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

Pierre-Malo Deniélou and Nobuko Yoshida

[1] Royal Holloway, University of London
[2] 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 construct a global specification from a set of distributed uncontrolled behaviours, this paper explores the problem of fully characterising multiparty session types in terms of communicating automata. 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 two LTSs, we identify a class of communicating automata that exactly correspond to the projected local types. 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.

1 Introduction

Over the last decade, *session types* [13, 20] have been studied as data types or functional types for communications and distributed systems. A recent discovery by [4, 22], 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 [2, 14] 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.

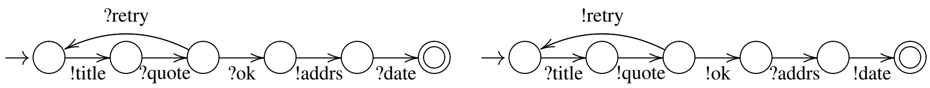
The motivation of this paper comes from our practical experiences that, in many situations, even where we start from the end-point projections of a choreography, we need to reconstruct a global type from distributed specifications. End-point specifications are usually available, either through inference 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 § 5 for applications. [18, 19]).This paper attempts to give the synthesis condition as a sound and complete characterisation of multiparty session types with respect to Communicating Finite State Machines (CFSMs) [3]. CFSMs have been a well-studied formalism for analysing distributed safety properties and are widely present in industry tools. They can been seen as generalised end-point specifications, therefore, an excellent target for a common comparison ground and for synthesis. As explained below, to identify a complete set of CFSMs for synthesis, we first need to answer a question – *what is 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 [12] 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:

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

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

The simplicity and tractability of binary sessions come from the notion of *duality* in interactions [11]. 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 following two machines.



We can observe that these CFSMs satisfy three conditions. First, the communications are *deterministic*: messages that are part of the same choice, ok and retry here, are distinct. Secondly, there is no mixed state (each state has either only sending actions or 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 breaking one of the first two conditions makes the compatibility checking undecidable [12, 21].

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 guarantees the safety as a direct consequence.

We give a simple example to illustrate the problem. The Commit protocol in Figure 1 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 paper presents a decidable notion of *multiparty compatibility* as a generalisation of duality of binary sessions, which in turns characterises a synthesis condition.

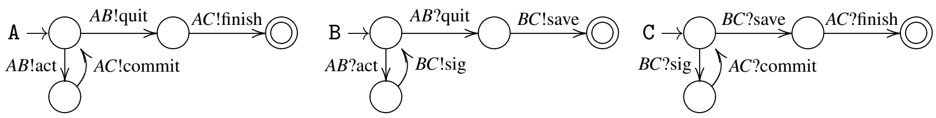


Fig. 1. Commit example: CFSMs

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 1, to check the compatibility of trace $BC?\text{sig}\ AC?\text{commit}$ in C, we execute the internal communications between A and B such that $AB!\text{act} \cdot AB?\text{act}$ and observes the dual trace $BC!\text{sig} \cdot AC!\text{commit}$ from B and A. 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 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 defines *multiparty compatibility*, studies its safety and liveness properties, gives an algorithm for the synthesis of global types from CFSMs, and proves the soundness and completeness results between global types and CFSMs. Section 5 discusses related work and concludes. The full proofs and applications of this work can be found in Appendix.

2 Communicating Finite State Machines

This section starts from some preliminary notations (following [7]). 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: $\mathcal{P} \subseteq \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \ldots, \mathtt{p}, \mathtt{q}, \ldots\}$.

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: $\mathit{subj}(\mathtt{pq}!a) = \mathit{subj}(\mathtt{qp}?a) = \mathtt{p}$.

A state $q \in Q$ whose outgoing transitions are all labelled with sending (resp. receiving) actions is called a *sending* (resp. *receiving*) state. A state $q \in Q$ which does not have any outgoing transition is called *final*. 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 *path* in M is a finite sequence of $q_0,\ldots,q_n$ ($n \geq 1$) such that $(q_i,\ell,q_{i+1}) \in \delta$ ($0 \leq i \leq n-1$), 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 $q' = q''$.[3]

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 $q \in Q_i$ is the *local state* of machine M_i.

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 § 4 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 [7, 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.

[3] "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 [7] in order to represent branching type constructs.

A sequence of transitions is said to be *k-bounded* if no channel of any intermediate configuration s_i contains more than k messages. 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 the number of send actions in φ never exceeds the number of 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. They are undecidable in general CFSMs.

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* [4]if for all $s \in RS(S)$, there exists $s \longrightarrow^* s'$ such that s' is final.

