Deciding Subtyping for Asynchronous Multiparty Sessions

Elaine Li[1(✉)], Felix Stutz[2], and Thomas Wies[1]

[1] New York University, New York, USA
efl9013@nyu.edu, wies@cs.nyu.edu
[2] Max Planck Institute for Software Systems, Kaiserslautern, Germany
fstutz@mpi-sws.org

Abstract. Multiparty session types (MSTs) are a type-based approach to verifying communication protocols, represented as global types in the framework. We present a precise subtyping relation for asynchronous MSTs with communicating state machines (CSMs) as implementation model. We address two problems: when can a local implementation safely substitute another, and when does an arbitrary CSM implement a global type? We define safety with respect to a given global type, in terms of subprotocol fidelity and deadlock freedom. Our implementation model subsumes existing work which considers local types with restricted choice. We exploit the connection between MST subtyping and refinement to formulate concise conditions that are directly checkable on the candidate implementations, and use them to show that both problems are decidable in polynomial time.

Keywords: Protocol verification · Multiparty session types · Communicating state machines · Subtyping · Refinement.

1 Introduction

Multiparty session types (MSTs) [31] are a type-based approach to verifying communication protocols. In MST frameworks, a communication protocol is expressed as a *global type*, which describes the interactions of all protocol participants from a birds-eye view. The key property of interest in MST frameworks is *implementability*, which asks whether there exists a collection of local implementations, one per protocol participant, that is deadlock-free and produces the same set of behaviors described by the global type. The latter property is known as *protocol fidelity*. Given an implementable global type, the *synthesis* problem asks to compute such a collection. To solve implementability and synthesis, MST frameworks are often equipped with 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

S. Weirich (Ed.): ESOP 2024, LNCS 14576, pp. 176–205, 2024.
https://doi.org/10.1007/978-3-031-57262-3_8

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 [39]: 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 [29, 31], other implementation models have since been investigated, including communicating session automata [21] and behavioral contracts [16]. 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, [2, 3] define safety as the successful completion of a single role in binary sessions, [36] defines safety as eventual reception and progress of all roles in multiparty sessions, and [26] 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 [9], 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.

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) [4] communicating via FIFO channels, which subsumes implementation models in prior work [20, 26, 36]. 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 [40] 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 [31], and borrows insights from recent work on a sound and complete projection operator for this class of global types [38].

Contributions. In this paper, 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 $\mathtt{p}$ be a role and let B be a safe implementation for $\mathtt{p}$ 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 polynomial time.

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 $\mathtt{q}$ that recognizes the projection of the global language onto $\mathtt{q}$'s alphabet $\Sigma_\mathtt{q}$, denoted $\mathcal{L}(\mathbf{G}_n)\Downarrow_{\Sigma_\mathtt{q}}$, 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 [23, Thm. 11].

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

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

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

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

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

As $\mathbf{G}_n$ is implementable, the subset projection [38] 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. $\mathcal{L}(\mathbf{G}_n)\Downarrow_{\Sigma_\mathtt{q}}$ for role $\mathtt{q}$ with alphabet $\Sigma_\mathtt{q}$. In this case, because $\mathcal{L}(\mathbf{G}_n)\Downarrow_{\Sigma_\mathtt{q}}$ 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 $\mu t.(\mathtt{p}\to\mathtt{q}:o.\,t+\mathtt{p}\to\mathtt{q}:b.\,0)$. The projection of the global language onto $\mathtt{q}$ limits $\mathtt{q}$ to only receiving a sequence of o messages terminated by a b message. However, an implementation for $\mathtt{q}$ can rely on $\mathtt{p}$ to send correct sequences of messages, and

(a) A (b) B

Fig. 1: Two state machines for role q

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. 1a and 1b. State machine A embodies the idea of input covariance [25] by adding receive actions, namely $\mathtt{p}\triangleleft\mathtt{q}?m$, 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 [26, 34] 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

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

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 $\mathtt{p}\triangleleft\mathtt{r}?m$ 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 $\mathtt{p}\triangleleft\mathtt{r}?m$ 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.

3 Preliminaries

We restate relevant definitions from [38].

Words. Let Σ be a finite 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 $u \cdot w = v$.

Message Alphabet. Let $\mathcal{P}$ be a set of roles and $\mathcal{V}$ be a set of messages. We define the set of *synchronous events* $\Sigma_{sync} := \{\mathtt{p} \to \mathtt{q} : m \mid \mathtt{p}, \mathtt{q} \in \mathcal{P} \text{ and } m \in \mathcal{V}\}$ where $\mathtt{p} \to \mathtt{q} : m$ denotes that message m is sent by $\mathtt{p}$ to $\mathtt{q}$ atomically. This is split for *asynchronous events*. For a role $\mathtt{p} \in \mathcal{P}$, we define the alphabet $\Sigma_{\mathtt{p},!} = \{\mathtt{p} \triangleright \mathtt{q}!m \mid \mathtt{q} \in \mathcal{P},\ m \in \mathcal{V}\}$ of *send* events and the alphabet $\Sigma_{\mathtt{p},?} = \{\mathtt{p} \triangleleft \mathtt{q}?m \mid \mathtt{q} \in \mathcal{P},\ m \in \mathcal{V}\}$ of *receive* events. The event $\mathtt{p} \triangleright \mathtt{q}!m$ denotes role $\mathtt{p}$ sending a message m to $\mathtt{q}$, and $\mathtt{p} \triangleleft \mathtt{q}?m$ denotes role $\mathtt{p}$ receiving a message m from $\mathtt{q}$. We write $\Sigma_{\mathtt{p}} = \Sigma_{\mathtt{p},!} \cup \Sigma_{\mathtt{p},?}$, $\Sigma_! = \bigcup_{\mathtt{p} \in \mathcal{P}} \Sigma_{\mathtt{p},!}$, and $\Sigma_? = \bigcup_{\mathtt{p} \in \mathcal{P}} \Sigma_{\mathtt{p},?}$. Finally, $\Sigma_{async} = \Sigma_! \cup \Sigma_?$. We say that $\mathtt{p}$ is *active* in $x \in \Sigma_{async}$ if $x \in \Sigma_{\mathtt{p}}$. For each role $\mathtt{p} \in \mathcal{P}$, we define a homomorphism $\Downarrow_{\Sigma_{\mathtt{p}}}$, where $x \Downarrow_{\Sigma_{\mathtt{p}}} = x$ if $x \in \Sigma_{\mathtt{p}}$ and ε otherwise. We fix $\mathcal{P}$ and $\mathcal{V}$ in the rest of the paper.

Global Types – Syntax. Global types for MSTs [40] are defined by the grammar:

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

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

We require each branch of a choice to be distinct: $\forall i, j \in I.\, i \neq j \Rightarrow (\mathtt{q}_i, m_i) \neq (\mathtt{q}_j, m_j)$, the sender and receiver of an event to be distinct: $\mathtt{p} \neq \mathtt{q}_i$ for each $i \in I$, and recursion to be guarded: in $\mu t.\, G$, there is at least one message between μt and each t in G. We omit $\sum$ for singleton choices. When working with a protocol described by a global type, we use $\mathbf{G}$ to refer to the top-level type, and G to refer to its subterms.

We use the extended definition of global types from [40] featuring *sender-driven choice*. This definition subsumes classical MSTs that only allow *directed choice* [31]. We focus on communication primitives and omit features like delegation or parametrization, and refer the reader to §7 for a discussion of different MST frameworks.

Global Types – Semantics. As a basis for the semantics of a global type $\mathbf{G}$, we construct a finite state machine $\mathsf{GAut}(\mathbf{G}) = (Q_{\mathbf{G}}, \Sigma_{sync}, \delta_{\mathbf{G}}, q_{0,\mathbf{G}}, F_{\mathbf{G}})$ where

- $Q_{\mathbf{G}}$ is the set of all syntactic subterms in $\mathbf{G}$ together with the term 0,
- $\delta_{\mathbf{G}}$ consists of the transitions $(\sum_{i \in I} \mathtt{p} \to \mathtt{q}_i : m_i . G_i, \mathtt{p} \to \mathtt{q}_i : m_i, G_i)$ 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 $F_{\mathbf{G}} = \{0\}$.

We define a homomorphism $\mathtt{split}$ onto the asynchronous alphabet:

$$\mathtt{split}(\mathtt{p} \to \mathtt{q} : m) := \mathtt{p} \triangleright \mathtt{q}!m.\, \mathtt{q} \triangleleft \mathtt{p}?m \ .$$

The semantics $\mathcal{L}(\mathbf{G})$ of a global type $\mathbf{G}$ is given by $\mathcal{C}^{\sim}(\mathtt{split}(\mathcal{L}(\mathsf{GAut}(\mathbf{G}))))$ where $\mathcal{C}^{\sim}$ is the closure under the indistinguishability relation $\sim$ [40]. Two events are independent if they are not related by the *happened-before* relation [33]. For instance, any two send events from distinct senders are independent. Two words are indistinguishable if one can be reordered into the other by repeatedly swapping consecutive independent events. The full definition can be found in the extended version [37].

We call a state $q_G \in Q_{\mathbf{G}}$ a *send-originating* state, denoted $q_G \in Q_{\mathbf{G},!}$ for role $\mathtt{p}$ if there exists a transition $q_G \xrightarrow{\mathtt{p} \to \mathtt{q} : m} q_{G'} \in \delta_{\mathbf{G}}$, and a *receive-originating* state, denoted $q_G \in Q_{\mathbf{G},?}$ for $\mathtt{p}$ if there exists a transition $q_G \xrightarrow{\mathtt{q} \to \mathtt{p} : m} q_{G'} \in \delta_{\mathbf{G}}$. We omit mention of role $\mathtt{p}$ when clear from context.

Communicating State Machine [4]. $\mathcal{A} = \{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ is a CSM over $\mathcal{P}$ and $\mathcal{V}$ if $A_\mathtt{p} = (Q_\mathtt{p}, \Sigma_\mathtt{p}, \delta_\mathtt{p}, q_{0,\mathtt{p}}, F_\mathtt{p})$ is a deterministic finite state machine over $\Sigma_\mathtt{p}$ for every $\mathtt{p} \in \mathcal{P}$. Let $\prod_{\mathtt{p}\in\mathcal{P}} Q_\mathtt{p}$ denote the set of global states and $\mathsf{Chan} = \{(\mathtt{p},\mathtt{q}) \mid \mathtt{p},\mathtt{q} \in \mathcal{P}, \mathtt{p} \neq \mathtt{q}\}$ 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 $\vec{s}_\mathtt{p}$ to denote the state of $\mathtt{p}$ in $\vec{s}$. The CSM transition relation, denoted $\to$, is defined as follows.

