Multiparty Session Types as Communicating Automata: Characterisation and Synthesis

Pierre-Malo Deniélou Nobuko Yoshida

Imperial College London

{malo,yoshida}@doc.ic.ac.uk

Abstract

Multiparty session types is a type discipline that can ensure the safety and liveness of distributed peers via the global specification of the interaction. 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 three classes of communicating automata that exactly correspond to the projected local types of three different multiparty session type theories. We exhibit decidable algorithms 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 that was valid for binary session types.

1. Introduction

Over the last decade, *session types* [24, 35] have been studied as data types or functional types for communications and distributed systems. A session type describes the protocol of a series of data inputs and outputs with a choice of interactions and recursions. A recent discovery by [9, 37], 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. On the practical side, various extensions of session types have been proposed to improve expressiveness and gain stronger safety guarantees. Multiparty session type discipline [4, 25] is one of the major generalisation of binary session types. It 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 (called *local types*), against which processes can be statically type-checked and verified to behave correctly.

This paper has two main motivations, one theoretical, one more practical. First, from a theoretical point of view, the question of the comparison of multiparty session types with other specification frameworks has not had a satisfying, i.e. formal, answer yet. In this paper, we attempt to answer this first question through the sound and complete characterisation of multiparty session types with respect to communicating automata, also called Communicating Finite State Machines (CFSMs). CFSMs [8] consist of a finite state representation of a fixed number of actors who can interact through unbounded buffered channels. CFSMs have been used as a standard for the analysis of distributed safety properties and are widely present in industry tools. They are therefore an excellent target for a common comparison ground.

The second motivation comes from our practical experiences that, in many situations, even where we start from the end-point projections of a choreography or business model description, we need to reconstruct a global type from distributed (most often locally updated) specifications. End-points 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. Global choreographies however usually demand some effort to be written, although they should arguably be present in the design and architecture documents of any well-planned distributed application project. If one knows the precise conditions under which a global type can be constructed, not only the global safety property which multiparty session types ensure is guaranteed, but also the generated global type or choreography business model notation [7] can be used as a refinement and be integrated within the distributed system development life-cycle (see § 8 for examples of applications [31, 33]).

Overall, the question we aim to answer in this paper is the following:

In the world of communicating automata,
what exactly are multiparty session types?

Characterisation of binary session types as communicating automata The answer to this question in the binary session types case has been recently discovered by Villard [36]: a two-machine subclass of CFSMs, where deadlock-freedom and orphan message-freedom are guaranteed, characterises exactly binary session type behaviours. This subclass was actually proposed by Gouda, Manning and Yu in 1984 [22] in a pure communicating automata context. Let us explain by example how the two formalisms coincide together.

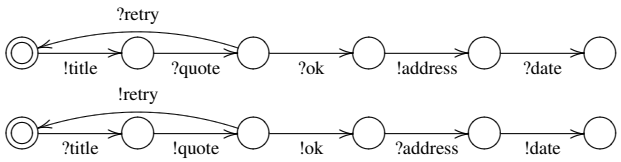
Consider a simple business protocol between a Buyer and a Seller from the Buyer's viewpoint: Buyer sends the title of a book, Seller answers with a quote. If Buyer is satisfied by the quote, then he sends his address and Seller sends back the delivery date; otherwise it retries the same conversation. This can be described by the following session type:

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

The session type above describes a communication pattern using several constructs. The operator !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 μt is recursion.

The simplicity and tractability of binary sessions come from the notion of *duality* in interactions [21]. 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 the above CFSMs satisfy three conditions. First, the communications are *deterministic*: the messages that are part of the same choice, ok and retry here, should be distinct. Secondly, they do not have mixed states (each state has either only sending actions or only receiving actions). Third, the 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 [22]. Villard's result [36] states that two-machine, deterministic, no mixed state, compatible communicating systems characterise binary session types and Singularity contracts [19].

Extension to the multiparty case 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 that 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 indefinitely satisfies safety (which is undecidable in the general case), inferring a global type allows to prove the safety as a consequence.

Example: Dispatch protocol In Figure 1, we give an example to illustrate the problem. The Dispatch protocol involves three machines, a dispatcher D and worker machines A and B. D is in charge of repeatedly sending tasks (messages m_1 and m_2) to B and C in alternation. However, A always waits for the completion of the task by A (acknowledgement message a_1) before sending B its new task, and same with B (acknowledgement message a_2).

The left part of Figure 1 features the three communicating automata for D, A and B. D's automaton is the most complex, as it needs to handle the possible interleavings of A and B's acknowledgements. The workers' automata however are straightforward two-states loops where they respectively receive m_1 and send a_1, and receive m_2 and send a_2.

Multiparty compatibility In this paper, we present a decidable notion of *multiparty compatibility* which extends the duality of binary session types and which can identify whether global choreography exists for a given set of communicating automata. The idea of this condition is to check the duality between each automaton and the rest, up to the internal communications that the other machines will independently perform. If this extended duality is valid for all the machines, then we can guarantee the existence of a global choreography and exhibit it algorithmically as a global type.

The right of Figure 1 shows a graphical representation (using a Choreography BPMN 2.0 notation [7]) of the global type for the Dispatch protocol. It describes concisely the orchestration of the messages that are exchanged. Its interpretation is the following: from the initial node (in green), each participant follows the transitions and the operators: + is choice (not present here) and merge (when two mutually exclusive flows have the same continuation), | is fork (two concurrent behaviours follow) and join (two concurrent lines synchronise). In particular here, it is explicit that b and a always alternate, that c and d always alternate and, through the fork-join combination, a and c alternate.

The collection of automata of Figure 1 can be proven to be multiparty compatible, and our inference algorithm is able to produce the global type on the right. The synthesis works, in the general case, by using Petri net representations of finite state machines, and relies on the memory-less behaviours of a multiparty compatible system.

In this paper, to provide a progressive path, and to tackle relevant subclasses, we study three representative classes of multiparty

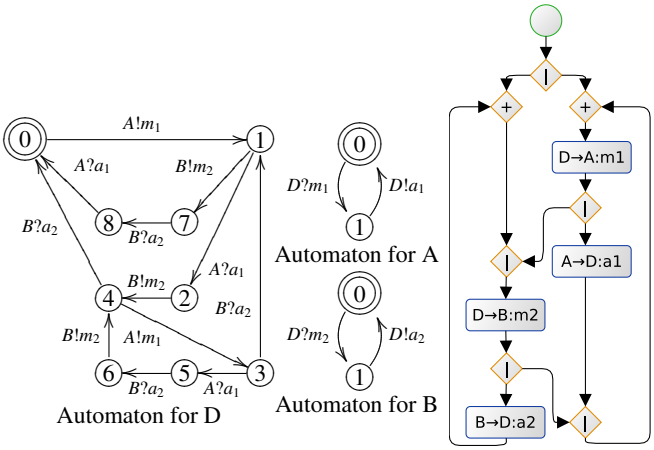


Figure 1. Dispatch example: CFSMs and global type

session types from the literature and we give their complete communicating automata characterisation and synthesis.

Contributions and Outline

§ 3 We define new labelled transition relations for global and local types that represent the usual observable behaviour of typed processes. We prove that a global type behaves exactly as its projected local types. We also prove a similar result between a single local type and its CFSMs interpretation. These correspondences are used for the synthesis algorithms and characterisations in § 4 and § 5.

§ 4 As a first step, we study the class of *sequential CFSMs*, where at any moment only one machine is allowed to send messages. In this simple class, we can directly identify a sound and complete characterisation of *sequential multiparty session types*, a subclass of multiparty session types studied in [5, 13]. The complexity of the synthesis is linear in the size of the CFSMs.

§ 5 Our first result is about the main class of global session types [4, 25], called *classical multiparty session types*, where all choices are sent to and received from the same peer, and where there is no parallel branches. We identify a sound and complete condition, *multiparty compatibility* for this class and give an algorithm for the synthesis of global types from local types. The complexity of the compatibility checking and synthesis is both bounded in polynomial in the size of the CFSMs.

§ 6 We extend our result to *generalised multiparty session types*, a recent class of multiparty session types [17] with graph-like control flow and parallelism. The same multiparty compatibility as in § 5 can be used without modification, although well-formedness condition need to be generalised. The synthesis algorithm relies on Petri net intermediate representations [15] and 1-bounded behavioural exploration. Our result is applicable to generate a core part of Choreography BPMN 2.0 specification [7] from CFSMs.

§ 7 discusses related work and § 8 concludes with the future work. Appendix [30] lists the full proofs and omitted definitions.

2. Communicating Finite State Machines

This section gives some preliminary notations of CFSMs (following [13]) and the definitions of some CFSMs properties relevant to the CFSM connection to multiparty session types.

2.1 Basic definitions and properties

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: $\mathcal{P} \subseteq \{\texttt{Alice}, \texttt{Bob}, \texttt{Carol}, \ldots, \texttt{p}, \texttt{q}, \ldots\}$.

DEFINITION 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 $\mathtt{q}$. $\ell, \ell', \ldots$ 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. A state $q \in Q$ which does not have any outgoing transition is called a *final* state. If q has both sending and receiving outgoing transitions, then q is called *mixed*. A sending (resp. receiving) state q is said to be *alternating* if all the outgoing transitions go to receiving (resp. sending) states. We say q is *directed* if it contains only sending (resp. receiving) actions to (resp. from) the same 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 $q' = q''$.[1]

We now define communicating systems of machines.

DEFINITION 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 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 $\mathtt{pq}$, while (2-b) gets the content a from a channel $\mathtt{pq}$. 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 the metavariable φ to designate sequences of transitions of the form $t_1 \cdots t_m$. 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 $Tr(S) = \{act(\varphi) \mid \exists s \in RS(S), s_0 \xrightarrow{\varphi} s\}$.

Properties We now define several properties about communicating systems and their configurations. These properties will be used in the following sections to characterise the systems that correspond to multiparty session types.

[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 [13] in order to represent branching type constructs.

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 [13, 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 $\mathtt{pq}$.

A sequence of transitions (an execution) $s_1 \xrightarrow{t_1} s_2 \cdots s_m \xrightarrow{t_m} s_{m+1}$ is said to be *k-bounded* if all channels of all intermediate configurations s_i do not contain 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 a 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 exceeds the number of receive actions by n. We then define the equivalences:

- $S \approx S'$ is $\forall \varphi,\ \varphi \in Tr(S) \Leftrightarrow \varphi \in Tr(S')$
- $S \approx_n S'$ is $\forall \varphi,\ bound(\varphi) \leq n \Rightarrow (\varphi \in Tr(S) \Leftrightarrow \varphi \in Tr(S'))$

$\approx$ and $\approx_n$ will be used to study the characterisation of several classes of CFSMs that correspond to types.

We now define the notion of stable systems, which always allow 1-buffer executions, and deterministic systems, where traces unequivocally describe the behaviour. Determinism is the condition under which the trace equivalence (presented above) is enough to fully describe the equivalence in behaviour between systems.

DEFINITION 4 (stability and determinism). 1. A communicating system 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 $s_0 \xrightarrow{\varphi''} s'$.
2. A communicating system S is *deterministic* if, for all $s \in RS(S)$, $s \xrightarrow{t_1} s_1$ and $s \xrightarrow{t_2} s_2$ and $act(t_1) = act(t_2)$ imply $s_1 = s_2$.

Note that the communicating system of Figure 1 is both stable and deterministic.

The following key properties will be examined throughout the paper as the properties that multiparty session type can enforce. They are undecidable in general CFSMs.

DEFINITION 5 (safety and liveness). 1. A communicating system S is *deadlock-free* (resp. *orphan message-free*, *reception error-free*) if $s \in RS(S)$, s is not a deadlock (resp. orphan message, unspecified reception) configuration.
2. A communicating system S satisfies the *liveness property*[2] if for all $s \in RS(S)$, there exists $s \longrightarrow^* s'$ such that s' is final.

3. Classical global and local types

This section presents the classical multiparty session types, our main object of study. For the syntax of types, we follow [4] which is the most widely used syntax in the MPST literature. We then introduce two labelled transition systems, one for local types and one for global types, and we show the equivalence between local types and communicating automata.

[2] The terminology follows [11].

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

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

$a_j \in \mathbb{A}$ 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. 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 should follow. We require $\mathtt{p} \neq \mathtt{p}'$ to prevent self-sent messages. Recursive type $\mu \mathtt{t}.G$ is for recursive protocols, assuming that type variables $(\mathtt{t}, \mathtt{t}', \ldots)$ 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 $\mathtt{p}?a$ or $\mathtt{p}?a$ for local.

We give below, in section 3.1, a formal interpretation of which behaviours global and local types represent. Beforehand, we give the definition of the classical projection algorithm.

Projection The relation between global and local types is formalised by projection. Instead of the restricted original projection [4], we use the extension with the merging operator $\bowtie$ from [16, 38]: it allows each branch of the global type to actually contain different interaction patterns.