3 Global and local types: the LTSs and translations

This section presents the multiparty session types, our main object of study. For the syntax of types, we follow [2] 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. $\mathtt{p}, \mathtt{q}, \dots \in \mathcal{P}$ denote participants (see § 2 for conventions). The syntax of types is given as:

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

$a_j \in \mathbb{A}$ 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 $\mathtt{p} \to \mathtt{p}' \colon \{a_j.G_j\}_{j \in J}$ states that participant $\mathtt{p}$ can send a message with one of the a_i labels to participant $\mathtt{p}'$ and that interactions described in G_j follow. We require $\mathtt{p} \neq \mathtt{p}'$ to prevent self-sent messages and $a_i \neq a_k$ for all $i \neq k \in J$. Recursive type $\mu \mathtt{t}.G$ is for recursive protocols, assuming that type variables $(\mathtt{t}, \mathtt{t}', \dots)$ are guarded in the standard way, i.e. they only occur under branchings. Type end represents session termination (often omitted). $\mathtt{p} \in G$ means that $\mathtt{p}$ appears in G.

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

Projection The relation between global and local types is formalised by projection. Instead of the restricted original projection [2], we use the extension with the merging

[4] The terminology follows [5].

operator $\bowtie$ from [8]: it allows each branch of the global type to actually contain different interaction patterns.

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

$$\mathtt{p}\to\mathtt{p}'\colon\{a_j.G_j\}_{j\in J}\upharpoonright\mathtt{q}=\begin{cases}\mathtt{p}!\{a_j.G_j\upharpoonright\mathtt{q}\}_{j\in J} & \mathtt{q}=\mathtt{p}\\ \mathtt{p}?\{a_j.G_j\upharpoonright\mathtt{q}\}_{j\in J} & \mathtt{q}=\mathtt{p}'\\ \sqcup_{j\in J}G_j\upharpoonright\mathtt{q} & \text{otherwise}\end{cases}\qquad (\mu\mathtt{t}.G)\upharpoonright\mathtt{p}=\begin{cases}\mu\mathtt{t}.G\upharpoonright\mathtt{p} & G\upharpoonright\mathtt{p}\neq\mathtt{t}\\ \mathsf{end} & \text{otherwise}\end{cases}$$

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

The mergeability relation $\bowtie$ is the smallest congruence relation over local types such that:

$$\frac{\forall i\in(K\cap J).T_i\bowtie T_i'\quad \forall k\in(K\setminus J),\forall j\in(J\setminus K).a_k\neq a_j}{\mathtt{p}?\{a_k.T_k\}_{k\in K}\bowtie\mathtt{p}?\{a_j.T_j'\}_{j\in J}}$$

When $T_1\bowtie T_2$ holds, we define the operation $\sqcup$ as a partial commutative operator over two types such that $T\sqcup T=T$ for all types and that:

$\mathtt{p}?\{a_k.T_k\}_{k\in K}\sqcup\mathtt{p}?\{a_j.T_j'\}_{j\in J}\ =\ \mathtt{p}?(\{a_k.(T_k\sqcup T_k')\}_{k\in K\cap J}\cup\{a_k.T_k\}_{k\in K\setminus J}\cup\{a_j.T_j'\}_{j\in J\setminus K})$

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

Example 3.1 (Commit). The global type for the commit protocol in Figure 1is:

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

Then C's local type is: $\mu\mathtt{t}.\mathtt{B}?\{sig.\mathtt{A}?\{commit.\mathtt{t}\},\ save.\mathtt{A}?\{finish.\mathsf{end}\}\}$.

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

The first step for giving a LTS semantics to global types (and then to local types) is to designate the observables $(\ell,\ell',...)$. 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′: $\ell\ ::=\ \mathrm{pp}'!a\mid\mathrm{pp}'?a$

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 $\mathtt{p}\rightsquigarrow\mathtt{p}'\colon j\ \{a_i.G_i\}_{i\in I}$ to represent the fact that 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 ($subj(\ell)$ is defined in § 2):

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

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

$$[\text{GR4}]\frac{\forall j\in I\quad G_j\xrightarrow{\ell}G_j'\quad \mathtt{p},\mathtt{q}\notin subj(\ell)}{\mathtt{p}\to\mathtt{q}\colon\{a_i.G_i\}_{i\in I}\xrightarrow{\ell}\mathtt{p}\to\mathtt{q}\colon\{a_i.G_i'\}_{i\in I}}\quad[\text{GR5}]\frac{G_j\xrightarrow{\ell}G_j'\quad \mathtt{q}\notin subj(\ell)\quad \forall i\in I\setminus j,G_i'=G_i}{\mathtt{p}\rightsquigarrow\mathtt{q}\colon j\ \{a_i.G_i\}_{i\in I}\xrightarrow{\ell}\mathtt{p}\rightsquigarrow\mathtt{q}\colon j\ \{a_i.G_i'\}_{i\in I}}$$

[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. For example, 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 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 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 $(\ell, \ell', ...)$ 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 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. $w_{\mathtt{pq}}$ represents the FIFO queue at channel pq. We write $\mathit{Tr}(G)$ to denote the set of the visible traces that can be obtained by reducing G. Similarly for $\mathit{Tr}(T)$ and $\mathit{Tr}(S)$. 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). [5] *Let G be a global type with participants $\mathcal{P}$ and let $\vec{T} = \{G \upharpoonright \mathtt{p}\}_{\mathtt{p}\in\mathcal{P}}$ be the local types projected from G. Then $G \approx (\vec{T}; \vec{\varepsilon})$.*

[5] The local type abstracts the behaviour of multiparty typed processes as proved in the subject reduction theorem in [14]. Hence this theorem implies that processes typed by global type G by the typing system in [2, 14] follow the LTS of G.

