{\mathsf{s}}]$. Because $\rho'[0..i_{\mathsf{s}}] = \alpha' \cdot s_3 \leq \rho''$, it follows from prefix transitivity that $\rho_c[0..i_{\bar{\mathsf{r}}}] \leq \rho''$, thus re-establishing the induction hypothesis for $w''y$ with ρ'' as a witness run that begins with $\rho_c[0..i_{\bar{\mathsf{r}}}]$.

This concludes our proof that $I(w')$ contains runs that begin with $\rho_c[0..i_{\bar{\mathsf{r}}}]$, and in turn our proof by contradiction that $y_{\bar{\mathsf{r}}}$ must be a send event.

Having established that $l_{i_{\bar{\mathsf{r}}}}$ is a send event for $\bar{\mathsf{r}}$, we can now reason from the canonicity of $\{\!\{T_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ and SC and conclude that there exists an outgoing transition from $\rho_c[i_{\bar{\mathsf{r}}}]$ and a maximal suffix such that the resulting run no longer disagrees with $w\Downarrow_{\Sigma_{\bar{\mathsf{r}}}}$. The reasoning is identical to that which is used to construct our candidate run ρ_c, and is thus omitted. We update our candidate run ρ_c with the correct transition label and maximal suffix, update the set of states $Q \in \mathcal{P}$ to the new set of participants that disagree with the new candidate run, and repeat the construction above on the new candidate run until Q is empty.

Termination is guaranteed in at most $|w|$ steps by the fact that the number of symbols in w that agree with the candidate run up to $i_{\bar{\mathsf{r}}}$ must increase.

Upon termination, the resulting ρ_c serves as our witness for $\bar{\rho}$ and $\bar{\rho}$ thus satisfies the final remaining property 3: $\bar{\rho} \in I(w)$. This concludes our proof by induction of the prefix-preservation of send transitions. □

Lemma 4.18 (Completeness). *Let $\mathcal{S}$ be a protocol. If $\mathcal{S}$ is implementable, then $\mathcal{S}$ satisfies* CC.

Proof. Let communicating LTS $\{\!|B_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}}$ implement $\mathcal{S}$. Specifically, we contradict protocol fidelity, and show that $\mathcal{L}(\mathcal{S}) \neq \mathcal{L}(\{\!|B_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}})$ by constructing a witness v_0 satisfying:

(a) v_0 is a trace of $\{\!|B_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}}$, and
(b) $I(v_0) = \emptyset$.

The reasoning for the sufficiency of the above two conditions is as follows. To prove the inequality of the two languages, it suffices to prove the inequality of their respective prefix sets, i.e.

$$\mathrm{pref}(\mathcal{L}(\mathcal{S})) \neq \mathrm{pref}(\mathcal{L}(\{\!|B_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}})) \ .$$

Specifically, we show the existence of a $v \in \Sigma^*$ such that

$$v \in \{u \mid u \leq w \wedge w \in \mathcal{L}(\{\!|B_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}})\} \wedge$$
$$v \notin \{u \mid u \leq w \wedge w \in \mathcal{L}(\mathcal{S})\} \ .$$

Because $\{\!|B_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}}$ is deadlock-free by assumption, every trace can be extended to a maximal trace. Therefore, every trace $v \in \Sigma^*$ of $\{\!|B_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}}$ is a member of the prefix set of $\{\!|B_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}}$, i.e.

$$\exists\, (\vec{s}, \xi).\ (\vec{s}_0, \xi_0) \xrightarrow{v}{}^* (\vec{s}, \xi) \implies v \in \{u \mid u \leq w \wedge w \in \mathcal{L}(\{\!|B_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}})\} \ .$$

For any $w \in \mathcal{L}(\mathcal{S})$, it holds that $I(w) \neq \emptyset$. Because $I(\text{-})$ is monotonically decreasing, if $I(w)$ is non-empty then for any $v \leq w$, $I(v)$ is non-empty. By the following, to show that a word v is not a member of the prefix set of $\mathcal{L}(\mathcal{S})$ it suffices to show that $I(v)$ is empty:

$$\forall v \in \Sigma^*.\ I(v) = \emptyset \implies \forall w.\ v \leq w \implies w \notin \mathcal{L}(\mathcal{S}) \ .$$

Send Coherence. Assume that SC does not hold for some transition $s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:m} s_2 \in T$. The negation of SC says that there exists a simultaneously reachable state with no post-state reachable on $\mathsf{p} \to \mathsf{q} : m$. Formally, let $s \in S$ be a state with $s \neq s_1$ and $u \in \Sigma^*_\mathsf{p}$ be a word such that $s_0 \overset{u}{\underset{\mathsf{p}}{\Longrightarrow}}{}^* s_1, s$. Then, there does not exist $s' \in S$ such that $s \overset{\mathsf{p}\to\mathsf{q}:m}{\underset{\mathsf{p}}{\Longrightarrow}}{}^* s'$.

