Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types

Nobuko Yoshida

Imperial College London

Abstract. Multiparty session types are a type system that can ensure the safety and liveness of distributed peers via the global specification of their interactions. To give a semantic justification of multiparty session types, this article explores the problem of fully characterising two classes of multiparty session types in terms of communicating automata. The sound and complete characterisation is then used for constructing of a global specification from a set of distributed uncontrolled behaviours. We equip global and local session types with labelled transition systems (LTSs) that faithfully represent asynchronous communications through unbounded buffered channels. Using the equivalence between the global and local LTSs, we identify two classes of communicating automata that exactly correspond to the projected local types of two different multiparty session type theories. We exhibit an algorithm to synthesise a global type from a collection of communicating automata. The key property of our findings is the notion of *multiparty compatibility* which non-trivially extends the duality condition for binary session types.

Key words: session types, global types, communicating automata, sound and complete characterisation, safety properties
ACM classification: F.1.1, F.1.2, D.3.1, D.3.3.

1 Introduction

Over the last decade, *session types* [22, 31] have been studied as data types or functional types for communications and distributed systems. A recent discovery by [7, 33], which establishes a Curry-Howard isomorphism between binary session types and linear logics, confirms that session types and the notion of duality between type constructs have canonical meanings. Multiparty session types [23, 3] were proposed as a major generalisation of binary session types. They can enforce communication safety and deadlock-freedom for more than two peers thanks to a choreographic specification (called *global type*) of the interaction. Global types are projected to end-point types (*local types*), against which processes can be statically type-checked and verified to behave correctly.

The motivation of this article comes from our practical experiences that, in many situations, even where we start from the end-point projections of a choreography, we need to reconstruct a global type from distributed specifications. End-point specifications are usually available, either through inference from the control flow, or through existing service interfaces, and always in forms akin to individual communicating finite state machines. If one knows the precise conditions under which a global type can be constructed (i.e. the conditions of *synthesis*), not only the global safety property which multiparty session types ensure is guaranteed, but also the generated global type can be used as a refinement and be integrated within the distributed system development life-cycle (see § 7 for applications [30, 29]). This article attempts to give the synthesis condition as a sound and complete characterisation of multiparty session types with respect to Communicating Finite State Machines (CFSMs) [6], also known as Communicating Automata. CFSMs are a classical model for protocol specification and verification. Before being used in many industrial contexts, CFSMs have been a pioneer theoretical formalism in which distributed safety properties could be formalised and studied. Building a connection between communicating automata and session types allows to answer some open questions in session types which have been asked since [22]. The first question is about expressiveness: to which class of CFSMs do session types correspond? The second question concerns the semantical correspondence between session types and CFSMs: how do the safety properties that session types guarantee relate to those of CFSMs? The third question is about efficiency: why do session types provide efficient algorithms for type-checking while general CFSMs are undecidable? CFSMs can been also seen as generalised end-point specifications, therefore an excellent target for a common comparison ground for protocol specification languages and for synthesis. As explained below, to identify a complete set of CFSMs for synthesis and to answer the three questions above, we first need to find out the canonical duality notion in multiparty session types.

Characterisation of binary session types as communicating automata The subclass which fully characterises *binary session types* was actually proposed by Gouda, Manning and Yu in 1984 [20] in a pure communicating automata context. Consider a simple business protocol between a Buyer and a Seller from the Buyer's viewpoint: Buyer sends the title of a book, Seller answers with a quote. If Buyer is satisfied by the quote, then he sends his address and Seller sends back the delivery date; otherwise it retries the same conversation. This can be described by the following session type:

$$\mu \mathtt{t}.!\mathsf{title};\ ?\mathsf{quote};\ !\{\ \mathsf{ok} :!\mathsf{addrs}; ?\mathsf{date}; \mathsf{end}, \quad \mathsf{retry} : \mathtt{t}\ \} \tag{1.1}$$

where the operator $!\mathsf{title}$ denotes an output of the title, whereas $?\mathtt{quote}$ denotes an input of a quote. The output choice features the two options $\mathtt{ok}$ and $\mathtt{retry}$ and ; denotes sequencing. $\mathtt{end}$ represents the termination of the session, and $\mu\mathtt{t}$ is recursion.

The simplicity and tractability of binary sessions come from the notion of *duality* in interactions [19]. The interaction pattern of the Seller is fully given as the dual of the type in (1.1) (exchanging input ! and output ? in the original type). When composing two parties, we only have to check they have mutually dual types, and the resulting communication is guaranteed to be deadlock-free. Essentially the same characterisation is given in communicating automata. Buyer and Seller's session types are represented by the two machines of Figure 1.

We can observe that these CFSMs satisfy three conditions. First, the communications are *deterministic*: messages that are part of the same choice, $\mathtt{ok}$ and $\mathtt{retry}$ here, are distinct. Secondly, there is no mixed state (each state has either only sending actions or

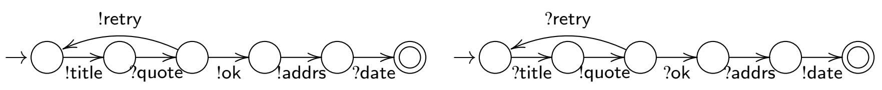


Fig. 1. Two dual communicating automata: Buyer and Seller

only receiving actions). Third, these two machines have *compatible* traces (i.e. dual): the Seller machine can be defined by exchanging sending to receiving actions and vice versa. Breaking one of these conditions allows deadlock situations and omitting one of the first two conditions makes the compatibility checking undecidable [20, 32].

Multiparty compatibility This notion of duality is no longer effective in multiparty communications, where the whole conversation cannot be reconstructed from only a single behaviour. To bypass the gap between binary and multiparty, we take the *synthesis* approach, that is to find conditions which allow a global choreography to be built from the local machine behaviour. Instead of directly trying to decide whether the communications of a system will satisfy safety (which is undecidable in the general case), inferring a global type is sufficient to guarantee the safety as a direct consequence.

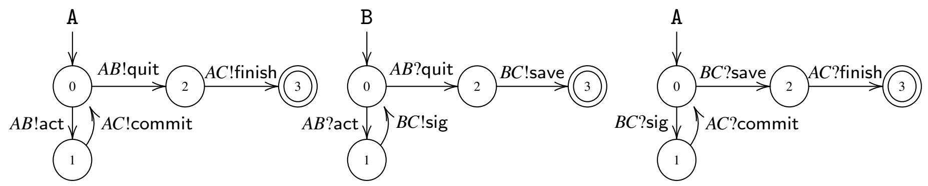


Fig. 2. Commit example: CFSMs

We give a simple example to illustrate the problem. The Commit protocol in Figure 2 involves three machines: Alice A, Bob B and Carol C. A orders B to act or quit. If act is sent, B sends a signal to C, and A sends a commitment to C and continues. Otherwise B informs C to save the data and A gives the final notification to C to terminate the protocol.

This article presents a decidable notion of *multiparty compatibility* as a generalisation of duality of binary sessions, which in turns characterises a synthesis condition. The idea is to check the duality between each automaton and the rest, up to the internal communications (1-bounded executions in the terminology of CFSMs, see § 2) that the other machines will independently perform. For example, in Figure 2, to check the compatibility of trace $AB!\mathsf{quit}\ AC!\mathsf{finish}$ in A, we observe the dual trace $AB?\mathsf{quit} \cdot AC?\mathsf{finish}$ from B and C executing the internal communications between B and C such that $BC!\mathsf{save} \cdot BC?\mathsf{save}$. If this extended duality is valid for all the machines from any 1-bounded reachable state, then they satisfy multiparty compatibility and can build a well-formed global choreography.

Contributions and Outline This article is an expanded version of [17]. The additional results newly presented in this article are (1) the case of a class of multiparty session types which is not included in [17]; and (2) the soundness which is not stated in [17]. In addition, we have given the full definitions, complete proofs and more examples. A extensive discussion of recent related work is also added.

Section 2 gives definitions of CFSMs and their safety and liveness properties. Section 3 defines new labelled transition systems for global and local types that represent the abstract observable behaviour of typed processes. We prove that a global type behaves exactly as its projected local types, and the same result between a single local type and its CFSMs interpretation. These correspondences are the key to prove the main theorems.

Section 4 studies the class of *sequential CFSMs*, where at any moment only one machine is allowed to send messages (this class is not included in [17]). In this class, the condition of local machines can directly identify a sound and complete characterisation of a subclass of multiparty session types studied in [4, 10].

Section 5 first defines the notion of *multiparty compatibility* and studies its safety and liveness properties. Secondly, we prove the soundness theorem: a set of CFSMs which are a translation of a global type are multiparty compatible. The first and second results link the safety properties of CFSMs and those of multiparty session types.

Section 6 develops an algorithm for the synthesis of global types from CFSMs. Then we prove the completeness results between global types and multiparty compatible CFSMs.

Section 7 discusses related work and Section 8 concludes. Appendix contains the detailed proofs omitted from the main sections.

2 Communicating Finite State Machines

This section starts from some preliminary notations (following [10]). ε is the empty word. $\mathbb{A}$ is a finite alphabet and $\mathbb{A}^*$ is the set of all finite words over $\mathbb{A}$. $|x|$ is the length of a word x and $x.y$ or xy the concatenation of two words x and y. Let $\mathcal{P}$ be a set of *participants* fixed throughout the paper: $\mathcal{P} \subseteq \{\mathtt{A},\mathtt{B},\mathtt{C},\dots,\mathtt{p},\mathtt{q},\dots\}$.

Definition 2.1 (CFSM). A communicating finite state machine is a finite transition system given by a 5-tuple $M = (Q, C, q_0, \mathbb{A}, \delta)$ where (1) Q is a finite set of *states*; (2) $C = \{\mathtt{pq} \in \mathcal{P}^2 \mid \mathtt{p} \neq \mathtt{q}\}$ is a set of channels; (3) $q_0 \in Q$ is an initial state; (4) $\mathbb{A}$ is a finite *alphabet* of messages, and (5) $\delta \subseteq Q \times (C \times \{!, ?\} \times \mathbb{A}) \times Q$ is a finite set of *transitions*.

In transitions, $\mathtt{pq}!a$ denotes the *sending* action of a from process $\mathtt{p}$ to process $\mathtt{q}$, and $\mathtt{pq}?a$ denotes the *receiving* action of a from $\mathtt{p}$ by $\mathtt{q}$. ℓ, ℓ' range over actions and we define the *subject* of an action ℓ as the principal in charge of it: $subj(\mathtt{pq}!a) = subj(\mathtt{qp}?a) = \mathtt{p}$.

A state $q \in Q$ whose outgoing transitions are all labelled with sending (resp. receiving) actions is called a *sending* (resp. *receiving*) state. If q has both sending and receiving outgoing transitions, q is called *mixed*. We say q is *directed* if it contains only sending (resp. receiving) actions to (resp. from) the same (identical) participant. A state $q \in Q$ which does not have any outgoing transition is called *final*. A *path* in M is a finite sequence of $q_0, \ell_0, q_1, \dots, \ell_{n-1}, q_n$ $(n \geq 1)$ such that $(q_i, \ell_i, q_{i+1}) \in \delta$ $(0 \leq i \leq n-1)$, and we write $q \xrightarrow{\ell} q'$ if $(q,\ell,q') \in \delta$. M is *connected* if for every state $q \neq q_0$, there is a path from q_0 to q. Hereafter we assume each CFSM is connected.

A CFSM $M = (Q,C,q_0,\mathbb{A},\delta)$ is *deterministic* if for all states $q \in Q$ and all actions ℓ, $(q,\ell,q'),(q,\ell,q'') \in \delta$ imply $q' = q''$.[1]

Definition 2.2 (CS). A (communicating) system S is a tuple $S = (M_\mathtt{p})_{\mathtt{p}\in\mathcal{P}}$ of CFSMs such that $M_\mathtt{p} = (Q_\mathtt{p}, C, q_{0\mathtt{p}}, \mathbb{A}, \delta_\mathtt{p})$.

For $M_\mathtt{p} = (Q_\mathtt{p}, C, q_{0\mathtt{p}}, \mathbb{A}, \delta_\mathtt{p})$, we define a *configuration* of $S = (M_\mathtt{p})_{\mathtt{p}\in\mathcal{P}}$ to be a tuple $s = (\vec{q};\vec{w})$ where $\vec{q} = (q_\mathtt{p})_{\mathtt{p}\in\mathcal{P}}$ with $q_\mathtt{p} \in Q_\mathtt{p}$ and where $\vec{w} = (w_\mathtt{pq})_{\mathtt{p}\neq\mathtt{q}\in\mathcal{P}}$ with $w_\mathtt{pq} \in \mathbb{A}^*$. The element $\vec{q}$ is called a *control state* and $q \in Q_\mathtt{p}$ is the *local state* of machine $M_\mathtt{p}$.

Definition 2.3 (reachable state). Let S be a communicating system. A configuration $s' = (\vec{q}\,';\vec{w}\,')$ is *reachable* from another configuration $s = (\vec{q};\vec{w})$ by the *firing of the transition* t, written $s \to s'$ or $s \xrightarrow{t} s'$, if there exists $a \in \mathbb{A}$ such that either:

1. $t = (q_\mathtt{p}, \mathtt{pq}!a, q'_\mathtt{p}) \in \delta_\mathtt{p}$ and (a) $q'_{\mathtt{p}'} = q_{\mathtt{p}'}$ for all $\mathtt{p}' \neq \mathtt{p}$; and (b) $w'_\mathtt{pq} = w_\mathtt{pq}.a$ and $w'_{\mathtt{p}'\mathtt{q}'} = w_{\mathtt{p}'\mathtt{q}'}$ for all $\mathtt{p}'\mathtt{q}' \neq \mathtt{pq}$; or
2. $t = (q_\mathtt{q}, \mathtt{pq}?a, q'_\mathtt{q}) \in \delta_\mathtt{q}$ and (a) $q'_{\mathtt{p}'} = q_{\mathtt{p}'}$ for all $\mathtt{p}' \neq \mathtt{q}$; and (b) $w_\mathtt{pq} = a.w'_\mathtt{pq}$ and $w'_{\mathtt{p}'\mathtt{q}'} = w_{\mathtt{p}'\mathtt{q}'}$ for all $\mathtt{p}'\mathtt{q}' \neq \mathtt{pq}$.

The condition (1-b) puts the content a to a channel pq, while (2-b) gets the content a from a channel pq. The reflexive and transitive closure of $\to$ is $\to^*$. For a transition $t = (s,\ell,s')$, we refer to ℓ by $act(t)$. We write $s_1 \xrightarrow{t_1 \cdots t_m} s_{m+1}$ for $s_1 \xrightarrow{t_1} s_2 \cdots \xrightarrow{t_m} s_{m+1}$ and use φ to denote $t_1 \cdots t_m$. We extend act to these sequences: $act(t_1 \cdots t_n) = act(t_1) \cdots act(t_n)$.

The *initial configuration* of a system is $s_0 = (\vec{q}_0;\vec{\varepsilon})$ with $\vec{q}_0 = (q_{0\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$. A *final configuration* of the system is $s_f = (\vec{q};\vec{\varepsilon})$ with all $q_\mathtt{p} \in \vec{q}$ final. A configuration s is *reachable* if $s_0 \to^* s$ and we define the *reachable set* of S as $RS(S) = \{s \mid s_0 \to^* s\}$. We define the traces of a system S to be $Tr(S) = \{act(\varphi) \mid \exists s \in RS(S), s_0 \xrightarrow{\varphi} s\}$.

We now define several properties about communicating systems and their configurations. These properties will be used in § 5 to characterise the systems that correspond to multiparty session types. Let S be a communicating system, t one of its transitions and $s = (\vec{q};\vec{w})$ one of its configurations. The following definitions of configuration properties follow [10, Definition 12].

1. s is *stable* if all its buffers are empty, i.e., $\vec{w} = \vec{\varepsilon}$.
2. s is a *deadlock configuration* if s is not final, and $\vec{w} = \vec{\varepsilon}$ and each $q_\mathtt{p}$ is a receiving state, i.e. all machines are blocked, waiting for messages.
3. s is an *orphan message configuration* if all $q_\mathtt{p} \in \vec{q}$ are final but $\vec{w} \neq \emptyset$, i.e. there is at least an orphan message in a buffer.
4. s is an *unspecified reception configuration* if there exists $\mathtt{q} \in \mathcal{P}$ such that $q_\mathtt{q}$ is a receiving state and $(q_\mathtt{q}, \mathtt{pq}?a, q'_\mathtt{q}) \in \delta$ implies that $|w_\mathtt{pq}| > 0$ and $w_\mathtt{pq} \notin a\mathbb{A}^*$, i.e $q_\mathtt{q}$ is prevented from receiving any message from buffer pq.

[1] "Deterministic" often means the same channel should carry a unique value, i.e. if $(q,c!a,q') \in \delta$ and $(q,c!a',q'') \in \delta$ then $a = a'$ and $q' = q''$. Here we follow a different definition [10] in order to represent branching type constructs.

A sequence of transitions $s_1 \xrightarrow{t_1} s_2 \cdots s_m \xrightarrow{t_m} s_{m+1}$ is said to be *k-bounded* if no channel of any intermediate configuration s_i contains more than k messages. More precisely, in each $w_{\mathtt{p}\mathtt{q}}$ in s_i, $|w_{\mathtt{p}\mathtt{q}}| \leq k$. We define the k-reachability set of S to be the largest subset $RS_k(S)$ of $RS(S)$ within which each configuration s can be reached by a k-bounded execution from s_0. Note that, given a communicating system S, for every integer k, the set $RS_k(S)$ is finite and computable. We say that a trace φ is n-bound, written $bound(\varphi) = n$, if at no point in the trace the number of past send actions exceeds the number of past receive actions by n. We then define the equivalences: (1) $S \approx S'$ is $\forall \varphi,\ \varphi \in Tr(S) \Leftrightarrow \varphi \in Tr(S')$; and (2) $S \approx_n S'$ is $\forall \varphi,\ bound(\varphi) \leq n \Rightarrow (\varphi \in Tr(S) \Leftrightarrow \varphi \in Tr(S'))$.