DEFINITION 6 (projection). The *projection of G onto $\mathtt{p}$* (written $G \upharpoonright \mathtt{p}$) is defined as:

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

$$(\mu \mathtt{t}.G) \upharpoonright \mathtt{p} = \begin{cases} \mu \mathtt{t}.G \upharpoonright \mathtt{p} & \mathtt{t} \in G \\ \mathsf{end} & \mathtt{t} \notin G \end{cases} \qquad \mathtt{t} \upharpoonright \mathtt{p} = \mathtt{t} \qquad \mathsf{end} \upharpoonright \mathtt{p} = \mathsf{end}$$

The mergeability relation $\bowtie$ is the smallest congruence relation over end-point types such that:

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

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:

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

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 G's projection is defined.

EXAMPLE 1 (Commit). As a simple example (three parties), consider the following global type:

$$\mu \mathtt{t}.\mathtt{Alice} \to \mathtt{Bob} : \{ \begin{array}{l} \mathit{act}.\mathtt{Bob} \to \mathtt{Carol} : \{\mathit{prep}.\mathtt{Alice} \to \mathtt{Carol} : \mathit{commit}.\mathtt{t}\ \}, \\ \mathit{quit}.\mathtt{Bob} \to \mathtt{Carol} : \{\mathit{save}.\mathtt{Alice} \to \mathtt{Carol} : \mathit{finish}.\mathsf{end}\} \} \end{array}$$

Then $\mathtt{Carol}$'s local type is given as:

$$\mu \mathtt{t}.\mathtt{Bob}?\{\mathit{prep}.\mathtt{Alice}?\{\mathit{commit}.\mathtt{t}\},\ \mathit{save}.\mathtt{Alice}?\{\mathit{finish}.\mathsf{end}\}\}$$

PROPOSITION 1 (well-formedness). *The time complexity of the projection algorithm is polynomial with respect to the size of G.*

3.1 Labelled transitions of classical global and local types

This subsection defines new labelled transition relations (LTS) for classical global and local types. In the previous literature [4, 25], the semantics of global and local types is not defined: they are usually indirectly given by the π-processes they type. Here, we propose their explicit semantics and show that the LTS of the local types obtained by projection from a global type is identical to the LTS of that global type. This result is essential to prove the sound and complete characterisations of classical global types.

LTS over classical global types 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 $\mathtt{p}$ to $\mathtt{p}'$: $\ell ::= \mathtt{pp}'!a \mid \mathtt{pp}'?a$

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

DEFINITION 7 (LTS over classical global types). The relation $G \xrightarrow{\ell} G'$ is defined as ($\mathit{subj}(\ell)$ is defined in § 2.1):

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

$$[\mathrm{GR2}] \quad \mathtt{p} \rightsquigarrow \mathtt{p}' \colon a.G \xrightarrow{\mathtt{pp}'?a} G \qquad [\mathrm{GR3}] \ \frac{G[\mu \mathtt{t}.G/\mathtt{t}] \xrightarrow{\ell} G'}{\mu \mathtt{t}.G \xrightarrow{\ell} G'}$$

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

$$[\mathrm{GR5}] \ \frac{G \xrightarrow{\ell} G' \quad \mathtt{q} \notin \mathit{subj}(\ell)}{\mathtt{p} \rightsquigarrow \mathtt{q} \colon a.G \xrightarrow{\ell} \mathtt{p} \rightsquigarrow \mathtt{q} \colon a.G'}$$

[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 (which allows some limited degree of concurrency), where the syntactic order of messages is enforced only for the participants that are involved. For example, in the case 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) 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 $\ell_4 = AC?b$. For The LTS allows the following three sequences of transitions:

$$\begin{array}{lllllll} G_1 & \xrightarrow{\ell_1} & \mathtt{A} \rightsquigarrow \mathtt{B} : a.\mathtt{A} \to \mathtt{C} : b.\mathsf{end} & \xrightarrow{\ell_2} & \mathtt{A} \to \mathtt{C} : b.\mathsf{end} & & \\ & \xrightarrow{\ell_3} & \mathtt{A} \rightsquigarrow \mathtt{C} : b.\mathsf{end} & \xrightarrow{\ell_4} & \mathsf{end} & & \\ G_1 & \xrightarrow{\ell_1} & \mathtt{A} \rightsquigarrow \mathtt{B} : a.\mathtt{A} \to \mathtt{C} : b.\mathsf{end} & & & & \\ & \xrightarrow{\ell_3} & \mathtt{A} \rightsquigarrow \mathtt{B} : a.\mathtt{A} \rightsquigarrow \mathtt{C} : b.\mathsf{end} & \xrightarrow{\ell_2} & \mathtt{A} \rightsquigarrow \mathtt{C} : b.\mathsf{end} & \xrightarrow{\ell_4} & \mathsf{end} \\ G_1 & \xrightarrow{\ell_1} & \mathtt{A} \rightsquigarrow \mathtt{B} : a.\mathtt{A} \to \mathtt{C} : b.\mathsf{end} & & & & \\ & \xrightarrow{\ell_3} & \mathtt{A} \rightsquigarrow \mathtt{B} : a.\mathtt{A} \rightsquigarrow \mathtt{C} : b.\mathsf{end} & \xrightarrow{\ell_4} & \mathtt{A} \rightsquigarrow \mathtt{B} : a.\mathsf{end} & \xrightarrow{\ell_2} & \mathsf{end} \end{array}$$

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 classical local types We now 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 8 (LTS over classical local types). The relation $T \xrightarrow{\ell} T'$, for the local type of role $\mathtt{p}$, is defined as:

$$[\text{LR1}] \quad \mathtt{q}!\{a_i.T_i\}_{i\in I} \xrightarrow{\mathtt{pq}!a_i} T_i$$

$$[\text{LR2}]\ \mathtt{q}?\{a_i.T_i\}_{i\in I} \to \mathtt{q}?a_j.T_j \quad (j\in I) \qquad [\text{LR3}]\ \mathtt{q}?a.T \xrightarrow{\mathtt{qp}?a} T$$

$$[\text{LR4}]\ \frac{T[\mu \mathtt{t}.T/\mathtt{t}] \xrightarrow{\ell} T'}{\mu \mathtt{t}.T \xrightarrow{\ell} T'}$$

The semantics of a local type follows the intuition that every action of the local type should obey the syntactic order. The only non-intuitive part are the rules [LR2, LR3] which differentiate the silent moment when a branch is chosen, from the moment the message is actually received. The reason for that difference is to reflect the fact that a branch is chosen at the selection point, and that the reception only acknowledges that choice.

We now define the LTS for collections of local types.

DEFINITION 9 (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 $\mathtt{p}'\mathtt{q}' \neq \mathtt{pq}$.

while the silent transitions $s_T \to s'_T = (\vec{T}';\vec{w}')$ are of the form:

3 $T_\mathtt{q} = \mathtt{p}?\{a_i.T_i\}_{i\in I} \to T'_\mathtt{q} = \mathtt{p}?a_j.T_j$ and (a) $T'_{\mathtt{p}'} = T_{\mathtt{p}'}$ for all $\mathtt{p}' \neq \mathtt{q}$; and (b) $w_\mathtt{pq} = w'_\mathtt{pq} = w''_\mathtt{pq}\cdot a_j$ 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, purposefully following the definition of the semantics of CFSMs. $w_\mathtt{pq}$ represents the FIFO queue at channel $\mathtt{pq}$. The last definition (3) allows a local type to silently get ready to input a value from a queue.

Traces 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 § 2.1 to global types and configurations of local types.

3.2 Soundness and completeness of operational semantics

This subsection states that the trace of a global type corresponds exactly to the traces of its projected local types.

First, to represent the intermediate states of global types, we need to extend projection to be a relation between global types extended with $\mathtt{p} \rightsquigarrow \mathtt{p}' : a.G$, and configurations of local types.

We define that the projected configuration $[\![G]\!]$ of a global type G is 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:

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

and where the projection algorithm $\upharpoonright \mathtt{q}$ is extended by:

$$\mathtt{p} \rightsquigarrow \mathtt{p}' : a.G \upharpoonright \mathtt{q} = \begin{cases} \mathtt{p}?a.G \upharpoonright \mathtt{q} & \mathtt{q} = \mathtt{p}' \\ G_j \upharpoonright \mathtt{q} & \text{otherwise} \end{cases}$$

We now define formally the soundness and completeness of projection with respect to the LTSs defined above.

THEOREM 2 (soundness and completeness of classical projection). *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 $G \approx (\vec{T};\vec{\varepsilon})$.*

Proof. See Appendix A.1. □

3.3 Translation between local types and CFSMs

Now that we have equipped local types with a transition system, we can 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. We write $T' \in T$ if T' occurs in T.

DEFINITION 10 (translation from local types to CFSMs). 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:

- $Q = \{T' \mid T' \in T_0,\ T' \neq \mathtt{t}\}$.
- $C = \{\mathtt{pq} \mid \mathtt{p},\mathtt{q} \in G\}$; $q_0 = T_0$; and $\mathbb{A}$ is the set of $\{a \in G\}$
- δ is defined as:

  If $T = \mathtt{p}'!\{a_j.T_j\}_{j\in J} \in T_0$,

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

  If $T = \mathtt{p}'?\{a_j.T_j\}_{j\in J} \in T_0$,

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

Local types are simple structures and their translation to CFSM preserves some of them:

PROPOSITION 3 (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 11 (translation from a basic CFSM to a local type). Fix $M_\mathtt{p} = (Q, C, q_0, \mathbb{A}, \delta)$ and assume $M_\mathtt{q}$ is basic. Then 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:

- $\mathcal{T}_{\tilde{q}}(q) = \mu \mathtt{t}_i.\mathtt{p}'!\{a_j.\mathcal{T}^\circ_{\tilde{q}\cdot q}(q_j)\}_{j\in J}$ if $(q, \mathtt{pp}'!a_j, q_j) \in \delta$
- $\mathcal{T}_{\tilde{q}}(q) = \mu \mathtt{t}_i.\mathtt{p}'?\{a_j.\mathcal{T}^\circ_{\tilde{q}\cdot q}(q_j)\}_{j\in J}$ if $(q, \mathtt{p}'\mathtt{p}?a_j, q_j) \in \delta$
- $\mathcal{T}^\circ_{\tilde{q}}(q) = \mathsf{end}$ if q is final;
- $\mathcal{T}^\circ_{\tilde{q}}(q) = \mathtt{t}_k$ if $(q, \ell, q_k) \in \delta$ and $q_k \in \tilde{q}$
- $\mathcal{T}^\circ_{\tilde{q}}(q) = \mathcal{T}_{\tilde{q}}(q)$ otherwise.

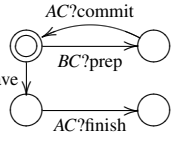
Finally, we replace $\mu \mathtt{t}.T$ by T if $\mathtt{t}$ is not in T.

The complexity of these translations are polynomial with respect to the size of the local type or basic CFSM, respectively. We now prove that the translations preserve the semantics.

PROPOSITION 4 (translations between CFSMs and local types). *If a CFSM M is basic, then $M \approx \mathcal{T}(M)$. If T is a classical local type, then $T \approx \mathcal{A}(T)$.*

EXAMPLE 2 (Commit). Following up on example 1, `Carol`'s local type and its equivalent CFSM are:

$$\mu \mathtt{t}.\mathtt{Bob}?\{\mathit{prep}.\mathtt{Alice}?\{\mathit{commit}.\mathtt{t}\},\ \mathit{save}.\mathtt{Alice}?\{\mathit{finish}.\mathsf{end}\}\}$$

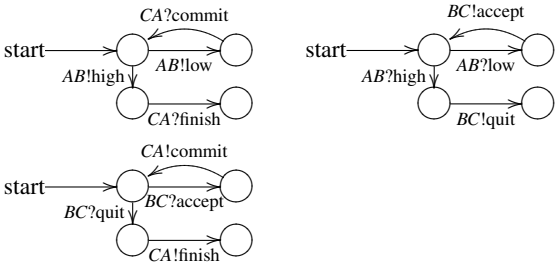


4. Sequential Multiparty Session Automata

This section examines the soundness and completeness characterisation of the sequential multiparty session automata class, defined by the projection from sequential global types, and which corresponds 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 for cryptographic protocols [5] (although it was later extended with concurrency [32]), as well as modellings radio communications where transceivers of several machines use the same frequency [13].

DEFINITION 12 (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 that all but one of the starting states are receiving states.

EXAMPLE 3 (sequential system). We give a simple sequential system which is extended from the binary example in § 1.



The corresponding global type is:

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

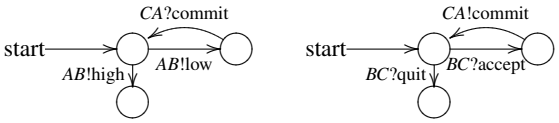
Sequential systems always start with a unique machine able to output. This condition implies that sequential systems of CFSMs are always deterministic. The alternation allows to prove that all accepted traces of sequential systems are 1-bounded.