- $(\vec{s},\xi) \xrightarrow{\mathtt{p}\triangleright\mathtt{q}!m} (\vec{s}',\xi')$ if $(\vec{s}_\mathtt{p}, \mathtt{p}\triangleright\mathtt{q}!m, \vec{s}'_\mathtt{p}) \in \delta_\mathtt{p}$, $\vec{s}_\mathtt{r} = \vec{s}'_\mathtt{r}$ for every role $\mathtt{r} \neq \mathtt{p}$, $\xi'(\mathtt{p},\mathtt{q}) = \xi(\mathtt{p},\mathtt{q}) \cdot m$ and $\xi'(c) = \xi(c)$ for every other channel $c \in \mathsf{Chan}$.
- $(\vec{s},\xi) \xrightarrow{\mathtt{q}\triangleleft\mathtt{p}?m} (\vec{s}',\xi')$ if $(\vec{s}_\mathtt{q}, \mathtt{q}\triangleleft\mathtt{p}?m, \vec{s}'_\mathtt{q}) \in \delta_\mathtt{q}$, $\vec{s}_\mathtt{r} = \vec{s}'_\mathtt{r}$ for every role $\mathtt{r} \neq \mathtt{q}$, $\xi(\mathtt{p},\mathtt{q}) = m \cdot \xi'(\mathtt{p},\mathtt{q})$ and $\xi'(c) = \xi(c)$ for every other channel $c \in \mathsf{Chan}$.

In the initial configuration $(\vec{s}_0, \xi_0)$, each role's state in $\vec{s}_0$ is the initial state $q_{0,\mathtt{p}}$ of $A_\mathtt{p}$, and ξ_0 maps each channel to ε. A configuration $(\vec{s},\xi)$ is said to be *final* iff $\vec{s}_\mathtt{p}$ is final for every $\mathtt{p}$ 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 $\mathcal{L}(\mathcal{A})$ of the CSM $\mathcal{A}$ 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 CSM is *deadlock-free* if no reachable configuration is a deadlock.

Definition 3.1 (Implementability). *We say that a CSM $\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ implements a global type $\mathbf{G}$ if the following two properties hold: (i)* protocol fidelity: *$\mathcal{L}(\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}) = \mathcal{L}(\mathbf{G})$, and (ii)* deadlock freedom: *$\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ is deadlock-free. A global type $\mathbf{G}$ is* implementable *if there exists a CSM that implements it.*

One candidate implementation for global types can be computed directly from $\mathsf{GAut}(\mathbf{G})$, by removing actions unrelated to each role and determinizing the result. The following two definitions define this candidate implementation in two steps.

Definition 3.2 (Projection by Erasure [38]). *Let $\mathbf{G}$ be some global type with its state machine $\mathsf{GAut}(\mathbf{G}) = (Q_\mathbf{G}, \Sigma_{sync}, \delta_\mathbf{G}, q_{0,\mathbf{G}}, F_\mathbf{G})$. For each role $\mathtt{p} \in \mathcal{P}$, we define the state machine $\mathsf{GAut}(\mathbf{G})\!\downarrow_\mathtt{p} = (Q_\mathbf{G}, \Sigma_\mathtt{p} \uplus \{\varepsilon\}, \delta_\downarrow, q_{0,\mathbf{G}}, F_\mathbf{G})$ where $\delta_\downarrow := \{q \xrightarrow{\mathtt{split}(a)\Downarrow_{\Sigma_\mathtt{p}}} q' \mid q \xrightarrow{a} q' \in \delta_\mathbf{G}\}$. By definition of $\mathtt{split}(\text{-})$, it holds that $\mathtt{split}(a)\Downarrow_{\Sigma_\mathtt{p}} \in \Sigma_\mathtt{p} \uplus \{\varepsilon\}$.*

We determinize $\mathsf{GAut}(\mathbf{G})\!\downarrow_\mathtt{p}$ via a standard subset construction [46] to obtain a deterministic local state machine for $\mathtt{p}$. Note that the construction ensures that $Q_\mathtt{p}$ only contains subsets of $Q_\mathbf{G}$ whose states are reachable via the same traces.

Definition 3.3 (Subset Construction [38]). *Let $\mathbf{G}$ be a global type and $\mathtt{p}$ be a role. Then, the* subset construction *for $\mathtt{p}$ is defined as*

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

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

– $\delta_{\mathtt{p}} := \delta|_{Q_{\mathtt{p}} \times \Sigma_{\mathtt{p}}}$, *and*
– $F_{\mathtt{p}} := \{s \in Q_{\mathtt{p}} \mid s \cap F_{\mathbf{G}} \neq \emptyset\}$.

Li et al. [38] showed that if $\mathbf{G}$ is implementable, then $\{\!\{\mathscr{C}(\mathbf{G}, \mathtt{p})\}\!\}_{\mathtt{p} \in \mathcal{P}}$ implements $\mathbf{G}$ and satisfies the following property:

Definition 3.4. *Let $\mathbf{G}$ be a global type. We call an implementation $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p} \in \mathcal{P}}$* local language preserving *with respect to $\mathbf{G}$ if $\mathcal{L}(A_{\mathtt{p}}) = \mathcal{L}(\mathbf{G})\Downarrow_{\Sigma_{\mathtt{p}}}$ for all $\mathtt{p} \in \mathcal{P}$.*

For the remainder of the paper, we fix a global type $\mathbf{G}$ that we assume is implementable.

4 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 $\mathcal{A}$.

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

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

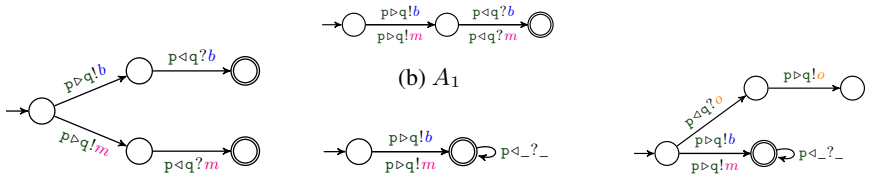
Theorem 4.1. *Let $\mathbf{G}$ be an implementable global type and $\mathcal{A}$ be a CSM. Then, the subset construction $\{\!\{\mathscr{C}(\mathbf{G}, \mathtt{p})\}\!\}_{\mathtt{p} \in \mathcal{P}}$ and $\mathcal{A}$ are equivalent if and only if C_1 is satisfied.*

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

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

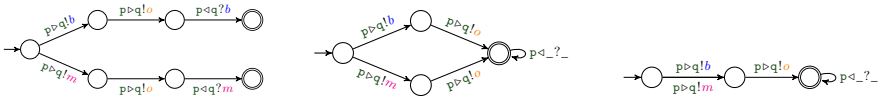
This global type is trivially implementable; the subset construction for role $\mathtt{q}$ obtained by the projection operator in [38] is depicted in Fig. 2a. Clearly, in any CSM implementing $\mathbf{G}_1$, the subset construction can be replaced with the more compact state machine A_1, shown in Fig. 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 $\mathtt{p}$ chooses to send $\mathtt{q}$ either m or b, $\mathtt{q}$ will guarantee that the correct message, i.e. the same one, is sent back to $\mathtt{p}$. Role $\mathtt{p}$'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 $\mathtt{p}$'s control states after sending with an accepting universal receive state, as shown in A_2 in Fig. 2c. Finally, we can add send transitions from unreachable states, as shown in A_3 in Fig. 2d.



(a) $\mathscr{C}(\mathbf{G}_1, \mathtt{p})$ (b) A_1 (c) A_2 (d) A_3

Fig. 2: Subset construction of $\mathbf{G}_1$ onto $\mathtt{p}$ and three alternative implementations



(a) $\mathscr{C}(\mathbf{G}_2, \mathtt{p})$ (b) A_4 (c) A_5

Fig. 3: Subset construction of $\mathbf{G}_2$ onto $\mathtt{p}$ and two alternative implementations

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

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

The subset construction from [38] yields the state machine for $\mathtt{p}$ shown in Fig. 3a.

Our reasoning above shows that A_4, depicted in Fig. 3b, is a correct alternative implementation for $\mathtt{p}$. Now observe that the pre-states of the two $\mathtt{p} \triangleright \mathtt{q}!o$ transitions can be collapsed because their continuations are identical. This yields another correct alternative implementation A_5, shown in Fig. 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 inspired by the Send and Receive Validity conditions that precisely characterize implementability for global types, given in [38]. We restate the conditions, in addition to relevant definitions, for clarity.

Definition 4.2 (Available messages [40]). *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 $get\mu(\mathbf{G})$:*

$$get\mu(0) := [\,] \qquad get\mu(t) := [\,] \qquad get\mu(\mu t.G) := [t \mapsto G] \cup get\mu(G)$$
$$get\mu(\textstyle\sum_{i \in I} \mathtt{p} \to \mathtt{q}_i : m_i.G_i) := \bigcup_{i \in I} get\mu(G_i)$$

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

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

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

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* $M^{\mathtt{p}}_{(G\ldots)}$ *for* $M^{\{\mathtt{p}\}}_{(G\ldots)}$*.*

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 following [38], but generalized from the subset construction automaton.

Definition 4.3 (Transition Origin and Destination). *Let* $\mathbf{G}$ *be a global type and let* $\delta_\downarrow$ *be the transition relation of* $\mathsf{GAut}(\mathbf{G})\downarrow_{\mathtt{p}}$*. For* $x \in \Sigma_{\mathtt{p}}$ *and* $s, s' \subseteq Q_{\mathbf{G}}$*, 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:*

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

Li et al. [38] showed that $\mathbf{G}$ is implementable if and only if the subset construction CSM $\{\!\{\mathscr{C}(\mathbf{G},\mathtt{p})\}\!\}_{\mathtt{p}\in\mathcal{P}}$ satisfies Send and Receive Validity for each $\mathscr{C}(\mathbf{G},\mathtt{p})$.