The following key properties will be examined throughout the paper as properties that multiparty session type can enforce.

Definition 2.4 (safety and liveness). (1) A communicating system S is *deadlock-free* (resp. *orphan message-free*, *reception error-free*) if for all $s \in RS(S)$, s is not a deadlock (resp. orphan message, unspecified reception) configuration. (2) S satisfies the *liveness property*[2] if for all $s \in RS(S)$, there exists $s \longrightarrow^* s'$ such that s' is final.

Half-duplex systems Half-duplex systems with two machines enjoy the decidability of these problems with a polynomial time complexity [10], and its subclass called compatible deterministic two machines without mixed states [20, 32] correspond to *binary* session types (see § 1).

We now define two classical generalisations of half-duplex systems to the multiparty case.

Definition 2.5 (multiparty half-duplex).

1. A system is *strict multiparty half-duplex* if, for all reachable configuration $(\vec{q};\vec{w})$, there is at most one $i \in \{1,...,p\}$ such that $w_i \neq \varepsilon$.
2. A system is *multiparty half-duplex* if, for all reachable configuration $s = (\vec{q};\vec{w})$ such that $\vec{q} = (q_1,\cdots,q_n)$, $C = \cup_{1\leq i\neq j\leq n}\{ij\}$ and $\vec{w} = (w_1,\cdots,w_p)$ with $(p = n \times (n-1))$, (1) ij is a unique channel used for sending from q_i to q_j (receiving by q_j), (2) if $w_k \neq \varepsilon$ with $k = ij$, then $w_h = \varepsilon$ with $h = ji$.

The strict multiparty half-duplex definition is called *the restricted generalisation* of half-duplex with two machines in [10, § 4.1.2]. The multiparty half-duplex definition is called *the natural generalisation* in [10, § 4.1.1] where each pair of machines linked by two channels, one in each direction, has a half-duplex communication.

Theorem 2.1 (multiparty half-duplex). *(Corollary 27, Theorem 36 in [10]) In half-duplex systems with two machines, the properties of Definition 2.4 are decidable. However, in multiparty half-duplex systems with three machines or more, these problems are undecidable.*

The above theorem, however, states that an extension to identify a decidable subset of multiparty half-duplex systems is very challenging since multiparty half-duplex systems with three machines are already expressive enough to simulate Turing Machines.

[2] The terminology follows [8].

Section 4 presents a subclass of strict multiparty half-duplex systems. Section 5 studies two examples of multiparty half-duplex systems which satisfy the above safety properties. They are given from corresponding global types by the projection algorithms defined in the next section.

3 Global and local types: the LTSs and translations

This section presents multiparty session types, our main object of study. For the syntax of types, we follow [3] which is the most widely used syntax in the literature. We introduce two labelled transition systems, for local types and for global types, and show the equivalence between local types and communicating automata.

Syntax A *global type*, written $G, G', ..$, describes the whole conversation scenario of a multiparty session as a type signature, and a *local type*, written by $T, T', ..$, type-abstract sessions from each end-point's view. $\mathsf{p}, \mathsf{q}, \dots \in \mathcal{P}$ denote participants (see § 2 for conventions). The syntax of types is given as:

$$\begin{array}{rcl} G & ::= & \mathsf{p} \to \mathsf{p}' \colon \{a_j.G_j\}_{j \in J} \;\big|\; \mu \mathsf{t}.G \;\big|\; \mathsf{t} \;\big|\; G_1 \mid G_2 \;\big|\; \mathsf{end} \\ T & ::= & \mathsf{p}?\{a_i.T_i\}_{i \in I} \;\big|\; \mathsf{p}!\{a_i.T_i\}_{i \in I} \;\big|\; \mu \mathsf{t}.T \;\big|\; \mathsf{t} \;\big|\; \mathsf{end} \end{array}$$

$a_j \in \mathbb{A}$ corresponds to the usual message label in session type theory. We omit the mention of the carried types from the syntax in this paper, as we are not directly concerned by typing processes. Global branching type $\mathsf{p} \to \mathsf{p}' \colon \{a_j.G_j\}_{j \in J}$ states that participant p can send a message with one of the a_i labels to participant p' and that interactions described in G_j follow. We require $\mathsf{p} \neq \mathsf{p}'$ to prevent self-sent messages and $a_i \neq a_k$ for all $i \neq k \in J$. Recursive type $\mu \mathsf{t}.G$ is for recursive protocols, assuming that type variables $(\mathsf{t}, \mathsf{t}', \dots)$ are guarded in the standard way, i.e. they only occur under branchings. Type $G_1 \mid G_2$ is a parallel composition. Type end represents session termination (often omitted). $\mathsf{p} \in G$ means that p appears in G and $\mathsf{fv}(G)$ denotes a set of free type variables in G. We identify $G_1 \mid G_2$ and $G_2 \mid G_1$; $(G_1 \mid G_2) \mid G_3$ and $G_1 \mid (G_2 \mid G_3)$; and $\mathsf{end} \mid G = G$.

Concerning local types, the *branching type* $\mathsf{p}?\{a_i.T_i\}_{i \in I}$ specifies the reception of a message from p with a label among the a_i. The *selection type* $\mathsf{p}!\{a_i.T_i\}_{i \in I}$ is its dual. The remaining type constructors are the same as global types. When branching is a singleton, we write $\mathsf{p} \to \mathsf{p}' : a.G'$ for global, and $\mathsf{p}!a.T$ or $\mathsf{p}?a.T$ for local.

Projection The relation between global and local types is formalised by projection [3, 23]. We define the projection with the merging operator $\bowtie$ from [15]: it allows each branch of the global type to actually contain different interaction patterns.

Definition 3.1 (projection). The *projection of G onto* $\mathtt{p}$ (written $G \upharpoonright \mathtt{p}$) is defined as:

$$\mathtt{p} \to \mathtt{p}'\colon \{a_j.G_j\}_{j\in J} \upharpoonright \mathtt{q} = \begin{cases} \mathtt{p}!\{a_j.G_j \upharpoonright \mathtt{q}\}_{j\in J} & \mathtt{q}=\mathtt{p} \\ \mathtt{p}?\{a_j.G_j \upharpoonright \mathtt{q}\}_{j\in J} & \mathtt{q}=\mathtt{p}' \\ \sqcup_{j\in J} G_j \upharpoonright \mathtt{q} & \text{otherwise} \end{cases}$$

$$(\mu \mathtt{t}.G) \upharpoonright \mathtt{p} = \begin{cases} \mu\mathtt{t}.G \upharpoonright \mathtt{p} & G \upharpoonright \mathtt{p} \neq \mathtt{t} \\ \mathsf{end} & \text{otherwise} \end{cases}$$

$$(G_1 \mid G_2) \upharpoonright \mathtt{p} = G_1 \upharpoonright \mathtt{p} \mid G_2 \upharpoonright \mathtt{p} \quad \text{if } \mathtt{p} \notin G_1 \text{ or } \mathtt{p} \notin G_2 \text{ and } \mathsf{fv}(G_1) \cap \mathsf{fv}(G_2) = \emptyset$$

$$\mathtt{t} \upharpoonright \mathtt{p} = \mathtt{t}$$

$$\mathsf{end} \upharpoonright \mathtt{p} = \mathsf{end}$$

where the merging operation $\sqcup$ is defined as a partial commutative operator over two types such that:

- $T \sqcup T = T$ for all types;
- $\mathtt{t} \sqcup \mathtt{p}?\{a_j.T_j'\}_{j\in J} = \mathtt{p}?\{a_j.T_j'\}_{j\in J}$ for all $\mathtt{p}?\{a_j.T_j'\}_{j\in J}$; or
- $\mathtt{p}?\{a_k.T_k\}_{k\in K} \sqcup \mathtt{p}?\{a_j.T_j'\}_{j\in J} = \mathtt{p}?(\{a_k.(T_k \sqcup T_k')\}_{k\in K\cap J} \cup \{a_k.T_k\}_{k\in K\setminus J} \cup \{a_j.T_j'\}_{j\in J\setminus K})$

and homomorphic for other types (i.e. $\mathscr{C}[T_1] \sqcup \mathscr{C}[T_2] = \mathscr{C}[T_1 \sqcup T_2]$ where $\mathscr{C}$ is a context for local types). The merging operation between T_1 and T_2 is defined only if $T_1 \bowtie T_2$ holds, where the *mergeability relation* $\bowtie$ is the smallest congruence relation over local types such that:

$$\frac{\forall i \in (K\cap J).T_i \bowtie T_i' \quad \forall k \in (K\setminus J), \forall j \in (J \setminus K).a_k \neq a_j}{\mathtt{p}?\{a_k.T_k\}_{k\in K} \bowtie \mathtt{p}?\{a_j.T_j'\}_{j\in J}} \qquad \mathtt{t} \bowtie \mathtt{p}?\{a_j.T_j'\}_{j\in J}$$

We say that G is *well-formed* if for all $\mathtt{p} \in \mathcal{P}$, $G \upharpoonright \mathtt{p}$ is defined.

The condition on parallel composition implies that, in well-formed global types, the set of participants of G_1 and G_2 in $G_1 \mid G_2$ are disjoint. It is required to be able to map to local types without using parallel compositions. Similarly the condition of recursive variables

Example 3.1 (Commit). The global type for the commit protocol in Figure 2 is:

$$\mu\mathtt{t}.\mathtt{A} \to \mathtt{B}\colon\{\mathsf{act}.\mathtt{B} \to \mathtt{C}\colon\{\mathsf{sig}.\mathtt{A} \to \mathtt{C}\colon\mathsf{commit}.\mathtt{t}\}, \mathsf{quit}.\mathtt{B} \to \mathtt{C}\colon\{\mathsf{save}.\mathtt{A} \to \mathtt{C}\colon\mathsf{finish}.\mathsf{end}\}\}$$

Then using the mergeability operator, $\mathtt{C}$'s local type is given as:

$$\mu\mathtt{t}.\mathtt{B}?\{\mathsf{sig}.\mathtt{A}?\{\mathsf{commit}.\mathtt{t}\}, \mathsf{save}.\mathtt{A}?\{\mathsf{finish}.\mathsf{end}\}\}$$

Example 3.2 (Merging inputs). We explain a merging rule between an input $\mathtt{p}?\{a_j.T_j'\}_{j\in J}$ and a recursive type variable $\mathtt{t}$. Consider the following global type:

$$\mu\mathtt{t}.\mathtt{A} \to \mathtt{B}\colon\{\mathsf{commit}.\mathtt{t}, \mathsf{stop}.\mathtt{A} \to \mathtt{C}\colon\mathsf{finish}.\mathsf{end}\}$$

Then $\mathtt{A}$, $\mathtt{B}$ and $\mathtt{C}$'s local types, $T_{\mathtt{A}}$, $T_{\mathtt{B}}$ and $T_{\mathtt{C}}$ are given as:

$$T_{\mathtt{A}} = \mu\mathtt{t}.\mathtt{B}!\{\mathsf{commit}.\mathtt{t}, \mathsf{stop}.\mathtt{C}!\mathsf{finish}.\mathsf{end}\}$$

$$T_{\mathtt{B}} = \mu\mathtt{t}.\mathtt{A}?\{\mathsf{commit}.\mathtt{t}, \mathsf{stop}.\mathsf{end}\} \qquad T_{\mathtt{C}} = \mathtt{A}?\mathsf{finish}.\mathsf{end}$$

To obtain $T_{\mathtt{C}}$, we use $\mathtt{A}?\mathsf{finish}.\mathsf{end} \sqcup \mathtt{t} = T_{\mathtt{C}}$. Later we show a translation of these types into CFSMs are multiparty compatible, hence safe.

Below we show the complexity of the mergeability operator. We define the size of $|G|$ as: $|\mathsf{p} \to \mathsf{p}' \colon \{a_j.G_j\}_{j\in J}| = 1 + \Sigma_{j\in J}(1+|G_j|)$, $|\mu \mathsf{t}.G| = 1+|G|$, $|\mathsf{t}| = |\mathsf{end}| = 1$, and $|G_1 \mid G_2| = |G_1| + |G_2| + 1$. We define *the maximum degree of labels* in G (denoted by $\max_L(G)$) as $\max_L(\mathsf{p} \to \mathsf{p}' \colon \{a_j.G_j\}_{j\in J}) = \max(|J|, \max_L(G_j)_{\{j\in J\}})$; $\max_L(G_1 \mid G_2) = \max(\max_L(G_1), \max_L(G_2))$; $\max_L(\mu \mathsf{t}.G) = \max_L(G)$; and $\max_L(\mathsf{t}) = \max_L(\mathsf{end}) = 0$.

Proposition 3.1. *The time complexity of the projection algorithm in Definition 3.1 is $n \times O(|L| \times (|G|+|L|))$ where n is a number of participants in G, $|G|$ is the size of G and $|L|$ is the maximum degree of labels in G. If $|L| \ll |G|$, then $n \times O(|L| \times |G|)$.*

Proof. The size of each local type is bounded by the size of the global type. The number of merging operations for each participant is limited by the size of its local type. Then the merging operator computes a difference of sets of labels in a branching, which is bounded by the maximum degree of labels in G. More precisely, $O(\mathsf{p} \to \mathsf{p}' \colon \{a_j.G_j\}_{j\in J})$ is calculated as $O(|J|^2 + \Sigma_{j\in J}|G_j|) = O(|J|^2 + (|J| \times \max(|G_j|))) = O(|J| \times (|J| + \max(|G_j|))) = O(|L| \times (|L| + |G|))$. □

We now present labelled transition relations (LTS) for global and local types and their sound and complete correspondence.

LTS over global types We first designate the observables $(\ell, \ell', \ldots)$. We choose here to follow the definition of actions for CFSMs where a label ℓ denotes the sending or the reception of a message of label a from p to p':

$$\ell \ ::= \ \mathsf{pp}'!a \mid \mathsf{pp}'?a$$

In order to define an LTS for global types, we need to represent intermediate states in the execution. For this reason, we introduce in the grammar of G the construct $\mathsf{p} \rightsquigarrow \mathsf{p}' \colon j\ \{a_i.G_i\}_{i\in I}$ to represent the fact that message a_j has been sent but not yet received.

Definition 3.2 (LTS over global types). The relation $G \xrightarrow{\ell} G'$ is defined as (*subj*(ℓ) is defined in § 2):

$$[\mathrm{GR1}] \quad \mathsf{p} \to \mathsf{p}' \colon \{a_i.G_i\}_{i\in I} \xrightarrow{\mathsf{pp}'!a_j} \mathsf{p} \rightsquigarrow \mathsf{p}' \colon j\ \{a_i.G_i\}_{i\in I} \quad (j \in I)$$

$$[\mathrm{GR2}] \quad \mathsf{p} \rightsquigarrow \mathsf{p}' \colon j\ \{a_i.G_i\}_{i\in I} \xrightarrow{\mathsf{pp}'?a_j} G_j \qquad [\mathrm{GR3}]\ \frac{G[\mu \mathsf{t}.G/\mathsf{t}] \xrightarrow{\ell} G'}{\mu \mathsf{t}.G \xrightarrow{\ell} G'}$$

$$[\mathrm{GR4}]\frac{\forall j \in I \quad G_j \xrightarrow{\ell} G_j' \quad \mathsf{p}, \mathsf{q} \notin \mathit{subj}(\ell)}{\mathsf{p} \to \mathsf{q} \colon \{a_i.G_i\}_{i\in I} \xrightarrow{\ell} \mathsf{p} \to \mathsf{q} \colon \{a_i.G_i'\}_{i\in I}}$$

$$[\mathrm{GR5}]\frac{G_j \xrightarrow{\ell} G_j' \quad \mathsf{q} \notin \mathit{subj}(\ell) \quad \forall i \in I \setminus j, G_i' = G_i}{\mathsf{p} \rightsquigarrow \mathsf{q} \colon j\ \{a_i.G_i\}_{i\in I} \xrightarrow{\ell} \mathsf{p} \rightsquigarrow \mathsf{q} \colon j\ \{a_i.G_i'\}_{i\in I}}$$

$$[\mathrm{GR6}]\ \frac{G_1 \xrightarrow{\ell} G_1'}{G_1 \mid G_2 \xrightarrow{\ell} G_1' \mid G_2} \qquad [\mathrm{GR7}]\ \frac{G_2 \xrightarrow{\ell} G_2'}{G_1 \mid G_2 \xrightarrow{\ell} G_1 \mid G_2'}$$

[GR1] represents the emission of a message while [GR2] describes the reception of a message. [GR3] governs recursive types. [GR4,5] define the asynchronous semantics of global types, where the syntactic order of messages is enforced only for the participants that are involved. They are explained in the following example. [GR6,7] define the parallel composition.

Example 3.3 (LTS of global types). We give examples of the LTS of global types. When the participants of two consecutive communications are disjoint, as in: $G_1 = \mathtt{A} \to \mathtt{B} : a.\mathtt{C} \to \mathtt{D} : b.\mathsf{end}$, we can observe the emission (and possibly the reception) of b before the emission (or reception) interactions of a (by [GR4]).

A more interesting example is: $G_2 = \mathtt{A} \to \mathtt{B} : a.\mathtt{A} \to \mathtt{C} : b.\mathsf{end}$. We write $\ell_1 = AB!a$, $\ell_2 = AB?a$, $\ell_3 = AC!b$ and $\ell_4 = AC?b$. The LTS allows the following three sequences:

$$G_2 \xrightarrow{\ell_1} \mathtt{A} \rightsquigarrow \mathtt{B} : a.\mathtt{A} \to \mathtt{C} : b.\mathsf{end} \xrightarrow{\ell_2} \mathtt{A} \to \mathtt{C} : b.\mathsf{end} \xrightarrow{\ell_3} \mathtt{A} \rightsquigarrow \mathtt{C} : b.\mathsf{end} \xrightarrow{\ell_4} \mathsf{end}$$

$$G_2 \xrightarrow{\ell_1} \mathtt{A} \rightsquigarrow \mathtt{B} : a.\mathtt{A} \to \mathtt{C} : b.\mathsf{end} \xrightarrow{\ell_3} \mathtt{A} \rightsquigarrow \mathtt{B} : a.\mathtt{A} \rightsquigarrow \mathtt{C} : b.\mathsf{end} \xrightarrow{\ell_2} \mathtt{A} \rightsquigarrow \mathtt{C} : b.\mathsf{end} \xrightarrow{\ell_4} \mathsf{end}$$

$$G_2 \xrightarrow{\ell_1} \mathtt{A} \rightsquigarrow \mathtt{B} : a.\mathtt{A} \to \mathtt{C} : b.\mathsf{end} \xrightarrow{\ell_3} \mathtt{A} \rightsquigarrow \mathtt{B} : a.\mathtt{A} \rightsquigarrow \mathtt{C} : b.\mathsf{end} \xrightarrow{\ell_4} \mathtt{A} \rightsquigarrow \mathtt{B} : a.\mathsf{end} \xrightarrow{\ell_2} \mathsf{end}$$

The last sequence is the most interesting: the sender A has to follow the syntactic order but the receiver C can get the message b before B receives a. The respect of these constraints is enforced by the conditions $\mathtt{p},\mathtt{q} \notin \mathit{subj}(\ell)$ and $\mathtt{q} \notin \mathit{subj}(\ell)$ in rules [GR4,5].