PROPOSITION 5 (boundedness of sequential systems). *Sequential systems are 1-bounded.*

PROPOSITION 6 (safety and liveness in sequential systems). *For sequential systems, the safety and liveness properties of Definition 5 are decidable.*

Proof. By Proposition 5, 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 [13]. □

EXAMPLE 4 (properties of sequential systems). We show sequential systems which do not satisfy the safety properties. For example, if we replace A and C to:



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

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

which does not satisfy the mergeability condition so that it is not well-formed. If we replace only A by the above automaton, then it has an orphan message $CA!\mathit{finish}$ in the queue. In Theorem 9, 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.

Now we define sequential global (and local) types, and show the trace equivalence with sequential communicating systems.

DEFINITION 13 (sequential global type). A well-formed classical global type G is *sequential* if for any subterm $\mathtt{p}_1 \to \mathtt{p}_2 : \{a_j.\mathtt{p}_{3\mathtt{i}} \to \mathtt{p}_{4\mathtt{i}} : \{b_i.G_i\}_{i\in I}\}_{j\in J} \in G'$ 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 7 (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* $\mathtt{p}_1' = \mathtt{p}_2$.

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 8 (synthesis of sequential systems [14]). *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:

- if $(q_1, ..., q_n)$ has been visited before, the global type is $\mathtt{t}_{q_1,...,q_n}$;
- 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 $(q_k, \mathtt{pp}'!a_i, q_i) \in \delta_\mathtt{p}$ for $i \in I$.
  - we check that machine $\mathtt{p}'$ is in a receiving state q_m such that $(q_m, \mathtt{pp}'?a_j, q_j') \in \delta_{\mathtt{p}'}$ with $j \in J$ and $I \subseteq J$. Otherwise the algorithm returns false.
  - we check that each q_i' $(i \in I)$ in machine $\mathtt{p}'$ are sending states or final. Otherwise the algorithm returns false.
  - we set $\mu\mathtt{t}_{q_1,...,q_n}.\mathtt{p} \to \mathtt{p}' : \{a_i.G(q_1, ..., q_k \leftarrow q_i, ..., q_m \leftarrow q_i', ..., q_n)\}_{i\in I}$ and continue by recursive calls. If q_i' is final, we set it to end.
- we erase unnecessary $\mu\mathtt{t}$ if $\mathtt{t}$ does not appear in G. □

Given a sequential S, the time complexity of the synthesis algorithm is linear with respect to the size of S.

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

THEOREM 9 (soundness and 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 $S \approx G$. Conversely, if G*

is sequential, there exists a sequential S which satisfies the and liveness properties (deadlock-freedom, reception error-freedom and orphan message-freedom), and $S \approx G$.

Proof. By Proposition 5, the sequential system S terminates either with 1 message in one queue or all are empty. Another possibility is that of deadlock. The first part of the statement is proved by Theorem 8 noting that if S is deadlock-free and orphan message-free, the algorithm always returns some sequential G. The second part is by Theorem 2 and Proposition 4 by setting $M_{\mathtt{p}} = \mathcal{A}(G \upharpoonright \mathtt{p})$. □

5. Classical multiparty session automata

This section studies the synthesis and sound and complete characterisation of the *classical multiparty session automata* (*CMSA*), which are generated from well-formed classical global types. We first note that basic CFSMs correspond to the natural generalisation of half-duplex systems [13, § 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 5 are however undecidable [13, Theorem 36]. We therefore need to find a stronger (and decidable) property to force basic CFSMs to behave as if they were the result of a projection from classical global types.

5.1 Multiparty compatibility

In the two machines case, there exists a sound and complete condition called *compatible* [22]. Let us define the isomorphism $\Phi : (C \times \{!,?\} \times \Sigma)^* \longrightarrow (C \times \{!,?\} \times \Sigma)^*$ such that $\Phi(j?a) = j!a$, $\Phi(j!a) = j?a$, $\Phi(t_1 \cdots t_n) = \Phi(t_1) \cdots \Phi(t_n)$, and $\Phi(\varepsilon) = \varepsilon$. 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 it were one CFSM which offers compatible traces $\Phi(Tr(M_{\mathtt{p}}))$, up to internal synchronisations (i.e. 1-bounded executions). To formalise this idea, we define a way to group CFSMs.

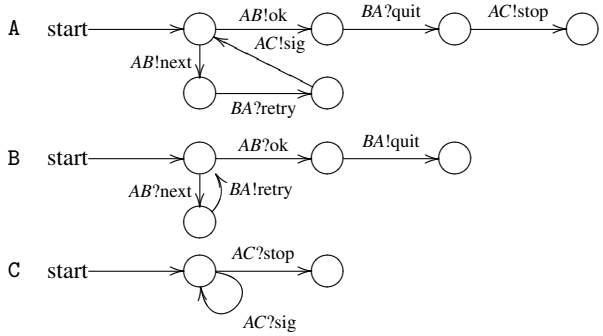
DEFINITION 14 (Definition 37, [13]). Let $M_i = (Q_i, C_i, q_{0i}, \Sigma_i, \delta_i)$. The *associated CFSM* of $S = (M_1, .., M_n)$ is $M = (Q, C, q_0, \Sigma, \delta)$ such that: $Q = Q_1 \times Q_2 \times \cdots \times Q_n$, $q_0 = (q_{01}, \ldots, q_{0n})$ and δ is the least relation verifying: $((q_1, ..., q_i, ..., q_n), \ell, (q_1, ..., q_i', ..., q_n)) \in \delta$ if $(q_i, \ell, q_i') \in \delta_i$ $(1 \leq i \leq n)$.

We now define below the 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 15 (multiparty compatible system). A system $S = (M_1, .., M_n)$ $(n \geq 2)$ is *multiparty compatible* if for any sequence of actions $\ell_1 \cdots \ell_k$ in M_i, there is a sequence of actions $\varphi_1 \cdot t_1 \cdot \varphi_2 \cdot t_2 \cdot \varphi_3 \cdots \varphi_k \cdot t_k$ from a CFSM corresponding to $S^{-i} = (M_1, .., M_{i-1}, M_{i+1}, .., M_n)$ where φ_j is empty or alternation, $\ell_j = \Phi(act(t_j))$ and $i \notin act(\varphi_j)$ and $1 \leq j \leq k$ (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 τ-actions. Note that sequential, orphan message-free and deadlock-free CFSMs in § 4 satisfy multiparty compatibility.

EXAMPLE 5 (multiparty compatibility). A simple example of basic automata follows:



Its corresponding global type is:

$$\mu \mathtt{t}.\mathtt{A} \to \mathtt{B}\{\quad ok.\mathtt{B} \to \mathtt{A} : quit.\mathtt{A} \to \mathtt{C} : stop.\mathtt{end}$$
$$next.\mathtt{B} \to \mathtt{A} : retry.\mathtt{A} \to \mathtt{C} : signal.\mathtt{t}\}$$

The compatibility from the viewpoint of A and B is trivial so we examine the compatibility from C. To check the compatibility for the action $AC?\mathrm{stop}$, we perform 1-bound execution such that $act(\varphi_1) = AB!\mathrm{stop} \cdot AB?\mathrm{stop} \cdot BA!\mathrm{quit} \cdot BA?\mathrm{quit}$ and $act(t_1) = AC!\mathrm{stop}$ from A; and to check the compatibility for the action $AC?\mathrm{sig}$, we perform 1-bound execution such that $act(\varphi_2) = AB!\mathrm{next} \cdot AB?\mathrm{next} \cdot BA!\mathrm{retry} \cdot BA?\mathrm{retry}$ and $act(t_2) = AC!\mathrm{sig}$ from A.

Our first goal is to prove that if M_i and compatible S^{-i} interact together, they are deadlock-free. The following lemma is useful for our proofs. 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 $act(t_2) = act(t_1')$.

LEMMA 10 (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 $a_1 \neq a_2$ with $s_1 \neq s_2$ or $s_1 = s_2$;*
   (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 $\mathtt{p}_1 \neq \mathtt{p}_2$ 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. By a straightforward case analysis. □

With this lemma, we can prove that, in a multiparty compatible basic system S, there exists a 1-bounded execution between M_i and S^{-i}, as we can permute ℓ_j (the action by M_i) and φ_j (the internal actions by S^{-i}) by the diamond property.

The next proposition states that after some appropriate executions, *any* reachable state can be translated into a 1-bounded execution. See § 2.1(1) for the definition of a stable property. In addition, they satisfy the three safety properties of Definition 5.

PROPOSITION 11. *Assume $S = (M_\mathtt{p})_{\mathtt{p}\in\mathcal{P}}$ is basic and multiparty compatible. Then S is stable, deadlock-free, orphan message-free and reception error-free.*

Proof. See Appendix B.1. We use Lemma 10. □

By Proposition 11, we can deduce liveness:

PROPOSITION 12 (liveness). *Suppose $S = \{M_\mathtt{p}\}_{\mathtt{p}\in\mathcal{P}}$ is basic and multiparty compatible. If there exists at least one $M_\mathtt{q}$ which includes a final state. Then S satisfies the liveness property.*

Finally, we check that multiparty compatibility is decidable.

PROPOSITION 13. *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. 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 other the other machines $\mathtt{p}'$ such that $\mathtt{p}' \in \mathcal{P} \setminus \mathtt{p}$. □

5.2 Synthesis of classical multiparty session automata

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-buffer executions, as it is the case in multiparty session types.

LEMMA 14 (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 execusion 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 do 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 B.2. □

The main theorems of this section follow. The synthesis algorithm for basic systems is a simple extension of the one for sequential systems.

THEOREM 15 (synthesis of classical systems). *Suppose S is a basic system. 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 $S = \{M_\mathtt{p}\}_{\mathtt{p}\in\mathcal{P}}$. We extend the algorithm in Theorem 8. The algorithm starts from the initial states of all machines $(q^{\mathtt{p}_1}{}_0, ..., q^{\mathtt{p}_n}{}_0)$.

Then we look at 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 7, 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 $I \subseteq J$. Otherwise the algorithm returns false.
  - we set $\mu\mathtt{t}_{q_1,...,q_n}.\mathtt{p} \to \mathtt{q}\colon \{a_i.G(\{q_1, ..., q_\mathtt{p} \leftarrow q_i, ..., q_\mathtt{q} \leftarrow q_i', ..., q_n\})\}_{i\in I}$ 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 $\mathtt{t}$ does not appear in G and check G satisfies Definition 6.

Since the algorithm only explores 1-bounded executions, the reconstructed G satisfies $G \approx_1 S$. By Theorem 2, we know that $G \approx (\{G \upharpoonright \mathtt{p}\}_{\mathtt{p}\in\mathcal{P}}; \vec{\varepsilon})$. Hence, by Proposition 4, 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 14, we get that $S' \approx S'$ and therefore that $S \approx G$. □

Considering that the complexity of the 1-bounded reachability is polynomial given S, the time complexity of the algorithm and multiparty compatibility is polynomial-bound with respect to the size of S.

We can now conclude our characterisation of classical multiparty session types in terms of basic, multiparty compatible communicating systems.

THEOREM 16 (soundness and completeness in CMSA). *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 S which satisfies the safety properties (deadlock-freedom, reception error-freedom and orphan message-freedom) and $S \approx G$.*

Proof. The first direction is by Theorem 15 and the second direction is by Theorem 2 and Proposition 4 (the safety properties are obtained by the fact that the classical global types are a special case of general global types in [17]). □

6. Generalised Multiparty Session Automata

In this section, we extend the results obtained on classical multiparty session types to tackle generalised multiparty session types [17], an extension with new features such as flexible fork, choice, merge and join operations for precise flow specification. It strictly subsumes classical MPST.

6.1 Generalised global and local types

In this subsection, we recall definitions from [17].

Generalised global types We first define *generalised global types*. The syntax is defined below.