Because $s_0 \overset{u}{\underset{\mathsf{p}}{\Longrightarrow}}{}^* s$, there exists a run $\alpha \cdot s$ such that $\mathtt{split}(\mathtt{trace}(\alpha \cdot s))\Downarrow_{\Sigma_\mathsf{p}} = u$.

Let $\bar{w}$ be $\mathtt{split}(\mathtt{trace}(\alpha \cdot s))$. Let $\bar{w} \cdot \mathsf{p} \triangleright \mathsf{q}!m$ be our witness v_0; we show that v_0 satisfies (a) and (b).

Because $\{\!|B_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}}$ implements $\mathcal{S}$, $\bar{w}$ is a trace of $\{\!|B_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}}$ and there exists a configuration $(\vec{t}, \xi)$ of $\{\!|B_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}}$ such that $(\vec{t}_0, \xi_0) \xrightarrow{\bar{w}}{}^* (\vec{t}, \xi)$. Because $s_0 \overset{u}{\underset{\mathsf{p}}{\Longrightarrow}}{}^* s_1$, there again exists a run $\alpha_1 \cdot s_1$ such that $\mathtt{split}(\mathtt{trace}(\alpha_1 \cdot s_1))\Downarrow_{\Sigma_\mathsf{p}} = u$. Thus, $\mathtt{split}(\mathtt{trace}(\alpha_1 \cdot s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:m} s_2))$ is a prefix of $\mathcal{L}(\mathcal{S})$ and consequently, $\mathtt{split}(\mathtt{trace}(\alpha_1 \cdot s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:m} s_2))\Downarrow_{\Sigma_\mathsf{p}}$ is a prefix of $\mathcal{L}(B_\mathsf{p})$. In other words, $u \cdot \mathsf{p} \triangleright \mathsf{q}!m$ is a prefix of $\mathcal{L}(B_\mathsf{p})$. Because B_p is deterministic, there exists an outgoing transition from $\vec{s}_\mathsf{p}$ labeled with $\mathsf{p} \triangleright \mathsf{q}!m$. Because send transitions are always enabled in a communicating LTS, $\bar{w} \cdot \mathsf{p} \triangleright \mathsf{q}!m$ is a trace of $\{\!|B_\mathsf{p}|\!\}_{\mathsf{p}\in\mathcal{P}}$. Thus, (a) is established for v_0.

It remains to show that v_0 satisfies (b), namely $I(\bar{w} \cdot \mathsf{p} \triangleright \mathsf{q}!m) = \emptyset$.

Claim. All runs in $I(\bar{w})$ begin with $\alpha \cdot s$.

Proof of Claim. This claim follows from the fact that $\mathcal{S}$ is deterministic and sender-driven. Assume by contradiction that $\rho' \in I(\bar{w})$ and ρ' does not begin with $\alpha \cdot s$. Because $\alpha \cdot s \neq \rho'$, and $\mathcal{S}$ is

deterministic, $\mathtt{trace}(\alpha \cdot s) \neq \mathtt{trace}(\rho')$. Let $l = \mathtt{trace}(\alpha \cdot s)$ and let $l' = \mathtt{trace}(\rho')$. Moreover, let $\bar{l}$ be the largest common prefix of l and l'. From the assumption that $\mathcal{S}$ is sender-driven, the first divergence between the traces of any two runs must correspond to a send action by some participant. Let $\mathtt{p}'$ be the sender in the first divergence between l and l'. Because $\rho' \in \mathrm{R}^{\mathcal{S}}_{\mathtt{p}'}(\bar{w})$, it holds that $\bar{w}\Downarrow_{\Sigma_{\mathtt{p}'}} \leq \mathtt{split}(\mathtt{trace}(\rho'))\Downarrow_{\Sigma_{\mathtt{p}'}}$. We can rewrite the inequality as $\mathtt{split}(l)\Downarrow_{\Sigma_{\mathtt{p}'}} \leq \mathtt{split}(l')\Downarrow_{\Sigma_{\mathtt{p}'}}$.