LTS over local types We define now the LTS over local types. This is done in two steps, following the model of CFSMs, where the semantics is given first for individual automata and then extended to communicating systems. We use the same labels $(\ell, \ell', \ldots)$ as the ones for CFSMs.

Definition 3.3 (LTS over local types). The relation $T \xrightarrow{\ell} T'$, for the local type of role $\mathtt{p}$, is defined as:

$$[\mathrm{LR1}]\ \mathtt{q}!\{a_i.T_i\}_{i\in I} \xrightarrow{\mathtt{pq}!a_i} T_i \quad [\mathrm{LR2}]\ \mathtt{q}?\{a_i.T_i\}_{i\in I} \xrightarrow{\mathtt{qp}?a_j} T_j \quad [\mathrm{LR3}]\ \frac{T[\mu\mathtt{t}.T/\mathtt{t}] \xrightarrow{\ell} T'}{\mu\mathtt{t}.T \xrightarrow{\ell} T'}$$

The semantics of a local type follows the intuition that every action of the local type should obey the syntactic order. We now define the LTS for collections of local types.

Definition 3.4 (LTS over collections of local types). A configuration $s = (\vec{T}; \vec{w})$ of a system of local types $\{T_\mathtt{p}\}_{\mathtt{p}\in\mathcal{P}}$ is a pair with $\vec{T} = (T_\mathtt{p})_{\mathtt{p}\in\mathcal{P}}$ and $\vec{w} = (w_{\mathtt{pq}})_{\mathtt{p}\neq\mathtt{q}\in\mathcal{P}}$ with $w_{\mathtt{pq}} \in \mathbb{A}^*$. We then define the transition system for configurations. For a configuration $s_T = (\vec{T}; \vec{w})$, the visible transitions of $s_T \xrightarrow{\ell} s'_T = (\vec{T}'; \vec{w}')$ are defined as:

1. $T_\mathtt{p} \xrightarrow{\mathtt{pq}!a} T'_\mathtt{p}$ and (a) $T'_{\mathtt{p}'} = T_{\mathtt{p}'}$ for all $\mathtt{p}' \neq \mathtt{p}$; and (b) $w'_{\mathtt{pq}} = w_{\mathtt{pq}} \cdot a$ and $w'_{\mathtt{p}'\mathtt{q}'} = w_{\mathtt{p}'\mathtt{q}'}$ for all $\mathtt{p}'\mathtt{q}' \neq \mathtt{pq}$; or
2. $T_\mathtt{q} \xrightarrow{\mathtt{pq}?a} T'_\mathtt{q}$ and (a) $T'_{\mathtt{p}'} = T_{\mathtt{p}'}$ for all $\mathtt{p}' \neq \mathtt{q}$; and (b) $w_{\mathtt{pq}} = a \cdot w'_{\mathtt{pq}}$ and $w'_{\mathtt{p}'\mathtt{q}'} = w_{\mathtt{p}'\mathtt{q}'}$ for all $\mathtt{p}'\mathtt{q}' \neq \mathtt{pq}$.

The semantics of local types is therefore defined over configurations, following the definition of the semantics of CFSMs. w_{pq} represents the FIFO queue at channel pq. We write $Tr(G)$ to denote the set of the visible traces that can be obtained by reducing G. Similarly for $Tr(T)$ and $Tr(S)$. We extend the trace equivalences $\approx$ and $\approx_n$ in § 2 to global types and configurations of local types.

We now state the soundness and completeness of projection w.r.t. the LTSs. The proof is given in Appendix A.1.

Theorem 3.1 (soundness and completeness). [3] *Let G be a global type with participants $\mathcal{P}$ and let $\vec{T} = \{G \upharpoonright \mathsf{p}\}_{\mathsf{p}\in\mathcal{P}}$ be the local types projected from G. Then $G \approx (\vec{T};\vec{\varepsilon})$.*

Local types and CFSMs Next we show how to algorithmically go from local types to CFSMs and back while preserving the trace semantics. We start by translating local types into CFSMs. Below we write $\mu\vec{\mathsf{t}}.T$ for $\mu\mathsf{t}_1.\mu\mathsf{t}_2...\mu\mathsf{t}_n.T$ with $n \geq 0$.

Definition 3.5 (translation from local types to CFSMs). Write $T' \in T$ if T' occurs in T. Let T_0 be the local type of participant p projected from G. The automaton corresponding to T_0 is $\mathcal{A}(T_0) = (Q, C, q_0, \mathbb{A}, \delta)$ where: (1) $Q = \{T' \mid T' \in T_0,\ T' \neq \mathsf{t}, T' \neq \mu\mathsf{t}.T\}$; (2) $q_0 = T_0'$ with $T_0 = \mu\vec{\mathsf{t}}.T_0'$ and $T_0' \in Q$; (3) $C = \{\mathsf{pq} \mid \mathsf{p},\mathsf{q} \in G\}$; (4) $\mathbb{A}$ is the set of $\{a \in G\}$; and (5) δ is defined as:

$$\text{If } T = \mathsf{p}'!\{a_j.T_j\}_{j\in J} \in Q, \text{ then } \begin{cases} (T,(\mathsf{pp}'!a_j),T') \in \delta & T_j = \mu\vec{\mathsf{t}}'.T', T' \in Q \\ (T,(\mathsf{pp}'!a_j),T') \in \delta & T_j = \mathsf{t},\ \mu\vec{\mathsf{t}}.T' \in T_0, \mathsf{t} \in \vec{\mathsf{t}}, T' \in Q \\ (T,(\mathsf{pp}'!a_j),T_j) \in \delta & \text{otherwise} \end{cases}$$

$$\text{If } T = \mathsf{p}'?\{a_j.T_j\}_{j\in J} \in Q, \text{ then } \begin{cases} (T,(\mathsf{p}'\mathsf{p}?a_j),T') \in \delta & T_j = \mu\vec{\mathsf{t}}'.T', T' \in Q \\ (T,(\mathsf{p}'\mathsf{p}?a_j),T') \in \delta & T_j = \mathsf{t},\ \mu\vec{\mathsf{t}}.T' \in T_0, \mathsf{t} \in \vec{\mathsf{t}}, T' \in Q \\ (T,(\mathsf{p}'\mathsf{p}?a_j),T_j) \in \delta & \text{otherwise} \end{cases}$$

The definition says that the set of states Q are the suboccurrences of branching or selection or end in the local type; the initial state q_0 is the occurrence of (the recursion body of) T_0; the channels and alphabets correspond to those in T_0; and the transition is defined from the state T to its body T_j with the action $\mathsf{pp}'!a_j$ for the output and $\mathsf{pp}'?a_j$ for the input. If T_j is a recursive type variable t, it points the state of the body of the corresponding recursive type.

Example 3.4. As an example, see C's local type in Example 3.1 is:

$$T = \mu\mathsf{t}.\mathsf{B}?\{\mathsf{sig}.\mathsf{A}?\{\mathsf{commit}.\mathsf{t}\},\ \mathsf{save}.\mathsf{A}?\{\mathsf{finish}.\mathsf{end}\}\}$$

Applying the translation gives the following transitions:

$$\begin{array}{l} (\mathsf{B}?\{\mathsf{sig}.\mathsf{A}?\{\mathsf{commit}.\mathsf{t}\},\ \mathsf{save}.\mathsf{A}?\{\mathsf{finish}.\mathsf{end}\}\}, (\mathsf{BC}?\mathsf{sig}), \mathsf{A}?\{\mathsf{commit}.\mathsf{t}\}) \\ (\mathsf{A}?\{\mathsf{commit}.\mathsf{t}\}, (\mathsf{AC}?\mathsf{commit}), \mathsf{B}?\{\mathsf{sig}.\mathsf{A}?\{\mathsf{commit}.\mathsf{t}\},\ \mathsf{save}.\mathsf{A}?\{\mathsf{finish}.\mathsf{end}\}\}) \\ (\mathsf{B}?\{\mathsf{sig}.\mathsf{A}?\{\mathsf{commit}.\mathsf{t}\},\ \mathsf{save}.\mathsf{A}?\{\mathsf{finish}.\mathsf{end}\}\}, (\mathsf{BC}?\mathsf{save}), \mathsf{AC}?\{\mathsf{finish}.\mathsf{end}\}) \\ (\mathsf{A}?\{\mathsf{finish}.\mathsf{end}\}, (\mathsf{AC}?\mathsf{finish}), \mathsf{end}) \end{array}$$

[3] The local type abstracts the behaviour of multiparty typed processes as proved in the subject reduction theorem in [23]. Hence this theorem implies that processes typed by global type G by the typing system in [23, 3] follow the LTS of G. In this article, we do not treat processes since we can use the same typing system with [3].

The corresponding automaton is drawn in Figure 2.

Local types are simple structures and their translation to CFSM preserves them.

Proposition 3.2 (local types to CFSMs). *Assume $T_{\mathtt{p}}$ is a local type. Then $\mathcal{A}(T_{\mathtt{p}})$ is deterministic, directed and has no mixed states.*

Proof. For the determinism, we note that all a_i in $\mathtt{p}?\{a_i.T_i\}_{i\in I}$ and $\mathtt{p}!\{a_i.T_i\}_{i\in I}$ are distinct. Directdness is by the syntax of branching and selection types. Finally, for non-mixed states, we can check a state is either sending or receiving state as one state represents either branching and selection type. □

We say that a CFSM is *basic* if it is deterministic, directed and has no mixed states. Any basic CFSM can be translated into a local type.

Definition 3.6 (translation from a basic CFSM to a local type). From a basic $M_{\mathtt{p}} = (Q, C, q_0, \mathbb{A}, \delta)$, we define the translation $\mathcal{T}(M_{\mathtt{p}})$ such that $\mathcal{T}(M_{\mathtt{p}}) = \mathcal{T}_{\varepsilon}(q_0)$ where $\mathcal{T}_{\tilde{q}}(q)$ is defined as:
(1) $\mathcal{T}_{\tilde{q}}(q) = \mu \mathtt{t}_q.\mathtt{p}'!\{a_j.\mathcal{T}^{\circ}_{\tilde{q}\cdot q}(q_j)\}_{j\in J}$ if $(q, \mathtt{p}\mathtt{p}'!a_j, q_j) \in \delta$;
(2) $\mathcal{T}_{\tilde{q}}(q) = \mu \mathtt{t}_q.\mathtt{p}'?\{a_j.\mathcal{T}^{\circ}_{\tilde{q}\cdot q}(q_j)\}_{j\in J}$ if $(q, \mathtt{p}'\mathtt{p}?a_j, q_j) \in \delta$;
(3) $\mathcal{T}^{\circ}_{\tilde{q}}(q) = \mathcal{T}_{\varepsilon}(q) = \mathsf{end}$ if q is final; (4) $\mathcal{T}^{\circ}_{\tilde{q}}(q) = \mathtt{t}_{q_k}$ if $(q, \ell, q_k) \in \delta$ and $q_k \in \tilde{q}$; and
(5) $\mathcal{T}^{\circ}_{\tilde{q}}(q) = \mathcal{T}_{\tilde{q}}(q)$ otherwise.

Finally, we replace $\mu \mathtt{t}.T$ by T if $\mathtt{t}$ is not in T.

In $\mathcal{T}_{\tilde{q}}$, $\tilde{q}$ records visited states; (1,2) translate the receiving and sending states to branching and selection types, respectively; (3) translates the final state to end; and (4) is the case of a recursion: since q_k was visited, ℓ is dropped and replaced by the type variable.

The following proposition states that these translations preserve the semantics.

Proposition 3.3 (translations between CFSMs and local types). *If a CFSM M is basic, then $M \approx \mathcal{T}(M)$. If T is a local type, then $T \approx \mathcal{A}(T)$.*

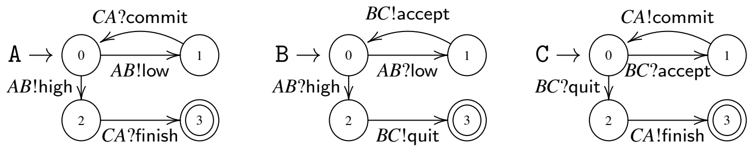
Proof. The first clause is by the induction of M using the translation of $\mathcal{T}$. The second clause is by the induction of T using the translation of $\mathcal{A}$. Both are mechanical. □

4 Sequential multiparty session types and sequential communicating automata

This section examines the soundness and completeness characterisation of sequential multiparty sessions, defined by the projection from sequential global types, and which correspond to a subset of systems of basic CFSMs. Sequential automata do not feature any concurrency: at any point of the execution, there is only one role that can do a send or a receive action. This class has been used as a specification language for the generation of cryptographic protocols [4], as well as modellings radio communications where transceivers of several machines use the same frequency [10].

Definition 4.1 (sequential CFSM). A CFSM $M = (Q, C, q_0, \mathbb{A}, \delta)$ is said to be *sequential* if it is basic and all its states in Q are alternating or final. A communicating system of CFSMs is *sequential* if all the machines are sequential and if all but one of the starting states are receiving states.

Example 4.1 (sequential system). We give a simple sequential system which is extended from the binary example in § 1.



The corresponding global type is:

$$G_3 \ = \ \mu t.\mathtt{A} \to \mathtt{B} : \{ \ \mathsf{low}.\mathtt{B} \to \mathtt{C} : \mathsf{accept}.\mathtt{C} \to \mathtt{A} : \mathsf{commit}.t \\ \mathsf{high}.\mathtt{B} \to \mathtt{C} : \mathsf{quit}.\mathtt{C} \to \mathtt{A} : \mathsf{finish}.\mathsf{end}\}$$

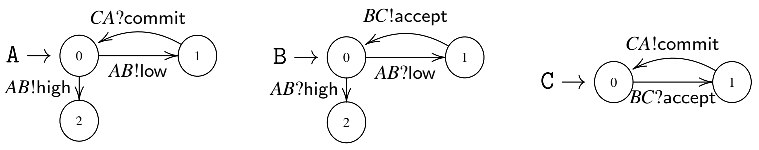
Sequential systems always start with a unique machine able to output. The alternation allows to prove that all accepted traces of sequential systems are 1-bounded.

Proposition 4.1 (boundedness of sequential systems). *Sequential systems are 1-bounded.*

Proposition 4.2 (safety and liveness in sequential systems). *For sequential systems, the safety and liveness properties of Definition 2.4 are decidable.*

Proof. By Proposition 4.1, sequential systems are 1-bounded and therefore finite. The reachability problem of 1-bounded system is decidable with a complexity that is polynomial in the number of states and of transitions [10, Theorem 26]. □

Example 4.2 (properties of sequential systems). We show an example of a sequential system which does not satisfy the safety properties. We change machines in the previous example:



Then the system is sequential but does not satisfy deadlock-freedom since, if high is selected, then C should wait forever. The corresponding global type is:

$$\mu t.\mathtt{A} \to \mathtt{B}\{ \ \mathsf{low}.\mathtt{B} \to \mathtt{C} : \mathsf{accept}.\mathtt{C} \to \mathtt{A} : \mathsf{commit}.t \\ \mathsf{high}.\mathsf{end}\}$$

which does not satisfy the mergeability condition: it is not well-formed. If we replace only A by the above automaton, but keep B and C from Example 4.1, then it can have an orphan message $CA!\mathsf{finish}$ in the queue. In Theorem 4.3, we show if the sequential system satisfies deadlock-freedom and orphan-message freedom, then it coincides with the well-formed sequential global type defined below.

We define sequential global (and local) types, and show the trace equivalence with sequential communicating systems. Below the 1-unfolding means unfolding each recursion once [23].

Definition 4.2 (sequential global type). A well-formed global type G is *sequential* if:

1. there is no parallel composition; and
2. for any subterm $\mathtt{p}_1 \to \mathtt{p}_2 \colon \{a_j.\mathtt{p}_{3i} \to \mathtt{p}_{4i} \colon \{b_i.G_i\}_{i\in I}\}_{j\in J} \in G'$ where G' is the 1-unfolding of G, then $\mathtt{p}_2 = \mathtt{p}_{3i}$ for all $i \in I$.

In sequential global types, the receiver of a message is always either terminating, or the next sender. The definition relies on 1-unfolding to adapt the definition to recursive types. The following proposition could be an alternative definition to sequential global types based on alternating transitions in reductions.

Proposition 4.3 (sequential global type). *Suppose G is sequential. Then, for any sequence of transitions $G \xrightarrow{\varphi} G_0 \xrightarrow{\ell_1} G_1 \xrightarrow{\ell_2} G_2$, we have either:*

- $\ell_1 = \mathtt{p}_1\mathtt{p}_1'!a$ *and* $\ell_2 = \mathtt{p}_2\mathtt{p}_2'?b$ *with* $\mathtt{p}_1 = \mathtt{p}_2$ *and* $\mathtt{p}_1' = \mathtt{p}_2'$ *and* $a = b$*; or*
- $\ell_1 = \mathtt{p}_1\mathtt{p}_1'?a$ *and* $\ell_2 = \mathtt{p}_2\mathtt{p}_2'!b$ *with* $\mathtt{p}_1' = \mathtt{p}_2$.