Definition 4.4 (Send Validity). $\mathscr{C}(\mathbf{G},\mathtt{p})$ *satisfies* Send Validity *iff every send transition* $s \xrightarrow{x} s' \in \delta_{\mathtt{p}}$ *is enabled in all states contained in* s*:*

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

Definition 4.5 (Receive Validity). $\mathscr{C}(\mathbf{G},\mathtt{p})$ *satisfies* Receive Validity *iff no receive transition is enabled in an alternative continuation that originates from the same source state:*

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

We wish to adapt these conditions to define C_1. However, unlike Send and Receive Validity, which are defined on special state machines, namely the subset construction for each role, the *Protocol Verification* problem asks whether arbitrary state machines implement the given $\mathbf{G}$.

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 $\mathsf{GAut}(\mathbf{G})\downarrow_{\mathtt{q}}$ on the same prefixes that reach the present state in the local state machine.

Definition 4.6 (State decoration with respect to G). *Let* $\mathtt{p} \in \mathcal{P}$ *be a role and let* $A = (Q, \Sigma_{\mathtt{p}}, s_0, \delta, F)$ *be a deterministic finite state machine for* $\mathtt{p}$*. Let* $\mathsf{GAut}(\mathbf{G})\downarrow_{\mathtt{p}} = (Q_{\mathbf{G}}, \Sigma_{\mathtt{p}} \uplus \{\varepsilon\}, \delta_{\downarrow}, q_{0,\mathbf{G}}, F_{\mathbf{G}})$ *be* $\mathtt{p}$*'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* $\mathsf{GAut}(\mathbf{G})\downarrow_{\mathtt{p}}$ *such that:*

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

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

Remark 4.7. 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 $s \in Q_{\mathtt{p}}$ where $Q_{\mathtt{p}}$ is the set of states of $\mathscr{C}(\mathbf{G}, \mathtt{p})$, $d(s) = s$.

We are now equipped to present C_1.

Definition 4.8 (C_1). *Let* $\mathbf{G}$ *be a global type and* $\mathcal{A}$ *be a CSM.* C_1 *is satisfied when for all* $\mathtt{p} \in \mathcal{P}$*, with* $A_{\mathtt{p}} = (Q_{\mathtt{p}}, \Sigma_{\mathtt{p}}, \delta_{\mathtt{p}}, s_{0,\mathtt{p}}, F_{\mathtt{p}})$ *denoting the state machine for* $\mathtt{p}$ *in* $\mathcal{A}$*, the following conditions hold:*

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

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

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

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

We want to show the following equivalence to prove Theorem 4.1:

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

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 $\mathtt{p}$ satisfies *Local Language Inclusion* if it satisfies $\mathcal{L}(\mathbf{G})\Downarrow_{\Sigma_{\mathtt{p}}} \subseteq \mathcal{L}(A)$. The following lemma, proven in the extended version [37], establishes that *Local Language Inclusion* follows from *Transition Exhaustivity* and *Final State Validity*.

Lemma 4.9. *Let $A_{\mathtt{p}} = (Q_{\mathtt{p}}, \Sigma_{\mathtt{p}}, \delta_{\mathtt{p}}, s_{0,\mathtt{p}}, F_{\mathtt{p}})$ denote the state machine for $\mathtt{p}$ in $\mathcal{A}$. Then,* Transition Exhaustivity *and* Final State Validity *imply $\mathcal{L}(\mathbf{G})\Downarrow_{\Sigma_{\mathtt{p}}} \subseteq \mathcal{L}(A_{\mathtt{p}})$.*

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 recall two definitions from [38] used in the soundness proof.

Definition 4.10 (Intersection sets). *Let $\mathbf{G}$ be a global type and $\mathsf{GAut}(\mathbf{G})$ be the corresponding state machine. Let $\mathtt{p}$ be a role and $w \in \Sigma^*_{async}$ be a word. We define the set of possible runs $\mathrm{R}^{\mathbf{G}}_{\mathtt{p}}(w)$ as all maximal runs of $\mathsf{GAut}(\mathbf{G})$ that are consistent with $\mathtt{p}$'s local view of w:*

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

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

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

Definition 4.11 (Unique splitting of a possible run). *Let $\mathbf{G}$ be a global type, $\mathtt{p}$ a role, and $w \in \Sigma^*_{async}$ a word. Let ρ be a possible run in $\mathrm{R}^{\mathbf{G}}_{\mathtt{p}}(w)$. We define the longest prefix of ρ matching w:*

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

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 $\mathtt{p}$ matching w. We omit the role $\mathtt{p}$ when obvious from context. This splitting is always unique because the maximal prefix of any $\rho \in \mathrm{R}^{\mathbf{G}}_{\mathtt{p}}(w)$ matching w is unique.

Lemma 4.12 (Soundness of C_1). *C_1 implies that $\mathcal{A}$ and $\{\!\{\mathscr{C}(\mathbf{G}, \mathtt{p})\}\!\}_{\mathtt{p}\in\mathcal{P}}$ are equivalent.*

Proof. The proof that C_1 implies $\{\!\{\mathscr{C}(\mathbf{G}, \mathtt{p})\}\!\}_{\mathtt{p}\in\mathcal{P}}$ refines $\mathcal{A}$ depends only on *Local Language Inclusion* and can be straightforwardly adapted from [38, Lemma 4.4]. We instead focus on showing that C_1 implies $\mathcal{A}$ refines $\{\!\{\mathscr{C}(\mathbf{G}, \mathtt{p})\}\!\}_{\mathtt{p}\in\mathcal{P}}$, which depends on the other two conditions in C_1. First, we prove that any trace in $\mathcal{A}$ is a trace in $\{\!\{\mathscr{C}(\mathbf{G}, \mathtt{p})\}\!\}_{\mathtt{p}\in\mathcal{P}}$:

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

We prove the claim by induction for all finite w. The infinite case follows from the finite case because $\{\!\{\mathscr{C}(\mathbf{G}, \mathtt{p})\}\!\}_{\mathtt{p}\in\mathcal{P}}$ is deterministic and all prefixes of w are traces of $\mathcal{A}$ and, hence, of $\{\!\{\mathscr{C}(\mathbf{G}, \mathtt{p})\}\!\}_{\mathtt{p}\in\mathcal{P}}$. 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 $x \in \Sigma_{async}$ such that wx is a trace of $\mathcal{A}$. We want to show that wx is also a trace of $\{\!|\mathscr{C}(\mathbf{G},\mathtt{p})|\!\}_{\mathtt{p}\in\mathcal{P}}$.

From the induction hypothesis, we know that w is a trace of $\{\!|\mathscr{C}(\mathbf{G},\mathtt{p})|\!\}_{\mathtt{p}\in\mathcal{P}}$. 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 $\{\!|\mathscr{C}(\mathbf{G},\mathtt{p})|\!\}_{\mathtt{p}\in\mathcal{P}}$ configuration reached on w.

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

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

Since $\{\!|\mathscr{C}(\mathbf{G},\mathtt{p})|\!\}_{\mathtt{p}\in\mathcal{P}}$ implements $\mathbf{G}$, all finite traces of $\{\!|\mathscr{C}(\mathbf{G},\mathtt{p})|\!\}_{\mathtt{p}\in\mathcal{P}}$ are prefixes of $\mathcal{L}(\mathbf{G})$. In other words, $w \in \mathrm{pref}(\mathcal{L}(\mathbf{G}))$. Let ρ be a run such that $\rho \in I(w)$; such a run must exist from [38, Lemma 6.3]. Let $\alpha \cdot G \xrightarrow{l} G' \cdot \beta$ be the unique splitting of ρ for $\mathtt{q}$ 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 $x \in \Sigma_{\mathtt{q},!}$. Let $x = \mathtt{q}\triangleright\mathtt{r}!m$. By assumption, there exists $s \xrightarrow{\mathtt{q}\triangleright\mathtt{r}!m} s'$ in $A_{\mathtt{q}}$. We instantiate *Send Decoration Validity* from C_1 with $\mathtt{q}$ and this transition to obtain:

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

  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 $t \xrightarrow{\mathtt{q}\triangleright\mathtt{r}!m} t'$ is a transition in $\mathscr{C}(\mathbf{G},\mathtt{p})$ follows immediately from the definition of $\mathscr{C}(\mathbf{G},\mathtt{q})$'s transition relation.
- Case $x \in \Sigma_{\mathtt{q},?}$. Let $x = \mathtt{q}\triangleleft\mathtt{r}?m$.

  From the fact that ρ is a maximal run in $\mathbf{G}$ with unique splitting $\alpha \cdot G \xrightarrow{l} G' \cdot \beta$ for $\mathtt{q}$ matching w, it holds that $w{\Downarrow_{\Sigma_{\mathtt{q}}}} \cdot \mathtt{split}(l){\Downarrow_{\Sigma_{\mathtt{q}}}} \in \mathrm{pref}(\mathcal{L}(\mathbf{G})){\Downarrow_{\Sigma_{\mathtt{q}}}}$. From [38, Lemma 4.3], $\mathcal{L}(\mathbf{G}){\Downarrow_{\Sigma_{\mathtt{q}}}} = \mathcal{L}(\mathscr{C}(\mathbf{G},\mathtt{q}))$. Therefore, there exists a t'' such that $t \xrightarrow{\mathtt{split}(l){\Downarrow_{\Sigma_{\mathtt{q}}}}} t''$ is a transition in $\mathscr{C}(\mathbf{G},\mathtt{q})$. From *Transition Exhaustivity*, there likewise exists an s'' such that $s \xrightarrow{\mathtt{split}(l){\Downarrow_{\Sigma_{\mathtt{q}}}}} s''$ is a transition in $A_{\mathtt{q}}$.
  We now proceed by showing that it must be the case that $\mathtt{split}(l){\Downarrow_{\Sigma_{\mathtt{q}}}} = x$. The reasoning closely follows that in [38, 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 extended version [37].