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.

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 $\mathtt{p}$ projected from G. The automaton corresponding to T_0 is $\mathcal{A}(T_0) = (Q, C, q_0, \mathbb{A}, \delta)$ where: (1) $Q = \{T' \mid T' \in T_0,\ T' \neq \mathtt{t}, T' \neq \mu\mathtt{t}.T\}$;(2) $q_0 = T_0'$ with $T_0 = \mu\vec{\mathtt{t}}.T_0'$ and $T_0' \in Q$; (3) $C = \{\mathtt{pq} \mid \mathtt{p}, \mathtt{q} \in G\}$; (4) $\mathbb{A}$ is the set of $\{a \in G\}$; and (5) δ is defined as:

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

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

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 $\mathtt{pp}'!a_j$ for the output and $\mathtt{pp}'?a_j$ for the input. If T_j is a recursive type variable $\mathtt{t}$, it points the state of the body of the corresponding recursive type. As an example, see $\mathtt{C}$'s local type in Example 3.1 and its corresponding automaton in Figure 1.

Proposition 3.1 (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.*

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 $(q, \mathtt{pp}'!a_j, q_j) \in \delta$;
(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 states that the translations preserve the semantics.

Proposition 3.2 (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 $T \approx \mathcal{A}(T)$.*

4 Completeness and synthesis

This section studies the synthesis and sound and complete characterisation of the multiparty session types as communicating automata. We first note that basic CFSMs correspond to the natural generalisation of half-duplex systems [7, § 4.1.1], in which each pair of machines linked by two channels, one in each direction, communicates in a half-duplex way. In this class, the safety properties of Definition 2.4 are however undecidable [7, Theorem 36]. We therefore need a stronger (and decidable) property to force basic CFSMs to behave as if they were the result of a projection from global types.

Multiparty compatibility In the two machines case, there exists a sound and complete condition called *compatible* [12]. Let us define the isomorphism $\Phi : (C \times \{!,?\} \times \mathbb{A})^* \longrightarrow (C \times \{!,?\} \times \mathbb{A})^*$ such that $\Phi(j?a) = j!a$, $\Phi(j!a) = j?a$, $\Phi(\varepsilon) = \varepsilon$, $\Phi(t_1 \cdots t_n) = \Phi(t_1) \cdots \Phi(t_n)$. Φ exchanges a sending action with the corresponding receiving one and vice versa. The compatibility of two machines can be immediately defined as $Tr(M_1) = \Phi(Tr(M_2))$ (i.e. the traces of M_1 are exactly the set of dual traces of M_2). The idea of the extension to the multiparty case comes from the observation that from the viewpoint of the participant $\mathtt{p}$, 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 $\Phi(Tr(M_{\mathtt{p}}))$, up to internal synchronisations (i.e. 1-bounded executions). Below we define a way to group CFSMs.

Definition 4.1 (Definition 37, [7]). Let $M_i = (Q_i, C_i, q_{0i}, \mathbb{A}_i, \delta_i)$. The *associated CFSM* of $S = (M_1, .., M_n)$ is $M = (Q, C, q_0, \Sigma, \delta)$ such that: $Q = Q_1 \times Q_2 \times \cdots \times Q_n$, $q_0 = (q_{01}, \ldots, q_{0n})$ and δ is the least relation verifying: $((q_1, ..., q_i, ..., q_n), \ell, (q_1, ..., q_i', ..., q_n)) \in \delta$ if $(q_i, \ell, q_i') \in \delta_i$ $(1 \le i \le n)$.

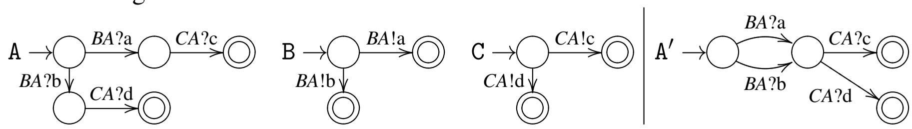
Below we define a notion of compatibility extended to more than two CFSMs. We say that φ is an *alternation* if φ is an alternation of sending and corresponding receive actions (i.e. the action $\mathtt{pq}!a$ is immediately followed by $\mathtt{pq}?a$).