First we prove the soundness.

Theorem 4.1 (soundness in sequential systems). *If G is sequential, there exists a sequential system S which satisfies the safety properties (deadlock-freedom, reception error-freedom and orphan message-freedom) in Definition 2.4. and $S \approx G$.*

Proof. We first prove if G is sequential, then S is sequential by induction on G. The case $G = \mathsf{end}$ is trivial. Suppose $\mathtt{p}_1 \to \mathtt{p}_2 \colon \{a_j.\mathtt{p}_2 \to \mathtt{p}_{3i} \colon \{b_i.G_i\}_{i\in I}\}_{j\in J} = G'$ where G' is the 1-unfolding of G. There are two cases.

Case 1 $\mathtt{p}_1 \neq \mathtt{p}_{3i}$: By the definition of mergeability, $\mathtt{p}_{3i} = \mathtt{p}_{3j} = \mathtt{p}_3$. Let $T_i = (\mathtt{p}_2 \to \mathtt{p}_{3i} \colon \{b_i.G_i\}_{i\in I}) \upharpoonright \mathtt{p}_{3i}$. Then by definition, $\mathcal{A}(T_i)$ starts from a receiving state with the labels $\{b_i\}_{i\in I}$, and by induction, $\mathcal{A}(T_i)$ is alternating. $\mathcal{A}(G \upharpoonright \mathtt{p}_1)$ is only an initial sending state with labels $\{a_j\}$ and $\mathcal{A}(G \upharpoonright \mathtt{p}_2)$ has two states whose initial state is receiving from $\mathtt{p}_1$ with labels $\{a_j\}$ and sending to $\mathtt{p}_3$ with labels $\{b_i\}$. Similarly by induction, for all $\mathtt{p}'$, $\{\mathcal{A}(G_i \upharpoonright \mathtt{p}')\}_{\mathtt{p}'}$ is a set of automata whose initial states are receiving with alternation or final. Hence $\{\mathcal{A}(G \upharpoonright \mathtt{p}')\}_{\mathtt{p}'}$ is sequential.

Case 2 $\mathtt{p}_1 = \mathtt{p}_{3i}$ for some i: Similar with Case 1 noting $\mathcal{A}(G \upharpoonright \mathtt{p}_1)$ has a sending initial state with labels $\{a_j\}$ which connects to receiving states from $\mathtt{p}_2$ with labels $\{b_i\}$.

The second half (safety) is immediate from Theorem 3.1 and Propositions 3.3 and 4.3. □

Now that we have identified exactly the sequential systems and sequential global types, we can turn to showing their equivalence. We start by the synthesis algorithm.

Theorem 4.2 (synthesis of sequential systems [11]). *If a system $S = (M_\mathtt{p})_{\mathtt{p}\in\mathcal{P}}$ is sequential, then there is an algorithm which successfully builds G such that $G \approx S$ if such G exists, and otherwise terminates.*

Proof. The algorithm starts from the initial states of all machines $(q^{\mathtt{p}_1}{}_0, ..., q^{\mathtt{p}_n}{}_0)$: we know that only one of them is a sending state. We apply the algorithm with the invariant that all buffers are empty, all machines are in receiving states except one. We define $G(q_1, ..., q_n)$, where we note q_k to be the sending state (corresponding to machine $\mathtt{p}$), as follows:

1. if $(q_1, ..., q_n)$ has been visited before, the global type is $\mathtt{t}_{q_1,...,q_n}$;
2. if all q_i in $(q_1, ..., q_n)$ are final, we set to end.
3. otherwise, in q_k, from machine $\mathtt{p}$, we know that all the transitions are sending actions towards $\mathtt{p}'$ (by directedness), i.e. of the form $(q_k, \mathtt{p}\mathtt{p}'!a_i, q_i) \in \delta_{\mathtt{p}}$ for $i \in I$.
   (a) we check that machine $\mathtt{p}'$ is in a receiving state q_m such that $(q_m, \mathtt{p}\mathtt{p}'?a_j, q'_j) \in \delta_{\mathtt{p}'}$ with $j \in J$ and $I \subseteq J$. Otherwise the algorithm returns false.
   (b) we check that each q'_i $(i \in I)$ in machine $\mathtt{p}'$ are sending states or final. Otherwise the algorithm returns false.
   (c) we set $\mu \mathtt{t}_{q_1,...,q_n}.\mathtt{p} \to \mathtt{p}' \colon \{a_i.G(q_1, ..., q_k \leftarrow q_i, ..., q_m \leftarrow q'_i, ..., q_n)\}_{i \in I}$ (where $q_k \leftarrow q_i$ means we replace q_k by q_i) and continue by recursive calls. If q'_i is final, we set it to end.
4. we erase unnecessary $\mu\mathtt{t}$ if $\mathtt{t}$ does not appear in G. □

Example 4.3 (synthesis of sequential systems). We show how to build a global type from the three sequential automata in Example 4.1. We denote, e.g. $q^{\mathtt{A}}_i$ for $\mathtt{A}$'s i-state.

1. We pick up the initial sending state $q^{\mathtt{A}}_0$ and its receiving state $q^{\mathtt{B}}_0$. We build the type $\mu \mathtt{t}_{q^{\mathtt{A}}_0, q^{\mathtt{B}}_0, q^{\mathtt{C}}_0}.\mathtt{A} \to \mathtt{B} \colon \{\mathsf{low}.S_1,\ \mathsf{high}.S_2\}$ and go to S_1.
2. From S_1, $q^{\mathtt{B}}_1$ is a sending state, and we find $q^{\mathtt{C}}_0$ as its receiving state. Hence we build $\mu \mathtt{t}_{q^{\mathtt{A}}_1, q^{\mathtt{B}}_1, q^{\mathtt{C}}_0}.\mathtt{B} \to \mathtt{C} \colon \mathsf{accept}.S_3$.
3. From S_3, $q^{\mathtt{C}}_1$ is a sending state, and we find $q^{\mathtt{A}}_1$ as its receiving state. Hence we denote $\mu \mathtt{t}_{q^{\mathtt{A}}_1, q^{\mathtt{B}}_0, q^{\mathtt{C}}_1}.\mathtt{C} \to \mathtt{A} \colon \mathsf{commit}.S_4$.
4. We go back S_2. Then $q^{\mathtt{B}}_2$ is a sending state, and we find $q^{\mathtt{C}}_0$ as its receiving state. Hence we denote $\mu \mathtt{t}_{q^{\mathtt{A}}_2, q^{\mathtt{B}}_2, q^{\mathtt{C}}_0}.\mathtt{B} \to \mathtt{C} \colon \mathsf{quit}.S_5$.
5. From S_5, $q^{\mathtt{C}}_2$ is a sending state, and we find $q^{\mathtt{A}}_2$ as its receiving state. Hence we denote $\mu \mathtt{t}_{q^{\mathtt{A}}_2, q^{\mathtt{B}}_3, q^{\mathtt{C}}_2}.\mathtt{C} \to \mathtt{A} \colon \mathsf{finish}.S_6$.
6. Since S_6 goes to the final states $(q^{\mathtt{A}}_3, q^{\mathtt{B}}_3, q^{\mathtt{C}}_3)$, we set it to end.
7. We delete all $\mu \mathtt{t}_{q^{\mathtt{A}}_i, q^{\mathtt{B}}_i, q^{\mathtt{C}}_i}$ except $\mu \mathtt{t}_{q^{\mathtt{A}}_0, q^{\mathtt{B}}_0, q^{\mathtt{C}}_0}$ and finish.

We prove that the synthesis is correct and preserves the traces.

Theorem 4.3 (completeness in sequential systems). *If $S = (M_{\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$ is sequential, orphan message-free and deadlock-free, then there exists sequential G such that $S \approx G$.*

Proof. By Proposition 4.1, the sequential system S terminates either with 1 message in one queue or all are empty. Another possibility is deadlock. Theorem 4.2 implies if S is deadlock-free and orphan message-free, the algorithm always returns a well-formed sequential G; otherwise terminates by Step 2-(a) and (b). Hence by Theorem 3.1 and Proposition 3.3 by setting $M'_{\mathtt{p}} = \mathcal{A}(G \upharpoonright \mathtt{p})$, we have $G \approx \{\mathcal{A}(G \upharpoonright \mathtt{p})\}_{\mathtt{p} \in \mathcal{P}} \approx S$. □

5 Multiparty compatibility and soundness

This section defines multiparty compatibility and proves the soundness theorem: the CFSMs which are equivalent to projected local types satisfy the multiparty compatibility. As stated in Theorem 2.1, restricting basic CFSMs to the natural generalisation of half-duplex systems [10, § 4.1.1] is not sufficient to fully characterise multiparty session types. We need a stronger (and decidable) property as the multiparty compatibility to force basic CFSMs to behave as if they were the result of a projection from global types. We then show that multiparty compatible automata satisfy the three safety properties. This result, together with soundness and Theorem 3.1, implies that all well-formed global types satisfy the safety properties, i.e. all traces made by a global type are safe.

Multiparty compatibility In the two machines case, there exists a sound and complete condition called *compatible* [20]. The idea of the extension to the multiparty case comes from the observation that from the viewpoint of the participant p, the rest of all the machines $(M_{\mathtt{q}})_{\mathtt{q}\in\mathcal{P}\setminus\mathtt{p}}$ should behave as if they were one CFSM which offers compatible traces, up to internal synchronisations (i.e. 1-bounded executions). Below we define a way to group CFSMs.

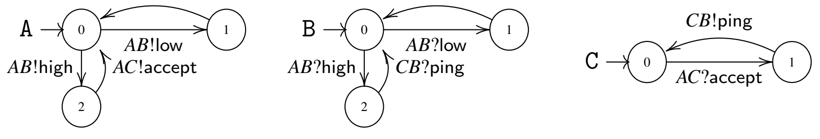
We now define a notion of compatibility extended to more than two CFSMs. We say that φ is an *alternation* if φ is an alternation of sending and corresponding receive actions (i.e. the action $\mathtt{pq}!a$ is immediately followed by $\mathtt{pq}?a$) or an empty sequence.

Definition 5.1 (multiparty compatible system). A system $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ is *multiparty compatible* if for any 1-bounded reachable stable state $s \in RS_1(S)$,

1. for any output action $\mathtt{pq}!a$ from s in $M_{\mathtt{p}}$, there exists an alternation $\varphi \cdot t$ from s in a system where $act(t) = \mathtt{pq}?a$ and $\mathtt{p} \notin act(\varphi)$; and
2. for any input action $\mathtt{qp}?a$ from s in $M_{\mathtt{p}}$, there exists an input action $\mathtt{qp}?b$ and an alternation $\varphi \cdot t$ from s where $act(t) = \mathtt{qp}!b$ and $\mathtt{p} \notin act(\varphi)$.

In above, $\mathtt{p} \notin act(\varphi)$ means φ does not contain actions to or from channel p. The definition states that for each $M_{\mathtt{p}}$, the rest of machines can produce the compatible (dual) actions by executing alternations. From $M_{\mathtt{p}}$, these intermediate alternations can be seen as non-observable internal actions. The asymmetry between output and input actions is due to branching subtyping [13]; the rest of the machines must be able to receive every output from a single machine, while the rest of the machines is only required to be able to output one message that is expected by *one of inputs* of a single machine. The former guarantees orphan message freedom, while the latter ensures deadlock-freedom. See the examples below.

Example 5.1 (multiparty compatibility). We first give a simple multiparty compatible system which is a modification of Example 4.1.



The corresponding global type is:

$$\begin{aligned} G_4 \;=\; \mu \mathtt{t}.\mathtt{A} \to \mathtt{B} : \{\; & \mathsf{low}.\mathtt{A} \to \mathtt{C} : \;\; \mathsf{accept}.\mathtt{C} \to \mathtt{B} : \mathsf{ping}.\mathtt{t}, \\ & \mathsf{high}.\mathtt{A} \to \mathtt{C} : \mathsf{accept}.\mathtt{C} \to \mathtt{B} : \mathsf{ping}.\mathtt{t} \;\} \end{aligned}$$

Note that the communicating automata and the global type are not sequential but well-formed.

We detail here how to check the compatibility from the point of view of A. To check the compatibility for the actions $act(t_1 \cdot t_2) = AB!\mathsf{high} \cdot AC!\mathsf{accept}$, the only possible actions are $AB?\mathsf{high}$ from B (a dual of $act(t_1)$); and $AC?\mathsf{accept}$ from C. Similarly for the actions $AB!\mathsf{low} \cdot AC!\mathsf{accept}$.

We then check the compatibility from the point of view of B. To check the compatibility for the actions $act(t_3 \cdot t_4) = AB?\mathsf{high} \cdot CB?\mathsf{ping}$, the actions are $AB!\mathsf{high}$ from A; then $AC!\mathsf{accept} \cdot AC?\mathsf{accept} \cdot CB!\mathsf{ping}$ from A and C where $AC!\mathsf{accept} \cdot AC?\mathsf{accept}$ is a 1-bounded execution.

We can verify similarly for $AB?\mathsf{low}$ from B, and the multiparty compatibility from the view point of C.

Remark 5.1 (multiparty compatibility without mergeability). The multiparty compatibility of Example 5.1 does not require the asymmetry between the output and input actions. The input case is defined as the same as the output case, i.e. Definition 5.1(2) is replaced as follows:

2. for any input action $\mathtt{qp}?a$ from s in $M_{\mathtt{p}}$, there exists an alternation $\varphi \cdot t$ from s where $act(t) = \mathtt{qp}!a$ and $\mathtt{p} \notin act(\varphi)$.

This is because an input from the state 0 of C has always a dual output from both states 1 and 2 in A. This input-output symmetric version corresponds to the projection rule in [23] which is given by replacing the third case $(\mathtt{q} \neq \mathtt{p}, \mathtt{q} \neq \mathtt{p}')$ of the branching case (meageablity) in Definition 3.1 with:

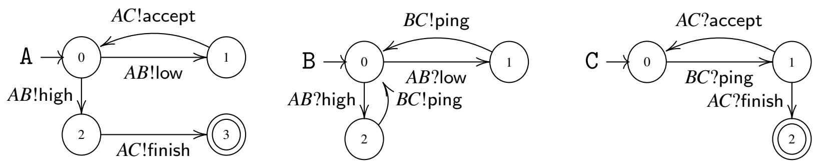
$$\mathtt{p} \to \mathtt{p}' : \{a_j.G_j\}_{j \in J} \upharpoonright \mathtt{q} = G_1 \upharpoonright \mathtt{q} \text{ where } \; G_i \upharpoonright \mathtt{q} = G_j \upharpoonright \mathtt{q}$$

Note that this rule is subsumed by Definition 3.1. See § 7 for more discussions.

Example 5.2 (multiparty compatibility for Figure 2). We can formally give the multiparty compatibility property on the commit example of Figure 2. To check the compatibility from the point of view of A, we check the actions $act(t_1 \cdot t_2) = AB!\mathsf{quit} \cdot AC!\mathsf{finish}$, the only possible action is $AB?\mathsf{quit}$ (a dual of $act(t_1)$) from B, then a 1-bounded execution is $BC!\mathsf{save} \cdot BC?\mathsf{save}$, and $AC?\mathsf{finish}$ (a dual of $act(t_2)$) from C. To check the compatibility for the actions $act(t_3 \cdot t_4) = AB!\mathsf{act} \cdot AC!\mathsf{commit}$, $AB?\mathsf{act}$ (a dual of t_3) from B, the 1-bounded execution is $BC!\mathsf{sig} \cdot BC?\mathsf{sig}$, and $AC?\mathsf{commit}$ (a dual of $act(t_4)$) from C.

Example 5.3 (multiparty compatibility with meargeablity). We explain the reason of the input multiparty compatibility in Definition 5.1(2). We revise Example 5.1 by replacing

A and C's CFSMs. The corresponding global type is given below:



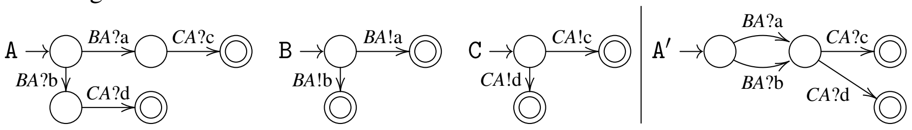
The corresponding global type is:

$$G_5 \;=\; \mu\mathtt{t}.\mathtt{A} \to \mathtt{B} : \{\; \mathsf{low}.\mathtt{B} \to \mathtt{C} : \mathsf{ping}.\mathtt{A} \to \mathtt{C} : \mathsf{accept}.\mathtt{t} \\ \qquad\qquad\qquad \mathsf{high}.\mathtt{B} \to \mathtt{C} : \mathsf{ping}.\mathtt{A} \to \mathtt{C} : \mathsf{finish}.\mathsf{end}\}$$

We observe that A performs only "accept" following "low" or only "finish" following "high". This automaton is given by using the mergeability projection rule from G_5.

We now verify the asymmetric rule for the input multiparty compatibility. For this purpose, we select C. Assume the 1-bounded execution $AB!\mathsf{low} \cdot AB?\mathsf{low} \cdot BC!\mathsf{ping} \cdot BC?\mathsf{ping}$. Then from the viewpoint of C, state 1 can execute $AC?\mathsf{finish}$. We note that a dual action $AC!\mathsf{finish}$ is not enable from state 1 of A (which was possible in Example 5.1). However there exits action $AC!\mathsf{accept}$ from state 1 of A which is dual to $AC?\mathsf{accept}$ from the same state 1 in C. Similarly for the case $AB!\mathsf{high} \cdot AB?\mathsf{high} \cdot BC!\mathsf{ping} \cdot BC?\mathsf{ping}$. In this way, one can verify the multiparty compatibility from C. Similarly we can check Example 3.2 is multiparty compatible.

Remark 5.2. In Definition 5.1, we check the compatibility from any 1-bounded reachable stable state in the case one branch is selected by different senders. Consider the following machines:



In A, B and C, each action in each machine has its dual but they do not satisfy multiparty compatibility. For example, if $BA!\mathrm{a} \cdot BA?\mathrm{a}$ is executed, $CA!\mathrm{d}$ does not have a dual action (hence they do not satisfy the safety properties). On the other hand, the machines A′, B and C satisfy the multiparty compatibility.

