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

Pierre-Malo Deniélou and Nobuko Yoshida

[1] Royal Holloway, University of London
[2] Imperial College London

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

1 Introduction

Over the last decade, *session types* [12,18] have been studied as data types or functional types for communications and distributed systems. A recent discovery by [4,20], which establishes a Curry-Howard isomorphism between binary session types and linear logics, confirms that session types and the notion of duality between type constructs have canonical meanings. Multiparty session types [2,13] were proposed as a major generalisation of binary session types. They can enforce communication safety and deadlock-freedom for more than two peers thanks to a choreographic specification (called *global type*) of the interaction. Global types are projected to end-point types (*local types*), against which processes can be statically type-checked and verified to behave correctly.

The motivation of this paper comes from our practical experiences that, in many situations, even where we start from the end-point projections of a choreography, we need to reconstruct a global type from distributed specifications. End-point specifications are usually available, either through inference from the control flow, or through existing service interfaces, and always in forms akin to individual communicating finite state machines. If one knows the precise conditions under which a global type can be constructed (i.e. the conditions of *synthesis*), not only the global safety property which multiparty session types ensure is guaranteed, but also the generated global type can be used as a refinement and be integrated within the distributed system development life-cycle (see [17]). This paper attempts to give the synthesis condition as a sound and complete characterisation of multiparty session types with respect to Communicating Finite State Machines (CFSMs) [3]. CFSMs have been a well-studied formalism for analysing distributed safety properties and are widely present in industry tools.

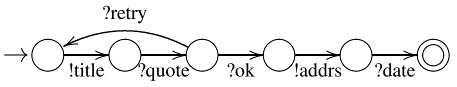
They can been seen as generalised end-point specifications, therefore an excellent target for a common comparison ground and for synthesis. As explained below, to identify a complete set of CFSMs for synthesis, we first need to answer a question – *what is the canonical duality notion in multiparty session types?*

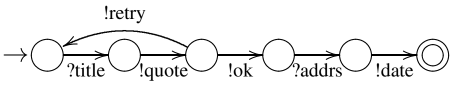
Characterisation of Binary Session Types as Communicating Automata. The subclass which fully characterises *binary session types* was actually proposed by Gouda, Manning and Yu in 1984 [11] in a pure communicating automata context. Consider a simple business protocol between a Buyer and a Seller from the Buyer's viewpoint: Buyer sends the title of a book, Seller answers with a quote. If Buyer is satisfied by the quote, then he sends his address and Seller sends back the delivery date; otherwise it retries the same conversation. This can be described by the following session type:

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

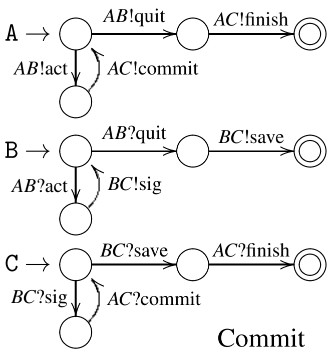
where the operator $!\mathsf{title}$ denotes an output of the title, whereas $?\mathsf{quote}$ denotes an input of a quote. The output choice features the two options ok and retry and ; denotes sequencing. end represents the termination of the session, and $\mu\mathsf{t}$ is recursion.

The simplicity and tractability of binary sessions come from the notion of *duality* in interactions [10]. The interaction pattern of the Seller is fully given as the dual of the type in (1.1) (exchanging input ! and output ? in the original type). When composing two parties, we only have to check they have mutually dual types, and the resulting communication is guaranteed to be deadlock-free. Essentially the same characterisation is given in communicating automata. Buyer and Seller's session types are represented by the following two machines.





We can observe that these CFSMs satisfy three conditions. First, the communications are *deterministic*: messages that are part of the same choice, ok and retry here, are distinct. Secondly, there is no mixed state (each state has either only sending actions or only receiving actions). Third, these two machines have *compatible* traces (i.e. dual): the Seller machine can be defined by exchanging sending to receiving actions and vice versa. Breaking one of these conditions allows deadlock situations and breaking one of the first two conditions makes the compatibility checking undecidable [11, 19].