Because $\bar{l}$ is the largest common prefix shared by l and l', $\mathtt{split}(l)\Downarrow_{\Sigma_{\mathtt{p}'}}$ and $\mathtt{split}(l')\Downarrow_{\Sigma_{\mathtt{p}'}}$ are respectively of the form $\bar{l}\Downarrow_{\Sigma_{\mathtt{p}'}} \cdot \mathtt{p}' \triangleright \mathtt{q}_i!m'_i \cdot z'$ and $\bar{l}\Downarrow_{\Sigma_{\mathtt{p}'}} \cdot \mathtt{p}' \triangleright \mathtt{q}_j!m'_j \cdot y'$, with $\mathtt{q}_i \neq \mathtt{q}_j$ or $m'_i \neq m'_j$. From this and $\bar{l}\Downarrow_{\Sigma_{\mathtt{p}'}} \cdot \mathtt{p}' \triangleright \mathtt{q}_i!m'_i \cdot z' \leq \bar{l}\Downarrow_{\Sigma_{\mathtt{p}'}} \cdot \mathtt{p}' \triangleright \mathtt{q}_j!m'_j \cdot y'$, we arrive at a contradiction.

End Proof of Claim.

Because $I(\text{-})$ is monotonically decreasing, $I(v_0) \subseteq I(\bar{w})$. With *Claim*, every run in $I(v_0)$ begins with $\alpha \cdot s$. From the negation of SC, there does not exist $s' \in S$ such that $s \xRightarrow[\mathtt{p}]{\mathtt{p}\to\mathtt{q}:m}{}^{*} s'$, and thus there does not exist a maximal run $\bar{\rho} \in \mathcal{S}$ such that $v_0\Downarrow_{\Sigma_{\mathtt{p}}} \leq \mathtt{split}(\mathtt{trace}(\bar{\rho}))\Downarrow_{\Sigma_{\mathtt{p}}}$.

Therefore, $\mathrm{R}^{\mathcal{S}}_{\mathtt{p}}(\bar{w} \cdot \mathtt{p} \triangleright \mathtt{q}!m) = \emptyset$, and $I(\bar{w} \cdot \mathtt{p} \triangleright \mathtt{q}!m) = \emptyset$ follows.

This concludes our proof by contradiction for the necessity of SC.

Receive Coherence. Assume that RC does not hold for a pair of transitions $s_1 \xrightarrow{\mathtt{p}\to\mathtt{q}:m} s_2, s \xrightarrow{\mathtt{r}\to\mathtt{q}:m} s' \in T$. Then, $s \neq s_1$, $\mathtt{r} \neq \mathtt{p}$ and let $u \in \Sigma^*_{\mathtt{q}}$ be a word such that $s_0 \xRightarrow[\mathtt{q}]{u}{}^{*} s_1, s$. Furthermore there exists $w \in \mathrm{pref}(\mathcal{L}(\mathcal{S}_{s'}))$ with $w\Downarrow_{\Sigma_{\mathtt{q}}} = \varepsilon \wedge \mathcal{V}(w\Downarrow_{\mathtt{p}\triangleright\mathtt{q}!\_}) = \mathcal{V}(w\Downarrow_{\mathtt{q}\triangleleft\mathtt{p}?\_}) \cdot m$.

Because $s_0 \xRightarrow[\mathtt{q}]{u}{}^{*} s_1, s$ and $s \xrightarrow{\mathtt{r}\to\mathtt{q}:m} s'$, there exists a run $\alpha \cdot s \xrightarrow{\mathtt{r}\to\mathtt{q}:m} s'$ such that $\mathtt{split}(\mathtt{trace}(\alpha \cdot s))\Downarrow_{\Sigma_{\mathtt{q}}} = u$.

Let $\mathtt{split}(\mathtt{trace}(\alpha \cdot s)) \cdot \mathtt{r} \triangleright \mathtt{q}!m \cdot w \cdot \mathtt{q} \triangleleft \mathtt{p}?m$ be our witness v_0; we show that v_0 satisfies (a) and (b).