Definition 4.2 (multiparty compatible system). A system $S = (M_1, .., M_n)$ $(n \ge 2)$ is *multiparty compatible* if for any 1-bounded reachable stable state $s \in RS_1(S)$, for any sequence of actions $\ell_1 \cdots \ell_k$ from s in M_i, there is a sequence of transitions $\varphi_1 \cdot t_1 \cdot \varphi_2 \cdot t_2 \cdot \varphi_3 \cdots \varphi_k \cdot t_k$ from s in a CFSM corresponding to $S^{-i} = (M_1, .., M_{i-1}, M_{i+1}, .., M_n)$ where φ_j is either empty or an alternation, $\ell_j = \Phi(act(t_j))$ and $i \notin act(\varphi_j)$ for $1 \le j \le k$ (i.e. φ_j does not contain actions to or from channel i).

The above definition states that for each M_i, the rest of machines S^{-i} can produce the compatible (dual) actions by executing alternations in S^{-i}. From M_i, these intermediate alternations can be seen as non-observable internal actions.

Example 4.1 (multiparty compatibility). As an example, we can test the multiparty compatibility property on the commit example of Figure 1. We only detail here how to check the compatibility from the point of view of $\mathtt{C}$. To check the compatibility for the actions $act(t_1 \cdot t_2) = BC?\text{sig} \cdot AC!\text{commit}$, the only possible 1-bound (i.e. alternating) execution is $AB!\text{act} \cdot AB?\text{act}$, and $\Phi(act(t_1)) = BC!\text{sig}$ sent from $\mathtt{B}$ and $\Phi(act(t_2)) = AC!\text{commit}$ sent from $\mathtt{A}$. To check the compatibility for the actions $act(t_3 \cdot t_4) = BC?\text{save} \cdot AC?\text{finish}$, the 1-bound execution is $AB!\text{quit} \cdot AB?\text{quit}$, and $\Phi(act(t_3)) = BC!\text{save}$ from $\mathtt{B}$ and $\Phi(act(t_4)) = AC!\text{finish}$ from $\mathtt{A}$.

Remark 4.1. In Definition 4.2, we require to 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 $\mathtt{A}$, $\mathtt{B}$ and $\mathtt{C}$, 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 $\mathtt{A}'$, $\mathtt{B}$ and $\mathtt{C}$ satisfy the multiparty compatibility.

Theorem 4.1. *Assume $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ is basic and multiparty compatible. Then S satisfies the three safety properties in Definition 2.4. Further, if there exists at least one $M_{\mathtt{q}}$ which includes a final state, then S satisfies the liveness property.*

Proof. We first prove that any basic S which satisfies multiparty compatible is *stable* (S is stable, 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 $s_0 \xrightarrow{\varphi''} s'$, i.e. any trace can be translated into a 1-bounded execution after some appropriate executions). The proof is non-trivial using a detailed analysis of causal relations to translate into a 1-bounded executions. Then the orphan message- and the reception error-freedom are its corollary. The deadlock-freedom is proved by the stable property and multiparty compatibility. Liveness is a consequence of the orphan message- and deadlock-freedom. See Appendix B. □

Proposition 4.1. *If all the CFSMs $M_{\mathtt{p}}$ ($\mathtt{p} \in \mathcal{P}$) are basic, there is an algorithm to check whether $(M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ is multiparty compatible.*

Proof. The algorithm to check $M_{\mathtt{p}}$'s compatibility with $S^{-\mathtt{p}}$ is defined using the set $RS_1(S)$ of reachable states using 1-bounded executions. Note that the set $RS_1(S)$ is decidable in the polynomial time for half-duplex systems [6, 7]. We start from $q = q_0$ and the initial configuration $s = s_0$. Suppose that, from q, we have the transitions $t_i = (q, \mathtt{qp}!a_i, q'_i) \in \delta_{\mathtt{p}}$. We then construct $RS_1(S)$ (without executing $\mathtt{p}$) until it includes s' such that $\{s' \xrightarrow{t_i} \xrightarrow{t'_j} s_j\}_{j\in J}$ where $act(t'_i) = \mathtt{qp}?a_i$ and $I \subseteq J$. If there exists no such s', it returns false and terminates. The case where, from q, we have receiving transitions $t = (q, \mathtt{qp}?a_i, q'_i)$ is dual. 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 $q \in Q_{\mathtt{p}}$. Then repeat for the other machines $\mathtt{p}'$ such that $\mathtt{p}' \in \mathcal{P} \setminus \mathtt{p}$. Then we repeat this procedure for all stable $s \in RS_1(S)$. □

The proof of Theorem 4.1 is non-trivial using a detailed analysis of causal relations.
Synthesis Below we state the lemma which will be crucial for the proof of the 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. See Appendix B.2 for the proof.