  Lemma 4.13. *Let $\mathcal{A}$ be a CSM, $\mathtt{q}$ be a role, and w, wx be traces of $\mathcal{A}$ such that $x = \mathtt{q}\triangleleft\mathtt{r}?m$. Let s be the state of $\mathtt{q}$'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 $\mathtt{p} \in \mathcal{P}$. $w{\Downarrow_{\Sigma_{\mathtt{p}}}} \leq \mathtt{split}(\mathtt{trace}(\rho)){\Downarrow_{\Sigma_{\mathtt{p}}}}$. Let $\alpha \cdot G \xrightarrow{l} G' \cdot \beta$ be the unique splitting of ρ for $\mathtt{q}$ matching w. If $\mathtt{r}\triangleright\mathtt{q}!m \notin M^{\mathtt{q}}_{(G'\ldots)}$, then $x = \mathtt{split}(l){\Downarrow_{\Sigma_{\mathtt{q}}}}$.*

We wish to apply Lemma 4.13 with ρ to conclude that $\texttt{split}(l)\Downarrow_{\Sigma_\texttt{q}} = x$. We satisfy the assumption that $\texttt{r}\triangleright\texttt{q}!m \notin M^\texttt{q}_{(G'\ldots)}$ by instantiating *Receive Decoration Validity* with $s \xrightarrow{\texttt{q}\triangleleft\texttt{r}?m} s'$, $s \xrightarrow{\texttt{split}(l)\Downarrow_{\Sigma_\texttt{q}}} s''$ and G'. The fact that $G' \in \text{tr-dest}(d(s) \xrightarrow{\texttt{split}(l)\Downarrow_{\Sigma_\texttt{q}}} d(s''))$ 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 4.6). Thus, we conclude from $\texttt{split}(l)\Downarrow_{\Sigma_\texttt{q}} = x$ that there exists a transition $t \xrightarrow{x} t''$ in $\mathscr{C}(\mathbf{G}, \texttt{q})$.

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

Claim 2: $\forall\, w \in \Sigma^*_{async}$. w is terminated in $\mathcal{A} \implies w$ is terminated in $\{\!\{\mathscr{C}(\mathbf{G}, \texttt{p})\}\!\}_{\texttt{p}\in\mathcal{P}}$ 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 $\{\!\{\mathscr{C}(\mathbf{G}, \texttt{p})\}\!\}_{\texttt{p}\in\mathcal{P}}$. Let ξ be the channel configuration uniquely determined by w. Let the $\{\!\{\mathscr{C}(\mathbf{G}, \texttt{p})\}\!\}_{\texttt{p}\in\mathcal{P}}$ 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 $\{\!\{\mathscr{C}(\mathbf{G}, \texttt{p})\}\!\}_{\texttt{p}\in\mathcal{P}}$, assume by contradiction that w is not terminated in $\{\!\{\mathscr{C}(\mathbf{G}, \texttt{p})\}\!\}_{\texttt{p}\in\mathcal{P}}$. Because $\{\!\{\mathscr{C}(\mathbf{G}, \texttt{p})\}\!\}_{\texttt{p}\in\mathcal{P}}$ is deadlock-free, there must exist a role that can take a step in $\{\!\{\mathscr{C}(\mathbf{G}, \texttt{p})\}\!\}_{\texttt{p}\in\mathcal{P}}$. Let $\texttt{q}$ be this role, and let x be the transition that is enabled from $\vec{t}_\texttt{q}$. From *Local Language Inclusion* and the fact that $\{\!\{\mathscr{C}(\mathbf{G}, \texttt{p})\}\!\}_{\texttt{p}\in\mathcal{P}}$ is deadlock-free, it holds that x is also enabled from $\vec{s}_\texttt{q}$. 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 $\{\!\{\mathscr{C}(\mathbf{G}, \texttt{p})\}\!\}_{\texttt{p}\in\mathcal{P}}$. From the fact that $\{\!\{\mathscr{C}(\mathbf{G}, \texttt{p})\}\!\}_{\texttt{p}\in\mathcal{P}}$ is deadlock-free, w is maximal in $\{\!\{\mathscr{C}(\mathbf{G}, \texttt{p})\}\!\}_{\texttt{p}\in\mathcal{P}}$: 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 4.14 (Completeness of C_1). *If $\mathcal{A}$ and $\{\!\{\mathscr{C}(\mathbf{G}, \texttt{p})\}\!\}_{\texttt{p}\in\mathcal{P}}$ 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 $\{\!\{\mathscr{C}(\mathbf{G}, \texttt{p})\}\!\}_{\texttt{p}\in\mathcal{P}}$ 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 extended version [37] for the full proofs.

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

Lemma 4.15. *If $\mathcal{A}$ violates* Transition Exhaustivity *or* Final State Validity*, then it does not hold that $\{\!\{\mathscr{C}(\mathbf{G}, \texttt{p})\}\!\}_{\texttt{p}\in\mathcal{P}}$ 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 4.16. *If $\mathcal{A}$ violates* Send Decoration Validity *or* Receive Decoration Validity*, then it does not hold that $\mathcal{A}$ and $\{\!\{\mathscr{C}(\mathbf{G}, \texttt{p})\}\!\}_{\texttt{p}\in\mathcal{P}}$ are equivalent.*

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

(b) State machine $B'_{\mathtt{q}}$

Fig. 4: CSM violating subprotocol fidelity with respect to $\mathbf{G}_{loop}$

5 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 5.1 (Protocol refinement with respect to G). *We say that a CSM* $\{\!|A_{\mathtt{p}}|\!\}_{\mathtt{p}\in\mathcal{P}}$ *refines a CSM* $\{\!|B_{\mathtt{p}}|\!\}_{\mathtt{p}\in\mathcal{P}}$ *with respect to a global type* $\mathbf{G}$ *if the following properties hold: (i)* subprotocol fidelity: $\exists S \subseteq \mathcal{L}(\mathsf{GAut}(\mathbf{G}))$. $\mathcal{L}(\{\!|A_{\mathtt{p}}|\!\}_{\mathtt{p}\in\mathcal{P}}) = \mathcal{C}^{\sim}(\mathtt{split}(S))$, *(ii)* language inclusion: $\mathcal{L}(\{\!|A_{\mathtt{p}}|\!\}_{\mathtt{p}\in\mathcal{P}}) \subseteq \mathcal{L}(\{\!|B_{\mathtt{p}}|\!\}_{\mathtt{p}\in\mathcal{P}})$, *and (iii)* deadlock freedom: $\{\!|A_{\mathtt{p}}|\!\}_{\mathtt{p}\in\mathcal{P}}$ *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 $\mathtt{p}$ and $B'_{\mathtt{q}}$, depicted in Fig. 4. This CSM recognizes only words of the form $(\mathtt{p}\triangleright\mathtt{q}!m)^{\omega}$. It is nonetheless considered to refine the global type $\mathbf{G}_{loop} := \mu t.\ \mathtt{p} \to \mathtt{q} : m.\ t$ according to the standard notion of refinement, despite the fact that $\mathtt{p}$'s messages are never received by $\mathtt{q}$. 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 $\sim$.

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 $\{\!|\mathscr{C}(\mathbf{G}, \mathtt{p})|\!\}_{\mathtt{p}\in\mathcal{P}}$ is an implementation for $\mathbf{G}$, we say that a CSM $\{\!|A_{\mathtt{p}}|\!\}_{\mathtt{p}\in\mathcal{P}}$ refines $\mathbf{G}$ if it refines $\{\!|\mathscr{C}(\mathbf{G}, \mathtt{p})|\!\}_{\mathtt{p}\in\mathcal{P}}$.

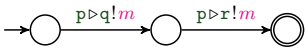
We motivate our formulation of the *Protocol Refinement* problem by posing the following variation of *Protocol Verification*, which we call *Monolithic Protocol Refinement*:

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

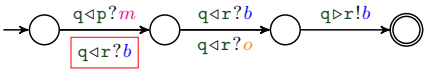
This variation asks for a condition, C'_1, that satisfies the equivalence:

$$C'_1 \Leftrightarrow \mathcal{A} \text{ refines } \{\!|\mathscr{C}(\mathbf{G}, \mathtt{p})|\!\}_{\mathtt{p}\in\mathcal{P}}.$$

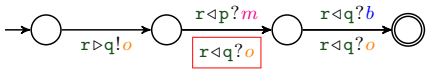
Clearly, C_1 is still a sound candidate as equivalence of two CSMs implies bidirectional protocol refinement. It is instructive to analyze why the completeness arguments for C_1 fail. Recall that the completeness proofs for *Send Decoration Validity*



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



(b) State machine $A'_{\mathtt{q}}$



(c) State machine $A'_{\mathtt{r}}$

Fig. 5: Subset construction for p and two state machines for q and r for $\mathbf{G}'$

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 $\{\!\{\mathscr{C}(\mathbf{G}, \mathtt{p})\}\!\}_{\mathtt{p}\in\mathcal{P}}$, and the assumption that $\{\!\{\mathscr{C}(\mathbf{G}, \mathtt{p})\}\!\}_{\mathtt{p}\in\mathcal{P}}$ refines $\mathcal{A}$.

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

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

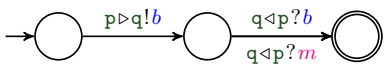
$$\mathbf{G}' := \mathtt{p}\to\mathtt{q}:m.\ +\ \begin{cases} \mathtt{r}\to\mathtt{q}:b.\,\mathtt{p}\to\mathtt{r}:m.\ +\ \begin{cases}\mathtt{q}\to\mathtt{r}:b.\,0\\ \mathtt{q}\to\mathtt{r}:o.\,0\end{cases}\\ \mathtt{r}\to\mathtt{q}:o.\,\mathtt{p}\to\mathtt{r}:m.\ +\ \begin{cases}\mathtt{q}\to\mathtt{r}:b.\,0\\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

Let the CSM $\mathcal{A}'$ consist of the subset construction automaton for p, and the state machines $A'_{\mathtt{q}}$ and $A'_{\mathtt{r}}$, given in Figs. 5b and 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 $\mathbf{G}$ 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.

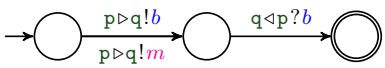
This example shows that any condition C'_1 that is compositional must sacrifice completeness. In fact, deciding whether an arbitrary CSM $\mathcal{A}$ refines the subset construction $\{\!\{\mathscr{C}(\mathbf{G}, \mathtt{p})\}\!\}_{\mathtt{p}\in\mathcal{P}}$ for some global type $\mathbf{G}$ can be shown to be PSPACE-hard via a reduction from the deadlock-freedom problem for 1-safe Petri nets [24]. We refer the reader to the extended version [37] for the full construction.