Commit

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

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

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

Contributions and Outline. Section 3 defines new labelled transition systems for global and local types that represent the abstract observable behaviour of typed processes. We prove that a global type behaves exactly as its projected local types, and the same result between a single local type and its CFSMs interpretation. These correspondences are the key to prove the main theorems. Section 4 defines *multiparty compatibility*, studies its safety and liveness properties, gives an algorithm for the synthesis of global types from CFSMs, and proves the soundness and completeness results between global types and CFSMs. Section 5 discusses related work and concludes. The full proofs and applications of this work can be found in [17].

2 Communicating Finite State Machines

This section starts from some preliminary notations (following [6]). ε is the empty word. $\mathbb{A}$ is a finite alphabet and $\mathbb{A}^*$ is the set of all finite words over $\mathbb{A}$. $|x|$ is the length of a word x and $x.y$ or xy the concatenation of two words x and y. Let $\mathcal{P}$ be a set of *participants* fixed throughout the paper: $\mathcal{P} \subseteq \{\mathtt{A},\mathtt{B},\mathtt{C},\dots,\mathtt{p},\mathtt{q},\dots\}$.

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

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

A state $q \in Q$ whose outgoing transitions are all labelled with sending (resp. receiving) actions is called a *sending* (resp. *receiving*) state. A state $q \in Q$ which does not have any outgoing transition is called *final*. If q has both sending and receiving outgoing transitions, q is called *mixed*. We say q is *directed* if it contains only sending (resp. receiving) actions to (resp. from) the same (identical) participant. A *path* in M is a finite sequence of $q_0, \dots, q_n$ $(n \geq 1)$ 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]

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

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

Definition 2.3 (reachable state). Let S be a communicating system. A configuration $s' = (\vec{q}';\vec{w}')$ is *reachable* from another configuration $s = (\vec{q};\vec{w})$ by the *firing of the transition t*, written $s \to s'$ or $s \xrightarrow{t} s'$, if there exists $a \in \mathbb{A}$ such that either: (1) $t = (q_{\mathtt{p}},\mathtt{pq}!a,q'_{\mathtt{p}}) \in \delta_{\mathtt{p}}$ and (a) $q'_{\mathtt{p}'} = q_{\mathtt{p}'}$ for all $\mathtt{p}' \neq \mathtt{p}$; and (b) $w'_{\mathtt{pq}} = w_{\mathtt{pq}}.a$ and $w'_{\mathtt{p}'\mathtt{q}'} = w_{\mathtt{p}'\mathtt{q}'}$ for all $\mathtt{p}'\mathtt{q}' \neq \mathtt{pq}$; or (2) $t = (q_{\mathtt{q}},\mathtt{pq}?a,q'_{\mathtt{q}}) \in \delta_{\mathtt{q}}$ and (a) $q'_{\mathtt{p}'} = q_{\mathtt{p}'}$ for all $\mathtt{p}' \neq \mathtt{q}$; and (b) $w_{\mathtt{pq}} = a.w'_{\mathtt{pq}}$ and $w'_{\mathtt{p}'\mathtt{q}'} = w_{\mathtt{p}'\mathtt{q}'}$ for all $\mathtt{p}'\mathtt{q}' \neq \mathtt{pq}$.

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

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

We now define several properties about communicating systems and their configurations. These properties will be used in § 4 to characterise the systems that correspond to multiparty session types. Let S be a communicating system, t one of its transitions and $s = (\vec{q};\vec{w})$ one of its configurations. The following definitions of configuration properties follow [6, Definition 12].

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

A sequence of transitions is said to be *k-bounded* if no channel of any intermediate configuration s_i contains more than k messages. We define the k-reachability set of S to be the largest subset $RS_k(S)$ of $RS(S)$ within which each configuration s can be

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

reached by a k-bounded execution from s_0. Note that, given a communicating system S, for every integer k, the set $RS_k(S)$ is finite and computable. We say that a trace φ is n-bound, written $bound(\varphi) = n$, if the number of send actions in φ never exceeds the number of receive actions by n. We then define the equivalences: (1) $S \approx S'$ is $\forall\varphi,\ \varphi \in Tr(S) \Leftrightarrow \varphi \in Tr(S')$; and (2) $S \approx_n S'$ is $\forall\varphi,\ bound(\varphi) \leq n \Rightarrow (\varphi \in Tr(S) \Leftrightarrow \varphi \in Tr(S'))$.