The next proposition states that after some appropriate executions, *any* reachable state can be translated into a 1-bounded execution. See § 2 for the definition of a stable property.

Proposition 5.1 (stable property). *Assume $S = (M_{\mathrm{p}})_{\mathrm{p}\in\mathcal{P}}$ is basic and multiparty compatible. Then S satisfies the stable property, i.e. if, for all $s \in RS(S)$, there exists an execution $\xrightarrow{\varphi}$ such that $s\xrightarrow{\varphi}s'$ and s' is stable, and there is a 1-bounded execution $s_0\xrightarrow{\varphi'}s'$.*

Proof. The proof is non-trivial using a detailed analysis of causal relations to translate into a 1-bounded executions. See Appendix B. □

Theorem 5.1 (safety and liveness). *Assume $S = (M_{\mathrm{p}})_{\mathrm{p}\in\mathcal{P}}$ is basic and multiparty compatible. Then S satisfies the three safety properties in Definition 2.4. Further, if there exists at least one M_{q} which includes a final state, then S satisfies the liveness property.*

Proof. The orphan message- and the reception error-freedom are the corollary of Proposition 5.1. The deadlock-freedom is proved by the stable property and multiparty compatibility. Liveness is a consequence of the orphan message- and deadlock-freedom. Thus we only have to prove the deadlock-freedom. Assume S is basic and satisfy the multiparty session compatibility. Hence S satisfies the stable property. Thus we only have to check for all $s \in RS_1(S)$, s is not deadlock. Suppose by the contradiction, s contains the receiving states t from the input state q of machine p. Then by the multiparty compatibility, there exists 1-bounded execution φ such that $s \xrightarrow{\varphi}\xrightarrow{t'} s'$ such that t' is dual of one of the inputs from q. Write t'' for such input. Hence $s \xrightarrow{\varphi}\xrightarrow{t'}\xrightarrow{t''} s''$ and s'' is stable and s'' should satisfy the multiparty compatibility again. We apply for the all input states except p enable from s'' repeatedly, which eventually leads to s^n with no input state which is stable. This contradicts the assumption. Thus we conclude the proof of the deadlock-freedom. □

Now we prove the soundness of the projection algorithm.

Theorem 5.2 (soundness). *Let G be a global type with participants $\mathcal{P}$ and $\vec{T} = \{G \upharpoonright \mathrm{p}\}_{\mathrm{p}\in\mathcal{P}}$ be the local types projected from G. Then $(\mathcal{A}(T_\mathrm{p}))_{\mathrm{p}\in\mathcal{P}}$ is multiparty compatible.*

Proof. By Theorem 3.1, we only have to check that the transitions generated by the LTS rules of global types satisfy the multiparty compatibility. See Appendix C.

6 Synthesis and completeness

This section studies the synthesis and the complete characterisation of multiparty session types as communicating automata.

We first state that multiparty compatibility is decidable.

Proposition 6.1. *If all the CFSMs M_p ($\mathrm{p} \in \mathcal{P}$) are basic, there is an algorithm to check whether $(M_\mathrm{p})_{\mathrm{p}\in\mathcal{P}}$ is multiparty compatible.*

Proof. The algorithm to check M_p's compatibility with $S^{-\mathrm{p}}$ is defined using the set $RS_1(S)$ of reachable states using 1-bounded executions. Note that the set $RS_1(S)$ is computable [9, 10]. We start from $q = q_0$ and the initial configuration $s = s_0$. Suppose that, from q, we have the transitions $t_i = (q, \mathrm{qp}!a_i, q'_i) \in \delta_\mathrm{p}$. We then construct $RS_1(S)$ (without executing p) until it includes s' such that $\{s' \xrightarrow{t_i}\xrightarrow{t'_i} s_j\}_{j\in J}$ where $act(t'_i) = \mathrm{qp}?a_i$ and $I \subseteq J$. If there exists no such s', it returns false and terminates. The case where, from q, we have receiving transitions $t = (q, \mathrm{qp}?a_i, q'_i)$, we check whether there exists $t' = (q, \mathrm{qp}?a_j, q'_j)$ such that, after the 1-bounded execution, there is a dual output. If it does not fail, we continue to check from state q'_i and configuration s_i for each $i \in I$. We repeat this procedure until we visit all $q \in Q_\mathrm{p}$. Repeat for the other machines p' such that $\mathrm{p}' \in \mathcal{P}\setminus \mathrm{p}$. Then we repeat this procedure for all stable $s \in RS_1(S)$. □

Synthesis Below we state the lemma which will be crucial for the proof of synthesis and completeness. The lemma comes from the intuition that the transitions of multiparty compatible systems are always permutations of one-bounded executions as it is the case in multiparty session types.

Lemma 6.1 (1-buffer equivalence). *Suppose S_1 and S_2 are two basic and multiparty compatible communicating systems such that $S_1 \approx_1 S_2$, then $S_1 \approx S_2$.*

Proof. We prove that $\forall n, S_1 \approx_n S_2 \implies S_1 \approx_{n+1} S_2$. Then the lemma follows. We assume $S_1 \approx_n S_2$ and then prove by induction on the length of an execution φ in S_1, that it is accepted by S_2. If $|\varphi| < n+1$, then the buffer usage of φ for S_1 cannot exceed n, therefore S_2 can realise φ since $S_1 \approx_n S_2$. Assume $|\varphi| = k+1$ and that the property holds for traces of length k or less. We apply a case analysis on φ and use multiparty compatibility to know the existence of matching receives for unmatched sends. By permutation, we are able to reduce the size of buffer used in S_1 and apply the induction hypothesis. By using the permutation backwards in S_2 we can conclude. See Appendix D. $\square$

Theorem 6.1 (synthesis). *Suppose S is a basic system and multiparty compatible. Then there is an algorithm which successfully builds well-formed G such that $S \approx_1 G$ and terminates.*

Proof. We assume $S = (M_{\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$. We first divide S into $S_1, S_2, ..., S_m$ where a set of participants appear in S_i are pairwise disjoint. Once we construct G_i from S_i, we obtain $G_1 \mid G_2 \mid ... \mid G_n$.

We show the algorithm which constructs G from S_i. The algorithm starts from the initial states of the machines S_i $({q^{\mathtt{p}_1}}_0, ..., {q^{\mathtt{p}_n}}_0)$. We take a pair of the initial states which is a sending state $q_0^{\mathtt{p}}$ and a receiving state $q_0^{\mathtt{q}}$ from $\mathtt{p}$ to $\mathtt{q}$. We note that by directness, if there are more than two pairs, by [G4] in Definition 3.2, the order does not matter. We apply the algorithm with the invariant that all buffers are empty and that we repeatedly pick up one pair such that $q_{\mathtt{p}}$ (sending state) and $q_{\mathtt{q}}$ (receiving state). We define $G(q_1, ..., q_n)$ where $(q_{\mathtt{p}}, q_{\mathtt{q}} \in \{q_1, ..., q_n\})$ as follows:

1. if $(q_1, ..., q_n)$ has already been examined and if all participants have been involved since then (or the ones that have not are in their final state), we set $G(q_1, ..., q_n)$ to be $\mathtt{t}_{q_1,...,q_n}$.
2. if $(q_1, ..., q_n)$ are all final states, we set $\mathtt{end}$.
3. otherwise, we select a pair sender/receiver from two participants that have not been involved (and are not final) and go to the next step;
4. in $q_{\mathtt{p}}$, from machine $\mathtt{p}$, we know that all the transitions are sending actions towards $\mathtt{p}'$ (by directedness), i.e. of the form $(q_{\mathtt{p}}, \mathtt{pq}!a_i, q_i) \in \delta_{\mathtt{p}}$ for $i \in I$.
   (a) we check that machine $\mathtt{q}$ is in a receiving state $q_{\mathtt{q}}$ such that $(q_{\mathtt{q}}, \mathtt{pq}?a_j, q'_j) \in \delta_{\mathtt{p}'}$ with $j \in J$ and $I \subseteq J$.
   (b) we set $\mu \mathtt{t}_{q_1,...,q_n}.\mathtt{p} \to \mathtt{q}\colon \{a_i.G(q_1, ..., q_{\mathtt{p}} \leftarrow q_i, ..., q_{\mathtt{q}} \leftarrow q'_i, ..., q_n)\}_{i \in I}$ (we replace $q_{\mathtt{p}}$ and $q_{\mathtt{q}}$ by q_i and q'_i, respectively) and continue by recursive calls.
   (c) if all sending states in $q_1, ..., q_n$ become final, then we set $G(q_1, ..., q_n) = \mathtt{end}$.
5. we erase unnecessary $\mu\mathtt{t}$ if $\mathtt{t} \notin G$. $\square$

We note that the algorithm for the sequential systems in Theorem 4.2 checks that the deadlock-freedom and orphan message freedom while building a sequential global type. On the other hand, by the assumption of the multiparty compatibility, the algorithm in Theorem 6.1 does not require this check. In the sequential system, we can easily combine the synthesis and safety-checking since only one active sender and receiver appear at the same time. We also note that the sequential system explores only the 1-bounded execution so that building $G \approx_1 S$ means $G \approx S$ in the sequential system.

In a general case studied in § 5, since the above algorithm only explores 1-bounded executions, the reconstructed G only satisfies $G \approx_1 S$. Thus we need to prove the following property.

Corollary 6.1 (synthesis for all traces). *Suppose S is a basic system and multiparty compatible and G is built by the algorithm in Theorem 6.1 from S. Then $G \approx S$.*

Proof. By the construction of the algorithm, the projection $\{G \upharpoonright \mathtt{p}\}_{\mathtt{p}\in\mathcal{P}}$ is defined. By Theorem 3.1, we know that $G \approx (\{G \upharpoonright \mathtt{p}\}_{\mathtt{p}\in\mathcal{P}}; \vec{\varepsilon})$. Hence, by Proposition 3.3, we have $G \approx S'$ where $S' = \{\mathcal{A}(G \upharpoonright \mathtt{p})\}_{\mathtt{p}\in\mathcal{P}}$, the communicating system translated from the projected local types $\{G \upharpoonright \mathtt{p}\}_{\mathtt{p}\in\mathcal{P}}$ of G. By $S \approx_1 G$, we know $S \approx_1 S'$. Hence by Lemma 6.1, $S \approx S'$ and therefore $S \approx G$. □

Example 6.1 (synthesis of basic and multiparty compatible systems). We show the algorithm can generate the global type in Example 3.1 from CFSMs in Figure 2.The steps are similar with Example 4.3. As in Example 4.3, we denote e.g. $q_i^{\mathtt{A}}$ for A's i-state.

1. We pick up the initial sending state $q_0^{\mathtt{A}}$ and its receiving state $q_0^{\mathtt{B}}$. We build the type $\mu \mathtt{t}_{q_0^{\mathtt{A}}, q_0^{\mathtt{B}}, q_0^{\mathtt{C}}}.\mathtt{A} \to \mathtt{B}\colon \{\mathsf{act}.S_1,\ \mathsf{quit}.S_2\}$ and go to S_1.
2. From S_1, $q_1^{\mathtt{B}}$ is a sending state, and we find $q_0^{\mathtt{C}}$ as its receiving state. Hence we build $\mu \mathtt{t}_{q_1^{\mathtt{A}}, q_1^{\mathtt{B}}, q_0^{\mathtt{C}}}.\mathtt{B} \to \mathtt{C}\colon \mathsf{sig}.S_3$.
3. From S_3, $q_1^{\mathtt{A}}$ is a sending state, and we find $q_1^{\mathtt{C}}$ as its receiving state. Hence we build $\mu \mathtt{t}_{q_1^{\mathtt{A}}, q_0^{\mathtt{B}}, q_0^{\mathtt{C}}}.\mathtt{A} \to \mathtt{C}\colon \mathsf{commit}.S_4$.
4. The next state for S_4 is an initial state $(q_0^{\mathtt{A}}, q_0^{\mathtt{B}}, q_0^{\mathtt{C}})$ which was visited before. Hence by Step 1, we set $S_4 = \mathtt{t}_{q_0^{\mathtt{A}}, q_0^{\mathtt{B}}, q_0^{\mathtt{C}}}$.
5. From S_2, $q_2^{\mathtt{B}}$ is a sending state and $q_0^{\mathtt{C}}$ is a receiving state. Hence we build the type $\mu \mathtt{t}_{q_2^{\mathtt{A}}, q_2^{\mathtt{B}}, q_0^{\mathtt{C}}}.\mathtt{B} \to \mathtt{C}\colon \mathsf{save}.S_5$.
6. From S_5, $q_2^{\mathtt{A}}$ is a sending state and $q_2^{\mathtt{C}}$ is a receiving state. Hence we build the type $\mu \mathtt{t}_{q_2^{\mathtt{A}}, q_3^{\mathtt{B}}, q_2^{\mathtt{C}}}.\mathtt{A} \to \mathtt{C}\colon \mathsf{finish}.S_6$.
7. Since S_6 goes to the final states $(q_3^{\mathtt{A}}, q_3^{\mathtt{B}}, q_3^{\mathtt{C}})$, we set S_6 to end.
8. We delete all $\mu \mathtt{t}_{q_i^{\mathtt{A}}, q_i^{\mathtt{B}}, q_i^{\mathtt{C}}}$ except $\mu \mathtt{t}_{q_0^{\mathtt{A}}, q_0^{\mathtt{B}}, q_0^{\mathtt{C}}}$ and finish.

The synthesis algorithm can build the global type

$$\mathtt{B} \to \mathtt{A} : \{a : \mathtt{C} \to \mathtt{A} : \{c : \mathsf{end}, d : \mathsf{end}\}, b : \mathtt{C} \to \mathtt{A} : \{c : \mathsf{end}, d : \mathsf{end}\}\}$$

from $\mathtt{A}'$, B and C in Remark 5.2. Note that

$$\mathtt{B} \to \mathtt{A} : \{a : \mathtt{C} \to \mathtt{A} : \{c : \mathsf{end}\}, b : \mathtt{C} \to \mathtt{A} : \{d : \mathsf{end}\}\}$$

generated from A, B and C in Remark 5.2 is neither projectable by Definition 3.1 nor multiparty compatible, hence not well-formed.

With Theorems 5.2, and 5.1, and Corollary 6.1, we can now conclude:

Theorem 6.2 (soundness and completeness). *Suppose S is basic and multiparty compatible. Then there exists G such that $S \approx G$. Conversely, if G is well-formed, then there exists a basic and multiparty compatible system S which satisfies the three safety properties in Definition 2.4 such that $S \approx G$.*

7 Related work

Our previous extended abstract [16] presented the first translation from global and local types into CFSMs. It only analysed the properties of the automata resulting from such a translation. The complete characterisation of global types independently from the projected local types was left open until our work in the extended abstract of this article [17], as was synthesis. This article presents a new proof of the soundness via the sound and complete LTS correspondence between global and local types, and fully develops the synthesis and completeness results.

There are a large number of articles that can be found in the literature about the synthesis of CFSMs. See [28] for a summary of recent results. The main distinction with CFSM synthesis is, apart from the formal setting (i.e. types), about the kind of the target specifications to be generated (global types in our case). Not only our synthesis is concerned about trace properties (languages) like the standard synthesis of CFSMs (the problem of the closed synthesis of CFSMs is usually defined as the construction from a regular language L of a machine satisfying certain conditions related to buffer boundedness, deadlock-freedom and words swapping), but we also generate concrete syntax or choreography descriptions as *types* of programs or software. Hence they are directly applicable to programming languages and can be straightforwardly integrated into the existing frameworks that are based on session types.

Within the context of multiparty session types, [27] first studied the reconstruction of a global type from its projected local types up to asynchronous subtyping and [24] recently offers a typing system to synthesise global types from local types. Our synthesis based on CFSMs is more general since CFSMs do not depend on the syntax. For example, [27, 24] cannot treat the synthesis for $\mathtt{A}'$, $\mathtt{B}$ and $\mathtt{C}$ in Remark 5.2. These works also do not study the completeness (i.e. they build a global type from a set of projected local types (up to subtyping), and do not investigate necessary and sufficient conditions to build a well-formed global type). A difficulty of the completeness result is that it is generally unknown if the global type constructed by the synthesis can simulate executions with arbitrary buffer bounds since the synthesis only directly looks at 1-bounded executions. In this article, we proved Lemma 6.1 and bridged this gap towards the complete characterisation.

As explained in Introduction, the original *binary* (two party) session types [22] correspond to the set of compatible half-duplex deterministic two-machine systems without mixed states [20, 32] (compatible means that each send is matched by a receive, and vice-versa). As stated in this article, the sequential system [12] satisfies strong sequentiality properties and are multiparty. They are shown to be *restricted half-duplex* in [10, § 4.1.2] (i.e. at most one queue is non-empty). The original multiparty session

[3, 23] types studied in this article are a subset of the *natural multiparty extension of half-duplex system* [10, § 4.1.2] where each pair of machines is linked by two buffered channels, one in each direction, such that at most one is non-empty. Gouda et al.'s pioneering work [20] and a formalisation of Sing# contracts [18] (deterministic and no-mixed CFSMs) in Villard's thesis [32] uses compatible half-duplex systems as specifications (contracts) for a proof system for copyless message passing programs. His following work with Lorenz [26] extends to unreliable systems, and proves that safety properties and boundedness are still decidable. These works [20, 32, 26] only treat the two-machine case.

Recent work by [8, 1, 2] as well as our work in [16, 25] focus on proving the semantic correspondence between more general forms of global and local descriptions. Global types in [8] are described by the fork ($\wedge$), choice ($\vee$) and repetition $(G)^*$ (which represents a finite loop of zero or more interactions of G), and choreographies in [1, 2] form a finite state machine with queues. The former investigates the conditions for well-formed choices and sequencing, and the latter studies the realisability (i.e. projectability) conditions for global descriptions. Their systems do not treat the fine-grained causality between sends and receives modelled by the LTSs in this article (i.e. the OO-causality or II-causality at different channels [23], since they either only observe send or receive actions). It explains why we used different LTSs for types that had not been investigated in [8, 1, 2]. Neither synthesis algorithm, a complete characterisation nor an inference of global descriptions from local specifications studied in [8, 1, 2]. The related work in [16, 25] offer a detailed comparison with these works from the perspective of the general forms of the global descriptions.