First, we show that v_0 is a trace of $\{\!|B_{\mathtt{p}}|\!\}_{\mathtt{p}\in\mathcal{P}}$. We reason about each extension of v_0 in turn, starting with $\mathtt{split}(\mathtt{trace}(\alpha \cdot s))$. It is clear that $\mathtt{split}(\mathtt{trace}(\alpha \cdot s))$ is a trace of $\{\!|B_{\mathtt{p}}|\!\}_{\mathtt{p}\in\mathcal{P}}$: this follows immediately from the assumption that $\{\!|B_{\mathtt{p}}|\!\}_{\mathtt{p}\in\mathcal{P}}$ implements $\mathcal{S}$. Let $(\vec{s}, \xi)$ be the $\{\!|B_{\mathtt{p}}|\!\}_{\mathtt{p}\in\mathcal{P}}$ configuration reached on $\mathtt{split}(\mathtt{trace}(\alpha \cdot s))$:

$$(\vec{s_0}, \xi_0) \xrightarrow{\mathtt{split}(\mathtt{trace}(\alpha\cdot s))}{}^{*} (\vec{s}, \xi)$$

Next, we reason about the extension $\mathtt{r} \triangleright \mathtt{q}!m \cdot w$ together. We first establish that $\mathtt{r} \triangleright \mathtt{q}!m \cdot w \in \mathrm{pref}(\mathcal{L}(\mathcal{S}_s))$. Because $w \in \mathrm{pref}(\mathcal{L}(\mathcal{S}_{s'}))$, there exists a maximal run $s' \cdot \beta$ such that $s' \cdot \beta \in I(w)$. Observe that $s \xrightarrow{\mathtt{r}\to\mathtt{q}:m} s' \cdot \beta \in I(\mathtt{r} \triangleright \mathtt{q}!m \cdot w)$ and that $\mathtt{r} \triangleright \mathtt{q}!m \cdot w$ remains channel-compliant due to the assumption that $w\Downarrow_{\Sigma_{\mathtt{q}}} = \varepsilon$. Thus, by Lemma 4.15 it holds that $\mathtt{r} \triangleright \mathtt{q}!m \cdot w \in \mathrm{pref}(\mathcal{L}(\mathcal{S}_s))$. Therefore, $\mathtt{split}(\mathtt{trace}(\alpha \cdot s)) \cdot \mathtt{r} \triangleright \mathtt{q}!m \cdot w \in \mathrm{pref}(\mathcal{L}(\mathcal{S}))$, and by the assumption that $\{\!|B_{\mathtt{p}}|\!\}_{\mathtt{p}\in\mathcal{P}}$ implements $\mathcal{S}$, $\mathtt{split}(\mathtt{trace}(\alpha \cdot s)) \cdot \mathtt{r} \triangleright \mathtt{q}!m \cdot w$ is a trace of $\{\!|B_{\mathtt{p}}|\!\}_{\mathtt{p}\in\mathcal{P}}$:

$$(\vec{s_0}, \xi_0) \xrightarrow{\mathtt{split}(\mathtt{trace}(\alpha\cdot s))}{}^{*} (\vec{s}, \xi) \xrightarrow{\mathtt{r}\triangleright\mathtt{q}!m\cdot w} (\vec{s}', \xi')$$

Finally, we reason about the extension $\mathtt{q} \triangleleft \mathtt{p}?m$. We show that there exists a $\{\!|B_{\mathtt{p}}|\!\}_{\mathtt{p}\in\mathcal{P}}$ configuration $(\vec{s}'', \xi'')$ such that $(\vec{s}', \xi') \xrightarrow{\mathtt{q}\triangleleft\mathtt{p}?m} (\vec{s}'', \xi'')$. To do so, we need to show that

(1) there exists an outgoing transition labeled with $\mathtt{q} \triangleleft \mathtt{p}?m$ from $\vec{s}'_{\mathtt{q}}$, and
(2) $\xi'(\mathtt{p}, \mathtt{q}) = m \cdot u'$, with $u' \in \mathcal{V}^*$.

We know that $s_0 \xRightarrow[\mathtt{q}]{u}{}^{*} s_1$ and $s_1 \xrightarrow{\mathtt{p}\to\mathtt{q}:m} s_2$, so there exists a run $\alpha_1 \cdot s_2$ such that $\mathtt{split}(\mathtt{trace}(\alpha_1 \cdot s_2))\Downarrow_{\Sigma_{\mathtt{q}}} = u \cdot \mathtt{q} \triangleleft \mathtt{p}?m$. Because $\mathtt{split}(\mathtt{trace}(\alpha_1 \cdot s_2))\Downarrow_{\Sigma_{\mathtt{q}}} \in \mathrm{pref}(\mathcal{L}(\mathcal{S}))\Downarrow_{\Sigma_{\mathtt{q}}}$ and $\{\!|B_{\mathtt{p}}|\!\}_{\mathtt{p}\in\mathcal{P}}$ implements $\mathcal{S}$, it follows that $u \cdot \mathtt{q} \triangleleft \mathtt{p}?m \in \mathrm{pref}(\mathcal{L}(B_{\mathtt{q}}))$. Let $t \in Q_{\mathtt{q}}$ be the state reached on u in $B_{\mathtt{q}}$. The state t