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

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

3 Global and Local Types: The LTSs and Translations

This section presents multiparty session types, our main object of study. For the syntax of types, we follow [2] which is the most widely used syntax in the literature. We introduce two labelled transition systems, for local types and for global types, and show the equivalence between local types and communicating automata.

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

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

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

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

Projection. The relation between global and local types is formalised by projection. Instead of the restricted original projection [2], we use the extension with the merging operator $\bowtie$ from [7]: it allows each branch of the global type to actually contain different interaction patterns. The *projection of G onto* $\mathtt{p}$ (written $G \upharpoonright \mathtt{p}$) is defined as:

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

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

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

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

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

$\mathtt{p}?\{a_k.T_k\}_{k\in K} \sqcup \mathtt{p}?\{a_j.T'_j\}_{j\in J} \ = \ \mathtt{p}?(\{a_k.(T_k \sqcup T'_k)\}_{k\in K\cap J} \cup \{a_k.T_k\}_{k\in K\setminus J} \cup \{a_j.T'_j\}_{j\in J\setminus K})$

and homomorphic for other types (i.e. $\mathscr{C}[T_1] \sqcup \mathscr{C}[T_2] = \mathscr{C}[T_1 \sqcup T_2]$ where $\mathscr{C}$ is a context for local types). We say that G is *well-formed* if for all $\mathtt{p} \in \mathcal{P}$, $G \upharpoonright \mathtt{p}$ is defined.

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

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

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

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

LTS over Global Types. We first designate the observables $(\ell, \ell', ...)$. We choose here to follow the definition of actions for CFSMs where a label ℓ denotes the sending or the reception of a message of label a from p to p′: $\ell \ ::= \ \mathtt{pp'}!a \mid \mathtt{pp'}?a$

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

Definition 3.1 (LTS over global types.). The relation $G \xrightarrow{\ell} G'$ is defined as ($subj(\ell)$ is defined in § 2):

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

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

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

[GR1] represents the emission of a message while [GR2] describes the reception of a message. [GR3] governs recursive types. [GR4,5] define the asynchronous semantics of global types, where the syntactic order of messages is enforced only for the participants that are involved. For example, when the participants of two consecutive communications are disjoint, as in: $G_1 = \mathtt{A} \to \mathtt{B} : a.\mathtt{C} \to \mathtt{D} : b.\mathsf{end}$, we can observe the emission (and possibly the reception) of b before the interactions of a (by [GR4]).

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

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

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

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

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

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

Definition 3.2 (LTS over local types). The relation $T \xrightarrow{\ell} T'$, for the local type of role $\mathtt{p}$, is defined as:

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

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

Definition 3.3 (LTS over collections of local types). A configuration $s = (\vec{T};\vec{w})$ of a system of local types $\{T_{\mathtt{p}}\}_{\mathtt{p}\in\mathcal{P}}$ is a pair with $\vec{T} = (T_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ and $\vec{w} = (w_{\mathtt{pq}})_{\mathtt{p}\neq\mathtt{q}\in\mathcal{P}}$ with $w_{\mathtt{pq}} \in \mathbb{A}^*$. We then define the transition system for configurations. For a configuration $s_T = (\vec{T};\vec{w})$, the visible transitions of $s_T \xrightarrow{\ell} s'_T = (\vec{T}';\vec{w}')$ are defined as: (1) $T_{\mathtt{p}} \xrightarrow{\mathtt{pq}!a} T'_{\mathtt{p}}$ and (a) $T'_{\mathtt{p}'} = T_{\mathtt{p}'}$ for all $\mathtt{p}' \neq \mathtt{p}$; and (b) $w'_{\mathtt{pq}} = w_{\mathtt{pq}} \cdot a$ and $w'_{\mathtt{p}'\mathtt{q}'} = w_{\mathtt{p}'\mathtt{q}'}$ for all $\mathtt{p}'\mathtt{q}' \neq \mathtt{pq}$; or (2) $T_{\mathtt{q}} \xrightarrow{\mathtt{pq}?a} T'_{\mathtt{q}}$ and (a) $T'_{\mathtt{p}'} = T_{\mathtt{p}'}$ for all $\mathtt{p}' \neq \mathtt{q}$; and (b) $w_{\mathtt{pq}} = a \cdot w'_{\mathtt{pq}}$ and $w'_{\mathtt{p}'\mathtt{q}'} = w_{\mathtt{p}'\mathtt{q}'}$ for all $\mathtt{p}'\mathtt{q}' \neq \mathtt{pq}$.