8 Conclusion and future work

This article investigated the sound and complete characterisation of two classes of multiparty session types into CFSMs and developed two decidable synthesis algorithms from basic CFSMs. The two synthesis algorithms terminate and can always output a global type which enforces the three safety properties of deadlock-freedom, reception error-freedom and orphan message-freedom (Definition 2.4). We summarise the equivalences in Table 1.

Multiparty Session Type		Communicating automata (CA)
Sequential MPST [4]	$\approx$	Sequential (basic, alternating) orphan message-free and deadlock-free CA
MPST [23, 3, 34]	$\approx$	Basic, multiparty compatible CA

Table 1. Summary of the equivalence results

The main tool we used is a new extension to multiparty interactions of the duality condition for binary session types, called *multiparty compatibility*. In sequential systems, multiparty compatibility is *omnipresent* as the execution in the sequential systems is automatically compatible. In the general class, the basic condition (coming from syntactic binary session types) and the multiparty compatibility property are a *necessary and sufficient condition* to obtain safe global types.

As stated in Theorem 2.1 (Corollary 27, Theorem 36 in [10]), in the literature, it had been unknown to identify a decidable class of safe multiparty half-duplex system. Basic multiparty compatible CFSMs define one of the few non-trivial decidable subclass of CFSMs which satisfy safety conditions. Our soundness theorem shows that, starting from a projection of a global type, using the translation from a local type to a basic CFSM, we can automatically generate a subclass of safe CFSMs in the polynomial time complexity.

Our aim is to offer a duality notion which would be applicable to extend other theoretical foundations such as the Curry-Howard correspondence with linear logics [7, 33] to multiparty communications.

The methods proposed here are palatable to a wide range of applications based on choreography protocol models and more widely, finite state machines. For example, the work in [5] extends the framework to timed specification [5]. The basic idea of multiparty compatibility would be applicable for extending the synthesis algorithm to build more expressive graph-based global types (*general global types* [16, 25]) which feature fork and join primitives.

We are currently working on two applications based on the theory developed in this article. The first application is the Testable Architecture [30] in Zero Deviation Life Cycle [36] which enables the communication structure of the implementation to be inferred and to be tested against the choreography. Another application is dynamic monitoring for a large scale cyberinfrastructure in [29]. In the system [29], a central controller checks distributed messages conform monitor specifications which form FSMs projected from a global specification language, Scribble [35, 21]. The implementation framework is explained in [14].

References

1. S. Basu, T. Bultan, and M. Ouederni. Deciding choreography realizability. In *POPL'12*, pages 191–202. ACM, 2012.
2. S. Basu, T. Bultan, and M. Ouederni. Synchronizability for verification of asynchonously communicating systems. In *VMCAI'12*, volume 7148 of *LNCS*, pages 56–71. Springer, 2012.
3. L. Bettini, M. Coppo, L. D'Antoni, M. D. Luca, M. Dezani-Ciancaglini, and N. Yoshida. Global progress in dynamically interleaved multiparty sessions. In *CONCUR*, volume 5201 of *LNCS*, pages 418–433, 2008.
4. K. Bhargavan, R. Corin, P.-M. Deniélou, C. Fournet, and J. Leifer. Cryptographic protocol synthesis and verification for multiparty sessions. In *CSF*, pages 124–140, 2009.
5. L. Bocchi, W. Yang, and N. Yoshida. Timed multiparty session types. In *CONCUR 2014*, volume 8704 of *LNCS*, pages 419–434. Springer, 2014.
6. D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30:323–342, April 1983.
7. L. Caires and F. Pfenning. Session types as intuitionistic linear propositions. In *CONCUR*, volume 6269 of *LNCS*, pages 222–236. Springer, 2010.
8. G. Castagna, M. Dezani-Ciancaglini, and L. Padovani. On global types and multi-party session. *LMCS*, 8(1), 2012.

9. G. Cécé and A. Finkel. Programs with quasi-stable channels are effectively recognizable. In *CAV*, volume 1254 of *LNCS*, pages 304–315. Springer, 1997.
10. G. Cécé and A. Finkel. Verification of programs with half-duplex communication. *Inf. Comput.*, 202(2):166–190, 2005.
11. R. Corin, P. Deniélou, C. Fournet, K. Bhargavan, and J. Leifer. A secure compiler for session abstractions. *Journal of Computer Security*, 16(5):573–636, 2008.
12. R. Corin, P.-M. Deniélou, C. Fournet, K. Bhargavan, and J. Leifer. Secure implementations for typed session abstractions. In *CSF*, pages 170–186, 2007.
13. R. Demangeon and K. Honda. Full abstraction in a subtyped pi-calculus with linear types. In *CONCUR*, volume 6901 of *LNCS*, pages 280–296. Springer, 2011.
14. R. Demangeon, K. Honda, R. Hu, R. Neykova, and N. Yoshida. Practical interruptible conversations: Distributed dynamic verication with multiparty session types and Python. *FMSD*, 2015.
15. P.-M. Deniélou and N. Yoshida. Dynamic multirole session types. In *POPL*, pages 435–446. ACM, 2011. Full version, Prototype at `http://www.doc.ic.ac.uk/~pmalo/dynamic`.
16. P.-M. Deniélou and N. Yoshida. Multiparty session types meet communicating automata. In *ESOP*, volume 7211 of *LNCS*, pages 194–213. Springer, 2012.
17. P.-M. Deniélou and N. Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *ICALP*, volume 7966 of *LNCS*, pages 174–186. Springer, 2013.
18. M. Fähndrich et al. Language Support for Fast and Reliable Message-based Communication in Singularity OS. In *EuroSys2006*, ACM SIGOPS, pages 177–190. ACM Press, 2006.
19. J.-Y. Girard. Linear logic. *TCS*, 50, 1987.
20. M. Gouda, E. Manning, and Y. Yu. On the progress of communication between two finite state machines. *Information and Control.*, 63:200–216, 1984.
21. K. Honda, A. Mukhamedov, G. Brown, T.-C. Chen, and N. Yoshida. Scribbling interactions with a formal foundation. In *ICDCIT 2011*, volume 6536 of *LNCS*, pages 55–75. Springer, 2011.
22. K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP'98*, volume 1381 of *LNCS*, pages 22–138. Springer, 1998.
23. K. Honda, N. Yoshida, and M. Carbone. Multiparty Asynchronous Session Types. In *POPL'08*, pages 273–284. ACM, 2008.
24. J. Lange and E. Tuosto. Synthesising choreographies from local session types. In *CONCUR*, volume 7454 of *LNCS*, pages 225–239. Springer, 2012.
25. J. Lange, E. Tuosto, and N. Yoshida. From communicating machines to graphical choreographies. In *POPL 2015*. ACM, 2015.
26. E. Lozes and J. Villard. Reliable contracts for unreliable half-duplex communications. In *WS-FM*, volume 7176 of *LNCS*, pages 2–16. Springer, 2011.
27. D. Mostrous, N. Yoshida, and K. Honda. Global principal typing in partially commutative asynchronous sessions. In *ESOP'09*, volume 5502 of *LNCS*, pages 316–332. Springer, 2009.
28. A. Muscholl. Analysis of communicating automata. In *LATA*, volume 6031 of *LNCS*, pages 50–57. Springer, 2010.
29. Ocean Observatories Initiative (OOI). `http://www.oceanobservatories.org/`.
30. Savara JBoss Project. `http://www.jboss.org/savara`.
31. K. Takeuchi, K. Honda, and M. Kubo. An interaction-based language and its typing system. In *PARLE'94*, volume 817 of *LNCS*, pages 398–413. Springer, 1994.
32. J. Villard. *Heaps and Hops*. PhD thesis, ENS Cachan, 2011.
33. P. Wadler. Proposition as Sessions. In *ICFP'12*, pages 273–286, 2012.
34. N. Yoshida, P.-M. Deniélou, A. Bejleri, and R. Hu. Parameterised multiparty session types. In *FoSSaCs*, volume 6014 of *LNCS*, pages 128–145, 2010.

35. N. Yoshida, R. Hu, R. Neykova, and N. Ng. The scribble protocol language. In *TGC 2013*, volume 8358 of *LNCS*, pages 22–41. Springer, 2013.
36. Zero Deviation Life Cycle Management Platform. `http://www.cognizant.com/businesscloud/ZDLC-zero-deviation-life-cycle-management-platform`.

A Appendix for Section 3

A.1 Proof of Theorem 3.1

Subtyping for Local Types In order to relate global and local types, we use the subtyping relation $\prec$ on local types [13] as follows:

$$\frac{\forall i \in I, T_i \prec T_i'}{\mathtt{p}!\{a_i.T_i\}_{i\in I} \prec \mathtt{p}!\{a_i.T_i'\}_{i\in I}} \qquad \frac{J \subseteq I \quad \forall j \in J, T_j \prec T_j'}{\mathtt{p}?\{a_i.T_i\}_{i\in I} \prec \mathtt{p}?\{a_j.T_j'\}_{j\in J}}$$

where double lines mean the rules are defined co-inductively. Local type T' is a supertype of local type T, written $T \prec T'$, if T' offers less receive transitions. We note that $\sqcup_{i\in I} T_i \prec T_i$.

This subtyping relation can be extended to configurations in the following way: $(\vec{T};\vec{w}) \prec (\vec{T'};\vec{w'})$ if $\vec{w} = \vec{w'}$ and $\forall \mathtt{p} \in \mathcal{P}, T_{\mathtt{p}} \prec T_{\mathtt{p}}'$. The main properties of this subtyping is (since we do not apply the subtyping for the outputs) that it preserves traces, i.e. if $s \prec s'$, then $s \approx s'$.

Extension of projection In order to prove Theorem 3.1, we extend the definition of projection to global intermediate states.

We represent the projected configuration $[\![G]\!]$ of a global type G as a configuration $\{G \upharpoonright \mathtt{p}\}_{\mathtt{p}\in\mathcal{P}}, [\![G]\!]_{\{\varepsilon\}_{\mathtt{qq'}\in\mathcal{P}}}$ where the content of the buffers $[\![G]\!]_{\{\varepsilon\}_{\mathtt{qq'}\in\mathcal{P}}}$ is given by:

$$\begin{aligned}
[\![\mathtt{p} \rightsquigarrow \mathtt{p'} \colon a_j.G_j]\!]_{\{w_{\mathtt{qq'}}\}_{\mathtt{qq'}\in\mathcal{P}}} &= [\![G_j]\!]_{\{w_{\mathtt{qq'}}\}_{\mathtt{qq'}\in\mathcal{P}}[w_{\mathtt{pp'}} = w_{\mathtt{pp'}}\cdot a_j]} \\
[\![\mathtt{p} \to \mathtt{p'} \colon a_j.G_j]\!]_{\{w_{\mathtt{qq'}}\}_{\mathtt{qq'}\in\mathcal{P}}} &= [\![G_j]\!]_{\{w_{\mathtt{qq'}}\}_{\mathtt{qq'}\in\mathcal{P}}} \\
[\![\mathtt{p} \to \mathtt{p'} \colon \{a_j.G_j\}_{j\in J}]\!]_{\{w_{\mathtt{qq'}}\}_{\mathtt{qq'}\in\mathcal{P}}} &= [\![G_1]\!]_{\{w_{\mathtt{qq'}}\}_{\mathtt{qq'}\in\mathcal{P}}} \quad \text{if } [\![G_i]\!]_{\{w_{\mathtt{qq'}}\}_{\mathtt{qq'}\in\mathcal{P}}} = [\![G_j]\!]_{\{w_{\mathtt{qq'}}\}_{\mathtt{qq'}\in\mathcal{P}}} \\
[\![\mu \mathtt{t}.G]\!]_{\{w_{\mathtt{qq'}}\}_{\mathtt{qq'}\in\mathcal{P}}} &= [\![G[\mu \mathtt{t}.G/\mathtt{t}]]\!]_{\{w_{\mathtt{qq'}}\}_{\mathtt{qq'}\in\mathcal{P}}} \\
[\![G_1 \mid G_2]\!]_{\{w_{\mathtt{qq'}}\}_{\mathtt{qq'}\in\mathcal{P}}} &= [\![G_i]\!]_{\{w_{\mathtt{qq'}}\}_{\mathtt{qq'}\in\mathcal{P}}} \quad \text{if } \mathtt{q}, \mathtt{q'} \in G_i \\
[\![\mathsf{end}]\!]_{\{w_{\mathtt{qq'}}\}_{\mathtt{qq'}\in\mathcal{P}}} &= \{w_{\mathtt{qq'}}\}_{\mathtt{qq'}\in\mathcal{P}}
\end{aligned}$$

The side condition of the branching (the third line) means the contents of the queue are unchanged (since an element is put after $\mathtt{p} \to \mathtt{p'}$ becomes $\mathtt{p} \rightsquigarrow \mathtt{p'}$). The side condition of the parallel comes from the projection algorithm.

The projection algorithm $G \upharpoonright \mathtt{q}$ is extended by:

$$\mathtt{p} \rightsquigarrow \mathtt{p'} \colon j\ \{a_i.G_i\}_{i\in I} \upharpoonright \mathtt{q} = \begin{cases} \mathtt{p}?\{a_i.G_i \upharpoonright \mathtt{q}\}_{i\in I} & \mathtt{q} = \mathtt{p'} \\ G_j \upharpoonright \mathtt{q} & \text{otherwise} \end{cases}$$

This extended projection allows us to match a global type and its projected local types step by step.

Proof for Theorem 3.1 We prove Theorem 3.1 by combining the local type subtyping and extended projection into a step equivalence lemma. Theorem 3.1 is a simple consequence of Lemma A.1.

Lemma A.1 (Step equivalence). *For all global type G and local configuration s, if $s \prec [\![G]\!]$, then we have $G \xrightarrow{\ell} G' \Leftrightarrow s \xrightarrow{\ell} s'$ and $s' \prec [\![G']\!]$.*

Proof. The proof is by induction on the possible global and local transitions.

Soundness By induction on the LTS rules of $G \xrightarrow{\ell} G'$. We prove that if $G \xrightarrow{\ell} G'$, then $[\![G]\!] \xrightarrow{\ell} s$ with $s \prec [\![G']\!]$. We use the fact that if $s \prec s'$, then $s \approx s'$, to consider only matching transition for $[\![G]\!]$.

[GR1] where $G = \mathsf{p} \to \mathsf{p}' \colon \{a_i.G_i\}_{i\in I} \xrightarrow{\mathsf{pp}'!a_j} G' = \mathsf{p} \rightsquigarrow \mathsf{p}' \colon j\, \{a_i.G_i\}_{i\in I}$. The projection of G is $[\![G]\!] = s_T = \{T_\mathsf{q}\}_{\mathsf{q}\in\mathcal{P}}, \{w_{\mathsf{qq}'}\}_{\mathsf{qq}'\in\mathcal{P}}$ with local types: $T_\mathsf{p} = G \upharpoonright \mathsf{p} = \mathsf{p}'!\{a_i.G_i \upharpoonright \mathsf{p}\}_{i\in I}$ and $T_{\mathsf{p}'} = G \upharpoonright \mathsf{p}' = \mathsf{p}?\{a_i.G_i \upharpoonright \mathsf{p}'\}_{i\in I}$ and (for $\mathsf{q} \notin \{\mathsf{p},\mathsf{p}'\}$) $T_\mathsf{q} = \sqcup_{i\in I} G_i \upharpoonright \mathsf{q}$. By Rule [LR1], we have $\mathsf{p}'!\{a_i.G_i \upharpoonright \mathsf{p}\}_{i\in I} \xrightarrow{\mathsf{pp}'!a_j} G_j \upharpoonright \mathsf{p}$. We therefore have $s_T \xrightarrow{\mathsf{pp}'!a_j} \{T'_\mathsf{q}\}_{\mathsf{q}\in\mathcal{P}}, \{w'_{\mathsf{qq}'}\}_{\mathsf{qq}'\in\mathcal{P}}$, with $T'_\mathsf{q} = T_\mathsf{q}$ if $\mathsf{q} \neq \mathsf{p}$, and $T'_\mathsf{p} = G_j \upharpoonright \mathsf{p}$, and with $w'_{\mathsf{qq}'} = w_{\mathsf{qq}'}$ if $\mathsf{qq}' \neq \mathsf{pp}'$, and $w'_{\mathsf{pp}'} = w_{\mathsf{pp}'} \cdot a_j$.
Since $\sqcup_{i\in I} G_i \upharpoonright \mathsf{q} \prec G_j \upharpoonright \mathsf{q}$, we have $\{T'_\mathsf{q}\}_{\mathsf{q}\in\mathcal{P}}, \{w'_{\mathsf{qq}'}\}_{\mathsf{qq}'\in\mathcal{P}} \prec [\![G']\!]$.

[GR2] where $G = \mathsf{p} \rightsquigarrow \mathsf{p}' \colon j\, \{a_i.G_i\}_{i\in I} \xrightarrow{\mathsf{pp}'?a_j} G' = G_j$. The projection of G is $[\![G]\!] = s_T = \{T_\mathsf{q}\}_{\mathsf{q}\in\mathcal{P}}, \{w_{\mathsf{qq}'}\}_{\mathsf{qq}'\in\mathcal{P}}$. The local types are: $T_\mathsf{p} = G \upharpoonright \mathsf{p} = G_j \upharpoonright \mathsf{p}$ and $T_{\mathsf{p}'} = G \upharpoonright \mathsf{p}' = \mathsf{p}?\{a_j.G_j \upharpoonright \mathsf{p}'\}$ and (for $\mathsf{q} \notin \{\mathsf{p},\mathsf{p}'\}$) $T_\mathsf{q} = G_j \upharpoonright \mathsf{q}$. We also know that $w_{\mathsf{pp}'}$ is of the form $w'_{\mathsf{pp}'} \cdot a_j$.
Using [LR2], $\{T_\mathsf{q}\}_{\mathsf{q}\in\mathcal{P}}, \{w_{\mathsf{qq}'}\}_{\mathsf{qq}'\in\mathcal{P}} \xrightarrow{\mathsf{pp}'?a_j} \{G_j \upharpoonright \mathsf{q}\}_{\mathsf{q}\in\mathcal{P}}, \{w'_{\mathsf{qq}'}\}_{\mathsf{qq}'\in\mathcal{P}}$ with $w'_{\mathsf{qq}'} = w_{\mathsf{qq}'}$ if $\mathsf{qq}' \neq \mathsf{pp}'$. The result of the transition is the same as the projection $[\![G']\!]$ of G'.

