Decision Problems for Global Protocol Specifications

by

Elaine Li

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

August, 2025

Professor Thomas Wies

Acknowledgements

The five joyful years I spent as a PhD student are due in no small part to the following individuals.

I am deeply indebted to my advisor, Thomas Wies. Thomas was exceedingly generous in offering his time, confidence, and the space to let me to grow as a researcher. Beyond his advisory duties, Thomas was an exemplary scientific role model. His discerning taste, spirit of inquiry, and sense of humor have all left an indelible mark on me, and I feel incredibly fortunate to have been his advisee.

I would like to thank my committee members: Joseph Tassarotti, Aurojit Panda, Rupak Majumdar and Ankush Das. Joe guided me through my first paper reviewing experience, and spent valuable time reviewing this thesis. Panda showed me what it meant to understand an idea at the technical, scientific, and stylistic levels. Rupak's enthusiasm encouraged me to pursue new problem settings. Ankush brought a refreshing perspective that challenged my thinking about existing paradigms.

I would like to thank my coauthors for their collaboration on several papers in this thesis: Damien Zufferey, for providing me with the proper orientation to the research area, and Felix Stutz, for teaching me everything I know about high-level message sequence charts and LaTeX.

I am grateful to the fellow members of NYU ACSys and inhabitants of 60 FA's fourth floor, past and present, for fostering a lively environment conducive to both work and play.

I am blessed to be in the fluffy company of Remy, my beloved chinchilla. A special thanks to YunFan, Mark, Ryan, Arasu, Rama, and Krithika for babysitting Remy during my travels.

I would like to thank my alma mater, Yale-NUS College. 2025 marks the College's last year in operation, and I owe much to my friends, professors, and the wider community for creating an intellectual milieu that has been instrumental to my academic journey thus far.

To my parents, Jing Li and Baojie Li, for steadfastly shaping me into the person I am today. They will always be the Dr. Li's that precede me. To the rest of the Li family, thank you for being my orchard.

My final thank you goes to Noah, my fiancé, my bud, and my home. I look forward to many more five years spent together.

Abstract

Concurrency is ubiquitous in modern computing, message passing is a major concurrency paradigm, and communication protocols are therefore a key target for formal verification. Writing implementations for each protocol participant individually, such that their composition is free from communication errors and deadlocks, is challenging and error-prone. In response, various verification methodologies center around the construct of a global protocol. Global protocol specifications synchronously describe the message-passing behaviors of all protocol participants from a bird's-eye view, and thus rule out large classes of communication errors by construction. Global protocols are adopted in industry by the ITU standard and UML, and are widely studied in academia in the form of high-level message sequence charts, session types and choreographic programs. Application domains for this top-down verification methodology include cryptographic security, cyber-physical systems, and web services. This thesis contributes decision procedures for three problems central to global protocol verification: implementability, synthesis, and subtyping. Implementability asks whether a protocol admits a distributed implementation, synthesis in turn computes one, and subtyping asks whether an admissible implementation can be substituted in whole or part to yield fewer behaviors. This thesis additionally contributes a Rocq mechanization of a precise implementability characterization for infinite-state protocols, and the Sprout tool for automatically verifying such protocols.

Contents

List of Figures

List of Tables

1 | Introduction

Concurrency is ubiquitous in our increasingly efficient and interconnected world. From geographically distributed financial transactions, to local Internet of Things networks, to multi-core hardware within a single device, many safety and operation-critical systems run on concurrent software. Errors in concurrent software can be extremely costly, and ensuring its correctness is therefore of paramount importance. Unfortunately, concurrency bugs are notoriously subtle and difficult to detect. In contrast with sequential programs, which run on a single machine, concurrent programs run on multiple independent machines, typically distributed across space and time. This gives rise to a potentially infinite number of interleaving behaviors, only a handful of which may exhibit existing bugs.

Formal verification is the use of mathematical reasoning to prove that a program meets its intended specification. Unlike testing, which examines only a finite number of behaviors, verification can rigorously guarantee that errors are absent from a potentially infinite set of behaviors. Approaches to formal verification vary by the kinds of programs they target and the level of automation they provide. While formal verification of sequential programs enjoys both solid theoretical foundations and widespread practical adoption, the same cannot be said for concurrent programs. My thesis contributes theoretical results and practical tooling for the automated verification of concurrent programs.

Message-passing is a key paradigm of concurrent programming. The message-passing behavior of independent processes in a distributed system is governed by communication protocols, which specify how processes cooperate to achieve a common goal, such as maintaining a consistent database, performing distributed computations, or negotiating a shared purchase. Writing a correct implementation for each process individually, such that their interactions are free from errors such as deadlocks, orphan messages and unspecified receptions, is made more challenging by the presence of network asynchrony: every occurrence of two processes being able to send a message at the same time doubles the search space for potential bugs. Errors in protocol design and implementation threaten the efficiency, availability and functional correctness of the application depending on it, making communication protocols a prime target for formal verification.

One salient methodology for verifying message-passing centers on the construct of a *global protocol*. Global protocols synchronously specify message exchanges between all processes from a birds-eye view. By specifying the sending and receiving of a message as a single atomic event, global protocols rule out large classes of communication errors by construction. Moreover, global protocols enjoy simpler checking of deadlock-freedom. As a result, global protocol specifications have been adopted in industry and are widely studied in academia. Message sequence charts are a visual formalism for describing communication protocols [Mauw and Reniers 1997; Genest et al. 2003; Genest and Muscholl 2005; Gazagnaire et al. 2007; Roychoudhury et al. 2012]. Message sequence charts found early industry adoption by the ITU standard [International Telecommunication Union 2011] in 1993, was subsequently incorporated into UML [Object Management Group] in 2005, and is part of the Web Service Choreography Description Language [Web Services Choreography Working Group 2005]. Global specifications are also featured in the contemporary programming languages frameworks of multiparty session types and choreographic programming. Multiparty session types (MSTs) have been implemented in over a dozen programming languages, including Python [Demangeon et al. 2015; Neykova and Yoshida 2017; Neykova et al. 2017], Java [Hu and Yoshida 2016, 2017], C [Ng et al. 2012], Go [Lange et al. 2018; Castro-Perez et al. 2019], Scala [Castro-Perez and Yoshida 2023], Rust [Cutner et al. 2022; Lagaillardie et al. 2022], OCaml [Imai et al. 2020] and F# [Neykova et al. 2018]. Application domains for MSTs include operating systems [Fähndrich et al. 2006], high performance computing [Honda et al. 2012; Niu et al. 2016; de Muijnck-Hughes and Vanderbauwhede 2019], cyber-physical systems [Majumdar et al. 2019, 2020], and web services [Yoshida et al. 2013]. Choreographic programming frameworks have been implemented in Java [Giallorenzo et al. 2024], Haskell [Shen et al. 2023], Rust [Languages, Systems, and Data Lab, UC Santa Cruz; Kashiwa et al. 2023] and applied to distributed architecture [Palma et al. 2024], cryptographic security protocols [Gancher et al. 2023], and cyber-physical systems [Cruz-Filipe and Montesi 2016]. We refer the reader to [Yoshida 2024] and [Montesi 2023] for a comprehensive survey of MST and choreography applicability respectively.

We tour key features of the top-down verification methodology embodied by global protocols using the example of the two-bidder protocol. The two-bidder protocol specifies the behavior of two bidders, B_1 and B_2, who negotiate to split the purchase of a book from seller S. We depict the protocol as a high-level message sequence chart (HMSC) in Fig. 1.1. In the HMSC visualism, protocol participants are represented using vertical lines, and synchronous communications are represented as arrows from a sender to a receiver. The outer arrows depict control flow, and double lines depict final states.

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 case B_2 continues bidding, 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 refinements ensure that the proposed bids are strictly increasing from one round to the next, thus enforcing termination.

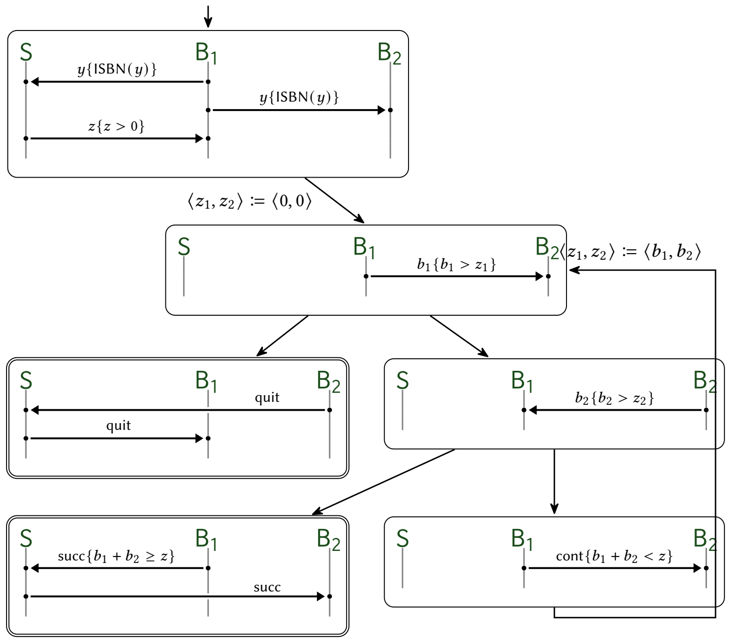


Figure 1.1: Two-bidder protocol.

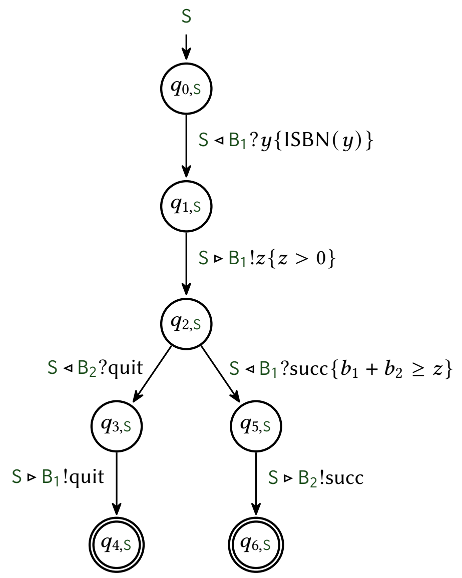
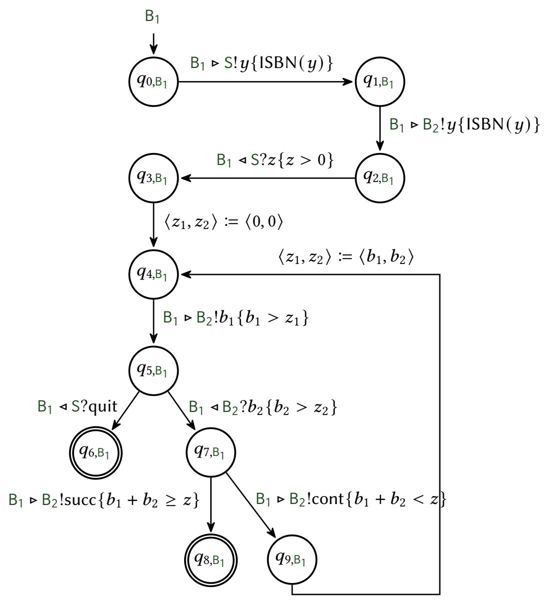


Figure 1.2: State machine for seller S for Fig. 1.1.

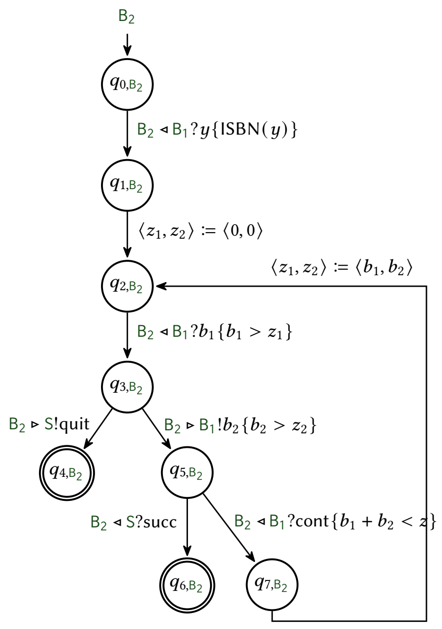
Using our global specification, we desire to *synthesize* a distributed implementation, namely a local implementation for each participant, that behaves according to the global protocol when executed concurrently on a reliable, asynchronous network in which messages can be delayed or reordered, but not dropped or duplicated. In particular, we require that the implementations never deadlock and that all participants behave consistently according to each locally chosen branch, executing send and receive actions exactly in the prescribed order. The latter property is known as *protocol fidelity*.

Figs. 1.2 to 1.4 show an admissible implementation for the two-bidder protocol in Fig. 1.1, consisting of a local implementation for each participant: S, B_1 and B_2. The transition labels specify their local behaviors: $B_1 \triangleright S!y\{\mathsf{ISBN}(y)\}$ specifies that B_1 sends a message y to S such that y satisfies $\mathsf{ISBN}(y)$, i.e. y is a valid ISBN number; $S \triangleleft B_1?y\{\mathsf{ISBN}(y)\}$ specifies that S receives y



B_1
q_{0,B_1}
$\mathsf{B}_1 \triangleright \mathsf{S}!y\{\mathsf{ISBN}(y)\}$
q_{1,B_1}
$\mathsf{B}_1 \triangleright \mathsf{B}_2!y\{\mathsf{ISBN}(y)\}$
q_{2,B_1}
$\mathsf{B}_1 \triangleleft \mathsf{S}?z\{z > 0\}$
q_{3,B_1}
$\langle z_1, z_2\rangle := \langle 0, 0\rangle$
q_{4,B_1}
$\langle z_1, z_2\rangle := \langle b_1, b_2\rangle$
$\mathsf{B}_1 \triangleright \mathsf{B}_2!b_1\{b_1 > z_1\}$
q_{5,B_1}
$\mathsf{B}_1 \triangleleft \mathsf{S}?\mathsf{quit}$
$\mathsf{B}_1 \triangleleft \mathsf{B}_2?b_2\{b_2 > z_2\}$
q_{6,B_1}
q_{7,B_1}
$\mathsf{B}_1 \triangleright \mathsf{B}_2!\mathsf{succ}\{b_1 + b_2 \geq z\}$
$\mathsf{B}_1 \triangleright \mathsf{B}_2!\mathsf{cont}\{b_1 + b_2 < z\}$
q_{8,B_1}
q_{9,B_1}

Figure 1.3: State machine for bidder B_1 for Fig. 1.1.



B_2
q_{0,B_2}
$\mathsf{B}_2 \triangleleft \mathsf{B}_1?y\{\mathsf{ISBN}(y)\}$
q_{1,B_2}
$\langle z_1, z_2\rangle := \langle 0, 0\rangle$
q_{2,B_2}
$\langle z_1, z_2\rangle := \langle b_1, b_2\rangle$
$\mathsf{B}_2 \triangleleft \mathsf{B}_1?b_1\{b_1 > z_1\}$
q_{3,B_2}
$\mathsf{B}_2 \triangleright \mathsf{S}!\mathsf{quit}$
$\mathsf{B}_2 \triangleright \mathsf{B}_1!b_2\{b_2 > z_2\}$
q_{4,B_2}
q_{5,B_2}
$\mathsf{B}_2 \triangleleft \mathsf{S}?\mathsf{succ}$
$\mathsf{B}_2 \triangleleft \mathsf{B}_1?\mathsf{cont}\{b_1 + b_2 < z\}$
q_{6,B_2}
q_{7,B_2}

Figure 1.4: State machine for bidder B_2 for Fig. 1.1.

from B_1, and can assume $\mathsf{ISBN}(y)$ holds of y.

The synthesis question first begs the *implementability* question, which asks whether an admissible implementation exists for a given global protocol. The implementability of Fig. 1.1 is witnessed by Figs. 1.2 to 1.4, which together exhibit the same behaviors as the global protocol and is never stuck. To see that the implementability problem is non-trivial, consider a variant of the protocol in Fig. 1.1 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.

The two-bidder protocol under consideration permits two paths to termination: either the bidders agree on a split of the book price and the protocol terminates successfully, or the second bidder chooses to quit early and the bidding is unsuccessful. We may wish to consider a variant in which only successful termination is possible. The problems of *subtyping and refinement* ask whether such a subset of the original protocol's behaviors remains implementable, and whether a given implementation suffices to implement it.

This thesis studies decision problems central to the top-down verification methodology of global protocols. Implementability, also known as realizability, asks whether a global protocol admits a distributed implementation that exhibits exactly the set of specified behaviors and is deadlock-free. Synthesis asks to compute such an admissible implementation. Subtyping asks whether all or part of an admissible distributed implementation can be replaced to yield a subset of the global protocol's specified behaviors. Protocols are non-implementable when they require local processes to act on information not observable to them, such as choices made by other processes. Synthesized implementations must preserve all global behaviors, without introducing new behaviors. Prior solutions to implementability and synthesis are imprecise [Honda et al. 2008; Coppo et al. 2015; Toninho and Yoshida 2017; Scalas et al. 2017], often conservatively rejecting protocols that are in fact implementable. New frameworks extending existing protocol fragments with additional features, such as data refinements [Zhou et al. 2020] and crash-stop failures [Brun and Dardha 2023], inherit the same sources of incompleteness from prior theory. More worryingly, some type systems have been shown to be unsound, typing programs that exhibit communication errors or deadlock [Scalas and Yoshida 2019]. The implementability and synthesis problems are only thoroughly understood for restricted protocol fragments, often with strong assumptions on finiteness and communication topology that limit their expressivity [Alur and Yannakakis 1999; Muscholl and Peled 1999; Morin 2002; Lohrey 2003; Genest et al. 2006b]. These theoretical limitations undermine the trustworthiness and applicability of top-down protocol verification frameworks.

This thesis contributes sound and complete characterizations for the aforementioned problems that improve prior work along the dimensions of expressivity, precision and optimality. Part I presents theoretical results, Part II presents mechanization and tool support for the theoretical results, and together the two parts contain results from the following papers:

- Elaine Li, Felix Stutz, Thomas Wies, and Damien Zufferey. Complete multiparty session type projection with automata. In Constantin Enea and Akash Lal, editors, *Computer*

Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2-23, Proceedings, Part III, volume 13966 of *Lecture Notes in Computer Science*, pages 350-373. Springer, 2023a. doi: 10.1007/978-3-031-37709_17. URL: https://doi.org/10.1007/978-3-031-37709-9_17

- Elaine Li, Felix Stutz and Thomas Wies. Deciding subtyping for asynchronous multiparty sessions. In Stephanie Weirich, editor, *Programming Languages and Systems - 33rd European Symposium on Programming, ESOP 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part I*, volume 14576 of *Lecture Notes in Computer Science*, pages 176-205. Springer, 2024. doi: 10.1007/978-3-031-57262-3_8. URL: https://doi.org/10.1007/978-3-031-57262-3_8

- Elaine Li, Felix Stutz, Thomas Wies, and Damien Zufferey. Characterizing implementability of global protocols with infinite states and data. *Proc. ACM Program. Lang.*, 9(OOPSLA1):1434-1463, 2025b. doi: 10.1145/3720493. URL: https://doi.org/10.1145/3720493

- Elaine Li, Felix Stutz, Thomas Wies, and Damien Zufferey. SPROUT: A verifier for symbolic multiparty protocols. In Ruzica Piskac and Zvonimir Rakamarić, editors, *Computer Aided Verification - 37th International Conference, CAV 2025, Zagreb, Croatia, July 23-25, 2025, Proceedings, Part III*, volume 15933 of *Lecture Notes in Computer Science*, pages 304-317. Springer, 2025a. doi: 10.1007/978-3-031-98682-6_16. URL: https://doi.org/10.1007/978-3-031-98682-6_16

- Elaine Li and Thomas Wies. Certified implementability of global multiparty protocols. To appear in *16th International Conference on Interactive Theorem Proving, ITP 2025, September 28-October 1, 2025, Reykjavík, Iceland.*

- Elaine Li and Thomas Wies. Implementability of global distributed protocols modulo network architectures. Under submission.

The thesis author is primary contributor and lead author of all aforementioned papers with the exception of [Li et al. 2023a], in which they are co-first author. Only the parts of [Li et al. 2023a] contributed by the thesis author are included in this thesis.

Contributions. The contributions of this thesis are summarized below.

- A precise characterization of implementability of global communicating labeled transition systems (GCLTS): a semantically-defined class of asynchronous communication protocols that subsumes most formalisms in the literature [Li et al. 2025b].
- Asymptotically optimal decision procedures and complexity analysis for implementability of finite GCLTS (co-NP complete), the syntactic fragment of multiparty session types (co-NP complete), and symbolic, finite-state GCLTS (PSPACE-complete) [Li et al. 2025b].
- A sound and relatively complete algorithm for checking implementability of symbolic, infinite-state GCLTS, and a tool implementation [Li et al. 2025b,a].
- A Rocq mechanization of the preciseness result from [Li et al. 2025b] [Li and Wies 2025b].
- A compatibility criterion that identifies asynchronous network architectures well-suited to global protocol specification methodology, and a generalization of all aforementioned results to compatible network architectures [Li and Wies 2025a].
- Sound and complete synthesis algorithms for finite protocols, and a blueprint for the general case [Li et al. 2023a].
- A precise characterization of various subtyping and refinement problems for multiparty session types, in addition to their complexity analysis [Li et al. 2024].

Funding Acknowledgement. The work in this thesis is supported by the National Science Foundation under grant agreement 2304758.

2 | Preliminaries

Words. Let Σ be an alphabet. Σ^* denotes the set of finite words over Σ, Σ^ω the set of infinite words, and Σ^∞ their union $\Sigma^* \cup \Sigma^\omega$. A word $u \in \Sigma^*$ is a *prefix* of word $v \in \Sigma^\infty$, denoted $u \leq v$, 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 $w = w_0 \ldots w_n$, 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. $w_0 \ldots w_i$.

Message Alphabets Let $\mathcal{P}$ be a possibly infinite set of participants and $\mathcal{V}$ be a possibly infinite data domain. We define the set of *synchronous events* $\Gamma_{sync} \coloneqq \{\mathsf{p} \to \mathsf{q} : m \mid \mathsf{p}, \mathsf{q} \in \mathcal{P} \text{ and } m \in \mathcal{V}\}$ where $\mathsf{p} \to \mathsf{q} : m$ denotes a message exchange of m from sender p to receiver q. For a participant $\mathsf{p} \in \mathcal{P}$, we define the alphabet $\Gamma_{\mathsf{p}} = \{\mathsf{p} \to \mathsf{q} : m \mid \mathsf{q} \in \mathcal{P},\ m \in \mathcal{V}\} \cup \{\mathsf{q} \to \mathsf{p} : m \mid \mathsf{q} \in \mathcal{P},\ m \in \mathcal{V}\}$, denoting the set of synchronous events in which p is either the sender or receiver in the message exchange. A synchronous event is split into a send and receive *asynchronous event* for the sender and receiver respectively. 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 asynchronous event $\mathsf{p} \triangleright \mathsf{q}!m$ denotes participant p sending a message m to q, and $\mathsf{p} \triangleleft \mathsf{q}?m$ denotes participant p receiving a message m from q. We write $\Sigma_{\mathsf{p}} = \Sigma_{\mathsf{p},!} \cup \Sigma_{\mathsf{p},?}$, $\Sigma_! = \bigcup_{\mathsf{p} \in \mathcal{P}} \Sigma_{\mathsf{p},!}$, and $\Sigma_? = \bigcup_{\mathsf{p} \in \mathcal{P}} \Sigma_{\mathsf{p},?}$. Finally, the set of *asynchronous* events is $\Sigma_{async} = \Sigma_! \cup \Sigma_?$.

Projections. We map synchronous words to asynchronous words using a homomorphism `split`, defined as $\mathtt{split}(\mathsf{p} \to \mathsf{q} : m) \coloneqq \mathsf{p} \triangleright \mathsf{q}!m.\, \mathsf{q} \triangleleft \mathsf{p}?m$. Because `split` is injective, there exists a unique inverse, which we denote $\mathtt{split}^{-1}$. We say that p is *active* in x if $x \in \Gamma_{sync}$ and $x \in \Gamma_{\mathsf{p}}$, or if $x \in \Sigma_{async}$ and $x \in \Sigma_{\mathsf{p}}$. For each participant $\mathsf{p} \in \mathcal{P}$, we define a homomorphism $\Downarrow_{\Gamma_{\mathsf{p}}}$, where $x\Downarrow_{\Gamma_{\mathsf{p}}} = x$ if $x \in \Gamma_{\mathsf{p}}$ and ε otherwise, and a homomorphism $\Downarrow_{\Sigma_{\mathsf{p}}}$, where $x\Downarrow_{\Sigma_{\mathsf{p}}} = x$ if $x \in \Sigma_{\mathsf{p}}$ and ε otherwise. We define a class of projections based on pattern-matching of alphabet symbols, denoted $\Downarrow_{\_}$. The result of the projection is determined by the unspecified parts of the pattern. For example, $\Downarrow_{\mathsf{p}\triangleright\_!\_}$ projects the event $\mathsf{p} \triangleright \mathsf{q}!m$ onto (q, m), and non-send events and send events that do not have p as the sender onto ε. The function $\Downarrow_{\mathsf{q}\triangleleft\mathsf{p}?\_}$ projects receive events of p from q of any message value onto the message value, and all other events to ε.

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

Our starting point for specifying global protocols is a labeled transition system over the synchronous alphabet Γ_{sync}.

Labeled Transition Systems A *labeled transition system* (LTS) is a tuple $\mathcal{S} = (S, \Gamma, T, s_0, F)$ where S is a set of states, Γ is a set of labels, T is a set of transitions from $S \times \Gamma \times S$, $F \subseteq S$ is a set of final states, and $s_0 \in S$ is the initial state. We use $p \xrightarrow{\alpha} q$ to denote the transition $(p, \alpha, q) \in T$. Runs and traces of an LTS are defined in the expected way. A run is *maximal* if it is either finite and ends in a final state, or is infinite. The language of an LTS $\mathcal{S}$, denoted $\mathcal{L}(\mathcal{S})$, is defined as the set of maximal traces. A state $s \in S$ is a *deadlock* if it is not final and has no outgoing transitions. An LTS is *deadlock-free* if no reachable state is a deadlock. An LTS is *deterministic* if for every $s \xrightarrow{x_1} s_1, s \xrightarrow{x_2} s_2 \in T$, $x_1 = x_2$ implies $s_1 = s_2$. 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).

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

Global communicating labeled transition systems. An LTS $\mathcal{S} = (S, \Gamma_{sync}, T, s_0, F)$ is a *global communicating labeled transition system* (GCLTS) if it satisfies the following conditions:

(1) *sink-finality*: for every final state $s \in F$, there does not exist $l \in \Gamma_{sync}$ 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 $l_1, l_2 \in \Gamma_{sync}$ such that $s \xrightarrow{l_i} s_i \in T$ for $i \in \{1, 2\}$, there is a participant $\mathsf{p} \in \mathcal{P}$ who is the sender for both, i.e. $\texttt{split}(l_i) \in \Sigma_{\mathsf{p},!}$ 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 [Dan et al. 2010]. We show that sink-finality is only required to ensure that the *finite* language of an implementation for global protocol $\mathcal{S}$ matches the *finite* semantics of $\mathcal{S}$. The condition can be waived if we define the semantics of our implementation model in terms of traces. We formalize this alternative semantics at the end of this section after introducing protocol implementability.

Condition (2) is a generalisation of most multiparty session types fragments, which require not only a dedicated sender but also a dedicated receiver, a condition we refer to as *directed choice*. In contrast, *mixed choice* lifts all restrictions on choice, and amounts to only requiring determinism. Lohrey [2003] showed that realizability is undecidable for high level message sequence charts satisfying determinism and Condition (3). 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.

To model real-world verification targets, we desire for our global 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. Formalisms such as choreography automata [Gheri et al. 2022], high-level message sequence charts [Mauw and Reniers 1997; Genest et al. 2003; Genest and Muscholl 2005; Gazagnaire et al. 2007; Roychoudhury et al. 2012; Alur et al. 2003; Lohrey 2003; Alur and Yannakakis 1999; Muscholl and Peled 1999; Morin 2002; Genest et al. 2006b] and session types [Honda et al. 2008; Bocchi et al. 2010, 2012; Toninho and Yoshida 2017; Zhou et al. 2020; Li et al. 2023a] correspond to syntactically-defined fragments that incorporate a selection of these features. GCLTS subsume many aforementioned fragments of asynchronous multiparty session types and choreography automata, and capture 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.

In the remainder of the thesis 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 p via domain restriction to Σ_{p}. Formally, given a protocol $\mathcal{S} = (S, \Gamma_{sync}, T, s_0, F)$, we define $\mathcal{S}_{\mathsf{p}} := (S, \Gamma_{\mathsf{p}} \uplus \{\varepsilon\}, T_{\mathsf{p}}, s_0, F)$ where $T_{\mathsf{p}} := \{s \xrightarrow{l \Downarrow_{\Gamma_{\mathsf{p}}}} s' \mid s \xrightarrow{l} s' \in T\}$ for a participant $\mathsf{p} \in \mathcal{P}$.

Next, we introduce our distributed implementation model.

Communicating LTS $\mathcal{T} = \{\!\{T_{\mathsf{p}}\}\!\}_{\mathsf{p} \in \mathcal{P}}$ 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 CLTS 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 CLTS $\mathcal{T}$ 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. Equivalently, a CLTS is deadlock-free if every trace can be extended to a maximal one.

Communicating state machines [Brand and Zafiropulo 1983] are a special case of CLTS where the LTS for each participant $\mathsf{p} \in \mathcal{P}$ is a deterministic finite state machine. Note that CLTS feature a peer-to-peer communication topology, and FIFO queues as its message channels. In Chapter 4, we generalize CLTS to alternative asynchronous network architectures, denoted $\mathbb{A}$, that vary their communication topology and message channel data structure, but for now we implicitly assume that $\mathbb{A}$ denotes peer-to-peer FIFO communication. Moreover, note that because CLTS describe asynchronous communication with message channels of unbounded size, they differ from Zielonka's *asynchronous automata* [Zielonka 1987], which actually describe *synchronously communicating systems* [Mukund 2002]. We refer the reader to [Diekert and Rozenberg 1995] for further details.

Executable words of a CLTS. A finite asynchronous word $w \in \Sigma^*_{async}$ is *executable* in a CLTS $\mathcal{T}$ if $w \in \mathrm{pref}(\mathcal{L}(\mathcal{T}))$. We say that $w \in \Sigma^*_{async}$ is *executable* if it is executable in some $\mathcal{T}$ and use $\mathcal{L}(\mathbb{A}) \subseteq \Sigma^*_{async}$ to denote all such words.

Global protocol semantics. We next define the asynchronous semantics of global protocol $\mathcal{S}$, denoted $\mathcal{C}^{\sim}(\mathcal{S}) \subseteq \Sigma^\infty$. The starting point for the semantics $\mathcal{C}^{\sim}(\mathcal{S})$ is the synchronous language $\mathcal{L}(\mathcal{S})$. Intuitively, synchronous words in $\mathcal{L}(\mathcal{S})$ specify the coordination of events across all protocol participants, in addition to a total order of events per participant. From $\mathcal{L}(\mathcal{S})$ we obtain a set of 1-synchronous asynchronous words through `split`, which simply splits each atomic message exchange into its send and receive counterparts, denoted `split`$(\mathcal{L}(\mathcal{S}))$. We want to include all asynchronous words that are equal to these 1-synchronous words under local projection and the given network architecture $\mathbb{A}$.

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

$$\mathcal{C}^{\sim}(\mathcal{S}) = \mathcal{C}^{\sim}_{\mathrm{fin}}(\mathcal{S}) \cup \mathcal{C}^{\sim}_{\mathrm{inf}}(\mathcal{S})$$

The finite semantics is obtained by following the above recipe, but restricting `split`$(\mathcal{L}(\mathcal{S}))$ to finite words:

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

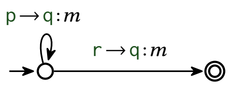
Our definition of finite words coincides with message sequence chart (MSC) semantics, which is defined order-theoretically, as the union of a total order of events for each participant, and a partial order capturing network behavior, see e.g. [Alur et al. 2003]. Unlike MSC semantics, we additionally provide a semantics for infinite words.

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

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

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

We illustrate our protocol semantics using the following example, whose semantics contains both finite and infinite words:



The synchronous runs of the protocol are either of the form $\mathsf{p} \rightarrow \mathsf{q} : m^{\infty}$, or of the form $(\mathsf{p} \rightarrow \mathsf{q} : m)^n \cdot \mathsf{r} \rightarrow \mathsf{q} : m$. The $\texttt{split}$ runs are subsequently of the form $(\mathsf{p} \triangleright \mathsf{q}!m \cdot \mathsf{q} \triangleleft \mathsf{p}?m)^{\infty}$ or $(\mathsf{p} \triangleright \mathsf{q}!m \cdot \mathsf{q} \triangleleft \mathsf{p}?m)^n \cdot \mathsf{r} \triangleright \mathsf{q}!m \cdot \mathsf{q} \triangleleft \mathsf{r}?m$. Because our infinite word semantics do not impose any fairness assumptions, the unfairly scheduled word $\mathsf{p} \triangleright \mathsf{q}!m^{\infty}$ is an infinite word of the protocol. The word $\mathsf{r} \triangleright \mathsf{q}!m \cdot \mathsf{p} \triangleright \mathsf{q}!m \cdot \mathsf{q} \triangleleft \mathsf{p}?m \cdot \mathsf{q} \triangleleft \mathsf{r}?m$ is a finite word of the protocol under a peer-to-peer FIFO network, where the network reorders the send events from p and r, but q receives in the specified protocol order, first from p and then from r.

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

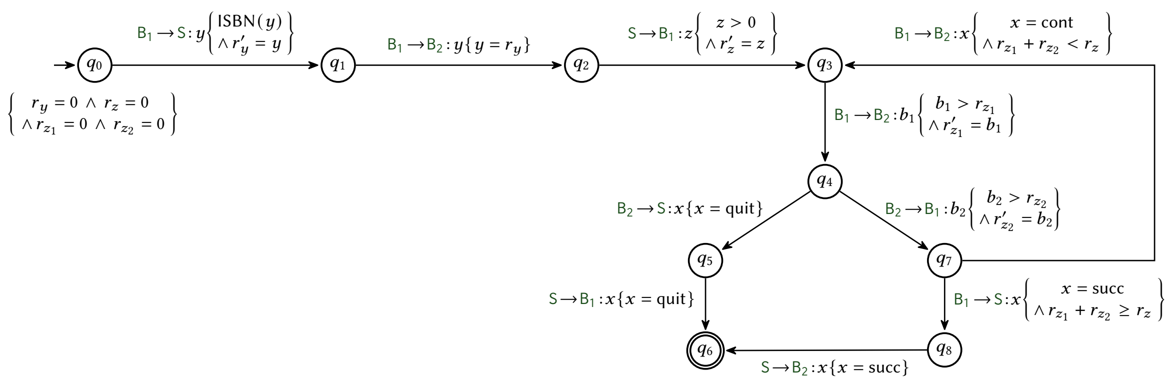

q_0
q_1
q_2
q_3
q_4
q_5
q_6
q_7
q_8
$\left\{\begin{array}{c} r_y = 0 \wedge r_z = 0 \\ \wedge r_{z_1} = 0 \wedge r_{z_2} = 0 \end{array}\right\}$
$\mathsf{B}_1 \rightarrow \mathsf{S} : y \left\{\begin{array}{c} \mathsf{ISBN}(y) \\ \wedge r'_y = y \end{array}\right\}$
$\mathsf{B}_1 \rightarrow \mathsf{B}_2 : y\{y = r_y\}$
$\mathsf{S} \rightarrow \mathsf{B}_1 : z \left\{\begin{array}{c} z > 0 \\ \wedge r'_z = z \end{array}\right\}$
$\mathsf{B}_1 \rightarrow \mathsf{B}_2 : x \left\{\begin{array}{c} x = \mathsf{cont} \\ \wedge r_{z_1} + r_{z_2} < r_z \end{array}\right\}$
$\mathsf{B}_1 \rightarrow \mathsf{B}_2 : b_1 \left\{\begin{array}{c} b_1 > r_{z_1} \\ \wedge r'_{z_1} = b_1 \end{array}\right\}$
$\mathsf{B}_2 \rightarrow \mathsf{S} : x\{x = \mathsf{quit}\}$
$\mathsf{B}_2 \rightarrow \mathsf{B}_1 : b_2 \left\{\begin{array}{c} b_2 > r_{z_2} \\ \wedge r'_{z_2} = b_2 \end{array}\right\}$
$\mathsf{S} \rightarrow \mathsf{B}_1 : x\{x = \mathsf{quit}\}$
$\mathsf{B}_1 \rightarrow \mathsf{S} : x \left\{\begin{array}{c} x = \mathsf{succ} \\ \wedge r_{z_1} + r_{z_2} \geq r_z \end{array}\right\}$
$\mathsf{S} \rightarrow \mathsf{B}_2 : x\{x = \mathsf{succ}\}$

Figure 2.1: The two-bidder protocol from Fig. 1.1 as a symbolic protocol with registers r_z, r_y, r_{z_1}, and r_{z_2}.

Finally, we define protocol implementability.

Protocol Implementability A protocol $\mathcal{S}$ is *implementable* if there exists a CLTS $\{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ satisfying the following two properties: (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 $\mathcal{S}$.

One could consider an alternative semantics for both global protocols and CLTS implementations that disposes of the notion of finite words in favor of prefixes. This alternative notion of implementability is defined as follows:

Protocol Prefix-Implementability A protocol $\mathcal{S}$ is *prefix-implementable* if there exists a CLTS $\{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ satisfying the following two properties: (i) *prefix protocol fidelity*: $\mathrm{pref}(\mathcal{L}(\{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}})) = \mathrm{pref}(C^{\sim}_{\mathbb{A}}(\mathcal{S}))$, and (ii) *deadlock freedom*: $\{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ is deadlock-free. We say that $\{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ prefix-implements $\mathcal{S}$.

In Chapter 6, we study a notion of implementability that relaxes language equality to language inclusion, and defer relevant definitions until then.

Next, we introduce our model for finitely representing infinite state protocols. We refer to these representations simply as *symbolic protocols*. Figure 2.1 shows the two-bidder protocol from Fig. 1.1 expressed as a symbolic protocol.

The formal definition of symbolic protocols is given below. In the 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 an infinite 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.

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,
- $\Delta \subseteq S \times \mathcal{P} \times X \times \mathcal{P} \times \mathcal{F} \times S$ 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' with the sent value x,
- $s_0 \in S$ is the initial control state,
- $\rho_0 \colon R \to \mathcal{V}$ is the initial register assignment, and
- $F \subseteq S$ is a set of final states.

To streamline our definition, we specify register updates and predicates describing the communicated message value altogether in one transition constraint φ. To specify register updates, for each register variable $r \in R$, we define a primed copy r' that refers to the same register in the post-state of a transition, and we define $R' = \{r' \mid r \in R\}$. We use r_1, r_2, r_3 to denote register variables, and x, y, z to denote communication variables. Thus, the free variables in φ are either variables from R describing registers in the pre-state, variables from R' describing registers in the post-state, or a communication variable x. For example, $\mathsf{p} \to \mathsf{q} : x\{even(x) \land r_1' = r_1 + 1 \land r_2' = x\}$

describes p sending q an even number x, incrementing the value of register r_1 by 1, and storing the value of x in register r_2.

We formally specify the two-bidder protocol from Fig. 1.1 as a symbolic protocol in Fig. 2.1 for demonstration purposes. Note that the transition predicate $\mathsf{ISBN}(y)$ from q_1 to q_2 is replaced with an equality. For readability and conciseness, we employ the following conventions. We treat communication variables as registers that are automatically assigned the communicated value, e.g. $\mathsf{S} \rightarrow \mathsf{B}_1 : z\{z > 0\}$ should be understood as $\mathsf{S} \rightarrow \mathsf{B}_1 : x\{x > 0 \wedge z' = x\}$ 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 $\mathsf{S} \rightarrow \mathsf{B}_2 : \mathsf{succ}\{\top\}$ instead of $\mathsf{S} \rightarrow \mathsf{B}_2 : x\{x = \mathsf{succ}\}$. We may further omit the condition $\top$, turning $\mathsf{S} \rightarrow \mathsf{B}_2 : \mathsf{succ}\{\top\}$ into $\mathsf{S} \rightarrow \mathsf{B}_2 : \mathsf{succ}$.

Symbolic protocols are specification-wise similar to symbolic register automata [D'Antoni et al. 2019], 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 concretization of a symbolic protocol below.

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 $S \times (R \rightarrow \mathcal{V})$.

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

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

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, the concrete protocol $\mathcal{S}_{\mathbb{S}}$ is a protocol over the alphabet Γ_{sync}. The language of a symbolic protocol $\mathbb{S}$ is defined as the language of its concretization $\mathcal{S}_{\mathbb{S}}$. Consequently, a symbolic protocol is implementable if its concretization is implementable.

Part I

Theory

3 | Implementability

3.1 Introduction

In this chapter, we study the implementability problem for GCLTS. GCLTS subsume many existing fragments of asynchronous multiparty session types and choreography automata. Our two-bidder example from Chapter 1 highlights 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 [Zhou et al. 2020].

- Infinite state: the protocol state contains registers that can be assigned values from an infinite domain. Registers are updated to store the last bid from each round z_1 and z_2, and to enforce that bidders make strictly increasing bids per round.

- Infinite message data: message payload values can be 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.

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. [2020] present a framework for synchronous, refined multiparty session types that soundly approximates implementability through its endpoint projection, but that may yield local specifications that are not implementable. Several works [Alur et al. 2005; Lohrey 2003; Stutz 2023; Li et al. 2023a] precisely characterize implementability for finite protocol specifications. However, the implementability check in [Alur et al. 2005; Li et al. 2023a] relies on synthesizing an implementation upfront, which is not possible for infinite-state protocols. Das and Pfenning [2020] study local session types with arithmetic refinements in a binary setting.

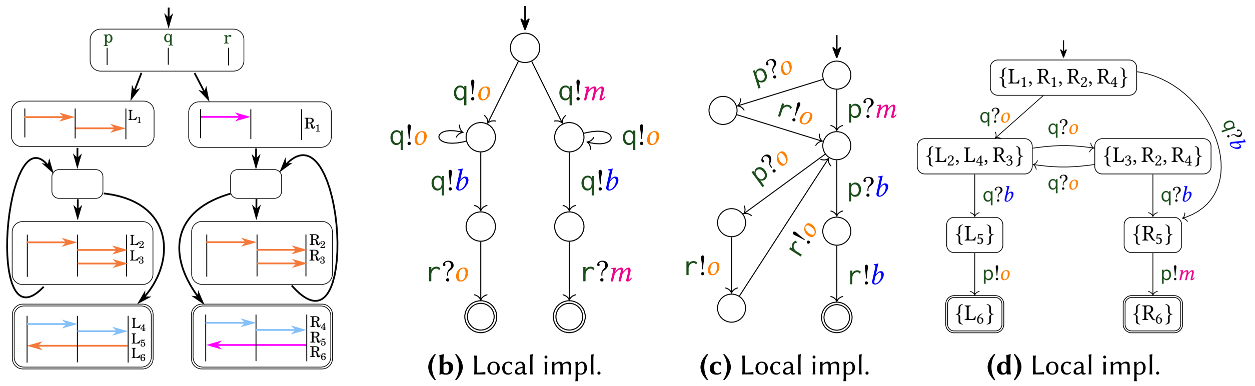
We address these challenges by decomposing the implementability problem into two steps. First, we give a precise, semantic characterization of implementability for GCLTS that we prove sound and complete once and for all. Our characterization is defined directly on the global specification, and thus forgoes 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. Specifically, we provide a sound and complete reduction to the first-order fixpoint logic μCLP [Unno et al. 2023]. 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. Our implementability characterization can therefore be checked by existing μCLP solvers. Second, we use this reduction to obtain a blueprint for solving implementability algorithmically. Our reduction yields algorithms that are sound and complete relative to an assumed oracle for solving μCLP validity, in addition to decision procedures with optimal complexity for various decidable classes.

Contributions. In summary, the contributions in this chapter 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 for explicit protocol representations that enumerate all states and transitions, the problem is co-NP-complete, and for symbolic protocol representations that encode states and transitions using predicates and variables, the problem is PSPACE-complete.
- As a corollary of the previous result, we obtain a co-NP decision procedure for implementability of global types, tightening a prior PSPACE upper bound [Li et al. 2023a,b].

The results from this chapter are published in [Li et al. 2025b]. Because GCLTS subsume multiparty session types, the implementability characterization proposed in this chapter subsumes the results from [Li et al. 2023a], which presents a sound and complete algorithm for implementability and synthesis of multiparty session types.

3.2 Overview



(a) Odd-even protocol **(b)** Local impl. for role p **(c)** Local impl. for role q **(d)** Local impl. for role r

Figure 3.1: Odd-even: An implementable but not syntactically projectable protocol and its local implementations

We begin with a discussion of the incompleteness of existing projection operators for the restricted fragment of GCLTS corresponding to multiparty session types.

Global Multiparty Session Types. Global types for MSTs 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. 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. The $\sum$ operator is omitted when $|I| = 1$, and often replaced with the infix operator $+$ for readability.

We adopt a more permissive choice construct for global types proposed in [Majumdar et al. 2021a]. In contrast to the original definition of global types introduced by Honda et al. [2008] and inherited by later works, our global types allow receivers in a choice $\mathsf{q}_i, \mathsf{q}_j$ to be distinct. We refer to this as *sender-driven choice*, and the original construct as *directed choice*.

A global type $\mathbf{G}$ can be equivalently represented using a finite state machine $\mathrm{GAut}(\mathbf{G}) = (Q_{\mathbf{G}}, \Gamma_{sync} \uplus \{\varepsilon\}, \delta_{\mathbf{G}}, q_{0,\mathbf{G}}, F_{\mathbf{G}})$ where $Q_{\mathbf{G}}$ is the set of all syntactic subterms in $\mathbf{G}$ together with the term 0, $\delta_{\mathbf{G}}$ 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 $\mu t. G'$, $q_{0,\mathbf{G}} = \mathbf{G}$ and $F_{\mathbf{G}} = \{0\}$. Each ε transition in $\mathrm{GAut}(\mathbf{G})$ is the only transition from the state it originates from, and thus can be removed to yield a protocol $\mathcal{S}_{\mathbf{G}} = (Q_{\mathbf{G}}, \Gamma_{sync}, \delta'_{\mathbf{G}}, q_{0,\mathbf{G}}, F_{\mathbf{G}})$, where $\delta'_{\mathbf{G}}$ contains only transitions labeled with $l \in \Gamma_{sync}$. It is easy to verify that $\mathcal{S}_{\mathbf{G}}$ is indeed a GCLTS.

MST frameworks typically solve synthesis and implementability simultaneously via an efficient syntactic *projection operator*. Abstractly, a projection operator is a partial map from global types to collections of implementations. A projection operator `proj` is sound when every global type $\mathbf{G}$ in its domain is implemented by `proj`$(\mathbf{G})$, and complete when every implementable global type is in its domain. In standard MST frameworks, both global protocols and distributed implementations are represented as syntactic types, and projection operators are therefore also syntactic in nature. Existing syntactic projection operators for MSTs are all incomplete or unsound with respect to implementability [Honda et al. 2008; Coppo et al. 2015; Toninho and Yoshida 2017; Scalas et al. 2017]. A key limitation of syntactic projection operators is that they can only compute local types that share a structure with the global type. However, structural similarity is not a necessary condition for admissible local types, and its enforcement can lead to incompleteness, as demonstrated by the following global type $\mathbf{G}_{oe}$:

$$+ \begin{cases} \mathsf{p} \to \mathsf{q} : \mathsf{o}.\, \mathsf{q} \to \mathsf{r} : \mathsf{o}.\, \mu t_1.\, (\mathsf{p} \to \mathsf{q} : \mathsf{o}.\, \mathsf{q} \to \mathsf{r} : \mathsf{o}.\, \mathsf{q} \to \mathsf{r} : \mathsf{o}.\, t_1 \;+\; \mathsf{p} \to \mathsf{q} : \mathsf{b}.\, \mathsf{q} \to \mathsf{r} : \mathsf{b}.\, \mathsf{r} \to \mathsf{p} : \mathsf{o}.\, 0) \\ \mathsf{p} \to \mathsf{q} : \mathsf{m}.\, \mu t_2.\, (\mathsf{p} \to \mathsf{q} : \mathsf{o}.\, \mathsf{q} \to \mathsf{r} : \mathsf{o}.\, \mathsf{q} \to \mathsf{r} : \mathsf{o}.\, t_2 \;+\; \mathsf{p} \to \mathsf{q} : \mathsf{b}.\, \mathsf{q} \to \mathsf{r} : \mathsf{b}.\, \mathsf{r} \to \mathsf{p} : \mathsf{m}.\, 0) \end{cases}$$

Fig. 3.1a visualizes $\mathbf{G}_{oe}$ as an HMSC. The top and bottom choice branches of $\mathbf{G}_{oe}$ correspond to the left and right sub-protocols, and the participants p, q and r are represented by the left, middle and right vertical lines respectively. Participant p initiates the protocol by choosing to send either o or m to q. In the left branch, q forwards the message o to r, whereas in the right branch, q does not forward the message m. Both branches then proceed identically: in a loop, p sends an o message to q, and q forwards this message twice to r. Participant p signals termination of the loop by sending b to q, which q again forwards to r. Upon receiving the termination message b, r must send a different message to p indicating its knowledge of p's initial choice: o for the left branch, and m for the right branch.

Figs. 3.1b to 3.1d depict the local implementations for participants p, q and r. Notice the structural similarity between the global protocol and the local implementations for p and q. For participant p, the reason is evident: p determines the control flow throughout the entire protocol, initially determining the choice of left or right branch, then determining the number of loop iterations before signaling termination. Participant q does not determine the control flow, but is immediately informed of p's choices when they are made. In the loop on each branch, participant q's actions are identical, and thus collapsed into a single sub-protocol. Participant r, on the other hand, neither determines the control flow nor learns of it either directly or indirectly, yet can deduce p's initial choice from the parity of the number of o messages it receives from q throughout the protocol: an odd number means p chose left, and an even number means p chose right. The resulting local implementation for r features transitions going back and forth between the two branches, reflecting r's belief update on p's choice every time it receives a new o message from q. Syntactic projection operators fail to create such transitions that are not present in the global protocol. Fundamentally, the cause of incompleteness lies in the fact that syntactic projection operators use a linear time algorithm to tackle a problem that turns out to be co-NP-complete, as we show later in this chapter.

The brittleness of syntactic projection operators has prompted the idea of abandoning global type specifications altogether in favor of model checking user-provided implementations [Scalas and Yoshida 2019; Lange and Yoshida 2019]. Correctly distinguishing implementable from non-

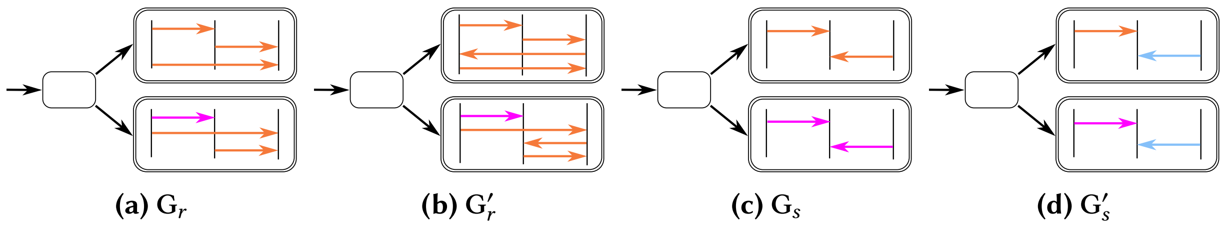


Figure 3.2: High-level message sequence charts for the global types of ??.

implementable global types is non-trivial, beyond the incompleteness of existing syntactic projection operators. Consider the following two pairs of global types, whose HMSC representations are depicted in Fig. 3.2:

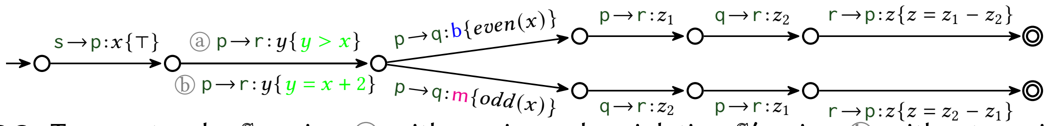
$$\mathbf{G}_r = + \begin{cases} \mathsf{p} \to \mathsf{q} : \mathsf{o}.\ \mathsf{q} \to \mathsf{r} : \mathsf{o}.\ \mathsf{p} \to \mathsf{r} : \mathsf{o}.\ 0 \\ \mathsf{p} \to \mathsf{q} : \mathsf{m}.\ \mathsf{p} \to \mathsf{r} : \mathsf{o}.\ \mathsf{q} \to \mathsf{r} : \mathsf{o}.\ 0 \end{cases} \qquad \mathbf{G}_s = + \begin{cases} \mathsf{p} \to \mathsf{q} : \mathsf{o}.\ \mathsf{r} \to \mathsf{q} : \mathsf{o}.\ 0 \\ \mathsf{p} \to \mathsf{q} : \mathsf{m}.\ \mathsf{r} \to \mathsf{q} : \mathsf{m}.\ 0 \end{cases}$$

$$\mathbf{G}'_r = + \begin{cases} \mathsf{p} \to \mathsf{q} : \mathsf{o}.\ \mathsf{q} \to \mathsf{r} : \mathsf{o}.\ \mathsf{r} \to \mathsf{p} : \mathsf{o}.\ \mathsf{p} \to \mathsf{r} : \mathsf{o}.\ 0 \\ \mathsf{p} \to \mathsf{q} : \mathsf{m}.\ \mathsf{p} \to \mathsf{r} : \mathsf{o}.\ \mathsf{r} \to \mathsf{q} : \mathsf{o}.\ \mathsf{q} \to \mathsf{r} : \mathsf{o}.\ 0 \end{cases} \qquad \mathbf{G}'_s = + \begin{cases} \mathsf{p} \to \mathsf{q} : \mathsf{o}.\ \mathsf{r} \to \mathsf{q} : \mathsf{b}.\ 0 \\ \mathsf{p} \to \mathsf{q} : \mathsf{m}.\ \mathsf{r} \to \mathsf{q} : \mathsf{b}.\ 0 \end{cases}$$

Similar to $\mathbf{G}_{oe}$, in all four examples, p chooses a branch by sending either o or m to q. The global type $\mathbf{G}_r$ is not implementable because r cannot learn which branch was chosen by p. For any local implementation of r to be able to execute both branches, it must be able to receive o from p and q in any order. Because the two send events $\mathsf{p} \triangleright \mathsf{r}!\mathsf{o}$ and $\mathsf{q} \triangleright \mathsf{r}!\mathsf{o}$ are independent of each other, they may be reordered. Consequently, any implementation of $\mathbf{G}_r$ must permit executions that are consistent with global behaviors not described by $\mathbf{G}_r$, such as $\mathsf{p} \to \mathsf{q} : \mathsf{m}.\ \mathsf{q} \to \mathsf{r} : \mathsf{o}.\ \mathsf{p} \to \mathsf{r} : \mathsf{o}$. In contrast, $\mathbf{G}'_r$ is implementable. In the top branch of $\mathbf{G}'_r$, role p can only send to r after it has received from r, which prevents the reordering of the send events $\mathsf{q} \triangleright \mathsf{r}!\mathsf{o}$ and $\mathsf{p} \triangleright \mathsf{r}!\mathsf{o}$. The bottom branch is symmetric. Hence, r learns p's choice based on which message it receives first.