Lemma 5.2. *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 $\mathbf{G}$. We use $\mathcal{A}[\cdot]_{\mathtt{p}}$ to denote a CSM context with a hole for role $\mathtt{p} \in \mathcal{P}$, and $\mathcal{A}[A]_{\mathtt{p}}$ 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 $\mathscr{C}(\mathbf{G}, \mathtt{p})$.



(a) Removing sends



(b) Removing receives

Fig. 6: Two candidate implementations for p

Definition 5.3 (Well-behaved CSM contexts with respect to G). *Let $\mathcal{A}[\cdot]_{\mathtt{p}}$ be a CSM context. We say that $\mathcal{A}[\cdot]_{\mathtt{p}}$ is well-behaved with respect to $\mathbf{G}$ if $\mathcal{A}[\mathscr{C}(\mathbf{G},\mathtt{p})]_{\mathtt{p}}$ refines $\mathbf{G}$. We omit $\mathbf{G}$ when clear from context.*

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

Theorem 5.4. *Let $\mathbf{G}$ be an implementable global type, $\mathtt{p}$ be a role, and A, B be state machines for role $\mathtt{p}$ such that for all well-behaved contexts $\mathcal{A}[\cdot]_{\mathtt{p}}$, $\mathcal{A}[B]_{\mathtt{p}}$ refines $\mathbf{G}$. Then, for all well-behaved contexts $\mathcal{A}[\cdot]_{\mathtt{p}}$, $\mathcal{A}[A]_{\mathtt{p}}$ refines $\mathcal{A}[B]_{\mathtt{p}}$ if and only if C_2 is satisfied.*

5.1 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 $\mathtt{p}$. That is, we present C_2' that satisfies the following equivalence:

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

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, $\mathbf{G}_1$:

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

Consider the candidate state machine for role p given in Fig. 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. 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 [8, 20, 26]: 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 5.5 (C_2'). *Let $\mathtt{p} \in \mathcal{P}$ be a role and let $A = (Q, \Sigma_\mathtt{p}, s_0, \delta, F)$ be a state machine for $\mathtt{p}$. 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:*
  $\forall s \in Q.\, \exists G \in Q_{\mathbf{G},!}.\, G \in d(t) \implies \exists x \in \Sigma_{\mathtt{p},!},\ s' \in Q.\, s \xrightarrow{x} s' \in \delta.$
- Receive Exhaustivity*: every receive transition that is enabled in some global state decorating s must be an outgoing transition from s:*
  $\forall s \in Q.\, \forall G \xrightarrow{x}{}^{*}\, G' \in \delta_{\downarrow}.\, G \in d(s) \wedge x \in \Sigma_{\mathtt{p},?} \implies \exists s' \in Q.\, s \xrightarrow{x} s' \in \delta.$

We want to show the following equivalence:

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

We first prove the soundness of C_2'.

Lemma 5.6 (Soundness of C_2'). *If C_2' holds, then for all well-behaved contexts $\mathcal{A}[\cdot]_\mathtt{p}$, $\mathcal{A}[A]_\mathtt{p}$ refines $\mathcal{A}[\mathscr{C}(\mathbf{G}, \mathtt{p})]_\mathtt{p}$.*

Proof. Let $\mathcal{A}[\cdot]_\mathtt{p}$ be a well-behaved context with respect to $\mathbf{G}$. Like before, we first prove that any trace in $\mathcal{A}[A]_\mathtt{p}$ is a trace in $\mathcal{A}[\mathscr{C}(\mathbf{G}, \mathtt{p})]_\mathtt{p}$.
Claim 1: $\forall\, w \in \Sigma^{\infty}_{async}.\ w$ is a trace in $\mathcal{A}[A]_\mathtt{p} \implies w$ is a trace in $\mathcal{A}[\mathscr{C}(\mathbf{G}, \mathtt{p})]_\mathtt{p}$.

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 $\mathtt{p}$. Correspondingly, $\mathcal{A}[A]_\mathtt{p}$ only differs from $\mathcal{A}[\mathscr{C}(\mathbf{G}, \mathtt{p})]_\mathtt{p}$ in $\mathtt{p}$'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 $x \in \Sigma_\mathtt{q}$ where $\mathtt{q} \neq \mathtt{p}$, the induction hypothesis is trivially re-established by the fact that $\mathtt{q}$'s state machine is identical in both CSMs. In the case that $x \in \Sigma_\mathtt{p}$, we proceed to reason that x can also be performed by $\mathscr{C}(\mathbf{G}, \mathtt{p})$ 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 $\mathtt{p}$ matching w and the fact that $G \in s$, there must exist a transition $s \xrightarrow{\mathtt{split}(l)\Downarrow_{\Sigma_\mathtt{p}}} s''$ in $\mathtt{p}$'s state machine. Lemma 4.13 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 $\mathtt{split}(l)\Downarrow_{\Sigma_\mathtt{p}} \in \Sigma_{\mathtt{p},?}$. Since A is permitted to remove send

events, if $\mathtt{split}(l)\Downarrow_{\Sigma_\mathtt{p}} \in \Sigma_{\mathtt{p},!}$, the transition $s \xrightarrow{\mathtt{split}(l)\Downarrow_{\Sigma_\mathtt{p}}} s''$ 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 $\mathtt{p}$ makes G a send-originating global state in $\mathtt{p}$'s projection by erasure automaton. *Send Preservation* thus guarantees that there exists a transition $s \xrightarrow{x'} s'''$ in A such that $x' \in \Sigma_{\mathtt{p},!}$. 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 $\mathtt{p}$ matching w and $\mathtt{split}(l')\Downarrow_{\Sigma_\mathtt{p}} = x'$. We satisfy the assumption that $\mathtt{r}\triangleright\mathtt{p}!m \notin M^{\mathtt{p}}_{(G''\ldots)}$ by instantiating *Receive Decoration Validity* with $\mathtt{p}$, $s \xrightarrow{x} s'$, $s \xrightarrow{\mathtt{split}(l')\Downarrow_{\Sigma_\mathtt{p}}} s''$ and G''. The fact that $G'' \in \text{tr-dest}(d_{\mathbf{G}}(s) \xrightarrow{\mathtt{split}(l')\Downarrow_{\Sigma_\mathtt{p}}} d_{\mathbf{G}}(s''))$ follows from the fact that $\alpha \cdot G \xrightarrow{l'} G'' \cdot \beta'$ is a run in $\mathbf{G}$ and Definition 4.6. Instantiating Lemma 4.13 with ρ', we obtain $\mathtt{split}(l')\Downarrow_{\Sigma_\mathtt{p}} = x$, which is a contradiction: x is a receive event and $\mathtt{split}(l')\Downarrow_{\Sigma_\mathtt{p}}$ is a send event. Thus, it cannot be the case that $\mathtt{split}(l')\Downarrow_{\Sigma_\mathtt{p}} \in \Sigma_{\mathtt{p},!}$.

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

The following claim completes our soundness proof:

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

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 $\mathcal{A}[A]_\mathtt{p}$. By Claim 1, it holds that w is a trace in $\mathcal{A}[\mathscr{C}(\mathbf{G},\mathtt{p})]_\mathtt{p}$. Let ξ be the channel configuration uniquely determined by w. Let $(\vec{s},\xi)$ be the $\mathcal{A}[\mathscr{C}(\mathbf{G},\mathtt{p})]_\mathtt{p}$ configuration reached on w, and let $(\vec{t},\xi)$ be the $\mathcal{A}[A]_\mathtt{p}$ configuration reached on w. To see that w is terminated in $\mathcal{A}[\mathscr{C}(\mathbf{G},\mathtt{p})]_\mathtt{p}$, suppose by contradiction that w is not terminated in $\mathcal{A}[\mathscr{C}(\mathbf{G},\mathtt{p})]_\mathtt{p}$. Because $\mathcal{A}[\mathscr{C}(\mathbf{G},\mathtt{p})]_\mathtt{p}$ is deadlock-free, and because the state machines for all non-$\mathtt{p}$ roles are identical between the two CSMs, it must be the case that $\mathtt{p}$ witnesses the non-termination of w, in other words, $\mathscr{C}(\mathbf{G},\mathtt{p})$ can take a transition that A cannot. Let $\vec{s}_\mathtt{p} \xrightarrow{x} s'$ be the transition that $\mathtt{p}$ can take from $\vec{s}_\mathtt{p}$. Let G be a state in $\vec{s}_\mathtt{p}$; 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 $w\Downarrow_{\Sigma_\mathtt{p}}$. By the fact that w is a trace of $\mathcal{A}[A]_\mathtt{p}$, it holds that s_0 reaches $\vec{s}_\mathtt{p}$ on $w\Downarrow_{\Sigma_\mathtt{p}}$ in A. By the definition of state decoration, $G \in d(\vec{t}_\mathtt{p})$.

- 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 $\vec{t}_\mathtt{p}$ is one, there exists a transition $\vec{t}_\mathtt{p} \xrightarrow{x'} t'$ such that $x' \in \Sigma_{\mathtt{p},!}$. Because send transitions in a CSM are always enabled, role $\mathtt{p}$ can take this transition in $\mathcal{A}[A]_\mathtt{p}$. We reach a contradiction to the fact that w is terminated in $\mathcal{A}[A]_\mathtt{p}$.
- 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 $d(\vec{t}_\mathtt{p})$ must also originate from $\vec{t}_\mathtt{p}$. Therefore, there must exist t' such that $\vec{t}_\mathtt{p} \xrightarrow{x} t'$ is a transition in $B'_\mathtt{p}$. Because the channel configuration is identical in both CSMs, role $\mathtt{p}$ can

take this transition in $\mathcal{A}[A]_{\mathtt{p}}$. We again reach a contradiction to the fact that w is terminated in $\mathcal{A}[A]_{\mathtt{p}}$.