[GR3] where $G = \mu\mathsf{t}.G' \xrightarrow{\ell} G''$.
By hypothesis, we know that $G'[\mu\mathsf{t}.G'/\mathsf{t}] \xrightarrow{\ell} G''$. By induction, we also know that $[\![G'[\mu\mathsf{t}.G'/\mathsf{t}]]\!] = s_T = \{T_\mathsf{q}\}_{\mathsf{q}\in\mathcal{P}}, \{w_{\mathsf{qq}'}\}_{\mathsf{qq}'\in\mathcal{P}}$ can do a reduction $\xrightarrow{\ell}$ to $[\![G'']\!] = s'_T = \{T'_\mathsf{q}\}_{\mathsf{q}\in\mathcal{P}}, \{w'_{\mathsf{qq}'}\}_{\mathsf{qq}'\in\mathcal{P}}$. By definition, $G'[\mu\mathsf{t}.G'/\mathsf{t}] \upharpoonright \mathsf{q} = G' \upharpoonright \mathsf{q}[\mu\mathsf{t}.G' \upharpoonright \mathsf{q}/\mathsf{t}]$. We use [LR4] to conclude.

[GR4] where $\mathsf{p} \to \mathsf{q} \colon \{a_i.G_i\}_{i\in I} \xrightarrow{\ell} \mathsf{p} \to \mathsf{q} \colon \{a_i.G'_i\}_{i\in I}$ and $\mathsf{p},\mathsf{q} \notin \mathit{subj}(\ell)$. By induction, we know that, $\forall i \in I, [\![G_i]\!] \xrightarrow{\ell} \prec [\![G'_i]\!]$. We need to prove that $[\![\mathsf{p} \to \mathsf{q} \colon \{a_i.G_i\}_{i\in I}]\!] \xrightarrow{\ell} [\![\mathsf{p} \to \mathsf{q} \colon \{a_i.G'_i\}_{i\in I}]\!]$. The projections for all participants are identical, except for $\mathsf{q}' = \mathit{subj}(\ell)$, whose projection is (computed by merging) $\sqcup_{i\in I} G_i \upharpoonright \mathsf{q}'$. Since $\forall i \in I, [\![G_i]\!] \xrightarrow{\ell} \prec [\![G'_i]\!]$, we know that all the $G_i \upharpoonright \mathsf{q}'$ have at least the prefix corresponding to ℓ, and that, using either [LR1] or [LR2], the continuations are the $G'_i \upharpoonright \mathsf{q}'$. We can then conclude that the $\sqcup_{i\in I} G_i \upharpoonright \mathsf{q}' \xrightarrow{\ell} \prec \sqcup_{i\in I} G'_i \upharpoonright \mathsf{q}'$.

[GR5] where $\mathsf{p} \rightsquigarrow \mathsf{q} \colon j\, \{a_i.G_i\}_{i\in I} \xrightarrow{\ell} \mathsf{p} \rightsquigarrow \mathsf{q} \colon j\, \{a_i.G'_i\}_{i\in I}$ and $\mathsf{q} \notin \mathit{subj}(\ell)$ with $G'_i = G_i$ for $i \neq j$. By induction, we know that, $[\![G_j]\!] \xrightarrow{\ell} \prec [\![G'_j]\!]$. We need to prove that $[\![\mathsf{p} \rightsquigarrow \mathsf{q} \colon j\, \{a_i.G_i\}_{i\in I}]\!] \xrightarrow{\ell} \prec [\![\mathsf{p} \rightsquigarrow \mathsf{q} \colon j\, \{a_i.G'_i\}_{i\in I}]\!]$. The projections for all participants are identical, except for $\mathsf{q}' = \mathit{subj}(\ell)$, whose projection is $G_j \upharpoonright \mathsf{q}'$. By induction, $G_j \upharpoonright \mathsf{q}' \xrightarrow{\ell} \prec G'_j \upharpoonright \mathsf{q}'$, which allows us to conclude.

[GR6,7] are straightforward by the induction of G_1 or G_2.

Completeness We prove by the rule induction on $[\![G]\!] = \{T_{\mathtt{p}}\}_{\mathtt{p}\in\mathcal{P}}, \{w_{\mathtt{qq'}}\}_{\mathtt{qq'}\in\mathcal{P}} \xrightarrow{\ell} \{T'_{\mathtt{p}}\}_{\mathtt{p}\in\mathcal{P}}, \{w'_{\mathtt{qq'}}\}_{\mathtt{qq'}\in\mathcal{P}}$ that $G \xrightarrow{\ell} G'$ with $\{T'_{\mathtt{p}}\}_{\mathtt{p}\in\mathcal{P}}, \{w'_{\mathtt{qq'}}\}_{\mathtt{qq'}\in\mathcal{P}} \prec [\![G']\!]$.

[LR1] There is $T_{\mathtt{p}} = G \upharpoonright \mathtt{p} = \mathtt{p}'!\{a_i.G_i \upharpoonright \mathtt{p}\}_{i\in I}$. By definition of projection, G has $\mathtt{p} \to \mathtt{q}\colon \{a_i.G_i\}_{i\in I}$ as subterm, possibly several times (by mergeability). By definition of projection, we note that no action in G can involve $\mathtt{p}$ before any of the occurrences of $\mathtt{p} \to \mathtt{q}\colon \{a_i.G_i\}_{i\in I}$. Therefore we can apply as many times as needed [GR4] and [GR5], and use [GR1] to reduce to $\mathtt{p} \rightsquigarrow \mathtt{q}\colon a_j.G_j$. The projection of the resulting global type corresponds to a subtype to the result of [LR1].

[LR2] There is $T_{\mathtt{p}} = G \upharpoonright \mathtt{p} = \mathtt{q}?\{a_j.G_j \upharpoonright \mathtt{p}\}_{j\in J}$. To activate [LR2], there should be a value a_j in the buffer $w_{\mathtt{pq}}$. By definition of projection, G has therefore $\mathtt{p} \rightsquigarrow \mathtt{q}\colon j\ \{a_i.G_i\}_{i\in I}$ as subterm, possibly several times (by mergeability). By definition of projection, no action in G can involve $\mathtt{p}$ before any of the occurrences of $\mathtt{p} \rightsquigarrow \mathtt{q}\colon j\ \{a_i.G_i\}_{i\in I}$. We can apply as many times as needed [GR4] and [GR5] and use [GR2] to reduce to G_j. The projection of the resulting global type corresponds to the result of [LR2].

[LR3] where $T = \mu\mathtt{t}.T'$. Projection is homomorphic with respect to recursion. Therefore G is of the same form. We can use [GR3] and induction to conclude.

B Proof of Proposition 5.1

We first prove lemmas which are useful for the proof.

B.1 Additional lemmas and definitions

We say that a configuration s with t_1 and t_2 satisfies the *one-step diamond property* if, assuming $s \xrightarrow{t_1} s_1$ and $s \xrightarrow{t_2} s_2$ with $t_1 \neq t_2$, there exists s' such that $s_1 \xrightarrow{t'_1} s'$ and $s_2 \xrightarrow{t'_2} s'$ where $act(t_1) = act(t'_2)$ and $act(t_2) = act(t'_1)$. We use the following lemma to permute the two actions.

Lemma B.1 (diamond property in basic machines). *Suppose $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ and S is basic. Assume $s \in RS(S)$ and $s \xrightarrow{t_1} s_1$ and $s \xrightarrow{t_2} s_2$.*

1. *If t_1 and t_2 are both sending actions such that $act(t_1) = \mathtt{p}_1\mathtt{q}_1!a_1$ and $act(t_2) = \mathtt{p}_2\mathtt{q}_2!a_2$, we have either:*
   (a) $\mathtt{p}_1 = \mathtt{p}_2$ and $\mathtt{q}_1 = \mathtt{q}_2$ and $a_1 = a_2$ with $s_1 = s_2$;
   (b) $\mathtt{p}_1 = \mathtt{p}_2$ and $\mathtt{q}_1 = \mathtt{q}_2$ and $a_1 \neq a_2$;
   (c) ($\mathtt{p}_1 \neq \mathtt{p}_2$ and $\mathtt{q}_1 = \mathtt{q}_2$) or ($\mathtt{p}_1 \neq \mathtt{p}_2$ and $\mathtt{q}_1 \neq \mathtt{q}_2$), and s with t_1 and t_2 satisfies the diamond property.
2. *If t_1 and t_2 are both receiving actions such that $act(t_1) = \mathtt{p}_1\mathtt{q}_1?a_1$ and $act(t_2) = \mathtt{p}_2\mathtt{q}_2?a_2$, we have either:*
   (a) $\mathtt{p}_1 = \mathtt{p}_2$ and $\mathtt{q}_1 = \mathtt{q}_2$ and $a_1 = a_2$ with $s_1 = s_2$;
   (b) ($\mathtt{p}_1 = \mathtt{p}_2$ and $\mathtt{q}_1 \neq \mathtt{q}_2$) or ($\mathtt{p}_1 \neq \mathtt{p}_2$ and $\mathtt{q}_1 \neq \mathtt{q}_2$), and s with t_1 and t_2 satisfies the diamond property.
3. *If t_1 is a receiving action and t_2 is a sending action such that $act(t_1) = \mathtt{p}_1\mathtt{q}_1?a_1$ and $act(t_2) = \mathtt{p}_2\mathtt{q}_2!a_2$, we have either:*

(a) $\mathtt{q}_1 = \mathtt{q}_2$ *and* $\mathtt{p}_1 \neq \mathtt{p}_2$*; or*
(b) $\mathtt{p}_1 = \mathtt{p}_2$ *and* $\mathtt{q}_1 \neq \mathtt{q}_2$*; or*
(c) $\mathtt{p}_1 \neq \mathtt{p}_2$ *and* $\mathtt{q}_1 \neq \mathtt{q}_2$
with $s_1 \neq s_2$*, and* s *with* t_1 *and* t_2 *satisfies the diamond property.*

Proof. For (1), there is no case such that $\mathtt{p}_1 = \mathtt{p}_2$ and $\mathtt{q}_1 \neq \mathtt{q}_2$ since S is directed. Then if $\mathtt{p}_1 = \mathtt{p}_2$ and $\mathtt{q}_1 = \mathtt{q}_2$ and $a_1 = a_2$, then $s_1 = s_2$ by the determinism. The diamond property in (c) is obvious since the sender is distinct. For (2), there is no case such that $\mathtt{p}_1 \neq \mathtt{p}_2$ and $\mathtt{q}_1 = \mathtt{q}_2$ since S is directed. Also there is no case such that $\mathtt{p}_1 = \mathtt{p}_2$ and $\mathtt{q}_1 = \mathtt{q}_2$ and $a_1 \neq a_2$ since the communication between the same peer is done via an FIFO queue. The diamond property in (c) is obvious since the receiver is distinct. For (3), there is no case such that $\mathtt{q}_1 = \mathtt{q}_2$ and $\mathtt{p}_1 = \mathtt{p}_2$ because of no-mixed state. The diamond property is obvious since because of no-mixed state. □

The following definition aims to explicitly describe the causality relation between the actions. These are useful to identify the permutable actions.

Definition B.1 (causality) 1. Suppose $s_0 \xrightarrow{\varphi} s$ and $\varphi = \varphi_0 \cdot t_1 \cdot \varphi_1 \cdot t_2 \cdot \varphi_2$. We write $t_1 \lhd t_2$ (t_2 depends on t_1) if either (1) $t_1 = \mathtt{pq}!a$ and $t_1 = \mathtt{pq}?a$ for some $\mathtt{p}$ and $\mathtt{q}$ or (2) $subj(t_1) = subj(t_2)$.

2. We say $\varphi = t_0 \cdot t_1 \cdot t_2 \cdots t_n$ is *the causal chain* if $s_0 \xrightarrow{\varphi'} s'$ and $\varphi \subseteq \varphi'$ with, for all $0 \leq k \leq n-1$, there exists i such that $i > k$ and $t_k \lhd t_i$. We call φ the maximum causal chain if there is no causal chain φ'' such that $\varphi \subsetneq \varphi'' \subseteq \varphi'$.
3. Suppose $s_0 \xrightarrow{\varphi} s$ and $\varphi = \varphi_0 \cdot t_1 \cdot \varphi_1 \cdot t_2 \cdot \varphi_2$. We write $t_i \sharp t_j$ if there is no causal chain from t_i to t_j with $i < j$.

By Lemma B.1, we have:

Lemma B.2 (maximum causality). *Suppose S is basic and $s \in RS(S)$. Then for all $s \xrightarrow{\varphi} s'$, we have $s \xrightarrow{\varphi_m \cdot \varphi''} s'$ and $s \xrightarrow{\varphi'' \cdot \varphi'_m} s'$ where φ_m, φ'_m are the maximum causal chain.*

Lemma B.3 (input availablity). *Assume $S = (M_\mathtt{p})_{\mathtt{p} \in \mathcal{P}}$ is basic and multiparty compatible. Then for all $s \in RS(S)$, if $s \xrightarrow{\mathtt{pp}'!a} s'$, then $s' \xrightarrow{\varphi} s_2 \xrightarrow{\mathtt{pp}'?a} s_3$.*

Proof. We use Lemma B.1 and Lemma B.2. Suppose $s \in RS(S)$ and $s \xrightarrow{t} s'$ such that $act(t) = \mathtt{pp}'!a$. By contradiction, we assume that there is no φ' such that $s' \xrightarrow{\varphi'} \xrightarrow{t'} s''$ with $act(t') = \mathtt{pp}'?a$. Then by the directedness of S, there should be some input state $(q, \mathtt{qp}'?b, q') \in \delta_{\mathtt{p}'}$ where $q \xrightarrow{\mathtt{qp}'?b'} q'' \xrightarrow{p_1} \xrightarrow{\mathtt{pp}'?a} q'''$ where $b \neq b'$ (hence $q' \neq q''$ by determinism), i.e. the action $\mathtt{qp}'?b$ leads to an incompatible path with one which leads to the action $\mathtt{qp}'?a$.

Suppose $s' \xrightarrow{\varphi_0} \xrightarrow{t_{bi}} s''$ with $t_{bi} = (q, \mathtt{qp}'?b, q')$. Then φ_0 should include the corresponding output action $act(t_{bo}) = \mathtt{qp}'!b$. By Lemma B.2, without loss of generality, we assume $\varphi_0 \cdot t_{bi}$ is the maximum causal chain to t_{bi}. Let us write $\varphi_0 = t_0 \lhd t_1 \lhd \cdots \lhd t_n$. By Lemma B.2, we can set $t_{bo} = t_n$. Note that for all i, $act(t_i) \neq \mathtt{pp}'?a'$ by the assumption: since if $act(t_i) = \mathtt{pp}'?a$, then it contradicts the assumption such that t does not have a corresponding input. Then there are three cases.

1. there is a chain from t to $t_n = t_{bo}$, i.e. there exists $0 \leq i \leq n$ such that $t \lhd t_i \lhd \cdots \lhd t_n$.

2. there is no direct chain from t to t_n but there is a chain to t_{bi}, i.e. there exists $0 \leq i \leq n$ such that $t \lhd t_i \lhd \cdots \lhd t_{bi}$.
3. there is no chain from t to either t_n or t_{bi}.

Case 1: By the assumption, there is no t_j such that $act(t_j) = \mathtt{p}\mathtt{p}'?a'$. Hence $t_i = \mathtt{p}\mathtt{p}''!a'$ for some a' and $\mathtt{p}''$.

Case 1-1: there is no input in t_j in $t \lhd t_i \lhd \cdots \lhd t_{n-1}$. Then $\mathtt{p} = \mathtt{q}$, i.e. $\mathtt{q}\mathtt{p}'!b = \mathtt{p}\mathtt{p}'!b$. Then by the definition of $s \xrightarrow{t} s'$ (i.e. by FIFO semantics at each channel), $\mathtt{p}\mathtt{p}'?b$ cannot perform before $\mathtt{p}\mathtt{p}'?a$. This case contradicts to the assumption $\mathtt{p}\mathtt{p}'?a$ is not available.

Case 1-2: there is an input t_j in $t \lhd t_i \lhd \cdots \lhd t_{n-1}$. By $t \lhd t_i$, $subj(act(t_i)) = \mathtt{p}$. Hence we have either $act(t_i) = \mathtt{p}\mathtt{q}_i!a_i$ with $\mathtt{q} \neq \mathtt{q}_i$ or $act(t_i) = \mathtt{q}_i\mathtt{p}?a_i$.

Case 1-2-1: $act(t_i) = \mathtt{p}\mathtt{q}_i!a_i$. Then there is a path $q \xrightarrow{\mathtt{p}\mathtt{p}'!a} \xrightarrow{\mathtt{p}\mathtt{q}_i!a_i} q'$ in $M_\mathtt{p}$. Hence by the multiparty compatibility, there should be the traces $\mathtt{p}\mathtt{p}'?a \cdot \varphi \cdot \mathtt{p}\mathtt{q}_i?a_i$ with φ alternation from the machine with respect to $\{M_\mathtt{r}\}_{\mathtt{r} \in \mathcal{P} \setminus \mathtt{p}}$. This contradicts to the assumption that $\mathtt{p}\mathtt{p}'?a$ is not available.

Case 1-2-2: $act(t_i) = \mathtt{q}_i\mathtt{p}?a_i$. Similarly with the case **Case 1-2-1**, by the multiparty compatibility and since we have a causal chain from t to t_i, there should be the traces $\mathtt{p}\mathtt{p}'?a \cdot \varphi \cdot \mathtt{p}\mathtt{q}_i?a_i$ with φ alternation from the machine with respect to $\{M_\mathtt{r}\}_{\mathtt{r} \in \mathcal{P} \setminus \mathtt{p}}$. Hence it contradicts to the assumption.