For the global type $\mathbf{G}_s$, role r again cannot learn the branch chosen by p, and subsequently cannot know whether to send o or m to q, leading inevitably to deadlocking executions. In contrast, $\mathbf{G}'_s$ is again implementable because the expected behavior of r is independent of the choice by p.

In [Li et al. 2023a], we present the first sound and complete projection operator for MSTs based on a novel, automata-theoretic approach to synthesis and implementability checking. Our projection operator separates synthesis from checking implementability, and critically relies on the observation that if a global type is implementable, then its canonical implementation implements it. Thus, we reduce the implementability problem to a set of sound and complete conditions, which we call *Send Validity*, *Receive Validity* and *No Mixed Choice*, that are checked directly on the canonical implementation. Precise characterizations of implementability exist for other classes of finite state GCLTS [Li et al. 2023a; Stutz 2023; Lohrey 2003]. Unfortunately, these techniques all rely on synthesizing an implementation upfront, which is not possible for general GCLTS. The examples above show that deciding implementability in the presence of network asynchrony and non-deterministic choice already presents a challenge for protocols with finite GCLTS specifications. Both of these features are present and moreover interact with dependent refinement predicates to make checking implementability for general GCLTS uniquely challenging. We illustrate these challenges in a series of examples below.



$\mathsf{s}\to\mathsf{p}:x\{\top\}$
ⓐ $\mathsf{p}\to\mathsf{r}:y\{y > x\}$
ⓑ $\mathsf{p}\to\mathsf{r}:y\{y = x + 2\}$
$\mathsf{p}\to\mathsf{q}:\mathsf{b}\{even(x)\}$
$\mathsf{p}\to\mathsf{q}:\mathsf{m}\{odd(x)\}$
$\mathsf{p}\to\mathsf{r}:z_1$
$\mathsf{q}\to\mathsf{r}:z_2$
$\mathsf{r}\to\mathsf{p}:z\{z = z_1 - z_2\}$
$\mathsf{q}\to\mathsf{r}:z_2$
$\mathsf{p}\to\mathsf{r}:z_1$
$\mathsf{r}\to\mathsf{p}:z\{z = z_2 - z_1\}$

Figure 3.3: Two protocols: $\mathbb{S}_1$ using ⓐ with receive order violation $\mathbb{S}'_1$ using ⓑ without receive order violation.

Consider the examples $\mathbb{S}_1$ (using ⓐ) and $\mathbb{S}'_1$ (using ⓑ) in Fig. 3.3, which are variations of the examples for Receive Validity [Li et al. 2023a] featuring dependent predicates. A transition label $\mathsf{p}\to\mathsf{r}:y\{y > x\}$, 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 $y > x$. In $\mathbb{S}_1$, 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 Li et al. [2023a], we propose one method of protocol repair: introducing a message sent by r on each branch that creates a causal dependency between the messages from p and q, so that r can no longer receive them in either order. The incorporation of dependent refinements enables a new method of protocol repair: one that does not change the communication events among the participants. The newly repaired protocol is depicted in $\mathbb{S}_1'$, in which 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 and thus correctly follow the protocol: if y is even, r receives from p first, and if y is odd, r receives from q first.

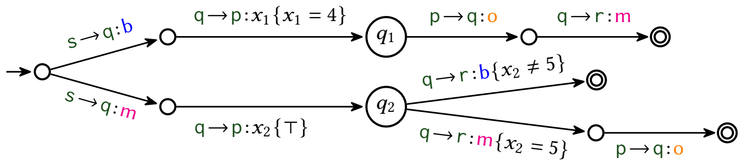


Figure 3.4: $\mathbb{S}_2$: An protocol with a send violation.

We now turn our attention to send violations. In the protocol shown in Fig. 3.4, 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 to send o to q upon receiving 4 from q, because whilst p cannot distinguish the two branches, both branches contain the transition $p \to q : o$. Upon closer inspection, the predicate guarding the transition immediately preceding $p \to q : o$ 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 $p \to q : o$, 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 Section 3.3 when we present our precise characterization of implementability. The rest of the chapter is structured as follows. Section 3.3 presents our semantic characterization of implementability for GCLTS in terms of (co)reachability, and proves that it is precise. Section 3.4 describes our sound and complete reduction from the characterization in Section 3.3 to logical formulas in μCLP [Unno et al. 2023], and additionally presents improved complexity results under certain finiteness assumptions on the GCLTS. Section 3.5 discusses related work and concludes.

3.3 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 Section 3.2 with a receiver violation, depicted in Fig. 3.3. The infinite-state LTS $\mathbb{S}_1$ contains the two concrete run prefixes depicted in Fig. 3.5, where the values of x, y are 2, 3 and 1, 3 respectively.

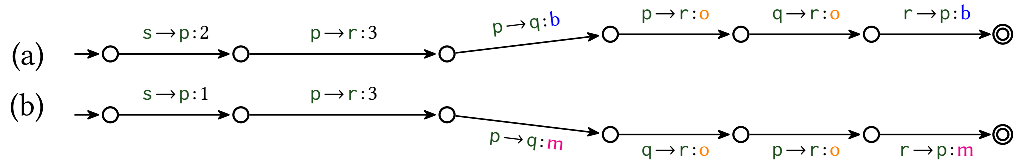


Figure 3.5: Two concrete runs of $\mathbb{S}_1$ (Fig. 3.3): (a) with $x = 2$ and $y = 3$ and (b) with $x = 1$ and $y = 3$.

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 the protocol $\mathbb{S}_2$ from Section 3.2 with a sender violation, depicted in Fig. 3.4. Again

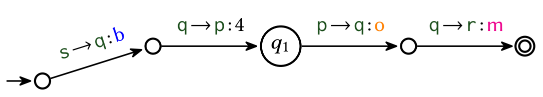


Figure 3.6: A concrete run of $\mathbb{S}_2$ (Fig. 3.4) with s choosing the top branch.

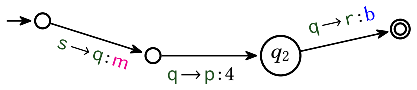


Figure 3.7: A concrete run of $\mathbb{S}_2$ (Fig. 3.4) with s choosing the bottom branch and $x_2 = 4$.

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. 3.6 and Fig. 3.7 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 information through conservative projection algorithms [Honda et al. 2008; Coppo et al. 2015; Toninho and Yoshida 2017; Scalas et al. 2017], Li et al. [2023a] 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 determinizing 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 checking can be done on the global protocol directly, without synthesizing a candidate implementation upfront. 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 §3.4 that the implementability check for global types by Li et al. [2023a] 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. 3.8 and Fig. 3.9.

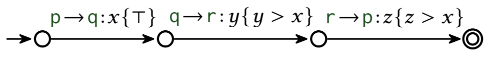


Figure 3.8: $\mathbb{S}_3$: A non-implementable protocol with dependent refinements.

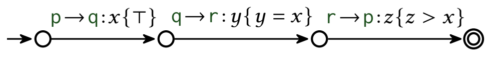


Figure 3.9: $\mathbb{S}_3'$: An implementable protocol with dependent refinements.

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. While an implementation for r could produce a subset of $\mathbb{S}_3$'s behaviors (e.g. by sending z such that $z > y$), or produce a superset of $\mathbb{S}_3$'s behaviors (e.g. by sending all values for z), no implementation can produce exactly the specified behaviors, as required by protocol fidelity. Zhou et al. [2020] address partial information of protocol variables by syntactically classifying whether a variable is known or unknown to a participant, and annotating 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 Fig. 3.10, 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. Observe that in this example,

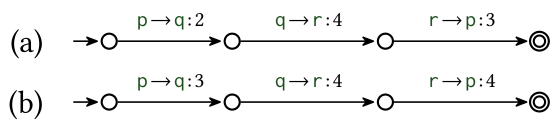


Figure 3.10: Two concrete runs of $\mathbb{S}_3$ (Fig. 3.8): (a) with $x = 2, y = 4, z = 3$ and (b) with $x = 3, y = 4, z = 4$.

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 $\mathcal{S} = (S, \Gamma_{sync}, T, s_0, F)$ 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 $w \in \Gamma_{sync}^*$, 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 $w \in \Gamma_{sync}^*$ such that $s_0 \xrightarrow{w}^* s \in T$ and $w{\Downarrow_{\Gamma_{\mathsf{p}}}} = u$, which we denote $s_0 \overset{u}{\underset{\mathsf{p}}{\Rightarrow}}^* s$. 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 $s_0 \overset{u}{\underset{\mathsf{p}}{\Rightarrow}}^* s_1, s_2$, if there exist $w_1, w_2 \in \Gamma_{sync}^*$ 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 indistinguishable if they are simultaneously reachable.

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

Definition 3.1 (Send Coherence)**.** A protocol $\mathcal{S} = (S, \Gamma_{sync}, T, s_0, F)$ satisfies Send Coherence (SC) when for every $s_1 \xrightarrow{\mathsf{p}\rightarrow\mathsf{q}:m} s_2 \in T, s_1' \in S$:

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

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 3.2 (Receive Coherence)**.** A protocol $\mathcal{S} = (S, \Gamma_{sync}, T, s_0, F)$ satisfies Receive Coherence (RC) when for every $s_1 \xrightarrow{\mathsf{p}\rightarrow\mathsf{q}:m} s_2, s_1' \xrightarrow{\mathsf{r}\rightarrow\mathsf{q}:m} s_2' \in T$:

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

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

Definition 3.3 (No Mixed Choice)**.** A protocol $\mathcal{S} = (S, \Gamma_{sync}, T, s_0, F)$ satisfies No Mixed Choice (NMC) when for every $s_1 \xrightarrow{\mathsf{p}\rightarrow\mathsf{q}:m} s_2, s_1' \xrightarrow{\mathsf{r}\rightarrow\mathsf{p}:m} s_2' \in T$: $(\exists u \in \Gamma_{\mathsf{p}}^*.\ s_0 \overset{u}{\underset{\mathsf{p}}{\Rightarrow}}{}^* s_1, s_1') \implies \bot$.

Our semantic characterization of protocol implementability is the conjunction of the above three conditions. In contrast to the syntactic analysis in [Zhou et al. 2020], our semantic approach is sound and complete. In contrast to the sound and complete approach in [Li et al. 2023a], our implementability conditions do not rely on synthesizing an implementation upfront.

Definition 3.4 (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 3.5. *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 3.5. We refer the reader to Chapter A for the complete proofs.

3.3.1 Soundness

Soundness requires us to show that if a protocol satisfies *CC*, then it is implementable. We begin by echoing the observation made in several prior works [Alur et al. 2003; Stutz 2023; Li et al. 2023a] that 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 3.6 (Canonical implementations). We say a CLTS $\{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ is a *canonical implementation* for a protocol $\mathcal{S} = (S, \Gamma_{sync}, T, s_0, F)$ if for every $\mathsf{p} \in \mathcal{P}$, T_p satisfies:
(i) $\forall w \in \Sigma_\mathsf{p}^*.\ w \in \mathcal{L}(T_\mathsf{p}) \Leftrightarrow w \in \mathcal{L}(\mathcal{S})\Downarrow_{\Sigma_\mathsf{p}}$, and (ii) $\mathrm{pref}(\mathcal{L}(T_\mathsf{p})) = \mathrm{pref}(\mathcal{L}(\mathcal{S})\Downarrow_{\Sigma_\mathsf{p}})$.

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

Lemma 3.7 (No Mixed Choice). *Let $\mathcal{S}$ be a protocol satisfying NMC (Definition 3.3) 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 (Chapter 2), soundness 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 3.8 (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), on the other hand, 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 $w \preceq^{\omega}_{\sim} w'$ 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 3.9 (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 $w \in \Sigma^*_{async}$, 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 show the inductive invariant that every trace in the canonical implementation of a protocol satisfying *CC* satisfies *intersection set non-emptiness.*

Definition 3.10 (LTS intersection sets)**.** Let $\mathcal{S}$ be an LTS. Let p be a participant and $w \in \Sigma^*_{async}$ be a word. We define the set of possible runs $\mathrm{R}^{\mathcal{S}}_{\mathsf{p}}(w)$ as all maximal runs of $\mathcal{S}$ that are consistent with p's local view of w:

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

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

Definition 3.11 (Unique splitting of a possible run)**.** Let $\mathcal{S}$ be an LTS, p a participant, and $w \in \Sigma^*_{async}$ a word. Let ρ be a run in $\mathrm{R}^{\mathcal{S}}_{\mathsf{p}}(w)$. We define the longest prefix of ρ matching w:

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

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 $\rho = s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:\mathsf{m}} s_2 \xrightarrow{\mathsf{r}\to\mathsf{q}:\mathsf{b}_1} s_3 \xrightarrow{\mathsf{r}\to\mathsf{q}:\mathsf{b}_2} s_4 \xrightarrow{\mathsf{q}\to\mathsf{p}:\mathsf{o}} s_5$ for p matching $w = \mathsf{r} \triangleright \mathsf{q}!\mathsf{b}_1 . \mathsf{p} \triangleright \mathsf{q}!\mathsf{m}$ is $\alpha \cdot s_3 \xrightarrow{\mathsf{r}\to\mathsf{q}:\mathsf{b}_2} s_4 \cdot \beta$, where $\alpha = s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:\mathsf{m}} s_2 \xrightarrow{\mathsf{r}\to\mathsf{q}:\mathsf{b}_1} s_3$ and $\beta = s_4 \xrightarrow{\mathsf{q}\to\mathsf{p}:\mathsf{o}} s_5$.

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 7.1 and Lemma 3.13 provide a characterization for each case respectively.

Lemma 3.12 (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 implementation for $\mathcal{S}$. Then, for every trace $w \in \Sigma_{async}^*$ of $\{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}}$, it holds that $I(w) \neq \emptyset$.*

Lemma 3.13 (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 implementation 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 3.14 (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 implementation 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 protocols that satisfy *CC* and intersection set non-emptiness have deadlock-free canonical implementations. The proof follows immediately from the following lemma and the fact that CLTS are deterministic, and is thus omitted.

Lemma 3.15 (Canonical implementation deadlock freedom). *Let $\mathcal{S} = (S, \Gamma_{sync}, T, s_0, F)$ 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 $w \in \Sigma_{async}^*$, 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 3.16 (Soundness of *CC*)**.** *Let $\mathcal{S}$ be a protocol. If $\mathcal{S}$ satisfies* CC*, then $\mathcal{S}$ is implementable.*

3.3.2 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*. Thus, we assume the negation of either SC, RC or NMC. 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 we obtain 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 using each witness in turn, we 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 3.17 (Completeness)**.** *Let $\mathcal{S}$ be a protocol. If $\mathcal{S}$ is implementable, then $\mathcal{S}$ satisfies* CC*.*

An immediate consequence of the soundness and completeness of *CC* is the following fact about the special case of binary protocols, when $|\mathcal{P}| = 2$:

Lemma 3.18. *Every binary protocol is implementable.*

In the binary case, participant-based reachability is equivalent to standard reachability, because both participants are involved in every synchronous communication. Because protocols are deterministic, there exist no two distinct states in a binary protocol that are simultaneously reachable for either participant, and thus *CC* holds vacuously.

3.4 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.

3.4.1 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 (Chapter 2). 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 substituting the new register and communication variables into the transition constraints and initial register assignment accordingly; 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.

Theorem 3.19 (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 3.20 (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,

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

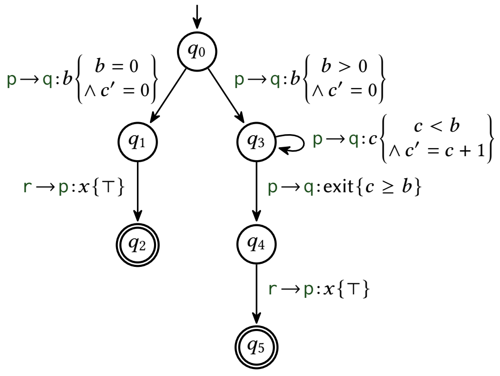
The second top-level disjunct in the definition after the base case handles the cases where $\mathbb{S}_1$ and $\mathbb{S}_2$ synchronize on a common action involving p. The remaining two disjuncts correspond to the cases where either $\mathbb{S}_1$ or $\mathbb{S}_2$ follows an ε transition.

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

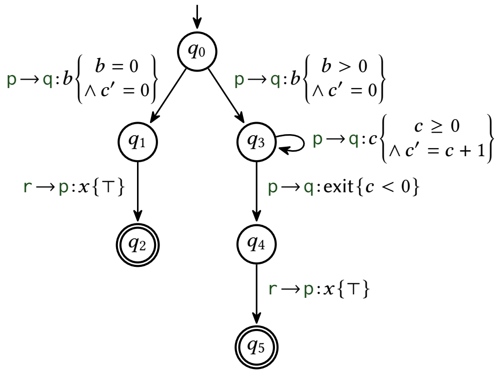
$$\mathsf{unreach}^{\varepsilon}_{\mathsf{p},\mathsf{q}}(s, \boldsymbol{r}, x) :=_\nu \quad (\bigwedge_{(s, \mathsf{p} \to \mathsf{q}: y\{\varphi\}, s') \in \Delta} \neg\varphi[x/y]\,) \wedge (\bigwedge_{\substack{(s, \mathsf{r} \to \mathsf{t}: y\{\varphi\}, s') \in \Delta \\ \mathsf{p} \neq \mathsf{r}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

The first conjunct checks that whenever p reaches a state with an outgoing send transition



q_0
$p\to q{:}b\{b = 0 \wedge c' = 0\}$
$p\to q{:}b\{b > 0 \wedge c' = 0\}$
q_1
q_3
$p\to q{:}c\{c < b \wedge c' = c + 1\}$
$r\to p{:}x\{\top\}$
$p\to q{:}\mathsf{exit}\{c \geq b\}$
q_2
q_4
$r\to p{:}x\{\top\}$
q_5

Figure 3.11: Example where states q_1 and q_3 satisfy Send Coherence for r.



q_0
$p\to q{:}b\{b = 0 \wedge c' = 0\}$
$p\to q{:}b\{b > 0 \wedge c' = 0\}$
q_1
q_3
$p\to q{:}c\{c \geq 0 \wedge c' = c + 1\}$
$r\to p{:}x\{\top\}$
$p\to q{:}\mathsf{exit}\{c < 0\}$
q_2
q_4
$r\to p{:}x\{\top\}$
q_5

Figure 3.12: Example where states q_1 and q_3 violate Send Coherence for r.

to q, it cannot send the value x because the transition constraint φ is not satisfied. The second conjunct 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 3.22 (Symbolic Send Coherence)**.** A symbolic protocol $\mathbb{S}$ satisfies Symbolic Send Coherence when for each transition $s_1 \xrightarrow{p\to q:x_1\{\varphi_1\}} s_1' \in \Delta_1$ 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. 3.11 and 3.12. 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. These states are reachable from q_3 if and only if the loop in q_3 terminates, which it does in Fig. 3.11 but not in Fig. 3.12.

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 cannot also receive x as the first message from p in the second state, in which it can also receive from a different participant r, 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 their 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 3.23 (Symbolic Availability)**.**

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

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 3.24 (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 3.25 (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 $s_2 \xrightarrow{\mathsf{r}\to\mathsf{p}:x_2\{\varphi_2\}} s_2' \in \Delta_2$:

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

We conclude this section with a discussion of how to check GCLTS assumptions, namely sink finality, sender-driven choice, and deadlock-freedom, on a symbolic protocol. Sink finality can be checked directly by examining the syntax of the symbolic protocol. Sender-driven choice without determinism can likewise be checked directly on the states of the symbolic protocol. Determinism and deadlock freedom are undecidable in general but can both be reduced to reachability. Thus, both our Symbolic Coherence Conditions and GCLTS assumptions can be discharged using off-the-shelf μCLP solvers. We leave such an implementation to future work.

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

3.4.2 Finite Protocols

We first consider finite protocols. Let $\mathcal{S} = (S, \Gamma_{sync}, T, s_0, F)$ be a protocol with finite S and T. Because S and T are 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

Algorithm 1 Check *CC* for finite protocols

```
▷ Let LTS $\mathcal{S} = (S, \Gamma_{sync}, T, s_0, F)$
▷ 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 $b \leftarrow b \vee \left(s \overset{\varepsilon}{\underset{\mathsf{p}}{\Rightarrow}}{}^* s_3\right)$
            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 $\top$
```

$\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(x, s, \boldsymbol{r})$, except on protocols instead of symbolic protocols.

It is easy to see that Send Coherence and No Mixed Choice can be checked in time polynomial in the size of $\mathcal{S}$. However, the inclusion of $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, s)$ as a subroutine for checking Receive Coherence yields the following complexity result.

Theorem 3.26. *Implementability of finite protocols is co-NP-complete.*

Proof. To see that implementability is in co-NP, observe that violations of Send Coherence and No Mixed Choice can be checked in NP, by guessing a participant p and a pair of states s_1, s_2 that satisfy the respective preconditions, and verifying simultaneous reachability of s_1 and s_2 for p. For Send Coherence, we guess an additional state s_3 with an outgoing transition labeled with $\mathsf{p} \to \mathsf{q} : m$, and check ε-reachability from s_1 to s_3. For Receive Coherence, $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, s_2)$ can be checked in NP by guessing a simple path in $\mathcal{S}$ from s_2 to some state s' with an outgoing transition labeled with $\mathsf{p} \to \mathsf{q} : m$. We then evaluate $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, s_2)$ 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, $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, s_2)$ is antitone in the blocked set.

We show NP-hardness of non-implementability via 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, with $1 \leq i \leq k$ and $1 \leq j \leq 3$. We construct a protocol $\mathcal{S}_\varphi$ over participants $\mathcal{P} = \{\mathsf{p}, \mathsf{q}, \mathsf{r}, \mathsf{x}_1, \overline{\mathsf{x}}_1, \ldots, \mathsf{x}_n, \overline{\mathsf{x}}_n\}$, such that φ is satisfiable iff $\mathcal{S}_\varphi$ is implementable. In particular, we ensure that $\mathcal{S}_\varphi$ is implementable iff $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, s)$ does not hold for some state s in $\mathcal{S}_\varphi$. The protocol $\mathcal{S}_\varphi$ is constructed from the following subprotocols:

1. Define a protocol $\mathcal{S}_X$ representing a truth assignment to variables x_i with states $s_1, \ldots, s_{n+1}$ as follows: for every $1 \leq i \leq n$ there are two paths of four transitions each between s_i and s_{i+1}. The paths consist of transitions labeled with $\mathsf{r} \to \mathsf{x_i} : \bot$, $\mathsf{r} \to \overline{\mathsf{x}}_\mathsf{i} : \top$, $\mathsf{r} \to \mathsf{q} : m_{x_i}$, $\mathsf{q} \to \mathsf{x_i} : m$, and $\mathsf{r} \to \overline{\mathsf{x}}_\mathsf{i} : \bot$, $\mathsf{r} \to \mathsf{x_i} : \top$, $\mathsf{r} \to \mathsf{q} : m_{\overline{x}_i}$, $\mathsf{q} \to \overline{\mathsf{x}}_\mathsf{i} : m$, respectively.

2. Define a protocol $\mathcal{S}_C$ representing the clauses C_i with states $t_1, \ldots, t_{k+1}$ as follows. For each $1 \leq i \leq k$ there are three paths of three transitions between each t_i and t_{i+1}, one for each $1 \leq j \leq 3$, labeled with $\mathsf{r} \to s : m_j$, $\mathsf{r} \to \mathsf{p} : m_r$, $s \to \mathsf{p} : m$, where $s = \mathsf{x}$ if $L_{ij} = x$ and $s = \overline{\mathsf{x}}$ if $L_{ij} = \neg x$ for $x \in \{x_1, \ldots, x_n\}$.

3. Define a protocol $\mathcal{S}_F$ with two states q'_f and q_f and a single transition from q'_f to q_f labeled with $\mathsf{p} \to \mathsf{q} : m$.

4. Define a protocol $\mathcal{S}_T$ with five states $q_1, \ldots, q_5$, and two paths $q_1 \xrightarrow{\mathsf{r}\to\mathsf{p}:m_1} q_2 \xrightarrow{\mathsf{r}\to\mathsf{q}:m} q_3$ and $q_1 \xrightarrow{\mathsf{r}\to\mathsf{p}:m_2} q_4 \xrightarrow{\mathsf{p}\to\mathsf{q}:m} q_5$.

We merge all of the above protocols to obtain $\mathcal{S}_\varphi$ by identifying the state q_3 with s_1, s_{n+1} with t_1 and t_{k+1} with q'_f. The initial state is q_1 and the final states are $\{q_5, q_f\}$.

Observe that the size of $\mathcal{S}_\varphi$ 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, and final states are the only states with no outgoing transitions, yielding sink-finality and deadlock-freedom.

We first establish that $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, q_3)$ holds in $\mathcal{S}_\varphi$ iff φ is satisfiable. Observe that the blocked set $\mathcal{B}$ computed by $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, q_3)$ along a path between s_1 and s_{n+1} contains for each variable x_i either $\mathsf{x_i}$ or $\overline{\mathsf{x}}_\mathsf{i}$. The blocked set $\mathcal{B}$ thus encodes a truth assignment $\rho_\mathcal{B}$ for the x_i's where $\rho_\mathcal{B}(x_i) = \top$ iff $\mathsf{x_i} \notin \mathcal{B}$. By construction of $\mathcal{S}_X$, 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 the paths 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 $t_{k+1} = q'_f$ adds p to $\mathcal{B}$ at t_i iff $\rho_\mathcal{B}$ does not satisfy at least one of the clauses C_i. Therefore, m is available in q_3 iff there exists a $\mathcal{B}$ such that $\rho_\mathcal{B}$ satisfies φ.

It remains to show that $\mathcal{S}_\varphi$ is non-implementable iff $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, q_3)$ holds in $\mathcal{S}_\varphi$. We argue that all participants except q have sufficient local information about the control flow of the protocol to behave accordingly. Participant r dictates the control flow at every branching point of the protocol, and thus is implementable. Participants $\mathsf{x_1}, \overline{\mathsf{x}}_\mathsf{1}, \ldots \mathsf{x_n}, \overline{\mathsf{x}}_\mathsf{n}$ learn the control flow via receiving messages from participant r, whose labels uniquely determine their next actions: receiving $\top$ means inaction, receiving $\bot$ means receive a further message from q, and receiving m means send a message encoding its own variable name to p. Participant p is likewise informed by r about the control flow, and only sends m to q upon either receiving m_2 or *top* from r. Upon receiving r's choice of disjunct for each clause, it anticipates a message from the participant encoding that disjunct.

Participant q, on the other hand, is not informed by r about r's initial choice at G_{x_1}, and can locally choose between receptions from p or r. In the case that $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, q_3)$ holds, there exists a path from $\overline{G}$ to $\mathbf{G}_\varphi$ in which p is not blocked. Thus, the message from p can be asynchronously reordered to arrive in q's channel such that both receptions are enabled, and q may violate implementability by receiving the message from p out of order. If $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, q_3)$ does not hold, only one reception is enabled, which uniquely informs q about r's choice. In the case that the reception from p is enabled, q terminates, otherwise it receives messages from r encoding participants to send further messages to, and terminates upon receiving the final message from p. Thus, $\mathcal{S}_\varphi$ is non-implementable iff q violates Receive Coherence for the transitions $q_2 \xrightarrow{\mathsf{r}\to\mathsf{q}:m} q_3$ and $q_4 \xrightarrow{\mathsf{p}\to\mathsf{q}:m} q_5$, i.e. $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, q_3)$ does not hold.

We obtain that $\mathcal{S}_\varphi$ is non-implementable iff $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, q_3)$ holds in $\mathcal{S}_\varphi$ iff φ is satisfiable. ◻

The same 3-SAT reduction can be adapted to show co-NP-completeness of implementability for global multiparty session types.

Lemma 3.27. *Implementability of global types is co-NP-complete.*

Our reduction shows that deciding the $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, s)$ predicate for global types is in co-NP. The proof of Lemma 3.27 can be found in Chapter A.

3.4.3 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 3.28. *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 [Godefroid and Yannakakis 2013]. 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 [Esparza and Nielsen 1994]. 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: $\mathbf{G}_N \coloneqq (\mathsf{r} \to \mathsf{s} : m_1\{\top\}. G_t + \mathsf{r} \to \mathsf{s} : m_2\{\top\}. \mathsf{p} \to \mathsf{q} : l\{\top\}. 0)$. 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:

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

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, thus satisfying Send Coherence trivially. It follows that Send Coherence for p holds iff M is reachable from M_0 in N. To see that Receive Coherence holds, observe that no participant receives messages from two different senders. No Mixed Choice similarly holds trivially.

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. □

3.5 Related Work

Table 3.1 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 3.1 effectively require *history-sensitivity*, meaning that a "predicate guaranteed by a [participant p] can only contain those interaction variables that [p] knows" [Bocchi et al. 2010], see also [Bocchi et al. 2012, Def. 2]. As discussed in §3.3, syntactic approaches to analyzing variable knowledge is overly conservative, and as a result no prior work

Table 3.1: Comparison of related work (in chronological order)

Paper	Communication paradigm	Branching restrictions	History sensitivity	Characterization
[Bocchi et al. 2010]	asynchronous	directed choice	required	incomplete
[Bocchi et al. 2012]	asynchronous	directed choice	required	incomplete
[Toninho and Yoshida 2017]	synchronous	directed choice	required	incomplete
[Zhou et al. 2020]	synchronous	directed choice	required	incomplete
[Gheri et al. 2022]	synchronous	well-sequencedness	required	unknown
this work	asynchronous	sender-driven choice	not required	relatively complete

can handle protocols such as the example in Fig. 3.9. In a similar vein, Zhou et al. [2020] impose the syntactic restriction that all participants in a loop must be able to update all loop registers, which rules out loops like the one in the two-bidder protocol (Fig. 1.1).

Furthermore, all prior works except for [Gheri et al. 2022] 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 shown in Fig. 2.1. Gheri et al. [2022] 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 following 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 [Gheri et al. 2022, Def. 3.2], a condition that has since been refined because it was shown to be flawed [Finkel and Lozes 2023]. Majumdar et al. [2021b] 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 [2024] conjectures that the framework in [Zhou et al. 2020] "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, several works [Dagnino et al. 2021; Castellani et al. 2022, 2024] 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 [Dagnino et al. 2021].

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 [Union 1996] and are well-studied in academia [Mauw and Reniers 1997; Genest et al. 2003; Genest and Muscholl 2005; Gazagnaire et al. 2007; Roychoudhury et al. 2012]. In the HMSC setting, implementability is called safe realizability, and is defined with respect to the implementation model of communicating finite state machines [Brand and Zafiropulo 1983]. Similar to our setting, a canonical implementation exists for any HMSC [Alur et al. 2003, 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 [Lohrey 2003]. Various fragments have since been identified in which the problem regains decidability. Lohrey [2003] showed implementability to be EXPSPACE-complete for bounded HMSCs [Alur and Yannakakis 1999; Muscholl and Peled 1999] and globally-cooperative HMSCs [Morin 2002; Genest et al. 2006b]. 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 [Lohrey 2003].

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; it was later shown in [Stutz 2023; Li et al. 2023a] that all but Gheri et al. [2022] are incomplete. Several works [Bocchi et al. 2010, 2012; Toninho and Yoshida 2017; Zhou et al. 2020] 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. 1.1). Zhou et al. [2020] 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. 3.8 and Fig. 3.9 are projectable in [Zhou et al. 2020]. 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 [D'Antoni and Veanes 2017; Shen et al. 2023] and certain classes of timed and register automata [Bertrand et al. 2015; Clemente et al. 2022] due to the existence of off-the-shelf determinization algorithms for these classes [Veanes et al. 2010; Veanes and Bjørner 2012; Bertrand et al. 2018].

Scalas and Yoshida [2019] 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 [Cruz-Filipe and Montesi 2020; Giallorenzo et al. 2021; Hirsch and Garg 2022] 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 [Hirsch and Garg 2021] 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 [Hirsch and Garg 2021] are not with respect to implementability, but with respect to the translation via endpoint projection. HasChor [Shen et al. 2023] rules out non-implementability by automatically incorporating location broadcasts when a choice is made. No formal correctness claims are made in [Shen et al. 2023]. Jongmans and van den Bos [2022] allow if- and while- statements to be annotated with a conjunction of conditional choices for each participant, which expresses decentralized decision-making in protocols. They 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. Griffith and Gunter [2013] refine binary sessions with basic data types, and shows decidability of the subtyping problem.

Gommerstadt et al. [2018] applies a similar type system for runtime monitoring of binary communication. Thiemann and Vasconcelos [2020] 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 [2020] study the undecidable problem of local type equality, and provide a sound approximate algorithm. Das et al. [Das and Pfenning 2022; Das et al. 2021] further apply binary session types with refinements to resource analysis of blockchain smart contracts and amortized cost analysis.

Actris [Hinrichsen et al. 2020] embeds binary session types into the Iris framework [Jung et al. 2018]. 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 [Hinrichsen et al. 2022]) and with linearity to prove both preservation and progress (LinearActris [Jacobs et al. 2024]). Multris [Hinrichsen et al. 2024] 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 [Scalas and Yoshida 2019] to correctness: given a collection of local types, the type system checks that they can be safely combined. Multris guarantees protocol fidelity but not progress.

4 Implementability Modulo Network Architectures

Global protocol specifications enjoy the illusion of synchrony, specifying send and receive events jointly from the perspective of an omniscient observer. Nonetheless, they are intended to specify distributed implementations, which run on an asynchronous network. The diversity of asynchronous network architectures complicates the landscape of theoretical results and verification methodologies. Different network architectures have been studied in relation to one another, see [Chevrou et al. 2016, 2019; Giusto et al. 2023], as well as in the context of different decision problems, such as synchronizability and reachability [Lohrey 2003; Finkel and Lozes 2023; Bollig et al. 2021; Delpy et al. 2024]. Because communication errors in one architecture do not necessarily arise in another, most existing results target a fixed communication architecture, and it is unclear if and why they can be generalized. In the case of HMSC and multiparty session types, peer-to-peer FIFO networks are the de facto choice of network architecture. Many theoretical results target communicating state machines [Brand and Zafiropulo 1983], which likewise features peer-to-peer FIFO channels. While this communication model is ubiquitous in practice, other network models have been studied both for their applicability and theoretical interest. For example, Erlang and Go assume a mailbox network, and many correctness proofs of classic distributed algorithms such as leader election [Gallager et al. 1983] and clock synchronization [Lamport 2019] rely on bag semantics.

In this chapter, we investigate the implementability problem for global specifications modulo the choice of the underlying (asynchronous) network architecture. To begin with, we observe that not all network architectures are suitable for the top-down design methodology of global specifications. Some well-known architectures such as FIFO mailbox networks defy common assumptions made by designers of global protocols, diminishing the utility of the approach. Our first contribution is a semantic property of network architectures that ensures their compatibility with the global specification methodology.

We then generalize the sound and complete characterization of implementability for global protocols presented in Chapter 3. The generalization abstracts from the concrete case of peer-to-peer FIFO networks to provide a characterization that is parametric in the network architecture. A key technical contribution of this generalization is an axiomatic network model that abstracts from low-level details of network behavior, enabling equational reasoning at the level of sequences of communication events.

To demonstrate the versatility of our result, we consider eight network architectures that pick among four choices of communication topologies (peer-to-peer, one-to-many, many-to-one, and one-to-one), and two choices of message buffer data structure (FIFO queues and multisets). Out of these eight architectures, six satisfy our compatibility criterion and all of these six also satisfy our axiomatic network model. We then use the generalized implementability characterization to obtain decidability and complexity results for implementability of finite-state global specifications, instantiated for each of the six compatible architectures.

The generalized implementability characterization is fully formalized in the Rocq proof assistant, and we build directly on the proof development of [Li and Wies 2025b], discussed in Chapter 7. To the best of our knowledge, our result is the first mechanized proof of a fundamental concurrency theory property that is parametric in the choice of network architecture. In contrast to existing generalization arguments that are reduction-based, our generalization is formal languages-based, and reasons about different channel architectures using generic, alphabetic

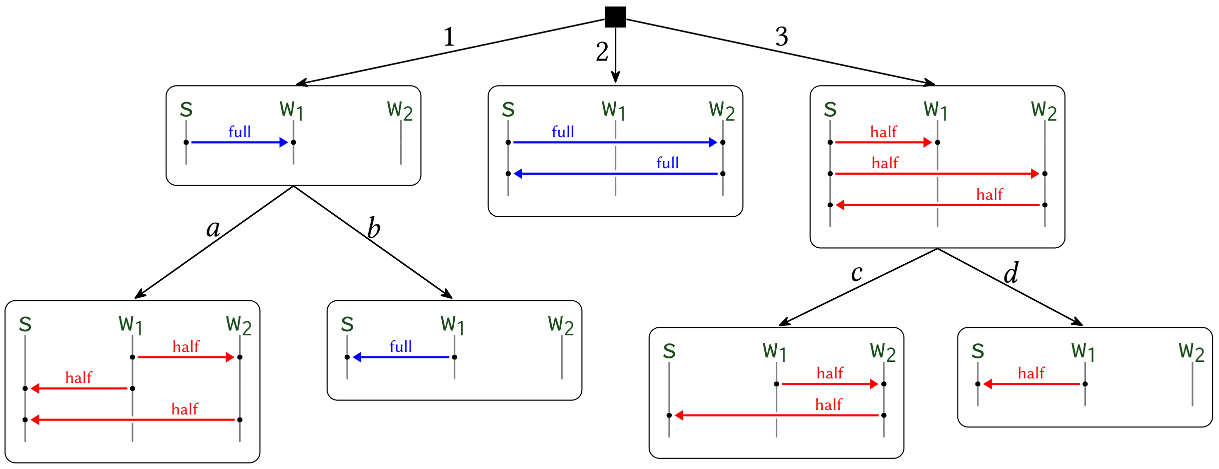


Figure 4.1: Task scheduler with task delegation.

definitions on words.

Contributions. In summary, the contributions in this chapter are:

- We introduce the implementability problem for global protocol specifications modulo network architectures.
- We provide a semantic characterization of whether a given network architecture is compatible with the global protocol design methodology.
- We give a sound and complete characterization of global specification implementability that is parametric in an axiomatic network model. All proofs are mechanized in Rocq.
- We derive decision procedures for the implementability problem modulo six concrete network architectures and provide precise complexity results.

4.1 Overview

We use a task scheduling protocol as our motivating example to illustrate the differences between asynchronous network architectures. The global specification of the protocol in HMSC notation is shown in Fig. 4.1. The protocol involves three participants: a scheduler s and two workers w_1 and w_2. Initially, s chooses to schedule the entire task to either only w_1 or only w_2, or it decides to split the workload between the two workers. The first two cases are depicted in branches 1 and 2, where s sends a full message to the respective worker. The third case is branch 3 where s sends a half message to both workers. Worker w_2 always completes its assigned task immediately and sends the result back to s by echoing the message it has received from s. However, whenever w_1 is assigned a task, it has the option to behave like w_2 (branches *b* and *d*) or to be lazy and delegate some or all of its work to w_2 (branches *a* and *c*). The protocol operates in a loop, but we omit the back-edges to the initial state in Fig. 4.1 for readability.

Implementability modulo network architectures We want to know whether there exist local implementations for the three participants that behave according to the given global protocol specification when executed concurrently on an asynchronous network architecture. In particular, we require that the implementations never deadlock and that all participants behave consistently according to each locally chosen branch, executing send and receive actions exactly in the prescribed order. The latter property is known as *protocol fidelity*. The network architecture is a parameter of the problem statement. A priori, we only require that the network is asynchronous and reliable: (1) messages are not duplicated and (2) messages can be delayed or reordered but not dropped.

Determining implementability A local implementation can only gain information about the global protocol state by making branching decisions and by receiving messages. Protocol violations may arise because a participant has insufficient information to decide what action to take next based on the decisions and observations it has made so far.

Let us start by analyzing the implementability question for the task scheduling protocol, assuming a standard peer-to-peer FIFO network architecture (also referred to as *peer-to-peer box* semantics in this chapter). Throughout the chapter, we use $\mathsf{p} \triangleright \mathsf{q}!m$ to denote a send event where participant p sends m to q. Likewise, we use $\mathsf{q} \triangleleft \mathsf{p}?m$ to denote an event where q receives message m that was previously sent from p.

Under the assumed architecture, the protocol is not implementable: w_2 cannot distinguish between a protocol run that follows branches 1 and a and a run that follows branches 3 and c. In both cases, w_2 may find itself in the same local state q where only the half message from w_1 is available in its associated channel buffer (i.e., in the $3c$ run, the half message from s to w_2 may be delayed). If the protocol is following the $1a$ run, w_2's next action should be to send a reply to s. However, in the $3c$ run it should first wait for the arrival of the half message from s. If w_2 were to wait in state q, this would lead to a deadlock in the $1a$ run and if it were to send to s, it would violate protocol fidelity in the $3c$ run.

Perhaps surprisingly, replacing the peer-to-peer box network by another asynchronous network architecture does not resolve this problem. The reason for non-implementability solely depends on the asynchronous nature of communication and the fact that the two send events $\mathsf{s} \triangleright \mathsf{w}_2!\mathsf{half}$ and $\mathsf{w}_1 \triangleright \mathsf{w}_2!\mathsf{half}$ in the $3a$ run do not causally depend on each other. They can therefore happen concurrently, causing the two messages to arrive at w_2 in any order. Thus, the protocol is non-implementable for any asynchronous network architecture.

However, in general, implementability depends on the specific network architecture. For example, consider a possible repair of the global specification that replaces the message value half of the send from w_1 to w_2 on branch $1a$ with delegate. Now, w_2 can tell the two branches apart: it can wait until either a half message is available in its buffer from s, indicating that the protocol follows branch 3, or a delegate message is available in the buffer from w_2, indicating that the other participants chose to follow branch $1a$. Since the two cases are exclusive, w_2 can make its decision as soon as it observes one of the two. This change renders the protocol implementable under peer-to-peer box semantics.

On the other hand, this repair does not help if the network architecture has a mailbox semantics (i.e., all messages sent to the same recipient are collected in one FIFO buffer). The issue with mailbox semantics is that in the 3*c* branch, the network may still asynchronously reorder the two messages sent to $\mathtt{w}_2$ by delaying the message from $\mathtt{s}$. Since messages are buffered in FIFO order of arrival, this would force $\mathtt{w}_2$ to first receive the message from $\mathtt{w}_1$ before being able to retrieve the one from $\mathtt{s}$ in the buffer. The resulting sequence of events would violate protocol fidelity.

If on the other hand, we change the network from mailbox to *mailbag* semantics where there is a single buffer per recipient but the buffer is unordered, then the protocol becomes implementable again with the proposed repair.

In this chapter, we provide a uniform characterization of the implementability problem that addresses these subtle differences in the semantics of the network architecture.

Compatibility with global specifications The case of mailbox semantics deserves some further discussion. Consider again the problematic run of 3*a* where the message from $\mathtt{w}_1$ arrives before the one from $\mathtt{s}$ in $\mathtt{w}_2$'s mailbox. Note that the ensuing violation of protocol fidelity has nothing to do with incomplete information by any of the protocol participants about what branch the protocol is following. Instead, it is solely due to the constraints imposed by the network architecture on the executions of individual protocol runs. In particular, the problem cannot be repaired by changing message payloads in the protocol specification, like we did for the peer-to-peer box network. In a way, the protocol asks for an implementation that can control aspects of the network behavior that are inherently not under the control of the sought-after local implementations.

As another contribution of this chapter, we propose a semantic property of network architectures that separates those architectures that are compatible with global specifications (like peer-to-peer box and mailbag) from those that are not (like mailbox).

Outline. We formally define the implementability problem for global specifications modulo network architectures in Section 4.2. In Section 4.3 we then introduce our compatibility property that characterizes well-suited network architectures. Section 4.4 presents an axiomatic abstraction of network architectures that enables us to reason modularly and equationally about network behaviors without committing to any particular architecture. Building on this abstraction, Section 4.5 then presents sound and complete conditions that characterize implementability of a given global specification. These conditions can be checked directly on the specification. We show that this characterization applies to all well-known network architectures that satisfy our compatibility criterion and we use it to obtain decidability results with optimal complexity bounds for implementability of finite state specifications, modulo each of these architectures. Finally, Section 4.6 provides a detailed comparison with related work before we conclude with a discussion of extensions and open problems in Section 4.7.

4.2 Implementability modulo network architectures

In this section, we present our network-parametric formalism implementation model for asynchronous communication protocols. We show how to generalize global protocol semantics to likewise be parametric in an implementation model with some choice of network architecture. We finally define the parametric implementability problem in a choice of architecture.

Our implementation model is based on communicating state machines (CSMs) [Brand and Zafiropulo 1983]. CSMs consist of a collection of finite state machines, one for each participant, that communicate via pairwise FIFO channels. We generalize CSMs along two key dimensions: the communication topology, and the data structure for message buffers. We also lift the restriction imposed by CSMs that the number of participants and the state spaces of the local state machines must be finite, as done in [Li et al. 2025b; Li and Wies 2025b]. In terms of communication topology, we consider four models studied in the literature: n-to-n, in which all senders and receivers share the same global channel, one-to-n, in which receivers share the same channel to receive from a single sender, n-to-one, in which senders share the same channel to send to a single receiver, and one-to-one, in which each sender and receiver pair have a unique channel. In terms of message buffers, we consider two options: ordered FIFO queues, and unordered multisets. To give some examples of communication architectures in this family that are used in practice: the FIFO n-to-n model is also known as a global bus, the one-to-one multiset model is referred to as a message soup and commonly used in leader election protocols, the one-to-n model is commonly seen in work-stealing parallel programming paradigms, the one-to-n is also known as mailbox communication, and the one-to-one or peer-to-peer model is the standard network architecture for CSMs and widely assumed in the theory and practice of verification.

Following [Delpy et al. 2024], we define the communication topology in terms of a map from a sender and receiver pair to a message buffer.

Definition 4.1 (Network architecture)**.** A *network architecture* over a set of participants $\mathcal{P}$ is a pair $\mathbb{A} = (\mathbb{X}, \mathsf{B}, \mathsf{bf})$ where $\mathbb{X}$ is an abstract type of channel states, B is a set of *channel contents* and $\mathsf{bf} : \mathbb{X} \to \mathcal{P} \times \mathcal{P} \to \mathsf{B}$ is a map that associates each sender and receiver pair with a channel contents. Intuitively, $\mathsf{bf}(\xi, \mathsf{p}, \mathsf{q})$ denotes the message buffer to which messages sent from p to q are deposited.

We define the considered network architectures in terms of the cross product of four communication topologies and two buffer types. The four communication topologies are defined as follows (with our naming conventions given in parenthesis, where "B" refers to the name of one of the buffer types below):

- n-to-n (peer-to-peer B): $\mathbb{X} = \mathcal{P} \times \mathcal{P} \to \mathsf{B}, \mathsf{bf}(\xi, \mathsf{p}, \mathsf{q}) = \xi(\mathsf{p}, \mathsf{q})$,
- one-to-n (mailB): $\mathbb{X} = \mathcal{P} \to \mathsf{B}, \mathsf{bf}(\xi, \mathsf{p}, \mathsf{q}) = \xi(\mathsf{p})$,
- n-to-one (senderB): $\mathbb{X} = \mathcal{P} \to \mathsf{B}, \mathsf{bf}(\xi, \mathsf{p}, \mathsf{q}) = \xi(\mathsf{q})$,
- one-to-one (monoB): $\mathbb{X} = \mathsf{B}, \mathsf{bf}(\xi, \mathsf{p}, \mathsf{q}) = \xi$,

and the two buffer types are:

- FIFO queues (B=box): $\mathsf{B} = \mathcal{V}^*$, and
- multisets (B=bag): $\mathsf{B} = \mathcal{V} \to \mathbb{N}$.

We assume that buffers in B are equipped with a total *insert* and a partial *remove* operation. The later is only defined when a given message is available in a channel. We lift the definitions of *insert* and *remove* to channels $\xi \in \mathbb{X}$ in the following way: $\mathit{insert}(\xi, \mathsf{p}, \mathsf{q}, m) = \xi'$ where $\mathsf{bf}(\xi', \mathsf{p}, \mathsf{q}) = \mathit{insert}(\mathsf{bf}(\xi, \mathsf{p}, \mathsf{q}))$ and all other message buffers remain unchanged; $\mathit{remove}(\xi, \mathsf{p}, \mathsf{q}, m) = \xi'$ where $\mathsf{bf}(\xi', \mathsf{p}, \mathsf{q}) = \mathit{remove}(\mathsf{bf}(\xi, \mathsf{p}, \mathsf{q}))$ and all other message buffers remain unchanged.

In the case of FIFO queues, *insert* corresponds to appending at the end of the queue, *remove* corresponds to removing from the head; in the case of multisets, *insert* is multiset addition, and *remove* multiset deletion. Throughout this chapter, we will use set notation for multisets and multiset operations. We also assume a unique empty channel state $b_0 \in \mathsf{B}$, which is ε in the case of FIFO queues and $\emptyset$ in the case of multiset buffers.

We present our definition of communicating labeled transition systems parametric in a choice of network architecture $\mathbb{A}$ below, generalizing the definition of CLTS from Chapter 2.

Network-parametric CLTS $\mathcal{T}_{\mathbb{A}} = \{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ is a CLTS over $\mathcal{P}$, $\mathcal{V}$ and $\mathbb{A}$ if T_p is a deterministic LTS over Σ_p for every $\mathsf{p} \in \mathcal{P}$, denoted $(Q_\mathsf{p}, \Sigma_\mathsf{p}, \delta_\mathsf{p}, q_{0,\mathsf{p}}, F_\mathsf{p})$. Let $\prod_{\mathsf{p}\in\mathcal{P}} Q_\mathsf{p}$ denote the set of global states. A *configuration* of $\mathcal{A}$ is a pair $(\vec{s}, \xi)$, where $\vec{s}$ is a global state and $\xi \in \mathbb{X}$ is a channel state. We use $\vec{s}_\mathsf{p}$ to denote the state of p in $\vec{s}$. The CLTS transition relation, denoted $\to$, is defined as follows.

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

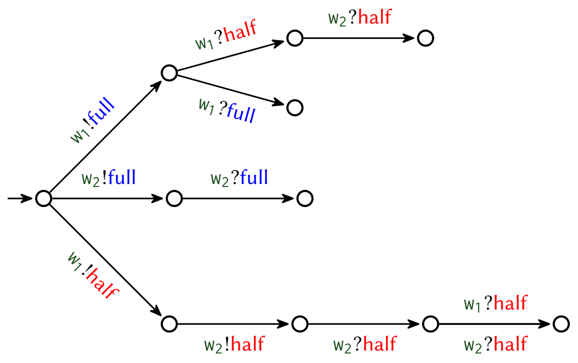
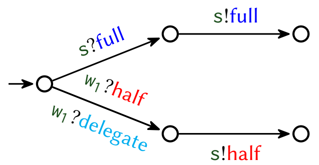


Figure 4.2: Local implementation for s



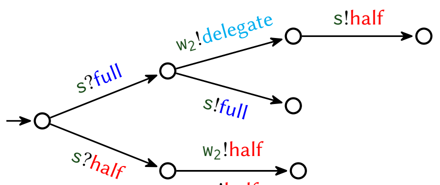


Figure 4.3: Local implementation for w_2 **Figure 4.4:** Local implementation for w_1

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

The local implementations for participants s, w_1 and w_2 of the repaired task scheduling protocol from Section 4.1 assuming a peer-to-peer box network are depicted in Fig. 4.2, Fig. 4.4 and Fig. 4.3 respectively. Note that the active participants are omitted from transition labels for clarity.

We present network-parametric definitions of executable words, and of global protocol semantics below. The incorporation of network-parametricity does not change much from the definitions assuming a peer-to-peer box network presented in Chapter 2.

Executable words of a network-parametric CLTS A finite asynchronous word $w \in \Sigma_{async}^*$ is *executable* in a CLTS $\mathcal{T}_{\mathbb{A}}$ if $w \in \mathrm{pref}(\mathcal{L}(\mathcal{T}_{\mathbb{A}}))$. We say that $w \in \Sigma_{async}^*$ is *executable under* $\mathbb{A}$ if it is executable in some $\mathcal{T}_{\mathbb{A}}$ and use $\mathcal{L}(\mathbb{A}) \subseteq \Sigma_{async}^*$ to denote all such words.

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

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

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

The finite semantics is obtained by following the above recipe, but restricting $\texttt{split}(\mathcal{L}(\mathcal{S}))$ to finite words:

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

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

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

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

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

Definition 4.2 (Network-parametric Protocol Implementability)**.** A protocol $\mathcal{S}$ is *implementable* under network architecture $\mathbb{A}$ if there exists a CLTS $\mathcal{T}_{\mathbb{A}} = \{\!\{T_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ such that the following two properties hold: (i) *protocol fidelity*: $\mathcal{L}(\{\!\{T_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}}) = \mathcal{L}_{\mathbb{A}}(\mathcal{S})$, 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 $\mathcal{S}$ under $\mathbb{A}$.

4.3 Global specification compatibility

Observe that the only controllable components of a CLTS specification are the LTSs for the local participants: the network is uncontrollable. For an implementation to realize a global specification, however, both the local implementations and the network must be well-behaved. Consider the simple straight line global specification:

$$\begin{tikzcd} {} \arrow[r] & \circ \arrow[r, "\mathsf{p}_1 \to \mathsf{q} : m"] & \circ \arrow[r, "\mathsf{p}_2 \to \mathsf{q} : m"] & \circledcirc \end{tikzcd}$$

Despite its simplicity, there does not exist a deadlock-free mailbox CLTS that implements this global specification. Any candidate CLTS must exhibit the following deadlocking trace:

$$\mathsf{p}_2 \triangleright \mathsf{q}!m \cdot \mathsf{p}_1 \triangleright \mathsf{q}!m \quad .$$

Because the network can reorder p_2's send event before p_1's send event, yet the local actions of q tell it to receive in the opposite order, the only way for the CLTS to not deadlock is for T_{q} to admit the local trace $\mathsf{q} \triangleleft \mathsf{p}_2?m \cdot \mathsf{q} \triangleleft \mathsf{p}_1?m$. The same issue arises in monobox CLTS, which can be seen as multiplexing all peer-to-peer FIFO queues into a single, global FIFO queue. Note, however, that deadlocks do not arise when FIFO queues are replaced with unordered multisets. Notice further that the issue ceases to arise in senderbox communication, in which all messages from a single sender to all other recipients are enqueued into a single FIFO queue, i.e., the dual of mailbox communication.

Intuitively, a network architecture $\mathscr{A}$ is compatible with global specifications if no global specification can prescribe that its $\mathbb{A}$-implementations distinguish between two asynchronous words that are fundamentally indistinguishable by $\mathbb{A}$. Thus, mailbox and monobox architectures are not compatible with global specifications, as demonstrated by the above example. We argue that this notion of compatibility provides a litmus test for whether a given network architecture is well-suited for the top-down protocol design methodology.

We formalize this compatibility property of a network architecture $\mathbb{A}$ as follows. Recall that an asynchronous word $w \in \Sigma$ is *executable* in a CLTS $\mathcal{T}_{\mathbb{A}}$ if $w \in \text{pref}(\mathcal{L}(\mathcal{T}_{\mathbb{A}}))$ and w is executable under $\mathbb{A}$ if it is executable in some $\mathcal{T}_{\mathbb{A}}$. We say that two asynchronous words w_1 and w_2 can be distinguished by $\mathbb{A}$ if there exists $\mathcal{T}_{\mathbb{A}}$ such that either w_1 or w_2 is executable in $\mathcal{T}_{\mathbb{A}}$, but not both.

Definition 4.3 (Compatibility)**.** A network architecture $\mathbb{A}$ is compatible with global specifications if no asynchronous words $w_1, w_2 \in \Sigma_{async}^*$ with $w_1 \equiv_{\mathcal{P}} w_2$ that are executable under $\mathbb{A}$ can be distinguished by $\mathbb{A}$.