$\mathbf{G}$	$::=$	$\mathsf{def}\ \widetilde{G}\ \mathsf{in}\ \mathbf{x}$	Global type		
G	$::=$	$\mathbf{x} = \mathtt{p} \to \mathtt{p}' : a\,; \mathbf{x}'$	Messages	$\mathbf{x} = \mathsf{end}$	End
		$\mathbf{x} = \mathbf{x}' \mid \mathbf{x}''$	Fork	$\mathbf{x} = \mathbf{x}' + \mathbf{x}''$	Choice
		$\mathbf{x} \mid \mathbf{x}' = \mathbf{x}''$	Join	$\mathbf{x} + \mathbf{x}' = \mathbf{x}''$	Merge

$$
\begin{array}{rcl}
\mathbf{G} = \mathsf{def}\ \mathbf{x}_0 & = & \mathbf{x}_1 \mid \mathbf{x}_2 \\
\mathbf{x}_1 + \mathbf{x}_5 & = & \mathbf{x}_3 \\
\mathbf{x}_3 & = & \mathtt{A} \to \mathtt{B} : \mathit{data}\ ; \mathbf{x}_4 \\
\mathbf{x}_4 & = & \mathbf{x}_5 + \mathbf{x}_6 \\
\mathbf{x}_6 & = & \mathtt{A} \to \mathtt{B} : \mathit{eof}\ ; \mathbf{x}_7 \\
\mathbf{x}_2 & = & \mathtt{A} \to \mathtt{C} : \mathit{log}\ ; \mathbf{x}_8 \\
\mathbf{x}_7 \mid \mathbf{x}_8 & = & \mathbf{x}_9 \\
\mathbf{x}_9 & = & \mathtt{B} \to \mathtt{C} : \mathit{save}\ ; \mathbf{x}_{10} \\
\mathbf{x}_{10} & = & \mathsf{endin}\ \mathbf{x}_0
\end{array}
$$

Data transfer example

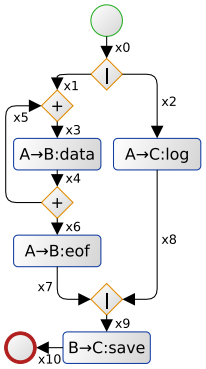


Figure 2. Generalised global type and graph representation

A global type $\mathbf{G} = \mathsf{def}\ \widetilde{G}\ \mathsf{in}\ \mathbf{x}_0$ describes an interaction between a fixed number of participants. We explain each of the constructs by example, in Figure 2, alongside the corresponding graphical representation inspired by the BPMN 2.0 business processing language. This example features three participants, with *A* sending data to *B* while *C* concurrently records a log entry of the transmission.

The prescribed interaction starts from $\mathbf{x}_0$, which we call the *initial state* (in green in the graphical representation), and proceeds according to the transitions specified in $\widetilde{G}$ (the diamond or boxes operators in the picture). The *state variables* $\mathbf{x}$ in $\widetilde{G}$ (the edges in the graph) represent the successive distributed states of the interaction. Transitions can be *message exchanges* of the form $\mathbf{x}_3 = \mathtt{A} \to \mathtt{B} : \mathit{data}\ ; \mathbf{x}_4$ where this transition specifies that *A* can go from $\mathbf{x}_3$ to the continuation $\mathbf{x}_4$ by sending message *data*, while *B* goes from $\mathbf{x}_3$ to $\mathbf{x}_4$ by receiving it. In the graph, message exchanges are represented by boxes with exactly one incoming and one outgoing edges. $\mathbf{x}_4 = \mathbf{x}_5 + \mathbf{x}_6$ represents the choice between continuing with $\mathbf{x}_5$ or $\mathbf{x}_6$ and $\mathbf{x}_0 = \mathbf{x}_1 \mid \mathbf{x}_2$ represents forking the interactions, allowing the interleaving of actions at $\mathbf{x}_1$ and $\mathbf{x}_2$. These forking threads are eventually collected by joining construct of the form $\mathbf{x}_7 \mid \mathbf{x}_8 = \mathbf{x}_9$. Similarly choices (i.e. mutually exclusive paths) are closed by merging construct $\mathbf{x}_1 + \mathbf{x}_5 = \mathbf{x}_3$, where they share a continuation. Forks, choices, joins and merges are represented by diamond ternary operators in the graphical notation. Fork and choice have one input and two outputs, join and merge have two inputs and one output. Fork and join use the diamond operator with the $\mid$ symbol, while choice and merge use a diamond with the $+$ symbol. The $\mathbf{x}_{10} = \mathsf{end}$ transition is represented by a red circle. Note that the two representations (syntax and graph) are equivalent.

The motivation behind this choice of syntax is to support general control flows, as classical global type syntax tree, even with added operators fork $\mid$ and choice $+$ [4, 11, 16, 25], is limited to series-parallel control flow graphs.

Generalised local types As for global types, a local type $\mathbf{T}$ follows a shape of a state machine-like definition: local types are of the form $\mathsf{def}\ \widetilde{T}\ \mathsf{in}\ \mathbf{x}_0$. The different actions include send ($\mathtt{p}!a$ is the action of sending to $\mathtt{p}$ a message *a*), receive ($\mathtt{p}?a$ is the action of receiving from $\mathtt{p}$ a message *a*), fork, internal choice, external choice, join, merge, indirection and end. Note that merge is used for both internal and external choices. Similarly to global types, an obvious graphical representation exists.

$$
\begin{array}{lcll}
\mathbf{T} & ::= & \mathsf{def}\ \widetilde{T}\ \mathsf{in}\ \mathbf{x} & \text{local type} \\
T & ::= & \mathbf{x} = \mathtt{p}!a.\mathbf{x}' & \text{send} \\
 & \mid & \mathbf{x} = \mathtt{p}?a.\mathbf{x}' & \text{receive} \\
 & \mid & \mathbf{x} = \mathbf{x}' \mid \mathbf{x}'' & \text{fork} \\
 & \mid & \mathbf{x}' \mid \mathbf{x}' = \mathbf{x}'' & \text{join} \\
 & \mid & \mathbf{x} = \mathsf{end} & \text{end} \\
 & \mid & \mathbf{x} = \mathbf{x}' \oplus \mathbf{x}'' & \text{internal choice} \\
 & \mid & \mathbf{x} = \mathbf{x}'\ \&\ \mathbf{x}'' & \text{external choice} \\
 & \mid & \mathbf{x} + \mathbf{x}' = \mathbf{x}'' & \text{merge} \\
 & \mid & \mathbf{x} = \mathbf{x}' & \text{indirection}
\end{array}
$$

$$
\frac{\mathbf{x}=\mathtt{p}\to\mathtt{p}':a\ ;\mathbf{x}'\in\widetilde{G} \quad \mathbf{X}_\mathtt{p}=\mathbf{X}[\mathbf{x}] \quad w_{\mathtt{p}\mathtt{p}'}\in\tilde{w}}{\mathsf{def}\ \widetilde{G}\ \mathsf{in}\ \tilde{\mathbf{X}},\tilde{w} \xrightarrow{\mathtt{p}\mathtt{p}'!a} \mathsf{def}\ \widetilde{G}\ \mathsf{in}\ \tilde{\mathbf{X}}[\mathbf{X}_\mathtt{p}\leftarrow\mathbf{X}[\mathbf{x}']],\tilde{w}[w_{\mathtt{p}\mathtt{p}'}\leftarrow w_{\mathtt{p}\mathtt{p}'}\cdot a]}\ \lfloor\text{GGR1}\rfloor
$$

$$
\frac{\mathbf{x}=\mathtt{p}\to\mathtt{p}':a\ ;\mathbf{x}'\in\widetilde{G} \quad \mathbf{X}_{\mathtt{p}'}=\mathbf{X}[\mathbf{x}] \quad w_{\mathtt{p}\mathtt{p}'}\in\tilde{w} \quad w_{\mathtt{p}\mathtt{p}'}=a\cdot w'_{\mathtt{p}\mathtt{p}'}}{\mathsf{def}\ \widetilde{G}\ \mathsf{in}\ \tilde{\mathbf{X}},\tilde{w} \xrightarrow{\mathtt{p}\mathtt{p}'?a} \mathsf{def}\ \widetilde{G}\ \mathsf{in}\ \tilde{\mathbf{X}}[\mathbf{X}_{\mathtt{p}'}\leftarrow\mathbf{X}[\mathbf{x}']],\tilde{w}[w_{\mathtt{p}\mathtt{p}'}\leftarrow w'_{\mathtt{p}\mathtt{p}'}]}\ \lfloor\text{GGR2}\rfloor
$$

$$
\frac{\mathbf{x}=\mathtt{p}\to\mathtt{p}':a\ ;\mathbf{x}'\in\widetilde{G} \quad \mathbf{X}_\mathtt{q}=\mathbf{X}[\mathbf{x}] \quad \mathtt{q}\notin\{\mathtt{p},\mathtt{p}'\} \quad \mathsf{def}\ \widetilde{G}\ \mathsf{in}\ \tilde{\mathbf{X}}[\mathbf{X}_\mathtt{q}\leftarrow\mathbf{X}[\mathbf{x}']],\tilde{w}\xrightarrow{\ell}\mathsf{def}\ \widetilde{G}\ \mathsf{in}\ \tilde{\mathbf{X}}',\tilde{w}'}{\mathsf{def}\ \widetilde{G}\ \mathsf{in}\ \tilde{\mathbf{X}},\tilde{w}\xrightarrow{\ell}\mathsf{def}\ \widetilde{G}\ \mathsf{in}\ \tilde{\mathbf{X}}',\tilde{w}'}\ \lfloor\text{GGR3}\rfloor
$$

$$
\frac{\mathbf{X}_\mathtt{p}=\mathbf{X} \quad \mathbf{X}\equiv_{\widetilde{G}}\mathbf{X}' \quad \mathsf{def}\ \widetilde{G}\ \mathsf{in}\ \tilde{\mathbf{X}}[\mathbf{X}_\mathtt{p}\leftarrow\mathbf{X}'],\tilde{w}\xrightarrow{\ell}\mathsf{def}\ \widetilde{G}\ \mathsf{in}\ \tilde{\mathbf{X}}',\tilde{w}'}{\mathsf{def}\ \widetilde{G}\ \mathsf{in}\ \tilde{\mathbf{X}},\tilde{w}\xrightarrow{\ell}\mathsf{def}\ \widetilde{G}\ \mathsf{in}\ \tilde{\mathbf{X}}',\tilde{w}'}\ \lfloor\text{GGR4}\rfloor
$$

Figure 3. Global LTS

The local types are obtained from the global type by successive projection to each participant. We define the projection of a well-formed global type $\mathbf{G}$ to the local type of participant $\mathtt{p}$ (written $\mathbf{G} \upharpoonright \mathtt{p}$). The projection is given in Appendix C because it is straightforward: for example, $\mathbf{x} = \mathtt{p} \to \mathtt{q} : a\ ; \mathbf{x}'$ is projected to the output $\mathbf{x} = \mathtt{p}'!a.\mathbf{x}'$ from $\mathtt{p}$'s viewpoint and an input $\mathbf{x} = \mathtt{p}?a.\mathbf{x}'$ from $\mathtt{q}$'s viewpoint; otherwise it creates an indirection link from $\mathbf{x}$ to $\mathbf{x}'$. Choice $\mathbf{x} = \mathbf{x}' + \mathbf{x}''$ is projected to the internal choice $\mathbf{x} = \mathbf{x}' \oplus \mathbf{x}''$ if $\mathtt{p}$ is the unique participant deciding on which branch to choose; otherwise the projection gives an external choice $\mathbf{x} = \mathbf{x}' \& \mathbf{x}''$ ([17] gives the definition). Forks, joins and merges are kept identical. As an example, Figure 5 features on the left, in graphical notation, the result of the projection to *A* from the global type $\mathbf{G}$ of Figure 2. Its structure is exactly the same as the original global type, except for the silent transition $\mathbf{x}_9 = \mathbf{x}_{10}$ which is silent from the point of view of *A* and therefore is just elided in the local type.

6.2 Labelled transitions of generalised global and local types

It is possible to define a labelled semantics for global and local types by considering the type (whether local or global) as a state machine specification in which each participant (or the participant, in the case of local type) can evolve, as they would in a CFSMs. As for CFSMs and classical multiparty session types, we keep the syntax of labels ($\ell, \ell', \ldots$).

We use the following notation to keep track of local states (with parallelism, each participant can now execute several transitions concurrently):

$$
\mathtt{X} ::= \mathbf{x}_i \mid \mathtt{X} \mid \mathtt{X} \qquad \mathtt{X}[\_] ::= \ \_ \mid \mathtt{X}[\_] \mid \mathtt{X} \mid \mathtt{X} \mid \mathtt{X}[\_]
$$

LTS for global types We first define, for a global type $\mathbf{G} = \mathsf{def}\ \widetilde{G}\ \mathsf{in}\ \mathbf{x}_0$, a transition system $\mathsf{def}\ \widetilde{G}\ \mathsf{in}\ \tilde{\mathbf{X}},\tilde{w} \xrightarrow{\ell} \mathsf{def}\ \widetilde{G}\ \mathsf{in}\ \tilde{\mathbf{X}}',\tilde{w}'$, where $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{X}}'$ represents a vector recording the state of each of the participants $\tilde{\mathbf{X}} = \{\mathbf{X}_\mathtt{p}\}_{\mathtt{p}\in\mathcal{P}}$ and where $\tilde{w}$ represents the content of the communication buffers $\{w_{\mathtt{q}\mathtt{q}'}\}_{\mathtt{q}\mathtt{q}'\in\mathcal{P}}$. The states for the global type $G = \mathsf{def}\ \widetilde{G}\ \mathsf{in}\ \mathbf{x}_0$ are equipped with an equivalence relation $\equiv_{\widetilde{G}}$, defined in Appendix C.1, which covers associativity, commutativity, forks and joins, choices and merges. Initially, $\tilde{\mathbf{X}}_0 = \{\mathbf{x}_0\}_{\mathtt{p}\in\mathcal{P}}$ and $\tilde{w}_0 = \{\varepsilon\}_{\mathtt{q}\mathtt{q}'\in\mathcal{P}}$. The LTS for global types is defined in Figure 3.