Case 2: Assume the chain such that $t \lhd t_i \lhd \cdots \lhd t_{bi}$ and $t \sharp t_n$. As the same reasoning as **Case 1**, $\mathtt{p} \neq \mathtt{q}$ and t_i is either $\mathtt{p}\mathtt{q}_i!a_i$ or $\mathtt{q}_i\mathtt{p}?a_i$. Then we use the multiparty compatibility.

Case 3: Suppose there exists $s_{04} \in RS(S)$ such that $s_{04} \xrightarrow{t_4} \xrightarrow{\varphi_4} \xrightarrow{\varphi_0} \xrightarrow{t_{bi}}$ and $s_{04} \xrightarrow{t_4'} \xrightarrow{\varphi_4'} \xrightarrow{t}$ where t_4 leads to t_{bi} and t_4' leads to t.

Case 3-1: Suppose t_4 and t_4' are both sending actions. By Lemma B.1, there are three cases.

(a) This case which corresponds to Lemma B.1(a) does not satisfy the assumption since $s_1 = s_2$.

(b) We set $act(t_4) = \mathtt{p}_4\mathtt{q}_4!d$ and $act(t_4') = \mathtt{p}_4\mathtt{q}_4!d'$ with $d \neq d'$. In this case, we cannot execute both t and t_{bi}. Hence there is no possible way to execute t_{bi}. This contradicts to the assumption.

(c) Since this case satisfy the diamond property, we apply the same routine from s' such that $s_{04} \xrightarrow{t_4} \xrightarrow{t_{41}} s'$ and $s_{04} \xrightarrow{t_4'} \xrightarrow{t_{42}} s'$ and $act(t_4) = act(t_{42})$ and $act(t_4') = act(t_{41})$ where the length of the sequences to t and t_{bi} is reduced (hence this case is eventually matched with other cases).

Case 3-2: Suppose t_4 and t_4' are both receiving actions. By Lemma B.1, there are two cases. The case **(a)** is as the same as the case **3-1-(b)** and the case **(b)** is as the same as the case **3-1-(c)**.

Case 3-3: Suppose t_4 is a sending action and t_4' is receiving action. This case is as the same as the case **3-1-(c)**. This concludes the proof. $\square$

We can extend the above lemma.

Lemma B.4 (general input availablity). *Assume $S = (M_\mathtt{p})_{\mathtt{p} \in \mathcal{P}}$ is basic and multiparty compatible. Then for all $s \in RS(S)$, if $s \xrightarrow{\mathtt{p}\mathtt{p}'!a} s_1 \xrightarrow{\varphi} s'$ with $\mathtt{p}\mathtt{p}'?a \notin \varphi$, then $s' \xrightarrow{\varphi'} s_2 \xrightarrow{\mathtt{p}\mathtt{p}'?a} s_3$.*

Proof. We use Lemma B.3. The proof proceeds by the induction of the length of φ. **Case $|\varphi| = 0$.** By Lemma B.3. **Case $|\varphi| = n+1$.** Let $\varphi = \varphi_0 \cdot t$ and $s \xrightarrow{\mathtt{pp'}!a} s_1 \xrightarrow{\varphi_0} s'_0 \xrightarrow{t} s'$. By the inductive hypothesis, there exists φ'_0 such that $s'_0 \xrightarrow{\varphi'_0} s_{20} \xrightarrow{\mathtt{pp'}?a} s_{30}$. By the same reasoning as Lemma B.3, $act(t) = \mathtt{qp'}?b'$ which leads to the incompatible path with one which leads to $\mathtt{pp'}?a$. Then the rest is the same as the proof in Lemma B.3. □

B.2 Proof of Proposition 5.1

Now we prove the stable property.

Proposition 5.1 *Assume $S = (M_\mathtt{p})_{\mathtt{p}\in\mathcal{P}}$ is basic and multiparty compatible. Then S satisfies the stable property, i.e. if, for all $s \in RS(S)$, there exists an execution $\xrightarrow{\varphi'}$ such that $s \xrightarrow{\varphi'} s'$ and s' is stable, and there is a 1-bounded execution $s_0 \xrightarrow{\varphi''} s'$.*

Proof. We proceed by the induction of the total number of messages (sending actions) which should be closed by the corresponding received actions. Once all messages are closed, we can obtain 1-bounded execution.

Suppose s_1, s_2 are the states such that $s_0 \xrightarrow{\varphi_1} s_1 \xrightarrow{t_1} s_2 \xrightarrow{\varphi'_1} s'$ where φ_1 is a 1-bounded execution and $s_1 \xrightarrow{t_1} s_2$ is the first transition which is not followed by the corresponding received action. Since φ_1 is a 1-bounded execution, there is s_3 such that $s_2 \xrightarrow{t_2} s_3$ where t_1 and t_2 are both sending actions. Then by the definition of the compatibility and Lemma B.3, we have

$$s_1 \xrightarrow{t_1} s_2 \xrightarrow{\varphi_2} \xrightarrow{\overline{t_1}} s'_3 \tag{B.1}$$

where φ_2 is an alternation execution and $\overline{t_1} = \mathtt{pq}?a$. Assume φ_2 is a minimum execution which leads to $\overline{t_1}$. We need to show

$$s_1 \xrightarrow{\varphi_2} \xrightarrow{t_1} \xrightarrow{\overline{t_1}} s'_3 \xrightarrow{t_2} s_4$$

Then we can apply the same routine for t_2 to close it by the corresponding receiving action $\overline{t_2}$. Applying this to the next sending state one by one, we can reach an 1-bounded execution. Let $\varphi_2 = t_4 \cdot \varphi'_2$. Then by the definition of multiparty compatibility, $act(t_4) = \mathtt{p'q'}!c$ and $\mathtt{p'} \neq \mathtt{p}$ and $\mathtt{q'} \neq \mathtt{q}$. Hence by Lemma B.1(1), there exists the execution such that

$$s_1 \xrightarrow{t_4} \xrightarrow{t_1} \xrightarrow{\varphi'_2} \xrightarrow{\overline{t_1}} s'_3 \xrightarrow{t_2} s_4$$

Let $\varphi'_2 = \overline{t_4} \cdot \varphi''_2$ where $\overline{t_1} = \mathtt{p'q'}?c$. Then this time, by Lemma B.1(2), we have:

$$s_1 \xrightarrow{t_4} \xrightarrow{\overline{t_4}} \xrightarrow{t_1} \xrightarrow{\varphi''_2} \xrightarrow{\overline{t_1}} s'_3 \xrightarrow{t_2} s_4$$

where $\varphi_1 \cdot t_4 \cdot \overline{t_4}$ is a 1-bounded execution. Applying this permutation repeatedly, we have

$$s_1 \xrightarrow{\varphi_3} \xrightarrow{t_1} \xrightarrow{\overline{t_1}} s'_3 \xrightarrow{t_2} s_4$$

where φ_3 is an 1-bounded execution. We apply the same routine for t_2 and conclude $s_1 \xrightarrow{\varphi'} s'$ for some stable s'. □

C Proof of Theorem 5.2.

We first note that the multiparty compatibility explores stable states which are only reachable by 1-boundned executions, i.e. $RS_1(S)$. By Theorem 3.1, we only have to check that the transitions generated by the LTS rules of global types satisfy the multiparty compatibility. Let us call $\mathtt{p} \rightsquigarrow \mathtt{p}' \colon j\ \{a_i.G_i\}_{i\in I}$ *runtime global type*. A stable state corresponds to the global type which does not contain any runtime global types. We also note that $G = \mathsf{end}$ immediately satisfies the multiparty compatibility. Hence we only have to consider

1. the output action from G which does not contain runtime global types; or
2. the input action $\mathtt{qp}?a$ from G which is reduced from some stable G_0 (which does not contain runtime global types) and G cannot input or output at $\mathtt{p}$ before $\mathtt{qp}?a$. This means $\mathtt{qp}?a$ is the first input which enables in G from participant $\mathtt{p}$ when G is projected into local types;

and prove (1) and (2) satisfy the conditions in (1) and (2) in Definition 5.1, respectively.

We use the following lemma whose proof is straightforward.

Lemma C.1. *Suppose $G_0 \xrightarrow{\mathtt{rr}'!b_k} G = C[\mathtt{r} \rightsquigarrow \mathtt{r}' \colon k\ \{b_k.G''_k\}_{k\in K}]$ by [GR1] or [GR4] where G_0 does not contain any runtime global types and $\mathtt{r}' \notin C$. Then there is an alternation $\mathtt{r}' \notin \vec{\ell}$ such that $G \xrightarrow{\vec{\ell}} \mathtt{r} \rightsquigarrow \mathtt{r}' \colon k\ \{b_k.G''_k\}_{k\in K}$.*

We prove the main theorem by the rule induction of the LTS of the global types.

Case [GR1] $\mathtt{p} \to \mathtt{p}' \colon \{a_i.G_i\}_{i\in I} \xrightarrow{\mathtt{pp}'!a_j} \mathtt{p} \rightsquigarrow \mathtt{p}' \colon j\ \{a_i.G_i\}_{i\in I} = G'$.

By [GR2], $G' \xrightarrow{\mathtt{pp}'?a_j} G''$. Hence it satisfies Definition 5.1(1).

Case [GR2] $G = \mathtt{p} \rightsquigarrow \mathtt{p}' \colon j\ \{a_i.G_i\}_{i\in I} \xrightarrow{\mathtt{pp}'?a_j} G_j = G'$.

This case corresponds to a state of a machine $\mathtt{p}'$ which has a set of input actions $\{\mathtt{pp}'?a_j\}$, and there was the output $\mathtt{pp}'!a_j$ from a machine $\mathtt{p}$. Hence this case immediately satisfies Definition 5.1(2) by Lemma C.1.

Case [GR3] By the inductive hypothesis.

Case [GR4]

$$\frac{\forall j \in I \quad G_j \xrightarrow{\ell} G'_j \quad \mathtt{p},\mathtt{q} \notin \mathit{subj}(\ell)}{\mathtt{p} \to \mathtt{q} \colon \{a_i.G_i\}_{i\in I} \xrightarrow{\ell} \mathtt{p} \to \mathtt{q} \colon \{a_i.G'_i\}_{i\in I}} = G'$$

Sub-case: output Suppose $\ell = \mathtt{rr}'!b_k$ and $\mathtt{r} \neq \mathtt{p}$ and $\mathtt{q} \neq \mathtt{r}$. Then G'_i contains one occurrence of $\mathtt{r} \rightsquigarrow \mathtt{r}' \colon k\ \{b_k.G''_i\}_{k\in I}$. Write $G' = C[\mathtt{r} \rightsquigarrow \mathtt{r}' \colon k\ \{b_k.G''_k\}_{k\in K}]$. We need to prove $G' \xrightarrow{\vec{\ell}\mathtt{rr}'?b_k} C'[G''_k]$ for some alternation $\vec{\ell}$. Since C does not contain any runtime global type, there is an alternation such that $C[\mathtt{r} \rightsquigarrow \mathtt{r}' \colon k\ \{b_k.G''_k\}_{k\in K}] \xrightarrow{\vec{\ell}} \mathtt{r} \rightsquigarrow \mathtt{r}' \colon k\ \{b_k.G''_k\}_{k\in K}$ by Lemma C.1. Hence we can apply [GR2] to obtain $G' \xrightarrow{\vec{\ell}\cdot\mathtt{rr}'?b_k} G''_k$.

Sub-case: input Similar with the case [GR2] by Lemma C.1.

Case [GR5]

$$\frac{G_j \xrightarrow{\ell} G'_j \quad \mathtt{q} \notin subj(\ell) \quad \forall i \in I \setminus j, G'_i = G_i}{\mathtt{p} \rightsquigarrow \mathtt{q} \colon j\ \{a_i.G_i\}_{i \in I} \xrightarrow{\ell} \mathtt{p} \rightsquigarrow \mathtt{q} \colon j\ \{a_i.G'_i\}_{i \in I}}$$

We first note that ℓ cannot be the output due to the form of $G = \mathtt{p} \rightsquigarrow \mathtt{q} \colon j\ \{a_i.G_i\}_{i \in I}$. Hence we suppose $\ell = \mathtt{rr'}?a$ such that $\mathtt{p} \neq \mathtt{r'}$ and $\mathtt{q} \neq \mathtt{r'}$ (since ℓ is an immediate input from the machine $\mathtt{r'}$). Then by the same reasoning as [GR2], there exists $G \xrightarrow{\varphi} G'$ where $\varphi = \mathtt{pq}?a_j \cdot \vec{\ell} \cdot \mathtt{rr'}!a$ and $\vec{\ell}$ is an alternation with $\mathtt{r'} \notin \vec{\ell}$.

Case [GR6,GR7] By the inductive hypothesis, noting the sets of participants in G_1 and G_2 are disjoint.

D Proof for Lemma 6.1

Proof. We prove by induction that $\forall n, S_1 \approx_n S_2 \implies S_1 \approx_{n+1} S_2$. Then the lemma follows.

We assume $S_1 \approx_n S_2$ and then prove, by induction on the length of any execution φ that uses less than n buffer space in S_1, that φ is accepted by S_2. If the length $|\varphi| < n+1$, then the buffer usage of φ for S_1 cannot exceed n, therefore S_2 can realise φ since $S_1 \approx_n S_2$.

Assume that a trace φ in S_1 has length $|\varphi| = k+1$, that φ is $(n+1)$-bound, and that any trace strictly shorter than φ or using less buffer space is accepted by S_2.

We denote the last action of φ as ℓ. We name ℓ_0 the last unmatched send transition $\mathtt{pq}!a$ of φ that is not ℓ. We can therefore write φ as $\varphi_0\ell_0\varphi_1\ell$, with φ_1 minimal. I.e. there is no permutation such that $\varphi_0\ell\varphi'_0\ell_0$. In S_1, we have

$$S_1 : s_0 \xrightarrow{\varphi_0}\xrightarrow{\ell_0}\xrightarrow{\varphi_1} s_1 \xrightarrow{\ell} s \tag{D.1}$$

By Lemma B.4, we have a trace φ_2 such that:

$$S_1 : s_0 \xrightarrow{\varphi_0}\xrightarrow{\ell_0}\xrightarrow{\varphi_1} s_1 \xrightarrow{\varphi_2}\xrightarrow{\overline{\ell_0}} s'_1 \tag{D.2}$$

Case $\varphi_2 = \varepsilon$. Hence

$$S_1 : s_0 \xrightarrow{\varphi_0}\xrightarrow{\ell_0}\xrightarrow{\varphi_1} s_1 \xrightarrow{\overline{\ell_0}} s'_1 \quad \text{and} \quad s_1 \xrightarrow{\ell} s \tag{D.3}$$

Let $\ell = \mathtt{p_1q_1}!b$. Then by Lemma B.1 (3), $s_1 \xrightarrow{\overline{\ell_0}}\xrightarrow{\ell} s''$ as required.

Case $\varphi_2 = \ell_1 \cdot \varphi'_2$.

1. If $\ell = \mathtt{p_1q_1}!b$ and $\ell_1 = \mathtt{p_2q_2}?c$, then by Lemma B.1 (3), $s_1 \xrightarrow{\ell_1}\xrightarrow{\ell} s''$. Hence we apply the induction on φ'_2.
2. If $\ell = \mathtt{p_1q_1}!b$ and $\ell_1 = \mathtt{p_2q_2}!c$, then by directedness, we have three cases:
   (a) $\mathtt{p_1} \neq \mathtt{p_2}$ and $\mathtt{q_1} \neq \mathtt{q_2}$. By Lemma B.1 (1), we have

$$s_1 \xrightarrow{\ell_2} s \xrightarrow{\ell} s'_2 \xrightarrow{\varphi'_2} s'_1 \tag{D.4}$$

   Hence we conclude by the induction on φ'_2.

(b) $\mathrm{p}_1 = \mathrm{p}_2$ and $\mathrm{q}_1 = \mathrm{q}_2$ and $b \neq c$.

In this case, by Lemma B.4, there exists φ_3 such that $s_1 \xrightarrow{\ell}\xrightarrow{\varphi_3}\xrightarrow{\overline{\ell_0}}$. Hence this case is subsumed into (a) or (c) below.

(c) $\mathrm{p}_1 = \mathrm{p}_2$ and $\mathrm{q}_1 = \mathrm{q}_2$ and $b = c$.

Since ℓ_0 and ℓ is not permutable, there is the causality such that $t_0 \lhd t_1 \lhd \cdots \lhd t_n \lhd \cdots \lhd t_{n+m}$ with $act(t_0) = \ell_0$, $act(t_n) = \ell$ and $act(t_{n+m}) = \overline{\ell_0}$. We note that since l_0 is the first outstanding output, by multiparty compatibility, t_i $(1 \leq i \leq n-1)$ does not include $\mathrm{p}_1\mathrm{q}_1?a$ for any a. Hence this case does not exist.

Applying Case (a), we can build in S_1 a sequence of transitions that allows ℓ using strictly less buffer space as:

$$S_1 : \ s_0 \xrightarrow{\varphi_0}\xrightarrow{\varphi_0'}\xrightarrow{\ell_0}\xrightarrow{\varphi_3}\xrightarrow{\overline{\ell_0}}\xrightarrow{\ell} \tag{D.5}$$

where φ_3 is the result of the combination of φ_1 and φ_2 using commutation.

By the assumption $(S_1 \approx_n S_2)$, S_2 can simulate this sequence as:

$$S_2 : \ s_0 \xrightarrow{\varphi_0}\xrightarrow{\varphi_0'}\xrightarrow{\ell_0}\xrightarrow{\varphi_3}\xrightarrow{\overline{\ell_0}}\xrightarrow{\ell} \tag{D.6}$$

All the commutation steps used in S_1 are also valid in S_2 since they are solely based on causalities of the transition sequences. We therefore can permute (D.6) back to:

$$S_2 : \ s_0 \xrightarrow{\varphi_0}\xrightarrow{\ell_0}\xrightarrow{\varphi_3}\xrightarrow{\ell} \tag{D.7}$$

It concludes this proof.