The remaining six network architectures aside from mailbox and monobox all satisfy this compatibility property.

Network architectures that violate compatibility violate the atomicity of straight-line specifications. Thus, they are ill-suited as implementation targets for global specifications, which can be viewed as the branching composition of straight line specifications. Implementability at heart concerns a problem of partial information about global branching control flow: given an incomplete view of the global protocol state, does each participant have enough information to correctly follow the protocol? In the case of the example protocol above, all participants have complete information over the global protocol, yet errors arise from uncontrollable network scheduling. Thus, we contend that protocols with such semantics constitute degenerate cases that should be excluded from global specification-based methodologies. In a word, specifications whose straight line constituents are themselves potentially non-realizable should not be composed.

4.4 Channel compliance: an alphabetic abstraction of network architectures

Recall that protocol semantics are centrally defined in terms of executability, which is in turn defined as the existence of a CLTS that executes a given word under the given network architecture. Thus far, our only understanding of executability is an operational one, in terms of the global transition relation of a CLTS. The global transition relation operates on the state space and channel space simultaneously, and thus concerns itself with both local implementation behavior and channel behavior. To complicate things further, channel behavior is highly specific to a given network architecture. These operational differences get in the way of general reasoning about CLTS with different network architectures all at once.

We address this challenge by wholesale replacing the operational view of network architectures we have presented thus far with a purely algebraic view. We introduce a key abstraction that enables us to reason about implementability in a network-parametric manner: *channel compliance*. Our notion of channel compliance first eliminates local implementations from the picture, separating out the controllable local implementations' behaviors from uncontrollable network behavior. This allows us to reason purely about sequences of send and receive events that are executable modulo assumptions about local implementations. Channel compliance then gives an algebraic specification of asynchronous words that satisfy the ordering restrictions imposed by a given network architecture. Our notion of channel compliance is inspired by and named after the corresponding definition for peer-to-peer box networks, introduced in [Majumdar et al. 2021a] to define multiparty session type semantics. Similar sequential specifications of communication models can also be found in [Chevrou et al. 2016].

The rest of the section is structured as follows. First, we present a correspondence between operational CLTS and algebraic channel compliance that ensures our notion of channel compliance is capturing exactly the right thing. Then, we present alphabetical definitions of channel compliance that precisely capture the remaining six network architectures that we study. Having established their correspondence with their target CLTS implementation models, we use them to simplify our global protocol semantics. Finally, we discuss the algebraic abstraction that unifies all these concrete channel compliance notions and show how it helps us to achieve a network-parametric characterization of implementability.

Throughout this section, we fix a network architecture $\mathbb{A}$. We use $C_{\mathbb{A}} \subseteq \Sigma^*$ to denote the set of $\mathbb{A}$-channel compliant words. The algebraic specification of $\mathbb{A}$ will be given later. First, we define what it means for $C_{\mathbb{A}}$ to accurately capture a network architecture in terms of the following correspondence, which can be viewed as a meta-correctness criterion for our proposed algebraic abstractions:

Definition 4.4 (Correspondence between channel compliance and CLTS network architecture)**.** Let $C_{\mathbb{A}} \subseteq \Sigma^*$ be a notion of channel compliance. We say that $C_{\mathbb{A}}$ *precisely captures* $\mathbb{A}$ if for any $\mathbb{A}$-CLTS $\mathcal{T}_{\mathbb{A}}$ and $w \in \Sigma^*_{async}$, w is a trace of $\mathcal{T}_{\mathbb{A}}$ if and only if $w \in C_{\mathbb{A}}$ and for all $\mathsf{p} \in \mathcal{P}$, $w{\Downarrow}_{\Sigma_{\mathsf{p}}} \in \operatorname{pref}(\mathcal{L}(T_{\mathsf{p}}))$.

Once we have established that $C_{\mathbb{A}}$ precisely captures $\mathbb{A}$, we can use it to simplify our global protocol semantics, by replacing $\mathcal{L}(\mathbb{A})$, which is defined in terms of the existence of a CLTS $\mathcal{T}_{\mathbb{A}}$, with $C_{\mathbb{A}}$.

Corollary 4.5 (Network-parametric Asynchronous Protocol Semantics)**.** *Let $\mathcal{S}$ be an LTS over Γ_{sync}. If $C_{\mathbb{A}}$ precisely captures $\mathbb{A}$, then*

$$\begin{aligned}\mathcal{L}_{\mathbb{A}}(\mathcal{S}) = {} & ([\Sigma^*_{async} \cap \mathtt{split}(\mathcal{L}(\mathcal{S}))]_{\equiv_{\mathcal{P}}} \cap C_{\mathbb{A}}) \\ & \cup \{w \in \Sigma^{\infty}_{async} \mid \forall u \leq w.\ u \in \operatorname{pref}([\mathtt{split}(\mathcal{L}(\mathcal{S}))]_{\equiv_{\mathcal{P}}} \cap C_{\mathbb{A}})\}\ .\end{aligned}$$

We propose alphabetical notions of channel compliance for each of the six network architectures we consider below that we have shown to satisfy the meta-correctness correspondence in our Rocq mechanization.

As a reminder, $\Downarrow_{-}$ means projection to either the identity or ε, depending on whether the symbol at hand pattern-matches -. Note that in the peer-to-peer case, it suffices to compare sequences of message values, because the sender and receiver pair is fixed. This is not true in general for other channel architectures. In other words, for other channel architectures, messages need to contain their sender and/or receiver ID in order to be properly handled by the network. In the following, let $w \in \Sigma^{\infty}$.

Definition 4.6 (Peer-to-peer box channel compliance)**.** We say that w is *p2p box channel compliant* if for all prefixes $w' \leq w$, for all $\mathsf{p} \neq \mathsf{q} \in \mathcal{P}$, $w'\Downarrow_{\mathsf{q} \triangleleft \mathsf{p}?\_} \leq w'\Downarrow_{\mathsf{p} \triangleright \mathsf{q}!\_}$.

Definition 4.7 (Peer-to-peer bag channel compliance)**.** We say that w is *p2p bag channel compliant* if for all prefixes $w' \leq w$, for all $\mathsf{p} \neq \mathsf{q} \in \mathcal{P}$, $\{m \mid \mathsf{q} \triangleleft \mathsf{p}?m \in w'\} \subseteq \{m \mid \mathsf{p} \triangleright \mathsf{q}!m \in w'\}$.

Definition 4.8 (Senderbox channel compliance)**.** We say that w is *senderbox channel compliant* if for all prefixes $w' \leq w$, for all $\mathsf{p} \in \mathcal{P}$, $w'\Downarrow_{\_ \triangleleft \mathsf{p}?\_} \leq w'\Downarrow_{\mathsf{p} \triangleright \_!\_}$.

Definition 4.9 (Senderbag channel compliance)**.** We say that w is *senderbag channel compliant* if for all prefixes $w' \leq w$, for all $\mathsf{p} \in \mathcal{P}$. $\{(m, \mathsf{q}) \mid \mathsf{q} \triangleleft \mathsf{p}?m \in w'\} \subseteq \{(m, \mathsf{q}) \mid \mathsf{p} \triangleright \mathsf{q}!m \in w'\}$.

Definition 4.10 (Mailbag channel compliance)**.** We say that w is *mailbag channel compliant* if for all prefixes $w' \leq w$, for all $\mathsf{q} \in \mathcal{P}$. $\{(m, \mathsf{p}) \mid \mathsf{q} \triangleleft \mathsf{p}?m \in w'\} \subseteq \{(m, \mathsf{p}) \mid \mathsf{p} \triangleright \mathsf{q}!m \in w'\}$.

Definition 4.11 (Monobag channel compliance)**.** We say that w is *monobag channel compliant* if for all prefixes $w' \leq w$, $\{(m, \mathsf{p}, \mathsf{q}) \mid \mathsf{q} \triangleleft \mathsf{p}?m \in w'\} \subseteq \{(m, \mathsf{p}, \mathsf{q}) \mid \mathsf{p} \triangleright \mathsf{q}!m \in w'\}$.

The correspondence for each $(\mathbb{A}, C_{\mathbb{A}})$ pair is proven by induction on word length.

In the remainder of the chapter, we refer to $\mathbb{A}$-channel compliance simply as channel compliance, and assume that the network parameter is implicit unless explicitly specified.

Next, we discuss the benefits that our alphabetic characterization of $\mathbb{A}$-executable confer in terms of 1) facilitating automation, 2) clarifying the relationship between network architectures, and 3) simplifying definitions.

Automation. Channel compliance reduces odious, operational CLTS reasoning about explicit channel states and transitions in each participant's local implementation, to algebraic, equational reasoning phrased in terms of word concatenation, homomorphisms, and prefixes ordering. This makes proofs more amenable to automation, a fact that we exploit in our Rocq development.

All bag CLTSs are equivalent. An immediate consequence of our alphabetic characterization is that all network architectures with multiset message buffers are equivalent.

Lemma 4.12. *Peer-to-peer bag, mailbag, senderbag and monobag channel compliance are equivalent.*

This can easily be shown by proving senderbag, mailbag and peer-to-peer bag channel compliance to all be equivalent to monobag channel compliance. From an implementation perspective, this equivalence has the following operational interpretation: in a monobag CLTS with a global message soup, since messages in the soup are all labeled with sender and receiver, one can "on demand" separate the message soup into $\mathcal{P}^2$ multisets, or $\mathcal{P}$ multisets by sender ID or receiver ID whenever messages are sent or received, and thus simulate the other network architectures.

The implication of this equivalence is that despite the difference in CLTS definitions for the four bag architectures, we can show a correspondence between each and monobag channel compliance, and prove that monobag channel compliance satisfies the required assumptions once and for all.

Receivability in terms of channel compliance. In the remainder of this section, we develop the algebraic abstraction of the six concrete channel compliance notions that we will later use for our proofs.

A central notion to reasoning about asynchronous communication is that of *receivability*. From the definitions of CLTS, we see that whether a receive transition $\bar{\mathsf{q}} \triangleleft \bar{\mathsf{p}}?\bar{m}$ is enabled from some execution prefix w depends on whether the message $\bar{m}$ is *available* in its respective message channel, both components of which take on a different meaning depending on the network architecture. Following in the spirit of channel compliance, one could consider also giving algebraic definitions of receivability for each network architecture as follows:

Definition 4.13 (Peer-to-peer box receivable)**.** We say that $\bar{m}$ from $\bar{\mathsf{p}}$ to $\bar{\mathsf{q}}$ is *p2p box receivable* in w if $w{\Downarrow}_{\mathsf{q} \triangleleft \mathsf{p}?\_} \cdot \bar{m} \leq w{\Downarrow}_{\mathsf{p} \triangleright \mathsf{q}!\_}$.

Definition 4.14 (Peer-to-peer bag receivable)**.** We say that $\bar{m}$ from $\bar{\mathsf{p}}$ to $\bar{\mathsf{q}}$ is *p2p bag receivable* in w if $\{m \mid \mathsf{q} \triangleleft \mathsf{p}?m \in w'\} \cup \{\bar{m}\} \subseteq \{m \mid \mathsf{p} \triangleright \mathsf{q}!m \in w'\}$.

Definition 4.15 (Senderbox receivable)**.** We say that $\bar{m}$ from $\bar{\mathsf{p}}$ to $\bar{\mathsf{q}}$ is *senderbox receivable* in w if $w{\Downarrow}_{\_ \triangleleft \mathsf{p}?\_} \cdot (\bar{m}, \bar{\mathsf{q}}) \leq w{\Downarrow}_{\mathsf{p} \triangleright \_!\_}$.

Definition 4.16 (Senderbag receivable)**.** We say that $\bar{m}$ from $\bar{\mathsf{p}}$ to $\bar{\mathsf{q}}$ is *senderbag receivable* in w if $\{(m, \mathsf{q}) \mid \mathsf{q} \triangleleft \mathsf{p}?m \in w\} \cup \{(\bar{m}, \bar{\mathsf{q}})\} \subseteq \{(m, \mathsf{q}) \mid \mathsf{p} \triangleright \mathsf{q}!m \in w\}$.

Definition 4.17 (Mailbag receivable)**.** We say that $\bar{m}$ from $\bar{\mathsf{p}}$ to $\bar{\mathsf{q}}$ is *mailbag receivable* in w if $\{(m, \mathsf{p}) \mid \mathsf{q} \triangleleft \mathsf{p}?m \in w'\} \cup \{(\bar{m}, \bar{\mathsf{p}})\} \subseteq \{(m, \mathsf{p}) \mid \mathsf{p} \triangleright \mathsf{q}!m \in w'\}$.

Definition 4.18 (Monobag receivable)**.** We say that $\bar{m}$ from $\bar{\mathsf{p}}$ to $\bar{\mathsf{q}}$ is *monobag receivable* in w if $\{(m, \mathsf{p}, \mathsf{q}) \mid \mathsf{q} \triangleleft \mathsf{p}?m \in w'\} \cup \{(\bar{m}, \bar{\mathsf{p}}, \bar{\mathsf{q}})\} \subseteq \{(m, \mathsf{p}, \mathsf{q}) \mid \mathsf{p} \triangleright \mathsf{q}!m \in w'\}$.

It turns out that receivability admits a very direct algebraic characterization in terms of channel compliance that unifies all these concrete notions of receivability. It is the first property defining the algebraic abstraction of channel compliance that we use for our proofs:

Definition 4.19 (Receivability in terms of channel compliance)**.** Let $w \in \Sigma^*$, $\mathsf{p}, \mathsf{q} \in \mathcal{P}$ and $m \in \mathcal{V}$, with $\mathsf{p} \neq \mathsf{q}$ and w $\mathbb{A}$-channel compliant. We say that m is *receivable* from p to q in w, denoted $\mathrm{receivable}(w, \mathsf{p}, \mathsf{q}, m)$ if $w \cdot \mathsf{q} \triangleleft \mathsf{p}?m$ is also $\mathbb{A}$-channel compliant.

Receivability thus serves as a salient example of a property concerning CLTS with some network architecture, that is possible to state and prove on the CLTS directly, but that is made much easier by an alphabetic abstraction of channel compliance.

We conclude with the remaining channel compliance related properties that we employ in our generalized proof. The first pair of properties situates the network architectures we consider squarely in the hierarchy of asynchronous communication models commonly found in the literature [Chevrou et al. 2016; Giusto et al. 2023], between the "lower bound" of purely asynchronous (`asy`) and "upper bound" of realizable with synchronous communication (`rsc`). The most permissive asynchronous model simply restricts messages to be sent before they are received, and otherwise imposes no other orders. Architecturally, this is equivalent to multiset CLTS. The least permissive asynchronous model requires messages that are sent to be immediately received. Architecturally, this is equivalent to a CLTS with a single global channel of size 1.

Definition 4.20 (1-synchronous words are channel compliant)**.** Let $u \in \Gamma_{sync}^*$. Then, $\texttt{split}(u)$ is channel compliant.

Definition 4.21 (Channel compliant words respect send-before-receive)**.** Let $w \in \Sigma_{async}^*$. If w is channel compliant, then w satisfies send-before-receive order.

The next set of algebraic properties discuss channel compliance under word concatenation.

Definition 4.22 (Channel compliance concatenation properties)**.** Channel compliance satisfies the following properties:

1. For all $w_1, w_2 \in \Sigma_{async}^*$ and $u \in \Gamma_{sync}^*$, if w_1 is channel compliant and $w_1 \equiv_{\mathcal{P}} \texttt{split}(u)$, then w_2 is channel compliant iff $w_1 \cdot w_2$ is channel compliant.

2. For all $w \in \Sigma_{async}^*$, $x \in \Sigma_!$, if w is channel compliant then wx is channel compliant.

3. For all $w \in \Sigma_{async}^*$, $x \in \Sigma_!$, $y \in \Sigma_?$, if wy is channel compliant then wxy is channel compliant.

Note that the second property amounts to saying that messages can always be sent while preserving channel compliance. The inclusion of this property means that our assumptions do not completely characterize compatible network architectures: bounded FIFO architectures, for example, are compatible but do not satisfy this property.

The final set of properties characterize the past and future of channel compliant words. We say that $w \in \Sigma_{async}^*$ *agrees* with a synchronous word u when for all participants $\mathsf{p} \in \mathcal{P}$, $w\Downarrow_{\Sigma_\mathsf{p}} \leq \mathtt{split}(u)\Downarrow_{\Sigma_\mathsf{p}}$. Given a synchronous word $u = u_1u_2$ and an agreeing asynchronous word w, we can "snip" u_1 from w as follows: for each participant p, we delete from w the larger of $u_1\Downarrow_{\Sigma_\mathsf{p}}$ and $w\Downarrow_{\Sigma_\mathsf{p}}$. The resulting w' satisfies that every participant who in w has not completed all events prescribed by u_1 is left with no events in w', i.e. $w'\Downarrow_{\Sigma_\mathsf{p}} = \varepsilon$, whereas every participant who in w has completed all events prescribed by u_1 and some additional events prescribed by u_2 is left with only the events prescribed by u_2, i.e. $w\Downarrow_{\Sigma_\mathsf{p}} = \mathtt{split}(u_1)\Downarrow_{\Sigma_\mathsf{p}} \cdot w'\Downarrow_{\Sigma_\mathsf{p}}$.

Definition 4.23 (Receivable is history insensitive)**.** Let $w, w' \in \Sigma_{async}^*$, $\alpha, \beta \in \Gamma_{sync}^*$, $\mathsf{p} \neq \mathsf{q} \in \mathcal{P}$ and $m \in \mathcal{V}$ such that w agrees with $\alpha\beta$, both w and w' are $\mathbb{A}$–channel compliant, $\mathtt{split}(\alpha)\Downarrow_{\Sigma_\mathsf{q}} \leq w\Downarrow_{\Sigma_\mathsf{q}}$ w' is the result of snipping α from w, and $\text{receivable}_\mathbb{A}(w, \mathsf{p}, \mathsf{q}, m)$. Then, it holds that $\text{receivable}_\mathbb{A}(w', \mathsf{p}, \mathsf{q}, m)$.

This property states that a channel compliant word can forget about a section of its past without changing the receivable status of a currently receivable message, under the condition that the receiver has completed all prescribed events in the past.

Looking to the future, the following property states that any channel compliant word that agrees with some synchronous word can be extended to be equal under local projection to its 1-synchronous counterpart.

Definition 4.24 (Channel compliant prefix extensibility)**.** Let $w \in \Sigma_{async}^*$, $\rho \in \Gamma_{sync}^*$ such that w is channel compliant and agrees with ρ. Then, there exists $u \in \Sigma_{async}^*$ such that wu is channel compliant, and $wu \equiv_\mathcal{P} \mathtt{split}(\rho)$.

Finally, to reason about finite CLTS words, we require that channel compliant words that are equal under local projection to a 1-synchronous word lead to a configuration in the CLTS in which all channels are empty. Note that this is equivalent to saying that such a word has an equal number of send and receive events for each sender, receiver and message value triple.

Definition 4.25 (Matched words mean empty channels)**.** Let $\mathcal{T}_{\mathbb{A}}$ be a CLTS, $w \in \Sigma_{async}^*$ and $u \in \Gamma_{sync}^*$ such that $w \equiv_{\mathcal{P}} \texttt{split}(u)$. Then, in the configuration reached on w, all channels are empty.

4.5 Characterization of generalized implementability

In this section, we present our generalized implementability characterization. We first revisit the characterization for the peer-to-peer box case, presented in [Li et al. 2025b] and Chapter 3 in the form of three Coherence Conditions (*CC*). The Coherence Conditions scrutinize pairs of global protocol states from which a participant can perform different actions, but whose distinction may not be locally observable to the participant. *CC* describes the kinds of local actions that are safe to perform in this state of unawareness. *Send Coherence* says that if a participant has the option to perform a send action from one state, it must have the option to perform the same send action from any indistinguishable state. *Receive Coherence* says that if a participant has the option to perform a receive action from one state, then this same receive action could not possibly be performed from any other indistinguishable state. *No Mixed Choice* says that a participant cannot equivocate between performing a send and receive action.

In a nutshell, *CC* are 2-hyperproperties stating that from two simultaneously reachable global protocol states, a participant can either perform a send action that is enabled in both states (Send Coherence), or perform a receive action that uniquely distinguishes the two states (Receive Coherence), but cannot choose between performing a send or receive action (No Mixed Choice).

In Chapter 3, we showed that *CC* is sound by invoking a *canonical implementation* (Definition 3.6), which serves as the witness to implementability.

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

Completeness of *CC* is established in Section 3.3 via modus tollens: from the negation of each Coherence Condition a trace is constructed that is compliant with no protocol run, yet must be admitted by any candidate implementation of the protocol. This suffices for an implementability violation, because either the trace leads to a deadlock, or to a maximal word not in the global protocol semantics.

In fact, two of three Coherence Conditions, Send Coherence and No Mixed Choice, remain sound and complete for network-parametric implementability. The reason for soundness boils down to the following assumption about channel compliance (Definition 4.22, (2)):

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

Intuitively, this assumption entails that send actions are equally dangerous across all considered network architectures, since they are enabled by the existence of a transition alone. As for the completeness of Send Coherence and No Mixed Choice, we observe that the witness constructed by our completeness proof in Section 3.3 can be adapted to one that is the prefix of a 1-synchronous trace. Concretely, the witnesses for both conditions assume the form of $\texttt{split}(\alpha) \cdot x$, where α is a synchronous word and x is a send event. Because traces of this form are universally channel compliant, they are executable under any network architecture, and thus if they are compliant with no protocol run, they constitute an implementability violation under every network architecture.

Receive Coherence unfortunately complicates the picture. In contrast to send events, receive events are conditioned on transitions as well as the availability of the message in question to be received. Message availability in turn highly depends on the network architecture, and requires considering possible asynchronous reorderings, as evidenced by participant w_2's plight in the task delegation protocol from §4.1.

Consider the protocols $\mathcal{S}_a$ and $\mathcal{S}_b$, depicted in Fig. 4.5 and Fig. 4.6, both from the perspective of the receiver q. In $\mathcal{S}_a$ under a monobag network architecture, q's bag can contain any number of m messages from p, in addition to a $\perp$ message. Participant q can never know when it is safe to receive the $\perp$ message and terminate locally, and thus the protocol is non-implementable. If we replace the monobag network with a peer-to-peer box network, the final $\perp$ message is ordered after all m messages have been sent, and thus participant q can always receive the message at the head of its FIFO queue from p. In $\mathcal{S}_b$ under a peer-to-peer box network, when message m from r is available to receive, the same message m could also be available from p simultaneously, leading q to a protocol violation. If we replace the peer-to-peer box network with a senderbox network, this ambiguity again goes away: r's message to s blocks the message to q from being available to receive, because s's reception depends on q first receiving from p, thus a unique message is available for q to receive regardless of the branch selected by p.

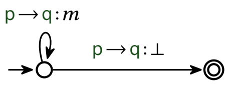


Figure 4.5: A global protocol $\mathcal{S}_a$ that is non-implementable on a bag network but implementable on a peer-to-peer box network.

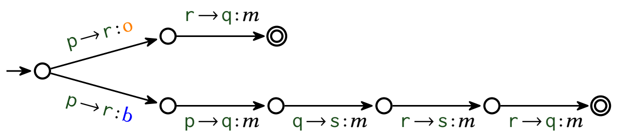


Figure 4.6: A global protocol $\mathcal{S}_b$ that is non-implementable on a peer-to-peer box network but implementable on a senderbox network.

From the two examples, it might appear that any generalization of Receive Coherence must necessarily hardcode the channel architecture of a particular network model. Fortunately, this turns out not to be the case, thanks to our simple notion of *receivability* defined in terms of channel compliance from §4.4. By giving an alphabetic characterization of receivability in terms of channel compliance, alongside the assumption that receivability is history insensitive (Definition 4.23), we can express generalized Receive Coherence parametric in the notion of channel compliance as follows:

Definition 4.26 (Generalized Receive Coherence)**.** A protocol $\mathcal{S} = (S, \Gamma_{sync}, T, s_0, F)$ satisfies $\mathbb{A}$–Receive Coherence when for every two simultaneously reachables states s_1, s_2 with $\forall\, s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:m} s_2, s_1' \xrightarrow{\mathsf{r}\to\mathsf{q}:m'} s_2' \in T$, $(\mathsf{r} \neq \mathsf{p} \;\vee\; m' \neq m) \implies \neg\, \exists\, w \in \mathrm{pref}(\mathcal{L}(\mathcal{S}_{s_1'})).\ w{\Downarrow_{\Sigma_\mathsf{q}}} = \varepsilon \;\wedge\; \mathsf{r} \triangleright \mathsf{q}!m' \leq w{\Downarrow_{\Sigma_\mathsf{r}}} \;\wedge\; \mathit{receivable}_{\mathbb{A}}(w, \mathsf{p}, \mathsf{q}, m)$.

The soundness and completeness proof for generalized Receive Coherence critically relies on the remaining channel compliance assumptions discussed in §4.4. The history insensitivity of receivability is especially tricky to establish for senderbox networks, because the property describes words from a per-participant perspective, whereas channel compliance describes the aggregate of receptions from all participants relative to a single sender. To bridge the gap between these two views of words, we made use of a length-based inductive invariant relating the number of unreceived messages in a senderbox to the number of receptions from the sender in the remainder of the word.

We conclude with a discussion of how to decide the consequent of generalized Receive Coherence involving the *receivable* predicate for each network architecture. More specifically, we present predicates that are equivalent to the following:

$$\exists\, w \in \mathrm{pref}(\mathcal{L}(\mathcal{S}_{s_1'})).\ w{\Downarrow_{\Sigma_\mathsf{q}}} = \varepsilon \;\wedge\; \mathsf{r} \triangleright \mathsf{q}!m' \leq w{\Downarrow_{\Sigma_\mathsf{r}}} \;\wedge\; \mathit{receivable}_{\mathbb{A}}(w, \mathsf{p}, \mathsf{q}, m)\ .$$

In Section 3.4, we defined an $\mathsf{avail}_{\mathsf{p},\mathsf{q},\mathcal{B}}(m, s)$ predicate, which captures whether the protocol starting from state s specifies a trace in which message m from p to q is available in channel (p, q), and no participants in the blocked set $\mathcal{B}$ have executed any actions. We present below avail defined for non-symbolic protocols.

Definition 4.27 (Receivable for peer-to-peer FIFO)**.**

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

The predicate $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m,s'_2)$ captures the negation of the consequent of Receive Coherence for the peer-to-peer box network. The requirement that $w{\Downarrow}_{\Sigma_\mathsf{q}} = \varepsilon$ is enforced by initializing the blocked set to $\{\mathsf{q}\}$, the requirement that $\mathsf{r} \triangleright \mathsf{q}!m' \leq w{\Downarrow}_{\Sigma_\mathsf{r}}$ is enforced by starting from state s'_2, and the requirement that message m from p is receivable by q is enforced by the disjuncts in avail. In a peer-to-peer box network architecture, messages sent from any sender, receiver pair not equal to (p,q) use a separate message channel, and thus cannot interfere with the receivability of message m from p to q. The first two disjuncts of avail skip over such message exchanges to continue searching for the message exchange $\mathsf{p}\to\mathsf{q}:m$.

We treat the remaining network architectures in the same way: we first negate the consequent of generalized Receive Coherence, then we define an avail predicate and instantiate it with p, q, $\{\mathsf{q}\}$ and s'_2. Next, we describe the avail predicate for the remaining network architectures.

Unlike peer-to-peer box networks, senderbox networks force all participants to share the same ordered FIFO queue in order to receive from a given sender. Because messages are enqueued according to the sender's event order, q can be blocked from receiving its message if p first sends a message m' to a non-q receiver, on the condition that said non-q receiver cannot proceed with receiving m'. Thus, when we encounter a message exchange of the form $\mathsf{p}\to\mathsf{q}':m'$ with $\mathsf{q}' \neq \mathsf{q}$ and q' blocked, we can quit early in our search for $\mathsf{p}\to\mathsf{q}:m$. This new stopping condition is reflected in the third disjunct of sbavail, the analogue of avail for the senderbox network architecture.

Definition 4.28 (Receivable for senderbox)**.**

$$
\begin{aligned}
\mathsf{sbavail}_{\mathsf{p},\mathsf{q},\mathcal{B}}(m, s) \coloneqq_\mu \quad & \Big(\bigvee_{\substack{(s,\, \mathsf{r}\to\mathsf{t}:m',\, s')\in\Delta \\ \mathsf{r}\in\mathcal{B} \\ \mathsf{r}\neq\mathsf{p}}} \mathsf{sbavail}_{\mathsf{p},\mathsf{q},\mathcal{B}\cup\{\mathsf{t}\}}(m, s') \Big) \\
& \vee \Big(\bigvee_{\substack{(s,\, \mathsf{r}\to\mathsf{t}:m',\, s')\in\Delta \\ \mathsf{r}\notin\mathcal{B} \\ \mathsf{r}\neq\mathsf{p}}} \mathsf{sbavail}_{\mathsf{p},\mathsf{q},\mathcal{B}}(m, s') \Big) \\
& \vee \Big(\bigvee_{\substack{(s,\, \mathsf{p}\to\mathsf{t}:m',\, s')\in\Delta \\ \mathsf{t}\notin\mathcal{B} \\ \mathsf{t}\neq\mathsf{q}}} \mathsf{sbavail}_{\mathsf{p},\mathsf{q},\mathcal{B}}(m, s') \Big) \vee \Big(\bigvee_{\substack{(s,\, \mathsf{p}\to\mathsf{q}:m,\, s')\in\Delta \\ \mathsf{p}\notin\mathcal{B}}} \top \Big) \ .
\end{aligned}
$$

For bag networks, sends and receives between the same pair of participants with different message labels do not interfere with one another. Thus, the "skipping" condition is more permissive: not only can we skip messages between other sender and receiver pairs, we can also skip messages between the same sender receiver pair with different message labels.

Definition 4.29 (Receivable for bag)**.**

$$
\begin{aligned}
\mathsf{bagavail}_{\mathsf{p},\mathsf{q},\mathcal{B}}(m, s) \coloneqq_\mu \quad & \Big(\bigvee_{\substack{(s,\, \mathsf{r}\to\mathsf{t}:m',\, s')\in\Delta \\ \mathsf{r}\in\mathcal{B} \\ \mathsf{r}\neq\mathsf{p}\vee\mathsf{t}\neq\mathsf{q}\vee m'\neq m}} \mathsf{bagavail}_{\mathsf{p},\mathsf{q},\mathcal{B}\cup\{\mathsf{t}\}}(m, s') \Big) \\
& \vee \Big(\bigvee_{\substack{(s,\, \mathsf{r}\to\mathsf{t}:m',\, s')\in\Delta \\ \mathsf{r}\notin\mathcal{B} \\ \mathsf{r}\neq\mathsf{p}\vee\mathsf{t}\neq\mathsf{q}\vee m'\neq m}} \mathsf{bagavail}_{\mathsf{p},\mathsf{q},\mathcal{B}}(m, s') \Big) \vee \Big(\bigvee_{\substack{(s,\, \mathsf{p}\to\mathsf{q}:m,\, s')\in\Delta \\ \mathsf{p}\notin\mathcal{B}}} \top \Big) \ .
\end{aligned}
$$

From the above, it is clear that senderbox receivable implies peer-to-peer box receivable implies bag receivable. Because the receivable predicate appears in a negative position in Receive Coherence, and the other two Coherence Conditions characterizing implementability are the same, we obtain the following.

Lemma 4.30 (Implementability relationships)**.** *Any bag-implementable global protocol is peer-to-peer box-implementable, and any peer-to-peer box-implementable global protocol is senderbox-*

implementable.

The strictness of the inclusions is witnessed by examples G_a and G_b in Fig. 4.5 and Fig. 4.6. We note that the relationship between the considered network architectures induced by Lemma 4.30 coincide with those established by both [Chevrou et al. 2016] and [Giusto et al. 2023]. We revisit this point in further detail in §4.6.

We state our generalized Coherence Conditions and generalized preciseness theorem below.

Definition 4.31 (Generalized Coherence Conditions)**.** A protocol $\mathcal{S} = (S, \Gamma_{sync}, T, s_0, F)$ satisfies generalized Coherence Conditions under network architecture $\mathbb{A}$ when it satisfies Send Coherence, generalized Receive Coherence under $\mathbb{A}$, and No Mixed Choice.

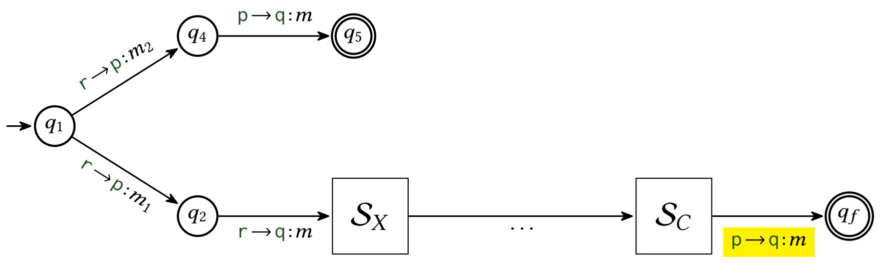
Theorem 4.32 (Preciseness of Generalized Coherence Conditions)**.** *Let $\mathcal{S}$ be a protocol and let $\mathbb{A}$ be a network architecture. Then, $\mathcal{S}$ is implementable under $\mathbb{A}$ if and only if it satisfies generalized Coherence Conditions under $\mathbb{A}$.*

4.5.1 Decidability and Complexity

Theorem 4.32 immediately yields a decision procedure for checking implementability of finite protocols modulo each of the six considered network architectures. We examine the complexity of problem for the finite fragment for each of these architectures in light of their receivability relationships. In Chapter 3 we showed that the co-NP completeness of peer-to-peer box implementability is determined by the co-NP completeness of deciding the avail predicate. It is easy to see that generalized Receive Coherence for each network architecture can still be checked in NP. For the lower bound, we show that the same construction with a small modification works for the other two notions of receivability.

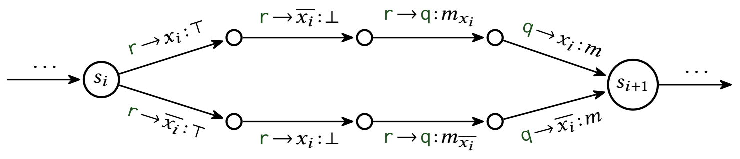
The proof of the co-NP lower bound works by a reduction from 3-SAT to implementability. The proof assumes a 3-SAT instance $\varphi = C_1 \wedge \ldots \wedge C_k$ with variables $x_1, \ldots, x_n$ and literals L_{ij}, denoting the jth literal of clause C_i, with $1 \leq i \leq k$ and $1 \leq j \leq 3$. From this, it constructs a global protocol $\mathcal{S}_\varphi$ such that ϕ is unsatisfiable iff $\mathcal{S}_\varphi$ is implementable. We summarize the construction pictorially in Fig. 4.7.

The construction relies on two gadgets: $\mathcal{S}_X$, a gadget that encodes a variable assignment to variables $x_1, \ldots, x_n$ (Fig. 4.8), and $\mathcal{S}_C$, a gadget that encodes literal selection for clauses $C_1, \ldots, C_k$ (Fig. 4.9). The highlighted message in Fig. 4.7 is available for participant q in q_4 if and only if φ is satisfiable. Consequently, protocol $\mathcal{S}_\varphi$ is non-implementable if and only if the highlighted message is available for participant q in q_4, if and only if φ is satisfiable. In order to show that the construction carries over to bag and senderbox networks, one is only required to analyze the unique message receptions that appear in each gadget. Thus, Fig. 4.8 and Fig. 4.9 each depict the smallest portion of each gadget necessary to establish our new complexity results.



$\mathsf{r}\to\mathsf{p}:m_2$
q_4
$\mathsf{p}\to\mathsf{q}:m$
q_5
q_1
$\mathsf{r}\to\mathsf{p}:m_1$
q_2
$\mathsf{r}\to\mathsf{q}:m$
$\mathcal{S}_X$
$\cdots$
$\mathcal{S}_C$
$\mathsf{p}\to\mathsf{q}:m$
q_f

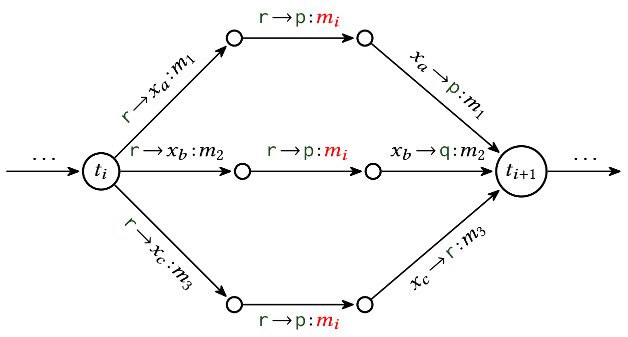
Figure 4.7: Illustration of 3-SAT reduction for implementability.



$\cdots$
s_i
$\mathsf{r}\to x_i:\top$
$\mathsf{r}\to\overline{x_i}:\bot$
$\mathsf{r}\to\mathsf{q}:m_{x_i}$
$\mathsf{q}\to x_i:m$
s_{i+1}
$\cdots$
$\mathsf{r}\to\overline{x_i}:\top$
$\mathsf{r}\to x_i:\bot$
$\mathsf{r}\to\mathsf{q}:m_{\overline{x_i}}$
$\mathsf{q}\to\overline{x_i}:m$

Figure 4.8: Illustration of variable assignment gadget $\mathcal{S}_X$.

Since both equivalences require reasoning about Receive Coherence, we must reconsider each in turn. First, we consider the bag network architecture. As established above, any message receivable in a peer-to-peer box network is also receivable in a bag network. Thus, we only need to show that the rest of $\mathcal{S}_\varphi$ is implementable, which amounts to checking that no other bag Receive Coherence violations occur. Participant r does not receive messages, and can thus



$\mathsf{r} \to \mathsf{p} : m_i$
$\mathsf{r} \to x_a : m_1$
$x_a \to \mathsf{p} : m_1$
$\mathsf{r} \to x_b : m_2$
$\mathsf{r} \to \mathsf{p} : m_i$
$x_b \to \mathsf{q} : m_2$
$\mathsf{r} \to x_c : m_3$
$x_c \to \mathsf{r} : m_3$
$\mathsf{r} \to \mathsf{p} : m_i$
t_i
t_{i+1}
...
...

Figure 4.9: Illustration of clause selection gadget $\mathcal{S}_C$. Highlighted parts indicate modification for generalized reduction.

be ignored. Unlike peer-to-peer FIFO Receive Coherence, bag Receive Coherence additionally constrains pairs of receptions from the same sender. For the variable participants in $\mathcal{S}_X$, each participant receives either a $\bot$ from r, or a $\top$ followed by an m message from q. Thus, bag Receive Coherence is satisfied. Inspecting $\mathcal{S}_C$, each variable participant only receives one kind of message, which is m from r, and if so it sends an m message to p. Thus, bag Receive Coherence is satisfied as well. Participant p is uninvolved in $\mathcal{S}_X$, but in $\mathcal{S}_C$ receives m messages from r which tells it to anticipate a message from some variable participant. The original encoding uses the same message payload from r to tell p to anticipate a message, but we can modify the construction to let r send p a message encoding precisely which variable participant to anticipate a message from. The eliminates what would otherwise constitute a bag RC violation for p, since r and the variable participants can overtake p's receptions. Finally, onto participant q, who is uninvolved in $\mathcal{S}_C$ and only involved in $\mathcal{S}_X$, q receives exactly n messages from r, that constitute n binary choices between receiving m_{x_i} and $m_{\bar{x}_i}$ interrupted by send events from q. Thus, we can conclude that the modified construction is non-implementable iff $\mathsf{bagavail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, q_3)$ holds in $\mathcal{S}_\varphi$ iff φ is satisfiable.

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

From this, we conclude that for all considered communication models, implementability remains co-NP complete for finite protocols.

Theorem 4.33. *Implementability is co-NP complete for bag and senderbox network architectures.*

4.6 Related Work

Network-parametric results from concurrency theory. Here, we discuss other results that are parametric in a choice of network architecture. Bollig et al. [2021] and Giusto et al. [2023] study the synchronizability problem for asynchronous communication models. Synchronizability in general asks whether an asynchronous implementation in the form of a communicating finite state machine can be soundly and completely approximated in terms of its "synchronous" executions, where "synchronous" has multiple interpretations. Because our global protocol semantics are synchronizable by definition, synchronizability is a necessary but insufficient condition for any candidate implementation of a global protocol. Bollig et al. [2021] presents a framework unifying several existing notions of synchronizability from the literature, such as universal and existential k-boundedness [Genest et al. 2006a], send-synchronizability [Basu and Bultan 2011], and weak k-synchrony [Bouajjani et al. 2018]. To complicate matters further, these existing notions of synchronizability are studied under different communication models, including peer-to-peer FIFO and mailbox, sometimes featuring unmatched send events. The proposed parametric framework in [Bollig et al. 2021] is based on MSO logic and special treewidth, and establishes decidability results of four notions of synchronizability for peer-to-peer and mailbox communication: weakly synchronous, weakly k-synchronous, existentially k-bounded, and universally

k-bounded.

Giusto et al. [2023] generalize Bollig et al. [2021] to additional communication models and notions of synchronizability. They define communication models as sets of message sequence charts (MSCs). Message sequence charts are partial order graphs on asynchronous events that contain a total order per participant, and thus represent sets of executions: MSCs can a priori specify unmatched sends and messages received out of order, but not branching behavior or recursion as supported by GCLTS. This broad notion of communication model thus also includes causally ordered communication (`co`), which is not purely expressible in an architectural manner, but rather should be considered a property of the specification. In other words, implementing a causally ordered network requires a form of global coordinator that has information about causal histories. In addition to five communication models considered in our work (`p2p,nn,onen,mb,asy`), [Giusto et al. 2023] also considers realizable with synchronous communication (`rsc`), which is excluded from our asynchronous implementation fragments.

The authors establish a strict hierarchical order on these seven communication models in terms of their linearizations: `com1` < `com2` means that any MSC specifying a linearization executable on `com1` must also specify a linearization executable on `com2`. The authors solve the $(\texttt{com}, C)$-synchronizability problem, where `com` is one of the seven communication models under consideration, and C is a bounding class of MSCs that resemble synchronous communication, including the aforementioned weakly synchronous and weakly k-synchronous [Bouajjani et al. 2018], universally and existentially k-bounded [Genest et al. 2006a]. The $(\texttt{com}, C)$-synchronizability problem asks whether all asynchronous executions of a given MSC under `com` are contained in the class C-synchronous.

Although our protocol semantics require existentially 1-bounded implementations, deciding whether a given implementation is existentially 1-bounded does not amount to deciding whether it implements a given global protocol. Non-implementable protocols can moreover have existentially 1-bounded canonical implementations, as evidenced by the global protocol in Fig. 4.10.

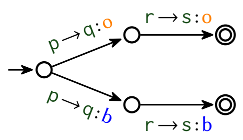


Figure 4.10: A non-implementable protocol with an existentially 1-bounded canonical implementation.

There exists no CLTS that implements this protocol, however the canonical implementation is existentially 1-bounded. Thus, existential 1-boundedness of the canonical implementation is a necessary but insufficient condition for implementability.

Giusto et al. [2023] show that because all communication models in com are MSO-definable, and Courcelle's theorem applies to any graph of bounded special treewidth, the (com, C)-synchronizability problem is decidable for any bounding class C that is MSO-definable and has bounded special treewidth. Similarly to our approach, their generalization rests upon identifying an abstract property of the communication models and bounding models considered: MSO-definability and treewidth boundedness.

Non-FIFO communication is considered alongside FIFO communication in [Lohrey 2003], which studies the safe realizability problem of HMSCs. The problem definition targets communicating, finite state machines, and thus safe realizability is equivalent to implementability in our setting. The class of HMSCs considered generalizes the finite fragment of global protocols considered in this work along two dimensions: firstly, while specifications are required to satisfy FIFO restrictions, they are not required to be 1-synchronous, meaning that they can specify e.g. unmatched sends and out-of-order receptions; secondly, branching choice is unrestricted, relaxing the sender-driven choice condition assumed by our work. Unlike our setting, however, HMSCs are not given infinite word semantics. As discussed in [Stutz 2024a], sender-driven choice is a boundary condition for decidability, and thus safe realizability is undecidable. Lohrey [2003] shows that all (un)decidability results generalize to non-FIFO communication, which is equivalent to our bag network architecture: the only ordering imposed is that messages are sent before they are received. Because MSCs can distinguish identical communication events using different event labels, the non-FIFO case additionally requires the assumption that MSCs do not specify message overtaking between the same pair of participants with the same message label. To transfer upper bounds of safe realizability for FIFO communication to non-FIFO communication, Lohrey makes use of a polynomial time construction that introduces a new participant (p, q, m) for each sender, receiver and message value triple, and replaces all $p \rightarrow q : m$ events in the HMSC with $p \rightarrow (p, q, m) : m$ followed by $(p, q, m) \rightarrow q : m$. Because messages sent are immediately acknowledged, realizability is preserved, and because messages carry no content and thus no ordering applies, the set of FIFO executions is equal to the set of non-FIFO executions. The constructions for lower bounds on FIFO communication already enjoy the property of messages sent being immediately acknowledged, and thus immediately transfer to the non-FIFO case. Our generalization of completeness shares a similar reasoning pattern to Lohrey's complexity generalizations, in that both work by appealing to the lowest common denominator of linearizations, namely 1-synchronous linearizations.

Chevrou et al. [2016] conduct a systematic study of the same seven communication models considered by [Giusto et al. 2023]. They establish a hierarchy that differs from [Giusto et al. 2023] in the placement of FIFO 1-n (senderbox) and FIFO n-1 (mailbox) communication: according to Chevrou et al. [2016], the two are incomparable, whereas according to Giusto et al. [2023], senderbox subsumes mailbox. This discrepancy arises from a difference in the definition of communication models, in particular, the difference between a universal and existential quantifier. Chevrou et al. define communication models as sets of linearizations, requiring that all members of the set satisfy the communication model, whereas Giusto et al. define communication models as sets of MSCs, requiring that there exists an MSC linearization that satisfies the communication model. Chevrou et al. focus more on establishing a taxonomy of communication models, and do not study decision problems concerning one or more of the communication models.

Exploiting synchrony in protocol verification. Next, we give an overview of verification approaches that leverage the attractive simplicity of synchrony in complementary ways to the aforementioned "top-down" approaches. The decision problems of boundedness and synchronizability have been thoroughly studied in the literature [Finkel and Lozes 2023; Delpy et al. 2024; Bollig et al. 2021], and are the inspiration behind several programming languages and verification frameworks [von Gleissenthall et al. 2019; Kragl et al. 2020, 2018; Zhang et al. 2024]. Zhang et al. [2024] present a framework, Kondo, for verifying safety properties of distributed protocols. Kondo encourages the user to state and prove correct a synchronous version of an asynchronous, distributed protocol, in which protocol transitions consist of a matching pair of send and receive events specified atomically, in addition to state updates. The synchronous protocol's safety can then be established using inductive invariants. Kondo classifies inductive invariants into "Regular Invariants" that capture protocol-agnostic properties such as the monotonicity of a history variable, and "Protocol Invariants", which are protocol-specific and contain key insights into why the protocol is correct. Given a synchronous specification and accompanying safety proof, Kondo then partially automates the construction of an asynchronous specification and accompanying safety proof in Dafny, with user assistance to complete proof obligations. The network architecture assumed by Kondo across all protocols is that of a global message soup, from which messages are deposited but never removed. Kondo provides empirical support that many protocols in practice, such as distributed locking and variants of Paxos, do not require more than a synchronous inductive invariant to be proven correct.

In contrast to Kondo, which takes a synchronous specification as its starting point, the work by von Gleissenthall et al. [2019] takes as input a collection of local implementations, and computes a synchronization. They present a synchronization algorithm that is sound and complete for a restricted class of local implementations called Stratified Pairwise Communication Protocols (SPCP). SPCPs are structured as a series of non-interfering rounds: this means, for example, that the SPCP formulation of two phase commit does not permit the coordinator to concurrently `issueVoteRequest` messages and receive `Vote` messages. Their definition of synchronizability differs from standard definitions [Finkel and Lozes 2023; Delpy et al. 2024; Bollig et al. 2021]: a program is synchronizable if all halting states are final and have empty message buffers, which resembles the standard definition of deadlock freedom. It is established by Finkel and Lozes [Finkel and Lozes 2023] that synchronizability is decidable for general bag CLTS.

In the context of asynchronous networks, the notion of indistinguishability by two implementations can be alternatively viewed as a commutativity relation: two asynchronous events commute if they can be reordered by a scheduler. The idea of using commutativity more generally to simplify program verification dates back to Lipton [Lipton 1975] and has been thoroughly investigated, especially for concurrent programs with shared memory, see [Farzan 2023] for an overview. With a widened space of commutativity relations, the challenge in employing them consists of identifying ones that are advantageous to verification. Farzan et al. [2023] show how to contextually exploit the program and property at hand to find the right commutativity relation.

Synchrony. Top-down verification for synchronous communication models has been an active topic of study, prominently in the form of synchronous multiparty session types [Udomsrirungruang and Yoshida 2025; Peters and Yoshida 2024; Chen et al. 2024; Ghilezan et al. 2019a]. In §4.7, we discuss variations of synchronous communication, Zielonka's asynchronous automata, and their connection to current and future work.

4.7 Discussion

We conclude this chapter with a discussion of synchronous communication models and our connection to Zielonka automata.

Synchronous communication. In this paper, we studied asynchronous communication models that are definable architecturally, i.e. in terms of communication topology and channel data structure. This is sometimes referred to as the “operational” view of communication models, in contrast to a denotational view, which includes communication models such as *causally ordering*. We have excluded synchronous communication models from our investigation. Within synchronous communication, a distinction is to be found between channel-less rendezvous synchronous communication, *à la* Zielonka’s asynchronous automata [Zielonka 1987], and synchronous communication that is in essence asynchronous communication with a global channel bound of size 1, *à la* rsc from [Chevrou et al. 2016] and [Giusto et al. 2023]. A rendezvous synchronous CLTS omits channel configurations, and replaces the transition relation with a single rule for $\vec{s} \xrightarrow{\mathsf{p}\to\mathsf{q}:m} \vec{s}\,'$, defined as:

$$(\vec{s}_{\mathsf{p}}, \mathsf{p} \triangleright \mathsf{q}!m, \vec{s}\,'_{\mathsf{p}}) \in \delta_{\mathsf{p}} \wedge (\vec{s}_{\mathsf{q}}, \mathsf{q} \triangleleft \mathsf{p}?m, \vec{s}\,'_{\mathsf{q}}) \in \delta_{\mathsf{q}} \wedge \forall\, \mathsf{r} \in \mathcal{P}.\ \mathsf{r} \neq \mathsf{p} \wedge \mathsf{r} \neq \mathsf{q} \implies \vec{s}_{\mathsf{r}} = \vec{s}\,'_{\mathsf{r}} \ .$$

In the rendezvous model, also known as the reader-writer model of synchrony, the sender and receiver in an event jointly decide on the transition to execute. Rendezvous synchrony in essence voids the designations of sender and receiver: both parties have equal power to select or veto a communication. In contrast, in rsc, the autonomy to send a message remains with the sender. To illustrate this point, we revisit the Send Validity violating protocol from [Li et al. 2023a], depicted in Fig. 4.11.

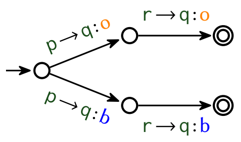


Figure 4.11: A global protocol $\mathcal{S}_{send}$ implementable under rendezvous synchrony, but not under rsc synchrony.

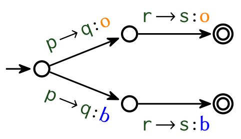


Figure 4.12: A global protocol $\mathcal{S}_{send'}$ not implementable under either rendezvous synchrony or rsc synchrony, but implementable by asynchronous automata.

$\mathcal{S}_{send}$ is implementable with a rendezvous synchronous CLTS, but non-implementable on any implementation of rsc synchrony. On the rsc model, participant r can send either o or b, and can thus yield the execution trace $\mathsf{p}\triangleright\mathsf{q}!\mathsf{o}\cdot\mathsf{q}\triangleleft\mathsf{p}?\mathsf{o}\cdot\mathsf{r}\triangleright\mathsf{q}!\mathsf{b}$; on the rendezvous model, q can communicate p and q's joint branch selection to r and veto r's proposal to fire the wrong transition.

Perhaps surprisingly, even if we completely hide p's choice from participants in the second transition, as shown in Fig. 4.12, $\mathcal{S}_{send'}$ is still implementable using Zielonka's asynchronous automata. This is because, as pointed out in [Akshay et al. 2013], asynchronous automata are not "purely" distributed: its transition relations and final states are specified globally. Thus, an implementation of $\mathcal{S}_{send'}$ can simply exclude the final configurations reached on the traces $\mathsf{p}\to\mathsf{q}:\mathsf{o}\cdot\mathsf{r}\to\mathsf{s}:\mathsf{b}$ and $\mathsf{p}\to\mathsf{q}:\mathsf{b}\cdot\mathsf{r}\to\mathsf{s}:\mathsf{o}$ from the set of final states. The global nature of asynchronous automata transitions and final states allows local participants to perform actions that are not visible to the language, potentially leading to deadlocking traces.

Akshay et al. [Akshay et al. 2013] define a fragment of "realistic" asynchronous automata that rules out this definitional loophole. Realistic asynchronous automata satisfy transition determinism: there is at most one post-state for each pre-state and transition label pair, deadlock-freedom: each reachable state is either final or has outgoing transitions, and local acceptance: final states only contain states that are final in each participant's local automaton. The authors then consider the realizability problem of regular, I-closed global specifications targeting realistic asynchronous automata. Zielonka's theorem, established in [Zielonka 1987] and optimized in [Cori et al. 1993; Diekert and Rozenberg 1995; Mukund and Sohoni 1997; Genest et al. 2010], states that every regular, I-closed language has an implementation in the form of an asynchronous automaton. Thus, realizability in [Akshay et al. 2013] amounts to identifying additional restrictions on the global specification such that the asynchronous automaton obtained from an optimized, standard construction [Genest et al. 2010] can be transformed to satisfy the realistic assumptions.

Notably, the Send Coherence condition employed in [Li et al. 2025b] and this work can be viewed as a special instance of the third semantical condition characterizing realizability, *causally closed* [Akshay et al. 2013, Definition 9, (LC3)]. Asynchronous automata further generalize the sender and receiver of a communication event a to a set of participants, denoted $\mathrm{dom}(a)$. (LC3) states that whenever w is a prefix of the specification language, given an event c, if for every participant in $\text{dom}(c)$ there exists another prefix v_p such that w and v_p are identical according to p's local view, then wc is also a prefix of the specification language. Send Coherence is (AC3) applied only to asynchronous send events, whose only participant is the sender.

Given that in both our work and [Akshay et al. 2013], the implementation models share salient similarities, and the realizability characterizations share a salient condition despite the apparent difference between synchronous vs. asynchronous communication, we are optimistic about the possibility of an implementability characterization in our setting that simultaneously handles synchronous and asynchronous communication.

Symbolic protocols. Finally, we observe that our generalized CC can be used to derive symbolic algorithms for checking implementability of symbolic protocols with dependent refinements, featuring infinite states and data, as demonstrated in Section 3.4 and implemented in [Li et al. 2025a]. One simply needs to update Symbolic Receive Coherence with the generalized condition, and replace the symbolic avail predicate with the appropriate version for the network architecture under consideration, as detailed in §4.5.

5 | Synthesis

5.1 Introduction

The synthesis problem asks to compute a candidate implementation from an implementable global protocol. We show that synthesis amounts to a generalized subset construction modulo the global protocol fragment, which can be computed in PSPACE. We present a sound and complete synthesis algorithm for finite protocols using a standard automata-theoretic construction. We then discuss fragments of symbolic protocols for which synthesis is decidable.