Lemma 4.1 (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 $S_1 \approx S_2$.*

Theorem 4.2 (synthesis). *Suppose S is a basic system and multiparty compatible. Then there is an algorithm which successfully builds well-formed G such that $S \approx G$ if such G exists, and otherwise terminates.*

Proof. We assume $S = (M_{\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$. The algorithm starts from the initial states of all machines $(q^{\mathtt{p}_1}{}_0, ..., q^{\mathtt{p}_n}{}_0)$. 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, the participants in two pairs are disjoint, and 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:

- 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 $\mathtt{t}_{q_1, ..., q_n}$. 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;
- otherwise, 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 $i \in I$.
  - 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 $I \subseteq J$.
  - 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.
  - if all sending states in $q_1, ..., q_n$ become final, then we set $G(q_1, ..., q_n) = \mathsf{end}$.
- we erase unnecessary $\mu \mathtt{t}$ if $\mathtt{t} \notin G$.

Since the algorithm only explores 1-bounded executions, the reconstructed G satisfies $G \approx_1 S$. By Theorem 3.1, we know that $G \approx (\{G \upharpoonright \mathtt{p}\}_{\mathtt{p} \in \mathcal{P}}; \vec{\varepsilon})$. Hence, by Proposition 3.2, we have $G \approx S'$ where S' is the communicating system translated from the projected local types $\{G \upharpoonright \mathtt{p}\}_{\mathtt{p} \in \mathcal{P}}$ of G. By Lemma 4.1, $S \approx S'$ and therefore $S \approx G$. $\square$

The algorithm can generate the global type in Example 3.1 from CFSMs in Figure 1 and 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 $\mathtt{A}'$, $\mathtt{B}$ and $\mathtt{C}$ in Remark 4.1. 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 by $\mathtt{A}$, $\mathtt{B}$ and $\mathtt{C}$ in Remark 4.1 is not projectableby Definition 3.1, hence not well-formed.

By Theorems 3.1 and 4.1, and Proposition 3.2, we can now conclude:

Theorem 4.3 (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 basic and multiparty compatible S such that $S \approx G$.*

5 Conclusion and related work

This paper investigated the sound and complete characterisation of multiparty session types into CFSMs and developed a decidable synthesis algorithm from basic CFSMs. The main tool we used is a new extension to multiparty interactions of the duality condition for binary session types, called *multiparty compatibility*. The basic condition (coming from the binary session types) and the multiparty compatibility property are a *necessary and sufficient condition* to obtain safe global types. 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 [4, 22] to multiparty communications. Basic multiparty compatible CFSMs also define one of the few non-trivial decidable subclass of CFSMs which satisfy deadlock-freedom. The methods proposed here are palatable to a wide range of applications based on choreography protocol models and more widely, finite state machines. Multiparty compatibility is applicable for extending the synthesis algorithm to build more expressive graph-based global types (*general global types* [9]) which feature fork and join primitives [10]. We are currently working on two applications based on the theory developed in this paper: the Testable Architecture [19] which enables the communication structure of the implementation to be inferred and to be tested against the choreography; and dynamic monitoring for a large scale cyberinfrastructure in [18] where a central controller can check that distributed update paths for monitor specifications (which form FSMs projected from a global specification) are safe by synthesis.

Our previous work [9] 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, as was synthesis. This present paper closes this open problem. There are a large number of paper that can be found in the literature about the synthesis of CFSMs. See [17] 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, [16] first studied the reconstruction of a global type from its projected local types up to asynchronous subtyping and [15] 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, [15, 16] cannot treat the synthesis for $\mathtt{A}'$, $\mathtt{B}$ and $\mathtt{C}$ in Remark 4.1. 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 paper, we proved Lemma 4.1 and bridged this gap towards the complete characterisation. Recent work by [1, 5] focus on proving the semantic correspondence between global and local descriptions (see [9] for more detailed comparison), but no synthesis algorithm is studied.

Acknowledgement. The work has been partially sponsored by Ocean Observatories Initiative and EPSRC EP/K011715/1,EP/K034413/1 and EP/G015635/1.