The semantics of global types, as given by the rules $\lfloor$GGR1,2$\rfloor$, follows the intuition of communicating systems: if the global type allows, a participant at the right state can put a value in a communication buffer and progress to the next state ($\lfloor$GGR1$\rfloor$) or, if a value can be read, a participant at the right state can consume it and proceed ($\lfloor$GGR2$\rfloor$). Rule $\lfloor$GGR3$\rfloor$ allows participants that are not concerned by a transition to go there for free. Fork, join, choice and merge transitions are passed through silently by rule $\lfloor$GGR4$\rfloor$.

$$\frac{\mathbf{x}=\mathtt{p}'!a.\mathbf{x}'\in\widetilde{T} \quad \mathbf{T}_\mathtt{p}=\mathsf{def}\ \widetilde{T}\ \mathsf{in}\ \mathtt{X}[\mathbf{x}] \quad w_{\mathtt{pp}'}\in\tilde{w}}{\widetilde{\mathbf{T}},\tilde{w}\xrightarrow{\mathtt{pp}'!a}\widetilde{\mathbf{T}}[\mathbf{T}_\mathtt{p}\leftarrow\mathsf{def}\ \widetilde{T}\ \mathsf{in}\ \mathtt{X}[\mathbf{x}']],\tilde{w}[w_{\mathtt{pp}'}\leftarrow w_{\mathtt{pp}'}\cdot\mathtt{pp}'!a]}\ \lfloor\text{GLR1}\rfloor$$

$$\frac{\mathbf{x}=\mathtt{p}'?a.\mathbf{x}'\in\widetilde{T} \quad \mathbf{T}_{\mathtt{p}'}=\mathsf{def}\ \widetilde{T}\ \mathsf{in}\ \mathbf{X}[\mathbf{x}] \quad w_{\mathtt{pp}'}\in\tilde{w} \quad w_{\mathtt{pp}'}=\mathtt{pp}'!a\cdot w'_{\mathtt{pp}'}}{\widetilde{\mathbf{T}},\tilde{w}\xrightarrow{\mathtt{pp}'?a}\widetilde{\mathbf{T}}[\mathbf{T}_{\mathtt{p}'}\leftarrow\mathsf{def}\ \widetilde{T}\ \mathsf{in}\ \mathbf{X}[\mathbf{x}']],\tilde{w}[w_{\mathtt{pp}'}\leftarrow w'_{\mathtt{pp}'}]}\ \lfloor\text{GLR2}\rfloor$$

$$\frac{\mathbf{T}_\mathtt{p}=\mathsf{def}\ \widetilde{T}\ \mathsf{in}\ \mathbf{X} \quad \mathbf{X}\equiv_{\widetilde{T}}\mathbf{X}' \quad \widetilde{T}[\mathbf{T}_\mathtt{p}\leftarrow\mathsf{def}\ \widetilde{T}\ \mathsf{in}\ \mathbf{X}'],\tilde{w}\xrightarrow{\ell}\widetilde{\mathbf{T}}',\tilde{w}'}{\widetilde{T},\tilde{w}\xrightarrow{\ell}\widetilde{T}',\tilde{w}'}\ \lfloor\text{GLR3}\rfloor$$

Figure 4. Local LTS

LTS for local types We define in Figure 4 a transition system $\widetilde{\mathbf{T}},\tilde{w}\xrightarrow{\ell}\widetilde{\mathbf{T}}',\tilde{w}'$, where $\widetilde{\mathbf{T}}$ represents a set of local types $\{\mathsf{def}\ \widetilde{T}\ \mathsf{in}\ \tilde{\mathbf{X}}_\mathtt{p}\}_{\mathtt{p}\in\mathcal{P}}$ and $\tilde{w}$ represents the content of the communication buffers $\{w_{\mathtt{qq}'}\}_{\mathtt{qq}'\in\mathcal{P}}$. Initially, $\widetilde{\mathbf{T}}_0$ sets all the local types to $\mathbf{x}_0$ and $\tilde{w}_0=\{\varepsilon\}_{\mathtt{qq}'\in\mathcal{P}}$. The principle is strictly identical to the LTS for global types, with, again, an omitted structural equivalence $\equiv_{\widetilde{T}}$ between local states.

Equivalence between generalised local and global types Given the similarity in principle between the global and local LTSs, and considering that the projection algorithm for generalised global types is quasi-homomorphic, we can easily get the trace equivalence between the local and global semantics.

THEOREM 17 (soundness and completeness of projection). *If* $\vec{\mathbf{T}}$ *is the projection of a global type* $\mathbf{G}$ *to all roles, then* $\mathbf{G}\approx(\vec{\mathbf{T}},\varepsilon)$.

6.3 Translations between general local types and CFSMs

Now that we have proved the equivalence from global to local types, we establish the conversion of local types to and from CFSMs.

Translation to CFSMs We first give the already known translation from local types to CFSMs [17]. The illustration of that translation on the Data transfer example is given on the top-right corner of Figure 5.

DEFINITION 16 (translation from local types to MSA [17]). If $\mathbf{T}=\mathsf{def}\ \widetilde{T}\ \mathsf{in}\ \mathbf{x}_0$ is the local type of participant $\mathtt{p}$ projected from $\mathbf{G}$, then the corresponding automaton is $\mathcal{A}(\mathbf{T})=(Q,C,q_0,\Sigma,\delta)$ where:

- Q is defined as the set of well-formed states $\mathtt{X}$ built from the recursion variables $\{\mathbf{x}_i\}$ of $\mathbf{T}$. Q is defined up to the equivalence relation $\equiv_{\widetilde{T}}$ mentioned in § 6.2.
- $C=\{\mathtt{pq}\mid\mathtt{p},\mathtt{q}\in\mathbf{G}\}$
- $q_0=\mathbf{x}_0$
- Σ is the set of $\{a\in\mathbf{G}\}$
- δ is defined by:
  - $(\mathtt{X}[\mathbf{x}],(\mathtt{pp}'!a),\mathtt{X}[\mathbf{x}'])\in\delta$ if $\mathbf{x}=\mathtt{p}'!a.\mathbf{x}'\in\widetilde{T}$.
  - $(\mathtt{X}[\mathbf{x}],(\mathtt{p}'\mathtt{p}?a),\mathtt{X}[\mathbf{x}'])\in\delta$ if $\mathbf{x}=\mathtt{p}'?a.\mathbf{x}'\in\widetilde{T}$.

Translations from CFSMs The converse translation is not as obvious as local types feature explicit forks and joins, while CFSMs only propose choices between interleaved sequences. The translation from a CFSM to a local type therefore comes in 3 steps.

First, we apply a generic translation from minimised CFSMs to Petri nets [15, 29]. This translation relies on the polynomial computation of the graph of regions [1], preserves the trace semantics of the CFSM and, by the minimality of the produced net, makes the concurrency explicit. Figure 5 illustrates on the Data transfer example the shape of the Petri net that can be produced by such a generic translation. Note that the produced Petri net is always safe and free choice.

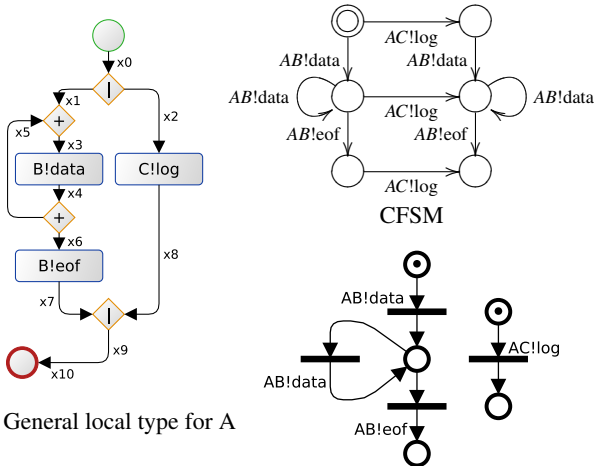


Figure 5. Data transfer example: local translations

The second step of the conversion is to take the Petri net with labelled transitions and enrich it with new silent transitions and new places so that it can be translated into local types. Notably, it should have only one initial marked place, one final place and all labelled transitions should have exactly one incoming and one outgoing arc. Then, we constrain all transitions to be linked with no more than 3 arcs (2 incoming and 1 outgoing for a join transition, or 1 incoming and 2 outgoing for a fork transition, 1 incoming and 1 outgoing for all the other transitions). Places should have no more that 2 incoming and 2 outgoing arcs: if there are two incoming (merge), then the transitions they come from should only have one incoming arc each; if there are 2 outgoing (choice), then the transitions they lead to should have only one outgoing arc each.

In the end, the translation to local type is simple, as each place corresponds to a state variable $\mathbf{x}$, and the different local type transitions can be simply identified. For the lightness of the presentation, instead of defining formally this last step, we describe the converse translation. From it, it is possible to infer the local type generation.

DEFINITION 17 (Petri net representation). Given a local type $\mathbf{T}=\mathsf{def}\ \widetilde{T}\ \mathsf{in}\ \mathbf{x}_0$, we define the Petri net $\mathbb{P}(\mathbf{T})$ by:

- Each state variable $\mathbf{x}\in\widetilde{T}$ is a place in $\mathbb{P}(\mathbf{T})$.
- All the places are initially empty, except for one token in $\mathbf{x}_0$.
- Transitions in $\widetilde{T}$ are translated as follows:
  - If $\mathbf{x}=\mathtt{p}!a.\mathbf{x}'\in\widetilde{T}$ then their is a transition labelled in $\mathbb{P}(\mathbf{T})$, whose unique input arc comes from $\mathbf{x}$ and whose unique output arc goes to $\mathbf{x}'$.
  - If $\mathbf{x}=\mathtt{p}?a.\mathbf{x}'\in\widetilde{T}$ then their is a transition in $\mathbb{P}(\mathbf{T})$, whose unique input arc comes from $\mathbf{x}$ and whose unique output arc goes to $\mathbf{x}'$.
  - If $\mathbf{x}_1=\mathbf{x}_2\mid\mathbf{x}_3\in\widetilde{T}$ then there is a transition in $\mathbb{P}(\mathbf{T})$, whose unique input arc comes from $\mathbf{x}_1$ and whose two outputs arcs go to $\mathbf{x}_2$ and $\mathbf{x}_3$.
  - If $\mathbf{x}_1=\mathbf{x}_2+\mathbf{x}_3\in\widetilde{T}$ (internal or external choice) then there are two transitions in $\mathbb{P}(\mathbf{T})$, that each have an input arc from $\mathbf{x}_1$ and that respectively have an output arc to $\mathbf{x}_2$ and $\mathbf{x}_3$.
  - If $\mathbf{x}_1+\mathbf{x}_2=\mathbf{x}_3\in\widetilde{T}$ then there are two transitions in $\mathbb{P}(\mathbf{T})$, that respectively have an input arc from $\mathbf{x}_1$ and $\mathbf{x}_2$ and that both have an output arc to $\mathbf{x}_3$.

- If $\mathbf{x}_1 \mid \mathbf{x}_2 = \mathbf{x}_3 \in \widetilde{T}$ then there is a transition in $\mathbb{P}(\mathbf{T})$, whose two input arcs respectively come from $\mathbf{x}_1$ and $\mathbf{x}_2$ and whose unique output arc goes to $\mathbf{x}_3$.

The idea of the translation back from a Petri net to a local type will then to identify the transitions and place patterns and convert them into local type transitions.

Note that, in Figure 5, the inferred Petri Net will not give back the local type on the left: in the general case, going through the translation from local type to CFSM and then back to local type will only give an isomorphic local type. The traces are of course preserved.

6.4 Parallelism and local choice condition

This subsection introduces the conditions that CFSMs should respect in order to correspond to well-formed local types projected from generalised global types. It extends the conditions that were sufficient for classical multiparty session types for two reasons. First, we now have concurrent interactions and the no-mixed choice condition does not hold anymore. Second, the well-formedness condition corresponding to projectability in classical multiparty session types needs to take into account the complex control flows of generalised multiparty session types.

We start by a commutativity condition for mixed states in CFSMs: a state is mixed parallel if any send transition satisfies the diamond property with any receive transition. Formally:

DEFINITION 18 (mixed parallel). Let $M = (Q, C, q_0, \mathbb{A}, \delta)$. We say local state q in M is *mixed parallel* if for all $(q, \ell_1, q_1'), (q, \ell_2, q_2') \in \delta$ such that ℓ_1 is a send and ℓ_2 is a receive we have $(q_1', \ell_2, q'), (q_2', \ell_1, q') \in \delta$ for some q'.

Next, we introduce two conditions for the choice that are akin to the local choice conditions with additional data of [20, Def. 2] or the "knowledge of choice" conditions of [11].