5.2 Synthesizing Finite Implementations

The construction of the candidate implementation for a finite protocol is carried out in two steps. First, for each participant $\mathsf{p} \in \mathcal{P}$, we define an intermediate state machine $\mathrm{GAut}(\mathbf{G})\!\downarrow_{\mathsf{p}}$ that is a homomorphism of $\mathrm{GAut}(\mathbf{G})$. We call $\mathrm{GAut}(\mathbf{G})\!\downarrow_{\mathsf{p}}$ the *projection by erasure* for p, defined below.

Definition 5.1 (Projection by Erasure)**.** Let $\mathbf{G}$ be a global type with state machine $\mathrm{GAut}(\mathbf{G}) = (Q_{\mathrm{G}}, \Gamma, \delta_{\mathrm{G}}, q_{0,\mathrm{G}}, F_{\mathrm{G}})$. For each role $\mathsf{p} \in \mathcal{P}$, we define the state machine $\mathrm{GAut}(\mathbf{G})\!\downarrow_{\mathsf{p}} = (Q_{\mathrm{G}}, \Sigma_{\mathsf{p}} \uplus \{\varepsilon\}, \delta_{\downarrow}, q_{0,\mathrm{G}}, F_{\mathrm{G}})$ where $\delta_{\downarrow} := \{q \xrightarrow{\mathtt{split}(a)\Downarrow_{\Sigma_{\mathsf{p}}}} q' \mid q \xrightarrow{a} q' \in \delta_{\mathrm{G}}\}$. By definition of $\mathtt{split}(\text{-})$, it holds that $\mathtt{split}(a)\Downarrow_{\Sigma_{\mathsf{p}}} \in \Sigma_{\mathsf{p}} \uplus \{\varepsilon\}$.

Then, we determinize $\mathrm{GAut}(\mathbf{G})\!\downarrow_{\mathsf{p}}$ via a standard subset construction to obtain a deterministic local state machine for p.

Definition 5.2 (Subset Construction)**.** Let $\mathbf{G}$ be a global type and p be a role. Then, the *subset construction* for p is defined as

$$\mathscr{C}(\mathbf{G}, \mathsf{p}) = \big(Q_{\mathsf{p}}, \Sigma_{\mathsf{p}}, \delta_{\mathsf{p}}, s_{0,\mathsf{p}}, F_{\mathsf{p}}\big) \text{ where}$$

- $\delta(s, a) \coloneqq \{q' \in Q_{\mathbf{G}} \mid \exists q \in s, q \xrightarrow{a}\xrightarrow{\varepsilon}{}^{*} q' \in \delta_{\downarrow}\}$, for every $s \subseteq Q_{\mathbf{G}}$ and $a \in \Sigma_{\mathsf{p}}$
- $s_{0,\mathsf{p}} \coloneqq \{q \in Q_{\mathbf{G}} \mid q_{0,\mathbf{G}} \xrightarrow{\varepsilon}{}^{*} q \in \delta_{\downarrow}\}$,
- $Q_{\mathsf{p}} \coloneqq \mathrm{lfp}^{\subseteq}_{\{s_{0,\mathsf{p}}\}} \lambda Q.\, Q \cup \{\delta(s, a) \mid s \in Q \wedge a \in \Sigma_{\mathsf{p}}\} \setminus \{\emptyset\}$, and
- $\delta_{\mathsf{p}} \coloneqq \delta|_{Q_{\mathsf{p}} \times \Sigma_{\mathsf{p}}}$
- $F_{\mathsf{p}} \coloneqq \{s \in Q_{\mathsf{p}} \mid s \cap F_{\mathbf{G}} \neq \emptyset\}$

Note that the construction ensures that Q_{p} only contains subsets of $Q_{\mathbf{G}}$ whose states are reachable via the same traces, i.e. we typically have $|Q_{\mathsf{p}}| \ll 2^{|Q_{\mathbf{G}}|}$.

Lemma 5.3. *Let* $\mathbf{G}$ *be a global type,* r *be a role, and* $\mathscr{C}(\mathbf{G}, \mathsf{r})$ *be its* subset construction*. If* w *is a trace of* $\mathsf{GAut}(\mathbf{G})$*,* $\mathtt{split}(w)\Downarrow_{\Sigma_{\mathsf{r}}}$ *is a trace of* $\mathscr{C}(\mathbf{G}, \mathsf{r})$*. If* u *is a trace of* $\mathscr{C}(\mathbf{G}, \mathsf{r})$*, there is a trace* w *of* $\mathsf{GAut}(\mathbf{G})$ *such that* $\mathtt{split}(w)\Downarrow_{\Sigma_{\mathsf{r}}} = u$*. It holds that* $\mathcal{L}(\mathbf{G})\Downarrow_{\Sigma_{\mathsf{r}}} = \mathcal{L}(\mathscr{C}(\mathbf{G}, \mathsf{r}))$*.*

Using this lemma, we show that the CSM $\{\!\{\mathscr{C}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$ preserves all behaviors of $\mathbf{G}$.

Lemma 5.4. *For all global types* $\mathbf{G}$*,* $\mathcal{L}(\mathbf{G}) \subseteq \mathcal{L}(\{\!\{\mathscr{C}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}})$*.*

We briefly sketch the proof here. Given that $\{\!\{\mathscr{C}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$ is deterministic, to prove language inclusion it suffices to prove the inclusion of the respective prefix sets:

$$\mathrm{pref}(\mathcal{L}(\mathbf{G})) \subseteq \mathrm{pref}(\mathcal{L}\{\!\{\mathscr{C}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}})$$

Let w be a word in $\mathcal{L}(\mathbf{G})$. If w is finite, membership in $\mathcal{L}(\{\!\{\mathscr{C}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}})$ is immediate from the claim above. If w is infinite, we show that w has an infinite run in $\{\!\{\mathscr{C}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$ using

König's Lemma. We construct an infinite graph $\mathcal{G}_w(V, E)$ with $V \coloneqq \{v_\rho \mid \texttt{trace}(\rho) \leq w\}$ and $E \coloneqq \{(v_{\rho_1}, v_{\rho_2}) \mid \exists\, x \in \Sigma.\ \texttt{trace}(\rho_2) = \texttt{trace}(\rho_1) \cdot x\}$. Because $\{\!\{\mathscr{C}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$ is deterministic, $\mathcal{G}_w$ is a tree rooted at v_ε, the vertex corresponding to the empty run. By König's Lemma, every infinite tree contains either a vertex of infinite degree or an infinite path. Because $\{\!\{\mathscr{C}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$ consists of a finite number of communicating state machines, the last configuration of any run has a finite number of next configurations, and $\mathcal{G}_w$ is finitely branching. Therefore, there must exist an infinite path in $\mathcal{G}_w$ representing an infinite run for w, and thus $w \in \mathcal{L}(\{\!\{\mathscr{C}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}})$.

The proof of the inclusion of prefix sets proceeds by structural induction and primarily relies on Lemma 5.3 and the fact that all prefixes in $\mathcal{L}(\mathbf{G})$ respect the order of send before receive events.

5.3 Synthesizing General Implementations

Recall from our proof of soundness in Chapter 3 that 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 5.5 (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 [D'Antoni and Veanes 2017; Shen et al. 2023] and certain classes of timed and register automata [Bertrand et al. 2015; Clemente et al. 2022], one can simply use off-the-shelf determinization algorithms to compute canonical implementations [Veanes et al. 2010; Veanes and Bjørner 2012; Bertrand et al. 2018]. Moreover, one implication of our discussion of controllability in §4.3, is that the chosen network architecture for the implementability problem does not affect synthesis: for all considered network architectures, the global protocol is implementable if and only if the canonical implementation obtainable by local projection implements it.

6 | Subtyping

6.1 Introduction

In this chapter, we study the subtyping problem for multiparty session types with sender-driven choice. In MST frameworks, the implementability and synthesis problems are solved simultaneously using a *projection operator*, which is a partial map from global types to a collection of local implementations. Projection operators compute a correct implementation for a given global type if one exists. However, projection operators only compute one candidate out of many possible implementations for a given global type, which narrows the usability of MST frameworks. As we demonstrate below, substituting this candidate can in some cases achieve an exponential reduction in the size of the local implementation. Furthermore, applications may sometimes require that an implementation produce only a subset of the global type's specified behaviors. We refer to this property as *subprotocol fidelity*. For example, a general client-server protocol may customize the set of requests it handles to the specific devices it runs on. Subtyping reintroduces this flexibility into MST frameworks, by characterizing when an implementation can replace another while preserving desirable correctness guarantees.

Formally, a subtyping relation is a reflexive and transitive relation that respects Liskov and Wing's substitution principle [Liskov and Wing 1994]: T' is a subtype of T when T' can be *safely* used in any context that expects a term of type T. While implementability for MSTs was originally defined on syntactic local types [Honda 1993; Honda et al. 2008], other implementation models have since been investigated, including communicating session automata [Deniélou and Yoshida 2012] and behavioral contracts [Castagna et al. 2009]. We motivate our work with the observation that a subtyping relation is only as powerful as its notion of safety, and the expressivity of its underlying implementation model. Existing subtyping relations adopt a notion of safety that is agnostic to a global specification. For example, [Barbanera and De'Liguoro 2015; Bernardi and Hennessy 2016] define safety as the successful completion of a single role in binary sessions, [Lange and Yoshida 2019] defines safety as eventual reception and progress of all roles in multiparty sessions, and [Ghilezan et al. 2019b] defines safety as the termination of all roles in multiparty sessions. As a result, these subtyping relations eagerly reject subtypes that are viable for the specific global type at hand. In addition, existing implementation models are restricted to local types with *directed choice* for branching, or equivalent representations thereof [Bravetti and Zavattaro 2021], which prohibit a role from sending messages to or receiving messages from different participants in a choice. This restrictiveness undermines the flexibility that subtyping is fundamentally designed to provide.

In this chapter, we present a subtyping relation that extends prior work along both dimensions. We define a stronger notion of safety with respect to a given global type: a substitution is safe if in all *well-behaved* contexts, the resulting implementation satisfies both deadlock freedom and subprotocol fidelity. We assume an implementation model of unrestricted communicating state machines (CSMs) [Brand and Zafiropulo 1983] communicating via FIFO channels, which subsumes implementation models in prior work [Lange and Yoshida 2019; Ghilezan et al. 2019b; Cutner et al. 2022]. We demonstrate that this generalization renders existing subtyping relations which are precise for a restrictive implementation model incomplete. As a result of both extensions, our subtyping relation requires reasoning about available messages [Majumdar et al. 2021a] for completeness, a novel feature that is absent from existing subtyping relations.

Our result applies to global types with *sender-driven* choice, which generalize global types from their original formulation with directed choice [Honda et al. 2008], and borrows insights from the sound and complete projection operator for this class of global types proposed in [Li et al. 2023a].

Contributions. In this chapter, we present the first precise subtyping relation that guarantees deadlock freedom and subprotocol fidelity with respect to a global type, and that assumes an unrestricted, asynchronous CSM implementation model. We solve the *Protocol Verification* problem and the *Protocol Refinement* problem with respect to global type $\mathbf{G}$ and a set of roles $\mathcal{P}$:

1. *Protocol Verification*: Given a CSM $\mathcal{A}$, does $\mathcal{A}$ implement $\mathbf{G}$?

2. *Protocol Refinement*: Let p be a role and let B be a safe implementation for p in any well-behaved context for $\mathbf{G}$. Given A, can A safely replace B in any well-behaved context for $\mathbf{G}$?

We exploit the connection between MST subtyping and CSM refinement to formulate concise conditions that are directly checkable on candidate state machines. Using this characterization, we show that both problems are decidable in co-NP, revising an incorrect complexity result published in [Li et al. 2024].

6.2 Motivation

We first showcase that sound and complete projection operators can yield local implementations that are exponential in the size of its global type, but can be reduced to constant size by subtyping. We then demonstrate the restrictiveness of existing subtyping relations both in terms of their notion of safety and their implementation model.

Subset projection with exponentially many states. We first construct a family of implementable global types $\mathbf{G}_n$ for $n \in \mathbb{N}$ such that $\mathbf{G}_n$ has size linear in n and the deterministic finite state machine for q that recognizes the projection of the global language onto q's alphabet Σ_{q}, denoted $\mathcal{L}(\mathbf{G}_n)\Downarrow_{\Sigma_{\mathsf{q}}}$, has size exponential in n.

The construction of the $\mathbf{G}_n$'s builds on the regular expression $(a^*(ab^*)^n a)^*$, which can only be recognized by a deterministic finite state machine that grows exponentially with n [Ellul et al. 2005, Thm. 11].

First, we construct the part for $(ab^*)^i a$ recursively. In global types, $\mathsf{p} \to \mathsf{q} : m$ denotes role p sending a message m to role q, $+$ denotes choice, μt binds a recursion variable t that can be used in the continuation, and 0 denotes termination.

$$G_i \coloneqq \mathsf{p} \to \mathsf{q} : a.\, \mu t_{3,i}.\, + \begin{cases} \mathsf{p} \to \mathsf{r} : m_3.\, \mathsf{p} \to \mathsf{q} : b.\, t_{3,i} \\ \mathsf{p} \to \mathsf{r} : n_3.\, G_{i-1} \end{cases} \quad \text{for } i > 0 \quad \text{and} \quad G_0 \coloneqq \mathsf{p} \to \mathsf{q} : a.\, t_1$$

Here, each G_i for $i > 0$ generates (ab^*) and G_0 adds the last a. Role p's choice to send either m_3 or n_3 to r respectively encodes the choice to continue iterating b's or to stop in b^*; q however, is not involved in this exchange and thus q's local language is isomorphic to $(ab^*)^i a$.

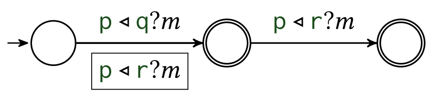
Next, we define some scaffolding $G(\text{-})$ for the outermost Kleene Star and the first a^*:

$$G(G') \coloneqq \mu t_1.\; + \begin{cases} \mathsf{p} \to \mathsf{r} : m_1.\, \mu t_2.\, + \begin{cases} \mathsf{p} \to \mathsf{r} : m_2.\, \mathsf{p} \to \mathsf{q} : a.\, t_2 \\ \mathsf{p} \to \mathsf{r} : n_2.\, G' \end{cases} \\ \mathsf{p} \to \mathsf{r} : n_1.\, 0 \end{cases} .$$

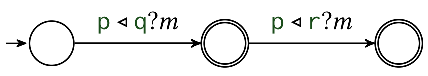
We combine both to obtain the family $\mathbf{G}_n \coloneqq G(G_n)$.

As $\mathbf{G}_n$ is implementable, the subset projection [Li et al. 2023a] for each role is defined. One feature of the implementations computed by this projection operator is *local language preservation*, meaning that the language recognized by the local implementation is precisely the projection of the global language onto its alphabet, e.g. $\mathcal{L}(\mathbf{G}_n)\Downarrow_{\Sigma_\mathsf{q}}$ for role q with alphabet Σ_q. In this case, because $\mathcal{L}(\mathbf{G}_n)\Downarrow_{\Sigma_\mathsf{q}}$ can only be recognized by a deterministic finite state machine with size exponential in n, the corresponding local language preserving implementation also has size exponential in n.

However, not all implementations need to satisfy local language preservation. Consider the type $\mu t.(\mathsf{p} \to \mathsf{q} : \mathsf{o}.\, t + \mathsf{p} \to \mathsf{q} : \mathsf{b}.\, 0)$. The projection of the global language onto q limits q to only



(a) *A*



(b) *B*

Figure 6.1: Two state machines for role q

receiving a sequence of o messages terminated by a b message. However, an implementation for q can rely on p to send correct sequences of messages, and instead accept any message that it receives. A similar pattern arises in the family $\mathbf{G}_n$, where the exponentially-sized implementation for role q can simply be substituted with an automaton that allows to receive any message from p.

The restrictiveness of existing MST subtyping relations. Consider the two implementations for role p, represented as finite state machines *A* and *B* in Figs. 6.1a and 6.1b. State machine *A* embodies the idea of input covariance [Gay and Hole 2005] by adding receive actions, namely $\boxed{\mathsf{p} \triangleleft \mathsf{q}?m}$, which denotes role p receiving a message *m* from role q. But is it the case that *A* is a subtype of *B*? A preliminary answer based on prior work [Lange and Yoshida 2016; Ghilezan et al. 2019b] is *no*, for the reason that *A* falls outside of the implementation models considered in these works: the initial state in *A* contains outgoing receive transitions from two distinct senders, q and r, and one of the final states contains an outgoing transition. Thus, there exists no local type representation of *A*.

As a first step, let us generalize the implementation model to machines with arbitrary finite state control, and revisit the question. It turns out that the answer now depends on what protocol role p, alongside the other roles in the context, is following. Consider the two global types

$$\mathbf{G}_1 \coloneqq \mathsf{q} \to \mathsf{p} : m.\, \mathsf{r} \to \mathsf{p} : m.\, 0 \qquad \text{and} \qquad \mathbf{G}_2 \coloneqq \mathsf{q} \to \mathsf{p} : m.\, 0 \quad .$$

We observe that *A* is a subtype of *B* under the context of $\mathbf{G}_2$, but not under the context of $\mathbf{G}_1$. Suppose that roles q and r are both following $\mathbf{G}_1$, and thus both roles send a message *m* to p. Under asynchrony, the two messages can arrive in p's channel in any order; this holds even in a synchronous setting. Therefore, there exists an execution trace in which p takes the transition labeled $\boxed{\mathsf{p} \triangleleft \mathsf{r}?m}$ in A and first receives from r. Role p then finds itself in a final state with a pending message from q that it is unable to receive, thus causing a deadlock in the CSM. On the other hand, if q were following $\mathbf{G}_2$, the addition of the receive transition $\boxed{\mathsf{p} \triangleleft \mathsf{r}?m}$ is safe because it is never enabled, and thus A can safely compose with any context following $\mathbf{G}_2$ without violating protocol fidelity and deadlock freedom.

6.3 Deciding Protocol Verification

Protocol Verification asks: Given a CSM $\mathcal{A}$, does $\mathcal{A}$ implement $\mathbf{G}$? For two CSMs $\mathcal{A}$ and $\mathcal{B}$, we say that $\mathcal{A}$ refines $\mathcal{B}$ if and only if every trace in $\mathcal{A}$ is a trace in $\mathcal{B}$, and a trace in $\mathcal{A}$ terminates maximally in $\mathcal{A}$ if and only if it terminates maximally in $\mathcal{B}$. If $\mathcal{A}$ and $\mathcal{B}$ refine each other, we say that they are equivalent. Further, in the case that $\mathcal{B}$ is deadlock-free, one can simplify the condition to the following: every trace in $\mathcal{A}$ is a trace in $\mathcal{B}$, and if a trace terminates in $\mathcal{A}$, then it terminates in $\mathcal{B}$ and is maximal in $\mathcal{A}$.

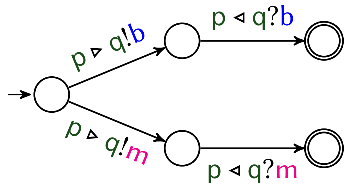
Using the fact that $\{\!\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\!\}_{\mathsf{p}\in\mathcal{P}}$ is an implementation for $\mathbf{G}$, we can recast *Protocol Verification* in terms of CSM refinement. Therefore, the question amounts to asking whether $\mathcal{A}$ and $\{\!\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\!\}_{\mathsf{p}\in\mathcal{P}}$ are equivalent.

Our goal is then to present a characterization C_1 that satisfies the following:

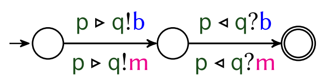
Theorem 6.1. *Let $\mathbf{G}$ be an implementable global type and $\mathcal{A}$ be a CSM. Then, the subset construction $\{\!\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\!\}_{\mathsf{p}\in\mathcal{P}}$ and $\mathcal{A}$ are equivalent if and only if C_1 is satisfied.*

We motivate our characterization for *Protocol Verification* using a series of examples. Consider the following simple global type $\mathbf{G}_1$:

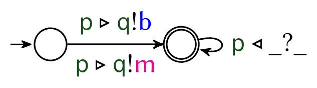
$$\mathbf{G}_1 := + \begin{cases} \mathsf{p} \to \mathsf{q} : \mathsf{b}.\ \mathsf{q} \to \mathsf{p} : \mathsf{b}.\ 0 \\ \mathsf{p} \to \mathsf{q} : \mathsf{m}.\ \mathsf{q} \to \mathsf{p} : \mathsf{m}.\ 0 \end{cases}$$



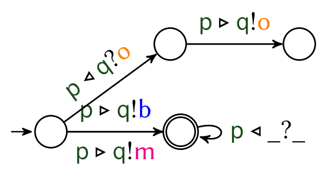
(a) $\mathscr{P}(\mathrm{G}_1, \mathsf{p})$



(b) A_1

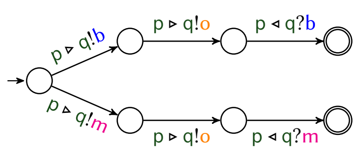


(c) A_2

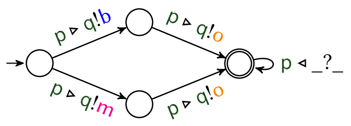


(d) A_3

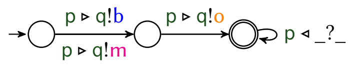
Figure 6.2: Subset construction of G_1 onto p and three alternative implementations



(a) $\mathscr{P}(\mathrm{G}_2, \mathsf{p})$



(b) A_4



(c) A_5

Figure 6.3: Subset construction of G_2 onto p and two alternative implementations

This global type is trivially implementable; the subset construction for role q obtained by the projection operator in [Li et al. 2023a] is depicted in Fig. 6.2a. Clearly, in any CSM implementing G_1, the subset construction can be replaced with the more compact state machine A_1, shown in Fig. 6.2b.

For a local state machine in a CSM, control flow is determined by both the local transition relation and the global channel state. However, in some cases, the local information is redundant: the role's channel contents alone are enough to enforce that it produces the correct behaviors. In the example above, after p chooses to send q either m or b, q will guarantee that the correct message, i.e. the same one, is sent back to p. Role p's state machine can rely on its channel contents to follow the protocol – it does not need separate control states for each message. In fact, we can further replace p's control states after sending with an accepting universal receive state, as shown in A_2 in Fig. 6.2c. Finally, we can add send transitions from unreachable states, as shown in A_3 in Fig. 6.2d.

Similar patterns arise for send actions. Consider the following variation of the first global type, $\mathbf{G}_2$:

$$\mathbf{G}_2 \coloneqq + \begin{cases} \mathsf{p} \to \mathsf{q} : \mathsf{b}.\ \mathsf{p} \to \mathsf{r} : \mathsf{o}.\ \mathsf{q} \to \mathsf{p} : \mathsf{b}.\ 0 \\ \mathsf{p} \to \mathsf{q} : \mathsf{m}.\ \mathsf{p} \to \mathsf{r} : \mathsf{o}.\ \mathsf{q} \to \mathsf{p} : \mathsf{m}.\ 0 \end{cases}$$

The subset construction from [Li et al. 2023a] yields the state machine for p shown in Fig. 6.3a.

Our reasoning above shows that A_4, depicted in Fig. 6.3b, is a correct alternative implementation for p. Now observe that the pre-states of the two $\mathsf{p} \triangleright \mathsf{q}!\mathsf{o}$ transitions can be collapsed because their continuations are identical. This yields another correct alternative implementation A_5, shown in Fig. 6.3c.

Informally, the subset construction takes a “maximalist” approach, creating as many distinct states as possible from the global type, and checking whether they are enough to guarantee that the role behaves correctly. However, sometimes this maximalism creates redundancy: just because two states are distinct according to the global type does not mean they need to be. In these cases, an implementation has the flexibility to merge certain distinct states together, or add transitions to a state. We wish to precisely characterize when such modifications to local state machines preserve protocol fidelity and deadlock freedom.

Our conditions for C_1 are derived from the Send and Receive Validity conditions that precisely characterize implementability for global types, given in [Li et al. 2023a]. We present relevant definitions below.

Definition 6.2 (Available messages [Majumdar et al. 2021a])**.** The set of available messages is recursively defined on the structure of the global type. For completeness, we need to unfold the distinct recursion variables once. For this, we define a map $get\mu$ from variable to subterms and write $get\mu_{\mathbf{G}}$ for $get\mu(\mathbf{G})$:

$$get\mu(0) \coloneqq [\,] \qquad get\mu(t) \coloneqq [\,] \qquad get\mu(\mu t.G) \coloneqq [t \mapsto G] \cup get\mu(G)$$

$$get\mu(\textstyle\sum_{i\in I} \mathsf{p} \to \mathsf{q}_i : m_i.G_i) \coloneqq \bigcup_{i\in I} get\mu(G_i)$$

The function $M^{\mathcal{B},T}_{(\text{-}\ldots)}$ keeps a set of unfolded variables T, which is empty initially.

$$M^{\mathcal{B},T}_{(0\ldots)} \coloneqq \emptyset \qquad M^{\mathcal{B},T}_{(\mu t.G\ldots)} \coloneqq M^{\mathcal{B},T\cup\{t\}}_{(G\ldots)} \qquad M^{\mathcal{B},T}_{(t\ldots)} \coloneqq \begin{cases} \emptyset & \text{if } t \in T \\ M^{\mathcal{B},T\cup\{t\}}_{(get\mu_{\mathrm{G}}(t)\ldots)} & \text{if } t \notin T \end{cases}$$

$$M^{\mathcal{B},T}_{(\sum_{i\in I} \mathsf{p}\to\mathsf{q}_i{:}m_i.G_i\ldots)} \coloneqq \begin{cases} \bigcup_{i\in I, m\in\mathcal{V}} (M^{\mathcal{B},T}_{(G_i\ldots)} \setminus \{\mathsf{p} \triangleright \mathsf{q}_i!m\}) \cup \{\mathsf{p} \triangleright \mathsf{q}_i!m_i\} & \text{if } \mathsf{p} \notin \mathcal{B} \\ \bigcup_{i\in I} M^{\mathcal{B}\cup\{\mathsf{q}_i\},T}_{(G_i\ldots)} & \text{if } \mathsf{p} \in \mathcal{B} \end{cases}$$

We write $M^{\mathcal{B}}_{(G\ldots)}$ for $M^{\mathcal{B},\emptyset}_{(G\ldots)}$. If $\mathcal{B}$ is a singleton set, we omit set notation and write $M^{\mathsf{p}}_{(G\ldots)}$ for $M^{\{\mathsf{p}\}}_{(G\ldots)}$.

Intuitively, the available messages definition captures all of the messages that can be at the head of their respective channels when a particular role is blocked from taking further transitions.

For notational convenience, we define the *origin* and *destination* of a transition generalized from the subset construction automaton.

Definition 6.3 (Transition Origin and Destination)**.** Let $\mathbf{G}$ be a global type and let $\delta_{\downarrow}$ be the transition relation of $\mathrm{GAut}(\mathbf{G}){\downarrow}_{\mathsf{p}}$. For $x \in \Sigma_{\mathsf{p}}$ and $s, s' \subseteq Q_{\mathbf{G}}$, we define the set of *transition origins* tr-orig($s \xrightarrow{x} s'$) and *transition destinations* tr-dest($s \xrightarrow{x} s'$) as follows:

$$\text{tr-orig}(s \xrightarrow{x} s') \coloneqq \{G \in s \mid \exists G' \in s'.\, G \xrightarrow{x}{}^{*} G' \in \delta_{\downarrow}\} \text{ and}$$
$$\text{tr-dest}(s \xrightarrow{x} s') \coloneqq \{G' \in s' \mid \exists G \in s.\, G \xrightarrow{x}{}^{*} G' \in \delta_{\downarrow}\} \ .$$

In Chapter 3, we showed that $\mathbf{G}$ is implementable if and only if the subset construction CSM $\{\!\{\mathscr{C}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p}\in\mathcal{P}}$ satisfies Send Validity, Receive Validity and No Mixed Choice for each $\mathscr{C}(\mathbf{G}, \mathsf{p})$.

Definition 6.4 (Send Validity)**.** $\mathscr{C}(\mathbf{G}, \mathsf{p})$ satisfies *Send Validity* iff every send transition $s \xrightarrow{x} s' \in \delta_{\mathsf{p}}$ is enabled in all states contained in s:

$$\forall s \xrightarrow{x} s' \in \delta_{\mathsf{p}}.\ x \in \Sigma_{\mathsf{p},!} \implies \text{tr-orig}(s \xrightarrow{x} s') = s \ .$$

Definition 6.5 (Receive Validity). $\mathscr{C}(\mathbf{G}, \mathsf{p})$ satisfies *Receive Validity* iff no receive transition is enabled in an alternative continuation that originates from the same source state:

$$\forall s \xrightarrow{\mathsf{p} \triangleleft \mathsf{q}_1 ? m_1} s_1,\ s \xrightarrow{\mathsf{p} \triangleleft \mathsf{q}_2 ? m_2} s_2 \in \delta_{\mathsf{p}}.$$
$$\mathsf{q}_1 \neq \mathsf{q}_2 \implies \forall\, G_2 \in \text{tr-dest}(s \xrightarrow{\mathsf{p} \triangleleft \mathsf{q}_2 ? m_2} s_2).\ \mathsf{q}_1 \triangleright \mathsf{p}!m_1 \notin M^{\mathsf{p}}_{(G_2 \ldots)} \ .$$

We wish to adapt these conditions to define C_1 on arbitrary state machines, not the subset construction for each participant.

We first present a *state decoration* function which maps local states in an arbitrary deterministic finite state machine to sets of global states in $\mathbf{G}$. Intuitively, state decoration captures all global states that can be reached in the projection by erasure automaton $\text{GAut}(\mathbf{G})\!\downarrow_{\mathsf{q}}$ on the same prefixes that reach the present state in the local state machine.

Definition 6.6 (State decoration with respect to $\mathbf{G}$). Let $\mathsf{p} \in \mathcal{P}$ be a role and let $A = (Q, \Sigma_{\mathsf{p}}, s_0, \delta, F)$ be a deterministic finite state machine for p. Let $\text{GAut}(\mathbf{G})\!\downarrow_{\mathsf{p}} = (Q_{\mathbf{G}}, \Sigma_{\mathsf{p}} \uplus \{\varepsilon\}, \delta_{\downarrow}, q_{0,\mathbf{G}}, F_{\mathbf{G}})$ be p's projection by erasure state machine for $\mathbf{G}$. We define a total function $d_{\mathbf{G},A} : Q \to 2^{Q_{\mathbf{G}}}$ that maps each state in A to a subset of states in $\text{GAut}(\mathbf{G})\!\downarrow_{\mathsf{p}}$ such that:

$$d_{\mathbf{G},A,\mathsf{p}}(s) = \{q \in Q_{\mathbf{G}} \mid \exists u \in \Sigma_{\mathsf{p}}^{*}.\ s_0 \xrightarrow{u}{}^{*} s \in \delta \wedge q_{0,\mathbf{G}} \xrightarrow{u}{}^{*} q \in \delta_{\downarrow}\} \ .$$

We refer to $d_{\mathbf{G},A,\mathsf{p}}(s)$ as the *decoration set* of s, and omit the subscripts $\mathbf{G}, A, \mathsf{p}$ when clear from context.

Remark 6.7. Note that the subset construction can be viewed as a special state machine for which the state decoration function is the identity function. In other words, for all $s \in Q_{\mathsf{p}}$ where Q_{p} is the set of states of $\mathscr{C}(\mathbf{G}, \mathsf{p})$, $d(s) = s$.

We are now equipped to present C_1.

Definition 6.8 (C_1)**.** Let $\mathbf{G}$ be a global type and $\mathcal{A}$ be a CSM. C_1 is satisfied when for all $\mathsf{p} \in \mathcal{P}$, with $A_\mathsf{p} = (Q_\mathsf{p}, \Sigma_\mathsf{p}, \delta_\mathsf{p}, s_{0,\mathsf{p}}, F_\mathsf{p})$ denoting the state machine for p in $\mathcal{A}$, the following conditions hold:

- *Send Decoration Validity*:

  every send transition $s \xrightarrow{x} s' \in \delta_\mathsf{p}$ is enabled in all states decorating s:
  $\forall s \xrightarrow{\mathsf{p} \triangleright \mathsf{q}!m} s' \in \delta_\mathsf{p}.\ \text{tr-orig}(d(s) \xrightarrow{\mathsf{p} \triangleright \mathsf{q}!m} d(s')) = d(s).$

- *Receive Decoration Validity*: no receive transition is enabled in an alternative continuation originating from the same state:
  $\forall s \xrightarrow{\mathsf{p} \triangleleft \mathsf{q}_1?m_1} s_1,\ s \xrightarrow{x} s_2 \in \delta_\mathsf{p}.\ x \neq \mathsf{p} \triangleleft \mathsf{q}_1?\_ \implies$

  $\forall G' \in \text{tr-dest}(d(s) \xrightarrow{x} d(s_2)).\ \mathsf{q}_1 \triangleright \mathsf{p}!m_1 \notin M^{\mathsf{p}}_{(G' \ldots)}.$

- *Transition Exhaustivity*: every transition that is enabled in some global state decorating s must be an outgoing transition from s:
  $\forall s \in Q.\ \forall G \xrightarrow{x}{}^{*} G' \in \delta_\downarrow.\ G \in d(s) \implies \exists s' \in Q.\ s \xrightarrow{x} s' \in \delta_\mathsf{p}.$

- *Final State Validity*: a reachable state with a non-empty decorating set is final if its decorating set contains a final global state:
  $\forall s \in Q.\ d(s) \neq \emptyset \implies (d(s) \cap F_\mathbf{G} \neq \emptyset \implies s \in F_\mathsf{p}).$

We want to show the following equivalence to prove Theorem 6.1:

$$C_1 \Leftrightarrow \mathcal{A} \text{ refines } \{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}} \text{ and } \{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}} \text{ refines } \mathcal{A}.$$

We address soundness (the forward direction) and completeness (the backward direction) in turn. Soundness states that C_1 is sufficient to show that $\mathcal{A}$ preserves all behaviors of the subset construction, and does not introduce new behaviors.

We say that a state machine A for role p satisfies *Local Language Inclusion* if it satisfies $\mathcal{L}(\mathbf{G})\Downarrow_{\Sigma_\mathsf{p}} \subseteq \mathcal{L}(A)$. The following lemma, proven in the appendix, establishes that *Local Language Inclusion* follows from *Transition Exhaustivity* and *Final State Validity*.

Lemma 6.9. *Let $A_\mathsf{p} = (Q_\mathsf{p}, \Sigma_\mathsf{p}, \delta_\mathsf{p}, s_{0,\mathsf{p}}, F_\mathsf{p})$ denote the state machine for p in $\mathcal{A}$. Then,* Transition Exhaustivity *and* Final State Validity *imply $\mathcal{L}(\mathbf{G})\Downarrow_{\Sigma_\mathsf{p}} \subseteq \mathcal{L}(A_\mathsf{p})$.*

The fact that $\mathcal{A}$ preserves behaviors follows immediately from *Local Language Inclusion*. The fact that $\mathcal{A}$ does not introduce new behaviors, on the other hand, is enforced by *Send Decoration Validity* and *Receive Decoration Validity*.

In the soundness proof for each of our conditions, we prove refinement via structural induction on traces. We show refinement in two steps, first showing that any trace in one CSM is a trace in the other, and then showing that any terminated trace in one CSM is terminated in the other and maximal.

We restate two key definitions used in the soundness proof.

Definition 6.10 (Intersection sets)**.** Let $\mathbf{G}$ be a global type and $\mathsf{GAut}(\mathbf{G})$ be the corresponding state machine. Let p be a role and $w \in \Sigma^*$ be a word. We define the set of possible runs $\mathrm{R}^{\mathbf{G}}_{\mathsf{p}}(w)$ as all maximal runs of $\mathsf{GAut}(\mathbf{G})$ that are consistent with p's local view of w:

$$\mathrm{R}^{\mathbf{G}}_{\mathsf{p}}(w) := \{\rho \text{ is a maximal run of } \mathsf{GAut}(\mathbf{G}) \mid w\Downarrow_{\Sigma_\mathsf{p}} \leq \mathtt{split}(\mathtt{trace}(\rho))\Downarrow_{\Sigma_\mathsf{p}}\} \ .$$

We denote the intersection of the possible run sets for all roles as

$$I(w) := \bigcap_{\mathsf{p} \in \mathcal{P}} \mathrm{R}^{\mathbf{G}}_{\mathsf{p}}(w) \ .$$

Definition 6.11 (Unique splitting of a possible run)**.** Let $\mathbf{G}$ be a global type, p a role, and $w \in \Sigma^*$

a word. Let ρ be a possible run in $R^{G}_{p}(w)$. We define the longest prefix of ρ matching w:

$$\alpha' := \max\{\rho' \mid \rho' \leq \rho \ \wedge\ \texttt{split}(\texttt{trace}(\rho'))\Downarrow_{\Sigma_p} \leq w\Downarrow_{\Sigma_p}\} \ .$$

If $\alpha' \neq \rho$, we can split ρ into $\rho = \alpha \cdot G \xrightarrow{l} G' \cdot \beta$ where $\alpha' = \alpha \cdot G$, G' denotes the state following G, and β denotes the suffix of ρ following $\alpha \cdot G \cdot G'$. We call $\alpha \cdot G \xrightarrow{l} G' \cdot \beta$ the unique splitting of ρ for p matching w. We omit the role p when obvious from context. This splitting is always unique because the maximal prefix of any $\rho \in R^{G}_{p}(w)$ matching w is unique.

Lemma 6.12 (Soundness of C_1). *C_1 implies that $\mathcal{A}$ and $\{\!\{\mathscr{P}(\mathbf{G}, p)\}\!\}_{p\in\mathcal{P}}$ are equivalent.*

Proof. The proof that C_1 implies $\{\!\{\mathscr{P}(\mathbf{G}, p)\}\!\}_{p\in\mathcal{P}}$ refines $\mathcal{A}$ depends only on *Local Language Inclusion* and can be straightforwardly adapted from [Li et al. 2023a, Lemma 4.4]. We instead focus on showing that C_1 implies $\mathcal{A}$ refines $\{\!\{\mathscr{P}(\mathbf{G}, p)\}\!\}_{p\in\mathcal{P}}$, which depends on the other two conditions in C_1. First, we prove that any trace in $\mathcal{A}$ is a trace in $\{\!\{\mathscr{P}(\mathbf{G}, p)\}\!\}_{p\in\mathcal{P}}$:
Claim 1: $\forall\, w \in \Sigma^{\infty}$. w is a trace in $\mathcal{A}$ implies w is a trace in $\{\!\{\mathscr{P}(\mathbf{G}, p)\}\!\}_{p\in\mathcal{P}}$.

We prove the claim by induction for all finite w. The infinite case follows from the finite case because $\{\!\{\mathscr{P}(\mathbf{G}, p)\}\!\}_{p\in\mathcal{P}}$ is deterministic and all prefixes of w are traces of $\mathcal{A}$ and, hence, of $\{\!\{\mathscr{P}(\mathbf{G}, p)\}\!\}_{p\in\mathcal{P}}$. The base cases, where $w = \varepsilon$, is trivially discharged by the fact that ε is a trace of all CSMs. In the inductive step, assume that w is a trace of $\mathcal{A}$. Let $x \in \Sigma$ such that wx is a trace of $\mathcal{A}$. We want to show that wx is also a trace of $\{\!\{\mathscr{P}(\mathbf{G}, p)\}\!\}_{p\in\mathcal{P}}$.

From the induction hypothesis, we know that w is a trace of $\{\!\{\mathscr{P}(\mathbf{G}, p)\}\!\}_{p\in\mathcal{P}}$. Let ξ be the channel configuration uniquely determined by w. Let $(\vec{s}, \xi)$ be the $\mathcal{A}$ configuration reached on w, and let $(\vec{t}, \xi)$ be the $\{\!\{\mathscr{P}(\mathbf{G}, p)\}\!\}_{p\in\mathcal{P}}$ configuration reached on w.

Let q be the role such that $x \in \Sigma_q$, and let s, t denote $\vec{s}_q$, $\vec{t}_q$ from the respective CSM configurations reached on w for $\mathcal{A}$ and $\{\!\{\mathscr{P}(\mathbf{G}, p)\}\!\}_{p\in\mathcal{P}}$.

To show that wx is a trace of $\{\!\{\mathscr{P}(\mathbf{G}, p)\}\!\}_{p\in\mathcal{P}}$, it thus suffices to show that there exists a state t' and a transition $t \xrightarrow{x} t'$ in $\mathscr{P}(\mathbf{G}, q)$.

Since $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p}\in\mathcal{P}}$ implements $\mathbf{G}$, all finite traces of $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p}\in\mathcal{P}}$ are prefixes of $\mathcal{L}(\mathbf{G})$. In other words, $w \in \operatorname{pref}(\mathcal{L}(\mathbf{G}))$. Let ρ be a run such that $\rho \in I(w)$; such a run must exist from [Li et al. 2023a, Lemma 6.3]. Let $\alpha \cdot G \xrightarrow{l} G' \cdot \beta$ be the unique splitting of ρ for q matching w. From the definition of state decoration, it holds that $G \in d(s)$. From the definition of the subset construction, it holds that $G \in t$.

We proceed by case analysis on whether x is a send or receive event.

- Case $x \in \Sigma_{\mathsf{q},!}$. Let $x = \mathsf{q} \triangleright \mathsf{r}!m$. By assumption, there exists $s \xrightarrow{\mathsf{q}\triangleright\mathsf{r}!m} s'$ in A_{q}. We instantiate *Send Decoration Validity* from C_1 with q and this transition to obtain:

$$\text{tr-orig}(d(s) \xrightarrow{\mathsf{q}\triangleright\mathsf{r}!m} d(s')) = d(s) \ .$$

  From $G \in d(s)$, it follows that there exists $G' \in Q_{\mathbf{G}}$ such that $G \xrightarrow{x}{}^{*} G' \in \delta_{\downarrow}$. Because $G \in t$, the existence of t' such that $t \xrightarrow{\mathsf{q}\triangleright\mathsf{r}!m} t'$ is a transition in $\mathscr{P}(\mathbf{G}, \mathsf{p})$ follows immediately from the definition of $\mathscr{P}(\mathbf{G}, \mathsf{q})$'s transition relation.

- Case $x \in \Sigma_{\mathsf{q},?}$. Let $x = \mathsf{q} \triangleleft \mathsf{r}?m$.

  From the fact that ρ is a maximal run in $\mathbf{G}$ with unique splitting $\alpha \cdot G \xrightarrow{l} G' \cdot \beta$ for q matching w, it holds that $w\Downarrow_{\Sigma_{\mathsf{q}}} \cdot \ \mathtt{split}(l)\Downarrow_{\Sigma_{\mathsf{q}}} \in \operatorname{pref}(\mathcal{L}(\mathbf{G}))\Downarrow_{\Sigma_{\mathsf{q}}}$. From [Li et al. 2023a, Lemma 4.3], $\mathcal{L}(\mathbf{G})\Downarrow_{\Sigma_{\mathsf{q}}} = \mathcal{L}(\mathscr{P}(\mathbf{G}, \mathsf{q}))$. Therefore, there exists a t'' such that $t \xrightarrow{\mathtt{split}(l)\Downarrow_{\Sigma_{\mathsf{q}}}} t''$ is a transition in $\mathscr{P}(\mathbf{G}, \mathsf{q})$. From *Transition Exhaustivity*, there likewise exists an s'' such that $s \xrightarrow{\mathtt{split}(l)\Downarrow_{\Sigma_{\mathsf{q}}}} s''$ is a transition in A_{q}.

  We now proceed by showing that it must be the case that $\mathtt{split}(l)\Downarrow_{\Sigma_{\mathsf{q}}} = x$. The reasoning closely follows that in [Li et al. 2023a, Lemma 6.4], which showed that if Receive Validity holds for the subset construction, and some role's subset construction automaton can perform a receive action, then the trace extended with the receive action remains consistent with any global run it was consistent with before. We generalize this property in terms of available message sets in the following lemma, whose proof can be found in the appendix.

Lemma 6.13. *Let $\mathcal{A}$ be a CSM, q be a role, and w, wx be traces of $\mathcal{A}$ such that $x = \mathsf{q} \triangleleft \mathsf{r}?m$. Let s be the state of q's state machine in the $\mathcal{A}$ configuration reached on w. Let ρ be a run that is consistent with w, i.e. for all $\mathsf{p} \in \mathcal{P}$. $w{\Downarrow_{\Sigma_\mathsf{p}}} \leq \mathtt{split}(\mathtt{trace}(\rho)){\Downarrow_{\Sigma_\mathsf{p}}}$. Let $\alpha \cdot G \xrightarrow{l} G' \cdot \beta$ be the unique splitting of ρ for q matching w. If $\mathsf{r} \triangleright \mathsf{q}!m \notin M^{\mathsf{q}}_{(G' \ldots)}$, then $x = \mathtt{split}(l){\Downarrow_{\Sigma_\mathsf{q}}}$.*

We wish to apply Lemma 6.13 with ρ to conclude that $\mathtt{split}(l){\Downarrow_{\Sigma_\mathsf{q}}} = x$. We satisfy the assumption that $\mathsf{r} \triangleright \mathsf{q}!m \notin M^{\mathsf{q}}_{(G' \ldots)}$ by instantiating *Receive Decoration Validity* with $s \xrightarrow{\mathsf{q} \triangleleft \mathsf{r}?m} s'$, $s \xrightarrow{\mathtt{split}(l){\Downarrow_{\Sigma_\mathsf{q}}}} s''$ and G'. The fact that $G' \in \text{tr-dest}(d(s) \xrightarrow{\mathtt{split}(l){\Downarrow_{\Sigma_\mathsf{q}}}} d(s''))$ follows from the fact that $\alpha \cdot G \xrightarrow{l} G' \cdot \beta$ is a run in $\mathbf{G}$ and the definition of state decoration (Definition 6.6). Thus, we conclude from $\mathtt{split}(l){\Downarrow_{\Sigma_\mathsf{q}}} = x$ that there exists a transition $t \xrightarrow{x} t''$ in $\mathscr{P}(\mathbf{G}, \mathsf{q})$.

This concludes our proof that any trace in $\mathcal{A}$ is also a trace of $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$.

Claim 2: $\forall\ w \in \Sigma^*.\ w$ is terminated in $\mathcal{A} \implies w$ is terminated in $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$ and w is maximal in $\mathcal{A}$.

Let w be a terminated trace in $\mathcal{A}$. By Claim 1, w is also a trace in $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$. Let ξ be the channel configuration uniquely determined by w. Let the $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$ configuration reached on w be $(\vec{t}, \xi)$, and let $(\vec{s}, \xi)$ be the $\mathcal{A}$ configuration reached on w. To see that every terminated trace in $\mathcal{A}$ is also terminated in $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$, assume by contradiction that w is not terminated in $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$. Because $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$ is deadlock-free, there must exist a role that can take a step in $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$. Let q be this role, and let x be the transition that is enabled from $\vec{t}_\mathsf{q}$. From *Local Language Inclusion* and the fact that $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$ is deadlock-free, it holds that x is also enabled from $\vec{s}_\mathsf{q}$. We arrive at a contradiction. To see that every terminated trace in $\mathcal{A}$ in maximal, from the above we know that w is terminated in $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$. From the fact that $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$ is deadlock-free, w is maximal in $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$: all states in $\vec{t}$ are final and all channels in ξ are empty. From *Local Language Inclusion*, it follows that all states in $\vec{s}$ are also final, and thus w is maximal in $\mathcal{A}$. □ □

Lemma 6.14 (Completeness of C_1)**.** *If $\mathcal{A}$ and $\{\!\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\!\}_{\mathsf{p}\in\mathcal{P}}$ are equivalent, then C_1 holds.*

We show completeness via modus tollens: we assume a violation in C_1 and the fact that $\mathcal{A}$ and $\{\!\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\!\}_{\mathsf{p}\in\mathcal{P}}$ are equivalent, and prove a contradiction. Since C_1 is a conjunction of four conditions, we derive a contradiction from the violation of each condition in turn. In the interest of proof reuse, we specify which of the two refinement conjuncts we contradict for each condition, and refer the reader to the appendix for the full proofs.

From the negation of *Transition Exhaustivity* and *Final State Validity*, we contradict the fact that $\{\!\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\!\}_{\mathsf{p}\in\mathcal{P}}$ refines $\mathcal{A}$.

Lemma 6.15. *If $\mathcal{A}$ violates* Transition Exhaustivity *or* Final State Validity*, then it does not hold that $\{\!\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\!\}_{\mathsf{p}\in\mathcal{P}}$ refines $\mathcal{A}$.*

Unlike the proofs for *Transition Exhaustivity* and *Final State Validity*, the proofs for the remaining two conditions require both refinement conjuncts to prove a contradiction. Both proofs find a contradiction by obtaining a witness from the violation of *Send Decoration Validity* and *Receive Decoration Validity* respectively, and showing that the same witness can be used to refute Send and Receive Validity for the subset construction.

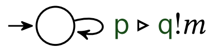
Lemma 6.16. *If $\mathcal{A}$ violates* Send Decoration Validity *or* Receive Decoration Validity*, then it does not hold that $\mathcal{A}$ and $\{\!\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\!\}_{\mathsf{p}\in\mathcal{P}}$ are equivalent.*

6.4 Deciding Protocol Refinement

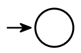
We now turn our attention to *Protocol Refinement*, which asks when an implementation can safely substitute another in all well-behaved contexts with respect to $\mathbf{G}$. Here, we introduce a new notion of refinement with respect to a global type.

Definition 6.17 (Protocol refinement with respect to G). We say that a CSM $\{\!\{A_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ refines a CSM $\{\!\{B_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ with respect to a global type $\mathbf{G}$ if the following properties hold: (i) *subprotocol fidelity:* $\exists S \subseteq \mathcal{L}(\mathsf{GAut}(\mathbf{G})).\ \mathcal{L}(\{\!\{A_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}}) = \mathcal{L}(\texttt{split}(S))$, (ii) *language inclusion:* $\mathcal{L}(\{\!\{A_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}}) \subseteq \mathcal{L}(\{\!\{B_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}})$, and (iii) *deadlock freedom:* $\{\!\{A_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ is deadlock-free.

Item i, *subprotocol fidelity*, sets our notion of refinement apart from standard refinement. We motivate this difference briefly using an example. Consider the CSM consisting of the subset construction for p and B'_{q}, depicted in Fig. 6.4. This CSM recognizes only words of the form $(\mathsf{p}\triangleright\mathsf{q}!m)^{\omega}$. It is nonetheless considered to refine the global type $\mathbf{G}_{loop} := \mu t.\ \mathsf{p} \to \mathsf{q} : m.\ t$ according to the standard notion of refinement, despite the fact that p's messages are never received by q. This is because $\mathcal{L}(\mathbf{G}_{loop})$, containing only infinite words, is defined in terms of an asymmetric downward closure operator $\preceq^{\omega}_{\sim}$, which allows receives to be infinitely postponed. We desire a notion of refinement that allows roles to select which runs to follow in a global type, but disallows them from selecting which words to implement among ones that follow the same run. More formally, our notion of protocol refinement prohibits selectively implementing words that are equivalent under the indistinguishability relation $\sim$: any CSM that refines another with respect to a global type has a language that is closed under $\sim$.



(a) State machine $\mathscr{C}(\mathbf{G},\mathsf{p})$



(b) State machine B'_{q}

Figure 6.4: CSM violating subprotocol fidelity with respect to $\mathbf{G}_{loop}$

In the remainder of the paper, we refer to refinement with respect to $\mathbf{G}$, and omit mention of $\mathbf{G}$ when clear from context. Again using the fact that $\{\!\{\mathscr{P}(\mathbf{G},\mathsf{p})\}\!\}_{\mathsf{p}\in\mathcal{P}}$ is an implementation for $\mathbf{G}$, we say that a CSM $\{\!\{A_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ refines $\mathbf{G}$ if it refines $\{\!\{\mathscr{P}(\mathbf{G},\mathsf{p})\}\!\}_{\mathsf{p}\in\mathcal{P}}$.

We motivate our formulation of the *Protocol Refinement* problem by posing the following variation of *Protocol Verification*, which we call *Monolithic Protocol Refinement*:

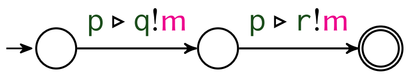
Given an implementable global type $\mathbf{G}$ and a CSM $\mathcal{A}$, does $\mathcal{A}$ *refine* $\{\!\{\mathscr{P}(\mathbf{G},\mathsf{p})\}\!\}_{\mathsf{p}\in\mathcal{P}}$?

This variation asks for a condition, C_1', that satisfies the equivalence:

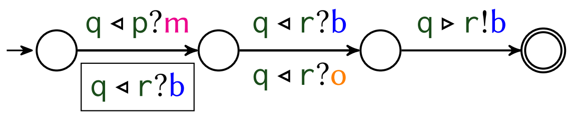
$$C_1' \Leftrightarrow \mathcal{A} \text{ refines } \{\!\!\{ \mathscr{P}(\mathbf{G}, \mathsf{p}) \}\!\!\}_{\mathsf{p} \in \mathcal{P}}.$$

Clearly, C_1 is still a sound candidate as equivalence of two CSMs implies bi-directional protocol refinement. It is instructive to analyze why the completeness arguments for C_1 fail. Recall that the completeness proofs for *Send Decoration Validity* and *Receive Decoration Validity* used the violation of each condition to obtain a local state with a non-empty decoration set, which in turn gives rise to a prefix in $\mathcal{L}(\mathbf{G})$ that must be a trace in the subset construction. This trace is then replayed in the arbitrary CSM, extended in the arbitrary CSM, and then replayed again in the subset construction. This sequence of replaying arguments critically relied on both the assumption that $\mathcal{A}$ refines $\{\!\!\{ \mathscr{P}(\mathbf{G}, \mathsf{p}) \}\!\!\}_{\mathsf{p} \in \mathcal{P}}$, and the assumption that $\{\!\!\{ \mathscr{P}(\mathbf{G}, \mathsf{p}) \}\!\!\}_{\mathsf{p} \in \mathcal{P}}$ refines $\mathcal{A}$.

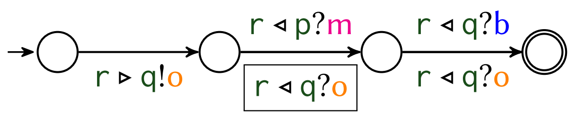
If we cannot assume that $\mathcal{A}$ recognizes every behavior of $\{\!\!\{ \mathscr{P}(\mathbf{G}, \mathsf{p}) \}\!\!\}_{\mathsf{p} \in \mathcal{P}}$, then the reachable local states of $\mathcal{A}$ are no longer precisely characterized by having a non-empty decoration set.



(a) State machine $\mathscr{C}(\mathbf{G}, \mathsf{p})$



(b) State machine A_{q}'



(c) State machine A_{r}'

Figure 6.5: Subset construction for p and two state machines for q and r for $\mathbf{G}'$

Consider the example global type $\mathbf{G}'$:

$$\mathbf{G}' := \mathsf{p} \to \mathsf{q} : \mathsf{m}. + \begin{cases} \mathsf{r} \to \mathsf{q} : \mathsf{b}. \mathsf{p} \to \mathsf{r} : \mathsf{m}. + \begin{cases} \mathsf{q} \to \mathsf{r} : \mathsf{b}. 0 \\ \mathsf{q} \to \mathsf{r} : \mathsf{o}. 0 \end{cases} \\ \mathsf{r} \to \mathsf{q} : \mathsf{o}. \mathsf{p} \to \mathsf{r} : \mathsf{m}. + \begin{cases} \mathsf{q} \to \mathsf{r} : \mathsf{b}. 0 \\ \mathsf{q} \to \mathsf{r} : \mathsf{o}. 0 \end{cases} \end{cases}$$

Let the CSM $\mathcal{A}'$ consist of the subset construction automaton for p, and the state machines A'_q and A'_r, given in Figs. 6.5b and 6.5c. The receive transitions highlighted in red are safe despite violating *Receive Decoration Validity*, because q and r coordinate with each other on which runs of **G** they eliminate: r chooses to never send a b to q, thus q's highlighted transition is safe, and conversely, q never chooses to send o to r, thus r's highlighted transition is safe. Consequently, $\mathcal{A}'$ refines $\mathbf{G}'$ despite violating C_1.