To see that w is maximal in $\mathcal{A}[A]_{\mathtt{p}}$, observe that for all roles $\mathtt{q} \neq \mathtt{p}$, $\vec{s}_{\mathtt{q}} = \vec{t}_{\mathtt{q}}$. Thus, it remains to show that $\vec{t}_{\mathtt{p}}$ is a final state in A. Because $\vec{s}_{\mathtt{p}}$ is a final state, by the definition of the subset construction there exists a global state $G \in \vec{s}_{\mathtt{p}}$ such that the projection erasure automaton reaches G on $w{\Downarrow_{\Sigma_{\mathtt{p}}}}$ and G is a final state. Because A reaches $\vec{t}_{\mathtt{p}}$ on $w{\Downarrow_{\Sigma_{\mathtt{p}}}}$, by Definition 4.6 it holds that $G \in d(\vec{t}_{\mathtt{p}})$. By *Final State Validity*, it holds that $\vec{t}_{\mathtt{p}}$ is a final state in A. This concludes our proof that any terminated trace in $\mathcal{A}[A]_{\mathtt{p}}$ is also a terminated trace in $\mathcal{A}[\mathscr{C}(\mathbf{G}, \mathtt{p})]_{\mathtt{p}}$, and is maximal in $\mathcal{A}[A]_{\mathtt{p}}$.

Together, Claim 1 and 2 establish that $\mathcal{A}[A]_{\mathtt{p}}$ satisfies language inclusion (Item ii) and deadlock freedom (Item iii). It remains to show that $\mathcal{A}[A]_{\mathtt{p}}$ satisfies subprotocol fidelity (Item i). This follows immediately from [40, Lemma 22], which states that all CSM languages are closed under the indistinguishability relation $\sim$. □

Lemma 5.7 (Completeness of C_2'). *If for all well-behaved contexts $\mathcal{A}[\cdot]_{\mathtt{p}}$, $\mathcal{A}[A]_{\mathtt{p}}$ refines $\mathcal{A}[\mathscr{C}(\mathbf{G}, \mathtt{p})]_{\mathtt{p}}$, 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 $\mathcal{A}[\cdot]_{\mathtt{p}}$ such that $\mathcal{A}[A]_{\mathtt{p}}$ does not protocol-refine $\mathcal{A}[\mathscr{C}(\mathbf{G}, \mathtt{p})]_{\mathtt{p}}$.

We first turn our attention to finding a well-behaved witness context $\mathcal{A}[\cdot]_{\mathtt{p}}$ 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 $\mathscr{C}(\mathbf{G})[\cdot]_{\mathtt{p}}$ and note that it is trivially well-behaved because $\mathscr{C}(\mathbf{G})[\mathscr{C}(\mathbf{G}, \mathtt{p})]_{\mathtt{p}} = \{\!\{\mathscr{C}(\mathbf{G}, \mathtt{p})\}\!\}_{\mathtt{p} \in \mathcal{P}}$.

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 $\{\!\{\mathscr{C}(\mathbf{G}, \mathtt{p})\}\!\}_{\mathtt{p} \in \mathcal{P}}$ 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 similarly use the fact that $\{\!\{\mathscr{C}(\mathbf{G}, \mathtt{p})\}\!\}_{\mathtt{p} \in \mathcal{P}}$ refines $\mathscr{C}(\mathbf{G})[A]_{\mathtt{p}}$. Instead, we must separately establish that every state with a non-empty decoration set can be reached on a trace shared by both $\mathscr{C}(\mathbf{G})[A]_{\mathtt{p}}$ and $\{\!\{\mathscr{C}(\mathbf{G}, \mathtt{p})\}\!\}_{\mathtt{p} \in \mathcal{P}}$. The following lemma achieves this:

Lemma 5.8. *Let A be a state machine for $\mathtt{p}$ and s be a state in A. Let $G \in d(s)$, and let $u \in \Sigma_{\mathtt{p}}^*$ 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 $\mathtt{split}(\mathtt{trace}(\alpha \cdot G)){\Downarrow_{\Sigma_{\mathtt{p}}}} = u$, $\mathtt{split}(\mathtt{trace}(\alpha \cdot G))$ is a trace in $\mathscr{C}(\mathbf{G})[A]_{\mathtt{p}}$ and in the CSM configuration reached on $\mathtt{split}(\mathtt{trace}(\alpha \cdot G))$, A is in state s.*

With Lemma 5.8 replacing the assumption that $\{\!\{\mathscr{C}(\mathbf{G}, \mathtt{p})\}\!\}_{\mathtt{p} \in \mathcal{P}}$ refines $\mathscr{C}(\mathbf{G})[A]_{\mathtt{p}}$, we can reuse the construction in Lemma 4.16 to obtain a word that is a trace in $\mathscr{C}(\mathbf{G})[A]_{\mathtt{p}}$ but not a trace in $\{\!\{\mathscr{C}(\mathbf{G}, \mathtt{p})\}\!\}_{\mathtt{p} \in \mathcal{P}}$, thus evidencing the necessity of *Send Decoration Validity* and *Receive Decoration Validity*. The proof of Lemma 5.9 proceeds identically to that of Lemma 4.16 and is thus omitted.

Lemma 5.9. *If A violates* Send Decoration Validity *or* Receive Decoration Validity*, then it does not hold that for all well-behaved contexts $\mathcal{A}[\cdot]_{\mathtt{p}}$, $\mathcal{A}[A]_{\mathtt{p}}$ refines $\mathscr{C}(\mathbf{G})[A]_{\mathtt{p}}$.*

We also use Lemma 5.8 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 5.8 and the violation of *Send Preservation*. To show the necessity of *Send Preservation*, we consider the largest extension v of u in $\mathscr{C}(\mathbf{G})[A]_{\mathtt{p}}$. In the case that u is terminated in $\mathscr{C}(\mathbf{G})[A]_{\mathtt{p}}$, 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 $\alpha \cdot G \xrightarrow{\mathtt{p}\to\mathtt{q}:m} G' \cdot \beta$ such that $\mathtt{split}(\mathtt{trace}(\alpha \cdot G \xrightarrow{\mathtt{p}\to\mathtt{q}:m} G' \cdot \beta)\Downarrow_{\Sigma_{\mathtt{p}}} = v\Downarrow_{\Sigma_{\mathtt{p}}}$. By subprotocol fidelity, $\mathtt{split}(\mathtt{trace}(\alpha \cdot G \xrightarrow{\mathtt{p}\to\mathtt{q}:m} G' \cdot \beta))$ is a trace in $\mathscr{C}(\mathbf{G})[A]_{\mathtt{p}}$. Consequently, $\mathtt{split}(\mathtt{trace}(\alpha \cdot G \xrightarrow{\mathtt{p}\to\mathtt{q}:m} G' \cdot \beta))\Downarrow_{\Sigma_{\mathtt{p}}}$ is a prefix in A. We find a contradiction from the fact that A is deterministic and there is no outgoing transition labeled $\mathtt{p}\triangleright\mathtt{q}!m$ 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 5.8 with this final state and show that u evidences a deadlock.

Lemma 5.10. *If A violates* Send Preservation*,* Receive Exhaustivity *or* Final State Validity*, then it does not hold that for all well-behaved contexts $\mathcal{A}[\cdot]_{\mathtt{p}}$, $\mathcal{A}[A]_{\mathtt{p}}$ refines $\mathscr{C}(\mathbf{G})[A]_{\mathtt{p}}$.*

5.2 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:

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

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 $\mathtt{p} \in \mathcal{P}$ be a role, let $B = (Q_B, \Sigma_{\mathtt{p}}, t_0, \delta_B, F_B)$ denote the supertype state machine for $\mathtt{p}$, and let $A = (Q_A, \Sigma_{\mathtt{p}}, s_0, \delta_A, F_A)$ denote the subtype state machine for $\mathtt{p}$. We modify our state decoration function in Definition 4.6 to map states of A to subsets of states in B that themselves have non-empty decoration sets with respect to $\mathbf{G}$.

Definition 5.11 (State decoration with respect to a supertype). *Let* $\mathbf{G}$ *be a global type. Let* $\mathtt{p} \in \mathcal{P}$ *be a role, and let further* $B = (Q_B, \Sigma_\mathtt{p}, t_0, \delta_B, F_B)$ *and* $A = (Q_A, \Sigma_\mathtt{p}, s_0, \delta_A, F_A)$ *be two deterministic finite state machines for* $\mathtt{p}$*. 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:*

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

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 4.6.

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 $\mathtt{p}$.

Definition 5.12 (Transition origin and destination with respect to a supertype). *Let* $\mathbf{G}$ *be a global type, and let* $B = (Q_B, \Sigma_\mathtt{p}, t_0, \delta_B, F_B)$ *be a state machine. For* $x \in \Sigma_\mathtt{p}$ *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:*

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

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

Definition 5.13 (C_2). *Let* $\mathbf{G}$ *be a global type,* $\mathtt{p} \in \mathcal{P}$ *be a role, and let further* $B = (Q_B, \Sigma_\mathtt{p}, t_0, \delta_B, F_B)$ *and* $A = (Q_A, \Sigma_\mathtt{p}, s_0, \delta_A, F_A)$ *be two deterministic state machines for* $\mathtt{p}$*.* 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*:*
  $\forall s \xrightarrow{\mathtt{p} \triangleright \mathtt{q}!m} s' \in \delta_A.\ \text{tr-orig}_B(d_B(s) \xrightarrow{\mathtt{p} \triangleright \mathtt{q}!m} d_B(s')) = d_B(s).$

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

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

- 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 5.4:

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

Lemma 5.14 (Soundness of C_2). *If C_2 holds, then for all well-behaved contexts $\mathcal{A}[\cdot]_{\mathtt{p}}$, $\mathcal{A}[A]_{\mathtt{p}}$ refines $\mathcal{A}[B]_{\mathtt{p}}$.*

Predictably, the proof of soundness is directly adapted from the proof for C_2' by applying suitable "liftings", and can be found in the extended version [37].