References

1. S. Basu, T. Bultan, and M. Ouederni. Deciding choreography realizability. In *POPL'12*, pages 191–202. ACM, 2012.
2. L. Bettini et al. Global progress in dynamically interleaved multiparty sessions. In *CONCUR*, volume 5201 of *LNCS*, pages 418–433, 2008.
3. D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30:323–342, April 1983.
4. L. Caires and F. Pfenning. Session types as intuitionistic linear propositions. In *CONCUR*, volume 6269 of *LNCS*, pages 222–236. Springer, 2010.
5. G. Castagna, M. Dezani-Ciancaglini, and L. Padovani. On global types and multi-party session. *LMCS*, 8(1), 2012.
6. G. Cécé and A. Finkel. Programs with quasi-stable channels are effectively recognizable. In *CAV*, volume 1254 of *LNCS*, pages 304–315. Springer, 1997.
7. G. Cécé and A. Finkel. Verification of programs with half-duplex communication. *Inf. Comput.*, 202(2):166–190, 2005.
8. 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`.
9. P.-M. Deniélou and N. Yoshida. Multiparty session types meet communicating automata. In *ESOP*, volume 7211 of *LNCS*, pages 194–213. Springer, 2012.
10. http://arxiv.org/abs/1304.1902.
11. J.-Y. Girard. Linear logic. *TCS*, 50, 1987.
12. M. Gouda, E. Manning, and Y. Yu. On the progress of communication between two finite state machines. *Information and Control.*, 63:200–216, 1984.
13. 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.
14. K. Honda, N. Yoshida, and M. Carbone. Multiparty Asynchronous Session Types. In *POPL'08*, pages 273–284. ACM, 2008.
15. J. Lange and E. Tuosto. Synthesising choreographies from local session types. In *CONCUR*, volume 7454 of *LNCS*, pages 225–239. Springer, 2012.
16. 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.
17. A. Muscholl. Analysis of communicating automata. In *LATA*, volume 6031 of *LNCS*, pages 50–57. Springer, 2010.
18. Ocean Observatories Initiative (OOI). `http://www.oceanobservatories.org/`.
19. Savara JBoss Project. `http://www.jboss.org/savara`.
20. 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.
21. J. Villard. *Heaps and Hops*. PhD thesis, ENS Cachan, 2011.
22. P. Wadler. Proposition as Sessions. In *ICFP'12*, pages 273–286, 2012.

A Appendix for Section 3

A.1 Proof of Theorem 3.1

Local Types Subtyping In order to relate global and local types, we define in Figure 2 a subtyping relation $\prec$ on local types. Local type T' is a super type of local type T, written $T \prec T'$, if it offers more receive transitions. We note that $T_i \prec \sqcup_{i \in I} T_i$.

$$\frac{\forall i{\in}I, T_i{\prec}T_i'}{\mathtt{p}!\{a_i.T_i\}_{i\in I}{\prec}\mathtt{p}!\{a_i.T_i'\}_{i\in I}} \qquad \frac{I{\subseteq}J \quad \forall i{\in}I, T_i{\prec}T_i'}{\mathtt{p}?\{a_i.T_i\}_{i\in I}{\prec}\mathtt{p}?\{a_j.T_j'\}_{j\in J}} \qquad \frac{}{\mathtt{t}{\prec}\mathtt{t}} \qquad \frac{T{\prec}T'}{\mu\mathtt{t}.T{\prec}\mu\mathtt{t}.T'}$$

Fig. 2. Subtyping between local types

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 subtyping is 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:

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

and where the projection algorithm $\upharpoonright \mathtt{q}$ 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 global type and projected local type transitions step by step.

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 $[\![G]\!] \prec s$, then we have $G \xrightarrow{\ell} G' \Leftrightarrow s \xrightarrow{\ell} s'$ and $[\![G']\!] \prec s'$.*

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

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

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

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

[GR3] where $G = \mu\mathtt{t}.G' \xrightarrow{\ell} G''$.
By hypothesis, we know that $G'[\mathtt{t}/\mu\mathtt{t}.G'] \xrightarrow{\ell} G''$. By induction, we know that $[\![G'[\mathtt{t}/\mu\mathtt{t}.G']]\!] = s_T = \{T_\mathtt{q}\}_{\mathtt{q}\in\mathcal{P}}, \{w_{\mathtt{qq}'}\}_{\mathtt{qq}'\in\mathcal{P}}$ can do a reduction $\xrightarrow{\ell}$ to $[\![G'']\!] = s_T = \{T'_\mathtt{q}\}_{\mathtt{q}\in\mathcal{P}}, \{w'_{\mathtt{qq}'}\}_{\mathtt{qq}'\in\mathcal{P}}$.
Projection is homomorphic for recursion, hence $G'[\mu\mathtt{t}.G'/\mathtt{t}] \upharpoonright \mathtt{q} = G' \upharpoonright \mathtt{q}[\mu\mathtt{t}.G' \upharpoonright \mathtt{q}/\mathtt{t}]$.
We use [LR4] to conclude.

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

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

Completeness We prove by 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 $[\![G']\!] \prec \{T'_\mathtt{p}\}_{\mathtt{p}\in\mathcal{P}}, \{w'_{\mathtt{qq}'}\}_{\mathtt{qq}'\in\mathcal{P}}$.

[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.

A.2 Local types and CFSMs

Proposition 3.1 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.

Proposition 3.2 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.