DEFINITION 19 (local choice condition). 1. The set of *receivers* of transitions $s_1 \xrightarrow{t_1 \cdots t_m} s_{m+1}$ is defined as $Rcv(t_1 \cdots t_m) = \{\mathtt{q} \mid \exists i \leq m, t_i = (s_i, \mathtt{pq}?a, s_{i+1})\}$.
2. The set of *active senders* are defined as $ASend(t_1 \cdots t_m) = \{\mathtt{p} \mid \exists i \leq m, t_i = (s_i, \mathtt{pq}!a, s_{i+1}) \wedge \forall k < i.\ t_k \neq (s_k, \mathtt{p'p}?b, s_{k+1})\}$ and represent the participants who could immediately send from state s_1.
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 $t_1 \triangleleft t_2$ (t_2 depends on t_1) if either (1) $\Phi(act(t_2)) = act(t_1)$ or (2) $subj(t_1) = subj(t_2)$ unless t_1 and t_2 are mixed parallel.
4. 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 $t_k \triangleleft t_i$.
5. S satisfies the *receiver property* if, for all $s \in RS(S)$ and $s \xrightarrow{t_1} s_1$ and $s \xrightarrow{t_2} s_2$ with $act(t_i) = \mathtt{pq}_i!a_i$, there exist $s_1 \xrightarrow{\varphi_1} s_1'$ and $s_2 \xrightarrow{\varphi_2} s_2'$ such that $Rcv(\varphi_1) = Rcv(\varphi_2)$.
6. S satisfies the *unique sender property* if $s_0 \xrightarrow{\varphi_1} s_1 \xrightarrow{t_1} s_1'$ and $s_0 \xrightarrow{\varphi_2} s_2 \xrightarrow{t_2} s_2'$, with $act(t_1) = \mathtt{p_1p}?a_1$ and $act(t_2) = \mathtt{p_2p}?a_2$ with $a_1 \neq a_2$, $\neg t_1 \triangleleft t_2$ and $\neg t_2 \triangleleft t_1$, then $ASend(\varphi_1' \cdot t_1) = ASend(\varphi_2' \cdot t_2) = \{\mathtt{q}\}$ where $\varphi_i' \subseteq \varphi_i$ and $\varphi_i' \cdot t_i$ is the maximum causal chain.

Together with multiparty compatibility, the receiver property ensures deadlock-freedom while the unique sender property guarantees orphan message-freedom.

PROPOSITION 18 (stability). *Suppose $S = \{M_\mathtt{p}\}_{\mathtt{p} \in \mathcal{P}}$ and each $M_\mathtt{p}$ is deterministic. If (1) S is multiparty compatible; (2) each mixed state in S is mixed parallel; and (3) for any local state that can do two receive transitions, either they commute (satisfy the diamond property) or the state satisfies the unique sender condition, then S is stable and satisfies the reception error freedom and orphan message-freedom properties.*

Proof. The proof is similar to Proposition 11, noting that the unique sender condition guarantees the input availability. See Appendix C. □

THEOREM 19 (deadlock-freedom). *Suppose $S = \{M_\mathtt{p}\}_{\mathtt{p} \in \mathcal{P}}$ satisfies the same conditions as Proposition 18. Assume, in addition, that S satisfies the receiver condition. Then S is deadlock-free.*

Proof. We deduce this theorem from the stability property and the receiver condition. The proof uses a similar reasoning as Proposition 11. □

We call the systems that satisfy the conditions of Theorem 19 *session-compatible*.

By the same algorithm, the multiparty compatibility property is decidable for systems of deterministic CFSMs. It is however undecidable to check the receiver and unique sender properties in general. On the other hand, once multiparty compatibility is assumed, we can restrict the checks to 1-bounded executions (i.e. we limit φ_1, φ_2, φ_1' and φ_2' to 1-bounded executions and $RS_1(S)$ in Definition 19). Then these properties become decidable. Combining the synthesis algorithm defined below, we can decide a subset of CFSMs which can build a general, well-formed global type.

6.5 Synthesis of general multiparty session automata

Now all the pieces are in place for the main results of this paper. We are able to identify the class of communicating systems that correspond to generalised multiparty session types.

The main theorems in this section follow:

THEOREM 20 (synthesis of general systems). *Suppose $S = \{M_\mathtt{p}\}_{\mathtt{p} \in \mathcal{P}}$ is a session-compatible system. Then there is an algorithm which builds G such that $S \approx \mathbf{G}$.*

Proof. The algorithms is the following. We consider $S = \{M_\mathtt{p}\}_{\mathtt{p} \in \mathcal{P}}$ as the definition of a transition system. In this transition system, we only consider the 1-bounded executions. This restriction produces a finite state LTS, where send transitions are immediately followed by the unique corresponding receive transition. In each of these cases, we replace the pair of transitions $\mathtt{pp'}!a$ and $\mathtt{pp'}?a$ by a unique transition $\mathtt{p} \to \mathtt{p'} : a$. To obtain the global type $\mathbf{G}$, we then follow first the standard conversion to Petri nets and the equivalence between Petri nets and global types (similar to the one between Petri nets and local types). We conclude the equivalence by a version of Lemma 14 adapted to session-compatible system. □

Using the synthesis theorem, we are able to provide a full characterisation of generalised multiparty session types in term of session-compatible systems.

THEOREM 21 (soundness and completeness in MSA). *Suppose $S = \{M_\mathtt{p}\}_{\mathtt{p} \in \mathcal{P}}$ is a session compatible system. Then there exits $\mathbf{G}$ such that $S \approx \mathbf{G}$. Conversely, if $\mathbf{G}$ is well-formed as in [17], then there exits S which satisfies the safety and liveness properties (deadlock-freedom, reception error-freedom and orphan message-freedom), and $S \approx \mathbf{G}$.*

Proof. By Theorem 20 and Theorem 17 with the same reasoning as in Theorem 16. □

7. Related work

Our previous work [17] introduced the generalised global and local types and presented a translation from them into CFSMs (Definition 16). 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. This present paper closes this open problem. No synthesis was studied in [17].

There are a large number of paper that can be found in the literature about the synthesis of CFSMs. See [28] for a summary of recent results. 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. We can observe that our synthesis is a special form of these general synthesis problems. The main distinction is, apart from the formal setting (i.e. types), about the kind of the target specifications to be generated (global types and graphs in our case). Not only our synthesis is concerned about trace properties (languages) like the standard synthesis of CFSMs, but we also generate concrete syntax or choreography descriptions as *types* of programs or software. Hence they are directly applicable 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. Recently, [26] proposed a typing system to synthesise global types from a set of local types. A significant limitation in [26, 27] is the expressiveness of the global types they treated. Notably they require concurrent interactions to concern disjoin sets of participants (for example, see the condition in [wf-|]-rule in Figure 1 of [26] which ensures the disjointness of the participants and recursive variables). This class does not cover the generalised class of global types we treat in § 6, for which a synthesis method requires the transformation via Petri nets (or at least a region analysis to infer the parallel branches). 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 under which one can always 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 14 and bridged this gap towards the complete characterisation. In addition, no complexity result of the algorithms is mentioned in [26, 27].

Recent work by [2, 3, 11] focus on proving the semantic correspondence between global and local descriptions. Global types in [11] 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 [2, 3] 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 paper (i.e. the OO-causality or II-causality at different channels [25], since they either only observe send or receive actions). It explains why we used different formulations and proof methods that had not been investigated in [2, 3, 11]. No synthesis algorithm is studied in [2, 3, 11]. See [17] for a more detailed comparison.

There are other type systems which treat multiparty interactions but do not rely on global descriptions. The conversation calculus [10] models the interactions between a client and various services, with dynamic joining into conversations, for a possibly unknown number of processes. More flexible checking is performed in the framework of contracts [12] which prescribe the abstract interaction behaviours of processes. In contracts, typable processes themselves may not always satisfy the properties of session types such as progress: it is proved later by checking whether a whole contract conforms to a certain form. Our work takes advantages from both the local approaches in [10, 12] and global approaches in [25]. None of [10, 12] studies the inference or synthesis of global descriptions. For further comparisons of session types with other service-oriented calculi and behavioural typing systems and a wide-range survey of the related literature, see [18].

8. Conclusion and future work

This paper investigated the sound and complete characterisation of three representative classes of multiparty session types, called *sequential*, *classical* and *generalised* global types, into CFSMs and developed their synthesis algorithms from CFSMs. We summarise the equivalences in Table 1. All of the three 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 5).

Multiparty Session Type	Communicating system (CS)
Sequential MPST	$\approx$ Sequential, orphan message-free and deadlock-free, CS
Classical MPST	$\approx$ Basic, multiparty compatible CS
Generalised MPST	$\approx$ Session-compatible CS

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 each class, the multiparty compatibility condition is decidable and uniformly applied to identify a set of "well-behaved" CFSMs. In sequential systems, multiparty compatibility is *omnipresent* as the execution in the sequential systems is automatically compatible. In classical systems, the multiparty compatibility property is a *necessary and sufficient condition* to obtain safe global types. In the third class, we require additional conditions for choice and parallel compositions, which are decidable under 1-bounded executions. Our completeness results rely on Lemma 14 which implies that building a global type which simulates only the 1-bounded traces is sufficient to also simulate all other traces.

The methods proposed here are palatable to a wide range of applications based on choreography protocol models including graphical notations such as BPMN 2.0, and more widely, finite state machines. We are currently working on two applications based on the theory developed in this paper. The Testable Architecture (TA) project [33], developed as an open source project by Red Hat and Cognizant, starts from the description of application scenarios, which are described as BPMN 2.0 choreographies (a superset of general global types) of interactions among components. A usecase written as a choreography is projected into a set of local models which will then be elaborated with implementation details. The TA enables the communication structure of the implementation to be inferred, and to be tested against the choreography. Updating global scenarios against local models plays an important role in different stages of SLC (software life cycle) in this architecture. Another application is the use of multiparty session types for dynamic monitoring for a large scale cyberinfrastructure over the US and beyond [31]. We use local CFSMs projected from Scribble protocols [34] (which is a industrial language to describe multiparty session types) for network monitoring. A central controller can check that distributed update paths for monitor specifications (i.e. local CFSM) which are geographically far apart are safe by synthesis.

Variants of our method would be also usable for distributed dynamic software update (e.g. extending [23] to distribution): for example, we could examine the actual programs that implement both versions and construct the combined session type and check that we can rebuild the global type. Extending our method to global types

with logical predicates [6], we can specify the desired properties locally, and then derive the combined global type that satisfies these properties. From this type, we could modify the attendant programs to implement the type, therefore aggregating and distributing all parties logical properties via synthesis.

References

[1] E. Badouel and P. Darondeau. Theory of regions. *Lectures on Petri Nets I: Basic Models*, pages 529–586, 1998.

[2] S. Basu, T. Bultan, and M. Ouederni. Deciding choreography realizability. In *POPL'12*, pages 191–202. ACM, 2012.

[3] 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.

[4] L. Bettini et al. Global progress in dynamically interleaved multiparty sessions. In *CONCUR*, volume 5201 of *LNCS*, pages 418–433, 2008.

[5] 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.

[6] L. Bocchi, K. Honda, E. Tuosto, and N. Yoshida. A theory of design-by-contract for distributed multiparty interactions. In *CONCUR'10*, volume 6269 of *LNCS*, pages 162–176. Springer, 2010.

[7] Business Process Model and Notation. `http://www.bpmn.org`.

[8] D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30:323–342, April 1983.

[9] L. Caires and F. Pfenning. Session types as intuitionistic linear propositions. In *CONCUR*, volume 6269 of *LNCS*, pages 222–236. Springer, 2010.

[10] L. Caires and H. T. Vieira. Conversation types. In *ESOP*, volume 5502 of *LNCS*, pages 285–300. Springer, 2009.

[11] G. Castagna, M. Dezani-Ciancaglini, and L. Padovani. On global types and multi-party session. *LMCS*, 8(1), 2012.

[12] G. Castagna and L. Padovani. Contracts for mobile processes. In *CONCUR 2009*, number 5710 in LNCS, pages 211–228, 2009.

[13] G. Cécé and A. Finkel. Verification of programs with half-duplex communication. *Inf. Comput.*, 202(2):166–190, 2005.

[14] 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.

[15] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving petri nets from finite transition systems. *Computers, IEEE Transactions on Computers*, 47(8):859–882, 1998.

[16] 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`.

[17] P.-M. Deniélou and N. Yoshida. Multiparty session types meet communicating automata. In *ESOP*, volume 7211 of *LNCS*, pages 194–213. Springer, 2012.

[18] M. Dezani-Ciancaglini and U. de' Liguoro. Sessions and Session Types: an Overview. In *WS-FM'09*, volume 6194 of *LNCS*, pages 1–28. Springer, 2010.

[19] 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.

[20] B. Genest, A. Muscholl, and D. Peled. Message sequence charts. In *Lectures on Concurrency and Petri Nets*, volume 3098 of *LNCS*, pages 537–558, 2004.

[21] J.-Y. Girard. Linear logic. *TCS*, 50, 1987.

[22] M. Gouda, E. Manning, and Y. Yu. On the progress of communication between two finite state machines. *Information and Control.*, 63:200–216, 1984.

[23] C. M. Hayden, E. K. Smith, M. Denchev, M. Hicks, and J. S. Foster. Kitsune: Efficient, general-purpose dynamic software updating for C. In *OOPSLA*, 2012.