This example shows that any condition C'_1 that is compositional must sacrifice completeness. In fact, deciding whether an arbitrary CSM $\mathcal{A}$ refines the subset construction $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$ for some global type **G** can be shown to be PSPACE-hard via a reduction from the deadlock-freedom problem for 1-safe Petri nets [Esparza and Nielsen 1994]. We refer the reader to the appendix for the full construction.

Lemma 6.18. *The* Monolithic Protocol Refinement *problem is PSPACE-hard.*

Fortunately, we can recover completeness and tractability by only allowing changes to one state machine in $\mathcal{A}$ at a time. Next, we formalize the notions of *CSM contexts* and *well-behavedness* with respect to **G**. We use $\mathcal{A}[\cdot]_\mathsf{p}$ to denote a CSM context with a hole for role $\mathsf{p} \in \mathcal{P}$, and $\mathcal{A}[A]_\mathsf{p}$ to denote the CSM obtained by instantiating the context with state machine A for p. We define well-behaved contexts in terms of the canonical implementation $\mathscr{P}(\mathbf{G}, \mathsf{p})$.

Definition 6.19 (Well-behaved CSM contexts with respect to **G**)**.** Let $\mathcal{A}[\cdot]_\mathsf{p}$ be a CSM context. We say that $\mathcal{A}[\cdot]_\mathsf{p}$ is well-behaved with respect to **G** if $\mathcal{A}[\mathscr{P}(\mathbf{G}, \mathsf{p})]_\mathsf{p}$ refines **G**. We omit **G** when clear from context.

Protocol Refinement asks to find a C_2 that satisfies the following:

Theorem 6.20. *Let* **G** *be an implementable global type,* p *be a role, and A, B be state machines for role* p *such that for all well-behaved contexts $\mathcal{A}[\cdot]_\mathsf{p}$, $\mathcal{A}[B]_\mathsf{p}$ refines* **G**. *Then, for all well-behaved contexts $\mathcal{A}[\cdot]_\mathsf{p}$, $\mathcal{A}[A]_\mathsf{p}$ refines $\mathcal{A}[B]_\mathsf{p}$ if and only if C_2 is satisfied.*

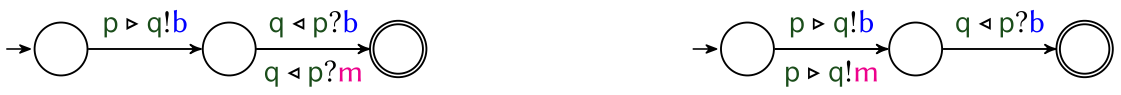
6.4.1 Protocol Refinement Relative to Subset Construction

As a stepping stone, we first consider the special case of *Protocol Refinement* when B is the subset construction automaton for role p. That is, we present C_2' that satisfies the following equivalence:

$$C_2' \Leftrightarrow \text{for all well-behaved contexts } \mathcal{A}[\cdot]_{\mathsf{p}}, \mathcal{A}[A]_{\mathsf{p}} \text{ refines } \mathcal{A}[\mathscr{P}(\mathbf{G}, \mathsf{p})]_{\mathsf{p}}.$$

The relaxation on language equality from *Protocol Verification* means that state machine A no longer needs to satisfy *Local Language Inclusion*, which grants us more flexibility: state machines are now permitted to remove send events. Let us revisit our example global type, $\mathbf{G}_1$:

$$\mathbf{G}_1 \coloneqq + \begin{cases} \mathsf{p} \to \mathsf{q} : \mathsf{b}.\ \mathsf{q} \to \mathsf{p} : \mathsf{b}.\ 0 \\ \mathsf{p} \to \mathsf{q} : \mathsf{m}.\ \mathsf{q} \to \mathsf{p} : \mathsf{m}.\ 0 \end{cases}$$



(a) Removing sends **(b)** Removing receives

Figure 6.6: Two candidate implementations for p

Consider the candidate state machine for role p given in Fig. 6.6a. The CSM obtained from inserting this state machine into any well-behaved context refines $\mathbf{G}$, despite the fact that p never sends m. In general, send events can safely be removed from reachable states in a local state machine without violating subprotocol fidelity or deadlock freedom, as long as *not all* of them are removed.

The same is not true of receive events, on the other hand. The state machine in Fig. 6.6b is not a safe candidate for p, because it causes a deadlock in the well-behaved context that consists of the subset construction for every other role.

Our characterization intuitively follows the notion that input types (receive events) are covariant, and output types (send events) are contravariant. However, note that the state machine above cannot be represented in existing works [Ghilezan et al. 2019b; Bravetti and Zavattaro 2019;

Cutner et al. 2022]: their local types support neither states with both outgoing send and receive events, nor states with outgoing send or receive events to/from different roles.

Our characterization C_2' reuses *Send Decoration Validity*, *Receive Decoration Validity* and *Final State Validity* from C_1, but splits *Transition Exhaustivity* into a separate condition for send and receive events, to reflect the aforementioned asymmetry between them.

Definition 6.21 (C_2')**.** Let $\mathsf{p} \in \mathcal{P}$ be a role and let $A = (Q, \Sigma_{\mathsf{p}}, s_0, \delta, F)$ be a state machine for p. C_2' is satisfied when the following conditions hold in addition to *Send Decoration Validity*, *Receive Decoration Validity* and *Final State Validity*:

- *Send Preservation*: every state containing a send-originating global state must have at least one outgoing send transition:
  $\forall s \in Q.\ \exists G \in Q_{\mathbf{G},!}.\ G \in d(t) \implies \exists x \in \Sigma_{\mathsf{p},!},\ s' \in Q.\ s \xrightarrow{x} s' \in \delta.$

- *Receive Exhaustivity*: every receive transition that is enabled in some global state decorating s must be an outgoing transition from s:
  $\forall s \in Q.\ \forall G \xrightarrow{x}{}^{*} G' \in \delta_{\downarrow}.\ G \in d(s) \wedge x \in \Sigma_{\mathsf{p},?} \implies \exists s' \in Q.\ s \xrightarrow{x} s' \in \delta.$

We want to show the following equivalence:

$$C_2' \Leftrightarrow \text{for all well-behaved contexts } \mathcal{A}[\cdot]_{\mathsf{p}},\ \mathcal{A}[A]_{\mathsf{p}} \text{ refines } \mathcal{A}[\mathscr{P}(\mathbf{G}, \mathsf{p})]_{\mathsf{p}}.$$

We first prove the soundness of C_2'.

Lemma 6.22 (Soundness of C_2')**.** *If C_2' holds, then for all well-behaved contexts $\mathcal{A}[\cdot]_{\mathsf{p}}$, $\mathcal{A}[A]_{\mathsf{p}}$ refines $\mathcal{A}[\mathscr{P}(\mathbf{G}, \mathsf{p})]_{\mathsf{p}}$.*

Proof. Let $\mathcal{A}[\cdot]_{\mathsf{p}}$ be a well-behaved context with respect to $\mathbf{G}$. Like before, we first prove that any trace in $\mathcal{A}[A]_{\mathsf{p}}$ is a trace in $\mathcal{A}[\mathscr{P}(\mathbf{G}, \mathsf{p})]_{\mathsf{p}}$.

Claim 1: $\forall\ w \in \Sigma^{\infty}.\ w$ is a trace in $\mathcal{A}[A]_{\mathsf{p}} \implies w$ is a trace in $\mathcal{A}[\mathscr{P}(\mathbf{G}, \mathsf{p})]_{\mathsf{p}}$.

The proof of Claim 1 for C_2' differs from that for C_1 in only two ways. We discuss the differences in detail below, and avoid repeating the rest of the proof.

1. C_1 grants that every role's state machine satisfies *Send Decoration Validity* and *Receive Decoration Validity*, whereas C_2 only guarantees the conditions for role p. Correspondingly, $\mathcal{A}[A]_{\mathsf{p}}$ only differs from $\mathcal{A}[\mathscr{P}(\mathbf{G}, \mathsf{p})]_{\mathsf{p}}$ in p's state machine; all other roles' state machines are identical between the two CSMs. Therefore, the induction step requires a case analysis on the role whose alphabet the event x belongs to. In the case that $x \in \Sigma_{\mathsf{q}}$ where $\mathsf{q} \neq \mathsf{p}$, the induction hypothesis is trivially re-established by the fact that q's state machine is identical in both CSMs. In the case that $x \in \Sigma_{\mathsf{p}}$, we proceed to reason that x can also be performed by $\mathscr{P}(\mathbf{G}, \mathsf{p})$ in the same well-behaved context.

2. C_1 includes *Transition Exhaustivity*, which allows us to conclude that given a run with unique splitting $\alpha \cdot G \xrightarrow{l} G' \cdot \beta$ for p matching w and the fact that $G \in s$, there must exist a transition $s \xrightarrow{\texttt{split}(l)\Downarrow_{\Sigma_{\mathsf{p}}}} s''$ in p's state machine. Lemma 6.13 can then be instantiated directly with $\alpha \cdot G \xrightarrow{l} G' \cdot \beta$ to complete the proof. C_2, on the other hand, splits *Transition Exhaustivity* into *Send Preservation* and *Receive Exhaustivity*, and we can only establish that such a transition exists and reuse the proof in the case that $\texttt{split}(l)\Downarrow_{\Sigma_{\mathsf{p}}} \in \Sigma_{\mathsf{p},?}$. Since A is permitted to remove send events, if $\texttt{split}(l)\Downarrow_{\Sigma_{\mathsf{p}}} \in \Sigma_{\mathsf{p},!}$, the transition $s \xrightarrow{\texttt{split}(l)\Downarrow_{\Sigma_{\mathsf{p}}}} s''$ may not exist at all in A. However, the existence of a run $\alpha \cdot G \xrightarrow{l} G' \cdot \beta$ where l is a send event for p makes G a send-originating global state in p's projection by erasure automaton. *Send Preservation* thus guarantees that there exists a transition $s \xrightarrow{x'} s'''$ in A such that $x' \in \Sigma_{\mathsf{p},!}$. By *Send Decoration Validity*, x' originates from G in the projection by erasure, and we can find another run ρ' such that $\alpha' \cdot G \xrightarrow{l'} G'' \cdot \beta'$ is the unique splitting for p matching w and $\texttt{split}(l')\Downarrow_{\Sigma_{\mathsf{p}}} = x'$. We satisfy the assumption that $\mathsf{r} \triangleright \mathsf{p}!m \notin M^{\mathsf{p}}_{(G'' \ldots)}$ by instantiating *Receive Decoration Validity* with p, $s \xrightarrow{x} s'$, $s \xrightarrow{\texttt{split}(l')\Downarrow_{\Sigma_{\mathsf{p}}}} s''$ and G''. The fact that $G'' \in \text{tr-dest}(d_{\mathrm{G}}(s) \xrightarrow{\texttt{split}(l')\Downarrow_{\Sigma_{\mathsf{p}}}} d_{\mathrm{G}}(s''))$ follows from the fact that $\alpha \cdot G \xrightarrow{l'} G'' \cdot \beta'$ is a

run in $\mathbf{G}$ and Definition 6.6. Instantiating Lemma 6.13 with ρ', we obtain $\texttt{split}(l')\Downarrow_{\Sigma_{\mathsf{p}}} = x$, which is a contradiction: x is a receive event and $\texttt{split}(l')\Downarrow_{\Sigma_{\mathsf{p}}}$ is a send event. Thus, it cannot be the case that $\texttt{split}(l')\Downarrow_{\Sigma_{\mathsf{p}}} \in \Sigma_{\mathsf{p},!}$.

This concludes our proof that any trace in $\mathcal{A}[A]_{\mathsf{p}}$ is also a trace in $\mathcal{A}[\mathscr{P}(\mathbf{G},\mathsf{p})]_{\mathsf{p}}$.

The following claim completes our soundness proof:

Claim 2: $\forall\, w \in \Sigma^*.\ w$ is terminated in $\mathcal{A}[A]_{\mathsf{p}} \implies w$ is terminated in $\mathcal{A}[\mathscr{P}(\mathbf{G},\mathsf{p})]_{\mathsf{p}}$ and w is maximal in $\mathcal{A}[A]_{\mathsf{p}}$.

The proof of Claim 2 for C_1 again relies on *Local Language Inclusion*, which is unavailable to C_2'. Instead, we turn to *Send Preservation*, *Receive Exhaustivity* and *Final State Validity* to establish this claim. Let w be a terminated trace in $\mathcal{A}[A]_{\mathsf{p}}$. By Claim 1, it holds that w is a trace in $\mathcal{A}[\mathscr{P}(\mathbf{G},\mathsf{p})]_{\mathsf{p}}$. Let ξ be the channel configuration uniquely determined by w. Let $(\vec{s},\xi)$ be the $\mathcal{A}[\mathscr{P}(\mathbf{G},\mathsf{p})]_{\mathsf{p}}$ configuration reached on w, and let $(\vec{t},\xi)$ be the $\mathcal{A}[A]_{\mathsf{p}}$ configuration reached on w. To see that w is terminated in $\mathcal{A}[\mathscr{P}(\mathbf{G},\mathsf{p})]_{\mathsf{p}}$, suppose by contradiction that w is not terminated in $\mathcal{A}[\mathscr{P}(\mathbf{G},\mathsf{p})]_{\mathsf{p}}$. Because $\mathcal{A}[\mathscr{P}(\mathbf{G},\mathsf{p})]_{\mathsf{p}}$ is deadlock-free, and because the state machines for all non-p roles are identical between the two CSMs, it must be the case that p witnesses the non-termination of w, in other words, $\mathscr{P}(\mathbf{G},\mathsf{p})$ can take a transition that A cannot. Let $\vec{s}_{\mathsf{p}} \xrightarrow{x} s'$ be the transition that p can take from $\vec{s}_{\mathsf{p}}$. Let G be a state in $\vec{s}_{\mathsf{p}}$; such a state is guaranteed to exist by the fact that no reachable states in the subset construction are empty. Then, in the projection by erasure automaton, the initial state reaches G on $w\Downarrow_{\Sigma_{\mathsf{p}}}$. By the fact that w is a trace of $\mathcal{A}[A]_{\mathsf{p}}$, it holds that s_0 reaches $\vec{s}_{\mathsf{p}}$ on $w\Downarrow_{\Sigma_{\mathsf{p}}}$ in A. By the definition of state decoration, $G \in d(\vec{t}_{\mathsf{p}})$.

- If $x \in \Sigma_!$, it follows that G is a send-originating global state. By *Send Preservation*, for any state in A that contains at least one send-originating global state, of which $\vec{t}_{\mathsf{p}}$ is one, there exists a transition $\vec{t}_{\mathsf{p}} \xrightarrow{x'} t'$ such that $x' \in \Sigma_{\mathsf{p},!}$. Because send transitions in a CSM are always enabled, role p can take this transition in $\mathcal{A}[A]_{\mathsf{p}}$. We reach a contradiction to the fact that w is terminated in $\mathcal{A}[A]_{\mathsf{p}}$.

- If $x \in \Sigma_?$, it follows that G is a receive-originating global state. From *Receive Exhaustivity*, any receive event that originates from any global state in $d(\vec{t}_p)$ must also originate from $\vec{t}_p$. Therefore, there must exist t' such that $\vec{t}_p \xrightarrow{x} t'$ is a transition in B'_p. Because the channel configuration is identical in both CSMs, role p can take this transition in $\mathcal{A}[A]_p$. We again reach a contradiction to the fact that w is terminated in $\mathcal{A}[A]_p$.

To see that w is maximal in $\mathcal{A}[A]_p$, observe that for all roles $q \neq p$, $\vec{s}_q = \vec{t}_q$. Thus, it remains to show that $\vec{t}_p$ is a final state in A. Because $\vec{s}_p$ is a final state, by the definition of the subset construction there exists a global state $G \in \vec{s}_p$ such that the projection erasure automaton reaches G on $w\Downarrow_{\Sigma_p}$ and G is a final state. Because A reaches $\vec{t}_p$ on $w\Downarrow_{\Sigma_p}$, by Definition 6.6 it holds that $G \in d(\vec{t}_p)$. By *Final State Validity*, it holds that $\vec{t}_p$ is a final state in A. This concludes our proof that any terminated trace in $\mathcal{A}[A]_p$ is also a terminated trace in $\mathcal{A}[\mathscr{P}(\mathbf{G}, p)]_p$, and is maximal in $\mathcal{A}[A]_p$.

Together, Claim 1 and 2 establish that $\mathcal{A}[A]_p$ satisfies language inclusion (Item ii) and deadlock freedom (Item iii). It remains to show that $\mathcal{A}[A]_p$ satisfies subprotocol fidelity (Item i). This follows immediately from [Majumdar et al. 2021a, Lemma 22], which states that all CSM languages are closed under the indistinguishability relation $\sim$. □

Lemma 6.23 (Completeness of C'_2)**.** *If for all well-behaved contexts $\mathcal{A}[\cdot]_p$, $\mathcal{A}[A]_p$ refines $\mathcal{A}[\mathscr{P}(\mathbf{G}, p)]_p$, then C'_2 holds.*

As before, we prove the modus tollens of this implication, which states that if C'_2 does not hold, then there exists a well-behaved context $\mathcal{A}[\cdot]_p$ such that $\mathcal{A}[A]_p$ does not protocol-refine $\mathcal{A}[\mathscr{P}(\mathbf{G}, p)]_p$.

We first turn our attention to finding a well-behaved witness context $\mathcal{A}[\cdot]_p$ such that we can refute subprotocol fidelity, language inclusion, or deadlock freedom. It turns out that the context consisting of the subset construction automaton for every other role is a suitable witness. We denote this context by $\mathscr{C}(\mathbf{G})[\cdot]_p$ and note that it is trivially well-behaved because

$\mathscr{C}(\mathbf{G})[\mathscr{P}(\mathbf{G},\mathsf{p})]_{\mathsf{p}} = \{\!\{\mathscr{P}(\mathbf{G},\mathsf{p})\}\!\}_{\mathsf{p}\in\mathcal{P}}$.

Recall from the completeness arguments for C_1 that we obtained a violating state in some state machine A with a non-empty decoration set from the negation of each condition in C_1. From this state's decoration set we obtained a witness global state G, and in turn a run $\alpha \cdot G$ in $\mathbf{G}$, and from the assumption that $\{\!\{\mathscr{P}(\mathbf{G},\mathsf{p})\}\!\}_{\mathsf{p}\in\mathcal{P}}$ refines $\mathcal{A}$, we argued that $\mathtt{split}(\mathtt{trace}(\alpha \cdot G))$ is a trace in $\mathcal{A}$. We then showed that A is in the violating state in the $\mathcal{A}$ configuration reached on $\mathtt{split}(\mathtt{trace}(\alpha \cdot G))$, and from there we used each violated condition to find a contradiction.

The completeness proof for C_2' cannot simply rely on the fact that $\{\!\{\mathscr{P}(\mathbf{G},\mathsf{p})\}\!\}_{\mathsf{p}\in\mathcal{P}}$ refines $\mathscr{C}(\mathbf{G})[A]_{\mathsf{p}}$. Instead, we must separately establish that every state with a non-empty decoration set can be reached on a trace shared by both $\mathscr{C}(\mathbf{G})[A]_{\mathsf{p}}$ and $\{\!\{\mathscr{P}(\mathbf{G},\mathsf{p})\}\!\}_{\mathsf{p}\in\mathcal{P}}$.

Lemma 6.24. *Let A be a state machine for p and s be a state in A. Let $G \in d(s)$, and let $u \in \Sigma_{\mathsf{p}}^*$ be a word such that $s_0 \xrightarrow{u}{}^* s$ in A. Then, there exists a run $\alpha \cdot G$ of* $\mathsf{GAut}(\mathbf{G})$ *such that* $\mathtt{split}(\mathtt{trace}(\alpha \cdot G))\Downarrow_{\Sigma_{\mathsf{p}}} = u$, $\mathtt{split}(\mathtt{trace}(\alpha \cdot G))$ *is a trace in $\mathscr{C}(\mathbf{G})[A]_{\mathsf{p}}$ and in the CSM configuration reached on* $\mathtt{split}(\mathtt{trace}(\alpha \cdot G))$*, A is in state s.*

With Lemma 6.24 replacing the assumption that $\{\!\{\mathscr{P}(\mathbf{G},\mathsf{p})\}\!\}_{\mathsf{p}\in\mathcal{P}}$ refines $\mathscr{C}(\mathbf{G})[A]_{\mathsf{p}}$, we can reuse the construction in Lemma 6.16 to obtain a word that is a trace in $\mathscr{C}(\mathbf{G})[A]_{\mathsf{p}}$ but not a trace in $\{\!\{\mathscr{P}(\mathbf{G},\mathsf{p})\}\!\}_{\mathsf{p}\in\mathcal{P}}$, thus evidencing the necessity of *Send Decoration Validity* and *Receive Decoration Validity*. The proof of Lemma 6.25 proceeds identically to that of Lemma 6.16 and is thus omitted.

Lemma 6.25. *If A violates* Send Decoration Validity *or* Receive Decoration Validity*, then it does not hold that for all well-behaved contexts $\mathcal{A}[\cdot]_{\mathsf{p}}$, $\mathcal{A}[A]_{\mathsf{p}}$ refines $\mathscr{C}(\mathbf{G})[A]_{\mathsf{p}}$.*

We also use Lemma 6.24 to show the necessity of *Send Preservation*, *Receive Exhaustivity* and *Final State Validity*. As a starting point, let A, s, u and $\alpha \cdot G$ be obtained from Lemma 6.24 and the violation of *Send Preservation*. To show the necessity of *Send Preservation*, we consider the largest extension v of u in $\mathscr{C}(\mathbf{G})[A]_{\mathsf{p}}$. In the case that u is terminated in $\mathscr{C}(\mathbf{G})[A]_{\mathsf{p}}$, we refute deadlock freedom from the fact that u is not maximal: $G \in s$ is a send-originating state, and final states in $\mathsf{GAut}(\mathbf{G})$ do not contain outgoing transitions. If $v \neq u$, there exists a run $\alpha \cdot G \xrightarrow{\mathsf{p} \to \mathsf{q}:m} G' \cdot \beta$ such that $\mathtt{split}(\mathtt{trace}(\alpha \cdot G \xrightarrow{\mathsf{p} \to \mathsf{q}:m} G' \cdot \beta) \Downarrow_{\Sigma_\mathsf{p}} = v \Downarrow_{\Sigma_\mathsf{p}}$. By subprotocol fidelity, $\mathtt{split}(\mathtt{trace}(\alpha \cdot G \xrightarrow{\mathsf{p} \to \mathsf{q}:m} G' \cdot \beta))$ is a trace in $\mathscr{C}(\mathbf{G})[A]_\mathsf{p}$. Consequently, $\mathtt{split}(\mathtt{trace}(\alpha \cdot G \xrightarrow{\mathsf{p} \to \mathsf{q}:m} G' \cdot \beta)) \Downarrow_{\Sigma_\mathsf{p}}$ is a prefix in A. We find a contradiction from the fact that A is deterministic and there is no outgoing transition labeled $\mathsf{p} \triangleright \mathsf{q}!m$ from s. Similar arguments can be used to show the necessity of *Receive Exhaustivity*. Finally, for *Final State Validity*, in the case that s is non-final in A but contains a final state in $\mathsf{GAut}(\mathbf{G})$, we can instantiate Lemma 6.24 with this final state and show that u evidences a deadlock.

Lemma 6.26. *If A violates* Send Preservation, Receive Exhaustivity *or* Final State Validity, *then it does not hold that for all well-behaved contexts $\mathcal{A}[\cdot]_\mathsf{p}$, $\mathcal{A}[A]_\mathsf{p}$ refines $\mathscr{C}(\mathbf{G})[A]_\mathsf{p}$.*

6.4.2 Protocol Refinement (General Case)

Equipped with the solution to a special case, we are ready to revisit the general case of *Protocol Refinement*, which asks to find a C_2 that satisfies the following:

$$C_2 \Leftrightarrow \text{for all well-behaved contexts } \mathcal{A}[\cdot]_\mathsf{p},\ \mathcal{A}[A]_\mathsf{p} \text{ refines } \mathcal{A}[B]_\mathsf{p}.$$

Critical to the former problems is the fact that the state decoration function precisely captures those states in a local state machine that are reachable in some CSM execution, under some assumptions on the context: a state is reachable if and only if its decoration set is non-empty. This allows the conditions in C_1 and C_2' to precisely characterize the reachable local states.

The second problem generalizes the subset projection to an arbitrary state machine B, and asks whether a candidate state machine A (the subtype) refines B (the supertype) in any well-behaved context. Unfortunately, we cannot simply decorate the subtype with the supertype's states, because not all states in the supertype are reachable. Instead, we need to restrict the set of states in the supertype to those that themselves have non-empty decoration sets with respect to $\mathbf{G}$.

In the remainder of this section, let $\mathsf{p} \in \mathcal{P}$ be a role, let $B = (Q_B, \Sigma_{\mathsf{p}}, t_0, \delta_B, F_B)$ denote the supertype state machine for p, and let $A = (Q_A, \Sigma_{\mathsf{p}}, s_0, \delta_A, F_A)$ denote the subtype state machine for p. We modify our state decoration function in Definition 6.6 to map states of A to subsets of states in B that themselves have non-empty decoration sets with respect to $\mathbf{G}$.

Definition 6.27 (State decoration with respect to a supertype)**.** Let $\mathbf{G}$ be a global type. Let $\mathsf{p} \in \mathcal{P}$ be a role, and let $B = (Q_B, \Sigma_{\mathsf{p}}, t_0, \delta_B, F_B)$ and $A = (Q_A, \Sigma_{\mathsf{p}}, s_0, \delta_A, F_A)$ be two deterministic finite state machines for p. We define a total function $d_{\mathbf{G},B,A} : Q' \to 2^Q$ that maps each state in A to a subset of states in B such that:

$$d_{\mathbf{G},B,A}(s) = \{t \in Q_B \mid \exists u \in \Sigma_{\mathsf{p}}^*.\ s_0 \xrightarrow{u}{}^{*} s \in \delta_A \land t_0 \xrightarrow{u}{}^{*} t \in \delta_B \land d(t) \neq \emptyset\}$$

We again omit the subscripts $\mathbf{G}$ and A when clear from context, but retain the subscript B to distinguish d_B from d in Definition 6.6.

We likewise require a generalization of tr-orig and tr-dest to be defined in terms of B, instead of the projection by erasure automaton for p.

Definition 6.28 (Transition origin and destination with respect to a supertype)**.** Let $\mathbf{G}$ be a global type, and let $B = (Q_B, \Sigma_{\mathsf{p}}, t_0, \delta_B, F_B)$ be a state machine. For $x \in \Sigma_{\mathsf{p}}$ and $s, s' \subseteq Q_B$, we define the set of *transition origins* $\text{tr-orig}(s \xrightarrow{x} s')$ and *transition destinations* $\text{tr-dest}(s \xrightarrow{x} s')$ as follows:

$$\begin{aligned}\text{tr-orig}_B(s \xrightarrow{x} s') &:= \{t \in s \mid \exists t' \in s'.\ t \xrightarrow{x}{}^{*} t' \in \delta_B\} \text{ and} \\ \text{tr-dest}_B(s \xrightarrow{x} s') &:= \{t' \in s' \mid \exists t \in s.\ t \xrightarrow{x}{}^{*} t' \in \delta_B\}\ .\end{aligned}$$

We present C_2 in terms of the newly defined decoration function d_B.

Definition 6.29 (C_2)**.** Let $\mathbf{G}$ be a global type, $\mathsf{p} \in \mathcal{P}$ be a role, and let further $B = (Q_B, \Sigma_{\mathsf{p}}, t_0, \delta_B, F_B)$ and $A = (Q_A, \Sigma_{\mathsf{p}}, s_0, \delta_A, F_A)$ be two deterministic state machines for p. C_2 is the conjunction of the following conditions:

- *Send Decoration Subtype Validity*: every send transition $s \xrightarrow{x} s' \in \delta_A$ must be enabled in all states of B decorating s:
$\forall s \xrightarrow{\mathsf{p} \triangleright \mathsf{q}!m} s' \in \delta_A.\ \text{tr-orig}_B(d_B(s) \xrightarrow{\mathsf{p} \triangleright \mathsf{q}!m} d_B(s')) = d_B(s).$

- *Receive Decoration Subtype Validity*: no receive transition is enabled in an alternative continuation originating from the same state:
$\forall s \xrightarrow{\mathsf{p} \triangleleft \mathsf{q}_1 ? m_1} s_1,\ s \xrightarrow{x} s_2 \in \delta_A.\ x \neq \mathsf{p} \triangleleft \mathsf{q}_1?\_ \implies$
$$\forall G \in \bigcup_{t \in d_B(s_2)} \{d(t) \mid t \in \text{tr-dest}_B(d_B(s) \xrightarrow{x} d_B(s_2))\}.\ \mathsf{q}_1 \triangleright \mathsf{p}!m_1 \notin M^{\mathsf{p}}_{(G\ldots)}.$$

- *Send Subtype Preservation*: every state decorated by a send-originating global state must have at least one outgoing send transition:
$\forall s \in Q_A.\ (\bigcup_{t \in d_B(s)} d(t) \cap Q_{\mathbf{G},!} \neq \emptyset) \implies \exists x \in \Sigma_{\mathsf{p},!},\ s' \in Q_A.\ s \xrightarrow{x} s' \in \delta_A.$

- *Receive Subtype Exhaustivity*: every receive transition that is enabled in some global state decorating s must be an outgoing transition from s:
$\forall s \in Q_A.\ \forall G \xrightarrow{x}{}^{*} G' \in \delta_{\downarrow}.\ G \in \bigcup_{t \in d_B(s)} d(t) \implies \exists s' \in Q_A.\ s \xrightarrow{x} s' \in \delta_A.$

- *Final State Validity*: a reachable state is final if its decorating set contains a final global state:
$\forall s \in Q_A.\ \bigcup_{t \in d_B(s)} d(t) \neq \emptyset \implies (\bigcup_{t \in d_B(s)} d(t) \cap F_{\mathbf{G}} \neq \emptyset) \implies s \in F_A.$

We want to show the following equivalence to prove Theorem 6.20:

$$C_2 \Leftrightarrow \text{for all well-behaved contexts } \mathcal{A}[\cdot]_{\mathsf{p}},\ \mathcal{A}[A]_{\mathsf{p}} \text{ refines } \mathcal{A}[B]_{\mathsf{p}}.$$

Lemma 6.30 (Soundness of C_2). *If C_2 holds, then for all well-behaved contexts $\mathcal{A}[\cdot]_{\mathsf{p}}$, $\mathcal{A}[A]_{\mathsf{p}}$ refines $\mathcal{A}[B]_{\mathsf{p}}$.*

Predictably, the proof of soundness is directly adapted from the proof for C_2' by applying suitable "liftings", and can be found in the appendix.

Lemma 6.31 (Completeness of C_2). *If for all well-behaved contexts $\mathcal{A}[\cdot]_{\mathsf{p}}$, $\mathcal{A}[A]_{\mathsf{p}}$ refines $\mathcal{A}[B]_{\mathsf{p}}$, then C_2 holds.*

Again, we prove the modus tollens of this implication, and we again are required to find a witness well-behaved context $\mathcal{A}[\cdot]_{\mathsf{p}}$, such that $\mathcal{A}[A]_{\mathsf{p}}$ does not refine $\mathcal{A}[B]_{\mathsf{p}}$ under the assumption of the negation of C_2. In the special case where B is the subset construction automaton, we observed that any state in A with a non-empty decoration set with respect to $\mathbf{G}$ is reachable by the CSM consisting of A and the subset construction context, denoted $\mathscr{C}(\mathbf{G})[A]_{\mathsf{p}}$. We were therefore able to use $\mathscr{C}(\mathbf{G})[\cdot]_{\mathsf{p}}$ as the witness well-behaved context. A similar characterization is true in the general case: a state in A is reachable by $\mathscr{C}(\mathbf{G})[A]_{\mathsf{p}}$ if it has a non-empty decoration set with respect to B. This in turn depends on the fact that we only label states in A with states in B that themselves have non-empty decorating sets with respect to $\mathbf{G}$. The following lemma lifts Lemma 6.24 to the general problem setting:

Lemma 6.32. *Let A, B be two state machines for p, such that for all well-behaved contexts $\mathcal{A}[\cdot]_{\mathsf{p}}$, $\mathcal{A}[B]_{\mathsf{p}}$ refines $\mathbf{G}$. Let s be a state in A, and let t be a state in B such that $t \in d_B(s)$. Let $u \in \Sigma_{\mathsf{p}}^*$ be a word such that $s_0 \xrightarrow{u}^* s$ in A. Then, there exists a run $\alpha \cdot G$ of $\mathsf{GAut}(\mathbf{G})$ such that $\mathtt{split}(\mathtt{trace}(\alpha \cdot G))\Downarrow_{\Sigma_{\mathsf{p}}} = u$, $\mathtt{split}(\mathtt{trace}(\alpha \cdot G))$ is a trace in both $\mathscr{C}(\mathbf{G})[A]_{\mathsf{p}}$ and $\mathscr{C}(\mathbf{G})[B]_{\mathsf{p}}$ and in the CSM configuration reached on $\mathtt{split}(\mathtt{trace}(\alpha \cdot G))$, A is in state s.*

Proof. From the fact that $t \in d_B(s)$ and the definition of state decoration (Definition 6.27), it holds that $d(t) \neq \emptyset$ and $t_0 \xrightarrow{u}^* t \in \delta_B$. Let $G \in d(t)$. We apply Lemma 6.24 to obtain a run $\alpha \cdot G$ such that $\mathtt{split}(\mathtt{trace}(\alpha \cdot G))\Downarrow_{\Sigma_{\mathsf{p}}} = u$, $\mathtt{split}(\mathtt{trace}(\alpha \cdot G))$ is a trace in $\mathscr{C}(\mathbf{G})[B]_{\mathsf{p}}$ and in the $\mathscr{C}(\mathbf{G})[B]_{\mathsf{p}}$

configuration reached on $\texttt{split}(\texttt{trace}(\alpha \cdot G))$, B is in state t. Because $s_0 \xrightarrow{u}^* s \in \delta_A$, and all non-$\mathsf{p}$ state machines are identical from $\mathscr{C}(\mathbf{G})[B]_{\mathsf{p}}$ to $\mathscr{C}(\mathbf{G})[A]_{\mathsf{p}}$, it is clear that $\texttt{split}(\texttt{trace}(\alpha \cdot G))$ is also a trace of $\mathscr{C}(\mathbf{G})[A]_{\mathsf{p}}$ and in the CSM configuration reached on $\texttt{split}(\texttt{trace}(\alpha \cdot G))$, A is in state s. □ □

Having found our witness well-behaved context $\mathscr{C}(\mathbf{G})[\cdot]_{\mathsf{p}}$, established Lemma 6.32 to replace Lemma 6.24, and observed that the violation of each condition in C_2 likewise yields a state with a non-empty decoration set with respect to B, completeness then amounts to showing the existence of a $w \in \Sigma^*$ such that w refutes subprotocol fidelity, language inclusion, or deadlock freedom. Recall that the proofs for the necessity of *Send Preservation*, *Receive Exhaustivity* and *Final State Validity* in the case where B is the subset construction constructed a trace that refuted either subprotocol fidelity or deadlock freedom. These two properties are identical across both formulations of the problem, and therefore the construction can be wholly reused to show the necessity of *Send Subtype Preservation*, *Receive Subtype Exhaustivity* and *Final State Subtype Validity*.

Lemma 6.33. *If* $\mathcal{A}[A]_{\mathsf{p}}$ *violates* Send Decoration Subtype Validity *or* Receive Decoration Subtype Validity, *then it does not hold that for all well-behaved contexts* $\mathcal{A}[\cdot]_{\mathsf{p}}$, $\mathcal{A}[A]_{\mathsf{p}}$ *refines* $\mathcal{A}[B]_{\mathsf{p}}$.

The proofs for the necessity of *Send Decoration Validity* and *Receive Decoration Validity*, on the other hand, construct a word that is a trace in $\mathcal{A}[A]_{\mathsf{p}}$ but not a trace in $\mathscr{C}(\mathbf{G})[A]_{\mathsf{p}}$. In the general case, we can show that the same construction is a trace in $\mathcal{A}[A]_{\mathsf{p}}$ but not a trace in $\mathcal{A}[B]_{\mathsf{p}}$. We omit the proofs to avoid redundancy.

Lemma 6.34. *If* $\{\!\{A_{\mathsf{p}}\}\!\}_{\mathsf{p} \in \mathcal{P}}$ *violates* Send Subtype Preservation, Receive Subtype Exhaustivity, *or* Final State Subtype Validity, *then it does not hold that for all well-behaved contexts* $\mathcal{A}[\cdot]_{\mathsf{p}}$, $\mathcal{A}[A]_{\mathsf{p}}$ *refines* $\mathcal{A}[B]_{\mathsf{p}}$.

6.5 Complexity Analysis

We complete our discussion with a complexity analysis of the two considered problems, building on the characterizations established in Theorem 6.1 and Theorem 6.20.

For the *Protocol Verification* problem, let m be the size of $\mathcal{A}$ and n the size of $\mathbf{G}$. Moreover, let A_{p} be the local implementation of some role p in $\mathcal{A}$. Observe that the sets $d_{\mathbf{G}}(s)$ for each state s of A_{p} as well as the sets $M^{\mathsf{p}}_{(G' \ldots)}$ for each subterm G' of $\mathbf{G}$ are at most of size n. It is then easy to see that C_1 can be checked in time polynomial in n and m. As established in Section 3.4, $M^{\mathsf{p}}_{(G' \ldots)}$ is computable in co-NP. Observe that the function $d_{\mathbf{G}}$ can be computed for the local implementation of each role $A_{\mathsf{p}} \in \mathcal{P}$ using a simple fixpoint loop. Each set $d_{\mathbf{G}}(s)$ can be represented as a bit vector of size n, making all set operations constant time. The loop inserts at most n subterms of $\mathbf{G}$ into each $d_{\mathbf{G}}(s)$, which takes time $O(mn)$ for all insertions. Moreover, for each G inserted into a set $d_{\mathbf{G}}(s)$ and each transition $s \xrightarrow{x} s'$ in A_{p}, we need to compute the set $\{G' \mid G \xrightarrow{x}^{*} G' \in \delta_{\downarrow}\}$ which is then added to $d_{\mathbf{G}}(s')$. Computing these sets takes time $O(mn)$ for each G and s.

Thus, we establish the following complexity characterization.

Theorem 6.35. *The* Protocol Verification *and* Protocol Refinement *problems are decidable in co-NP.*

6.6 Related Work

Session types were first introduced in binary form by Honda in 1993 [Honda 1993]. Binary session types describe interactions between two participants, and communication safety of binary sessions amounts to channel duality. Binary session types were generalized to multiparty session types – describing interactions between more than two participants – by Honda, Yoshida and Carbone in 2008 [Honda et al. 2008], and the corresponding notion of safety was generalized from duality to multiparty consistency. Binary session types were inspired by and enjoy a close connection to linear logic [Girard 1987; Wadler 2014; Caires et al. 2016]. Horne generalizes this connection to multiparty session types and non-commutative extensions of linear logic [Horne 2020]. The connection between multiparty session types and logic is also explored in [Carbone et al. 2016; Caires and Pérez 2016; Carbone et al. 2017]. MSTs have since been extensively studied and widely adopted in practical programming languages; we refer the reader to [Coppo et al. 2015] for a comprehensive survey.

Session type syntax. Session type frameworks have enjoyed various extensions since their inception. In particular, the choice operator for both global and local types has received considerable attention over the years. MSTs were originally introduced as global types, with a *directed* choice operator that restricted a sender to sending different messages to the same recipient. [Castagna et al. 2012] and [Majumdar et al. 2021a] relax this restriction to *sender-driven choice*, which allows a sender to send different messages to different recipients, and increases the expressivity of global types. The results in this chapter target global types with sender-driven choice. For local types, a direct comparison can be drawn to the π-calculus, for which *mixed choice* was shown to be strictly more expressive than *separate choice* [Palamidessi 2003]. Mixed choices allow both send and receive actions, whereas separate choices consist purely of either sends or receives. [Li et al. 2023a] showed that any global type with sender-driven choice can be implemented by a CSM with only separate choice. Mixed choice for binary local types was investigated in [Casal et al. 2022], although [Peters and Yoshida 2022] later showed that this variant falls short of the full expressive power of mixed choice π-calculus, and instead can only express separate choice π-calculus. Other communication primitives have also been studied, such as channel delegation [Honda et al. 2008, 1998; Castellani et al. 2020], dependent predicates [Toninho et al. 2011, 2021], parametrization [Deniélou et al. 2012; Charalambides et al. 2016] and data refinement [**?**].

Session type semantics. MSTs were introduced in [Honda et al. 2008] with a process algebra semantics. The connection to CSMs was established in [Deniélou and Yoshida 2012], which defines a class of CSMs whose state machines can be represented as local types, called *Communicating*

Session Automata (CSA). CSAs inherit from the local types they represent restrictions on choice discussed above, "tree-like" restrictions on the structure (see [Stutz 2023] for a characterization), and restrictions on outgoing transitions from final states. The CSM implementation model in our work assumes none of the above restrictions, and is thus true to its name.

Session subtyping. Session subtyping was first introduced by [Gay and Hole 2005] in the context of the π-calculus, which was in turn inspired by Pierce and Sangiorgi's work on subtyping for channel endpoints [Pierce and Sangiorgi 1996]. The session types literature distinguishes between two notions of subtyping based on the network assumptions of the framework: *synchronous* and *asynchronous subtyping*. Both notions respect Liskov and Wing's substitution principle [Liskov and Wing 1994], but differ in the guarantees provided. We discuss each in turn.

Synchronous subtyping follows the notions of covariance and contravariance introduced by [Gay and Hole 2005], and checks that a subtype contains fewer sends and more receives than its supertype. For binary synchronous session types, Lange and Yoshida [Lange and Yoshida 2016] show that subtyping can be decided in quadratic time via model checking of a characteristic formulae in the modal μ-calculus. For multiparty synchronous session types, Ghilezan et al. [Ghilezan et al. 2019b] present a precise subtyping relation that is universally quantified over all contexts, and restricts the local type syntax to directed choice. As mentioned in **??**, [Ghilezan et al. 2019b], their subtyping relation is incomplete when generalized to asynchronous multiparty sessions with directed choice. As discussed in **??**, their subtyping relation is further incomplete when generalized to asynchronous multiparty sessions with mixed choice, due to the "peculiarity [...] that, apart from a pair of inactive session types, only inputs and outputs from/to a same participant can be related" [Ghilezan et al. 2019b]. The complexity of the subtyping relation in [Ghilezan et al. 2019b] is not mentioned.

Unlike subtyping relations for synchronous sessions which preserve language inclusion, subtyping relations for asynchronous sessions instead focus on deadlock-free optimizations that permute roles' local order of send and receive actions, also called *asynchronous message reordering*, or

AMR [Cutner et al. 2022]. First proposed for binary sessions by Mostrous and Yoshida [Mostrous and Yoshida 2009], and for multiparty sessions by Mostrous et al. [Mostrous et al. 2009], this notion of subtyping does not satisfy subprotocol fidelity in general; indeed, in some cases, the set of behaviors recognized by a supertype is entirely disjoint from that of its subtype [Bravetti et al. 2021a]. Asynchronous subtyping was shown to be undecidable for both binary and multiparty session types [Lange and Yoshida 2017; Bravetti et al. 2018]. Existing works are thus either restricted to binary protocols [Lange and Yoshida 2017; Bravetti et al. 2021a, 2018; Bacchiani et al. 2021], prohibit non-deterministic choice involving multiple receivers [Ghilezan et al. 2021; Bravetti et al. 2021b], or make strong fairness assumptions on the network [Bravetti et al. 2021b].

The connection between session subtyping and behavioral contract refinement has been studied only in the context of binary session types, and is thus out of scope of our work. We refer the reader to [Ghilezan et al. 2019b] for a survey.

Part II

Implementation

7 | Rocq Mechanization

7.1 Introduction

The difficulty of correctly implementing distributed, message-passing protocols is mirrored by the difficulty of proving the metacorrectness of their verification methodologies. Unsound implementability checks in verification frameworks may result in implementations that exhibit communication errors or deadlocks, whereas incomplete implementability checks undermine their utility. Despite the fact that the vast majority of existing implementability checks are conservative and do not aim for completeness, multiple unsound implementability checks have been proposed [Caires and Pérez 2016; Chen 2015; Deniélou et al. 2012; Deniélou and Yoshida 2012; Toninho and Yoshida 2017], in addition to false claims about the decidability of implementability for various protocol classes [Gheri et al. 2022] that were later refuted.

Mechanization has proven to be an effective way to fortify the correctness of pen-and-paper results. In the domain of process calculi, a mechanization of [Lanese et al. 2008] called HO-Core [Maksimovic and Schmitt 2015] revealed and subsequently fixed several major flaws in the existing proofs. Proof assistants especially excel at preventing inexhaustive case analysis, which was shown by [Finkel and Lozes 2017] to be the cause of erroneous prior works claiming the decidability of the realizability and synchronizability problems for systems of asynchronously communicating state machines.

However, all existing works in the intersection of mechanization and protocol implementability consider restricted implementation models that support only synchronous communication [Tirore et al. 2023; Hirsch and Garg 2022] or asynchronous communication with directed choice [Castro-Perez et al. 2021]. Moreover, specifications are restricted to protocols with finitely many participants and completeness is stated relative to projection operators that are themselves incomplete for implementability.

In this chapter, we present a mechanization of the precise implementability characterization for a large class of protocols that subsumes many existing formalisms in the literature presented in Chapter 3. Our protocols and implementations model asynchronous commmunication, and can exhibit infinite behavior. Our semantic model of protocols unifies two distinct formalisms from Chapter 2 under one general definition, which is capable of expressing syntactic formalisms from other frameworks. We improve upon the results in Chapter 3 by simplifying existing proof arguments, elaborating on the construction of canonical implementations, and even uncovering a subtle bug in the semantics for infinite words. As a corollary of our mechanization, we show that the characterization in Chapter 3 applies even to protocols with infinitely many participants. We also contribute a reusable library for reasoning about generic communicating state machines, which can serve as a basis for formalizing other theoretical results in concurrency theory.

7.2 Mechanization

We focus our exposition in this chapter on aspects of the Rocq mechanization that improve upon the pen-and-paper proofs from Chapter 3. In §7.2.1, we present our purely semantic definition of protocols, which collapses the distinction between GCLTS and symbolic protocols in Chapter 2, and can easily encode existing protocol models. In §7.2.2, we discuss a subtle flaw identified in the infinite word semantics introduced in [Majumdar et al. 2021a] and inherited by several subsequent works [Stutz and Zufferey 2022; Stutz 2023; Li et al. 2023a, 2024; Stutz 2024b; Li et al. 2025c], its implications on the pen-and-paper proofs, and propose a revised infinite word semantics.

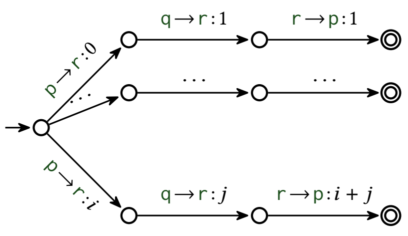


Figure 7.1: Addition GCLTS.

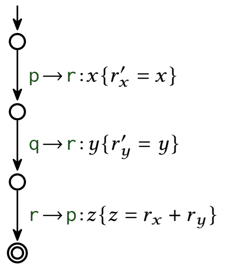


Figure 7.2: Addition symbolic protocol.

As a byproduct, we obtain the generalization from finite to infinite participant sets for free. In §7.2.3, we present our novel existence proof of canonical implementations. In §7.2.4, we present a simplification to a key soundness lemma that features nested induction.

7.2.1 Protocols as Labeled Transition Systems

In Chapter 2, we introduce *symbolic protocols* as an additional model for finitely representing potentially infinite GCLTS. Symbolic protocol states store a set of registers, and transitions are labeled with dependent predicates that can refer to communication variables and register variables, and can thus describe register updates. The semantics and implementability of a symbolic protocol is defined in terms of the concrete GCLTS it represents.

For illustration purposes, the GCLTS and symbolic protocol representations of a simple addition protocol between three participants p, q and r are depicted in Fig. 7.1 and Fig. 7.2. In the protocol, participants p and q send two natural number values x and y to participant r, who replies to participant p with the sum $z = x + y$, after which the protocol terminates.

In Chapter 3, we extend the Coherence Conditions to a set of Symbolic Coherence Conditions for algorithmically checking implementability of symbolic protocols, as well as investigating complexity of various decidable symbolic protocol fragments.

Thanks to Rocq's type universe, we unify the two disparate definitions under a single formal definition in our mechanization, which represents protocols simply as an LTS parametric in a state and alphabet type, containing a transition relation, an initial state, and a final state relation.

```
Record LTS {A: Type} :=
  mkLTS { transition: State -> A -> State -> Prop;
          s0: State;
          final: State -> Prop; }.
```

We define LTS semantics using an inductive relation to represent reachability, lists to represent finite traces, and streams to represent infinite traces. Despite the apparent inconvenience imposed by the type-level distinction between finite and infinite words, we will see in §7.2.4 that we can greatly delay the acknowledgement of this distinction in key proofs, and moreover, that doing so simplifies the existing pen-and-paper proofs.

7.2.2 Infinite Protocol Semantics

In this section, we examine asynchronous protocol semantics for infinite words.

The protocol semantics of $\mathcal{S}$ is defined in steps: we begin with the LTS semantics of $\mathcal{S}$, then apply a homomorphism `split` to obtain a set of asynchronous words that remain "synchronously ordered", i.e. matching send and receive events are adjacent to each other. In an asynchronous network with peer-to-peer, FIFO channels, certain events can be reordered, and are thus considered independent. For example, the synchronous trace $\mathtt{p} \to \mathtt{q} : m \cdot \mathtt{r} \to \mathtt{s} : m$ yields the asynchronous trace $u_1 = \mathtt{p} \triangleright \mathtt{q}!m \cdot \mathtt{r} \triangleright \mathtt{s}!m \cdot \mathtt{s} \triangleleft \mathtt{r}?m \cdot \mathtt{q} \triangleleft \mathtt{p}?m$, as well as $u_2 = \mathtt{r} \triangleright \mathtt{s}!m \cdot \mathtt{p} \triangleright \mathtt{q}!m \cdot \mathtt{s} \triangleleft \mathtt{r}?m \cdot \mathtt{q} \triangleleft \mathtt{p}?m$, in which the independent sends by $\mathtt{p}$ and $\mathtt{r}$ are reordered.

We call two words $w, w' \in \Sigma_{async}^{\omega}$ *indistinguishable* when any asynchronous implementation recognizing one word necessarily recognizes the other. Note that indistinguishability is specific to the assumed communication architecture: two words that are indistinguishable in a peer-to-peer FIFO setting may not be in a mailbox setting.

Allowing the semantics of global protocols to selectively exclude indistinguishable behaviors, e.g. by including u_1 but excluding u_2, would render protocols spuriously non-implementable. Thus, we desire for our protocol semantics to be closed under this notion of indistinguishability. For finite words, the indistinguishability relation is intuitive to formalize. Prior works that give language-theoretic semantics to session types, such as [Majumdar et al. 2021a], define indistinguishability in terms of a binary relation on asynchronous events capturing when they can be reordered, and a notion of *channel compliance* that captures valid traces with respect to peer-to-peer, FIFO semantics.[1] In the message sequence chart literature, linearizations are required to satisfy the union of per-participant total orders and the send-before-receive partial order on events, coinciding with the definition from [Majumdar et al. 2021a]. A key observation is that in pairs of indistinguishable finite words, the sequence of events for each participant is identical. Thus, to show that *any* asynchronous implementation recognizes w', we do not need to know more about each participant's local implementation beyond the fact that it accepts $w\Downarrow_{\Sigma_\mathsf{p}}$, which is given from the fact that the implementation as a whole recognizes w.

For infinite words, however, indistinguishability can no longer be defined purely alphabetically. Consider the pair of infinite words $v_1 = \mathsf{p} \triangleright \mathsf{q}!m^\omega$ and $v_2 = \mathsf{r} \triangleright \mathsf{s}!m \cdot \mathsf{p} \triangleright \mathsf{q}!m^\omega$. Are v_1 and v_2 indistinguishable? On our previous notion of indistinguishability, the answer is unfortunately, no. The fact that w is a trace of an arbitrary asynchronous implementation gives us no information about the local implementation of participant r, yet to show that w' is also a trace of said implementation, we need to additionally know that r's local implementation admits the trace $\mathsf{r} \triangleright \mathsf{s}!m$. This discrepancy arises from the fact that infinite traces in an asynchronous implementation can infinitely reorder independent events, in this case every occurrence of $\mathsf{p} \triangleright \mathsf{q}!m$ with $\mathsf{r} \triangleright \mathsf{s}!m$, achieving the effect of indefinitely postponing $\mathsf{r} \triangleright \mathsf{s}!m$.

Equipped with an understanding of the importance of indistinguishability-closed global se-

[1]We identify a minor erratum in the original formulation of the indistinguishability relation [Majumdar et al. 2021a] used in later works [Stutz 2023; Li et al. 2023a, 2024]: cases (3) and (4) are not symmetric, and thus the relation is not an equivalence relation as claimed.

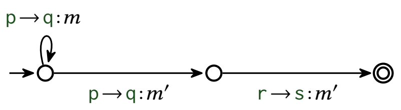


Figure 7.3: Example infinite protocol $\mathcal{S}_{inf}$.

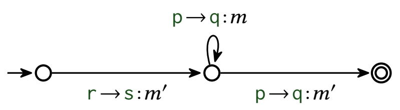


Figure 7.4: Example infinite protocol $\mathcal{S}'_{inf}$.

mantics, we revisit the definition of infinite protocol semantics as stated in [Li et al. 2025c]:

$$C^{\sim}(\mathcal{S})^{\omega} = \{w' \in \Sigma_{async}^{\omega} \mid \exists w \in \Sigma_{async}^{\omega}.\ w \in \texttt{split}(\mathcal{L}(\mathcal{S})) \land \forall v' \leq w'.\ \exists u, u' \in \Sigma_{async}^{*}.$$
$$v' \cdot u' \text{ is channel-compliant } \land\ u \leq w\ \land \forall \mathsf{p} \in \mathcal{P}.\ (v' \cdot u')\Downarrow_{\Sigma_{\mathsf{p}}} = u\Downarrow_{\Sigma_{\mathsf{p}}}\}\ .$$

We show via counterexample that $C^{\sim}(\mathcal{S})^{\omega}$ is not indistinguishability-closed. Consider the simple protocol depicted in Fig. 7.3, involving four participants p, q, r, s and two message values m, m'. As per the above definition, $\mathcal{L}(\mathcal{S}_{inf})$ does not include the infinite word $\mathsf{r} \triangleright \mathsf{s}!m \cdot \mathsf{p} \triangleright \mathsf{q}!m^{\omega}$. In contrast, $\mathcal{L}(\mathcal{S}'_{inf})$, whose protocol is obtained by a simple state renaming of Fig. 7.4 and is depicted in Fig. 7.4, does include $\mathsf{r} \triangleright \mathsf{s}!m \cdot \mathsf{p} \triangleright \mathsf{q}!m^{\omega}$.