B Appendix for Section 4

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 $\mathtt{q}_1 \neq \mathtt{q}_2$ with $a_1 \neq a_2$, 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) $\mathtt{p}_1 \neq \mathtt{p}_2$ 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.
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 $\mathtt{p}_1 \neq \mathtt{p}_2$ and $\mathtt{q}_1 = \mathtt{q}_2$ 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. 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. 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. $\square$

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 (output-input dependency). *Suppose S is basic. Then there is no causal chain $t_0 \cdot t_1 \cdot t_2 \cdots t_n$ such that $act(t_0) = \mathtt{pq}!a$ and $act(t_n) = \mathtt{pq}'?b$ with $a \neq b$ and $act(t_i) \neq \mathtt{pq}?c$ for any c ($1 \leq i \leq n-1$).*

Proof. We use the following definition. The causal chain $\varphi = t_0 \cdot t_1 \cdots t_n$ is called

1. *O-causal chain* if for all $1 \leq i \leq n$, $t_i = \mathtt{pq}_i!a_i$ with some $\mathtt{q}_i$ and a_i.
2. *I-causal chain* if for all $1 \leq i \leq n$, $t_i = \mathtt{q}_i\mathtt{p}?a_i$ with some $\mathtt{q}_i$ and a_i.

Then any single causal chain $\varphi = \tilde{t}_0 \cdot \tilde{t}_1 \cdots \tilde{t}_n$ can be decomposed into alternating O and I causal chains where $t_i = \cdot t_{i0} \cdots t_{in_i}$ with either (1) $act(t_{in_i}) = \mathtt{pq}!a$ and $act(t_{i+10}) = \mathtt{q}'\mathtt{p}?b$; (2) $act(t_{in_i}) = \mathtt{pq}?a$ and $act(t_{i+10}) = \mathtt{qp}'!b$; or (3) $act(t_{in_i}) = \mathtt{pq}!a$ and $act(t_{i+10}) = \mathtt{pq}?a$. In the case of (1,2), we note $subj(t_{ih}) = subj(t_{i+1k})$ for all $0 \leq h \leq n_i$ and $0 \leq k \leq n_{i+1}$.

Now assume S is basic and there is a sequence $\varphi = t_0 \cdot t_1 \cdots t_n$ such that $act(t_0) = \mathtt{p}_0\mathtt{q}_0!a_0$ and $act(t_n) = \mathtt{p}_n\mathtt{q}_n?a_n$ with $\mathtt{p}_0 = \mathtt{q}_n$, $a_0 \neq a_n$ and $act(t_i) \neq \mathtt{p}_0\mathtt{q}_0?a$ for any a ($1 \leq i \leq n-1$). We prove φ is not a causal chain by the induction of the length of φ.

Case $n = 1$. By definition, $t_0 \sharp t_n$.

Case $n > 1$. If φ is a causal chain, there is a decomposition into O and I causal chains such that $\varphi = \tilde{t}_0 \cdot \tilde{t}_1 \cdots \tilde{t}_m$ where $t_i = t_{i0} \cdots t_{in_i}$. By the condition $t_i \neq \mathtt{p}_0\mathtt{q}_0?a$ for any a ($1 \leq i \leq n-1$), the case (3) above is excluded. Hence we have $subj(t_{ih}) = subj(t_{i+1k})$ for all $0 \leq h \leq n_i$ and $0 \leq k \leq n_{i+1}$. This implies

1. $p_0 = p_{ij}$ with i even (in the O causal chains)
2. $q_{ij} = q_0$ with i odd (in the I causal chains); and
3. $p_{in_i} = q_{i+10}$ with i even.

This implies $p_0 = q_0$ which contradicts the definition of the channels of CFSMs (i.e. $p_0 \neq q_0$ if p_0q_0 is a channel). Hence there is no causal chain from $act(t_0) = p_0q_0!a_0$ to $act(t_n) = p_0q_0?a_n$ if $act(t_i) \neq p_0q_0?a$ and $a_0 \neq a_n$.

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

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

Suppose $s' \xrightarrow{\varphi_0} \xrightarrow{t_{bi}} s''$ with $t_{bi} = (q, qp'?b, q')$. Then φ_0 should include the corresponding output action $act(t_{bo}) = qp'!b$. 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.1, we can set $t_{bo} = t_n$. Note that for all i, $act(t_i) \neq pp'?a'$ by the assumption: since if $act(t_i) \neq pp'?a$, then it contradicts the assumption such that t does not have a corresponding input; and if $act(t_i) = pp'?a'$ with $a \neq a'$ then, by directedness of S, it contradicts to the assumption that t_{bi} is the first input which leads to the incompatible path. 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 $act(t_j) = pp'?a'$. Hence $t_i = pp''!a'$ for some a' and p''.

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

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

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

Case 1-2-2: $act(t_i) = q_ip?a_i$. Similarly with the case **Case 1-2-1**, by the multiparty compatibility, there should be the traces $pq?a \cdot \varphi \cdot pq_i?a_i$ with φ alternation from the machine with respect to $\{M_r\}_{r\in\mathcal{P}\setminus p}$. 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 $\mathtt{pq}_i!a_i$ 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 $act(t_4) = t_{42}$ and $act(t_4') = t_{41}$ 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.5 (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 $s \xrightarrow{\mathtt{pp}'!a} s_1 \xrightarrow{\varphi} s'$ with $\mathtt{pp}'?a \notin \varphi$, then $s' \xrightarrow{\varphi'} s_2 \xrightarrow{\mathtt{pp}'?a} s_3$.*

Proof. We use Lemma B.4. The proof proceeds by the induction of the length of φ.