Lemma 5.15 (Completeness of C_2). *If for all well-behaved contexts $\mathcal{A}[\cdot]_{\mathtt{p}}$, $\mathcal{A}[A]_{\mathtt{p}}$ refines $\mathcal{A}[B]_{\mathtt{p}}$, 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 $\mathcal{A}[\cdot]_{\mathtt{p}}$, such that $\mathcal{A}[A]_{\mathtt{p}}$ does not refine $\mathcal{A}[B]_{\mathtt{p}}$ 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 $\mathscr{C}(\mathbf{G})[A]_{\mathtt{p}}$. We were therefore able to use $\mathscr{C}(\mathbf{G})[\cdot]_{\mathtt{p}}$ as the witness well-behaved context. A similar characterization is true in the general case: a state in A is reachable by $\mathscr{C}(\mathbf{G})[A]_{\mathtt{p}}$ 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 5.8 to the general problem setting:

Lemma 5.16. *Let A, B be two state machines for $\mathtt{p}$, such that for all well-behaved contexts $\mathcal{A}[\cdot]_{\mathtt{p}}$, $\mathcal{A}[B]_{\mathtt{p}}$ 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 $u \in \Sigma_{\mathtt{p}}^*$ 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 $\mathtt{split}(\mathtt{trace}(\alpha \cdot G))\Downarrow_{\Sigma_{\mathtt{p}}} = u$, $\mathtt{split}(\mathtt{trace}(\alpha \cdot G))$ is a trace in both $\mathscr{C}(\mathbf{G})[A]_{\mathtt{p}}$ and $\mathscr{C}(\mathbf{G})[B]_{\mathtt{p}}$ 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 5.11), it holds that $d(t) \neq \emptyset$ and $t_0 \xrightarrow{u}{}^* t \in \delta_B$. Let $G \in d(t)$. We apply Lemma 5.8 to obtain a run $\alpha \cdot G$ such that $\mathtt{split}(\mathtt{trace}(\alpha \cdot G))\Downarrow_{\Sigma_{\mathtt{p}}} = u$, $\mathtt{split}(\mathtt{trace}(\alpha \cdot G))$ is a trace in $\mathscr{C}(\mathbf{G})[B]_{\mathtt{p}}$ and in the $\mathscr{C}(\mathbf{G})[B]_{\mathtt{p}}$ configuration reached on $\mathtt{split}(\mathtt{trace}(\alpha \cdot G))$, B is in state t. Because $s_0 \xrightarrow{u}{}^* s \in \delta_A$, and all non-$\mathtt{p}$ state machines are identical from $\mathscr{C}(\mathbf{G})[B]_{\mathtt{p}}$ to $\mathscr{C}(\mathbf{G})[A]_{\mathtt{p}}$, it is clear that $\mathtt{split}(\mathtt{trace}(\alpha \cdot G))$ is also a trace of $\mathscr{C}(\mathbf{G})[A]_{\mathtt{p}}$ and in the CSM configuration reached on $\mathtt{split}(\mathtt{trace}(\alpha \cdot G))$, A is in state s. □

Having found our witness well-behaved context $\mathscr{C}(\mathbf{G})[\cdot]_{\mathtt{p}}$, established Lemma 5.16 to replace Lemma 5.8, 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 $w \in \Sigma_{async}^*$ 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 5.17. *If* $\mathcal{A}[A]_{\mathtt{p}}$ *violates* Send Decoration Subtype Validity *or* Receive Decoration Subtype Validity*, then it does not hold that for all well-behaved contexts* $\mathcal{A}[\cdot]_{\mathtt{p}}$*,* $\mathcal{A}[A]_{\mathtt{p}}$ *refines* $\mathcal{A}[B]_{\mathtt{p}}$.

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 $\mathcal{A}[A]_{\mathtt{p}}$ but not a trace in $\mathscr{C}(\mathbf{G})[A]_{\mathtt{p}}$. In the general case, we can show that the same construction is a trace in $\mathcal{A}[A]_{\mathtt{p}}$ but not a trace in $\mathcal{A}[B]_{\mathtt{p}}$. We omit the proofs to avoid redundancy.

Lemma 5.18. *If* $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ *violates* Send Subtype Preservation*,* Receive Subtype Exhaustivity*, or* Final State Subtype Validity*, then it does not hold that for all well-behaved contexts* $\mathcal{A}[\cdot]_{\mathtt{p}}$*,* $\mathcal{A}[A]_{\mathtt{p}}$ *refines* $\mathcal{A}[B]_{\mathtt{p}}$.

6 Complexity Analysis

We complete our discussion with a complexity analysis of the two considered problems, building on the characterizations established in Theorem 4.1 and Theorem 5.4.

For the *Protocol Verification* problem, let m be the size of $\mathcal{A}$ and n the size of $\mathbf{G}$. Moreover, let $A_{\mathtt{p}}$ be the local implementation of some role $\mathtt{p}$ in $\mathcal{A}$. Observe that the sets $d_{\mathbf{G}}(s)$ for each state s of $A_{\mathtt{p}}$ as well as the sets $M^{\mathtt{p}}_{(G'\ldots)}$ 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, provided that the sets $d_{\mathbf{G}}(s)$ and $M^{\mathtt{p}}_{(G'\ldots)}$ are also computable in polynomial time.

To see this for the sets $M^{\mathtt{p}}_{(G'\ldots)}$, observe that the definition expands each occurrence of a recursion variable in $\mathbf{G}$ at most once. So the traversal takes time $O(n^2)$. For each traversed event $\mathtt{p}\to\mathtt{q}:m$ in $\mathbf{G}$, we need to perform a constant number of lookup, insertion, and deletion operations on a set of size at most n, which takes time $O(\log n)$. The time for computing $M^{\mathtt{p}}_{(G'\ldots)}$ is thus in $O(n^2 \log n)$.

Similarly, observe that the function $d_{\mathbf{G}}$ can be computed for the local implementation of each role $A_{\mathtt{p}} \in \mathcal{P}$ 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 $A_{\mathtt{p}}$, 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.

Following analogous reasoning, we can also establish that C_2 is checkable in polynomial time.

Theorem 6.1. *The* Protocol Verification *and* Protocol Refinement *problems are decidable in polynomial time.*

7 Related Work

Session types were first introduced in binary form by Honda in 1993 [29]. 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 [31], 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 [11, 28, 50]. Horne generalizes this connection to multiparty session types and non-commutative extensions of linear logic [32]. The connection between multiparty session types and logic is also explored in [10, 12, 13]. MSTs have since been extensively studied and widely adopted in practical programming languages; we refer the reader to [19] 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. [15] and [40] 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. Our paper targets 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* [43]. Mixed choices allow both send and receive actions, whereas separate choices consist purely of either sends or receives. [38] 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 [14], although [44] 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 [17, 30, 31], dependent predicates [48, 49], parametrization [18, 22] and data refinement [51].

Session type semantics. MSTs were introduced in [31] with a process algebra semantics. The connection to CSMs was established in [21], 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 [47] 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 [25] in the context of the π-calculus, which was in turn inspired by Pierce and Sangiorgi's work on subtyping for channel endpoints [45]. 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 [39], but differ in the guarantees provided. We discuss each in turn.

Synchronous subtyping follows the notions of covariance and contravariance introduced by [25], and checks that a subtype contains fewer sends and more receives than its supertype. For binary synchronous session types, Lange and Yoshida [34] 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. [26] present a precise subtyping relation that is universally quantified over all contexts, and restricts the local type syntax to directed choice. As mentioned in §1, [26], their subtyping relation is incomplete when generalized to asynchronous multiparty sessions with directed choice. As discussed in §2, 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" [26]. The complexity of the subtyping relation in [26] 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 [20]. First proposed for binary sessions by Mostrous and Yoshida [41], and for multiparty sessions by Mostrous et al. [42], 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 [5]. Asynchronous subtyping was shown to be undecidable for both binary and multiparty session types [6, 35]. Existing works are thus either restricted to binary protocols [1,5,6,35], prohibit non-deterministic choice involving multiple receivers [7,27], or make strong fairness assumptions on the network [7].

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 [26] for a survey.

Acknowledgements The authors thank Damien Zufferey for discussions and feedback. This work is funded in parts by the National Science Foundation under grant CCF-2304758. Felix Stutz was supported by the Deutsche Forschungsgemeinschaft project 389792660 TRR 248—CPEC.

References

1. Bacchiani, L., Bravetti, M., Lange, J., Zavattaro, G.: A session subtyping tool. In: Damiani, F., Dardha, O. (eds.) Coordination Models and Languages - 23rd IFIP WG 6.1 International Conference, COORDINATION 2021, Held as Part of the 16th International Federated Conference on Distributed Computing Techniques, DisCoTec 2021, Valletta, Malta, June 14-18, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12717, pp. 90–105. Springer (2021). https://doi.org/10.1007/978-3-030-78142-2_6
2. Barbanera, F., De'Liguoro, U.: Sub-behaviour relations for session-based client/server systems. Mathematical Structures in Computer Science **25**(6), 1339–1381 (2015). https://doi.org/10.1017/S096012951400005X