Before proposing a revised infinite word semantics that resolves this discrepancy, we discuss the implications of this counterexample on the results from [Li et al. 2025c]. It is easy to verify that $\mathcal{S}_{inf}$ is a GCLTS and satisfies CC. However, $\mathcal{S}_{inf}$ is not implementable: there exists no CLTS that recognizes the finite word $\mathsf{p} \triangleright \mathsf{q}!m^{n} \cdot \mathsf{p} \triangleright \mathsf{q}!m' \cdot \mathsf{r} \triangleright \mathsf{s}!m'$ for all values of $n \in \mathbb{N}$ yet does not recognize the infinite word $\mathsf{r} \triangleright \mathsf{s}!m \cdot \mathsf{p} \triangleright \mathsf{q}!m^{\omega}$. This contradicts the soundness of the Coherence Conditions as stated in [Li et al. 2025c]. The error lies in the case for infinite words in the proof of [Li et al. 2025c, Lemma 4.9], which concludes from the fact that every prefix of an infinite word w in the canonical implementation has a non-empty intersection run set $I(w)$, that $w \in C^{\sim}(\mathcal{S})^{\omega}$. To show that $w \in C^{\sim}(\mathcal{S})^{\omega}$, one needs to find a witness infinite run ρ in $\mathcal{S}$, such that for every prefix $v' \leq w$, there exists an extension u' and a prefix of rho ρ'_{v} such that for all participants, the events prescribed by ρ'_{v} and $v' \cdot u'$ are identical. To show the existence of such a run, the authors appeal to König's Lemma, and argue that in a finitely-branching infinite tree containing possible run prefixes for every prefix of w', there exists a ray representing an infinite run. This argument appears likewise inherited from earlier works on finite, multiparty session types [Majumdar et al. 2021a]. We discover that not only is König's Lemma not applicable in the infinite setting of [Li et al. 2025c] where GCLTS states can have infinitely many transitions, the existence of a ray is insufficient to prove membership of w in $\mathcal{C}^{\sim}(\mathcal{S})^{\omega}$. The latter implies that the proof using König's Lemma in prior works such as [Majumdar et al. 2021a; Li et al. 2023a] is flawed: indeed, $\mathcal{S}'_{inf}$ is expressible in the multiparty session type fragments defined in these works that assume finitely many participants, states and transitions. The gap in the reasoning lies in showing that the infinite run obtained from König's Lemma is indeed a suitable existential witness required by the infinite protocol semantics. In the infinite tree constructed for $\mathcal{S}_{inf}$ and word $\mathtt{r} \triangleright \mathtt{s}!m' \cdot \mathtt{p} \triangleright \mathtt{q}!m^{\omega}$, the prefix $\mathtt{r} \triangleright \mathtt{s}!m$ contributes a vertex labeled with the run prefix $\mathtt{p} \to \mathtt{q} : m \cdot \mathtt{p} \to \mathtt{q} : m' \cdot \mathtt{r} \to \mathtt{s} : m'$. Subsequent prefixes of the form $\mathtt{r} \triangleright \mathtt{s}!m' \cdot \mathtt{p} \triangleright \mathtt{q}!m^{n}$ contribute vertices labeled with run prefixes $\mathtt{p} \to \mathtt{q} : m^{n} \cdot \mathtt{p} \to \mathtt{q} : m' \cdot \mathtt{r} \to \mathtt{s} : m'$. A ray exists in this finite-degree, infinite tree representing the run $\mathtt{p} \to \mathtt{q} : m^{\omega}$. This is clearly an infinite run in $\mathcal{S}_{inf}$, but unfortunately does not satisfy the conditions required to show membership of w' in $\mathcal{L}(\mathcal{S}_{inf})$: for prefix $\mathtt{r} \triangleright \mathtt{s}!m'$ of w, there exists no prefix of $\mathtt{p} \to \mathtt{q} : m^{\omega}$ that matches $\mathtt{r}$'s events.

Fortunately, the flawed infinite word semantics from [Li et al. 2025c] can easily be amended to accurately reflect the desired, indistinguishability-closed semantics. This amendment is reflected in the published version of [Li et al. 2025c]: [Li et al. 2025b], as well as in Chapter 2 of this thesis. Our revised infinite word semantics is as follows:

$$\begin{aligned}\mathcal{C}^{\sim}_{alt}(\mathcal{S})^{\omega} = \{ w' \in \Sigma^{\omega}_{async} \mid \forall v' \leq w'.\ \exists \rho \in \Gamma^{*}_{sync}, u' \in \Sigma^{*}_{async}.\ \rho \in \mathrm{pref}(\mathcal{L}(\mathcal{S})) \wedge \\ v' \cdot u' \text{ is channel-compliant } \wedge\ \forall \mathtt{p} \in \mathcal{P}.\ (v' \cdot u') \Downarrow_{\Sigma_{\mathtt{p}}} = \mathtt{split}(\rho) \Downarrow_{\Sigma_{\mathtt{p}}} \}\ .\end{aligned}$$

$\mathcal{C}^{\sim}_{alt}(\mathcal{S})^{\omega}$ swaps the first two quantifiers in the original definition, and weakens the requirement that w come from an infinite run to the requirement that w come from a finite run prefix (that could be part of a finite or infinite maximal run in $\mathcal{S}$). We hypothesize that this revised condition faithfully represents what prior works intended to capture with their infinite protocol semantics. It also more closely matches simulation-based notions of trace equivalence, for example in [Zhou et al. 2020]. This is further evidenced by the fact that the requisite changes to the overall proof were minimal: the flawed König's Lemma argument could simply be omitted in favor of appealing directly to the intersection set non-emptiness inductive invariant, and the completeness proof remained largely unchanged. The latter is due to the fact that for any infinite word w, $w \in C^{\sim}(\mathcal{S})^{\omega} \implies w \in C^{\sim}_{alt}(\mathcal{S})^{\omega}$.

7.2.3 Constructing Canonical Implementations

Showing that a global protocol is implementable amounts to finding a witness CLTS that implements it. The soundness proof of *CC* in Chapter 3 chooses a particular CLTS as witness, namely the *canonical implementation* (Definition 3.6). Our proofs in Chapter 3 assume the existence of this canonical implementation for any protocol. Establishing the existence of a canonical implementation formally in our mechanization requires constructing an explicit, albeit non-constructive, witness CLTS. The construction is conceptually straightforward; nonetheless, we illustrate key steps here as it is novel to our mechanization.

We begin by observing that because canonicity is defined on a per-participant basis, and with respect to an LTS that is deadlock-free, the definition can be weakened to use the LTS semantics of $\mathcal{S}$ rather than its protocol semantics. The weaker definition avoids reasoning about asynchronous reorderings and channel compliance, and is formalized in Rocq as follows.

In the Rocq definitions below, `S` is a protocol of type `LTS SyncAlphabet State`, and `p` is a participant. We choose `State -> Prop` for the state type of local implementations, so `S_p` is an LTS of type `LTS AsyncAlphabet (State -> Prop)`.

```
Definition canonical_implementation_local_naive S p S_p :=
  (forall w:FinAsyncWord, is_finite_word S_p w ->
  exists w':FinSyncWord, is_finite_word S w' /\ wproj (split w') p = w)
  /\
  (forall w:FinSyncWord, is_finite_word S w ->
  is_finite_word S_p (wproj (split w) p))
  /\
  (forall w:FinAsyncWord, is_trace S_p w ->
  exists w':FinSyncWord, is_trace S w' /\ wproj (split w') p = w)
  /\
  (forall w:FinSyncWord, is_trace S w ->
  is_trace S_p (wproj (split w) p)).
```

The four conjuncts correspond to four inclusions that altogether define the two equalities in Definition 3.6, and need to be stated separately due to the type mismatch between finite and infinite words.

Our construction for each participant's local implementation can be expressed simply as a composition of two purely automata-theoretic operations: applying the homomorphism $\Downarrow_{\Sigma_p}$ for each participant, followed by determinization. This coincides with the subset construction automaton as defined in Chapter 5, which we name our definitions after. Formally, for each participant $p \in \mathcal{P}$, the result of the second step is an LTS over $\Sigma_p \cup \{\epsilon\}$. To avoid introducing this compounded alphabet and reasoning about identity elements, we define both operations declaratively in one shot, to obtain a local LTS over the alphabet `AsyncAlphabet`, whose states are of type `State -> Prop`, representing subsets of Q.

The initial state is defined relationally as the set of all states reachable on ϵ from s_0 in $\mathcal{S}$. States in the subset construction are the set of non-empty subsets of states in $\mathcal{S}$. Final states are defined relationally as sets of states containing at least one final state from $\mathcal{S}$.

```
Definition initial_subset_construction_state S p :=
 fun s => exists (w : list SyncAlphabet), lts.Reachable S (s0 S) w s
```

```
  /\ wproj (split w) p = [].

Definition subset_construction_state S p :=
  fun lstate => exists (s : State), lstate s.

Definition final_subset_construction_state S p :=
  fun lstate => subset_construction_state S p lstate /\
  exists (s : State), lstate s /\ final S s.
```

The transition relation describes triples (`ls, a, ls'`) where `ls` is a pre-state in the subset construction, `a` is an asynchronous alphabet symbol in `p`'s restricted alphabet, and `ls'` is a post-state. The relation states that `ls'` contains all states from $\mathcal{S}$ that are either an immediate post-state of some state s in `ls`, or is ϵ-reachable from an immediate post-state.

```
Definition subset_construction_transition_relation S p :=
  fun lstate1 a lstate2 => is_active p a
  /\ subset_construction_state S p lstate1
  /\ subset_construction_state S p lstate2
  /\ forall (s':State), lstate2 s' <->
  (exists (s:State), lstate1 s /\ transition S s (async_to_sync a) s')
  \/
  (exists (s s_inter:State), lstate1 s /\
   transition S s (async_to_sync a) s_inter /\
   exists (v_epsilon:list SyncAlphabet),
    lts.Reachable S s_inter v_epsilon s' /\
    wproj (split v_epsilon) p = []).
```

The former two conjuncts are implied by the latter two conjuncts together with the definition of final states in the subset construction. The latter two conjuncts state that every asynchronous trace in a participant's canonical local implementation corresponds to a synchronous trace in $\mathcal{S}$, and every synchronous trace in $\mathcal{S}$ corresponds to an asynchronous trace in the participant's canonical local implementation.

Unfortunately, these two properties are not themselves inductive: in both cases, the induction hypothesis is not strong enough to show that the respective traces can be extended. We state and prove two inductive invariants that explicitly quantify over states of $\mathcal{S}$ in a participant's canonical local implementation S_p, and weaken them to obtain the third and fourth conjuncts. The strengthened inductive properties respectively state that for every reachable state ls on some asynchronous word w in S_p, for every global state s in ls, one can find a corresponding synchronous word w' such that w' and w agree on participant p's events, and $\mathcal{S}$ reaches s on w'; conversely, for every reachable global state s on some synchronous word w in $\mathcal{S}$, one can find a corresponding local state ls' and asynchronous word w' such that w' and w agree on participant p's events, and S_p reaches ls' on w'.

Finally, we define the canonical CLTS by mapping each participant to their subset construction. To show that the map thus defined is indeed a CLTS, we additionally need to prove that each local implementation is deterministic, and moreover operates on its own restricted alphabet. Both proofs are straightforward by definition of the subset construction; our proof of determinism uses the axioms of functional and propositional extensionality from Rocq's Logic library to establish the equality of local states of type State -> Prop. We conclude with the existence lemma:

```
Lemma canonical_implementation_exists :
  forall (S : @LTS SyncAlphabet State),
  deadlock_free S ->
  exists (T : CLTS),
  @canonical_implementation (State -> Prop) S T.
```

7.2.4 Simplification of Soundness

The core argument for soundness in Section 3.3 relies on proving the following inductive invariant:

> Let $\mathcal{S}$ be a protocol satisfying CC, and let $\{\!\{T_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ be a canonical CLTS for $\mathcal{S}$. Let w be a trace of $\{\!\{T_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}}$. Then, $I(w) \neq \emptyset$.

The set $I(w)$ contains finite or infinite maximal runs in $\mathcal{S}$ that are *possible* with respect to the trace w. Formally, $\rho \in I(w)$ means that for every participant $\mathsf{p} \in \mathcal{P}$, $w\Downarrow_{\Sigma_{\mathsf{p}}} \leq \mathtt{split}(\rho)\Downarrow_{\Sigma_{\mathsf{p}}}$, i.e. each participant's local events in w agree with what ρ prescribes. The proof proceeds by induction on the length of w, with case analysis in the inductive step on whether the next event is a send or receive event.

The non-emptiness of $I(w)$ amounts to an existential quantification over a disjunction. In our mechanization, however, due to the type-level distinction between finite and infinite runs, this property takes the form of a disjunction over existentials:

```
Definition I_set_non_empty (S: LTS) (w: FinAsyncWord) :=
 (exists (run: FinSyncWord), finite_possible_run S run w)
 \/
 (exists (run: InfSyncWord), infinite_possible_run S run w).
```

Although our soundness arguments from Section 3.3 are mechanizable using this definition of intersection set non-emptiness, doing so would involve repetitive reasoning to deal with finite and infinite runs separately that does not shed additional insight on the problem. We instead prove that every canonical CLTS trace has a possible run *prefix*. Our new inductive invariant factors out the distinction between finite and infinite runs, and is additionally more expressive than its pen-and-paper counterpart: it makes explicit the construction of a possible run prefix for wx from one for w. When x is a receive event, our lemma states that the exact same run prefix can be reused. When x is a send event, a run prefix can be constructed incrementally by processing w in increasing length order, and appealing to CC to incrementally extend a prefix of the possible run prefix for wx.

We focus the exposition below on our simplified proof for the inductive step when x is a send event. We restate the relevant lemma below.

Lemma 7.1 (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 implementation 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 $\alpha \cdot s_{pre} \leq \rho'$.*

The *unique splitting* of a run for a participant with respect to a trace is the largest prefix of the run that matches the participant's actions in the trace, formalized as follows:

```
Definition is_alpha (run alpha:FinSyncWord) (w:FinAsyncWord) p :=
  prefix alpha run /\ wproj w p = wproj (split alpha) p /\
  (forall (u: FinSyncWord), wproj w p = wproj (split u) p ->
  prefix u run -> prefix u alpha).
```

For example, the unique splitting of run $\rho = \mathsf{p} \to \mathsf{q} : m \cdot \mathsf{r} \to \mathsf{s} : m \cdot \mathsf{r} \to \mathsf{q} : m \cdot \mathsf{q} \to \mathsf{p} : m$ for participant p with respect to trace $u = \mathsf{p} \triangleright \mathsf{q}!m \cdot \mathsf{r} \triangleright \mathsf{s}!m \cdot \mathsf{r} \triangleright \mathsf{q}!m$ is $\mathsf{p} \to \mathsf{q} : m \cdot \mathsf{r} \to \mathsf{s} : m \cdot \mathsf{r} \to \mathsf{q} : m$, because p has only completed the first event prescribed by ρ in u, namely sending m to r, but has not completed the second event, namely receiving m from q. Because the two synchronous events in between these two events in ρ do not concern p, they are included in the *largest* prefix. If a run disagrees with a trace on some participant's actions, the unique splitting is ϵ, for example ρ's unique splitting for participant r with respect to trace $\mathsf{r} \triangleright \mathsf{s}!m'$.

Our adapted formalization of the lemma is thus stated as follows:

```
Lemma send_preserves_run_prefixes_finite :
  forall S T w x rho_fin alpha,
GCLTS S -> NMC S -> SCC S -> RCC S ->
canonical_implementation S T ->
is_clts_trace T w -> is_clts_trace T (w ++ [x]) -> is_snd x ->
possible_run_prefix S rho_fin w ->
is_alpha rho alpha w (sender_async x) ->
exists (rho': FinSyncWord),
 prefix alpha rho' /\ possible_run_prefix S rho' (w ++ [x]).
```

The proof of Lemma 7.1 relies on a nested induction argument. We illustrate the key steps in order to elucidate the structure of the nested induction and explain our simplified proof. From the induction hypothesis, we are granted a canonical CLTS trace `w` and a possible run prefix for `w`. Let the send extention to `w` be `x = Snd p q m`. We can then define the largest prefix of `rho` matching `w` for participant `p`, and because the premise grants that `alpha <> rho`, there must exist a next action prescribed by `rho` for `p`, which we denote `l`. As a reminder, since `rho` is a run of the global protocol, which is an LTS over the synchronous alphabet, `l` is a synchronous alphabet symbol. By the induction hypothesis, `rho` is compliant with all participants:

```
forall (p: participant), prefix (wproj w p) (wproj (split rho) p)
```

The induction step asks to construct an existential witness for a new possible run prefix, `rho'`, that is compliant with `wx`. In the case that `l = Event p q m`, we can directly reuse `rho` as our witness, and the three conjuncts required of `rho'` are trivially satisfied when `rho' = rho`. When this is not the case, we must construct a different witness. We first appeal to Send Coherence Condition to show that we can find a different continuation from `alpha` that agrees with `x`, in other words, **`l'`** `= [Event p q m]` and `alpha ++` **`l'`** is a run in the global protocol.

With this extension and removal of the original suffix from `rho`, however, we are left only with a guarantee about `p`'s compliance:

```
prefix (wproj (w ++ [x]) p) (wproj (split (alpha ++ [Event p q m])) p)
```

In the case that all of the actions in `w` were already contained in `alpha`, we can use `alpha ++ [l']` directly as our witness for `rho'`. However, in the case that some of `w`'s actions were contained in the now removed suffix, it is no longer true that all participants are compliant with `alpha ++ [l']`. Therefore, the next step of the proof involves restoring a suffix that is "long enough" to contain all of the actions that were originally in `w`.

Our pen-and-paper argument for suffix restoration is algorithmic in nature, and is captured by the pseudocode in Algorithm 2. The algorithm initializes the candidate run ρ_c as $\alpha \cdot \mathbf{l'}$ appended to an arbitrary run suffix β to form a maximal run. The outer while loop then "fixes" disagreements between w and the current candidate run ρ_c one symbol at a time, updating ρ_c after each fix. Termination is guaranteed by the fact that w has finite length and that each event in w is fixed at most once. The outer while loop invariant relates ρ'_c with ρ_c, and guarantees that the largest common prefix shared by ρ'_c and ρ_c between each loop iteration is strictly increasing. Because the initial candidate run is picked such that it includes $\alpha \cdot \mathbf{l'}$ as a prefix, and the common prefix between runs can only get longer, it holds by transitivity that when the while loop terminates, the final candidate run must have $\alpha \cdot \mathbf{l'}$ as a prefix, and furthermore is compliant with all events in w.

Formalizing the above algorithm in addition to its loop invariants would require a custom inductive predicate that relates the candidate run with disagreeing events in w. The fact that the loop invariant depends on both the current and previous candidate run introduces significant additional complexity.

We find a weaker inductive invariant that eliminates this dependency: it suffices to show that

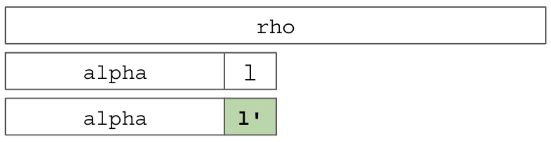


Figure 7.5: Induction hypothesis.

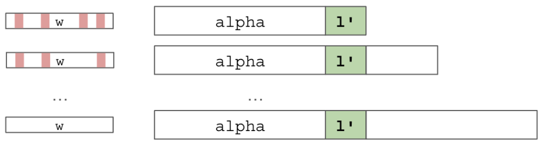


Figure 7.6: Inner induction hypothesis.

Algorithm 2 Algorithmic representation of Lemma 4.16 [Li et al. 2025b]

```
  ▷ Let $\rho_c$ be $\alpha \cdot l' \cdot \beta$, where $\beta$ is an arbitrary maximal suffix
  $\rho_c \leftarrow \alpha \cdot l' \cdot \beta$
  while $\neg(\forall \mathsf{p} \in \mathcal{P}.\ w\Downarrow_{\Sigma_\mathsf{p}} \leq \mathtt{split}(\rho_c)\Downarrow_{\Sigma_\mathsf{p}})$ do
      ▷ $i$ is the index of the earliest disagreeing event in $\rho_c$
      $i \leftarrow length(\rho_c)$
      ▷ $j$ iterates over all prefixes of $w$
      $j \leftarrow 0$
      for $j \in \{0..length(w)\}$ do
          $k \leftarrow \max\{k' \mid \forall \mathsf{p} \in \mathcal{P}.\ \mathtt{split}(\rho_c[0..k'-1])\Downarrow_{\Sigma_\mathsf{p}} \leq w[0..j]\Downarrow_{\Sigma_\mathsf{p}}\}$
          if $k < i$ then
              $i \leftarrow k$
          $j \leftarrow j + 1$
      ▷ $y$ is the earliest disagreeing event in $\rho_c$
      $y \leftarrow \mathtt{split}(\rho_c)[i]$
      ▷ y' is obtained from SCC to no longer disagree with $w$
      $\rho_c \leftarrow \rho_c[0..i-1] \cdot$ y'
```

$\alpha \cdot$ **l'** $\leq \rho_c$ remains a prefix of the candidate run. This holds trivially upon entry to the while loop, and is preserved by each iteration from the fact that α comes from the original ρ that is compliant with w, and thus no events in w can disagree with events in α.

In our new inductive invariant, $\alpha \cdot$ **l'** can now be treated as a constant. We convert our pen-and-paper algorithmic reasoning to the following inner induction hypothesis:

```
H_inner: forall (w': FinAsyncWord), prefix w' w ->
  (exists (beta': FinSyncWord),
   possible_run_prefix S (alpha ++ [Event p q m] ++ beta') w')).
```

We prove `H_inner` directly by induction on prefixes of `w` using `rev_ind` from the standard `List` library. Fig. 7.6 visualizes the simplified inductive argument: the red symbols in `w` depict disagreeing events in `w` as a result of removing the suffix from `rho`.

To prove `send_preserves_run_prefixes_finite`, we needed to consider cases glossed over in our pen-and-paper proof, and in some case develop arguments from scratch. For example, in the special case when `alpha` is a possible run prefix for participant `p` for trace `w`, but prescribes exactly as many events as are in `w`, we needed to show that either `w` is a maximal CLTS trace, or a different `alpha` can be found which prescribes more events, by appealing to the sink-finality of $\mathcal{S}$.

7.3 Related Work

Most closely related to our mechanization effort are the works [Castro-Perez et al. 2021] and [Tirore et al. 2023]. Zooid [Castro-Perez et al. 2021] is a mechanized domain-specific language for specifying and implementing asynchronous multiparty session types. [Tirore et al. 2023] mechanizes the soundness and completeness proofs for the projection operator for synchronous multiparty session types proposed in [Ghilezan et al. 2019a]. A key conceptual difference is that our proofs follow a semantic argument grounded in formal language theory whereas both [Castro-Perez et al. 2021] and [Tirore et al. 2023] follow more standard syntactic arguments. More fundamentally, the class of protocol specifications considered in this paper generalizes that of [Castro-Perez et al. 2021; Tirore et al. 2023] along several dimensions: [Tirore et al. 2023] considers synchronous rather than asynchronous communication and both works, internal choice syntactically disallows a sender from choosing among multiple receivers. Moreover, both papers restrict specifications to finitely many participants and states, and abstract message values in terms of simple types without data refinements. Finally, the notion of completeness considered in [Tirore et al. 2023] is defined relative to the coinductive definition of endpoint projection introduced in [Ghilezan et al. 2019a]. The latter is itself incomplete for our semantically defined notion of implementability. The end-point projection of [Castro-Perez et al. 2021] is likewise incomplete.

Pirouette [Hirsch and Garg 2022] introduces a language of *functional choreographies* that are converted to a distributed implementation via endpoint projection. The language supports session delegation and higher-order functions, neither of which we include in our model of GCLTS. However, functional coreographies are much more restricted in their distributed behavior than the protocols in our model: communication is synchronous and all participants must remain in lock step. The latter is enforced by requiring that the programmer inserts potentially redundant synchronization messages into the coreography. A proof that the implementations obtained by projection are deadlock-free has been mechanized in Rocq. Similar to [Tirore et al. 2023], the completeness theorem is stated relative to completeness of syntactic projection rather than semantic implementability.

There is a large number of other recent mechanization efforts for session type languages [Hinrichsen et al. 2020, 2022; Jacobs et al. 2023; Hinrichsen et al. 2024; Thiemann 2019; Rouvoet et al. 2020; Hinrichsen et al. 2021; Jacobs et al. 2022; Tassarotti et al. 2017; Ekici and Yoshida 2024]. However, these focus on the formalization of language semantics, compiler correctness, or on proving soundness of session type systems that check implementations against *local* types. The latter describe the behavior of individual participants or communication channels and may be obtained by prior endpoint projection from a global type or specified directly by the programmer. We therefore consider these efforts orthogonal to our work.

8 | SPROUT

8.1 Introduction

In this chapter, we present Sprout, the first sound and complete implementability checker for symbolic, multiparty protocols. Sprout takes as input a symbolic protocol, and first checks whether the protocol is GCLTS-eligible. If so, it proceeds to generate μCLP instances corresponding to the Symbolic Coherence Conditions from Chapter 3, which it then discharges to the μCLP solver MuVal [Unno et al. 2023]. If all instances return invalid, Sprout reports that the protocol is implementable; if one instance returns valid, Sprout reports non-implementable along with the specific states and transitions that violate implementability; otherwise Sprout returns inconclusive. Sprout is sound and complete relative to the completeness and soundness of MuVal.

Sprout extends the results from Chapter 3 with explicit GCLTS checking, optimized μCLP encodings of the Symbolic Coherence Conditions, and support for verification of functional correctness properties beyond implementability. We evaluate Sprout's expressivity, precision and efficiency against comparable tools [Zhou et al. 2020; Vassor and Yoshida 2024] on an expanded benchmark suite containing both implementable and non-implementable examples. Sprout is able to correctly classify protocols that are out of reach of its competitors, outperforming them in terms of expressivity and precision. In terms of efficiency, Sprout's performance is competitive. On multiparty protocols, its verification times vary with the size of the protocol and are largely

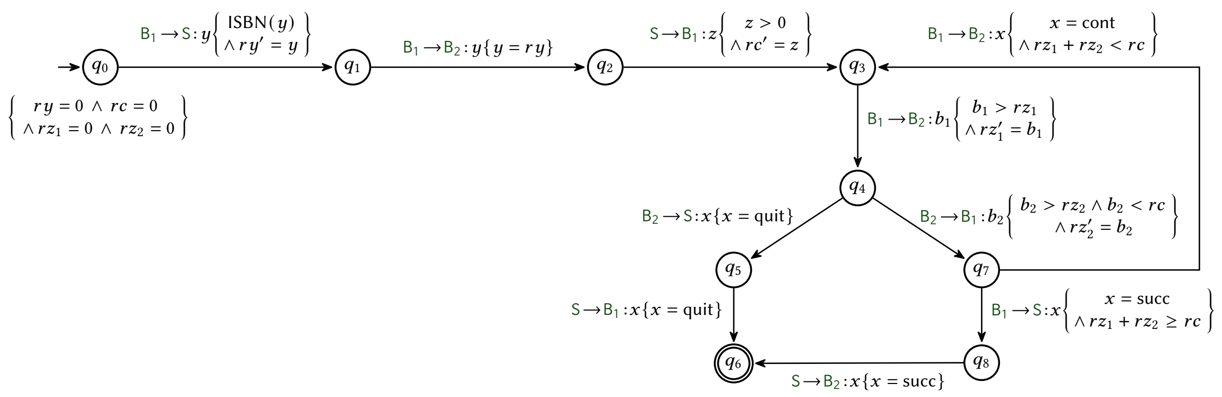

q_0
q_1
q_2
q_3
q_4
q_5
q_6
q_7
q_8
$\left\{ \begin{array}{c} ry = 0 \land rc = 0 \\ \land rz_1 = 0 \land rz_2 = 0 \end{array} \right\}$
$\mathsf{B}_1 \to \mathsf{S} : y \left\{ \begin{array}{c} \mathsf{ISBN}(y) \\ \land ry' = y \end{array} \right\}$
$\mathsf{B}_1 \to \mathsf{B}_2 : y\{y = ry\}$
$\mathsf{S} \to \mathsf{B}_1 : z \left\{ \begin{array}{c} z > 0 \\ \land rc' = z \end{array} \right\}$
$\mathsf{B}_1 \to \mathsf{B}_2 : x \left\{ \begin{array}{c} x = \mathsf{cont} \\ \land rz_1 + rz_2 < rc \end{array} \right\}$
$\mathsf{B}_1 \to \mathsf{B}_2 : b_1 \left\{ \begin{array}{c} b_1 > rz_1 \\ \land rz_1' = b_1 \end{array} \right\}$
$\mathsf{B}_2 \to \mathsf{S} : x\{x = \mathsf{quit}\}$
$\mathsf{B}_2 \to \mathsf{B}_1 : b_2 \left\{ \begin{array}{c} b_2 > rz_2 \land b_2 < rc \\ \land rz_2' = b_2 \end{array} \right\}$
$\mathsf{S} \to \mathsf{B}_1 : x\{x = \mathsf{quit}\}$
$\mathsf{B}_1 \to \mathsf{S} : x \left\{ \begin{array}{c} x = \mathsf{succ} \\ \land rz_1 + rz_2 \geq rc \end{array} \right\}$
$\mathsf{S} \to \mathsf{B}_2 : x\{x = \mathsf{succ}\}$

Figure 8.1: Candidate specification for the two-bidder protocol.

bottlenecked by the efficiency of MuVal, although remaining in the order of seconds in most cases. We envision Sprout as a complementary intermediate step in existing top-down code generation toolchains for multiparty protocols whose implementability checks are incomplete.

8.2 Overview

We introduce Sprout using the running example of the two-bidder protocol from Chapter 1. A candidate specification for the two-bidder protocol is depicted in Fig. 8.1. Sprout's input format closely follows the definition of *symbolic protocols*, formally defined over a set of participants $\mathcal{P}$ in Chapter 2. The input file for our candidate specification is given in Fig. 8.2.

```
Initial state: (0)
Initial register assignments: ry=0, rc=0, rz1=0, rz2=0
(0) B1->S:y{(y>987000000000/\y<9880000000000)/\ry'=y} (1)
(1) B1->B2:y{y=ry} (2)
(2) S->B1:z{z>0/\rc'=z} (3)
(3) B1->B2:b1{b1>rz1/\rz1'=b1} (4)
(4) B2->S:quit{quit=0} (5)
(5) S->B1:quit{quit=0} (6)
(4) B2->B1:b2{b2>rz2/\b2<rc/\rz2'=b2} (7)
(7) B1->S:succ{succ=1/\rz1+rz2>=rc} (8)
(8) S->B2:succ{succ=1} (6)
(7) B1->B2:cont{cont=2/\rz1+rz2<rc} (3)
Final states: (6)
```

Figure 8.2: Sprout input file for protocol specification in Fig. 8.1.

Before checking implementability, Sprout first determines GCLTS eligibility. GCLTSs satisfy four assumptions: sink-finality, sender-driven choice, determinism, and deadlock-freedom. Sink-finality states that only non-final states have outgoing transitions, sender-driven choice states that all outgoing transitions from the same state have a unique sender, determinism states that no transition can lead to two distinct post-states, and deadlock freedom states that every protocol run can be extended to a maximal run.

After confirming that our protocol is GCLTS-eligible, Sprout proceeds to generate μCLP instances corresponding to our three Symbolic Coherence Conditions: Symbolic Send Coherence, Symbolic Receive Coherence and Symbolic No Mixed Choice. Sprout generates the queries in negation form, and discharges them to the μCLP solver MuVal [Unno et al. 2023]. Sprout reports implementable if and only if all instances return invalid, indicating that all conditions are satisfied.

Unfortunately, Sprout reports a violation to Symbolic Send Coherence for B_2 and the transition: `(4) B2->B1:b2{b2>rz2/\b2<rc/\rz2'=b2} (7)`. The violation indicates the existence of two global protocol states both with control state q_4 that are indistinguishable from B_2's point of view, and a message value, such that sending the value to B_1 follows the protocol in one case but violates the protocol in the other. Closer inspection of this transition's constraint reveals that B_2 is required to send a bid that is strictly less than the price of the book c. However, c is not disclosed to B_2 during the protocol: B_2 is bidding in the dark. Thus, depending on the initial exchanges between B_1 and S, which are not observable to B_2, a bid could either satisfy or violate the middle conjunct, subsequently following or violating the entire protocol.

We can repair our candidate protocol by either omitting `b2<rc` from the aforementioned transition constraint, or by including a transition informing B_2 of the book's price before the bidding loop begins. Upon incorporating either fix, we find that all instances now return invalid as expected, and Sprout reports that the repaired two-bidder protocol is implementable in ~19s.

Sprout also provides support for the verification of functional correctness properties beyond implementability. For example, we can verify that the sum of B_1 and B_2's bids never decreases once they enter the bidding loop. This verification problem can be expressed in negation form as a μCLP instance as follows, where `stcon` is a least fixpoint predicate describing st-connectivity between two states in the global protocol:

```
exists  (s1: int) (ry1: int) (rc1: int) (rza1: int) (rzb1: int)
(s2: int) (ry2: int) (rc2: int) (rza2: int) (rzb2: int).
s1 > 3 /\
s2 > 3 /\
stcon s1 ry1 rc1 rza1 rzb1 s2 ry2 rc2 rza2 rzb2 /\
rza2 + rzb2 < rza1 + rzb1
s.t.
stcon (s1: int) ... : bool =mu
...
```

Sprout provides a suite of least and greatest fixpoint predicate definitions for defining custom verification queries that are then discharged to MuVal. MuVal confirms that this instance is indeed invalid in ~9s.

8.3 Implementation

Sprout is implemented in ~3500 lines of OCaml code. In this section, we describe aspects of its implementation, focusing on differences from the theory.

8.3.1 GCLTS Eligibility

Our Coherence Conditions are precise for the GCLTS fragment of symbolic protocols, namely protocols that satisfy sink-finality, sender-driven choice, determinism and deadlock-freedom. Sink-finality and sender-driven are syntactic conditions that are checked on the input protocol straightforwardly using OCaml functions. Determinism and deadlock freedom are undecidable in general. Sprout encodes the latter two as μCLP instances and discharges them to MuVal. We present the formal definition and μCLP encoding of each property below, assuming a symbolic protocol $\mathbb{S} = (S, R, \Delta, s_0, \rho_0, F)$ in the remainder of the section.

Determinism states that from a reachable protocol state, no transition can simultaneously satisfy two transition constraints that lead to two distinct post-states. Reachability is expressed as a least fixpoint in μCLP as follows:

Definition 8.1 (Reachability in symbolic protocol)**.** Let $s \in S$. Then,

$$\mathrm{reach}(s', \boldsymbol{r}') :=_{\mu} (\, s' = s_0 \land \boldsymbol{r}' = \rho_0 \,) \qquad \lor (\bigvee_{(s,\, \mathsf{p} \to \mathsf{q} : x\{\varphi\},\, s') \in \Delta} \exists x\, \boldsymbol{r}.\ \mathrm{reach}(s, r) \land \varphi \,)\ .$$

The reach predicate takes as its arguments a control state s' and a set of registers $\boldsymbol{r}'$, which together constitute a symbolic protocol state. The first disjunct covers the base case in which s' is the initial state, and $\boldsymbol{r}'$ satisfy the initial register assignments. The second disjunct ranges over all transitions with s' as the post-state, and represents following a transition to reach s', which requires the transition predicate φ to hold in addition to *reach* on the pre-state s.

Equipped with the predicate reach, determinism is defined as follows.

Definition 8.2 (Determinism of symbolic protocol)**.** $\mathbb{S}$ is deterministic when for each pair of transitions $s \xrightarrow{\mathsf{p} \to \mathsf{q} : x_1\{\varphi_1\}} s_1, s \xrightarrow{\mathsf{p} \to \mathsf{q} : x_2\{\varphi_2\}} s_2 \in \Delta$, the following is valid:

$$\forall x\, \boldsymbol{r}\, \boldsymbol{r}_1'\, \boldsymbol{r}_2'.\ \mathrm{reach}(s, \boldsymbol{r}) \land \varphi_1[x/x_1, \boldsymbol{r}_1'/\boldsymbol{r}'] \land \varphi_2[x/x_2, \boldsymbol{r}_2'/\boldsymbol{r}'] \implies s_1 = s_2 \land \boldsymbol{r}_1' = \boldsymbol{r}_2'\ .$$

Deadlock freedom states that every run in the protocol can be extended to a maximal run, meaning that it is either infinite or ends in a final state. Equivalently, we require that every reachable protocol state has an enabled outgoing transition, stated as follows.

Definition 8.3 (Deadlock freedom of symbolic protocol)**.** $\mathbb{S}$ is deadlock-free when for each non-final state $s \in S \setminus F$, the following is valid:

$$\forall \boldsymbol{r}.\ \mathrm{reach}(s, \boldsymbol{r}) \implies \bigvee_{(s,\, \mathsf{p} \to \mathsf{q} : x\{\varphi\},\, s') \in \Delta} \exists x.\ \varphi\ .$$

For determinism, Sprout generates one μCLP query per state; for deadlock freedom, Sprout generates one μCLP query per pair of transitions sharing a pre-state. If the input protocol is not GCLTS-eligible, Sprout reports specifically which assumption is violated by which state or transitions.

The GCLTS checking step of Sprout is sound and relatively complete with respect to the completeness of MuVal, and Sprout only checks implementability of GCLTS-eligible protocols.

8.3.2 Optimizations

Binary protocols. By Lemma 3.18, protocols involving only two participants represent a special case that are always implementable if they satisfy GCLTS assumptions. Sprout thus elides implementability checking for binary protocols, After checking GCLTS eligibility and before generating μCLP instances for checking implementability, Sprout checks whether the input protocol is binary, and if so, returns implementable immediately. This optimization enables Sprout to achieve performance within the same order of magnitude as existing tools on binary protocols, which represent a large subset of benchmarks in the multiparty protocol literature.

Decomposition of μCLP instances. The second and primary Sprout optimization decomposes the intractably large naive μCLP encoding of the Symbolic Coherence Conditions into smaller instances. We briefly revisit the conditions and explain their naive encoding before describing our decomposition.

The conditions universally quantify over participants in the protocol, and then universally quantify over pairs of simultaneously reachable protocol states from the perspective of a participant. Together, the conditions rely on three recursive predicates: $\mathsf{prodreach}_{\mathsf{p}}(s_1, \boldsymbol{r}_1, s_2, \boldsymbol{r}_2)$, which captures simultaneous reachability from a participant's local perspective, $\mathsf{unreach}^{\varepsilon}_{\mathsf{p},\mathsf{q}}(s_2, \boldsymbol{r}_2, x_1)$, which captures send transitions that are disabled from ε-reachable states, and $\mathsf{avail}_{\mathsf{p},\mathsf{q},\mathcal{B}}(x_1, s_2, \boldsymbol{r}_2)$, which captures messages that can be asynchronously reordered to be available in the present state. We recall Symbolic Send Coherence (Definition 8.4, Chapter 3) below.

Definition 8.4 (Symbolic Send Coherence). $\mathbb{S}$ satisfies Symbolic Send Coherence when for each participant p, 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:

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

A μCLP instance is a pair $(\phi, \mathcal{R})$ of a *query* ϕ, which is a first order formula over a background theory, and a *body* $\mathcal{R}$, which is a sequence of inductive predicates with least or greatest fixpoint semantics. Symbolic Send Coherence in negation form thus naturally corresponds to one μCLP instance per participant. Each instance's query existentially quantifies over control states and registers, and is a series of $|Q| * |Q|$ disjuncts that perform case analysis over pairs of control states, i.e. each disjunct is of the form

$$s_1 = q_1 \wedge s_2 = q_2 \wedge \mathsf{prodreach}_{\mathsf{p}}(s_1, \boldsymbol{r_1}, s_2, \boldsymbol{r_2}) \ \wedge \ \varphi_1 \wedge \mathsf{unreach}^{\varepsilon}_{\mathsf{p,q}}(s_2, \boldsymbol{r_2}, x_1)$$

where $q_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:x_1\{\varphi_1\}} q_2 \in \Delta$. Each instance's body comprises the inductive predicates prodreach and unreach, defined as least and greatest fixpoints respectively:

$$\mathsf{prodreach}_{\mathsf{p}}(s_1, \boldsymbol{r_1}, s_2, \boldsymbol{r_2}) =_{\mu} \ldots; \qquad\qquad \mathsf{unreach}^{\varepsilon}_{\mathsf{p,q}}(s_2, \boldsymbol{r_2}, x_1) =_{\nu} \ldots;$$

Naively encoding the three Symbolic Coherence Conditions results in only $3 * |\mathcal{P}|$ μCLP instances per protocol. Each instance, however, is orders of magnitude larger than the average benchmark in MuVal's benchmark suite[1], and the verification time for our running example using this naive approach exceeds 10 minutes. Thus, Sprout takes a different approach to structuring the Symbolic Coherence Conditions as μCLP instances. First, Sprout distributes each disjunct into a separate instance, yielding $|\mathcal{P}| * |Q| * |Q|$ instances for each condition. Next, Sprout decomposes the prodreach and unreach predicates by "currying" state arguments, generating one prodreach predicate per participant per pair of states, amounting to $|\mathcal{P}| * |Q| * |Q|$ predicate definitions, and one unreach predicate per pair of participants per state, amounting to $|\mathcal{P}| * |\mathcal{P}| * |Q|$ predicate

[1] https://github.com/hiroshi-unno/coar/tree/main/benchmarks/muCLP/popl2023mod

definitions. We show in Section 8.4 that decomposing large instances into multiple instances with smaller queries and more inductive predicates improves the running time of MuVal by over two orders of magnitude for most protocols.

Overapproximating simultaneous reachability. Thirdly, Sprout implements an overapproximation of simultaneous reachability that pre-filters pairs of control states before generating μCLP instances. Approximate simultaneous reachability disregards message values, only considering the sender and receiver of each event in a trace, e.g. $\mathsf{p} \triangleright \mathsf{q}!4 \cdot \mathsf{p} \triangleleft \mathsf{r}?7 \cdot \mathsf{s} \triangleright \mathsf{q}!5$ is abstracted to $\mathsf{p} \triangleright \mathsf{q}!\text{-} \cdot \mathsf{p} \triangleleft \mathsf{r}?\text{-} \cdot \mathsf{s} \triangleright \mathsf{q}!\text{-}$. This optimization preserves soundness and completeness of the tool: if two states are not approximately simultaneously reachable, then the Coherence Conditions say nothing about them; if two states are approximately simultaneously reachable, then the corresponding instances will be generated and checked, and in the case that they are not actually simultaneously reachable, will simply return invalid due to the prodreach conjunct being false.

Constraining simultaneously reachable control states. Finally, for Send Coherence instances concerning simultaneously reachable states that share a control state, we add a conjunct to the μCLP query requiring that not all register values in the two simultaneously reachable states are equal. This eliminates quantifier instantiations that simplify to the trivially false formula: $\mathsf{prodreach}_{\mathsf{p}}(s, \boldsymbol{r}, s, \boldsymbol{r}) \ \wedge \ \varphi \wedge \mathsf{unreach}^{\varepsilon}_{\mathsf{p},\mathsf{q}}(s, \boldsymbol{r}, x)$.

Bugs found in MuVal While implementing Sprout, we discovered a soundness bug in MuVal's `parallel` and `parallel_exc` modes that led its output to depend on the order of least and greatest fixpoint predicates in μCLP instances containing only one kind of fixpoint. We also discovered a minor bug in MuVal's constraint simplifier when optimizing queries containing negation or implication. Both bugs were reported to and subsequently fixed by MuVal's developers.[2]

[2] https://github.com/hiroshi-unno/coar/commit/bbe75fe7d5d4dcfc4b2eace94329a56bce9490e7, https://github.com/hiroshi-unno/coar/commit/1d49999975b00f1430b3c9d10b90ab00b561e836

Example	$\lvert\mathcal{P}\rvert$	$\lvert\Delta\rvert$	Sprout	time	Naive[Li et al. 2025b]	time
figure12-yes	3	2	impl.	0.6s	impl.	2.4s
figure12-no	3	2	non-impl.	0.6s	non-impl.	2.3s
TwoBuyer	3	9	impl.	3.2s	timeout (300s)	311.2s
higher-lower-ultimate	3	9	impl.	17.9s	out of memory	610.4s
higher-lower-no	3	9	non-impl.	24.5s	non-impl.	349.8s
symbolic-two-bidder-yes	3	10	impl.	17.0s	timeout (300s)	648.4s
symbolic-two-bidder-no	3	11	non-impl.	17.7s	out of memory	891.5s

Table 8.1: Comparison of verification times with and without optimizations.

8.4 Evaluation

All experiments in this section are run on a 2024 MacBook Air with an Apple M3 chip and 16GB of RAM. Verification times reported are the sum of GCLTS checking time and implementability checking time, with timeouts for individual μCLP instances specified separately.

8.4.1 Optimization efficacy

We first evaluate the efficacy of Sprout's optimizations, detailed in Section 8.3. We compare the verification times of Sprout's pre-filtered, optimized μCLP instances against the naive encoding of definitions in Section 3.4. We benchmark on examples of various sizes, measured by the number of transitions in the protocol specification. All examples are non-binary so as to reflect a difference in implementability checking time. The results in Table 8.1 show that naively encoding our conditions from Section 3.4 renders verification intractable for protocols with more than 2 transitions, and that Sprout's optimizations yield a speedup by over two orders of magnitude.

8.4.2 Evaluation and comparison against Session*

Next, we evaluate Sprout in terms of expressivity, precision and efficiency.

Expressivity. To evaluate expressivity, we took the union of two benchmark suites from tools most closely related to Sprout: Session* [Zhou et al. 2020] and Rumpsteak with refinements

[Vassor and Yoshida 2024]. Both works target multiparty protocols with refinements, however both differ from SPROUT in that they provide code generation functionality. SESSION*'s benchmark suite contains 11 examples, all of which utilize refinements. Despite the title of [Vassor and Yoshida 2024], Rumpsteak's suite of 10 examples contains only 5 with refinements, and 4 that are multiparty, for a total of 2/10 multiparty examples with refinements. We omitted finite, binary protocols that can be handled by existing sound and complete tools for finite multiparty session types, such as [Li et al. 2023a], leaving us with 6 examples from Rumpsteak. SPROUT was able to express all 17 examples from the literature. We then attempted to translate SESSION*'s examples into Rumpsteak's syntax, and vice versa, in an attempt to compare all three tools. Although both SESSION* and Rumpsteak adopt a SCRIBBLE-like syntax, we found that SESSION* could express all 6 of Rumpsteak's examples, whereas Rumpsteak could only express 3/11 of SESSION*'s examples, even after accommodating minor discrepancies that were immaterial to the high-level protocol intent. The key expressivity gap lay in the fact that SPROUT and SESSION* both support loop recursion variables, e.g. in the two-bidder protocol, z_1 and z_2 that track B_1 and B_2's respective last bids, whereas Rumpsteak does not.

PRECISION. The benchmark suites of both SESSION* and Rumpsteak exclusively contain implementable examples. In evaluating precision, we are interested in both the *soundness* and *completeness* of the tool: does it correctly accept implementable protocols, and correctly reject non-implementable ones? Thus, we expand our benchmark suite with a new set of examples based on protocols from prior works [Li et al. 2023a, 2025b; Cruz-Filipe et al. 2022], where for each protocol we include *both* an implementable and non-implementable version. We also introduce implementable and non-implementable variations on common protocols in the literature (e.g. two-bidder, higher lower guessing game). Some of the non-implementable examples were inspired by bugs inadvertently introduced in the process of translating examples into SPROUT, and most non-implementable examples have a small edit distance to their implementable counterpart. In translating our new examples to Session* and Rumpsteak, we found a similar pattern as before: Session* could express 20/21 examples, whereas Rumpsteak could only express 10/21. `Calculator` was not expressible in Session* due to lack of support for multiplication, whereas `higher-lower-no`'s implementability bug was ruled out by Session*'s type checker.

The result of evaluating Session* and Sprout on the overall set of 37 examples is given in Table 8.2. We omitted evaluation results from Rumpsteak due to the tool's lack of formal guarantees and limited expressivity. To achieve a faithful comparison, verification times reported for Session* are only for checking projectability of global types and computing local types for each role.

The incompleteness of Session* is made apparent by our evaluation: of the 20 new examples expressible in Session*, containing an even mix of implementable and non-implementable protocols, Session* rejected all but 3/20. The source of incompleteness is twofold. For one, Session*'s notion of implementability is relative to *local types*, whose syntax a priori rules out communication patterns such as receiver choice from different senders. In contrast, Sprout's notion of implementability is relative to the more expressive semantic model of communicating labeled transition systems. For two, Session* implements the merge-based projection operator from [Honda et al. 2008]. This projection operator is inherently incomplete even for global types without refinements (see [Li et al. 2023a] for a detailed discussion), and thus the refinement type system presented in [Zhou et al. 2020] inherits all sources of incompleteness.

Efficiency. In terms of efficiency, Session*'s verification times were mostly below 5s, whereas Sprout's verification times varied widely depending on the number of transitions in the protocol, and whether the protocol is binary. For binary protocols, the verification times of Sprout are competitive with those of Session*. For multiparty protocols, most examples returned in less than 10s, with the exception of 3 timeouts, whose timeout limits were set to 30s per μCLP instance.[3] As

[3]Note that when Sprout returns non-implementable for protocols containing instances that timeout, the verification time may increase directly with the timeout limit.

Source	Example	$\lvert\mathcal{P}\rvert$	Impl.	Sprout	Time	Session*	Time
[Zhou et al. 2020]	Calculator	2	✓	✓	0.6s	N/A	2.0s
	Fibonacci	2	✓	✓	0.5s	✓	1.8s
	HigherLower	3	✓	✓	15.2s	✓	3.9s
	HTTP	2	✓	✓	0.4s	✓	1.9s
	Negotiation	2	✓	✓	1.0s	✓	1.9s
	OnlineWallet	3	✓	✓	9.4s	✓	3.3s
	SH	3	✓	✓	237.1s	✓	5.6s
	Ticket	2	✓	✓	0.6s	✓	1.9s
	TravelAgency	2	✓	✓	9.2s	✓	3.1s
	TwoBuyer	3	✓	✓	3.8s	✓	2.8s
[Vassor and Yoshida 2024]	DoubleBuffering	3	✓	✓	1.5s	✓	2.3s
	OAuth	3	✓	✓	6.2s	✓	2.3s
	PlusMinus	3	✓	✓	5.2s	×	2.1s
	RingMax	7	✓	✓	3.7s	✓	4.7s
	SimpleAuth	2	✓	✓	0.5s	✓	2.0s
	TravelAgency2	2	✓	✓	1.7s	✓	1.8s
[Li et al. 2023a]	send-validity-yes	4	✓	✓	1.9s	×	2.1s
	send-validity-no	4	×	×	1.9s	×	2.1s
	receive-validity-yes	3	✓	✓	5.1s	×	2.3s
	receive-validity-no	3	×	×	3.6s	×	2.0s
[Li et al. 2025b]	symbolic-two-bidder-yes	3	✓	✓	27.4s	×	2.0s
	symbolic-two-bidder-no1	3	×	×	30.0s	×	2.1s
	figure12-yes	3	✓	✓	2.0s	✓	2.0s
	figure12-no	3	×	×	3.0s	✓	3.0s
	symbolic-send-validity-yes	4	✓	✓	6.5s	×	2.5s
	symbolic-send-validity-no	4	×	×	5.3s	×	2.6s
	symbolic-receive-validity-yes	3	✓	✓	6.6s	×	2.8s
	symbolic-receive-validity-no	3	×	×	7.6s	×	2.8s
[Cruz-Filipe et al. 2022]	fwd-auth-yes	3	✓	✓	10.3s	×	2.3s
	fwd-auth-no	3	×	?	T/O	×	2.2s
new	symbolic-two-bidder-no2	3	×	×	23.9s	×	2.8s
	higher-lower-ultimate	3	✓	✓	11.1s	×	2.4s
	higher-lower-winning	3	✓	?	T/O	✓	229.8s
	higher-lower-no	3	×	×	7.3s	N/A	2.2s
	higher-lower-encrypt-yes	4	✓	✓	9.3s	×	2.3s
	higher-lower-encrypt-no	4	×	×	177.3s	×	2.4s
	higher-lower-mixed	3	×	×	19.3s	×	2.3s

Table 8.2: Comparison of verification times with [Zhou et al. 2020]. For each example, we report the number of participants ($\lvert\mathcal{P}\rvert$), ground truth implementability (✓ or ×), verification times for Session* [Zhou et al. 2020] and Sprout with a 30s timeout per μCLP instance (T/O), and the result: ✓ for implementable/projectable, × for non-implementable/non-projectable, and ? for inconclusive due to timeout. Examples not expressible in Session* are marked with N/A.

mentioned in Section 8.3, the verification bottleneck of Sprout lies in the efficiency of MuVal– instance generation introduces negligible overhead. The modularity of our Coherence Conditions means Sprout's efficiency could be improved by running all generated μCLP instances in parallel.

A | Appendix

Lemma 3.27. *Implementability of global types is co-NP-complete.*

Proof. The arguments for co-NP membership of implementability for global types are identical to those for general finite protocols, and are thus omitted.

As in the proof of Theorem 5.8, we show NP-hardness of non-implementability via 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, with $1 \leq i \leq k$ and $1 \leq j \leq 3$. We construct a global type $\mathbf{G}_\varphi$ over participants $\mathcal{P} = \{\mathsf{p}, \mathsf{q}, \mathsf{r}, \mathsf{x}_1, \overline{\mathsf{x}}_1, \ldots, \mathsf{x}_\mathsf{n}, \overline{\mathsf{x}}_\mathsf{n}\}$, such that φ is satisfiable iff $\mathbf{G}_\varphi$ is implementable. In particular, we ensure that $\mathbf{G}_\varphi$ is implementable iff $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, G')$ does not hold for some subterm G' in $\mathbf{G}_\varphi$.