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

We now state the soundness and completeness of projection w.r.t. the LTSs.

Theorem 3.1 (soundness and completeness). [2] *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 $G \approx (\vec{T};\vec{\varepsilon})$.*

Local types and CFSMs Next we show how to algorithmically go from local types to CFSMs and back while preserving the trace semantics. We start by translating local types into CFSMs.

Definition 3.4 (translation from local types to CFSMs). Write $T' \in T$ if T' occurs in T. Let T_0 be the local type of participant p projected from G. The automaton corresponding to T_0 is $\mathcal{A}(T_0) = (Q, C, q_0, \mathbb{A}, \delta)$ where: (1) $Q = \{T' \mid T' \in T_0,\ T' \neq \mathtt{t}, T' \neq \mu\mathtt{t}.T\}$; (2) $q_0 = T'_0$ with $T_0 = \mu\vec{\mathtt{t}}.T'_0$ and $T'_0 \in Q$; (3) $C = \{\mathtt{pq} \mid \mathtt{p},\mathtt{q} \in G\}$; (4) $\mathbb{A}$ is the set of $\{a \in G\}$; and (5) δ is defined as:

[2] The local type abstracts the behaviour of multiparty typed processes as proved in the subject reduction theorem in [13]. Hence this theorem implies that processes typed by global type G by the typing system in [2, 13] follow the LTS of G.

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

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

The definition says that the set of states Q are the suboccurrences of branching or selection or end in the local type; the initial state q_0 is the occurrence of (the recursion body of) T_0; the channels and alphabets correspond to those in T_0; and the transition is defined from the state T to its body T_j with the action $\mathtt{pp}'!a_j$ for the output and $\mathtt{pp}'?a_j$ for the input. If T_j is a recursive type variable $\mathtt{t}$, it points the state of the body of the corresponding recursive type. As an example, see C's local type in Example 3.1 and its corresponding automaton in § 1.

Proposition 3.1 (local types to CFSMs). *Assume $T_\mathtt{p}$ is a local type. Then $\mathcal{A}(T_\mathtt{p})$ is deterministic, directed and has no mixed states.*

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

Definition 3.5 (translation from a basic CFSM to a local type). From a basic $M_\mathtt{p} = (Q, C, q_0, \mathbb{A}, \delta)$, we define the translation $\mathcal{T}(M_\mathtt{p})$ such that $\mathcal{T}(M_\mathtt{p}) = \mathcal{T}_\varepsilon(q_0)$ where $\mathcal{T}_{\tilde{q}}(q)$ is defined as:

(1) $\mathcal{T}_{\tilde{q}}(q) = \mu\mathtt{t}_q.\mathtt{p}'!\{a_j.\mathcal{T}^{\circ}_{\tilde{q}\cdot q}(q_j)\}_{j\in J}$ if $(q, \mathtt{pp}'!a_j, q_j) \in \delta$;
(2) $\mathcal{T}_{\tilde{q}}(q) = \mu\mathtt{t}_q.\mathtt{p}'?\{a_j.\mathcal{T}^{\circ}_{\tilde{q}\cdot q}(q_j)\}_{j\in J}$ if $(q, \mathtt{p}'\mathtt{p}?a_j, q_j) \in \delta$;
(3) $\mathcal{T}^{\circ}_{\tilde{q}}(q) = \mathcal{T}_\varepsilon(q) = \mathtt{end}$ if q is final; (4) $\mathcal{T}^{\circ}_{\tilde{q}}(q) = \mathtt{t}_{q_k}$ if $(q, \ell, q_k) \in \delta$ and $q_k \in \tilde{q}$; and
(5) $\mathcal{T}^{\circ}_{\tilde{q}}(q) = \mathcal{T}_{\tilde{q}}(q)$ otherwise.

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

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

The following proposition states that these translations preserve the semantics.