3. Bernardi, G.T., Hennessy, M.: Modelling session types using contracts. Math. Struct. Comput. Sci. **26**(3), 510–560 (2016). https://doi.org/10.1017/S0960129514000243
4. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM **30**(2), 323–342 (1983). https://doi.org/10.1145/322374.322380
5. Bravetti, M., Carbone, M., Lange, J., Yoshida, N., Zavattaro, G.: A sound algorithm for asynchronous session subtyping and its implementation. Log. Methods Comput. Sci. **17**(1) (2021), https://lmcs.episciences.org/7238
6. Bravetti, M., Carbone, M., Zavattaro, G.: On the boundary between decidability and undecidability of asynchronous session subtyping. Theor. Comput. Sci. **722**, 19–51 (2018). https://doi.org/10.1016/j.tcs.2018.02.010
7. Bravetti, M., Lange, J., Zavattaro, G.: Fair refinement for asynchronous session types. In: Kiefer, S., Tasson, C. (eds.) 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. Lecture Notes in Computer Science, vol. 12650, pp. 144–163. Springer (2021). https://doi.org/10.1007/978-3-030-71995-1_8
8. Bravetti, M., Zavattaro, G.: Relating session types and behavioural contracts: The asynchronous case. In: Ölveczky, P.C., Salaün, G. (eds.) Software Engineering and Formal Methods. pp. 29–47. Springer International Publishing, Cham (2019)
9. Bravetti, M., Zavattaro, G.: Asynchronous session subtyping as communicating automata refinement. Softw. Syst. Model. **20**(2), 311–333 (apr 2021). https://doi.org/10.1007/s10270-020-00838-x
10. Caires, L., Pérez, J.A.: Multiparty session types within a canonical binary theory, and beyond. In: Albert, E., Lanese, I. (eds.) Formal Techniques for Distributed Objects, Components, and Systems - 36th IFIP WG 6.1 International Conference, FORTE 2016, Held as Part of the 11th International Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete, Greece, June 6-9, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9688, pp. 74–95. Springer (2016). https://doi.org/10.1007/978-3-319-39570-8_6
11. Caires, L., Pfenning, F., Toninho, B.: Linear logic propositions as session types. Math. Struct. Comput. Sci. **26**(3), 367–423 (2016). https://doi.org/10.1017/S0960129514000218
12. Carbone, M., Lindley, S., Montesi, F., Schürmann, C., Wadler, P.: Coherence generalises duality: A logical explanation of multiparty session types. In: Desharnais, J., Jagadeesan, R. (eds.) 27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada. LIPIcs, vol. 59, pp. 33:1–33:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016). https://doi.org/10.4230/LIPIcs.CONCUR.2016.33
13. Carbone, M., Montesi, F., Schürmann, C., Yoshida, N.: Multiparty session types as coherence proofs. Acta Informatica **54**(3), 243–269 (2017). https://doi.org/10.1007/s00236-016-0285-y
14. Casal, F., Mordido, A., Vasconcelos, V.T.: Mixed sessions. Theor. Comput. Sci. **897**, 23–48 (2022). https://doi.org/10.1016/j.tcs.2021.08.005
15. Castagna, G., Dezani-Ciancaglini, M., Padovani, L.: On global types and multi-party session. Log. Methods Comput. Sci. **8**(1) (2012). https://doi.org/10.2168/LMCS-8(1:24)2012

16. Castagna, G., Gesbert, N., Padovani, L.: A theory of contracts for web services. ACM Trans. Program. Lang. Syst. **31**(5), 19:1–19:61 (2009). https://doi.org/10.1145/1538917.1538920
17. Castellani, I., Dezani-Ciancaglini, M., Giannini, P., Horne, R.: Global types with internal delegation. Theor. Comput. Sci. **807**, 128–153 (2020). https://doi.org/10.1016/j.tcs.2019.09.027
18. Charalambides, M., Dinges, P., Agha, G.A.: Parameterized, concurrent session types for asynchronous multi-actor interactions. Sci. Comput. Program. **115-116**, 100–126 (2016). https://doi.org/10.1016/j.scico.2015.10.006
19. Coppo, M., Dezani-Ciancaglini, M., Padovani, L., Yoshida, N.: A gentle introduction to multiparty asynchronous session types. In: Bernardo, M., Johnsen, E.B. (eds.) Formal Methods for Multicore Programming - 15th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2015, Bertinoro, Italy, June 15-19, 2015, Advanced Lectures. Lecture Notes in Computer Science, vol. 9104, pp. 146–178. Springer (2015). https://doi.org/10.1007/978-3-319-18941-3_4
20. Cutner, Z., Yoshida, N., Vassor, M.: Deadlock-free asynchronous message reordering in rust with multiparty session types. In: Lee, J., Agrawal, K., Spear, M.F. (eds.) PPoPP '22: 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Seoul, Republic of Korea, April 2 - 6, 2022. pp. 246–261. ACM (2022). https://doi.org/10.1145/3503221.3508404
21. Deniélou, P., Yoshida, N.: Multiparty session types meet communicating automata. In: Seidl, H. (ed.) Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7211, pp. 194–213. Springer (2012). https://doi.org/10.1007/978-3-642-28869-2_10
22. Deniélou, P., Yoshida, N., Bejleri, A., Hu, R.: Parameterised multiparty session types. Log. Methods Comput. Sci. **8**(4) (2012). https://doi.org/10.2168/LMCS-8(4:6)2012
23. Ellul, K., Krawetz, B., Shallit, J.O., Wang, M.: Regular expressions: New results and open problems. J. Autom. Lang. Comb. **10**(4), 407–437 (2005). https://doi.org/10.25596/jalc-2005-407
24. Esparza, J., Nielsen, M.: Decidability issues for petri nets - a survey. J. Inf. Process. Cybern. **30**(3), 143–160 (1994)
25. Gay, S.J., Hole, M.: Subtyping for session types in the pi calculus. Acta Informatica **42**(2-3), 191–225 (2005). https://doi.org/10.1007/s00236-005-0177-z
26. Ghilezan, S., Jakšić, S., Pantović, J., Scalas, A., Yoshida, N.: Precise subtyping for synchronous multiparty sessions. Journal of Logical and Algebraic Methods in Programming **104**, 127–173 (2019). https://doi.org/https://doi.org/10.1016/j.jlamp.2018.12.002
27. Ghilezan, S., Pantovic, J., Prokic, I., Scalas, A., Yoshida, N.: Precise subtyping for asynchronous multiparty sessions. Proc. ACM Program. Lang. **5**(POPL), 1–28 (2021). https://doi.org/10.1145/3434297
28. Girard, J.: Linear logic. Theor. Comput. Sci. **50**, 1–102 (1987). https://doi.org/10.1016/0304-3975(87)90045-4

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

30. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: Hankin, C. (ed.) 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. Lecture Notes in Computer Science, vol. 1381, pp. 122–138. Springer (1998). https://doi.org/10.1007/BFb0053567

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

32. Horne, R.: Session subtyping and multiparty compatibility using circular sequents. In: Konnov, I., Kovács, L. (eds.) 31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference). LIPIcs, vol. 171, pp. 12:1–12:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020). https://doi.org/10.4230/LIPIcs.CONCUR.2020.12

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

34. Lange, J., Yoshida, N.: Characteristic formulae for session types. In: Chechik, M., Raskin, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 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. Lecture Notes in Computer Science, vol. 9636, pp. 833–850. Springer (2016). https://doi.org/10.1007/978-3-662-49674-9_52

35. Lange, J., Yoshida, N.: On the undecidability of asynchronous session subtyping. In: Esparza, J., Murawski, A.S. (eds.) 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. Lecture Notes in Computer Science, vol. 10203, pp. 441–457 (2017). https://doi.org/10.1007/978-3-662-54458-7_26

36. Lange, J., Yoshida, N.: Verifying asynchronous interactions via communicating session automata. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11561, pp. 97–117. Springer (2019). https://doi.org/10.1007/978-3-030-25540-4_6

37. Li, E., Stutz, F., Wies, T.: Deciding subtyping for asynchronous multiparty sessions. CoRR **abs/2401.16395** (2024). https://doi.org/10.48550/arXiv.2401.16395

38. Li, E., Stutz, F., Wies, T., Zufferey, D.: Complete multiparty session type projection with automata. In: Enea, C., Lal, A. (eds.) Computer Aided Verification. pp. 350–373. Springer Nature Switzerland, Cham (2023)
39. Liskov, B., Wing, J.M.: A behavioral notion of subtyping. ACM Trans. Program. Lang. Syst. **16**(6), 1811–1841 (1994). https://doi.org/10.1145/197320.197383
40. Majumdar, R., Mukund, M., Stutz, F., Zufferey, D.: Generalising projection in asynchronous multiparty session types. In: Haddad, S., Varacca, D. (eds.) 32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference. LIPIcs, vol. 203, pp. 35:1–35:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPIcs.CONCUR.2021.35
41. Mostrous, D., Yoshida, N.: Session-based communication optimisation for higher-order mobile processes. In: Curien, P. (ed.) Typed Lambda Calculi and Applications, 9th International Conference, TLCA 2009, Brasilia, Brazil, July 1-3, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5608, pp. 203–218. Springer (2009). https://doi.org/10.1007/978-3-642-02273-9_16
42. Mostrous, D., Yoshida, N., Honda, K.: Global principal typing in partially commutative asynchronous sessions. In: Castagna, G. (ed.) 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. Lecture Notes in Computer Science, vol. 5502, pp. 316–332. Springer (2009). https://doi.org/10.1007/978-3-642-00590-9_23
43. Palamidessi, C.: Comparing the expressive power of the synchronous and asynchronous pi-calculi. Math. Struct. Comput. Sci. **13**(5), 685–719 (2003). https://doi.org/10.1017/S0960129503004043
44. Peters, K., Yoshida, N.: On the expressiveness of mixed choice sessions. In: Castiglioni, V., Mezzina, C.A. (eds.) 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. EPTCS, vol. 368, pp. 113–130 (2022). https://doi.org/10.4204/EPTCS.368.7
45. Pierce, B.C., Sangiorgi, D.: Typing and subtyping for mobile processes. Math. Struct. Comput. Sci. **6**(5), 409–453 (1996). https://doi.org/10.1017/s096012950007002x
46. Sipser, M.: Introduction to the theory of computation. PWS Publishing Company (1997)
47. Stutz, F.: Asynchronous multiparty session type implementability is decidable - lessons learned from message sequence charts. In: Ali, K., Salvaneschi, G. (eds.) 37th European Conference on Object-Oriented Programming, ECOOP 2023, July 17-21, 2023, Seattle, Washington, United States. LIPIcs, vol. 263, pp. 32:1–32:31. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023). https://doi.org/10.4230/LIPIcs.ECOOP.2023.32
48. Toninho, B., Caires, L., Pfenning, F.: Dependent session types via intuitionistic linear type theory. In: Schneider-Kamp, P., Hanus, M. (eds.) Proceedings of the 13th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 20-22, 2011, Odense, Denmark. pp. 161–172. ACM (2011). https://doi.org/10.1145/2003476.2003499
49. Toninho, B., Caires, L., Pfenning, F.: A decade of dependent session types. In: 23rd International Symposium on Principles and Practice of Declarative Programming. PPDP 2021,

Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3479394.3479398

50. Wadler, P.: Propositions as sessions. J. Funct. Program. **24**(2-3), 384–418 (2014). https://doi.org/10.1017/S095679681400001X
51. Zhou, F., Ferreira, F., Hu, R., Neykova, R., Yoshida, N.: Statically verified refinements for multiparty protocols. Proceedings of the ACM on Programming Languages **4**, 1–30 (11 2020). https://doi.org/10.1145/3428216