The construction idea for $\mathbf{G}_\varphi$ is identical to that for $\mathcal{S}_\varphi$ from Theorem 5.8, but with several modifications to yield a tree-shaped protocol which corresponds to a global type. First, for each branching state from which r selects variables or clauses, represented as μt terms, we introduce a new branch that acts as a forward edge connecting to the next branching state. Because branches in a global type can only join at a single state via recursion variables, and recursion variables must appear in scope of their μt terms, variable and clause selection proceeds by recursing "backwards" towards the top-level global type. Due to this reversal of traversal order, the initial choice by r and the message exchange $\mathsf{p} \rightarrow \mathsf{q} : m$ potentially violating Receive Coherence swap places in the protocol. The construction of global type $\mathbf{G}_\varphi$ is detailed below:

1. Define for every variable x_i with $2 < i < n$ a global type G_{x_i} representing a truth assignment

to variable x_i as follows:

$$G_{x_i} \coloneqq \mu t_{x_i}.\ + \begin{cases} \mathsf{r} \to \mathsf{x_i} : \bot.\, \mathsf{r} \to \overline{\mathsf{x}}_\mathsf{i} : \top.\, \mathsf{r} \to \mathsf{q} : m_{x_i}.\, \mathsf{q} \to \mathsf{x_i} : m.\, t_{x_{i+1}} \\ \mathsf{r} \to \overline{\mathsf{x}}_\mathsf{i} : \bot.\, \mathsf{r} \to \mathsf{x_i} : \top.\, \mathsf{r} \to \mathsf{q} : m_{\overline{x}_i}.\, \mathsf{q} \to \overline{\mathsf{x}}_\mathsf{i} : m.\, t_{x_{i+1}} \\ \mathsf{r} \to \mathsf{x_i} : \mathit{next}.\, \mathsf{r} \to \overline{\mathsf{x}}_\mathsf{i} : \mathit{next}.\, \mathsf{r} \to \mathsf{q} : \mathit{next}.\, G_{x_{i-1}} \end{cases}$$

For x_2 and x_n, the construction is modified as follows. For G_{x_n}, the recursion variable in the first and second branches is replaced with t_{C_1}. For G_{x_2}, the following is added before G_{x_1} in the third branch:

$$\mathsf{r} \to \mathsf{q} : \mathit{last}.\, \mathsf{r} \to \mathsf{p} : \mathit{last}.\, \mathsf{r} \to \mathsf{x_1} : \mathit{last}.\, \mathsf{r} \to \overline{\mathsf{x}}_1 : \mathit{last}.\, \mathsf{q} \to \mathsf{p} : m.\, \mathsf{q} \to \mathsf{x_1} : m.\, \mathsf{q} \to \overline{\mathsf{x}}_1 : m.$$

2. Define for every clause $C_i = L_{i1} \vee L_{i2} \vee L_{i3}$ with $2 \leq i < k$ a global type G_{C_i} as follows, where x_{ij} is defined as x if $L_{ij} = x$ and $\overline{\mathsf{x}}$ if $L_{ij} = \neg x$:

$$G_{C_i} \coloneqq \mu t_{C_i}.\ + \begin{cases} \Sigma_{j=1..3}\, \mathsf{r} \to x_{ij} : m.\, \mathsf{r} \to \mathsf{p} : m_{x_{ij}}.\, x_{ij} \to \mathsf{p} : m.\, t_{C_{i+1}} \\ \mathsf{r} \to x_{i1} : \mathit{next}.\, \mathsf{r} \to x_{i2} : \mathit{next}.\, \mathsf{r} \to x_{i3} : \mathit{next}.\, \mathsf{r} \to \mathsf{p} : \mathit{next}.\, G_{C_{i-1}} \end{cases}$$

For C_1 and C_k, the construction is modified as follows. For G_{C_1}, the last branch continues with G_{x_n}. For G_{C_k}, the recursion variable in the first three branches is replaced with t.

3. Define G_{x_1} for variable x_1 as follows:

$$G_{x_1} \coloneqq + \begin{cases} \mathsf{r} \to \mathsf{p} : m_1.\, \mathsf{r} \to \mathsf{q} : m.\, \overline{G} \\ \mathsf{r} \to \mathsf{p} : m_2.\, \mathsf{p} \to \mathsf{q} : m.\, 0 \end{cases}$$

$$\overline{G} \coloneqq + \begin{cases} \mathsf{r} \to \mathsf{x_1} : \bot.\, \mathsf{r} \to \overline{\mathsf{x}}_1 : \top.\, \mathsf{r} \to \mathsf{q} : m_{x_1}.\, \mathsf{q} \to \mathsf{x_1} : m.\, t_{x_2} \\ \mathsf{r} \to \overline{\mathsf{x}}_1 : \bot.\, \mathsf{r} \to \mathsf{x_1} : \top.\, \mathsf{r} \to \mathsf{q} : m_{\overline{x}_1}.\, \mathsf{q} \to \overline{\mathsf{x}}_1 : m.\, t_{x_2} \end{cases}$$

The global type $\mathbf{G}_\varphi$ is thus defined as:

$$\mathbf{G}_\varphi \coloneqq \mu t.\, \mathsf{r} \to \mathsf{q} : \mathit{top}.\, \mathsf{p} \to \mathsf{q} : m.\, G_{C_k}$$

Observe that $\mathbf{G}_\varphi$ is linear in the size of φ.

We first establish that $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, \overline{G})$ holds in $\mathbf{G}_\varphi$ iff φ is satisfiable. Observe that the G_{x_i}'s contain two branches that recurse "backwards" to the previous $G_{x_{i+1}}$, and one branch that proceeds "forwards" towards G_{x_1}. Each time a backward branch is taken, either $\mathtt{x_i}$ or $\overline{\mathtt{x_i}}$ is added to the blocked set $\mathcal{B}$ along the path. Forward branches do not change the blocked set, as participant q does not send messages in them. Thus, the path computed by $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, \overline{G})$ from $\overline{G}$ to G_{C_1} must contain for each variable x_i either $\mathtt{x_i}$ or $\overline{\mathtt{x_i}}$. The blocked set $\mathcal{B}$ thus encodes a truth assignment $\rho_\mathcal{B}$ for the x_i's where $\rho_\mathcal{B}(x_i) = \top$ iff $\mathtt{x_i} \notin \mathcal{B}$. By construction of G_{x_i}, for every truth assignment ρ, there exists at least one path between $\overline{G}$ and G_{C_1} such that $\rho = \rho_\mathcal{B}$ for the blocked set $\mathcal{B}$ computed along that path.

The G_{C_i} terms allow p to proceed backwards towards $\mathbf{G}_\varphi$ by selecting a branch whose participant x is not in $\mathcal{B}$, i.e. C_i is satisfied by $\rho_\mathcal{B}$. Thus, a path from G_{C_1} to $\mathbf{G}_\varphi$ adds p to $\mathcal{B}$ at t_i iff $\rho_\mathcal{B}$ does not satisfy at least one of the clauses C_i. Therefore, m is available in $\overline{G}$ iff there exists a $\mathcal{B}$ such that $\rho_\mathcal{B}$ satisfies φ.

The reasoning that $\mathbf{G}_\varphi$ is implementable iff $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, \overline{G})$ does not hold again follows that for $\mathcal{S}_\varphi$, and below we only discuss new behavior introduced by the structural changes to $\mathcal{S}_\varphi$.

Participant r still dictates the control flow in the global type, but now additionally sends *next* messages to inform participants in the branch when a forward edge is taken, *last* messages to inform p, q, $\mathsf{x_1}$ and $\overline{\mathsf{x_1}}$ when the last forward edge is taken, and *top* to q to inform q to receive m from q. Receiving *next* messages means inaction for all other participants. Receiving *last* prompts q to send a message to p, $\mathsf{x_1}$ and $\overline{\mathsf{x_1}}$, which they anticipate by receiving *last* first from r.

As before, the only potential source of non-implementability lies in participant q, who can violate Receive Coherence for transitions labeled with $\mathsf{r} \to \mathsf{q} : m$ and $\mathsf{p} \to \mathsf{q} : m$ in G_{x_1} when $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, \overline{G})$ does not hold, and the message from p can be received out of order.

We obtain that $\mathbf{G}_\varphi$ is non-implementable iff $\mathsf{avail}_{\mathsf{p},\mathsf{q},\{\mathsf{q}\}}(m, \overline{G})$ holds in $\mathbf{G}_\varphi$ iff φ is satisfiable. □

Lemma 3.7 (No Mixed Choice). *Let $\mathcal{S}$ be a protocol satisfying NMC (Definition 3.3) 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 3.3) with $s_2 \xrightarrow{l_2} s_2'$, s_1 and w to obtain a contradiction. □

Lemma A.1 (Channel compliance and intersection set non-emptiness implies prefix)**.** *Let* $\mathcal{S} = (S, \Gamma_{sync}, T, s_0, F)$ *be a protocol and let* $w \in \Sigma_{async}^*$ *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 [Majumdar et al. 2021a, 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 $w \in \text{pref}(\mathcal{L}(\mathcal{S}))$ 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 [Majumdar et al. 2021a, 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:

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

For each participant $\mathsf{p} \in \mathcal{P}$, let ρ_{p} be the largest prefix of ρ with $\mathtt{split}(\mathtt{trace}(\rho_{\mathsf{p}})){\Downarrow_{\Sigma_{\mathsf{p}}}} = w{\Downarrow_{\Sigma_{\mathsf{p}}}}$. 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 $u = u_1 u_2$. 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 $\mathsf{p} \in \mathcal{P}$, $wu_1{\Downarrow_{\Sigma_{\mathsf{p}}}} = \mathtt{split}(\mathtt{trace}(\rho_{\mathsf{s}})){\Downarrow_{\Sigma_{\mathsf{p}}}}$. 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 $v' = \mathtt{split}(\mathtt{trace}(\rho_{\mathsf{s}})) \leq w'$

and $u' \in \Sigma_{async}*$ 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 [Majumdar et al. 2021a, Lemma 23] to conclude that $vu' \sim v'$. □

Lemma 3.8 (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 $\mathrm{pref}(\mathcal{L}(\mathcal{S}))\Downarrow_{\Sigma_\mathsf{p}} \subseteq \mathrm{pref}(\mathcal{L}(T_\mathsf{p}))$ (**??**). 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 (Proposition A.2) and the induction hypothesis. This concludes our proof of prefix set inclusion. *End Proof of Claim 1.*

Case: $w \in \Sigma_{async}^*$. 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: $w \in \Sigma_{async}^{\omega}$. The infinite case when $w \in \Sigma_{async}^{\omega}$ is immediate from Claim 1. □

Lemma 3.9 (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 $w \in \Sigma_{async}^{*}$, 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, $w{\Downarrow_{\Sigma_{\mathsf{p}}}} \cdot \mathsf{p}\triangleright\mathsf{q}!m \in \text{pref}(\mathcal{L}(T_{\mathsf{p}}))$, and by the canonicity of T_{p}, $w{\Downarrow_{\Sigma_{\mathsf{p}}}} \cdot \mathsf{p} \triangleright \mathsf{q}!m \in \text{pref}(\mathcal{L}(\mathcal{S})){\Downarrow_{\Sigma_{\mathsf{p}}}}$. 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 $s_{fin} \overset{\mathsf{p}\to\mathsf{q}:m}{\underset{\mathsf{p}}{\Longrightarrow}}{}^{*} s_2'$. We find a contradiction to the assumption that final states in $\mathcal{S}$ do not have outgoing transitions.

Next, we show that for every $\rho \in I(w)$ and 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 Q such that for every $\mathsf{r} \in Q$, it holds that $w{\Downarrow_{\Sigma_{\mathsf{r}}}} < \left(\mathtt{split}(\mathtt{trace}(\rho))\right){\Downarrow_{\Sigma_{\mathsf{r}}}}$ (*). 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 [Majumdar et al. 2021a, 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:

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

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:

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

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 $V := \bigcup_{u \leq w} V_u$ and $E := \{(\rho_1, \rho_2) \mid \rho_1 \leq \rho_2\}$. 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,

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

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 $w' = \texttt{split}(\texttt{trace}(\rho'))$. It is clear that $w' \in \Sigma^\omega_{async}$ and $w' \in \texttt{split}(\mathcal{L}(\mathcal{S}))$. It remains to show that $w \preceq^\omega_\sim w'$. By the definition of $\preceq^\omega_\sim$, it further suffices to show that:

$$\forall u \leq w,\ \exists u' \leq w', v \in \Sigma^*_{async}.\ uv \sim u' \quad .$$

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

For each participant $\mathsf{p} \in \mathcal{P}$, let ρ'_p be the largest prefix of ρ' with $\texttt{split}(\texttt{trace}(\rho'_\mathsf{p})){\Downarrow_{\Sigma_\mathsf{p}}} = u{\Downarrow_{\Sigma_\mathsf{p}}}$. 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 $v{\Downarrow_{\Sigma_\mathsf{p}}} \leq \texttt{split}(\texttt{trace}(\rho')){\Downarrow_{\Sigma_\mathsf{p}}}$.

Let s be the participant with the maximum $|\rho'_\mathsf{s}|$ in $\mathcal{P}$. Let $u' = \texttt{split}(\texttt{trace}(\rho'_\mathsf{s}))$. Clearly, $u' \leq w'$. Because u' is $\texttt{split}(\texttt{trace}(\rho'_\mathsf{s}))$ 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

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

Let y_{p} be defined in this way for each participant. We construct $v \in \Sigma^*_{async}$ 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 [Majumdar et al. 2021a, Lemma 22]. □

Lemma 3.12 (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 implementation for $\mathcal{S}$. Then, for every trace $w \in \Sigma^*_{async}$ 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 $x \in \Sigma_{async}$.

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 7.1, 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 3.13, $I(wx) = I(w)$. $I(wx) \neq \emptyset$ follows trivially from the induction hypothesis and this equality. ☐

Proposition A.2 (CLTS traces are channel-compliant)**.** *Let $\{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ be a CLTS, and let $w \in \Sigma^*_{async}$ be a trace of $\{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}}$. Let $(\vec{s}, \xi)$ be the $\{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ configuration reached on w. Then, w is channel-compliant, and for every pair of participants $\mathsf{p} \neq \mathsf{q} \in \mathcal{P}$, $\mathcal{V}(w{\Downarrow}_{\mathsf{p}\triangleright\mathsf{q}!\_}) = \mathcal{V}(w{\Downarrow}_{\mathsf{q}\triangleleft\mathsf{p}?\_}) \cdot \xi(\mathsf{p},\mathsf{q})$.*

The proof of the same proposition for communicating state machines can be generalized directly to CLTSs, and thus we refer the reader to [Majumdar et al. 2021a, Lemma 19].

Lemma 3.13 (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 implementation 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 3.7, 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 Proposition A.2, $\mathcal{V}(w{\Downarrow}_{\mathsf{q}\triangleright\mathsf{p}!\_}) = \mathcal{V}(w{\Downarrow}_{\mathsf{p}\triangleleft\mathsf{q}?\_}) \cdot \xi(\mathsf{q},\mathsf{p})$. 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 Proposition A.2, 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 A.1, 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 $l_i := \mathsf{p}_i \to \mathsf{q}_i : m_i$. 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 $l_1 \ldots l_n$ must find and delete a matching pair of send and receive actions in v, thus preserving the above equality.

$\square$

Lemma 3.14 (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 implementation 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 3.1, 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 $\mathrm{R}^{\mathcal{S}}_{\mathsf{r}}(w) = \mathrm{R}^{\mathcal{S}}_{\mathsf{r}}(wx)$. Therefore, to prove the existential claim, it suffices to construct a run $\bar{\rho}$ that satisfies:

1. $\bar{\rho} \in \mathrm{R}^{\mathcal{S}}_{\mathsf{p}}(wx)$,
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

$\mathsf{q} \rightarrow \mathsf{p} : m$. Because $\alpha \cdot s_{pre} \xrightarrow{\mathsf{q}\rightarrow\mathsf{p}:m} s_{post} \cdot \beta$ is a maximal run in $\mathcal{S}$, we have that $(w \cdot \mathsf{p} \triangleleft \mathsf{q}?m){\Downarrow_{\Sigma_\mathsf{p}}} \in \operatorname{pref}(\mathcal{L}(\mathcal{S}){\Downarrow_{\Sigma_\mathsf{p}}})$. By the canonicity of $\{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}}$, it holds that $\operatorname{pref}(\mathcal{L}(\mathcal{S}){\Downarrow_{\Sigma_\mathsf{p}}}) \subseteq \operatorname{pref}(\mathcal{L}(T_\mathsf{p}))$, and therefore $(w \cdot \mathsf{p} \triangleleft \mathsf{q}?m){\Downarrow_{\Sigma_\mathsf{p}}} \in \operatorname{pref}(\mathcal{L}(T_\mathsf{p}))$. By assumption that wx is a trace of $\{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}}$, it holds that $(wx){\Downarrow_{\Sigma_\mathsf{p}}} \in \operatorname{pref}(\mathcal{L}(T_\mathsf{p}))$. 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 3.7. Therefore, l must be of the form $\mathsf{p} \rightarrow \mathsf{q}' : m'$, 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 $wx{\Downarrow_{\Sigma_\mathsf{p}}} \in \operatorname{pref}(\mathcal{L}(T_\mathsf{p}))$. By the canonicity of $\{\!\{T_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ (??(ii)), we have $\operatorname{pref}(\mathcal{L}(T_\mathsf{p})) \subseteq \operatorname{pref}((\mathcal{L}(\mathcal{S}){\Downarrow_{\Sigma_\mathsf{p}}}))$ and hence, $wx{\Downarrow_{\Sigma_\mathsf{p}}} \in \operatorname{pref}(\mathcal{L}(\mathcal{S}){\Downarrow_{\Sigma_\mathsf{p}}})$. 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 $wx{\Downarrow_{\Sigma_\mathsf{p}}} \leq \texttt{split}(\texttt{trace}(\rho')){\Downarrow_{\Sigma_\mathsf{p}}}$. The unique splitting of ρ' for p with respect to w gives us a transition $s_1 \xrightarrow{\mathsf{p}\rightarrow\mathsf{q}:m} s_2 \in T$.

If $s_1 = s_{pre}$, then $\alpha \cdot s_{pre} \xrightarrow{\mathsf{p}\rightarrow\mathsf{q}:m} s_2$ is a run in $\mathcal{S}$. Otherwise, we instantiate SC (Definition 3.1) with $s_1 \xrightarrow{\mathsf{p}\rightarrow\mathsf{q}:m} s_2$, s_{pre} and the witness $w{\Downarrow_{\Sigma_\mathsf{p}}}$. Then, there exists s' such that $s_{pre} \overset{\mathsf{p}\rightarrow\mathsf{q}:m}{\underset{\mathsf{p}}{\Longrightarrow}}{}^{*} s'$. We argue that, in fact, $s_{pre} \xrightarrow{\mathsf{p}\rightarrow\mathsf{q}:m} s' \in T$. 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, $\alpha \cdot s_{pre} \xrightarrow{\mathsf{p}\rightarrow\mathsf{q}:m} s'$ 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 $\alpha \cdot s_{pre} \xrightarrow{\mathsf{p}\rightarrow\mathsf{q}:m} \bar{s}'$ be a run in $\mathcal{S}$. Then, for all choices of $\bar{\beta}$ such that $\alpha \cdot s_{pre} \xrightarrow{\mathsf{p}\rightarrow\mathsf{q}:m} \bar{s}' \cdot \bar{\beta}$ is a maximal run, both $wx{\Downarrow_{\Sigma_\mathsf{p}}} \leq \texttt{split}(\texttt{trace}(\alpha \cdot s_{pre} \xrightarrow{\mathsf{p}\rightarrow\mathsf{q}:m} \bar{s}'))$ and $\alpha \cdot s_{pre} \leq \alpha \cdot s_{pre} \xrightarrow{\mathsf{p}\rightarrow\mathsf{q}:m} \bar{s}' \cdot \bar{\beta}$ 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 $\alpha \cdot s_{pre} \xrightarrow{\mathsf{p}\rightarrow\mathsf{q}:m} \bar{s}' \cdot \bar{\beta}$ 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 $\mathsf{r} \in \mathcal{Q}$,

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

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

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

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

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

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

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

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

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

It is clear that in order for both A.5 and A.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 $\mathcal{Q}$ with the *earliest disagreement* in $\mathtt{split}(\mathtt{trace}(\rho_c))$: let $\bar{\mathsf{r}}$ be the participant in $\mathcal{Q}$ 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 $\mathtt{trace}(\rho_c)[i_{\bar{\mathsf{r}}}]$, 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 $\mathtt{split}(\mathtt{trace}(\rho_c[i_{\bar{\mathsf{r}}}]))\Downarrow_{\Sigma_{\bar{\mathsf{r}}}}$.

Claim I $\quad y_{\bar{\mathsf{r}}}$ 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 $w'\Downarrow_{\Sigma_{\bar{\mathsf{r}}}} \leq \mathtt{split}(\mathtt{trace}(\rho_c))\Downarrow_{\Sigma_{\bar{\mathsf{r}}}}$. By definition, $w'\Downarrow_{\Sigma_{\bar{\mathsf{r}}}} = \mathtt{split}(\mathtt{trace}(\rho_c)[0..i_{\bar{\mathsf{r}}}-1])\Downarrow_{\Sigma_{\bar{\mathsf{r}}}}$. 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 (3.7) we have that $z \in \Sigma_{\bar{\mathsf{r}},?}$.

By assumption, $w'z \not\leq \mathtt{split}(\mathtt{trace}(\rho_c)[0..i_{\bar{\mathsf{r}}}])$. 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 3.13 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 $\quad \forall w'' \leq w'.\, I(w'')$ 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 3.13 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:

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

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 $\mathcal{Q} \in \mathcal{P}$ to the new set of participants that disagree with the new candidate run, and repeat the construction above on the new candidate run until $\mathcal{Q}$ 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 3.17 (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.

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

Specifically, we show the existence of a $v \in \Sigma_{async}^*$ such that

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

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 $v \in \Sigma_{async}^*$ 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.

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

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 $s_0 \overset{u}{\underset{\mathsf{p}}{\Rightarrow}}{}^* s_1, s$. Then, there does not exist $s' \in S$ such that $s \xRightarrow[\mathsf{p}]{\mathsf{p}\to\mathsf{q}:m}{}^* s'$.

Because $s_0 \overset{u}{\underset{\mathsf{p}}{\Rightarrow}}{}^* s$, 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 $s_0 \overset{u}{\underset{\mathsf{p}}{\Rightarrow}}{}^* s_1$, 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 p' be the sender in the first divergence between l and l'. Because $\rho' \in \mathrm{R}^{\mathcal{S}}_{\mathsf{p}'}(\bar{w})$, it holds that $\bar{w}{\Downarrow_{\Sigma_{p'}}} \leq \texttt{split}(\texttt{trace}(\rho')){\Downarrow_{\Sigma_{p'}}}$. We can rewrite the inequality as $\texttt{split}(l){\Downarrow_{\Sigma_{p'}}} \leq \texttt{split}(l'){\Downarrow_{\Sigma_{p'}}}$.

Because $\bar{l}$ is the largest common prefix shared by l and l', $\texttt{split}(l){\Downarrow_{\Sigma_{p'}}}$ and $\texttt{split}(l'){\Downarrow_{\Sigma_{p'}}}$ are respectively of the form $\bar{l}{\Downarrow_{\Sigma_{p'}}} \cdot \mathsf{p}' \triangleright \mathsf{q}_i!m'_i \cdot z'$ and $\bar{l}{\Downarrow_{\Sigma_{p'}}} \cdot \mathsf{p}' \triangleright \mathsf{q}_j!m'_j \cdot y'$, with $\mathsf{q}_i \neq \mathsf{q}_j$ or $m'_i \neq m'_j$. From this and $\bar{l}{\Downarrow_{\Sigma_{p'}}} \cdot \mathsf{p}' \triangleright \mathsf{q}_i!m'_i \cdot z' \leq \bar{l}{\Downarrow_{\Sigma_{p'}}} \cdot \mathsf{p}' \triangleright \mathsf{q}_j!m'_j \cdot y'$, 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 $s \overset{\mathsf{p}\to\mathsf{q}:m}{\underset{\mathsf{p}}{\Longrightarrow}}{}^* s'$, and thus there does not exist a maximal run $\bar{\rho} \in \mathcal{S}$ such that $v_0{\Downarrow_{\Sigma_p}} \leq \texttt{split}(\texttt{trace}(\bar{\rho})){\Downarrow_{\Sigma_p}}$.

Therefore, $\mathrm{R}^{\mathcal{S}}_{\mathsf{p}}(\bar{w} \cdot \mathsf{p} \triangleright \mathsf{q}!m) = \emptyset$, and $I(\bar{w} \cdot \mathsf{p} \triangleright \mathsf{q}!m) = \emptyset$ 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 $s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:m} s_2, s \xrightarrow{\mathsf{r}\to\mathsf{q}:m} s' \in T$. Then, $s \neq s_1$, $\mathsf{r} \neq \mathsf{p}$ and let $u \in \Sigma^*_{\mathsf{q}}$ be a word such that $s_0 \overset{u}{\underset{\mathsf{q}}{\Rightarrow}}{}^* s_1, s$. Furthermore there exists $w \in \mathrm{pref}(\mathcal{L}(\mathcal{S}_{s'}))$ with $w{\Downarrow_{\Sigma_q}} = \varepsilon \wedge \mathcal{V}(w{\Downarrow_{\mathsf{p}\triangleright\mathsf{q}!\_}}) = \mathcal{V}(w{\Downarrow_{\mathsf{q}\triangleleft\mathsf{p}?\_}}) \cdot m$.

Because $s_0 \overset{u}{\underset{\mathsf{q}}{\Rightarrow}}{}^* s_1, s$ and $s \xrightarrow{\mathsf{r}\to\mathsf{q}:m} s'$, there exists a run $\alpha \cdot s \xrightarrow{\mathsf{r}\to\mathsf{q}:m} s'$ such that $\texttt{split}(\texttt{trace}(\alpha \cdot s)){\Downarrow_{\Sigma_q}} = u$.

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

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

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

Next, we reason about the extension $\mathsf{r} \triangleright \mathsf{q}!m \cdot w$ together. We first establish that $\mathsf{r} \triangleright \mathsf{q}!m \cdot w \in$

$\text{pref}(\mathcal{L}(\mathcal{S}_s))$. Because $w \in \text{pref}(\mathcal{L}(\mathcal{S}_{s'}))$, there exists a maximal run $s' \cdot \beta$ such that $s' \cdot \beta \in I(w)$. Observe that $s \xrightarrow{\mathsf{r}\to\mathsf{q}:m} s' \cdot \beta \in I(\mathsf{r} \triangleright \mathsf{q}!m \cdot w)$ and that $\mathsf{r} \triangleright \mathsf{q}!m \cdot w$ remains channel-compliant due to the assumption that $w{\Downarrow_{\Sigma_\mathsf{q}}} = \varepsilon$. Thus, by Lemma A.1 it holds that $\mathsf{r} \triangleright \mathsf{q}!m \cdot w \in \text{pref}(\mathcal{L}(\mathcal{S}_s))$. Therefore, $\texttt{split}(\texttt{trace}(\alpha \cdot s)) \cdot \mathsf{r} \triangleright \mathsf{q}!m \cdot w \in \text{pref}(\mathcal{L}(\mathcal{S}))$, and by the assumption that $\{\!\{B_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ implements $\mathcal{S}$, $\texttt{split}(\texttt{trace}(\alpha \cdot s)) \cdot \mathsf{r} \triangleright \mathsf{q}!m \cdot w$ is a trace of $\{\!\{B_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}}$:

$$(\vec{s}_0, \xi_0) \xrightarrow{\texttt{split}(\texttt{trace}(\alpha\cdot s))}{}^{*} (\vec{s}, \xi) \xrightarrow{\mathsf{r}\triangleright\mathsf{q}!m\cdot w} (\vec{s}\,', \xi')$$

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

(1) there exists an outgoing transition labeled with $\mathsf{q} \triangleleft \mathsf{p}?m$ from $\vec{s}\,'_\mathsf{q}$, and

(2) $\xi'(\mathsf{p}, \mathsf{q}) = m \cdot u'$, with $u' \in \mathcal{V}^*$.

We know that $s_0 \overset{u}{\underset{\mathsf{q}}{\Rightarrow}}{}^{*} s_1$ and $s_1 \xrightarrow{\mathsf{p}\to\mathsf{q}:m} s_2$, so there exists a run $\alpha_1{\cdot}s_2$ such that $\texttt{split}(\texttt{trace}(\alpha_1 \cdot s_2)){\Downarrow_{\Sigma_\mathsf{q}}} = u{\cdot}\mathsf{q}{\triangleleft}\mathsf{p}?m$. Because $\texttt{split}(\texttt{trace}(\alpha_1{\cdot}s_2)){\Downarrow_{\Sigma_\mathsf{q}}} \in \text{pref}(\mathcal{L}(\mathcal{S})){\Downarrow_{\Sigma_\mathsf{q}}}$ and $\{\!\{B_\mathsf{p}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ implements $\mathcal{S}$, it follows that $u \cdot \mathsf{q} \triangleleft \mathsf{p}?m \in \text{pref}(\mathcal{L}(B_\mathsf{q}))$. Let $t \in Q_\mathsf{q}$ be the state reached on u in B_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 $(\texttt{split}(\texttt{trace}(\alpha \cdot s)) \cdot \mathsf{r} \triangleright \mathsf{q}!m \cdot w){\Downarrow_{\Sigma_\mathsf{q}}} = u$, 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 $\texttt{split}(\texttt{trace}(\alpha \cdot s))$, 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 $\texttt{split}(\texttt{trace}(\alpha \cdot s)) \cdot \mathsf{r} \triangleright \mathsf{q}!m \cdot w$, and thus m is at the head of channel $\xi'(\mathsf{p}, \mathsf{q})$:

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

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 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 $s_0 \overset{u}{\underset{\mathsf{q}}{\Rightarrow}}{}^* s_1, s$.

Because $s_0 \overset{u}{\underset{\mathsf{p}}{\Rightarrow}}{}^* s$, 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. □

Lemma 6.9. *Let $A_\mathsf{p} = (Q_\mathsf{p}, \Sigma_\mathsf{p}, \delta_\mathsf{p}, s_{0,\mathsf{p}}, F_\mathsf{p})$ denote the state machine for* p *in $\mathcal{A}$. Then,* Transition Exhaustivity *and* Final State Validity *imply $\mathcal{L}(\mathbf{G}){\Downarrow}_{\Sigma_\mathsf{p}} \subseteq \mathcal{L}(A_\mathsf{p})$.*

Proof. First, we show that every trace in $\mathcal{L}(\mathbf{G}){\Downarrow}_{\Sigma_\mathsf{p}}$ is a trace in A_p. Let u be a trace in $\mathcal{L}(\mathbf{G}){\Downarrow}_{\Sigma_\mathsf{p}}$. We proceed by induction on the length of u. In the base case, $u = \varepsilon$, and ε is trivially a trace of every state machine. In the induction step, let ux be a prefix in $\mathcal{L}(\mathbf{G}){\Downarrow}_{\Sigma_\mathsf{p}}$. From the induction hypothesis, we know that u is a prefix in $\mathcal{L}(A_\mathsf{p})$. Let $s \in Q_\mathsf{p}$ be the state reached on u in A_p. Because ux is a prefix in $\mathcal{L}(\mathbf{G}){\Downarrow}_{\Sigma_\mathsf{p}}$, there exists a run $q_{0,\mathrm{G}} \xrightarrow{u}{}^* G \xrightarrow{x}{}^* G'$ in the projection by erasure automaton for p. By the definition of state decoration, it holds that $G \in d_\mathrm{G}(s)$. By *Transition Exhaustivity*, it holds that there exists a state $s' \in Q_\mathsf{p}$ such that $s \xrightarrow{x} s' \in \delta_\mathsf{p}$, and therefore ux is also a prefix in $\mathcal{L}(A_\mathsf{p})$. This concludes our proof by induction that every prefix in $\mathcal{L}(\mathbf{G}){\Downarrow}_{\Sigma_\mathsf{p}}$ is a prefix in $\mathcal{L}(A_\mathsf{p})$.

Let $w \in \mathcal{L}(\mathbf{G}){\Downarrow}_{\Sigma_\mathsf{p}}$. To show that $w \in \mathcal{L}(A_\mathsf{p})$ for $w \in \Sigma^*$, it remains to show that w reaches a final state in A_p. Let $G'' \in F_\mathrm{G}$ be the state reached on w in the projection by erasure automaton, and let s'' be the state reached on w in A_p. By the state decoration function it holds that $G'' \in d_\mathrm{G}(s'')$, and therefore by *Final State Validity*, $s'' \in F_\mathsf{p}$ and w is a word in $\mathcal{L}(A_\mathsf{p})$. The case for

$w \in \Sigma^\infty$ follows from the fact that every trace of $\mathcal{L}(\mathbf{G})\Downarrow_{\Sigma_p}$ is a trace of $\mathcal{L}(A_p)$ and the fact that A_p is deterministic. □

Lemma 6.13. *Let $\mathcal{A}$ be a CSM, q be a role, and w, wx be traces of $\mathcal{A}$ such that $x = q \triangleleft r?m$. Let s be the state of q's state machine in the $\mathcal{A}$ configuration reached on w. Let ρ be a run that is consistent with w, i.e. for all $p \in \mathcal{P}$. $w\Downarrow_{\Sigma_p} \leq \texttt{split}(\texttt{trace}(\rho))\Downarrow_{\Sigma_p}$. Let $\alpha \cdot G \xrightarrow{l} G' \cdot \beta$ be the unique splitting of ρ for q matching w. If $r \triangleright q!m \notin M^{q}_{(G' \ldots)}$, then $x = \texttt{split}(l)\Downarrow_{\Sigma_q}$.*

Proof. Suppose by contradiction that $x \neq \texttt{split}(l)\Downarrow_{\Sigma_q}$. By the definition of unique splittings, q is the active role in l. We proceed by case analysis on l: (1) either l is of the form $r \to q : m'$, with r sending q a different message $m' \neq m$, or (2) l is of the form $s \to q : m$, with a different role $s \neq r$ sending q a message, or l is of the form $q \xrightarrow{-}: \_$, with q sending a message. We prove a contradiction in each case.

First, we establish a claim that is used in both cases, and relies only on the fact that ρ is consistent with w and wx is a trace of $\mathcal{A}$.

Let ρ_q denote the largest consistent prefix of ρ for q; it is clear that $\rho_q = \alpha \cdot G$. Formally,

$$\rho_q = max\{\rho' \mid \rho' \leq \rho \ \land \ \left(\texttt{split}(\texttt{trace}(\rho'))\right)\Downarrow_{\Sigma_q} \leq w\Downarrow_{\Sigma_q}\} \ .$$

Let ρ_r be defined analogously.

Claim: $\rho_q < \rho_r$. Intuitively, p is ahead of q in ρ due to the half-duplex property of CSMs and the fact that r is the sender. Formally, [Majumdar et al. 2021a, Lemma 19] implies $\xi(r, q) = u$ where $\mathcal{V}(w\Downarrow_{r \triangleright q!\_}) = \mathcal{V}(w\Downarrow_{q \triangleleft r?\_}).u$. Because $\xi(r, q)$ contains at least m by assumption, $|\mathcal{V}(w\Downarrow_{r \triangleright q!\_})| > |\mathcal{V}(w\Downarrow_{q \triangleleft r?\_})|$. Because $\mathcal{V}(w\Downarrow_{q \triangleleft r?\_}) < \mathcal{V}(w\Downarrow_{r \triangleright q!\_})$ and traces of CSMs are channel-compliant [Majumdar et al. 2021a, Lemma 19], it holds that ρ_r contains all $|\mathcal{V}(w\Downarrow_{q \triangleleft r?\_})|$ transition labels of the form $r \to q : \_$ that are contained in ρ_r, plus at least one more of the form $r \to q : m$. Because both ρ_q and ρ_r are prefixes of ρ, it must be the case that $\rho_q < \rho_r$. This concludes the proof of the above claim.

Case: $l = \mathsf{r} \to \mathsf{q} : m'$ and $m' \neq m$. We discharge this case by showing a contradiction to the fact that m is at the head of the channel between r and q.

Because $\alpha \cdot G \leq \rho_{\mathsf{q}}$ and $\rho_{\mathsf{q}} < \rho_{\mathsf{r}}$ from the claim above, it must be the case that $\alpha \cdot G \xrightarrow{l} G' \leq \rho_{\mathsf{r}}$ and $\mathsf{r} \triangleright \mathsf{q}!m'$ is in $w\Downarrow_{\Sigma_{\mathsf{r}}}$. From [Majumdar et al. 2021a, Lemma 19], it follows that $\mathcal{V}(w\Downarrow_{\mathsf{r}\triangleright\mathsf{q}!\_}) = \mathcal{V}(w\Downarrow_{\mathsf{q}\triangleleft\mathsf{r}?\_}).m'.u'$ and $\xi(\mathsf{r},\mathsf{q}) = m'.u'$, i.e. m' is at the head of the channel between r and q. We reach a contradiction.

Case: $\forall m'.\ l \neq \mathsf{r} \to \mathsf{q} : m'$. It follows that $\mathtt{split}(l)\Downarrow_{\Sigma_{\mathsf{q}}} \neq \mathsf{q} \triangleleft \mathsf{r}?m'$ for any m'. We discharge this case by showing that

$$\mathsf{r} \triangleright \mathsf{q}!m \in M^{\mathsf{q}}_{(G' \ldots)} \quad .$$

Recall that $\alpha \cdot G \xrightarrow{l} G' \leq \rho_{\mathsf{r}}$. Then, there exists a transition labeled $\mathsf{r} \to \mathsf{q} : m$ that occurs in the suffix $G' \cdot \beta$. Let $G_0 \xrightarrow{\mathsf{r}\to\mathsf{q}:m} G'_0$ be the earliest occurrence of such a transition in the suffix, then:

$$\rho_{\mathsf{r}} = \alpha \cdot G \xrightarrow{l} G' \ldots G_0 \xrightarrow{\mathsf{r}\to\mathsf{q}:m} G'_0 \ldots \quad .$$

Note that G_0 must be a syntactic subterm of G'. In order for $\mathsf{r} \triangleright \mathsf{q}!m \in M^{\mathsf{q}}_{(G' \ldots)}$ to hold, it suffices to show that $\mathsf{r} \notin \mathcal{B}$ in the recursive call to $M^{\mathcal{B}}_{(G' \ldots)}$. We argue this from the definition of M and the fact that $\rho_{\mathsf{q}} = \alpha \cdot G$. Suppose for the sake of contradiction that $\mathsf{r} \in \mathcal{B}$. Because M only adds receivers of already blocked senders to $\mathcal{B}$ and $M^{\mathsf{q}}_{(G' \ldots)}$ starts with $\mathcal{B} = \{\mathsf{q}\}$, there must exist a chain of message exchanges $\mathsf{s}_{i+1} \to \mathsf{s}_i : m_i$ in G' with $1 \leq i < n$, $\mathsf{q} = \mathsf{s}_n$, and $\mathsf{r} = \mathsf{s}_1$. That is, $G' \cdot \beta$ must be of the form

$$G' \ldots G_{n-1} \xrightarrow{\mathsf{q}\to\mathsf{s}_{n-1}:m_{n-1}} G'_{n-1} \ldots G_1 \xrightarrow{\mathsf{s}_2\to\mathsf{r}:m_1} G'_1 \ldots G_0 \xrightarrow{\mathsf{r}\to\mathsf{q}:m} G'_0 \ldots \quad .$$

Let $m_0 = m$ and $\mathsf{s}_0 = \mathsf{q}$. We show by induction over i that for all $i \in [1, n]$

$$\alpha \cdot G \xrightarrow{l} G' \ldots G_i \xrightarrow{\mathsf{s}_i\to\mathsf{s}_{i-1}:m_{i-1}} G'_i \ \leq\ \rho_{\mathsf{s}_i} \quad .$$

We then obtain the desired contradiction with the fact that $\rho_{\mathsf{s}_n} = \rho_{\mathsf{q}} = \alpha \cdot G'$. The base case of the induction follows immediately from the construction. For the induction step, assume that

$$\alpha \cdot G \xrightarrow{l} G' \ldots G_i \xrightarrow{\mathsf{s}_i \to \mathsf{s}_{i-1}: m_{i-1}} G'_i \ \leq \ \rho_{\mathsf{s}_i} \quad .$$

From the definition of ρ_{s_i} and the fact that s_i is the active role in $\mathsf{s}_i \triangleleft \mathsf{s}_{i+1}?m_i$, it follows that $\mathsf{s}_i \triangleleft \mathsf{s}_{i+1}?m_i \in w$. Hence, we must also have $\mathsf{s}_{i+1} \triangleright \mathsf{s}_i!m_i \in w$. Since s_{i+1} is the active role in $\mathsf{s}_{i+1} \triangleright \mathsf{s}_i!m_i$, we can conclude

$$\alpha \cdot G \xrightarrow{l} G' \ldots G_i \xrightarrow{\mathsf{s}_{i+1} \to \mathsf{s}_i: m_i} G'_{i+1} \ \leq \ \rho_{\mathsf{s}_{i+1}} \quad .$$

This concludes the proof of Lemma 6.13. □

Lemma 6.15. *If $\mathcal{A}$ violates* Transition Exhaustivity *or* Final State Validity*, then it does not hold that $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$ refines $\mathcal{A}$.*

Proof. From the negation of *Transition Exhaustivity*, we find a witness trace v such that v is a trace in $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$ but not a trace in $\mathcal{A}$, thus contradicting the fact that $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$ refines $\mathcal{A}$. Let p be a role that violates *Transition Exhaustivity*. Let s be a state such that there exists $G \in d_{\mathbf{G}}(s)$ with $G \xrightarrow{x}^* G' \in \delta_{\downarrow}$ but no transition outgoing from s labeled with x. By the definition of state decoration, there exists $u \in \Sigma_{\mathsf{p}}^*$ such that A_{p} reaches s on u from its initial state, and the projection by erasure automaton for p reaches G on u from its initial state. Because $G \xrightarrow{x}^* G' \in \delta_{\downarrow}$, it holds that $q_{0,\mathbf{G}} \xrightarrow{u}^* G \xrightarrow{x}^* G' \in \delta_{\downarrow}$ is a run in the projection by erasure automaton for p. Let ρ denote this run, and let $w = \texttt{split}(\texttt{trace}(\rho))$. Then, it holds that $ux \leq w\Downarrow_{\Sigma_{\mathsf{p}}}$. Because $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$ implements $\mathbf{G}$, w is a trace of $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$. Consequently, $w\Downarrow_{\Sigma_{\mathsf{p}}}$ is a prefix of A_{p}. Because ux is a prefix of $w\Downarrow_{\Sigma_{\mathsf{p}}}$, ux is thus also a prefix of A_{p}. Because A_{p} is deterministic, A_{p} reaches s on u. However, there does not exist an outgoing transition labeled with x from s, and we reach a contradiction to the fact that ux is a prefix of A_{p}.

From the negation of *Final State Validity*, we find a witness trace v that is maximally terminated in $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p}\in\mathcal{P}}$, but not maximally terminated in $\mathcal{A}$, thus contradicting the fact that $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p}\in\mathcal{P}}$ refines $\mathcal{A}$. Let p be a role that violates *Final State Validity*. Let s be a state such that there exists $G \in d_{\mathbf{G}}(s)$ with $G \in F_{\mathbf{G}}$ but $s \notin F_{\mathsf{p}}$. Let $w \in \mathcal{L}(\mathbf{G})$ such that $w\Downarrow_{\Sigma_{\mathsf{p}}}$ reaches G in the projection by erasure automaton on $w\Downarrow_{\Sigma_{\mathsf{p}}}$; such a word is guaranteed to exist. Because $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p}\in\mathcal{P}}$ refines $\mathcal{A}$, $w \in \mathcal{L}(\mathcal{A})$. Because A_{p} is deterministic, A_{p} reaches s on $w\Downarrow_{\Sigma_{\mathsf{p}}}$. In other words, in the $\mathcal{A}$ configuration reached on w, A_{p} is in state s. However, $s \notin F_{\mathsf{p}}$. Therefore, w is not terminated in $\mathcal{A}$ and $w \notin \mathcal{L}(\mathcal{A})$. We reach a contradiction. □

Lemma 6.16. *If $\mathcal{A}$ violates* Send Decoration Validity *or* Receive Decoration Validity, *then it does not hold that $\mathcal{A}$ and $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p}\in\mathcal{P}}$ are equivalent.*

Proof. Because $\mathbf{G}$ is implementable, $\mathscr{P}(\mathbf{G}, \mathsf{p})$ satisfies Send Validity and Receive Validity [Li et al. 2023a, Theorem 7.1]. For each condition, we assume the violation of the condition and the fact that $\mathcal{A}$ and $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p}\in\mathcal{P}}$ are equivalent, and show a contradiction to Send Validity and Receive Validity in turn.

Let p be a role that violates *Send Decoration Validity*. Let s be a state and $s \xrightarrow{\mathsf{p}\triangleright\mathsf{q}!m} s'$ be a transition in A_{p} such that

$$\text{tr-orig}(d(s) \xrightarrow{\mathsf{p}\triangleright\mathsf{q}!m} d(s')) \neq d(s) \ \ .$$

Let G be a state in $d(s) \setminus \text{tr-orig}(d(s) \xrightarrow{\mathsf{p}\triangleright\mathsf{q}!m} d(s'))$. Such a G exists by the negation of *Send Decoration Validity*. Let $\alpha \cdot G$ be a run in $\text{GAut}(\mathbf{G})$; such a run must exist by the fact that G is a syntactic subterm of $\mathbf{G}$. Let $w = \mathtt{split}(\mathtt{trace}(\alpha \cdot G))$. Because $w \in \text{pref}(\mathcal{L}(\mathbf{G}))$, it holds that w is a trace of $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p}\in\mathcal{P}}$. Because $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p}\in\mathcal{P}}$ refines $\mathcal{A}$ by assumption, w is a trace in $\mathcal{A}$, and there exists an $\mathcal{A}$ configuration reached on w in which A_{p} is in state s. Because send actions are always enabled, wx is a trace in $\mathcal{A}$. Now because $\mathcal{A}$ refines $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p}\in\mathcal{P}}$, wx is also a trace in $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p}\in\mathcal{P}}$. By definition, let t be the state of p in the $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p}\in\mathcal{P}}$ configuration reached on w. Because $w = \mathtt{split}(\mathtt{trace}(\alpha \cdot G))$, it holds that $w\Downarrow_{\Sigma_{\mathsf{p}}} \in \text{pref}(\mathcal{L}(\mathscr{P}(\mathbf{G}, \mathsf{p})))$, and by Definition 5.2, it holds that $G \in t$. Then, there exists a t' such that $t \xrightarrow{x} t'$ is a transition in $\mathscr{P}(\mathbf{G}, \mathsf{p})$. We find a contradiction to Send Validity for this transition by using G as a witness.

Let p be a role that violates *Receive Decoration Validity*. Let s be a state and let $s \xrightarrow{\mathsf{p} \triangleleft \mathsf{q}_1 ? m_1} s_1$, $s \xrightarrow{x} s_2$ be two transitions in A_{p}, with $G_2 \in \text{tr-dest}(d(s) \xrightarrow{x} d(s_2))$ such that

$$x \neq \mathsf{p} \triangleleft \mathsf{q}_1 ?\_ \quad \wedge \quad \mathsf{q}_1 \triangleright \mathsf{p} ! m_1 \in M^{\mathsf{p}}_{(G_2 \ldots)} \ .$$

Following the construction in[Li et al. 2023a, Theorem 7.1], we can construct a witness trace w in $\mathcal{A}$ such that both $w \cdot \mathsf{p} \triangleleft \mathsf{q}_1 ? m_1$ and $w \cdot x$ are traces in $\mathcal{A}$. Because $\mathcal{A}$ refines $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$ by assumption, both $w \cdot \mathsf{p} \triangleleft \mathsf{q}_1 ? m_1$ and $w \cdot x$ are also traces in $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$. Let t be the state reached by $\{\!\{\mathscr{P}(\mathbf{G}, \mathsf{p})\}\!\}_{\mathsf{p} \in \mathcal{P}}$ on w. Then, there must exist two transitions $t \xrightarrow{\mathsf{p} \triangleleft \mathsf{q}_1 ? m_1} t'$ and $t \xrightarrow{x} t''$ in $\mathscr{P}(\mathbf{G}, \mathsf{p})$. Either $x \in \Sigma_{\mathsf{p},!}$ and No Mixed Choice [Li et al. 2023a, Corollary 5.5] is violated, or $x \in \Sigma_{\mathsf{p},?}$ and Receive Validity is violated. □

Lemma 6.18. *The* Monolithic Protocol Refinement *problem is PSPACE-hard.*

Proof. We show the PSPACE-hardness of the monolithic refinement problem by a reduction from the PSPACE-hard problem of deciding deadlock freedom for 1-safe Petri nets [Esparza and Nielsen 1994]. Let (N, M_0) be a 1-safe Petri net, with $N = (S, T, F)$.

We construct a CSM $\mathcal{A}_N$ and a global type $\mathbf{G}_N$ such that $\mathcal{A}_N$ refines $\mathbf{G}_N$ if and only if the Petri net is deadlock-free.

We first describe the construction of $\mathcal{A}_N$. $\mathcal{A}_N$ consists of one state machine per place in S, one state machine per transition in T, and one special coordinator role, which we denote p. Each place state machine tracks whether its place is marked by 0 or 1, and responds to messages to increment or decrement its marking. Each transition state machine communicates with its input and output place state machines to check whether its transition is enabled, and to update place markings. The coordinator p first asks each transition state machine whether its transition is enabled. This querying can be performed in an arbitrary fixed order on T. If at least one transition is enabled, p then non-deterministically picks a transition to fire. Depending on whether the picked transition is enabled, the input and output place state machines update the configuration, and the transition state machine returns the control flow to p, which repeats this process with the new configuration. If no transition is enabled, p enters a sink state with no outgoing transitions, thus causing a deadlock in $\mathcal{A}_N$.

Each message exchange between roles is echoed with an acknowledgement, and the CSM thus constructed is 1-bounded: there is at most one message in flight at any point during its execution. Intuitively, $\mathcal{A}_N$ simulates the firing of transitions in the Petri nets via message exchanges, and represents all valid execution traces of the Petri net as CSM traces.

Correspondingly, we construct a global type $\mathbf{G}_N$ whose language includes not only all execution traces of $\mathcal{A}_N$, but also traces that do not correspond to valid execution traces in the Petri net. $\mathbf{G}_N$ achieves this by mimicing the control flow of the $\mathcal{A}_N$, but decoupling the message contents from the underlying Petri net configuration: at each control flow point, roles non-deterministically choose a message to send.

If the Petri net is deadlock-free, then $\mathcal{A}_N$ is also deadlock-free and $\mathcal{L}(\mathcal{A}_N)$ includes only infinite words: because each configuration has at least one enabled transition, p's sink state will never be reached. Because $\mathcal{L}(\mathcal{A}_N) \subseteq \mathcal{L}(\mathbf{G}_N)$ by construction, it holds that $\mathcal{A}_N$ refines $\mathbf{G}_N$. On the contrary, if $\mathcal{A}_N$ refines $\mathbf{G}_N$ and is thus deadlock-free, then the Petri net is also deadlock-free, as $\mathcal{A}_N$ can simulate all valid execution traces of the Petri net.

□

Lemma 6.30 (Soundness of C_2). *If C_2 holds, then for all well-behaved contexts $\mathcal{A}[\cdot]_{\mathsf{p}}$, $\mathcal{A}[A]_{\mathsf{p}}$ refines $\mathcal{A}[B]_{\mathsf{p}}$.*

Proof. First, we prove that any trace in $\mathcal{A}[A]_{\mathsf{p}}$ is a trace in $\mathcal{A}[B]_{\mathsf{p}}$:

Claim 1: $\forall\, w \in \Sigma^*.\ w$ is a trace in $\mathcal{A}[A]_{\mathsf{p}} \implies w$ is a trace in $\mathcal{A}[B]_{\mathsf{p}}$.

We prove the claim by induction on w. The base case, where $w = \varepsilon$, is trivially discharged by the fact that ε is a trace of all CSMs. In the inductive step, assume that w is a trace of $\mathcal{A}[A]_{\mathsf{p}}$. Let $x \in \Sigma$ such that wx is a trace of $\mathcal{A}[A]_{\mathsf{p}}$. We want to show that wx is also a trace of $\mathcal{A}[B]_{\mathsf{p}}$.

From the induction hypothesis, we know that w is also a trace of $\mathcal{A}[B]_{\mathsf{p}}$. Let ξ be the channel configuration uniquely determined by w. Let $(\vec{s}, \xi)$ be the $\mathcal{A}[A]_{\mathsf{p}}$ configuration reached on w, and let $(\vec{t}, \xi)$ be the $\mathcal{A}[B]_{\mathsf{p}}$ configuration reached on w.

Let q be the role such that $x \in \Sigma_{\mathsf{q}}$, and let s, t denote $\vec{s}_{\mathsf{q}}$, $\vec{t}_{\mathsf{q}}$ from the respective CSM configurations reached on w for $\mathcal{A}[A]_{\mathsf{p}}$ and $\mathcal{A}[B]_{\mathsf{p}}$.

To show that wx is a trace of $\mathcal{A}[B]_{\mathsf{p}}$, it suffices to show that there exists a state t' and a transition $t \xrightarrow{x} t'$ in B.

By the definition of state decoration (Definition 6.27), it follows that $t \in d_B(s)$. Because $\mathcal{A}[B]_{\mathsf{p}}$ refines $\mathbf{G}$ and is deadlock-free, it holds that all traces of $\mathcal{A}[B]_{\mathsf{p}}$ are prefixes of $\mathcal{L}(\mathbf{G})$. In other words, $w \in \text{pref}(\mathcal{L}(\mathbf{G}))$. Let ρ be a run such that $\rho \in I(w)$; such a run must exist from [Li et al. 2023a, Theorem 6.1] and [Li et al. 2023a, Lemma 6.3]. Let $\alpha \cdot G \xrightarrow{l} G' \cdot \beta$ be the unique splitting of ρ for q matching w. From Definition 6.6, it holds that $G \in d(t)$.

We proceed by case analysis on whether x is a send or receive event.

- Case $x \in \Sigma_{\mathsf{p},!}$. Let $x = \mathsf{p} \triangleright \mathsf{q}!m$. By assumption, there exists $s \xrightarrow{\mathsf{p}\triangleright\mathsf{q}!m} s'$ in δ_A. We instantiate *Send Decoration Subtype Validity* from C_2 with this transition to obtain:

$$\text{tr-orig}_B(d_B(s) \xrightarrow{\mathsf{p}\triangleright\mathsf{q}!m} d_B(s')) = d_B(s) \quad .$$

  From $t \in d_B(s)$, it follows immediately that there exists t' such that $t \xrightarrow{x} t'$ is a transition in B.

- Case $x \in \Sigma_{\mathsf{p},?}$. Let $x = \mathsf{p} \triangleleft \mathsf{q}?m$.

  We proceed by case analysis on $\mathtt{split}(l)\Downarrow_{\Sigma_{\mathsf{p}}}$. When $\mathtt{split}(l)\Downarrow_{\Sigma_{\mathsf{p}}} \in \Sigma_{\mathsf{p},?}$, from Lemma 6.26 there exists a transition $t \xrightarrow{\mathtt{split}(l)\Downarrow_{\Sigma_{\mathsf{p}}}} t'$ in δ_B, and from *Receive Subtype Exhaustivity* there exists a transition $s \xrightarrow{\texttt{split}(l)\Downarrow_{\Sigma_q}} s''$ in δ_A. We can apply Lemma 6.13 with ρ to conclude that $\texttt{split}(l)\Downarrow_{\Sigma_p} = x$: we satisfy the assumption that $q \triangleright p!m \notin M^{p}_{(G' \ldots)}$ by instantiating *Receive Decoration Subtype Validity* with $s \xrightarrow{x} s'$, $s \xrightarrow{\texttt{split}(l)\Downarrow_{\Sigma_q}} s''$, and G'. The fact that $t' \in \text{tr-dest}_B(d_B(s) \xrightarrow{\texttt{split}(l)\Downarrow_{\Sigma_p}} d_B(s''))$ follows from the existence of $t \xrightarrow{\texttt{split}(l)\Downarrow_{\Sigma_p}} t'$ in δ_B and the definition of state decoration (Definition 6.27). The fact that $G' \in \text{tr-dest}_B(d(t) \xrightarrow{\texttt{split}(l)\Downarrow_{\Sigma_p}} d(t'))$ follows from the fact that $\alpha \cdot G \xrightarrow{l} G' \cdot \beta$ is a run in $\mathbf{G}$ and Definition 6.6.

In the case that $\texttt{split}(l)\Downarrow_{\Sigma_p} \in \Sigma_{p,!}$, we again prove a contradiction. Because G is a send-originating global state, *Send Subtype Preservation* guarantees that there exists a transition $s \xrightarrow{x'} s''$ in A such that $x' \in \Sigma_{p,!}$. By *Send Decoration Validity*, x' originates from G in the projection by erasure, and we can find another run ρ' such that $\alpha' \cdot G \xrightarrow{l'} G'' \cdot \beta'$ is the unique splitting for p matching w, and $\texttt{split}(l')\Downarrow_{\Sigma_p} = x'$.