Proposition 3.2 (translations between CFSMs and local types). *If a CFSM M is basic, then $M \approx \mathcal{T}(M)$. If T is a local type, then $T \approx \mathcal{A}(T)$.*

4 Completeness and Synthesis

This section studies the synthesis and sound and complete characterisation of multiparty session types as communicating automata. A first idea would be to restrict basic CFSMs to the natural generalisation of half-duplex systems [6, § 4.1.1], in which each pair of machines linked by two channels, one in each direction, communicates in a half-duplex way. In this class, the safety properties of Definition 2.4 are however undecidable [6, Theorem 36]. We therefore need a stronger (and decidable) property to force basic CFSMs to behave as if they were the result of a projection from global types.

Multiparty compatibility In the two machines case, there exists a sound and complete condition called *compatible* [11]. Let us define the isomorphism $\Phi : (C \times \{!,?\} \times \mathbb{A})^* \longrightarrow (C \times \{!,?\} \times \mathbb{A})^*$ such that $\Phi(j?a) = j!a$, $\Phi(j!a) = j?a$, $\Phi(\varepsilon) = \varepsilon$, $\Phi(t_1 \cdots t_n) = \Phi(t_1) \cdots \Phi(t_n)$. Φ exchanges a sending action with the corresponding receiving one and vice versa. The compatibility of two machines can be immediately defined as $Tr(M_1) = \Phi(Tr(M_2))$ (i.e. the traces of M_1 are exactly the set of dual traces of M_2). The idea of the extension to the multiparty case comes from the observation that from the viewpoint of the participant p, the rest of all the machines $(M_\mathtt{q})_{\mathtt{q} \in \mathcal{P} \setminus \mathtt{p}}$ should behave as if they were one CFSM which offers compatible traces $\Phi(Tr(M_\mathtt{p}))$, up to internal synchronisations (i.e. 1-bounded executions). Below we define a way to group CFSMs.

Definition 4.1 (Definition 37, [6]). Let $M_i = (Q_i, C_i, q_{0i}, \mathbb{A}_i, \delta_i)$. The *associated CFSM* of a system $S = (M_1, .., M_n)$ 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 smallest relation for which: if $(q_i, \ell, q_i') \in \delta_i$ $(1 \leq i \leq n)$, then $((q_1, ..., q_i, ..., q_n), \ell, (q_1, ..., q_i', ..., q_n)) \in \delta$.

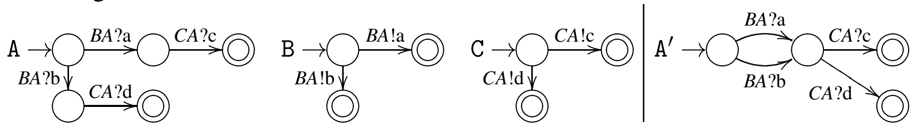
We now define a notion of compatibility extended to more than two CFSMs. We say that φ is an *alternation* if φ is an alternation of sending and corresponding receive actions (i.e. the action $\mathtt{pq}!a$ is immediately followed by $\mathtt{pq}?a$).

Definition 4.2 (multiparty compatible system). A system $S = (M_1, .., M_n)$ $(n \geq 2)$ is *multiparty compatible* if for any 1-bounded reachable stable state $s \in RS_1(S)$, for any sending action ℓ and for at least one receiving action ℓ from s in M_i, there exists a sequence of transitions $\varphi \cdot t$ from s in a CFSM corresponding to $S^{-i} = (M_1, \ldots, M_{i-1}, M_{i+1}, \ldots, M_n)$ where φ is either empty or an alternation and $\ell = \Phi(act(t))$ and $i \notin act(\varphi)$ (i.e. φ does not contain actions to or from channel i).

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

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

Remark 4.1. In Definition 4.2, we check the compatibility from any 1-bounded reachable stable state in the case one branch is selected by different senders. Consider the following machines:



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

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

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

The proof of Theorem 4.1 is non-trivial, using a detailed analysis of causal relations. The proof of Proposition 4.1 comes from the finiteness of $RS_1(S)$. See [17] for details.

Synthesis. Below we state the lemma which will be crucial for the proof of synthesis and completeness. The lemma comes from the intuition that the transitions of multiparty compatible systems are always permutations of one-bounded executions as it is the case in multiparty session types. See [17] for the proof.