[24] 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.

[25] K. Honda, N. Yoshida, and M. Carbone. Multiparty Asynchronous Session Types. In *POPL'08*, pages 273–284. ACM, 2008.

[26] J. Lange and E. Tuosto. Synthesising choreographies from local session types. In *CONCUR'12*, LNCS. Springer, 2012. To appear.

[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] M. Nielsen, G. Rozenberg, and P. Thiagarajan. Elementary transition systems. In *Theoretical Computer Science*, volume 96, pages 3–33. Elsevier Science Publishers Ltd., 1992.

[30] Online Appendix. `http://www.doc.ic.ac.uk/~yoshida/cfsm/`.

[31] Ocean Observatories Initiative (OOI). `http://www.oceanobservatories.org/`.

[32] J. Planul, R. Corin, and C. Fournet. Secure enforcement for global process specifications. *CONCUR 2009-Concurrency Theory*, pages 511–526, 2009.

[33] Savara JBoss Project. `http://www.jboss.org/savara`.

[34] Scribble JBoss Project. `http://www.jboss.org/scribble`.

[35] 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.

[36] J. Villard. *Heaps and Hops*. PhD thesis, ENS Cachan, 2011.

[37] P. Wadler. Proposition as Sessions. In *ICFP'12*, 2012.

[38] N. Yoshida, P.-M. Deniélou, A. Bejleri, and R. Hu. Parameterised multiparty session types. In *FoSSaCs*, volume 6014 of *LNCS*, pages 128–145, 2010.

A. Appendix: Section 3

Proof of The size of a global type G, denoted by $size(G)$, is defined as follows:

$$\begin{array}{lcl} size(\mathtt{p} \to \mathtt{p}' \colon \{k.G_j\}_{j\in J}) & = & 1+\sum_{j\in J}(1+size(G_j)) \\ size(\mu \mathtt{t}.G') & = & size(G')+1 \\ size(\mathtt{t}) = size(\mathsf{end}) & = & 1 \end{array}$$

The third line of the projection definition checks that each T_i ($i \in I$) is mergeable or not with T_j ($j \neq i \in I$). Since the size of the global types is the number of the branching of G, to check $G \upharpoonright \mathtt{p}$ for each $\mathtt{p}$ is bound in polynomial.

Proposition 3 Obvious by definition.

Proposition 4 By the construction.

A.1 Proofs of Theorem 2

Soundness By induction on $G \xrightarrow{\ell} G'$:

GR1 where $G = \mathtt{p} \to \mathtt{p}' \colon \{a_i.G_i\}_{i\in I} \xrightarrow{\mathtt{pp}'!a_j} G' = \mathtt{p} \rightsquigarrow \mathtt{p}' \colon a_j.G_j$. 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: $T_\mathtt{p} = G \upharpoonright \mathtt{p} = \mathtt{p}'!\{a_i.G_i \upharpoonright \mathtt{p}\}_{i\in I}$ and $T_{\mathtt{p}'} = G \upharpoonright \mathtt{p}' = \mathtt{p}?\{a_i.G_i \upharpoonright \mathtt{p}'\}_{i\in I}$ and (for $\mathtt{q} \notin \{\mathtt{p},\mathtt{p}'\}$) $T_\mathtt{q} = \sqcup_{i\in I} G_j \upharpoonright \mathtt{q}$. 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 $w'_{\mathtt{pp}'} = w_{\mathtt{pp}'} \cdot a_j$. By rule LR2, we can have the silent $\mathtt{p}?\{a_i.G_i \upharpoonright \mathtt{p}'\}_{i\in I} \to \mathtt{p}?a_j.(Gj \upharpoonright \mathtt{p}')$, which as a configuration transition is $\{T'_\mathtt{q}\}_{\mathtt{q}\in\mathcal{P}}, \{w'_{\mathtt{qq}'}\}_{\mathtt{qq}'\in\mathcal{P}} \to \{T''_\mathtt{q}\}_{\mathtt{q}\in\mathcal{P}}, \{w'_{\mathtt{qq}'}\}_{\mathtt{qq}'\in\mathcal{P}}$ with $T''_\mathtt{q} = T'_\mathtt{q}$ if $\mathtt{q} \neq \mathtt{p}'$, and $T''_{\mathtt{p}'} = \mathtt{p}?a_j.(Gj \upharpoonright \mathtt{p}')$. This corresponds exactly to the projection $[\![G']\!]$ of G'.

GR2 where $G = \mathtt{p} \rightsquigarrow \mathtt{p}' \colon a_j.G_j \xrightarrow{\mathtt{pp}'?a_j} G' = G_j$. 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 LR3, $\{T_\mathtt{q}\}_{\mathtt{q}\in\mathcal{P}}, \{w_{\mathtt{qq}'}\}_{\mathtt{qq}'\in\mathcal{P}} \xrightarrow{\mathtt{pp}'?a_j} \{G_j \upharpoonright \mathtt{q}\}_{\mathtt{q}\in\mathcal{P}}, \{w'_{\mathtt{qq}'}\}_{\mathtt{qq}'\in\mathcal{P}}$ 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 $G = \mu \mathtt{t}.G'$. Projection is homomorphic with respect to recursion. We can use induction and LR4 to conclude.

GR4 where $\mathtt{p} \to \mathtt{p}' : \{a_i.G_i\}_{i\in I} \xrightarrow{\ell} \mathtt{p} \to \mathtt{p}' : \{a_i.G'_i\}_{i\in I}$. This is direct by inductive hypothesis.

GR5 Direct by inductive hypothesis.

Completeness By induction on
$\{T_\mathtt{p}\}_{\mathtt{p}\in\mathcal{P}}, \{w_{\mathtt{qq}'}\}_{\mathtt{qq}'\in\mathcal{P}} \xrightarrow{\ell} \{T'_\mathtt{p}\}_{\mathtt{p}\in\mathcal{P}}, \{w'_{\mathtt{qq}'}\}_{\mathtt{qq}'\in\mathcal{P}}$
such that there is a G and $\{T_\mathtt{p}\}_{\mathtt{p}\in\mathcal{P}}, \{w_{\mathtt{qq}'}\}_{\mathtt{qq}'\in\mathcal{P}} = [\![G]\!]$:

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 $\mathtt{p} \to \mathtt{q} : \{a_i.G_i\}_{i\in I}$ 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 $\mathtt{p} \to \mathtt{q} : \{a_i.G_i\}_{i\in I}$. Therefore we can apply as many times as needed GR4 and GR5, and use GR1 to reduce to $\mathtt{p} \rightsquigarrow \mathtt{q} : a_j.G_j$. The projection of the resulting global type corresponds to the result of LR3.

LR2 It can be completed by [LR3].

LR3 There is $T_\mathtt{p} = G \upharpoonright \mathtt{p} = \mathtt{q}?a_j.T_j$. To activate LR3, there should be a value a_j in the buffer $w_{\mathtt{pq}}$. By definition of projection, G has therefore $\mathtt{p} \rightsquigarrow \mathtt{q} : a_j.G_j$ 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 $\mathtt{p} \rightsquigarrow \mathtt{q} : a_j.G_j$. 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 LR3.

LR4 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. Appendix for Section 5

B.1 Proofs of Proposition 11

Stable Property 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, 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, $act(t_4) = \mathtt{p}'\mathtt{q}'!c$ and $\mathtt{p}' \neq \mathtt{p}$ and $\mathtt{q}' \neq \mathtt{q}$. Hence by Lemma 10(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 $\overline{t_1} = \mathtt{p}'\mathtt{q}'?c$. Then this time, by Lemma 10(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'. □

Deadlock-freedom Assume that S satisfies the above conditions. We show by contradiction that no configuration s in $RS(S)$ ($RS(S)$ is the set of all reachable states of S) is a deadlock. Suppose q_r is a deadlock state in s. Since q is in $RS(S)$, there exists $q_0, ..., q_r$ such that q_0 is the initial state and q_i follows q_{i-1} in $M_\mathtt{p}$ for some $\mathtt{p}$. The set of states corresponds to the direct path $p_\mathtt{p}$ which starts from q_0 in $M_\mathtt{p}$. Similarly we set p as the direct path $q'_0 \cdots q'_r$ from the view of $\mathtt{p}$ in $S^{-\mathtt{p}}$ (we do not have to consider the alternations φ_i from the direct path in $S^{-\mathtt{p}}$ since it does not affect to the interaction with $\mathtt{p}$ by Lemma 10). Since s is the deadlock state, $|p_\mathtt{p}| = |p|$. Then there are two cases to consider.

Case (a) $p_\mathtt{p}$ and p are compatible. In this case, if the path $p_\mathtt{p}$ is extended into $p'_\mathtt{p}$ in $M_\mathtt{p}$, then no directed path p' in $S^{-\mathtt{p}}$. This contradicts the assumption that S is compatible.

Case (b) $p_\mathtt{p}$ and p are not compatible. By the assumption, there exists a path p' in $S^{-\mathtt{p}}$ such that $p_\mathtt{p}$ and p' are compatible. Clearly $|p_\mathtt{p}| = |p| = |p'|$ and p and p' are not identical. Let q and q' the first different states in p and p'. Then q and q' are in the same machine $M_\mathtt{q}$ and q and q' have the same previous sate q_h in $M_\mathtt{q}$. Then $q_h \xrightarrow{\ell} q$ and $q_h \xrightarrow{\ell'} q'$ in $M_\mathtt{q}$. Since $M_\mathtt{q}$ has no mixed states, ℓ and ℓ' are both sending actions or receiving actions which are compatible with the action in $M_\mathtt{p}$. Since $M_\mathtt{q}$ is deterministic, if $\ell = \ell'$, then $q = q'$ which contradicts the assumption such that $q \neq q'$.

Other safety properties are similarly proved. □

B.2 Proof for Lemma 14

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 a 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 hold for traces of length k or less.

We assume the last $k+1$th action is 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 $\varphi_0 \ell_0 \varphi_1 \ell$. In S_1, we have

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

On the other hand, by the multiparty compatibility,

$$S_1 : s_0 \xrightarrow{\varphi_0} \xrightarrow{\ell_0} s_1 \xrightarrow{\varphi'} \xrightarrow{\overline{\ell_0}} s'_1 \tag{B.3}$$

where $\xrightarrow{\varphi'}$ is an alternation with $\mathtt{p} \notin subj(act(\varphi'))$ and $\overline{\ell_0} = \mathtt{pq}?a$. Hence φ' does not gain the buffer size from s_1. Let us set $\varphi_1 = \varphi'_1 \cdot \varphi''_1$ and $\varphi' = \varphi'_1 \cdot \varphi''$. Note that $\mathtt{pq}?a \notin act(\varphi_1)$ since ℓ_0 is an unmatched sending. Then we can set $\varphi''_1 \cap \varphi'' = \varepsilon$. Hence we have:

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

By permutation, we have:

$$S_1 : s_0 \xrightarrow{\varphi_0} \xrightarrow{\ell_0} s_1 \xrightarrow{\varphi_1} \xrightarrow{\varphi''} \xrightarrow{\overline{\ell_0}} \xrightarrow{\ell} s' \tag{B.5}$$

Note that since ℓ_0 is now consumed by $\overline{\ell_0}$, the size of the buffer of this execution is n.

By the inductive hypothesis, in S_2, we have:

$$S_2 :\ s'_0 \xrightarrow{\varphi_0} \xrightarrow{\ell_0} s_1 \xrightarrow{\varphi_1} \xrightarrow{\varphi''} \xrightarrow{\overline{\ell_0}} \xrightarrow{\ell} s_2 \tag{B.6}$$

Because $\varphi''_1 \cap \varphi'' = \varepsilon$, we have:

$$S_2 :\ s'_0 \xrightarrow{\varphi_0} \xrightarrow{\ell_0} s'_1 \xrightarrow{\varphi_1} \xrightarrow{\ell} \xrightarrow{\varphi''} \xrightarrow{\overline{\ell_0}} s'_2 \tag{B.7}$$

Hence S_2 can simulate S_1's trace in (B.2) whose buffer-bound is $n+1$.