is unique since B_{q} is deterministic. Because $u \cdot \mathsf{q} \triangleleft \mathsf{p}?m$ is a prefix in B_{q}, there exists a transition $t \xrightarrow{\mathsf{q} \triangleleft \mathsf{p}?m} t_1 \in \delta_{\mathsf{q}}$. It holds that $(\mathtt{split}(\mathtt{trace}(\alpha \cdot s)) \cdot \mathsf{r} \triangleright \mathsf{q}!m \cdot w)\Downarrow_{\Sigma_{\mathsf{q}}} = u$, so it follows that $\vec{s'}_{\mathsf{q}} = t$ and there exists an outgoing transition from $\vec{s'}_{\mathsf{q}}$ labeled with $\mathsf{q} \triangleleft \mathsf{p}?m$. This establishes (1).

(2) is established from the fact that send actions are immediately followed by their matching receive action in $\mathtt{split}(\mathtt{trace}(\alpha \cdot s))$, and therefore all channels in ξ are empty, including $\xi(\mathsf{p}, \mathsf{q})$. Because $\mathsf{r} \triangleright \mathsf{q}!m$ does not concern $\xi(\mathsf{p}, \mathsf{q})$, m remains the first unmatched send action from p to q in $\mathtt{split}(\mathtt{trace}(\alpha \cdot s)) \cdot \mathsf{r} \triangleright \mathsf{q}!m \cdot w$, and thus m is at the head of channel $\xi'(\mathsf{p}, \mathsf{q})$:

$$(\vec{s}_0, \xi_0) \xrightarrow{\mathtt{split}(\mathtt{trace}(\alpha \cdot s))}{}^{*} (\vec{s}, \xi) \xrightarrow{\mathsf{r} \triangleright \mathsf{q}!m \cdot w} (\vec{s'}, \xi') \xrightarrow{\mathsf{q} \triangleleft \mathsf{p}?m} (\vec{s''}, \xi'') \quad .$$

This concludes our proof of (a).

Next, we argue that $I(v_0) = \emptyset$. This claim follows trivially from the observation that every run in $I(v_0)$ must begin with $\alpha \cdot s \xrightarrow{\mathsf{r} \to \mathsf{q}:m} s'$, and therefore v_0 must satisfy $v_0\Downarrow_{\Sigma_{\mathsf{q}}} \leq u \cdot \mathsf{q} \triangleleft \mathsf{r}?m$, yet $v_0\Downarrow_{\Sigma_{\mathsf{q}}} = u \cdot \mathsf{q} \triangleleft \mathsf{p}?m$ and we find a contradiction.

No Mixed Choice. Assume that NMC does not hold for a pair of transitions $s_1 \xrightarrow{\mathsf{p} \to \mathsf{q}:m} s_2, s \xrightarrow{\mathsf{r} \to \mathsf{p}:m} s' \in T$. The negation of NMC says that s_1 and s are simultaneously reachable. Let $u \in \Sigma_{\mathsf{q}}^*$ be a word such that $s_0 \overset{u}{\underset{\mathsf{q}}{\Longrightarrow}}{}^{*} s_1, s$.

Because $s_0 \overset{u}{\underset{\mathsf{p}}{\Longrightarrow}}{}^{*} s$, there exists a run $\alpha \cdot s$ such that $\mathtt{split}(\mathtt{trace}(\alpha \cdot s))\Downarrow_{\Sigma_{\mathsf{p}}} = u$.

Let $\bar{w}$ be $\mathtt{split}(\mathtt{trace}(\alpha \cdot s))$. Let $\bar{w} \cdot \mathsf{r} \triangleright \mathsf{p}!m \cdot \mathsf{p} \triangleright \mathsf{q}!m$ be our witness v_0; we show that v_0 satisfies (a) and (b).

The reasoning is similar to that for the witness constructed for Send Coherence Condition, and is thus omitted. □