Lemma 4.1 (1-buffer equivalence). *Suppose S_1 and S_2 are two basic and multiparty compatible communicating systems such that $S_1 \approx_1 S_2$, then $S_1 \approx S_2$.*

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

Proof. We assume $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$. The algorithm starts from the initial states of all machines $(q^{\mathtt{p}_1}{}_0, ..., q^{\mathtt{p}_n}{}_0)$. We take a pair of the initial states which is a sending state $q_0^{\mathtt{p}}$ and a receiving state $q_0^{\mathtt{q}}$ from p to q. We note that by directness, if there are more than two pairs, the participants in two pairs are disjoint, and by [G4] in Definition 3.1, 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 p, we know that all the transitions are sending actions towards 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 q is in a receiving state $q_{\mathtt{q}}$ such that $(q_{\mathtt{q}}, \mathtt{pq}?a_j, q'_j) \in \delta_{\mathtt{p}'}$ with $j \in J$ and $I \subseteq J$.
  - we set $\mu \mathtt{t}_{q_1,...,q_n}.\mathtt{p} \to \mathtt{q}\colon \{a_i.G(q_1, ..., q_{\mathtt{p}} \leftarrow q_i, ..., q_{\mathtt{q}} \leftarrow q'_i, ..., q_n)\}_{i\in I}$ (we replace $q_{\mathtt{p}}$ and $q_{\mathtt{q}}$ by q_i and q'_i, respectively) and continue by recursive calls.
  - if all sending states in $q_1, ..., q_n$ become final, then we set $G(q_1, ..., q_n) = \mathsf{end}$.
– we erase unnecessary $\mu\mathtt{t}$ if $\mathtt{t} \notin G$.

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

The algorithm can generate the global type in Example 3.1 from CFSMs in § 1and the global type $\mathtt{B} \to \mathtt{A}\{a : \mathtt{C} \to \mathtt{A} : \{c : \mathsf{end}, d : \mathsf{end}\}, b : \mathtt{C} \to \mathtt{A} : \{c : \mathsf{end}, d : \mathsf{end}\}\}$ from $\mathtt{A}'$, $\mathtt{B}$ and $\mathtt{C}$ in Remark 4.1. Note that $\mathtt{B} \to \mathtt{A}\{a : \mathtt{C} \to \mathtt{A} : \{c : \mathsf{end}\}, b : \mathtt{C} \to \mathtt{A} : \{d : \mathsf{end}\}\}$ generated by $\mathtt{A}$, $\mathtt{B}$ and $\mathtt{C}$ in Remark 4.1 is not projectable, hence not well-formed.

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

Theorem 4.3 (soundness and completeness). *Suppose S is basic and multiparty compatible. Then there exists G such that $S \approx G$. Conversely, if G is well-formed, then there exists a basic and multiparty compatible system S such that $S \approx G$.*

5 Conclusion and Related Work

This paper investigated the sound and complete characterisation of multiparty session types into CFSMs and developed a decidable synthesis algorithm from basic CFSMs. The main tool we used is a new extension to multiparty interactions of the duality condition for binary session types, called *multiparty compatibility*. The basic condition (coming from binary session types) and the multiparty compatibility property are a *necessary and sufficient condition* to obtain safe global types. Our aim is to offer a duality notion which would be applicable to extend other theoretical foundations such as the Curry-Howard correspondence with linear logics [4,20] to multiparty communications. Basic multiparty compatible CFSMs also define one of the few non-trivial decidable subclass of CFSMs which satisfy deadlock-freedom. The methods proposed here are palatable to a wide range of applications based on choreography protocol models and more widely, finite state machines. Multiparty compatibility is applicable for extending the synthesis algorithm to build more expressive graph-based global types (*general global types* [8]) which feature fork and join primitives [9].

Our previous work [8] presented the first translation from global and local types into CFSMs. It only analysed the properties of the automata resulting from such a translation. The complete characterisation of global types independently from the projected local types was left open, as was synthesis. This present paper closes this open problem. There are a large number of paper that can be found in the literature about the synthesis of CFSMs. See [16] for a summary of recent results. The main distinction with CFSM synthesis is, apart from the formal setting (i.e. types), about the kind of the target specifications to be generated (global types in our case). Not only our synthesis is concerned about trace properties (languages) like the standard synthesis of CFSMs (the problem of the closed synthesis of CFSMs is usually defined as the construction from a regular language L of a machine satisfying certain conditions related to buffer boundedness, deadlock-freedom and words swapping), but we also generate concrete syntax or choreography descriptions as *types* of programs or software. Hence they are directly applicable to programming languages and can be straightforwardly integrated into the existing frameworks that are based on session types.