C. Appendix for Section 6

Projection We define the projection from a global type to a local type where *ASend* means that a set of active senders, which corresponds to the same definition in CFSMs (see [17]).

$$\begin{array}{rcll}
\mathsf{def}\ \widetilde{G}\ \mathsf{in}\ \mathbf{x} \upharpoonright \mathtt{p} & = & \mathsf{def}\ \widetilde{G} \upharpoonright_{\widetilde{G}} \mathtt{p}\ \mathsf{in}\ \mathbf{x} & \\
\mathbf{x} = \mathtt{p} \to \mathtt{p}' : a\,; \mathbf{x}' \upharpoonright_{\widetilde{G}} \mathtt{p} & = & \mathbf{x} = \mathtt{p}'!a.\mathbf{x}' & \\
\mathbf{x} = \mathtt{p} \to \mathtt{p}' : a\,; \mathbf{x}' \upharpoonright_{\widetilde{G}} \mathtt{p}' & = & \mathbf{x} = \mathtt{p}?a.\mathbf{x}' & \\
\mathbf{x} = \mathtt{p} \to \mathtt{p}' : a\,; \mathbf{x}' \upharpoonright_{\widetilde{G}} \mathtt{p}'' & = & \mathbf{x} = \mathbf{x}'\ (\mathtt{p} \notin \{\mathtt{p}, \mathtt{p}'\}) & \\
\mathbf{x} \mid \mathbf{x}' = \mathbf{x}'' \upharpoonright_{\widetilde{G}} \mathtt{p} & = & \mathbf{x} \mid \mathbf{x}' = \mathbf{x}'' & \\
\mathbf{x} = \mathbf{x}' \mid \mathbf{x}'' \upharpoonright_{\widetilde{G}} \mathtt{p} & = & \mathbf{x} = \mathbf{x}' \mid \mathbf{x}'' & \\
\mathbf{x} = \mathbf{x}' + \mathbf{x}'' \upharpoonright_{\widetilde{G}} \mathtt{p} & = & \mathbf{x} = \mathbf{x}' \oplus \mathbf{x}'' & (\text{if } \mathtt{p} = \mathit{ASend}(\widetilde{G})(\mathbf{x})) \\
\mathbf{x} = \mathbf{x}' + \mathbf{x}'' \upharpoonright_{\widetilde{G}} \mathtt{p} & = & \mathbf{x} = \mathbf{x}' \,\&\, \mathbf{x}'' & (\text{otherwise}) \\
\mathbf{x} + \mathbf{x}' = \mathbf{x}'' \upharpoonright_{\widetilde{G}} \mathtt{p} & = & \mathbf{x} + \mathbf{x}' = \mathbf{x}'' & \\
\mathbf{x} = \mathsf{end} \upharpoonright_{\widetilde{G}} \mathtt{p} & = & \mathbf{x} = \mathsf{end} &
\end{array}$$

C.1 Global type equivalence

Below we define the equivalence relation $\equiv_{\widetilde{G}}$ used in the LTS of the global types.

$$\mathtt{X} \mid \mathtt{X}' \equiv_{\widetilde{G}} \mathtt{X}' \mid \mathtt{X} \qquad \mathtt{X} \mid (\mathtt{X}' \mid \mathtt{X}'') \equiv_{\widetilde{G}} (\mathtt{X} \mid \mathtt{X}') \mid \mathtt{X}''$$

$$\frac{\mathbf{x} = \mathbf{x}' \in \widetilde{G}}{\mathtt{X}[\mathbf{x}] \equiv_{\widetilde{G}} \mathtt{X}[\mathbf{x}']} \qquad \frac{\mathbf{x} = \mathbf{x}' \mid \mathbf{x}'' \in \widetilde{G}}{\mathtt{X}[\mathbf{x}] \equiv_{\widetilde{G}} \mathtt{X}[\mathbf{x}' \mid \mathbf{x}'']} \qquad \frac{\mathbf{x} \mid \mathbf{x}' = \mathbf{x}'' \in \widetilde{G}}{\mathtt{X}[\mathbf{x} \mid \mathbf{x}'] \equiv_{\widetilde{G}} \mathtt{X}[\mathbf{x}'']}$$

$$\frac{\mathbf{x} = \mathbf{x}' + \mathbf{x}'' \in \widetilde{G}}{\mathtt{X}[\mathbf{x}] \equiv_{\widetilde{G}} \mathtt{X}[\mathbf{x}']} \qquad \frac{\mathbf{x} = \mathbf{x}' + \mathbf{x}'' \in \widetilde{G}}{\mathtt{X}[\mathbf{x}] \equiv_{\widetilde{G}} \mathtt{X}[\mathbf{x}'']} \qquad \frac{\mathbf{x} + \mathbf{x}' = \mathbf{x}'' \in \widetilde{G}}{\mathtt{X}[\mathbf{x}] \equiv_{\widetilde{G}} \mathtt{X}[\mathbf{x}'']} \qquad \frac{\mathbf{x} + \mathbf{x}' = \mathbf{x}'' \in \widetilde{G}}{\mathtt{X}[\mathbf{x}'] \equiv_{\widetilde{G}} \mathtt{X}[\mathbf{x}'']}$$

Below we define the equivalence relation $\equiv_{\widetilde{T}}$ used in the translation in Definition 16.

$$\mathtt{X} \mid \mathtt{X}' \equiv_{\widetilde{T}} \mathtt{X}' \mid \mathtt{X} \qquad \mathtt{X} \mid (\mathtt{X}' \mid \mathtt{X}'') \equiv_{\widetilde{T}} (\mathtt{X} \mid \mathtt{X}') \mid \mathtt{X}''$$

$$\frac{\mathbf{x} = \mathbf{x}' \in \widetilde{T}}{\mathtt{X}[\mathbf{x}] \equiv_{\widetilde{T}} \mathtt{X}[\mathbf{x}']} \qquad \frac{\mathbf{x} = \mathbf{x}' \mid \mathbf{x}'' \in \widetilde{T}}{\mathtt{X}[\mathbf{x}] \equiv_{\widetilde{T}} \mathtt{X}[\mathbf{x}' \mid \mathbf{x}'']} \qquad \frac{\mathbf{x} \mid \mathbf{x}' = \mathbf{x}'' \in \widetilde{T}}{\mathtt{X}[\mathbf{x} \mid \mathbf{x}'] \equiv_{\widetilde{T}} \mathtt{X}[\mathbf{x}'']}$$

$$\frac{\mathbf{x} = \mathbf{x}' \,\&\, \mathbf{x}'' \in \widetilde{T}}{\mathtt{X}[\mathbf{x}] \equiv_{\widetilde{T}} \mathtt{X}[\mathbf{x}']} \qquad \frac{\mathbf{x} = \mathbf{x}' \,\&\, \mathbf{x}'' \in \widetilde{T}}{\mathtt{X}[\mathbf{x}] \equiv_{\widetilde{T}} \mathtt{X}[\mathbf{x}'']} \qquad \frac{\mathbf{x} = \mathbf{x}' \oplus \mathbf{x}'' \in \widetilde{T}}{\mathtt{X}[\mathbf{x}] \equiv_{\widetilde{T}} \mathtt{X}[\mathbf{x}']} \qquad \frac{\mathbf{x} = \mathbf{x}' \oplus \mathbf{x}'' \in \widetilde{T}}{\mathtt{X}[\mathbf{x}] \equiv_{\widetilde{T}} \mathtt{X}[\mathbf{x}'']}$$

$$\frac{\mathbf{x} + \mathbf{x}' = \mathbf{x}'' \in \widetilde{T}}{\mathtt{X}[\mathbf{x}] \equiv_{\widetilde{T}} \mathtt{X}[\mathbf{x}'']} \qquad \frac{\mathbf{x} + \mathbf{x}' = \mathbf{x}'' \in \widetilde{T}}{\mathtt{X}[\mathbf{x}'] \equiv_{\widetilde{T}} \mathtt{X}[\mathbf{x}'']}$$

C.2 Proof of Proposition 18

Essentially we have the same as the proof of Proposition 11. Only difference is that we need to use the unique sender condition to ensure that the action $\overline{t_1}$ is possible in (B.1) in the proof of Proposition 11 (note that $\overline{t_1}$ is always possible in basic CFSMs since they are directed).

Suppose, in (B.1) in the proof of Proposition 11, the action $\overline{t_1}$ is not possible: i.e. $s_1 \xrightarrow{t_1} s_2 \xrightarrow{\varphi_2} s'_2$ but s'_2 cannot perform $\xrightarrow{\overline{t_1}}$. The only possibility is that some $M_{\mathtt{q}}$ contains the receiver state q such that $(q, \mathtt{pq}?a, q'), (q, \mathtt{p}'\mathtt{q}?b, q'') \in \delta_{\mathtt{q}}$ which does not satisfy the parallel condition (since if so, s'_2 can perform $\xrightarrow{\overline{t_1}}$), and φ_2 contains the action $\mathtt{p}'\mathtt{q}?b$, which implies φ_2 contains the action $\mathtt{p}'\mathtt{q}!b$. By the unique sender condition, there is the unique $\mathtt{q}'$ such that $s'_0 \xrightarrow{\varphi \cdot \mathtt{pq}!a} s_1$ and $s'_0 \xrightarrow{\varphi \cdot \mathtt{pq}!a \cdot \varphi' \cdot \mathtt{p}'\mathtt{q}!b \cdot \varphi''} s'_2$ with $\mathit{ASend}(\varphi \cdot \mathtt{pq}!a) = \mathit{ASend}(\varphi \cdot \mathtt{pq}!a \cdot \varphi' \cdot \mathtt{p}'\mathtt{q}!b) = \{\mathtt{q}'\}$. Since $\mathtt{p}'\mathtt{q}!b$ cannot be reordered before $\mathtt{pq}!a$ or after φ_1, to satisfy the unique sender property, φ' should include $\mathtt{pq}?a$. This contradicts that the assumption that φ_2 does not include $\mathtt{pq}?a$.

C.3 Proof of Theorem 19

By (reception error freedom) and (orphan message-freedom), together with (stable-property), we only have to check, there is no input is waiting with an empty queue forever. Suppose by contradiction, there is $s \in RS(S)$ such that $s = (\vec{q}; \vec{\varepsilon})$ and there exists input state $q_{\mathtt{p}} \in \vec{q}$ and no output transition from q_k such that $k \neq \mathtt{q}$.

Then by assumption, there is a 1-buffer execution φ and since φ is not taken (if so, $q_{\mathtt{p}}$ can perform an input), then there is another execution φ' such that it leads to state s which is deadlock at $q_{\mathtt{p}}$.

Case (1) Suppose φ does not include input actions at $\mathtt{q}$ except a, i.e. a is the first input action at $\mathtt{q}$ in φ. We let φ_0 for the prefix before the actions of $\mathtt{qp}!a \cdot \mathtt{qp}?a$.

By (receiver condition), we know $\mathtt{p} \in \mathit{Rcv}(\varphi')$.

By the determinacy, the corresponding input action has a different label from a, i.e. $\mathtt{q}'\mathtt{p}?a' \in \varphi'$. By the diamond property, $\mathtt{q}'\mathtt{p}?a'$ and $\mathtt{qp}?a$ can be appeared from the same state, i.e. this state is under the assumption of the parallel condition. Hence by the multiparty compatibility, the both corresponding outputs $\mathtt{q}'\mathtt{p}!a'$ and $\mathtt{qp}!a$ can be always fired if one of them is. This contradicts the assumption that $q_{\mathtt{p}}$ is deadlock with label a.

Case (2) Suppose φ includes other input actions at $\mathtt{q}$ before $\mathtt{qp}?a$, i.e. $\mathtt{p} \in \mathit{Rcv}(\varphi_0)$. Let $\mathtt{q}'\mathtt{p}?a'$ the action which first occurs in φ_0. By $\mathtt{p} \in \mathit{Rcv}(\varphi')$, there exists $\mathtt{q}''\mathtt{p}?a'' \in \varphi'$. If $\mathtt{q}''\mathtt{p}?a'' \neq \mathtt{q}'\mathtt{p}?a'$, by the same reasoning as (1), the both corresponding outputs are available. Hence we assume the case $\mathtt{q}''\mathtt{p}?a'' = \mathtt{q}'\mathtt{p}?a'$. Let s is the first state from which a transition in φ_0 and a transition in φ' are separated. Then by assumption, if $s \xrightarrow{\varphi_0 \cdot \mathtt{q}'\mathtt{p}!a' \cdot \mathtt{q}'\mathtt{p}?a'} s_1$ and $s \xrightarrow{\varphi_1 \cdot \mathtt{q}'\mathtt{p}!a' \cdot \mathtt{q}'\mathtt{p}?a'} s_2$, by assumption $a' \notin \varphi_0 \cup \varphi_1$, hence $s \xrightarrow{\mathtt{q}'\mathtt{p}!a' \cdot \mathtt{q}'\mathtt{p}?a'} s'_1 \xrightarrow{\varphi'_0} s_1$ and $s \xrightarrow{\mathtt{q}'\mathtt{p}!a' \cdot \mathtt{q}'\mathtt{p}?a'} s'_2 \xrightarrow{\varphi'_1} s_2$ by the diamond property again. Since s_1 can perform an input at $\mathtt{q}$ by the assumption (because of $\mathtt{qp}?a$), φ'_1 should contain an input at $\mathtt{q}$ by the receiver condition. If it contains the input to $\mathtt{q}$ in φ'_1, then we repeat Case (2) noting that the length of φ'_1 is shorter than the length of $\varphi_1 \cdot \mathtt{q}'\mathtt{p}!a' \cdot \mathtt{q}'\mathtt{p}?a'$; else we use Case (1) to lead the contradiction; otherwise if it contains the same input as $\mathtt{qp}?a$, then it contradicts the assumption that $q_{\mathtt{p}}$ is deadlock.