We can instantiate Lemma 6.13 with ρ' and $q \triangleright p!m \notin M^{p}_{(G'' \ldots)}$ to yield $\texttt{split}(l')\Downarrow_{\Sigma_p} = x$, which is a contradiction: x is a receive event and $\texttt{split}(l')\Downarrow_{\Sigma_p}$ is a send event.

This concludes our proof of Claim 1.

Next, we show that any trace that terminates in $\mathcal{A}[A]_p$ also terminates in $\mathcal{A}[B]_p$ and is maximal in $\mathcal{A}[A]_p$.

Claim 2: $\forall\, w \in \Sigma^*.\ w$ is terminated in $\mathcal{A}[A]_p \implies w$ is terminated in $\mathcal{A}[B]_p$ and w is maximal in $\mathcal{A}[A]_p$.

Let w be a terminated trace in $\mathcal{A}[A]_p$. Let ξ be the channel configuration uniquely determined by w. Let $(\vec{s}, \xi)$ be the $\mathcal{A}[A]_p$ configuration reached on w, and let $(\vec{t}, \xi)$ be the $\mathcal{A}[B]_p$ configuration reached on w. Let s, t denote $\vec{s}_p$, $\vec{t}_p$. First suppose by contradiction that w is not terminated in $\mathcal{A}[B]_p$. Because the state machines for all non-p roles are identical between the two CSMs, and because $\mathcal{A}[B]_p$ is deadlock-free by assumption, it must be the case that p witnesses the non-termination of w, in other words, B can perform an action that A cannot. Let x be the action that p can perform from t. Let G be a state in $d(t)$, such a state is guaranteed to exist by Claim 1 and the fact that no reachable states in B have empty decorating sets. Then, $w\Downarrow_{\Sigma_\mathsf{p}}$ reaches G from the initial state in the projection by erasure automaton. By the fact that w is a trace of $\mathcal{A}[A]_\mathsf{p}$, it holds that there exists a run with trace $w\Downarrow_{\Sigma_\mathsf{p}}$ in A. By the definition of state decoration, $t \in d_B(s)$.

- If $x \in \Sigma_!$, it follows that G is a send-originating global state. By *Send Subtype Preservation*, for any state in A that is decorated by a state in B that itself is decorated by at least one send-originating global state, of which t is one, there exists a transition $s \xrightarrow{x'} s'$ such that $x' \in \Sigma_{\mathsf{p},!}$. Because send transitions in a CSM are always enabled, role p can take this transition in $\mathcal{A}[A]_\mathsf{p}$. We reach a contradiction to the fact that w is terminated in $\mathcal{A}[A]_\mathsf{p}$.

- If $x \in \Sigma_?$, it follows that G is a receive-originating global state. From *Receive Subtype Exhaustivity*, any receive action that originates from any global state in $d(t)$ for any state $t \in d_B(s)$ must also originate from s. Therefore, there must exist s' such that $s \xrightarrow{x} s'$ is a transition in A. Thus, role p can take this transition in $\mathcal{A}[A]_\mathsf{p}$. We again reach a contradiction to the fact that w is terminated in $\mathcal{A}[A]_\mathsf{p}$.

To see that every terminated trace in $\mathcal{A}[A]_\mathsf{p}$ in maximal, from the above we know that w is terminated in $\mathcal{A}[B]_\mathsf{p}$. From the fact that $\mathcal{A}[B]_\mathsf{p}$ is deadlock-free, w is maximal in $\mathcal{A}[B]_\mathsf{p}$: all states in $\vec{t}$ are final and all channels in ξ are empty. Because t is a final state, by that fact that $\mathcal{A}[B]_\mathsf{p}$ refines $\mathbf{G}$ there exists a global state $G \in t$ such that the projection erasure automaton reaches G on $w\Downarrow_{\Sigma_\mathsf{p}}$ and G is a final state. Because A reaches s on $w\Downarrow_{\Sigma_\mathsf{p}}$, by the definitions of state decorations (Definitions 6.6 and 6.27), it holds that $G \in \bigcup_{t \in d_B(s)} d(t)$. By *Final State Validity*, it holds that s is a final state in A. This concludes our proof that any terminated trace in $\mathcal{A}[A]_\mathsf{p}$ is also a terminated trace in $\mathcal{A}[B]_\mathsf{p}$, and is maximal in $\mathcal{A}[A]_\mathsf{p}$.

Together, Claim 1 and 2 establish that $\mathcal{A}[A]_\mathsf{p}$ satisfies language inclusion with respect to $\mathcal{A}[B]_\mathsf{p}$ (Item ii), and deadlock freedom (Item iii). It remains to show that $\mathcal{A}[A]_\mathsf{p}$ also satisfies subprotocol fidelity (Item i). This follows immediately from [Majumdar et al. 2021a, Lemma 22], which states that all CSM languages are closed under $\sim$. □

Bibliography

S. Akshay, Ionut Dinca, Blaise Genest, and Alin Stefanescu. Implementing realistic asynchronous automata. In Anil Seth and Nisheeth K. Vishnoi, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, December 12-14, 2013, Guwahati, India*, volume 24 of *LIPIcs*, pages 213–224. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013. doi: 10.4230/LIPICS.FSTTCS.2013.213. URL https://doi.org/10.4230/LIPIcs.FSTTCS.2013.213.

Rajeev Alur and Mihalis Yannakakis. Model checking of message sequence charts. In Jos C. M. Baeten and Sjouke Mauw, editors, *CONCUR '99: Concurrency Theory, 10th International Conference, Eindhoven, The Netherlands, August 24-27, 1999, Proceedings*, volume 1664 of *Lecture Notes in Computer Science*, pages 114–129. Springer, 1999. doi: 10.1007/3-540-48320-9_10. URL https://doi.org/10.1007/3-540-48320-9_10.

Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Inference of message sequence charts. *IEEE Trans. Software Eng.*, 29(7):623–633, 2003. doi: 10.1109/TSE.2003.1214326. URL https://doi.org/10.1109/TSE.2003.1214326.

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

Lorenzo Bacchiani, Mario Bravetti, Julien Lange, and Gianluigi Zavattaro. A session subtyping tool. In Ferruccio Damiani and Ornela Dardha, editors, *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*, volume 12717 of *Lecture Notes in Computer Science*, pages 90–105. Springer, 2021. doi: 10.1007/978-3-030-78142-2_6. URL https://doi.org/10.1007/978-3-030-78142-2_6.

Franco Barbanera and Ugo De'Liguoro. Sub-behaviour relations for session-based client/server systems. *Mathematical Structures in Computer Science*, 25(6):1339–1381, 2015. doi: 10.1017/S096012951400005X.

Samik Basu and Tevfik Bultan. Choreography conformance via synchronizability. In Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors, *Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011*, pages 795–804. ACM, 2011. doi: 10.1145/1963405.1963516. URL https://doi.org/10.1145/1963405.1963516.

Giovanni Tito Bernardi and Matthew Hennessy. Modelling session types using contracts. *Math. Struct. Comput. Sci.*, 26(3):510–560, 2016. doi: 10.1017/S0960129514000243. URL https://doi.org/10.1017/S0960129514000243.

Nathalie Bertrand, Amélie Stainer, Thierry Jéron, and Moez Krichen. A game approach to determinize timed automata. *Formal Methods Syst. Des.*, 46(1):42–80, 2015. doi: 10.1007/S10703-014-0220-1. URL https://doi.org/10.1007/s10703-014-0220-1.

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

Laura Bocchi, Kohei Honda, Emilio Tuosto, and Nobuko Yoshida. A theory of design-by-contract for distributed multiparty interactions. In Paul Gastin and François Laroussinie, editors, *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*, volume 6269 of *Lecture Notes in Computer Science*, pages 162–176. Springer, 2010. doi: 10.1007/978-3-642-15375-4_12. URL https://doi.org/10.1007/978-3-642-15375-4_12.

Laura Bocchi, Romain Demangeon, and Nobuko Yoshida. A multiparty multi-session logic. In Catuscia Palamidessi and Mark Dermot Ryan, editors, *Trustworthy Global Computing - 7th International Symposium, TGC 2012, Newcastle upon Tyne, UK, September 7-8, 2012, Revised Selected Papers*, volume 8191 of *Lecture Notes in Computer Science*, pages 97–111. Springer, 2012. doi: 10.1007/978-3-642-41157-1_7. URL https://doi.org/10.1007/978-3-642-41157-1_7.

Benedikt Bollig, Cinzia Di Giusto, Alain Finkel, Laetitia Laversa, Étienne Lozes, and Amrita Suresh. A unifying framework for deciding synchronizability. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPIcs*, pages 14:1–14:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi: 10.4230/LIPICS.CONCUR.2021.14. URL https://doi.org/10.4230/LIPIcs.CONCUR.2021.14.

Ahmed Bouajjani, Constantin Enea, Kailiang Ji, and Shaz Qadeer. On the completeness of verifying message passing programs under bounded asynchrony. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II*, volume 10982 of *Lecture Notes in Computer Science*, pages 372–391. Springer, 2018. doi: 10.1007/978-3-319-96142-2_23. URL https://doi.org/10.1007/978-3-319-96142-2_23.

Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983. doi: 10.1145/322374.322380. URL https://doi.org/10.1145/322374.322380.

Mario Bravetti and Gianluigi Zavattaro. Relating session types and behavioural contracts: The asynchronous case. In Peter Csaba Ölveczky and Gwen Salaün, editors, *Software Engineering and Formal Methods*, pages 29–47, Cham, 2019. Springer International Publishing. ISBN 978-3-030-30446-1.

Mario Bravetti and Gianluigi Zavattaro. Asynchronous session subtyping as communicating automata refinement. *Softw. Syst. Model.*, 20(2):311–333, apr 2021. ISSN 1619-1366. doi: 10.1007/s10270-020-00838-x. URL https://doi.org/10.1007/s10270-020-00838-x.

Mario Bravetti, Marco Carbone, and Gianluigi Zavattaro. On the boundary between decidability and undecidability of asynchronous session subtyping. *Theor. Comput. Sci.*, 722:19–51, 2018. doi: 10.1016/j.tcs.2018.02.010. URL https://doi.org/10.1016/j.tcs.2018.02.010.

Mario Bravetti, Marco Carbone, Julien Lange, Nobuko Yoshida, and Gianluigi Zavattaro. A sound algorithm for asynchronous session subtyping and its implementation. *Log. Methods Comput. Sci.*, 17(1), 2021a. URL https://lmcs.episciences.org/7238.

Mario Bravetti, Julien Lange, and Gianluigi Zavattaro. Fair refinement for asynchronous session types. In Stefan Kiefer and Christine Tasson, editors, *Foundations of Software Science and Computation Structures - 24th International Conference, FOSSACS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings*, volume 12650 of *Lecture Notes in Computer Science*, pages 144–163. Springer, 2021b. doi: 10.1007/978-3-030-71995-1_8. URL https://doi.org/10.1007/978-3-030-71995-1_8.

Matthew Alan Le Brun and Ornela Dardha. Magπ: Types for failure-prone communication. In Thomas Wies, editor, *Programming Languages and Systems - 32nd European Symposium on Programming, ESOP 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings*, volume 13990 of *Lecture*

Notes in Computer Science, pages 363–391. Springer, 2023. doi: 10.1007/978-3-031-30044-8_14. URL https://doi.org/10.1007/978-3-031-30044-8_14.

Luís Caires and Jorge A. Pérez. Multiparty session types within a canonical binary theory, and beyond. In Elvira Albert and Ivan Lanese, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 36th IFIP WG 6.1 International Conference, FORTE 2016, Held as Part of the 11th International Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete, Greece, June 6-9, 2016, Proceedings*, volume 9688 of *Lecture Notes in Computer Science*, pages 74–95. Springer, 2016. doi: 10.1007/978-3-319-39570-8_6. URL https://doi.org/10.1007/978-3-319-39570-8_6.

Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logic propositions as session types. *Math. Struct. Comput. Sci.*, 26(3):367–423, 2016. doi: 10.1017/S0960129514000218. URL https://doi.org/10.1017/S0960129514000218.

Marco Carbone, Sam Lindley, Fabrizio Montesi, Carsten Schürmann, and Philip Wadler. Coherence generalises duality: A logical explanation of multiparty session types. In Josée Desharnais and Radha Jagadeesan, editors, *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, volume 59 of *LIPIcs*, pages 33:1–33:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi: 10.4230/LIPIcs.CONCUR.2016.33. URL https://doi.org/10.4230/LIPIcs.CONCUR.2016.33.

Marco Carbone, Fabrizio Montesi, Carsten Schürmann, and Nobuko Yoshida. Multiparty session types as coherence proofs. *Acta Informatica*, 54(3):243–269, 2017. doi: 10.1007/s00236-016-0285-y. URL https://doi.org/10.1007/s00236-016-0285-y.

Filipe Casal, Andreia Mordido, and Vasco T. Vasconcelos. Mixed sessions. *Theor. Comput. Sci.*, 897:23–48, 2022. doi: 10.1016/j.tcs.2021.08.005. URL https://doi.org/10.1016/j.tcs.2021.08.005.

Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A theory of contracts for web services. *ACM Trans. Program. Lang. Syst.*, 31(5):19:1–19:61, 2009. doi: 10.1145/1538917.1538920. URL https://doi.org/10.1145/1538917.1538920.

Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, and Luca Padovani. On global types and multi-party session. *Log. Methods Comput. Sci.*, 8(1), 2012. doi: 10.2168/LMCS-8(1:24)2012. URL https://doi.org/10.2168/LMCS-8(1:24)2012.

Ilaria Castellani, Mariangiola Dezani-Ciancaglini, Paola Giannini, and Ross Horne. Global types with internal delegation. *Theor. Comput. Sci.*, 807:128–153, 2020. doi: 10.1016/j.tcs.2019.09.027. URL https://doi.org/10.1016/j.tcs.2019.09.027.

Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Paola Giannini. Asynchronous sessions with input races. In Marco Carbone and Rumyana Neykova, editors, *Proceedings of the 13th International Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software, PLACES@ETAPS 2022, Munich, Germany, 3rd April 2022*, volume 356 of *EPTCS*, pages 12–23, 2022. doi: 10.4204/EPTCS.356.2. URL https://doi.org/10.4204/EPTCS.356.2.

Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Paola Giannini. Global types and event structure semantics for asynchronous multiparty sessions. *Fundam. Informaticae*, 192(1):1–75, 2024. doi: 10.3233/FI-242188. URL https://doi.org/10.3233/FI-242188.

David Castro-Perez and Nobuko Yoshida. Dynamically updatable multiparty session protocols: Generating concurrent go code from unbounded protocols. In Karim Ali and Guido Salvaneschi, editors, *37th European Conference on Object-Oriented Programming, ECOOP 2023, July 17-21, 2023, Seattle, Washington, United States*, volume 263 of *LIPIcs*, pages 6:1–6:30. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi: 10.4230/LIPICS.ECOOP.2023.6. URL https://doi.org/10.4230/LIPIcs.ECOOP.2023.6.

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

David Castro-Perez, Francisco Ferreira, Lorenzo Gheri, and Nobuko Yoshida. Zooid: a DSL for certified multiparty computation: from mechanised metatheory to certified multiparty processes. In Stephen N. Freund and Eran Yahav, editors, *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, pages 237–251. ACM, 2021. doi: 10.1145/3453483.3454041. URL https://doi.org/10.1145/3453483.3454041.

Minas Charalambides, Peter Dinges, and Gul A. Agha. Parameterized, concurrent session types for asynchronous multi-actor interactions. *Sci. Comput. Program.*, 115-116:100–126, 2016. doi: 10.1016/j.scico.2015.10.006. URL https://doi.org/10.1016/j.scico.2015.10.006.

Tzu-Chun Chen. Lightening global types. *J. Log. Algebraic Methods Program.*, 84(5):708–729, 2015. doi: 10.1016/J.JLAMP.2015.06.003. URL https://doi.org/10.1016/j.jlamp.2015.06.003.

Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. On the preciseness of subtyping in session types: 10 years later. In Alessandro Bruni, Alberto Momigliano, Matteo Pradella, Matteo Rossi, and James Cheney, editors, *Proceedings of the 26th International Symposium on Principles and Practice of Declarative Programming, PPDP 2024, Milano, Italy, September 9-11, 2024*, pages 2:1–2:3. ACM, 2024. doi: 10.1145/3678232.3678258. URL https://doi.org/10.1145/3678232.3678258.

Florent Chevrou, Aurélie Hurault, and Philippe Quéinnec. On the diversity of asynchronous communication. *Formal Aspects Comput.*, 28(5):847–879, 2016. doi: 10.1007/S00165-016-0379-X. URL https://doi.org/10.1007/s00165-016-0379-x.

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

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

Mario Coppo, Mariangiola Dezani-Ciancaglini, Luca Padovani, and Nobuko Yoshida. A gentle introduction to multiparty asynchronous session types. In Marco Bernardo and Einar Broch Johnsen, editors, *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*, volume 9104 of *Lecture Notes in Computer Science*, pages 146–178. Springer, 2015. doi: 10.1007/978-3-319-18941-3_4. URL https://doi.org/10.1007/978-3-319-18941-3_4.

Robert Cori, Yves Métivier, and Wieslaw Zielonka. Asynchronous mappings and asynchronous cellular automata. *Inf. Comput.*, 106(2):159–202, 1993. doi: 10.1006/INCO.1993.1052. URL https://doi.org/10.1006/inco.1993.1052.

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

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

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

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

Francesco Dagnino, Paola Giannini, and Mariangiola Dezani-Ciancaglini. Deconfined global types for asynchronous sessions. In Ferruccio Damiani and Ornela Dardha, editors, *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*, volume 12717 of *Lecture Notes in Computer Science*, pages 41–60. Springer, 2021. doi: 10.1007/978-3-030-78142-2_3. URL https://doi.org/10.1007/978-3-030-78142-2_3.

Haitao Dan, Robert M. Hierons, and Steve Counsell. Non-local choice and implied scenarios. In José Luiz Fiadeiro, Stefania Gnesi, and Andrea Maggiolo-Schettini, editors, *8th IEEE International Conference on Software Engineering and Formal Methods, SEFM 2010, Pisa, Italy, 13-18 September 2010*, pages 53–62. IEEE Computer Society, 2010. doi: 10.1109/SEFM.2010.14. URL https://doi.org/10.1109/SEFM.2010.14.

Loris D'Antoni and Margus Veanes. The power of symbolic automata and transducers. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, volume 10426 of *Lecture Notes in Computer Science*, pages 47–67. Springer, 2017. doi: 10.1007/978-3-319-63387-9_3. URL https://doi.org/10.1007/978-3-319-63387-9_3.

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

Ankush Das and Frank Pfenning. Session types with arithmetic refinements. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPIcs*, pages 13:1–13:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi: 10.4230/LIPICS.CONCUR.2020.13. URL https://doi.org/10.4230/LIPIcs.CONCUR.2020.13.

Ankush Das and Frank Pfenning. Rast: A language for resource-aware session types. *Log. Methods Comput. Sci.*, 18(1), 2022. doi: 10.46298/LMCS-18(1:9)2022. URL https://doi.org/10.46298/lmcs-18(1:9)2022.

Ankush Das, Stephanie Balzer, Jan Hoffmann, Frank Pfenning, and Ishani Santurkar. Resource-aware session types for digital contracts. In *34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, June 21-25, 2021*, pages 1–16. IEEE, 2021. doi: 10.1109/CSF51468.2021.00004. URL https://doi.org/10.1109/CSF51468.2021.00004.

Jan de Muijnck-Hughes and Wim Vanderbauwhede. A Typing Discipline for Hardware Interfaces. In Alastair F. Donaldson, editor, *33rd European Conference on Object-Oriented Programming (ECOOP 2019)*, volume 134 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:27, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-111-5. doi: 10.4230/LIPIcs.ECOOP.2019.6. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ECOOP.2019.6.

Romain Delpy, Anca Muscholl, and Grégoire Sutre. An automata-based approach for synchronizable mailbox communication. In Rupak Majumdar and Alexandra Silva, editors, *35th International Conference on Concurrency Theory, CONCUR 2024, September 9-13, 2024, Calgary, Canada*, volume 311 of *LIPIcs*, pages 22:1–22:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi: 10.4230/LIPICS.CONCUR.2024.22. URL https://doi.org/10.4230/LIPIcs.CONCUR.2024.22.

Romain Demangeon, Kohei Honda, Raymond Hu, Rumyana Neykova, and Nobuko Yoshida. Practical interruptible conversations: distributed dynamic verification with multiparty session types and python. *Formal Methods Syst. Des.*, 46(3):197–225, 2015. doi: 10.1007/S10703-014-0218-8. URL https://doi.org/10.1007/s10703-014-0218-8.

Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty session types meet communicating automata. In Helmut Seidl, editor, *Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7211 of *Lecture Notes in Computer Science*, pages 194–213. Springer, 2012. doi: 10.1007/978-3-642-28869-2_10. URL https://doi.org/10.1007/978-3-642-28869-2_10.

Pierre-Malo Deniélou, Nobuko Yoshida, Andi Bejleri, and Raymond Hu. Parameterised multiparty session types. *Log. Methods Comput. Sci.*, 8(4), 2012. doi: 10.2168/LMCS-8(4:6)2012. URL https://doi.org/10.2168/LMCS-8(4:6)2012.

Volker Diekert and Grzegorz Rozenberg, editors. *The Book of Traces*. World Scientific, 1995. ISBN 978-981-02-2058-7. doi: 10.1142/2563. URL https://doi.org/10.1142/2563.

Burak Ekici and Nobuko Yoshida. Completeness of asynchronous session tree subtyping in Coq. In *15th International Conference on Interactive Theorem Proving, ITP 2024, September 9-14, 2024, Tbilisi, Georgia*, volume 309 of *LIPIcs*, pages 13:1–13:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi: 10.4230/LIPICS.ITP.2024.13. URL https://doi.org/10.4230/LIPIcs.ITP.2024.13.

Keith Ellul, Bryan Krawetz, Jeffrey O. Shallit, and Ming-wei Wang. Regular expressions: New results and open problems. *J. Autom. Lang. Comb.*, 10(4):407–437, 2005. doi: 10.25596/jalc-2005-407. URL https://doi.org/10.25596/jalc-2005-407.

Javier Esparza and Mogens Nielsen. Decidability issues for petri nets - a survey. *J. Inf. Process. Cybern.*, 30(3):143–160, 1994.

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

Azadeh Farzan. Commutativity in automated verification. In *38th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2023, Boston, MA, USA, June 26-29, 2023*, pages 1–7. IEEE, 2023. doi: 10.1109/LICS56636.2023.10175734. URL https://doi.org/10.1109/LICS56636.2023.10175734.

Azadeh Farzan, Dominik Klumpp, and Andreas Podelski. Stratified commutativity in verification algorithms for concurrent programs. *Proc. ACM Program. Lang.*, 7(POPL):1426–1453, 2023. doi: 10.1145/3571242. URL https://doi.org/10.1145/3571242.

Alain Finkel and Étienne Lozes. Synchronizability of communicating finite state machines is not decidable. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 122:1–122:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi: 10.4230/LIPIcs.ICALP.2017.122. URL https://doi.org/10.4230/LIPIcs.ICALP.2017.122.

Alain Finkel and Étienne Lozes. Synchronizability of communicating finite state machines is not decidable. *Log. Methods Comput. Sci.*, 19(4), 2023. doi: 10.46298/LMCS-19(4:33)2023. URL https://doi.org/10.46298/lmcs-19(4:33)2023.

Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983. doi: 10.1145/357195.357200. URL https://doi.org/10.1145/357195.357200.

Joshua Gancher, Sydney Gibson, Pratap Singh, Samvid Dharanikota, and Bryan Parno. Owl: Compositional verification of security protocols via an information-flow type system. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 1130–1147. IEEE, 2023. doi: 10.1109/SP46215.2023.10179477. URL https://doi.org/10.1109/SP46215.2023.10179477.

Simon J. Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Informatica*, 42(2-3):191–225, 2005. doi: 10.1007/s00236-005-0177-z. URL https://doi.org/10.1007/s00236-005-0177-z.

Thomas Gazagnaire, Blaise Genest, Loïc Hélouët, P. S. Thiagarajan, and Shaofa Yang. Causal message sequence charts. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *CONCUR 2007 - Concurrency Theory, 18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceedings*, volume 4703 of *Lecture Notes in Computer Science*, pages 166–180. Springer, 2007. doi: 10.1007/978-3-540-74407-8_12. URL https://doi.org/10.1007/978-3-540-74407-8_12.

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

Blaise Genest, Anca Muscholl, and Doron A. Peled. Message sequence charts. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *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]*, volume 3098 of *Lecture Notes in Computer Science*, pages 537–558. Springer, 2003. doi: 10.1007/978-3-540-27755-2_15. URL https://doi.org/10.1007/978-3-540-27755-2_15.

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

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

Blaise Genest, Hugo Gimbert, Anca Muscholl, and Igor Walukiewicz. Optimal zielonka-type construction of deterministic asynchronous automata. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 52–63. Springer, 2010. doi: 10.1007/978-3-642-14162-1_5. URL https://doi.org/10.1007/978-3-642-14162-1_5.

Lorenzo Gheri, Ivan Lanese, Neil Sayers, Emilio Tuosto, and Nobuko Yoshida. Design-by-contract for flexible multiparty session protocols. In Karim Ali and Jan Vitek, editors, *36th European Conference on Object-Oriented Programming, ECOOP 2022, June 6-10, 2022, Berlin, Germany*, volume 222 of *LIPIcs*, pages 8:1–8:28. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi: 10.4230/LIPICS.ECOOP.2022.8. URL https://doi.org/10.4230/LIPIcs.ECOOP.2022.8.

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

Silvia Ghilezan, Svetlana Jakšić, Jovanka Pantović, Alceste Scalas, and Nobuko Yoshida. Precise subtyping for synchronous multiparty sessions. *Journal of Logical and Algebraic Methods in Programming*, 104:127–173, 2019b. ISSN 2352-2208. doi: https://doi.org/10.1016/j.jlamp.2018.12.002. URL https://www.sciencedirect.com/science/article/pii/S2352220817302237.

Silvia Ghilezan, Jovanka Pantovic, Ivan Prokic, Alceste Scalas, and Nobuko Yoshida. Precise subtyping for asynchronous multiparty sessions. *Proc. ACM Program. Lang.*, 5(POPL):1–28, 2021. doi: 10.1145/3434297. URL https://doi.org/10.1145/3434297.

Saverio Giallorenzo, Fabrizio Montesi, Marco Peressotti, David Richter, Guido Salvaneschi, and Pascal Weisenburger. Multiparty languages: The choreographic and multitier cases (pearl).

In Anders Møller and Manu Sridharan, editors, *35th European Conference on Object-Oriented Programming, ECOOP 2021, July 11-17, 2021, Aarhus, Denmark (Virtual Conference)*, volume 194 of *LIPIcs*, pages 22:1–22:27. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi: 10.4230/LIPIcs.ECOOP.2021.22. URL https://doi.org/10.4230/LIPIcs.ECOOP.2021.22.

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

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

Cinzia Di Giusto, Davide Ferré, Laetitia Laversa, and Étienne Lozes. A partial order view of message-passing communication models. *Proc. ACM Program. Lang.*, 7(POPL):1601–1627, 2023. doi: 10.1145/3571248. URL https://doi.org/10.1145/3571248.

Patrice Godefroid and Mihalis Yannakakis. Analysis of boolean programs. In Nir Piterman and Scott A. Smolka, editors, *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*, volume 7795 of *Lecture Notes in Computer Science*, pages 214–229. Springer, 2013. doi: 10.1007/978-3-642-36742-7_16. URL https://doi.org/10.1007/978-3-642-36742-7_16.

Hannah Gommerstadt, Limin Jia, and Frank Pfenning. Session-typed concurrent contracts. In Amal Ahmed, editor, *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*, volume 10801 of *Lecture Notes in Computer Science*, pages 771–798. Springer, 2018. doi: 10.1007/978-3-319-89884-1_27. URL https://doi.org/10.1007/978-3-319-89884-1_27.

Dennis Griffith and Elsa L. Gunter. Liquidpi: Inferrable dependent session types. In Guillaume Brat, Neha Rungta, and Arnaud Venet, editors, *NASA Formal Methods, 5th International Symposium, NFM 2013, Moffett Field, CA, USA, May 14-16, 2013. Proceedings*, volume 7871 of *Lecture Notes in Computer Science*, pages 185–197. Springer, 2013. doi: 10.1007/978-3-642-38088-4_13. URL https://doi.org/10.1007/978-3-642-38088-4_13.

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

Jonas Kastberg Hinrichsen, Daniël Louwrink, Robbert Krebbers, and Jesper Bengtson. Machine-checked semantic session typing. In Catalin Hritcu and Andrei Popescu, editors, *CPP '21: 10th ACM SIGPLAN International Conference on Certified Programs and Proofs, Virtual Event, Denmark, January 17-19, 2021*, pages 178–198. ACM, 2021. doi: 10.1145/3437992.3439914. URL https://doi.org/10.1145/3437992.3439914.

Jonas Kastberg Hinrichsen, Jesper Bengtson, and Robbert Krebbers. Actris 2.0: Asynchronous session-type based reasoning in separation logic. *Log. Methods Comput. Sci.*, 18(2), 2022. doi: 10.46298/LMCS-18(2:16)2022. URL https://doi.org/10.46298/lmcs-18(2:16)2022.

Jonas Kastberg Hinrichsen, Jules Jacobs, and Robbert Krebbers. Multris: Functional verification of multiparty message passing in separation logic. 2024. URL https://jihgfee.github.io/papers/multris_manuscript.pdf.

Andrew K. Hirsch and Deepak Garg. Pirouette: Higher-order typed functional choreographies. *CoRR*, abs/2111.03484, 2021. URL https://arxiv.org/abs/2111.03484.

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

Kohei Honda. Types for dyadic interaction. In Eike Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993. doi: 10.1007/3-540-57208-2_35. URL https://doi.org/10.1007/3-540-57208-2_35.

Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In Chris Hankin, editor, *Programming Languages and Systems - ESOP'98, 7th European Symposium on Programming, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 1998. doi: 10.1007/BFb0053567. URL https://doi.org/10.1007/BFb0053567.

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

Kohei Honda, Eduardo R. B. Marques, Francisco Martins, Nicholas Ng, Vasco Thudichum Vasconcelos, and Nobuko Yoshida. Verification of MPI programs using session types. In Jesper Larsson Träff, Siegfried Benkner, and Jack J. Dongarra, editors, *Recent Advances in the Message Passing Interface - 19th European MPI Users' Group Meeting, EuroMPI 2012, Vienna, Austria, September 23-26, 2012. Proceedings*, volume 7490 of *Lecture Notes in Computer Science*, pages 291–293. Springer, 2012. doi: 10.1007/978-3-642-33518-1_37. URL https://doi.org/10.1007/978-3-642-33518-1_37.

Ross Horne. Session subtyping and multiparty compatibility using circular sequents. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR*

2020, September 1-4, 2020, Vienna, Austria (Virtual Conference), volume 171 of *LIPIcs*, pages 12:1–12:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi: 10.4230/LIPIcs.CONCUR.2020.12. URL https://doi.org/10.4230/LIPIcs.CONCUR.2020.12.

Raymond Hu and Nobuko Yoshida. Hybrid session verification through endpoint API generation. In Perdita Stevens and Andrzej Wasowski, editors, *Fundamental Approaches to Software Engineering - 19th International Conference, FASE 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9633 of *Lecture Notes in Computer Science*, pages 401–418. Springer, 2016. doi: 10.1007/978-3-662-49665-7_24. URL https://doi.org/10.1007/978-3-662-49665-7_24.

Raymond Hu and Nobuko Yoshida. Explicit connection actions in multiparty session types. In Marieke Huisman and Julia Rubin, editors, *Fundamental Approaches to Software Engineering - 20th International Conference, FASE 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10202 of *Lecture Notes in Computer Science*, pages 116–133. Springer, 2017. doi: 10.1007/978-3-662-54494-5_7. URL https://doi.org/10.1007/978-3-662-54494-5_7.

Keigo Imai, Rumyana Neykova, Nobuko Yoshida, and Shoji Yuen. Multiparty session programming with global protocol combinators. In Robert Hirschfeld and Tobias Pape, editors, *34th European Conference on Object-Oriented Programming, ECOOP 2020, November 15-17, 2020, Berlin, Germany (Virtual Conference)*, volume 166 of *LIPIcs*, pages 9:1–9:30. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi: 10.4230/LIPICS.ECOOP.2020.9. URL https://doi.org/10.4230/LIPIcs.ECOOP.2020.9.

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

Jules Jacobs, Stephanie Balzer, and Robbert Krebbers. Connectivity graphs: a method for proving deadlock freedom based on separation logic. *Proc. ACM Program. Lang.*, 6(POPL):1–33, 2022. doi: 10.1145/3498662. URL https://doi.org/10.1145/3498662.

Jules Jacobs, Jonas Kastberg Hinrichsen, and Robbert Krebbers. Dependent session protocols in separation logic from first principles (functional pearl). *Proc. ACM Program. Lang.*, 7(ICFP): 768–795, 2023. doi: 10.1145/3607856. URL https://doi.org/10.1145/3607856.

Jules Jacobs, Jonas Kastberg Hinrichsen, and Robbert Krebbers. Deadlock-free separation logic: Linearity yields progress for dependent higher-order message passing. *Proc. ACM Program. Lang.*, 8(POPL):1385–1417, 2024. doi: 10.1145/3632889. URL https://doi.org/10.1145/3632889.

Sung-Shik Jongmans and Petra van den Bos. A predicate transformer for choreographies - computing preconditions in choreographic programming. In Ilya Sergey, editor, *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*, volume 13240 of *Lecture Notes in Computer Science*, pages 520–547. Springer, 2022. doi: 10.1007/978-3-030-99336-8_19. URL https://doi.org/10.1007/978-3-030-99336-8_19.

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

Shun Kashiwa, Gan Shen, Soroush Zare, and Lindsey Kuper. Portable, efficient, and practical library-level choreographic programming, 2023. URL https://arxiv.org/abs/2311.11472.

Bernhard Kragl, Shaz Qadeer, and Thomas A. Henzinger. Synchronizing the asynchronous. In Sven Schewe and Lijun Zhang, editors, *29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China*, volume 118 of *LIPIcs*, pages 21:1–21:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi: 10.4230/LIPICS.CONCUR.2018.21. URL https://doi.org/10.4230/LIPIcs.CONCUR.2018.21.

Bernhard Kragl, Constantin Enea, Thomas A. Henzinger, Suha Orhun Mutluergil, and Shaz Qadeer. Inductive sequentialization of asynchronous programs. In Alastair F. Donaldson and Emina Torlak, editors, *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, pages 227–242. ACM, 2020. doi: 10.1145/3385412.3385980. URL https://doi.org/10.1145/3385412.3385980.

Nicolas Lagaillardie, Rumyana Neykova, and Nobuko Yoshida. Stay safe under panic: Affine rust programming with multiparty session types (artifact). *Dagstuhl Artifacts Ser.*, 8(2):09:1–09:16, 2022. doi: 10.4230/DARTS.8.2.9. URL https://doi.org/10.4230/DARTS.8.2.9.

Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. In *Concurrency: the Works of Leslie Lamport*, pages 179–196. 2019. doi: 10.1145/3335772.3335934. URL https://doi.org/10.1145/3335772.3335934.

Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi, and Alan Schmitt. On the expressiveness and decidability of higher-order process calculi. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 145–155. IEEE Computer Society, 2008. doi: 10.1109/LICS.2008.8. URL https://doi.org/10.1109/LICS.2008.8.

Julien Lange and Nobuko Yoshida. Characteristic formulae for session types. In Marsha Chechik and Jean-François Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Sys-*

tems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings, volume 9636 of *Lecture Notes in Computer Science*, pages 833–850. Springer, 2016. doi: 10.1007/978-3-662-49674-9_52. URL https://doi.org/10.1007/978-3-662-49674-9_52.

Julien Lange and Nobuko Yoshida. On the undecidability of asynchronous session subtyping. In Javier Esparza and Andrzej S. Murawski, editors, *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10203 of *Lecture Notes in Computer Science*, pages 441–457, 2017. doi: 10.1007/978-3-662-54458-7_26. URL https://doi.org/10.1007/978-3-662-54458-7_26.

Julien Lange and Nobuko Yoshida. Verifying asynchronous interactions via communicating session automata. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 97–117. Springer, 2019. doi: 10.1007/978-3-030-25540-4_6. URL https://doi.org/10.1007/978-3-030-25540-4_6.

Julien Lange, Nicholas Ng, Bernardo Toninho, and Nobuko Yoshida. A static verification framework for message passing in go using behavioural types. In Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman, editors, *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pages 1137–1148. ACM, 2018. doi: 10.1145/3180155.3180157. URL https://doi.org/10.1145/3180155.3180157.

Languages, Systems, and Data Lab, UC Santa Cruz. Chorus: Choreographic programming in rust. https://lsd-ucsc.github.io/ChoRus/introduction.html. Accessed: 2025-07-10.

Elaine Li and Thomas Wies. Implementability of global protocols modulo network architectures. Under submission, 2025a.

Elaine Li and Thomas Wies. Certified implementability of global multiparty protocols. *To appear at Interactive Theorem Proving 2025*, 2025b.

Elaine Li, Felix Stutz, Thomas Wies, and Damien Zufferey. Complete multiparty session type projection with automata. In Constantin Enea and Akash Lal, editors, *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part III*, volume 13966 of *Lecture Notes in Computer Science*, pages 350–373. Springer, 2023a. doi: 10.1007/978-3-031-37709-9_17. URL https://doi.org/10.1007/978-3-031-37709-9_17.

Elaine Li, Felix Stutz, Thomas Wies, and Damien Zufferey. Complete multiparty session type projection with automata. *CoRR*, abs/2305.17079, 2023b. doi: 10.48550/ARXIV.2305.17079. URL https://doi.org/10.48550/arXiv.2305.17079.

Elaine Li, Felix Stutz, and Thomas Wies. Deciding subtyping for asynchronous multiparty sessions. In Stephanie Weirich, editor, *Programming Languages and Systems - 33rd European Symposium on Programming, ESOP 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part I*, volume 14576 of *Lecture Notes in Computer Science*, pages 176–205. Springer, 2024. doi: 10.1007/978-3-031-57262-3_8. URL https://doi.org/10.1007/978-3-031-57262-3_8.

Elaine Li, Felix Stutz, Thomas Wies, and Damien Zufferey. Sprout: A verifier for symbolic multiparty protocols. 2025a. doi: 10.1007/978-3-031-98682-6. URL https://doi.org/10.1007/978-3-031-98682-6. 1st edition, ISBN for softcover and eBook.

Elaine Li, Felix Stutz, Thomas Wies, and Damien Zufferey. Characterizing implementability of global protocols with infinite states and data. *Proc. ACM Program. Lang.*, 9(OOPSLA1):1434–1463, 2025b. doi: 10.1145/3720493. URL https://doi.org/10.1145/3720493.

Elaine Li, Felix Stutz, Thomas Wies, and Damien Zufferey. Characterizing implementability of global protocols with infinite states and data, 2025c. URL https://arxiv.org/abs/2411.05722.

Richard J. Lipton. Reduction: A method of proving properties of parallel programs. *Commun. ACM*, 18(12):717–721, 1975. doi: 10.1145/361227.361234. URL https://doi.org/10.1145/361227.361234.

Barbara Liskov and Jeannette M. Wing. A behavioral notion of subtyping. *ACM Trans. Program. Lang. Syst.*, 16(6):1811–1841, 1994. doi: 10.1145/197320.197383. URL https://doi.org/10.1145/197320.197383.

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

Rupak Majumdar, Marcus Pirron, Nobuko Yoshida, and Damien Zufferey. Motion session types for robotic interactions (brave new idea paper). In Alastair F. Donaldson, editor, *33rd European Conference on Object-Oriented Programming, ECOOP 2019, July 15-19, 2019, London, United Kingdom*, volume 134 of *LIPIcs*, pages 28:1–28:27. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi: 10.4230/LIPIcs.ECOOP.2019.28. URL https://doi.org/10.4230/LIPIcs.ECOOP.2019.28.

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

Rupak Majumdar, Madhavan Mukund, Felix Stutz, and Damien Zufferey. Generalising projection in asynchronous multiparty session types. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPIcs*, pages 35:1–35:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021a. doi: 10.4230/LIPICS.CONCUR.2021.35. URL https://doi.org/10.4230/LIPIcs.CONCUR.2021.35.

Rupak Majumdar, Madhavan Mukund, Felix Stutz, and Damien Zufferey. Generalising projection in asynchronous multiparty session types. *CoRR*, abs/2107.03984, 2021b. URL https://arxiv.org/abs/2107.03984.

Petar Maksimovic and Alan Schmitt. Hocore in coq. In Christian Urban and Xingyuan Zhang, editors, *Interactive Theorem Proving - 6th International Conference, ITP 2015, Nanjing, China, August 24-27, 2015, Proceedings*, volume 9236 of *Lecture Notes in Computer Science*, pages 278–293. Springer, 2015. doi: 10.1007/978-3-319-22102-1_19. URL https://doi.org/10.1007/978-3-319-22102-1_19.

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

Fabrizio Montesi. *Introduction to Choreographies*. Cambridge University Press, 2023. doi: 10.1017/9781108981491.

Rémi Morin. Recognizable sets of message sequence charts. In Helmut Alt and Afonso Ferreira, editors, *STACS 2002, 19th Annual Symposium on Theoretical Aspects of Computer Science, Antibes - Juan les Pins, France, March 14-16, 2002, Proceedings*, volume 2285 of *Lecture Notes in Computer Science*, pages 523–534. Springer, 2002. doi: 10.1007/3-540-45841-7_43. URL https://doi.org/10.1007/3-540-45841-7_43.

Dimitris Mostrous and Nobuko Yoshida. Session-based communication optimisation for higher-order mobile processes. In Pierre-Louis Curien, editor, *Typed Lambda Calculi and Appli-*

cations, 9th International Conference, TLCA 2009, Brasilia, Brazil, July 1-3, 2009. Proceedings, volume 5608 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2009. doi: 10.1007/978-3-642-02273-9_16. URL https://doi.org/10.1007/978-3-642-02273-9_16.

Dimitris Mostrous, Nobuko Yoshida, and Kohei Honda. Global principal typing in partially commutative asynchronous sessions. In Giuseppe Castagna, editor, *Programming Languages and Systems, 18th European Symposium on Programming, ESOP 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5502 of *Lecture Notes in Computer Science*, pages 316–332. Springer, 2009. doi: 10.1007/978-3-642-00590-9_23. URL https://doi.org/10.1007/978-3-642-00590-9_23.

Madhavan Mukund. *From Global Specifications to Distributed Implementations*, pages 19–35. Springer US, Boston, MA, 2002. ISBN 978-1-4757-6656-1. doi: 10.1007/978-1-4757-6656-1_2. URL https://doi.org/10.1007/978-1-4757-6656-1_2.

Madhavan Mukund and Milind A. Sohoni. Keeping track of the latest gossip in a distributed system. *Distributed Comput.*, 10(3):137–148, 1997. doi: 10.1007/S004460050031. URL https://doi.org/10.1007/s004460050031.

Anca Muscholl and Doron A. Peled. Message sequence graphs and decision problems on mazurkiewicz traces. In Miroslaw Kutylowski, Leszek Pacholski, and Tomasz Wierzbicki, editors, *Mathematical Foundations of Computer Science 1999, 24th International Symposium, MFCS'99, Szklarska Poreba, Poland, September 6-10, 1999, Proceedings*, volume 1672 of *Lecture Notes in Computer Science*, pages 81–91. Springer, 1999. doi: 10.1007/3-540-48340-3_8. URL https://doi.org/10.1007/3-540-48340-3_8.

Rumyana Neykova and Nobuko Yoshida. Multiparty session actors. *Log. Methods Comput. Sci.*, 13

(1), 2017. doi: 10.23638/LMCS-13(1:17)2017. URL https://doi.org/10.23638/LMCS-13(1:17)2017.

Rumyana Neykova, Laura Bocchi, and Nobuko Yoshida. Timed runtime monitoring for multiparty conversations. *Formal Aspects Comput.*, 29(5):877–910, 2017. doi: 10.1007/S00165-017-0420-8. URL https://doi.org/10.1007/s00165-017-0420-8.

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

Nicholas Ng, Nobuko Yoshida, and Kohei Honda. Multiparty session C: safe parallel programming with message optimisation. In Carlo A. Furia and Sebastian Nanz, editors, *Objects, Models, Components, Patterns - 50th International Conference, TOOLS 2012, Prague, Czech Republic, May 29-31, 2012. Proceedings*, volume 7304 of *Lecture Notes in Computer Science*, pages 202–218. Springer, 2012. doi: 10.1007/978-3-642-30561-0_15. URL https://doi.org/10.1007/978-3-642-30561-0_15.

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

Object Management Group. Unified Modeling Language (UML) Website. https://www.uml.org/. Accessed: 2025-07-10.

Catuscia Palamidessi. Comparing the expressive power of the synchronous and asynchronous pi-calculi. *Math. Struct. Comput. Sci.*, 13(5):685–719, 2003. doi: 10.1017/S0960129503004043. URL https://doi.org/10.1017/S0960129503004043.

Giuseppe De Palma, Saverio Giallorenzo, Jacopo Mauro, Matteo Trentin, and Gianluigi Zavattaro. Towards a function-as-a-service choreographic programming language: Examples and applications, 2024. URL https://arxiv.org/abs/2406.09099.

Kirstin Peters and Nobuko Yoshida. On the expressiveness of mixed choice sessions. In Valentina Castiglioni and Claudio Antares Mezzina, editors, *Proceedings Combined 29th International Workshop on Expressiveness in Concurrency and 19th Workshop on Structural Operational Semantics, EXPRESS/SOS 2022, and 19th Workshop on Structural Operational Semantics Warsaw, Poland, 12th September 2022*, volume 368 of *EPTCS*, pages 113–130, 2022. doi: 10.4204/EPTCS.368.7. URL https://doi.org/10.4204/EPTCS.368.7.

Kirstin Peters and Nobuko Yoshida. Separation and encodability in mixed choice multiparty sessions. In Pawel Sobocinski, Ugo Dal Lago, and Javier Esparza, editors, *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024, Tallinn, Estonia, July 8-11, 2024*, pages 62:1–62:15. ACM, 2024. doi: 10.1145/3661814.3662085. URL https://doi.org/10.1145/3661814.3662085.

Benjamin C. Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Math. Struct. Comput. Sci.*, 6(5):409–453, 1996. doi: 10.1017/s096012950007002x. URL https://doi.org/10.1017/s096012950007002x.

Arjen Rouvoet, Casper Bach Poulsen, Robbert Krebbers, and Eelco Visser. Intrinsically-typed definitional interpreters for linear, session-typed languages. In Jasmin Blanchette and Catalin Hritcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Pro-*

grams and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020, pages 284–298. ACM, 2020. doi: 10.1145/3372885.3373818. URL https://doi.org/10.1145/3372885.3373818.

Abhik Roychoudhury, Ankit Goel, and Bikram Sengupta. Symbolic message sequence charts. *ACM Trans. Softw. Eng. Methodol.*, 21(2):12:1–12:44, 2012. doi: 10.1145/2089116.2089122. URL https://doi.org/10.1145/2089116.2089122.

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

Alceste Scalas, Ornela Dardha, Raymond Hu, and Nobuko Yoshida. A linear decomposition of multiparty sessions for safe distributed programming. In Peter Müller, editor, *31st European Conference on Object-Oriented Programming, ECOOP 2017, June 19-23, 2017, Barcelona, Spain*, volume 74 of *LIPIcs*, pages 24:1–24:31. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi: 10.4230/LIPICS.ECOOP.2017.24. URL https://doi.org/10.4230/LIPIcs.ECOOP.2017.24.

Gan Shen, Shun Kashiwa, and Lindsey Kuper. Haschor: Functional choreographic programming for all (functional pearl). *CoRR*, abs/2303.00924, 2023. doi: 10.48550/ARXIV.2303.00924. URL https://doi.org/10.48550/arXiv.2303.00924.

Felix Stutz. Asynchronous multiparty session type implementability is decidable - lessons learned from message sequence charts. In Karim Ali and Guido Salvaneschi, editors, *37th European Conference on Object-Oriented Programming, ECOOP 2023, July 17-21, 2023, Seattle, Washington, United States*, volume 263 of *LIPIcs*, pages 32:1–32:31. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi: 10.4230/LIPICS.ECOOP.2023.32. URL https://doi.org/10.4230/LIPIcs.ECOOP.2023.32.

Felix Stutz. *Implementability of Asynchronous Communication Protocols - The Power of Choice*. PhD thesis, Kaiserslautern University of Technology, Germany, 2024a. URL `https://kluedo.ub.rptu.de/frontdoor/index/index/docId/8077`.

Felix Stutz. *Implementability of Asynchronous Communication Protocols - The Power of Choice*. doctoralthesis, Rheinland-Pfälzische Technische Universität Kaiserslautern-Landau, 2024b. URL `https://nbn-resolving.de/urn:nbn:de:hbz:386-kluedo-80778`.

Felix Stutz and Damien Zufferey. Comparing channel restrictions of communicating state machines, high-level message sequence charts, and multiparty session types. In Pierre Ganty and Dario Della Monica, editors, *Proceedings of the 13th International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2022, Madrid, Spain, September 21-23, 2022*, volume 370 of *EPTCS*, pages 194–212, 2022. doi: 10.4204/EPTCS.370.13. URL `https://doi.org/10.4204/EPTCS.370.13`.

Joseph Tassarotti, Ralf Jung, and Robert Harper. A higher-order logic for concurrent termination-preserving refinement. In Hongseok Yang, editor, *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10201 of *Lecture Notes in Computer Science*, pages 909–936. Springer, 2017. doi: 10.1007/978-3-662-54434-1_34. URL `https://doi.org/10.1007/978-3-662-54434-1_34`.

Peter Thiemann. Intrinsically-typed mechanized semantics for session types. In Ekaterina Komendantskaya, editor, *Proceedings of the 21st International Symposium on Principles and Practice of Programming Languages, PPDP 2019, Porto, Portugal, October 7-9, 2019*, pages 19:1–19:15. ACM, 2019. doi: 10.1145/3354166.3354184. URL `https://doi.org/10.1145/3354166.3354184`.

Peter Thiemann and Vasco T. Vasconcelos. Label-dependent session types. *Proc. ACM Program.*

Lang., 4(POPL):67:1–67:29, 2020. doi: 10.1145/3371135. URL https://doi.org/10.1145/3371135.

Dawit Legesse Tirore, Jesper Bengtson, and Marco Carbone. A sound and complete projection for global types. In Adam Naumowicz andL René Thiemann, editor, *14th International Conference on Interactive Theorem Proving, ITP 2023, July 31 to August 4, 2023, Białystok, Poland*, volume 268 of *LIPIcs*, pages 28:1–28:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi: 10.4230/LIPICS.ITP.2023.28. URL https://doi.org/10.4230/LIPIcs.ITP.2023.28.

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

Bernardo Toninho, Luís Caires, and Frank Pfenning. Dependent session types via intuitionistic linear type theory. In Peter Schneider-Kamp and Michael Hanus, editors, *Proceedings of the 13th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 20-22, 2011, Odense, Denmark*, pages 161–172. ACM, 2011. doi: 10.1145/2003476.2003499. URL https://doi.org/10.1145/2003476.2003499.

Bernardo Toninho, Luís Caires, and Frank Pfenning. A decade of dependent session types. In *23rd International Symposium on Principles and Practice of Declarative Programming*, PPDP 2021, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450386890. doi: 10.1145/3479394.3479398. URL https://doi.org/10.1145/3479394.3479398.

Thien Udomsrirungruang and Nobuko Yoshida. Top-down or bottom-up? complexity analyses of synchronous multiparty session types. *Proc. ACM Program. Lang.*, 9(POPL):1040–1071, 2025. doi: 10.1145/3704872. URL https://doi.org/10.1145/3704872.

International Telecommunication Union. Z.120: Message sequence chart. Technical report, International Telecommunication Union, October 1996. URL https://www.itu.int/rec/T-REC-Z.120.

Hiroshi Unno, Tachio Terauchi, Yu Gu, and Eric Koskinen. Modular primal-dual fixpoint logic solving for temporal verification. *Proc. ACM Program. Lang.*, 7(POPL):2111–2140, 2023. doi: 10.1145/3571265. URL https://doi.org/10.1145/3571265.

Martin Vassor and Nobuko Yoshida. Refinements for multiparty message-passing protocols: Specification-agnostic theory and implementation. In Jonathan Aldrich and Guido Salvaneschi, editors, *38th European Conference on Object-Oriented Programming, ECOOP 2024, September 16-20, 2024, Vienna, Austria*, volume 313 of *LIPIcs*, pages 41:1–41:29. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi: 10.4230/LIPICS.ECOOP.2024.41. URL https://doi.org/10.4230/LIPIcs.ECOOP.2024.41.

Margus Veanes and Nikolaj S. Bjørner. Symbolic automata: The toolkit. In Cormac Flanagan and Barbara König, editors, *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*, volume 7214 of *Lecture Notes in Computer Science*, pages 472–477. Springer, 2012. doi: 10.1007/978-3-642-28756-5_33. URL https://doi.org/10.1007/978-3-642-28756-5_33.

Margus Veanes, Peli de Halleux, and Nikolai Tillmann. Rex: Symbolic regular expression explorer. In *Third International Conference on Software Testing, Verification and Validation, ICST 2010, Paris, France, April 7-9, 2010*, pages 498–507. IEEE Computer Society, 2010. doi: 10.1109/ICST.2010.15. URL https://doi.org/10.1109/ICST.2010.15.

Klaus von Gleissenthall, Rami Gökhan Kici, Alexander Bakst, Deian Stefan, and Ranjit Jhala. Pretend synchrony: synchronous verification of asynchronous distributed programs. *Proc.*

ACM Program. Lang., 3(POPL):59:1–59:30, 2019. doi: 10.1145/3290372. URL https://doi.org/10.1145/3290372.

Philip Wadler. Propositions as sessions. *J. Funct. Program.*, 24(2-3):384–418, 2014. doi: 10.1017/S095679681400001X. URL https://doi.org/10.1017/S095679681400001X.

Web Services Choreography Working Group. Web services choreography description language version 1.0. W3c candidate recommendation, World Wide Web Consortium (W3C), November 2005. Available at http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/.

Nobuko Yoshida. Programming language implementations with multiparty session types. In Frank S. de Boer, Ferruccio Damiani, Reiner Hähnle, Einar Broch Johnsen, and Eduard Kamburjan, editors, *Active Object Languages: Current Research Trends*, volume 14360 of *Lecture Notes in Computer Science*, pages 147–165. Springer, 2024. doi: 10.1007/978-3-031-51060-1_6. URL https://doi.org/10.1007/978-3-031-51060-1_6.

Nobuko Yoshida, Raymond Hu, Rumyana Neykova, and Nicholas Ng. The scribble protocol language. In Martín Abadi and Alberto Lluch-Lafuente, editors, *Trustworthy Global Computing - 8th International Symposium, TGC 2013, Buenos Aires, Argentina, August 30-31, 2013, Revised Selected Papers*, volume 8358 of *Lecture Notes in Computer Science*, pages 22–41. Springer, 2013. doi: 10.1007/978-3-319-05119-2_3. URL https://doi.org/10.1007/978-3-319-05119-2_3.

Tony Nuda Zhang, Travis Hance, Manos Kapritsos, Tej Chajed, and Bryan Parno. Inductive invariants that spark joy: Using invariant taxonomies to streamline distributed protocol proofs. In Ada Gavrilovska and Douglas B. Terry, editors, *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*, pages 837–853. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/zhang-nuda.

Fangyi Zhou. *Refining Multiparty Session Types.* PhD thesis, Imperial College London, 2024.

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

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