Within the context of multiparty session types, [15] first studied the reconstruction of a global type from its projected local types up to asynchronous subtyping and [14] recently offers a typing system to synthesise global types from local types. Our synthesis based on CFSMs is more general since CFSMs do not depend on the syntax. For example, [14, 15] cannot treat the synthesis for $\mathtt{A}'$, $\mathtt{B}$ and $\mathtt{C}$ in Remark 4.1. These works also do not study the completeness (i.e. they build a global type from a set of projected local types (up to subtyping), and do not investigate necessary and sufficient conditions to build a well-formed global type). A difficulty of the completeness result is that it is generally unknown if the global type constructed by the synthesis can simulate executions with arbitrary buffer bounds since the synthesis only directly looks at 1-bounded executions. In this paper, we proved Lemma 4.1 and bridged this gap towards the complete characterisation. Recent work by [1, 5] focus on proving the semantic correspondence between global and local descriptions (see [8] for more detailed comparison), but no synthesis algorithm is studied.

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

References

1. Basu, S., Bultan, T., Ouederni, M.: Deciding choreography realizability. In: POPL 2012, pp. 191–202. ACM (2012)
2. Bettini, L., Coppo, M., D'Antoni, L., De Luca, M., Dezani-Ciancaglini, M., Yoshida, N.: Global progress in dynamically interleaved multiparty sessions. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 418–433. Springer, Heidelberg (2008)
3. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM 30, 323–342 (1983)
4. Caires, L., Pfenning, F.: Session types as intuitionistic linear propositions. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 222–236. Springer, Heidelberg (2010)
5. Castagna, G., Dezani-Ciancaglini, M., Padovani, L.: On global types and multi-party session. LMCS 8(1) (2012)
6. Cécé, G., Finkel, A.: Verification of programs with half-duplex communication. Inf. Comput. 202(2), 166–190 (2005)
7. Deniélou, P.-M., Yoshida, N.: Dynamic multirole session types. In: POPL, pp. 435–446. ACM, Full version, Prototype at `http://www.doc.ic.ac.uk/~pmalo/dynamic`
8. Deniélou, P.-M., Yoshida, N.: Multiparty session types meet communicating automata. In: Seidl, H. (ed.) ESOP 2012. LNCS, vol. 7211, pp. 194–213. Springer, Heidelberg (2012)
9. `http://arxiv.org/abs/1304.1902`
10. Girard, J.-Y.: Linear logic. TCS 50 (1987)
11. Gouda, M., Manning, E., Yu, Y.: On the progress of communication between two finite state machines. Information and Control 63, 200–216 (1984)
12. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: Hankin, C. (ed.) ESOP 1998. LNCS, vol. 1381, pp. 122–138. Springer, Heidelberg (1998)
13. Honda, K., Yoshida, N., Carbone, M.: Multiparty Asynchronous Session Types. In: POPL 2008, pp. 273–284. ACM (2008)

14. Lange, J., Tuosto, E.: Synthesising choreographies from local session types. In: Koutny, M., Ulidowski, I. (eds.) CONCUR 2012. LNCS, vol. 7454, pp. 225–239. Springer, Heidelberg (2012)
15. Mostrous, D., Yoshida, N., Honda, K.: Global principal typing in partially commutative asynchronous sessions. In: Castagna, G. (ed.) ESOP 2009. LNCS, vol. 5502, pp. 316–332. Springer, Heidelberg (2009)
16. Muscholl, A.: Analysis of communicating automata. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 50–57. Springer, Heidelberg (2010)
17. DoC Technical Report, Imperial College London, Computing, DTR13-5 (2013)
18. Takeuchi, K., Honda, K., Kubo, M.: An interaction-based language and its typing system. In: Halatsis, C., Philokyprou, G., Maritsas, D., Theodoridis, S. (eds.) PARLE 1994. LNCS, vol. 817, pp. 398–413. Springer, Heidelberg (1994)
19. Villard, J.: Heaps and Hops. PhD thesis, ENS Cachan (2011)
20. Wadler, P.: Proposition as Sessions. In: ICFP 2012, pp. 273–286 (2012)