Case $|\varphi| = 0$. By Lemma B.4. **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.4, $act(t) = \mathtt{qp}'?b'$ 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.4. $\square$

B.1 Proofs of Theorem 4.1

We first prove the following stable property.

Proposition B.1 (stable property). *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-bound 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.4, 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 $\overline{t_1} = \mathsf{pq}?a$. 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, $act(t_4) = \mathsf{p}'\mathsf{q}'!c$ and $\mathsf{p}' \neq \mathsf{p}$ and $\mathsf{q}' \neq \mathsf{q}$. 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 $\overline{t_1} = \mathsf{p}'\mathsf{q}'?c$. 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'. □

From the stable property, the orphan message- and the reception error-freedom are immediate. Also the liveness is a corollary by the orphan message- and deadlock-freedom. Hence we only prove the deadlock-freedom assuming the stable property.

Deadlock-freedom Assume S is basic and satisfy the multiparty session compatibility. By the above lemma, S satisfies the stable property. Hence we only have to check for all $s \in RS_1(S)$, s is not dead-lock. Suppose by the contradiction, s contains the receiving states $t_1, ..., t_n$. Then by the multiparty compatibility, there exists 1-bounded execution φ such that $s \xrightarrow{\varphi} \xrightarrow{\overline{t}_1} s'$. Hence $s' \xrightarrow{t_1} s''$ and s'' is stable. Applying this routine to the rest of receiving states $t_2, ..., t_n$, we conclude the proof. □

B.2 Proof for Lemma 4.1

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 $\mathsf{pq}!a$ 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

$$S_1 : \ s_0 \xrightarrow{\varphi_0}\xrightarrow{\ell_0}\xrightarrow{\varphi_1} s_1 \xrightarrow{\ell} s \tag{B.2}$$

By Lemma B.5, we have a trace φ_2 such that:

$$S_1 : \ s_0 \xrightarrow{\varphi_0}\xrightarrow{\ell_0}\xrightarrow{\varphi_1} s_1 \xrightarrow{\varphi_2}\xrightarrow{\overline{\ell_0}} s_1' \tag{B.3}$$

Case $\varphi_2 = \varepsilon$. Hence

$$S_1 : \ s_0 \xrightarrow{\varphi_0}\xrightarrow{\ell_0}\xrightarrow{\varphi_1} s_1 \xrightarrow{\overline{\ell_0}} s_1' \quad \text{and} \quad s_1 \xrightarrow{\ell} s \tag{B.4}$$

Let $\ell = \mathsf{p}_1\mathsf{q}_1!b$. 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 $\ell = \mathsf{p}_1\mathsf{q}_1!b$ and $\ell_1 = \mathsf{p}_2\mathsf{q}_2?c$, then by Lemma B.1 (3), $s_1 \xrightarrow{\ell_1}\xrightarrow{\ell} s''$. Hence we apply the induction on φ_2'.
2. If $\ell = \mathsf{p}_1\mathsf{q}_1!b$ and $\ell_1 = \mathsf{p}_2\mathsf{q}_2!c$, then by directedness, we have three cases:
   (a) $\mathsf{p}_1 \neq \mathsf{p}_2$ and $\mathsf{q}_1 \neq \mathsf{q}_2$. By Lemma B.1 (1), we have

   $$s_1 \xrightarrow{\ell_2} s \xrightarrow{\ell} s_2' \xrightarrow{\varphi_2'} s_1' \tag{B.5}$$

   Hence we conclude by the induction on φ_2'.
   (b) $\mathsf{p}_1 = \mathsf{p}_2$ and $\mathsf{q}_1 = \mathsf{q}_2$ and $b \neq c$.
   In this case, by Lemma B.5, 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) $\mathsf{p}_1 = \mathsf{p}_2$ and $\mathsf{q}_1 = \mathsf{q}_2$ 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 $\mathsf{p}_1\mathsf{q}_1?a$. Then by Lemma B.3, 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:

$$S_1 : \ s_0 \xrightarrow{\varphi_0}\xrightarrow{\varphi_0'}\xrightarrow{\ell_0}\xrightarrow{\varphi_3}\xrightarrow{\overline{\ell_0}}\xrightarrow{\ell} \tag{B.6}$$

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:

$$S_2 : \ s_0 \xrightarrow{\varphi_0}\xrightarrow{\varphi_0'}\xrightarrow{\ell_0}\xrightarrow{\varphi_3}\xrightarrow{\overline{\ell_0}}\xrightarrow{\ell} \tag{B.7}$$

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 (B.7) back to:

$$S_2: \; s_0 \xrightarrow{\varphi_0} \xrightarrow{\ell_0} \xrightarrow{\varphi_3} \xrightarrow{\ell} \tag{B.8}$$

It concludes this proof.